The XCore Formal Instruction Set Architecture Specification*

Stephen Wright
Department of Computer Science, Bristol University

September 26th 2011

*Funded by EPSRC Grant EP/H500316/1

Date	***Issue***	***Change***
26/9/2011	1	Original release

Table of Contents

1 Introduction 7
2 Related Work 8
3 Model Retrieval 10
4 Objectives 11
5 Development Context 12
6 Employed Tools 15
6.1 Rodin Core Platform 15
6.2 Text Editor 16
6.3 Provers 16
6.4 Relevance Filter 16
6.5 B2C 16
6.6 Eclipse 16
6.7 Visual C++ 16
6.8 XMOS Development Tools 16
7 Modelling Methodology 17
7.1 Event-B 17
7.2 Context Construction 18
7.3 Event Construction 18
7.4 Refinement Hierarchy 19
7.5 Partitioning 19
7.6 Proof Obligation Discharge 19
7.7 Simulation and Testing 20
8 Model Description 21
8.1 Model Structure 21
8.2 Machines Summary 21
8.2.1 XCore1 21
8.2.2 XCore2 21
8.2.3 XCoreFinal 22
8.2.4 XCoreIsaArithmetic1 22
8.2.5 XCoreIsaArithmeticFinal 22
8.2.6 XCoreIsaBitManipulation1 22
8.2.7 XCoreIsaBitManipulationFinal 22
8.2.8 XCoreIsaBranching1 22
8.2.9 XCoreIsaBranchingFinal 22
8.2.10 XCoreIsaChannelEnd1 22
8.2.11 XCoreIsaChannelEndFinal 22
8.2.12 XCoreIsaClock1 22
8.2.13 XCoreIsaClock2 22
8.2.14 XCoreIsaClockFinal 23
8.2.15 XCoreIsaLoadAddress1 23
8.2.16 XCoreIsaLoadAddressFinal 23
8.2.17 XCoreIsaLock1 23
8.2.18 XCoreIsaLockFinal 23
8.2.19 XCoreIsaMemoryRead1 23
8.2.20 XCoreIsaMemoryReadFinal 23
8.2.21 XCoreIsaMemoryWrite1 23
8.2.22 XCoreIsaMemoryWriteFinal 23
8.2.23 XCoreIsaPort1 23

8.2.24 XCoreIsaPortFinal 23
8.2.25 XCoreIsaPortConnections 23
8.2.26 XCoreIsaProcessorState1 24
8.2.27 XCoreIsaProcessorStateFinal 24
8.2.28 XCoreIsaRegisterSet1 24
8.2.29 XCoreIsaRegisterSetFinal 24
8.2.30 XCoreIsaResource1 24
8.2.31 XCoreIsaResourceFinal 24
8.2.32 XCoreIsaThread1 24
8.2.33 XCoreIsaThreadFinal 24
8.2.34 XCoreIsaTimer1 24
8.2.35 XCoreIsaTimerFinal 24
8.2.36 XCoreSupportMch 24
9 Testing 26
9.1 Basic Output 26
9.2 Logic 26
9.3 Arithmetic 26
9.4 Parallel Threads 26
9.5 Channel Communication 26
9.6 Port Communication 26
9.7 Timer Event 26
9.8 Channel Event 26
9.9 Timer / Channel Event 26
9.10 Interrupt Handling 27
9.11 Exception Handling 27
9.12 Test Coverage 27
10 Results 28
10.1 Specification Improvements 28
10.2 Tool Usage Experiences 28
10.3 Event Ratio 29
11 Future Work 30
11.1 Model Enhancement 30
11.2 Document generation 30
11.3 Enhanced PO Discharge 30
11.4 Abstract model 30
11.5 Model Checking 30
11.6 Enhanced Test-suite 31
12 Summary 32
13 References 33
14 Formal Specification Listing 34
14.1 Machines 34
14.1.1 machine XCore1 34
14.1.2 machine XCore2 52
14.1.3 machine XCoreFinal 56
14.1.4 machine XCoreIsaArithmetic1 58
14.1.5 machine XCoreIsaArithmeticFinal 68
14.1.6 machine XCoreIsaBitManipulation1 79
14.1.7 machine XCoreIsaBitManipulationFinal 87
14.1.8 machine XCoreIsaBranching1 97
14.1.9 machine XCoreIsaBranchingFinal 103

14.1.10 machine XCoreIsaChannelEnd1 113
14.1.11 machine XCoreIsaChannelEndFinal 139
14.1.12 machine XCoreIsaClock1 162
14.1.13 machine XCoreIsaClock2 165
14.1.14 machine XCoreIsaClockFinal 168
14.1.15 machine XCoreIsaLoadAddress1 174
14.1.16 machine XCoreIsaLoadAddressFinal 176
14.1.17 machine XCoreIsaLock1 181
14.1.18 machine XCoreIsaLockFinal 185
14.1.19 machine XCoreIsaMemoryRead1 188
14.1.20 machine XCoreIsaMemoryReadFinal 192
14.1.21 machine XCoreIsaMemoryWrite1 204
14.1.22 machine XCoreIsaMemoryWriteFinal 212
14.1.23 machine XCoreIsaPort1 221
14.1.24 machine XCoreIsaPortFinal 241
14.1.25 machine XCoreIsaPortConnections 261
14.1.26 machine XCoreIsaProcessorState1 264
14.1.27 machine XCoreIsaProcessorStateFinal 270
14.1.28 machine XCoreIsaRegisterSet1 275
14.1.29 machine XCoreIsaRegisterSetFinal 281
14.1.30 machine XCoreIsaResource1 288
14.1.31 machine XCoreIsaResourceFinal 292
14.1.32 machine XCoreIsaThread1 296
14.1.33 machine XCoreIsaThreadFinal 309
14.1.34 machine XCoreIsaTimer1 317
14.1.35 machine XCoreIsaTimerFinal 327
14.1.36 machine XCoreSupportMch 337
14.2 Contexts 340
14.2.1 context ArithmeticTheorems 340
14.2.2 context OvrTheorems 341
14.2.3 context Bit 351
14.2.4 context Bits 352
14.2.5 context BitsToNat 353
14.2.6 context BitsConcatenation 358
14.2.7 context BitsExtraction 360
14.2.8 context BitsManipulation 363
14.2.9 context Arithmetic 365
14.2.10 context Fifo 368
14.2.11 context ArraySearch 369
14.2.12 context Status 370
14.2.13 context XCoreStatus 371
14.2.14 context ThreadState 372
14.2.15 context ThreadDom 372
14.2.16 context ThreadArray 372
14.2.17 context RegArrayDom 373
14.2.18 context RegArrayDomE1 373
14.2.19 context RegArray 374
14.2.20 context RegArrayE1 374
14.2.21 context ProcessorState 375
14.2.22 context PcDom 376

14.2.23 context PcArray 376
14.2.24 context MemArrayDom 377
14.2.25 context MemArrayDomE1 377
14.2.26 context MemArrayDomE2 377
14.2.27 context MemArrayDomE3 378
14.2.28 context MemArrayDomE4 378
14.2.29 context MemArray 379
14.2.30 context MemArrayE1 379
14.2.31 context MemArrayE2 379
14.2.32 context MemArrayE3 379
14.2.33 context MemAlignment 380
14.2.34 context MemAlignmentE1 380
14.2.35 context Macros 381
14.2.36 context InstSize 383
14.2.37 context InstArray 383
14.2.38 context Loader 384
14.2.39 context XCoreBasic 385
14.2.40 context XCoreChannelEnds 386
14.2.41 context XCoreClocks 389
14.2.42 context XCoreDataSize 390
14.2.43 context XCoreFields 391
14.2.44 context XCoreFormats 397
14.2.45 context XCoreOperands 400
14.2.46 context XCoreSubOperands 404
14.2.47 context XCoreInstSize 406
14.2.48 context XCoreLocks 407
14.2.49 context XCoreMemArrayDom 408
14.2.50 context XCoreMemArray 408
14.2.51 context XCorePorts 409
14.2.52 context XCoreProcessorState 411
14.2.53 context XCoreRegArray 417
14.2.54 context XCoreResourceClassifications 419
14.2.55 context XCoreResources 421
14.2.56 context XCoreStatusCodes 423
14.2.57 context XCoreSynchronisers 425
14.2.58 context XCoreThreads 426
14.2.59 context XCoreTimers 428

1 Introduction

This document describes a formal specification of the XMOS XCore microprocessor Instruction Set Architecture (ISA) [15], constructed in the Event-B formal notation using the Rodin formal development environment.

The work was completed between October 2010 and October 2011 by Dr. Stephen Wright of Bristol University Computer Science Department, under a Knowledge Transfer Secondment (KTS) to XMOS Ltd (Grant EP/H500316/1).

2 Related Work

Techniques combining formal specification and informal comments began with the use by Bowen of well commented and structured Z for specification of the 6800 [7]. Limited abstraction in the form of instruction group definitions was introduced. Boyer and Yu used Nqthm to prove various correctness properties for a series of short subroutines for the 68020 [8], using a form of the Lisp high-level language to both specify the 68020 ISA, and re-code the executed algorithms for functional comparison against conventional generated binaries. Low-level properties such as guaranteed termination the routines without exceptions being generated, correct incrementing of the PC, correct location of results, correct restoration of register context in the post-condition, and avoidance of invalid memory accesses were proved. High-level properties such as equivalence between the binary functions and their re-coded algorithms were proved. The technique involved per-instruction re-specification of a sub-set of the ISA.

In the domain of Virtual Machine instruction sets, Basin, Friedrich and Gawkowski used HOL to perform type-safety checking of Java Virtual Machine (JVM) bytecode executables [4], i.e. checking that operands were applied to appropriately typed operations and no stack overflows or underflows occurred. The work re-used a formal specification of a subset of the JVM, allowing verification to be performed in the Isabelle prover. Moore used A Computational Logic for Applicative Common Lisp (ACL2), an evolution of Nqthm, to perform a similar analysis of JVM bytecode executables [17].

The boundary between verification of binary executables and micro-architectures is marked by the verification of processor microcode. Arons et al. of Intel performed functional equivalence checking of successive microcode versions in order to confirm backward compatibility [1]. Again the employed technique was a per-instruction restatement of each instruction's functionality, here in the form of a template coded in a proprietary language and applied to every instruction of the two micro-programs to be compared. Some abstraction allowed formal analysis for state transition similarity to be performed and introduced bit-level descriptions.

Moore et al. at AMD used ACL2 to verify correct operation of floating-point divide microcode [18]. These included high-level properties such as correct division results and also low-level properties such as no instruction aborting or register overflows. Once again the program was translated into the input notation of the tool, and specifications of the employed instruction subset and correctness definitions were given in the same language.

Since the 1980s there has been much activity in the field of automatic verification of processor micro-architecture (i.e. the hardware used to implement the ISA). Two approaches have been taken: definition of simplified academic examples capable of being studied in their entirety, and verification of properties for pre-existing commercial processors, usually down to the level of gate logic. The work described here has been evolved using initial simplified academic examples, before expansion to accommodate a commercial device.

An early example using the simplified processor approach is Hunt using Nqthm to verify the FM8501 (a sixteen bit microprocessor based on the PDP-11) [12]. This work constructs the familiar two levels of a per-instruction specification and an implementation in simple Boolean logic, both within the Nqthm environment. The tool's proving capability was then used to prove a functional correlation.
Graham and Birtwistle used HOL to partially verify a hardware implementation of the SECD (Stack, Environment, Code, Dump) Virtual Machine [11]. This work again adopts the per-instruction specification approach, but uses three levels: a top-level state transition representation, a register transfer representation, and gate-level implementation.

In the domain of commercial grade microprocessors, the VIPER (Verifiable Integrated Processor for Enhanced Reliability) project at the Royal Signals and Radar Establishment in the UK achieved a partial proof of a thirty-two bit machine [9]. The work introduced four levels of abstraction, significantly beginning with an initial model of the machine state transitions abstract from particular instruction actions. Major state machine and logic block levels were then refined to an implementation at gate level. The work used Logic for Computable Functions/Logic of Sequential Machines (LCF-LSM) notation above the implementation level. The limitations of available tools prevented complete verification of refinement between levels: verification was achieved through use of a combination of "pencil-and-paper", translation to HOL, and exhaustive testing techniques.

Srivas and Miller of NASA used the Prototype Verification System (PVS), another theorem proving tool and input notation, to verify the AAMP5 proprietary microprocessor [22]. The work used only two modelling levels, instruction-set and register-transfer, and verified behavioural equivalence for a representative subset of instructions. The work demonstrated formal verification in an industrial environment, introducing tools packaged for use by engineers used to conventional techniques.
Fox at the University of Cambridge Computer Laboratory used HOL to verify the ARM6 processor by again constructing two models at the ISA and micro-architectural levels and proving equivalence [10]. This project covered all instructions apart from an extension package and coprocessor instructions.

Beyer et al. used PVS to perform a complete verification of a micro-architecture implementation for the DLX processor (a simplification of the MIPS commercial processor) [6], using a similar two-level approach. Significantly, this work included automatic generation of a functioning implementation capable of program execution on a Field Programmable Gate Array. This was achieved by the creation of a conventionally developed (i.e. not formally verified) tool capable of translating a subset of the PVS input notation to the Verilog hardware descriptor language.

Wright and Eder use Event-B to incrementally construct an academic ISA from a generic abstract model, through to a refinement suitable for automatic translation to a Virtual Machine capable of executing compiled binaries [25]. The XCore ISA specification described here is a direct extension of this work.

3 Model Retrieval

All source code described here is available for download at:
http://deploy-eprints.ecs.soton.ac.uk/346/.

Two files are available for download at this location: an archive *XCoreBundle.zip* and a covering document *XCore Deploy Covering Document.pdf*. *XCoreBundle.zip* is an archive containing source code for the Event-B model and its supporting tools.

XCore Deploy Covering Document.pdf gives complete instructions for installation, building and running of the model, tools, virtual machines and test programs from the archive. The procedure given below re-iterates the parts of the instructions related to installation of the Rodin tool and loading of the MIDAS Event-B model for inspection.

4 Objectives

The XCore ISA is published in the document "The XMOS XS1 Architecture" (https://www.xmos.com/download/public/The-XMOS-XS1-Architecture%281%29.pdf). This is similar to many other published ISA specifications, listing each instruction in alphabetical order of its assembler mnemonic, and containing a mixture of natural language description and semi-formal pseudo-code notation.

The primary objective of the model construction project was to provide a rigorous re-statement of the specification, with particular emphasis on checking for errors in the existing document. Errors could be in the form of direct misstatement of behaviour (such as reversed instruction fields), omission of behaviours under corner-case conditions (such as undocumented exceptions), and ambiguities (such as implicit modulo arithmetic not being explicitly stated). The model is not yet intended as the definitive specification for the ISA's behaviour.

Once constructed, the formal specification provides the basis for the development of more abstract models linked to this final "refinement", with a view to identifying faults or potential optimisations in the ISA's high-level design. This application is particularly appropriate to the special instructions provided by the ISA to support multi-thread execution and synchronisation, and inter-device and inter-thread communication.

The XCore ISA is specified within XMOS via a linked database using non-formal notation and verified by rigorous conventional test-based verification techniques. A formal statement of the specification provides the basis for an expansion of this capability, allowing static analysis of the design and directed test generation.

5 Development Context

Knowledge Transfer Secondments (KTS) grants support the secondment of EPSRC-funded staff into organisations that can exploit their research results. KTS funding can also be used to host researchers from industry working on specific projects which build on the results of earlier EPSRC-funded research. This work was completed as part of a Bristol University KTS (Grant EP/H500316/1). The project applied and extended the Event-B and RODIN based techniques for ISA analysis developed by Dr Stephen Wright during his doctoral research [26] to an industrial setting. To that end, XMOS Ltd hosted Dr Wright in the period October 2010 to October 2011.

The achievements of the project with respect to the initially outlined goals are summarised below:

Task 1.1: Architecture and Tool Chain Familiarization
This task covered initial familiarization with all aspects of the XMOS XCore, with particular emphasis on the ISA. It included understanding of the ISA documentation and its assembly-language representation, understanding those aspects of the micro-architecture that affect the ISA, plus compilation and execution of test executables on real and emulated ISAs. The task was fully completed: a detailed understanding of all instructions in the XCore ISA and their interactions was gained in order to precisely capture their functionality into the formal model. Detailed understanding of the XMOS tool-chain was achieved in order to develop the executable test-suite developed in Task 3.4.

Deliverable D1: Hand-coded VM
This task developed a hand-coded ISA-level VM using existing libraries, in keeping with the process developed in [2]. Execution of XMOS compiler derived executables will allow checking of understanding gained in Task 1.1. The VM and executables were later used for reference in Task 3.4. The task was partially completed, covering the core instructions of the XCore ISA. The deliverable is to be published alongside the Formal Model.

Task 1.3: Rodin Tool Set Review
Rodin was the tool set used by Dr Wright during his PhD research. This task aimed to review the developments made by the Rodin development and user community since January 2010, with particular emphasis on editing, automatic refinement, automatic proof discharge techniques and automatic code generation. The task also covered installing the latest stable release at XMOS and familiarization with this latest release. The task was fully completed: a detailed review of the available Rodin tool-set was completed, and key extensions (specifically the “Camille” text editor and “Relevance Filter” prover extension) were selected to aid the development process.

Task 2.1: Rodin ISA Model Editor
In order to facilitate modelling the XCore ISA, a model editor was to be developed that automated much of the manual model formalization. This task was fully completed by use of the Rodin text editor extension and expansion of the B2C code generation tool.

Task 2.2: Rodin ISA-specific Proof Tactics
Many of the formal proofs performed in MIDAS, although conceptually very similar, required proof guidance resulting in frequent, time consuming manual interventions. Based on the similarity of the proof structure and the fact that the formal models to be developed in the context of this secondment are restricted to the domain of microprocessor ISAs, this task aimed to define a set of domain-specific proof tactics to automatically guide the formal proof tools, thus drastically reducing the need for manual proof guidance, ideally resulting in completely automatic proof discharge. The task was partially completed: automatic discharge of many additional formal proof obligations was achieved by selection of the "MetaProver" extension and enhancement of the Rodin automatic proof rules. However, complete automatic discharge was not achieved, and the model is expected to provide a valuable test article for "Artificial Intelligence for Formal Methods" project (http://www.ai4fm.org/).

Task 2.3: Update of Generic Modelling Framework
The generic modelling framework developed in MIDAS was to be customized and enhanced following "Future Work" proposals from MIDAS before its adoption as the basis of the XCore formal model. The task partially completed, covering the core instructions of XCore, but not the special instructions.

Deliverable D2: Report
This report constitutes this deliverable.

Task 3.1: 1st Iteration of XCore ISA Formalization
This task first identified the core part of the XCore ISA to be formalized. An XCore-specific refinement of the generic modelling framework formalizing this core part would then be performed. This task was fully completed early in 2011.

Task 3.2: 2nd Iteration of XCore ISA Formalization
In a second iteration, selected other parts of the XCore ISA would be formalized; potentially the entire ISA. The formalisation of the whole ISA was completed in September 2011. The complexity of the XCore special instructions (particularly multi-thread management and communication), and their interactions, called for considerable expansion of the modelling methodology and tools. This task dominated the project, consuming approximately 6 months.

Deliverable D3: XCore Formal Model
This was fully completed. Publishing of this complete model to the customer and academic communities represents the most significant output of the project.

Task 3.3: B2C XCore ISA Refinement
This task further refined the formal XCore ISA model from Tasks 3.1/3.2 ready for automatic code generation. This task was fully achieved by the considerable enhancement of the B2C tool, allowing a VM to be derived directly from the formal specification without need for a translation refinement step.

Task 3.4: Automatic VM Generation and Test
This task covered the automatic generation of the formally derived XCore VM and its testing by running executables developed in Task 1.2. This was fully completed. The

automatically generated VM produced in Task 3.3 allowed a detailed test-suite to be developed and executed using the XMOS development tools. This task also proved essential in validation of the model.

Task 4.1: Formally Derived ISA Specification
This task covered the generation of a natural language document specifying the XCore ISA behaviour under all conditions derived from the formal Specification, for review against existing documentation. This was not completed. A complete re-statement of the ISA specification was not achieved. However, numerous corrections to the existing ISA specifications were identified and reported via the XMOS reporting system.

Deliverable D4: Formal ISA Specification
This was not achieved due to the time overruns encountered during Task 3.2.

6 Employed Tools

The flow of information through the various tools employed during development is summarized in Figure 1 below.

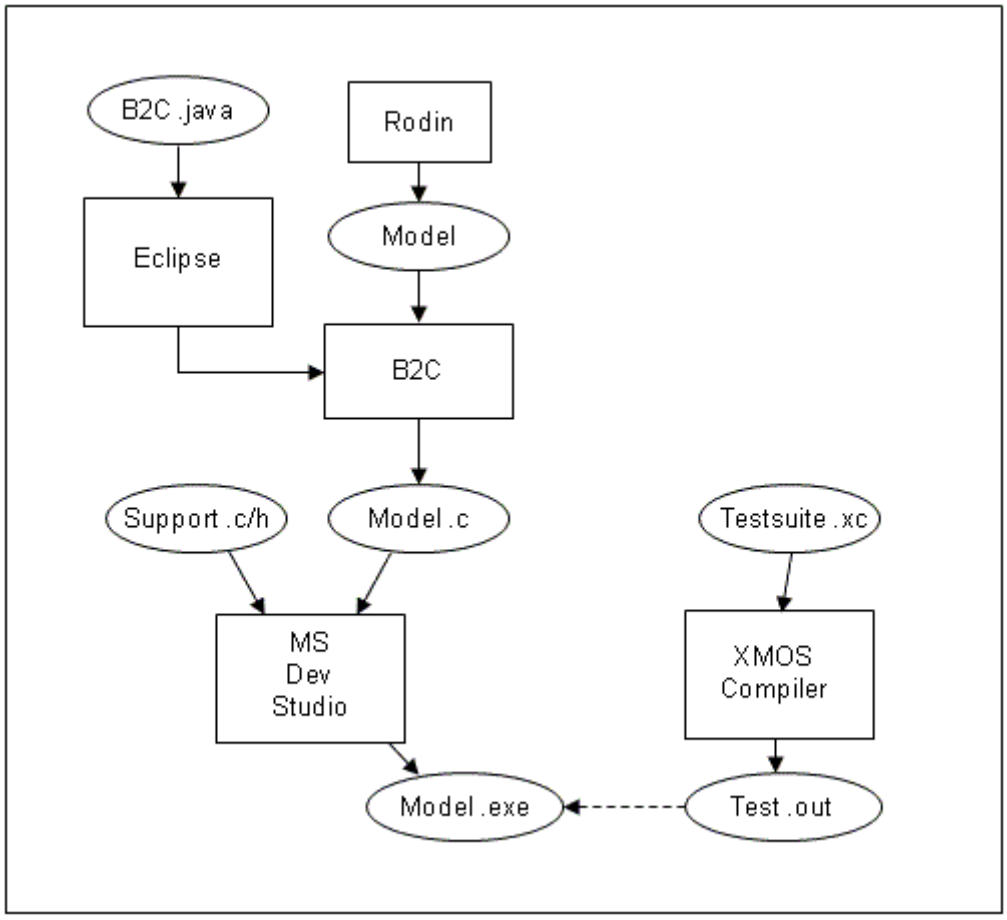

Figure 1: Development Tool Flow

The Eclipse SDK is used to compile Java source code to the B2C plug-in for Rodin. Rodin is then used to browse and edit the model and generate a C output file via B2C. This B2C-generated XCore C file is compiled along with various supporting C files by Visual C++ to create an instruction set emulator Virtual Machine (VM) as a Windows console executable. The XMOS XCore tool chain is then used to compile the demonstration C and XC [23] files to an XCore binary executable, allowing it to be executed on the Event-B derived VM.

6.1 Rodin Core Platform

The Rodin Core [2] provides a central database containing all information on the model under development, plus tools and user interfaces for its basic manipulation. Also included is the "Proof Obligation Generator", used to generate all POs from the model, and an initial theorem prover within an integrated proving tool. The proving tool also includes methods to apply a range of proving "tactics" for applying the available theorem provers and other proving rules.

6.2 Text Editor
All editing of the XCore model was performed via the “Camille” text editing Rodin plug-in [5].

6.3 Provers
As well as the “New Predicate Prover” (NewPP) installed as part of the default Rodin core, the Atelier “Predicate Prover” (PP) and “Mono Lemma” (ML) provers, supplied by Systerel, are installed as an additional plug-in.

6.4 Relevance Filter
Rodin's external provers (PP, NewPP, and sometimes also ML) tend to perform poorly in the presence of irrelevant hypotheses. This plugin [20] (also known as the MetaProver) applies several heuristics for selecting relevant hypotheses and thus improving automatic PO discharge rates.

6.5 B2C
The Event-B to C source code tool used for the XCore model is an updated version of that developed for the original MIDAS project [24]. The updated plug-in is provided by the download procedure discussed in Section 3.

6.6 Eclipse
The B2C plug-in is developed and supported using the standard Eclipse Java Development environment.

6.7 Visual C++
Microsoft Visual C++ 5.0 is required for compilation of the VM and support server used for the ISA testing described in Section 9.

6.8 XMOS Development Tools
The XMOS compiler/assembler/linker tool-chain is used to compile the XC/C/assembler test-suite described in Section 9. The command-line version of the tools are employed, using scripts supplied as part of the project download.

7 Modelling Methodology

7.1 Event-B

Event-B [16] is a formal method that combines mathematical techniques, set theory, and first-order logic. It is an evolution of B-Method (now often referred to as Classic-B), which was itself derived from Z notation. Event-B is a simplification of Classic-B, intended to allow more convenient practical development.

Event-B is a model-based method allowing the definition of the stored state within a system as globally scoped state variables, and events, the modifications that may be performed on them. These consist of actions, the description of the state modification, and guards, the conditions under which the action may occur. Events may also define local variables with scope limited to the event to model function arguments and intermediate concepts used within the guards or actions. Set membership statements, fundamentally derived from integers, Booleans, or completely abstract sets, provide variable type information.

A significant feature of Event-B is refinement. Abstract model variables may be refined to one or more concrete variables, which may be of a different type to the abstract version. The relationship between abstract and concrete versions of a variable is stated by a gluing invariant. Thus any events containing refined variables must also be refined in the concrete version of the model in order to maintain the relationship stated in the gluing invariant. Other event refinements are possible: guards may be restated or strengthened, that is replaced by a logical sub-set of the abstract guard (i.e. making them harder to satisfy). Events may be split during refinement into two or more concrete events, in which the guards and actions of both must be valid refinements of the abstract version. Event-B syntax explicitly references an event's abstract version, allowing different names to be used in the concrete version, as well as providing scoping information during the proving process.

Central to the use of Event-B as a formal method is the use of proof obligations (PO). These are sequents, i.e. requirements to logically prove a goal, given a set of hypotheses. POs serve two roles: verification of the internal consistency of a model with respect to the basic rules of set theory, and verification of some desired property of the model. For models employing multiple steps of refinement, internal consistency POs may be further categorized into those confirming the correct semantics of the modelling statements (including those linking refinement steps), and those confirming that each step is a valid refinement of its abstract. Invariant statements define the conditions that must hold true under all variable modifications: these are enforced by the generation and discharging of POs.

The only requirements of a valid Event-B model that are enforced by default are consistency (preservation of all invariants), and refinement correctness (all events of a model preserve the gluing invariants). All other required properties must be explicitly stated, for example, model determinism, i.e. any given condition resulting in only one possible set of actions (although the actions themselves may still contain indeterminism). The developer must explicitly enforce these by the introduction of appropriate invariants or theorems.

Event-B's primary advantage is its flexibility, both in the notation itself and its supporting tools. Event-B does not strictly define a notation, but a methodology for the construction and analysis of linked logical objects, and the notation seen when using support tools is actually an arbitrary front-end for users familiar with Classic-B and Z notation. Event-B was therefore developed closely alongside a supporting open-source tool-set, Rodin [1], which allows flexibility in the presentation and manipulation of an underlying linked database representing the loaded model. Rodin has a modular architecture based on the Eclipse framework with clearly defined Application Programming Interfaces (API) exposing the database, and supports expansion via the addition of Eclipse plug-ins. Rodin is supported by a mixed academic and industrial development community, which has contributed many plug-ins ranging in function from top-level user-interface support [21] to low-level translation of refined models to C [24].

7.2 Context Construction

Event-B *contexts* are used to define a range of static properties of the model. Type definitions are constructed as nested sub-sets of two fundamental sets: integers and bit-strings. Various enumerations are constructed as sub-sets of the integers.

Mapping functions are used to define array types such as that used for the memory system, uninterpreted functions to be resolved outside the formal domain, and "macros" containing logical definitions and nested references to other functions.

Theorem statements are used to create intermediate hypotheses in order to accelerate the PO discharge process.

7.3 Event Construction

Throughout the model all the functionality of a single instruction execution is contained within a single event. Thus the ordering of unconnected actions during an instruction execution is deliberately abstracted, simplifying the model and maximising the range of possible implementations.

The model consists of events corresponding to execution of instructions of the ISA under various conditions, plus a small number of other events, e.g. loading of executable into memory during boot, and operation of embedded resources. The instruction events of the ISA are constructed using the methodology described in [26]. That is events corresponding to successful instruction execution are constructed as guarded atomic actions, and exception cases are constructed by analysis of these guards, allowing events to be specified for guards not being satisfied. Exception events are constructed by extension of generic exception events, specified by earlier Event-B machines.

This refinement strategy implies non-determinism in the model; i.e. no precedence is defined, and therefore in the case of multiple exceptions occurring simultaneously, the precise event to be triggered is not defined. This strategy reduces the complexity of the model and reflects the fact that precedence is not defined by the specification and may be resolved in any manner by a particular implementation.

7.4 Refinement Hierarchy

In contrast to many Event-B models, which employ *refinement* to derive a final specification from an initial abstract representation, the XCore specification is currently "flat", i.e. the complete specification is presented at a full level of detail, providing all information necessary to execute compiled binaries. However, the model is structured with the intention of being the basis of a refined abstract model for some or all of the ISAs functionality.

7.5 Partitioning

In order to assist understanding by a human reader, enhance reuse of repeated constructs, and simplify manipulation, Event-B features are used to partition the specification into many smaller Event-B *machines*. Partitioning of the model may be considered to be performed *vertically*, in which instructions are grouped according to their general properties, and *horizontally*, in which events are incrementally enriched with additional guards and actions via the Event-B *EXTENDS* mechanism.

The vertical partitioning of the model is partly driven by precise Formal Methods constraints. The special features of the XCore ISA demand that additional state variables be introduced over and above the generic ISA state variables that are manipulated by nearly all events (e.g. register file, program counter). These special state variables are introduced in a single machine and its extensions, and therefore all events acting on a special state variable must be contained in that machine and its extensions.

Many events do not act on special state variables, and simply describe events acting on the generic state introduced in the early, common *machines*. In these cases no particular division of the events into separate *machines* is demanded by the rules of Event-B, but they have been partitioned by their general properties in order to assist the human reviewer. For example, events corresponding to arithmetic operation instructions are grouped, as are events corresponding to flow-control instructions. Vertical partitioning of the ISA effectively divides the ISA into several separate Event-B models, albeit contained within the same Event-B project. However, these separate components are effectively remerged by the B2C plugin, which generates a single .c and .h C source files for the leaf events of all models found within a project. Thus the events may be tested as a single coherent machine, as described in Section 7.7.

7.6 Proof Obligation Discharge

The model yielded 4783 POs, all of which were discharged, 3037 entirely automatically, 1746 manually. Both automatic and manual proof was achieved using a combination of the ML, PP and NewPP provers. However, for automatic discharge, several tactic *profiles* were employed, requiring multiple sweeps through the PO list. For manual proving a signal very simple tactic profile was employed, which did not invoke any of the provers: these were called explicitly from the Proof Control interface when needed.

Although only 3037 POs were discharged entirely automatically, many of the POs were in fact *Well-Definedness* theorems consisting of multiple sub-POs, many of which were resolved automatically.

7.7 Simulation and Testing

In addition to static ISA specification and analysis, the methodology allows the specification to be analyzed dynamically, by execution of a derived implementation in the form of an automatically generated C-coded instruction set emulator VM. This approach has multiple advantages. “Animation” of a formal model is a common technique for *validation* of the model, i.e. confirming to a developer that the needs of the specification has been captured, and execution of compiled binaries on the VM provides this function very efficiently. Secondly, a VM derived directly from a formal Event-B model provides a highly efficient method for generation of test-suites with good functional coverage, as the ISAs behaviour has been systematically decomposed to a list on individual guarded atomic actions, and tests can be written to target each of them. There is a common misconception that Formal Methods seek to replace all testing, whereas it may actually complement it by directing a user to create more complete test-suites, and discover scenarios that may not be intuitively obvious. Finally, on a purely human level, execution of a derived implementation serves to greatly instil confidence in the methodology in general and the specification in particular, especially to an audience unfamiliar with Formal Methods.

As described in Section 6, a VM is produced as automatically generated C source code via the B2C plug-in. The formally constructed functionality is encapsulated within a .c and .h file (named XCoreIsa.c/h in this case), containing initialization and iterate functions. Hand-coded C functions are then supplied to provide all other functionality, such as scheduling, instrumentation and executable loading.
For demonstration and test of the ISA the VM is invoked and, after initialization, is iterated continuously until a halt is signalled. In the case of the XCore, halt is detected by a hand-coded function detecting that all threads have entered the waiting state. Instrumentation calls are extended to log the triggering of events during execution, and at execution completion a display function is called to summarize the number of hits detected for each event-mapped function. Thus a clear coverage metric is maintained. The test environment is shown in Figure 2: Simulation Environment.

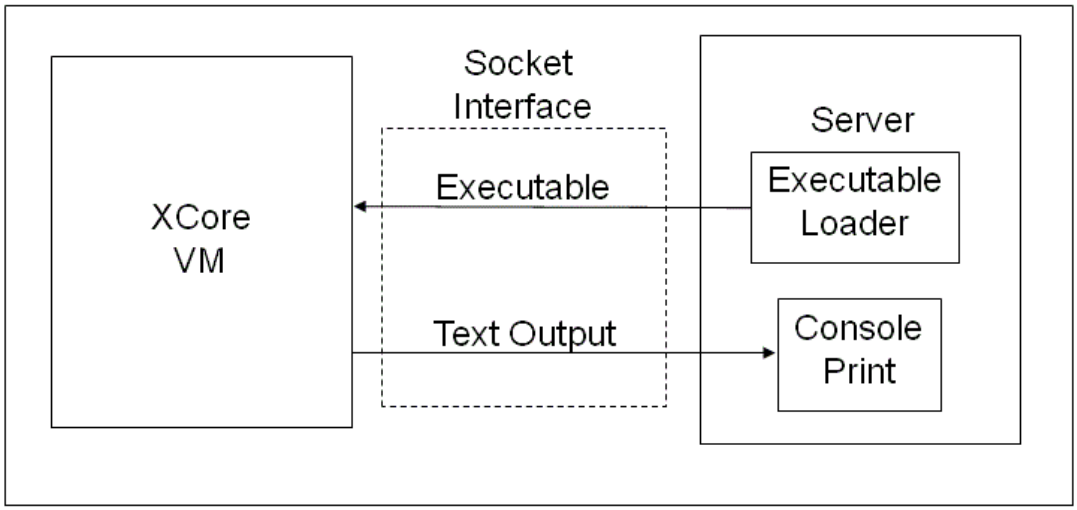

Figure 2: Simulation Environment

The scheduling code also provides executable loading and a virtual console mechanism via a remote server, allowing instrumentation from within the demonstration executable to be displayed in a separate console to that of the VM.

8 Model Description

8.1 Model Structure

The general methodology for developing the XCore model from multiple linked Event-B *machines* is described in Section 7. The relationships between the many machines that compose the model are shown in Figure 3: Model Partitioning Summary.

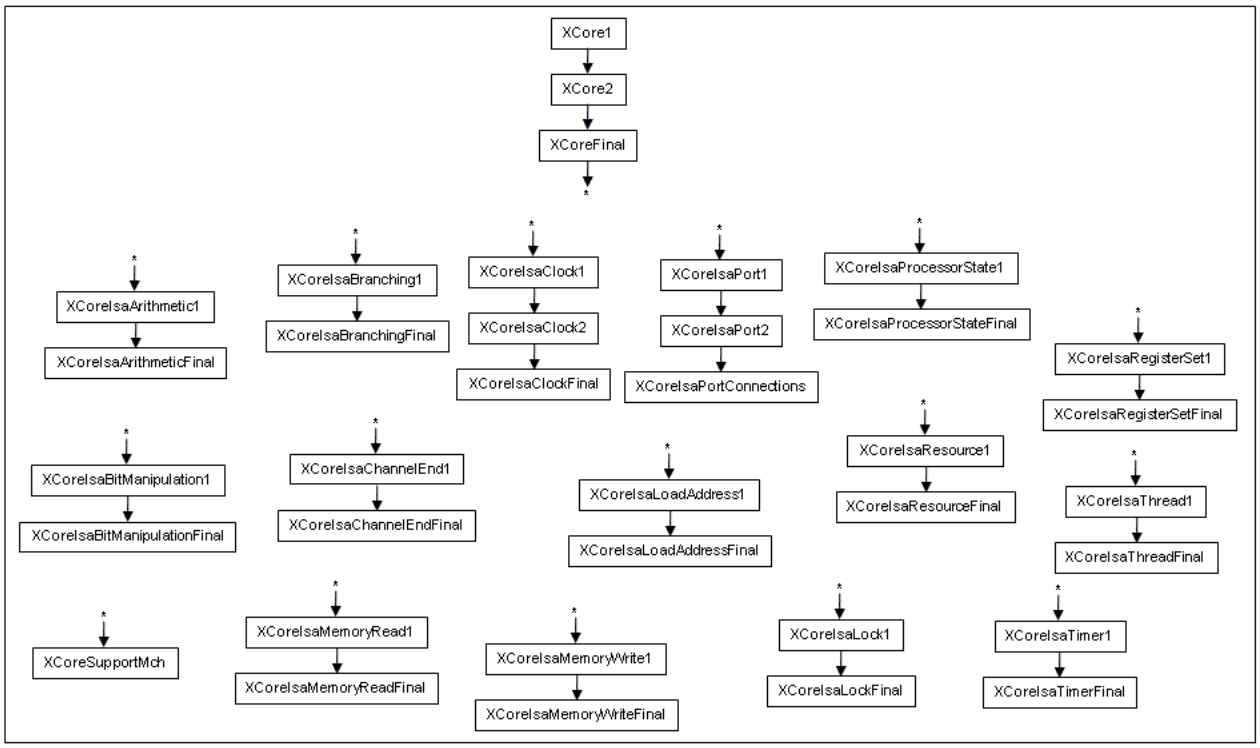

Figure 3: Model Partitioning Summary

A description of the contribution of each of these machines is given in Section 8.2. The complete listing of the machines and their included Event-B *contexts* is given in Section 14.

8.2 Machines Summary

8.2.1 XCore1

XCore1 is the top-level machine of model, defining the state common to generic ISAs. These are: the execution mode of the overall machine (i.e. booting, normal or debug execution), the register file, the Program Counter and the Memory system. As no refinement is employed in the model, some state is also included that is specific to the XCore ISA: i.e. management of multiple register file contexts.

Basic events are defined here, specifying the limited forms of modification allowed to this state by the ISA events constructed in later extensions.

8.2.2 XCore2

The machine extends XCore1, splitting a generic Exception event into the specific types occurring on the XCore.

8.2.3 XCoreFinal
This extends XCore2, and provides a dummy layer before the multi-way split to the instruction-specific machines, in order to allow other layers to be inserted without needing to rename the “refines” statements in all the later machines.

8.2.4 XCoreIsaArithmetic1
This extends XCoreFinal. It provides the functionality of all arithmetic operations (e.g. addition, less-than-comparison).

8.2.5 XCoreIsaArithmeticFinal
This extends XCoreArithmetic1, providing the instruction-specific decoding for the operations defined previously.

8.2.6 XCoreIsaBitManipulation1
This extends XCoreFinal. It provides the functionality of all bit-wise operations (e.g. AND, bit-shifting, bit-extension).

8.2.7 XCoreIsaBitManipulationFinal
This extends XCoreArithmetic1, providing the instruction-specific decoding for the operations defined previously.

8.2.8 XCoreIsaBranching1
This extends XCoreFinal. It provides the functionality of all control flow operations, including conditional exception generation.

8.2.9 XCoreIsaBranchingFinal
This extends XCoreBranching1, providing the instruction-specific decoding for the operations defined previously.

8.2.10 XCoreIsaChannelEnd1
This extends XCoreFinal.It provides the functionality of all operations interacting with the channel-end resource type. Thus the machine introduces various state variables managing this functionality for all the 32 channel-ends provided by the XCore.

8.2.11 XCoreIsaChannelEndFinal
This extends XCoreChannelEnd1, providing the instruction-specific decoding for the operations defined previously.

8.2.12 XCoreIsaClock1
This extends XCoreFinal. It provides the basic functionality of all operations interacting with the clock resource type.

8.2.13 XCoreIsaClock2
This extends XCoreIsaClock1. It extends the events defined in XCoreIsaClock1 to provide the exact operations operations interacting with the clock resource type. Clock-specific state variables are introduced in order to effect this.

8.2.14 XCoreIsaClockFinal
This extends XCoreClock2, providing the instruction-specific decoding for the operations defined previously.

8.2.15 XCoreIsaLoadAddress1
This extends XCoreFinal. It provides the basic form of the load-address operations (i.e. load-forward and load-backward).

8.2.16 XCoreIsaLoadAddressFinal
This extends XCoreLoadAddress1, providing the instruction-specific decoding for the operations defined previously.

8.2.17 XCoreIsaLock1
This extends XCoreFinal. It provides the functionality of all operations interacting with the lock resource type. The machine introduces state variables managing this functionality.

8.2.18 XCoreIsaLockFinal
This extends XCoreLock1, providing the instruction-specific decoding for the operations defined previously.

8.2.19 XCoreIsaMemoryRead1
This extends XCoreFinal. It provides the functionality of all operations reading from the memory system (i.e. 8, 16 and 32-bit).

8.2.20 XCoreIsaMemoryReadFinal
This extends XCoreMemoryRead1, providing the instruction-specific decoding for the operations defined previously.

8.2.21 XCoreIsaMemoryWrite1
This extends XCoreFinal. . It provides the functionality of all operations writing to the memory system (i.e. 8, 16 and 32-bit).

8.2.22 XCoreIsaMemoryWriteFinal
This extends XCoreMemoryWrite1, providing the instruction-specific decoding for the operations defined previously.

8.2.23 XCoreIsaPort1
This extends XCoreFinal. It provides the functionality of all operations interacting with the lock resource type. The full configurability of the port devices is not yet specified. The machine introduces state variables managing this functionality.

8.2.24 XCoreIsaPortFinal
This extends XCorePort1, providing the instruction-specific decoding for the operations defined previously.

8.2.25 XCoreIsaPortConnections
This extends XCoreIsaPortFinal. It provides example loop-backs for some of the ports on the device, in order to allow port-to-port communication tests to be implemented.

8.2.26 XCoreIsaProcessorState1
This extends XCoreFinal. It provides the functionality of all operations interacting with the processor state internal registers. The machine introduces a state variable managing this functionality.

8.2.27 XCoreIsaProcessorStateFinal
This extends XCoreProcessorState1, providing the instruction-specific decoding for the operations defined previously.

8.2.28 XCoreIsaRegisterSet1
This extends XCoreFinal. It provides the functionality of all register-to-register transfer operations, including inter-thread register transfers.

8.2.29 XCoreIsaRegisterSetFinal
This extends XCoreRegisterSet1, providing the instruction-specific decoding for the operations defined previously.

8.2.30 XCoreIsaResource1
This extends XCoreFinal. It provides the functionality of instructions that interact with multiple resource types, when operating in conditions common to all (e.g. unknown resource type exceptions).

8.2.31 XCoreIsaResourceFinal
This extends XCoreResource1, providing the instruction-specific decoding for the operations defined previously.

8.2.32 XCoreIsaThread1
This extends XCoreFinal. It provides the functionality of all thread-manipulation operations, including their interactions with the inter-thread synchroniser resource type. The machine introduces various state variables managing this functionality. This machine encapsulates the most functionally complex part of the ISA.

8.2.33 XCoreIsaThreadFinal
This extends XCoreThread1, providing the instruction-specific decoding for the operations defined previously.

8.2.34 XCoreIsaTimer1
This extends XCoreFinal. It provides the functionality of all operations interacting with the timer resource type. The machine introduces state variables managing this functionality.

8.2.35 XCoreIsaTimerFinal
This extends XCoreTimer1, providing the instruction-specific decoding for the operations defined previously.

8.2.36 XCoreSupportMch
This extends XCoreFinal. It provides the functionality of all operations occurring outside the decoding of valid instructions. These include loading of the executable

during boot, exceptions triggered during fetching of an instruction, and selection of the current active thread.

9 Testing

A complete coverage test-suite as described in Section 7.7 has not yet been produced for the XCore. However, a "demonstration" test-suite is provided, that exercises the most significant features of the ISA. This is described in more detail in the following sections.

9.1 Basic Output

Coded in XC, a simple integer for-loop outputting a message and count to the server's virtual console via a printf call.

9.2 Logic

Coded in XC, bitwise OR and XOR operations with results outputted via printf.

9.3 Arithmetic

Coded in XC, modulo divide operations with results outputted via printf.

9.4 Parallel Threads

Coded in XC, three parallel threads are launched using the XC par operation. Each thread performs a simple integer for-loop outputting a message and count via printf, before terminating.

9.5 Channel Communication

Coded in XC, two parallel threads are launched using the XC par operation. One thread transmits a 32-bit integer to the other via a channel. Transmitted and received data values are outputted via printf.

A similar test is then executed, communicating via a channel in streaming mode.

9.6 Port Communication

Coded in XC, two parallel threads are launched using the XC par operation. One thread transmits a 32-bit integer to the other via two ports. This test relies on the looping back of ports 0 and 1 in order to allow inter-communication, as implemented by machine *XCoreIsaPortConnections* described in Section 8.2.25. Transmitted and received data values are outputted via printf.

9.7 Timer Event

Coded in XC. A single thread uses the XC select/ timerafter construct to wait for 1000 timer ticks and unblock on timer timeout. Timer results are outputted via printf.

9.8 Channel Event

Coded in XC, two parallel threads are launched using the XC par operation. One thread transmits a 32-bit integer to the other via a channel. The receiving thread uses the XC select construct to wait for received data and unblock on receipt. Transmitted and received data values are outputted via printf.

9.9 Timer / Channel Event

Coded in XC, two parallel threads are launched using the XC par operation. One thread transmits a 32-bit integer to the other via a channel. The receiving thread uses

the XC select and timeafter construct to simultaneously wait for received data and timer timeout and unblock when either occurs. Transmitted and received data values, and timer values, are outputted via printf.

9.10 Interrupt Handling

Coded in XC, C and XCore assembler, an XC-coded interrupt handler is installed on port-1. A 32-integer is transmitted from port-0 to port-1 via the loopback described in Section 9.6. A received data value is outputted via printf from within the interrupt handler. The interrupt handler is then uninstalled.

9.11 Exception Handling

Coded in XC, C and XCore assembler, an XC-coded exception handler is installed. An exception is generated by performing a divide-by-zero calculation and the exception signalled via printf from within the handler, before control-flow is restored back to the test. The exception handler is then uninstalled.

9.12 Test Coverage

The XC/C/assembler test-suite described here currently exercises only 185 of the 690 events constructed by the model.

10 Results

10.1 Specification Improvements

Formal construction of the ISA specification revealed several issues in the published specification, generally falling into three classes. Direct errors occurred where a compiled executable would not execute under the specified behaviour: examples of these were certain incorrectly specified operand field locations within an instruction. Ambiguities can exist which can be open to misinterpretation and potentially lead to differences in behaviour: for example addition calculations being in fact modulo operation rather than a conventional addition with supporting exception condition on overflow. Finally, omissions of certain functional conditions were discovered, typically esoteric exception scenarios. For example, the requirement for dissimilar destination registers during dual-destination instructions enabled discovery of an undocumented exception implemented by the XCore for these instructions: the actual machine robustly detects this error condition and yields an exception. Another instruction was specified with different behaviours depending on the value of one of its included immediate fields. Separate refinements were constructed for the two forms of the instruction from separate abstract events, effectively defining separate instructions, with differing exception conditions. This was found to match the approach taken by the XCore implementation.

10.2 Tool Usage Experiences

Two subsequent developments of the Rodin tool were found to be instrumental in scaling the construction methodology from the small-scale academic MIDAS ISA [26] to that of the XCore. The release of the Camille text editor allowed rapid manipulation of large Event-B machines and events, and allowed the development of a bespoke editing tool to be avoided, by the simple instruction of copy-and-paste and search-and-replace functionality. The relatively recent introduction of the Event-B EXTENDS capability enabled the elimination of vast amounts resource-consuming an potentially error-inducing repetition of model constructs, by allowing common properties to be placed in shared machines, as described in Section 8.1. The absence of either of these features would probably have resulted in construction of the model not being possible in the 12 month timescale of the project.

Although not essential to the completion of the project, the task of Proof Obligation discharge was greatly enhanced by the availability of the Relevance Filter plugin, the addition of configurable tactic profiles in the Rodin core, and the introduced opportunity for the developer to introduce custom proof tactics.

As experienced during the MIDAS project, plugin development was well supported by the Rodin architecture, allowing efficient plugin development. Rodin internal API migration was also found to be well supported, allowing the initial uplift of the B2C source from Rodin 0.8.2 to the then-current (2.1) to be completed quickly. However, extraction of a complete, coherent source-code set for the Rodin core from the open source domain was often problematic.

As expected at the outset of the project, the XCore ISA application revealed various scaling limitations within the Rodin tool, generally falling into three classes: editing, building, and proving. The Camille text editor, and its synchronisation mechanism

with the Rodin core database, was found to be very unreliable for large file, and the vertical partitioning described in Section 7.5 was initially introduced to overcome editing limitations. Rebuilding of the project context (excluding PO discharge) was also found to be slow compared to rebuilding of a conventional software project of similar sized, and again vertical partitioning was needed to mitigate this. Finally, the expected problems of scaling of automatic PO discharge were mitigated to some extent by the developments described before. However, the perennial problems of selection of pertinent hypotheses from large lists generated for large models, and the instantiation of universal quantifiers were encountered. As before this issues required developer intervention to achieve semi-automatic PO discharge.

10.3 Event Ratio

The project yielded one academic curiosity: the XCore ISA has a total of 209 decoded instructions (i.e. 176 operations, with 33 having two encodings, which thus yield separate events). The 690 events of the model therefore imply an instruction/event ratio of 3.3. For the MIDAS ISA [26], 34 instructions yielded 109 fully decomposed events, implying an instruction/event ratio of 3.2.

In the absence of other information, the MIDAS-derived figure was used to estimate the magnitude of the project at its beginning, and would seem to be a reasonable initial estimate when planning modelling and test-case coverage projects in future.

11 Future Work

This section lists suggested future developments based on the Event-B XCore model.

11.1 Model Enhancement

The *XCoreSupportMch* machine currently contains an event *UnknownInstruction*, intended to capture all undecoded instruction codes within the instruction space. A single guard detecting a zero-register format with parameters 0/0/0 has been included in the event as an example. Explicit events detecting all the undecoded instructions in the ISA instruction space should be added.

Events and state for all configuration effects for the Clock and Port resource types should be inserted. This task represents a significant project in its own right.

As described in Section 7.5, much unnecessary repetition of common terms has been eliminated by use of axiom-defined macro statements and structuring using the EXTENDS mechanism. However, much term repetition still remains, particularly in the XCore<Name>Final machines, in which individual instruction decoding is defined. This may in future be alleviated by Event-B "recomposition" methods under development, and should be employed if possible.

11.2 Document generation

As described in Section 5, the model is intended as the basis of a natural language document specifying the XCore ISA behaviour under all conditions: this task is to be completed. The model could also be the basis of a linked mark-up language text document, automatically generated via a B2C-derived plugin. This would provide easy browsing and a well-indexed skeleton for the natural language version.

11.3 Enhanced PO Discharge

Is described in Section 7.6, automatic proof discharge rates for greatly enhanced by the introduction of customised proof tactics and tactic proofs, as well as the available "Relevance Filter" plugin. However, considerable opportunities exist for improving on these figures, by further enhancements to the Rodin tool and deployment of now-maturing plugins such as [14].

11.4 Abstract model

As described in Section 7.4 the model is structured with the intention of being the basis of a refined abstract model for some or all of the ISAs functionality, in order to allow superior review and analysis of the ISAs behaviour. A particularly relevant aspect of the XCore ISA's behaviour is the multi-thread synchronisation mechanism.

11.5 Model Checking

As well as the abstract modelling techniques described in Section 11.4, the existing model may be analysed using both model checking tools [13], and the introduction of partition-wide or model-wide invariant theorems.

11.6 Enhanced Test-suite

As described in Section 9.12, the existing test-suite only covers 185 of the 690 events of the model. Therefore a complete coverage test-suite, employing the techniques described in [26] would be highly desirable.

12 Summary

A formal functional specification of the complete ISA of the XMOS XCore microprocessor has been produced as a product of a Bristol University Knowledge Transfer Secondment (KTS). The ISA was constructed in the Event-B notation, using the Rodin toolset. The project applied and extended the Event-B and RODIN based techniques for Instruction Set Architecture (ISA) analysis, developed by Dr Stephen Wright.

The project achieved most but not all of the goals outlined in the initial proposal. The project's primary goal, the construction of an Event-B description of the complete ISA, was achieved and has been transferred to XMOS Ltd. XMOS has agreed to release the specification in the public domain. XMOS's Technical Director of Software Tools has been appointed to nominally oversee the model and its public availability will allow its further development by Dr Wright and the Rodin community.

Completion of the model has driven the enhancement of the employed methodology and tools, and numerous corrections to the existing ISA specification have been identified and reported via the XMOS reporting system. The technical aspect of the project has been dominated by the complexity of functionality specific to the XCore ISA, and the model provides a valuable test-article for further analysis. Enhancements made to the Rodin tool's automatic prover have been submitted to the open-source community for inclusion into further releases.

The results of the project have imformed future development of the model itself, the ISA modelling methodology, and the Rodin toolset.

13 References

[1] Abrial,J-R Butler,M Hallerstede,S Voisin,L 2006. “An Open Extensible Tool Environment for Event-B”, Formal Methods and Software Engineering

[2] Abrial,J-R Butler,M Hallerstede,S 2008. “Rodin: an open toolset for modelling and reasoning in Event-B”, International Journal On Software Tools For Technology Transfer

[3] Arons,T Elster,E Fix,L Mador-Haim,S Mishaeli,M Shalev,J Singerman,E Tiemeyer,A Vardi,M Zuck,L 2005. "Formal Verification of Backward Compatibility of Microcode", Verification of Hardware, Microcode, and Synchronous Systems

[4] Basin,D Friedrich,S Gawkowski,M 2006. "Bytecode Verification by Model Checking", Journal of Automated Reasoning

[5] Bendisposto,J Fritz,F Jastram,M Leuschel,M Weigelt,I 2011 "Developing Camille, a text editor for Rodin", Software: Practice and Experience

[6] Beyer,S Jacobi,C Kröning,D Leinenbach,D Paul,W 2006. "Putting it all together – Formal verification of the VAMP", International Journal on Software Tools for Technology Transfer

[7] Bowen,J 1987. “Formal specification and documentation of microprocessor instruction sets”, Microprocessing and Microprogramming Volume 21, Issue 1-5

[8] Boyer,R Yu,Y 1996. "Automated proofs of object code for a widely used microprocessor", Journal of the ACM Volume 43 Issue 1

[9] Brock,B Hunt,W 1991. “Report on the Formal Specification and Partial Verification of the VIPER Microprocessor”, Proceedings of the Sixth Annual Conference on Computer Assurance, Systems Integrity, Software Safety and Process Security

[10] Fox,A 2003. "Formal Specification and Verification of ARM6", Theorem Proving in Higher Order Logics

[11] Graham,B Birtwistle,G 1990. "Formalising the design of an SECD chip", Hardware Specification, Verification and Synthesis: Mathematical Aspects

[12] Hunt, W 1994. “FM8501: A Verified Microprocessor”, Lecture Notes in Artificial Intelligence Subseries of Lecture Notes in Computer Science, Springer-Verlag

[13] Leuschel,M Butler,M “ProB: A Model Checker for B”, *Proceedings Formal Methods Europe 2003*, 2003

[14] Maamria,I Butler,M, 2010 "On an Extensible Rule-based Prover for Event-B" ABZ2010

[15] May, D 2008. “The XMOS XS1 Architecture”, XMOS Ltd

[16] Metayer, C Abrial, J-R, Voisin, L 2005. “RODIN Deliverable 3.2 Event-B Language”, http//rodin.cs.ncl.ac.uk

[17] Moore,J 2003. "Proving theorems about Java and the JVM with ACL2", Proceedings of the NATO Advanced Study Institute on Models, Algebras and Logic of Engineering Software, Marktoberdorf

[18] Moore,J Lynch,T Kaufmann,M 1998. "A mechanically checked proof of the AMD5K86 floating-point division program", IEEE Transactions on Computers Volume 47 Issue 9

[19] Page,D 2008 “CRISP: A Cryptographic RISC Processor”, page@cs.bris.ac.uk.

[20] Röder, J 2010 "Relevance Filters for Event-B", ETH Zürich

[21] Snook,C Butler,M 2006. “UML-B: Formal modeling and design aided by UML”, ACM Transactions on Software Engineering and Methodology

[22] Srivas,M Miller,S 1995. “Formal Verification of an Avionics Microprocessor”, Langley Research Center

[23] Watt,D 2009 “Programming XC on XMOS Devices”, XMOS Ltd.

[24] Wright,S 2009 “Automatic Generation of C from Event-B”, Workshop on Integration of Model-based Formal Methods and Tools.

[25] Wright,S 2009 “Using Event-B to Create Instruction Set Architectures”, Formal Aspects of Computing Journal

[26] Wright,S 2009 “Formal Construction of Instruction Set Architectures”, University of Bristol

14 Formal Specification Listing

14.1 Machines

14.1.1 machine XCore1

```
machine XCore1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

invariants

  @statusType             status ∈ CoreStatus
  @interruptTriggeredType interruptTriggered ∈ BOOL
  @memArrayType           memArray ∈ ExtMemArrayDom → Bits
  @resourceTriggeredType  resourceTriggered ∈ BOOL

  @threadIndex            threadIndex ∈ ThreadIndexDom // Currently processed thread
  @threadStatusType       threadStatus ∈ ThreadIndexDom → ThreadStatus
  @threadRegArrayType     threadRegArrays ∈ ThreadIndexDom → RegArray
  @threadPcType           threadPc ∈ ThreadIndexDom → Bits32Nat

  theorem @nat4InRegADom    Bits4Nat⊆RegArrayDom
  theorem @threadPcThrdIdx  threadPc(threadIndex)∈Bits32Nat
  theorem @regArrayDom      dom(threadRegArrays(threadIndex))=RegArrayDom
  theorem @threadStatusRaw  threadStatus∈ℤ ⇸ ℤ
  theorem @threadPcRaw      threadPc∈ℤ ⇸ ℤ
  theorem @thrdRegArrayRaw  threadRegArrays(threadIndex)∈ℤ ⇸ ℙ(ℤ × Bit)
  theorem @threadRegArrays   ∀i·i∈RegArrayDom ⇒ (threadRegArrays(threadIndex))(i)∈Bits
  theorem @getMemBits32Bits  ∀a·a∈MemArrayDom ⇒ GetMemBits32(memArray ↦ a)∈Bits

  theorem @runningOkPcCheck RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered) =
                            bool(ThreadActive(status↦threadStatus↦threadIndex)=TRUE ∧
                                  PcOk(threadPc↦threadIndex)=TRUE ∧
                                   (interruptTriggered=FALSE))

  theorem @threadPcInPcDom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
```

```
                        threadPc(threadIndex)∈PcDom

theorem @getInstrIsInst RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        GetInstr(memArray ↦ threadPc(threadIndex))∈Inst

theorem @X1r_op1_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         X1r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X1r_op1_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         threadRegArrays(threadIndex)(X1r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X2r_op1_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         X2r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X2r_op1_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         threadRegArrays(threadIndex)(X2r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X2r_op2_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         X2r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X2r_op2_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         threadRegArrays(threadIndex)(X2r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL2r_op1_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         XL2r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL2r_op1_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         threadRegArrays(threadIndex)(XL2r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL2r_op2_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         XL2r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL2r_op2_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         threadRegArrays(threadIndex)(XL2r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XR2r_op1_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         XR2r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XR2r_op1_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                         threadRegArrays(threadIndex)(XR2r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XR2r_op2_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
```

```
                        XR2r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XR2r_op2_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XR2r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XLr2r_op1_dom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XLr2r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XLr2r_op1_Bits RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XLr2r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XLr2r_op2_dom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XLr2r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XLr2r_op2_Bits RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XLr2r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X2rus_op1_dom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        X2rus_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X2rus_op1_Bits RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(X2rus_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X2rus_op2_dom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        X2rus_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X2rus_op2_Bits RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(X2rus_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL2rus_op1_dom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL2rus_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL2rus_op1_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL2rus_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL2rus_op2_dom  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL2rus_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL2rus_op2_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL2rus_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X3r_op1_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
```

```
                        X3r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X3r_op1_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(X3r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X3r_op2_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        X3r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X3r_op2_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(X3r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @X3r_op3_dom    RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        X3r_op3(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @X3r_op3_Bits   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(X3r_op3(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL3r_op1_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL3r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL3r_op1_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL3r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL3r_op2_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL3r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL3r_op2_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL3r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL3r_op3_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL3r_op3(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL3r_op3_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL3r_op3(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL4r_op1_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL4r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL4r_op1_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL4r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL4r_op2_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
```

```
                        XL4r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL4r_op2_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL4r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL4r_op3_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL4r_op3(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL4r_op3_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL4r_op3(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL4r_op4_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL4r_op4(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL4r_op4_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL4r_op4(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL5r_op1_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL5r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL5r_op1_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL5r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL5r_op2_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL5r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL5r_op2_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL5r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL5r_op3_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL5r_op3(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL5r_op3_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL5r_op3(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL5r_op4_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL5r_op4(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL5r_op4_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL5r_op4(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL5r_op5_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
```

```
                        XL5r_op5(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL5r_op5_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL5r_op5(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL6r_op1_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL6r_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL6r_op1_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL6r_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL6r_op2_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL6r_op2(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL6r_op2_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL6r_op2(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL6r_op3_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL6r_op3(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL6r_op3_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL6r_op3(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL6r_op4_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL6r_op4(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL6r_op4_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL6r_op4(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL6r_op5_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL6r_op5(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL6r_op5_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL6r_op5(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XL6r_op6_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        XL6r_op6(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

theorem @XL6r_op6_Bits  RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                        threadRegArrays(threadIndex)(XL6r_op6(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

theorem @XRus_op1_dom   RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
```

```
                        XRus_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

  theorem @XRu6_op1_dom      RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                             XRu6_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

  theorem @XRu6_op1_Bits     RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                             threadRegArrays(threadIndex)(XRu6_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

  theorem @XLru6_op1_dom      RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                             XLru6_op1(GetInstr(memArray ↦ threadPc(threadIndex)))∈dom(threadRegArrays(threadIndex))

  theorem @XLru6_op1_Bits     RunningOkPcCheck(status↦threadStatus ↦ threadPc↦threadIndex↦interruptTriggered)=TRUE ⇒
                             threadRegArrays(threadIndex)(XLru6_op1(GetInstr(memArray ↦ threadPc(threadIndex))))∈Bits

events

  event INITIALISATION
    then
      @changeStatus     status ≔  LOADING
      @initIntTriggered interruptTriggered ≔  FALSE
      @initResTriggered resourceTriggered ≔  FALSE

      @initMemArray     memArray :∈ ExtMemArrayDom → Bits

      @threadIndex      threadIndex ≔  0 // Master thread taken from the start

      @threadStatus     threadStatus ≔  {0↦RUNNING} ⩤ {1↦FREE} ⩤ {2↦FREE} ⩤ {3↦FREE} ⩤
                                        {4↦FREE} ⩤    {5↦FREE} ⩤ {6↦FREE} ⩤ {7↦FREE}

      @threadRegs       threadRegArrays :∈ ThreadIndexDom → RegArray

      @threadPc         threadPc ≔  {0↦XCoreBootVector} ⩤ {1↦XCoreBootVector} ⩤ {2↦XCoreBootVector} ⩤ {3↦XCoreBootVector} ⩤
                                    {4↦XCoreBootVector} ⩤ {5↦XCoreBootVector} ⩤ {6↦XCoreBootVector} ⩤ {7↦XCoreBootVector}

  end

  event Loading
    where
      @statusCheck status = LOADING
    then
      @loadMem      memArray :∈ ExtMemArrayDom → Bits
      @changeStatus status :∈ {LOADING,RUNNING}
  end
```

```
event RunExternalInputs
  where
    @checkStat status = RUNNING
  then
    @interruptTriggered interruptTriggered:∈BOOL  // Interrupt state can be set or left false here
end

event SelectNextThread
  where
    @checkStat status = RUNNING
  then
    @threadIndex threadIndex :∈ ThreadIndexDom
end

event Exception
  any eeble ieble inenb inint ink sink waiting fast
      spcBits ssrBits etBits sedBits edBits srBits
      offset nextPc
  where
    @checkStat   status=RUNNING
    @checkStatus threadStatus(threadIndex)=RUNNING
    @spcBits     spcBits = NatToBits32(threadPc(threadIndex))
    @ssrBits     ssrBits=(threadRegArrays(threadIndex))(SrIndex)
    @sedBits     sedBits=(threadRegArrays(threadIndex))(EdIndex)
    @etBitsType  etBits∈Bits
    @edBitsType  edBits∈Bits
    @eeble       eeble   = EebleStatus(threadRegArrays↦threadIndex)
    @ieble       ieble   = IebleStatus(threadRegArrays↦threadIndex)
    @inenb       inenb   = InenbStatus(threadRegArrays↦threadIndex)
    @inint       inint   = InintStatus(threadRegArrays↦threadIndex)
    @ink         ink     = InkStatus(threadRegArrays↦threadIndex)
    @sink        sink    = SinkStatus(threadRegArrays↦threadIndex)
    @waiting     waiting = WaitingStatus(threadRegArrays↦threadIndex)
    @fast        fast    = FastStatus(threadRegArrays↦threadIndex)
    @nextSr      srBits  = EightBits1ToBits32(BitClr↦BitClr↦inenb↦inint↦BitSet↦sink↦waiting↦fast)
    @offsetType  offset ∈ Bits32Nat
    @nextPcType  nextPc  = ModuloAddUnsigned(RoundUnsigned(Bits32ToNat((threadRegArrays(threadIndex))(KepIndex))↦128)↦offset)
  then
    @changePc  threadPc(threadIndex) ≔ nextPc
    @changeReg threadRegArrays  ≔ threadRegArrays  {threadIndex↦{SpcIndex↦spcBits}}  {threadIndex↦{SsrIndex↦ssrBits}} 
                                                   {threadIndex↦{EtIndex↦etBits}}    {threadIndex↦{SedIndex↦sedBits}} 
                                                   {threadIndex↦{EdIndex↦edBits}}    {threadIndex↦{SrIndex↦srBits}}
```

```
end

event Interrupt
  any triggeredThreadIndex
      eeble ieble inenb inint ink sink waiting fast
      spcBits ssrBits etBits sedBits edBits srBits
      nextPc
  where
    @checkStat   status=RUNNING
    @thdIdxType  triggeredThreadIndex∈ThreadIndexDom
    @spcBits     spcBits = NatToBits32(threadPc(triggeredThreadIndex))
    @ssrBitsType ssrBits=(threadRegArrays(triggeredThreadIndex))(SrIndex)
    @sedBitsType sedBits=(threadRegArrays(triggeredThreadIndex))(EdIndex)
    @etBitsType  etBits ∈Bits
    @edBitsType  edBits ∈Bits
    @eeble       eeble   = EebleStatus(threadRegArrays↦triggeredThreadIndex)
    @ieble       ieble   = IebleStatus(threadRegArrays↦triggeredThreadIndex)
    @inenb       inenb   = InenbStatus(threadRegArrays↦triggeredThreadIndex)
    @inint       inint   = InintStatus(threadRegArrays↦triggeredThreadIndex)
    @ink         ink     = InkStatus(threadRegArrays↦triggeredThreadIndex)
    @sink        sink    = SinkStatus(threadRegArrays↦triggeredThreadIndex)
    @waiting     waiting = WaitingStatus(threadRegArrays↦triggeredThreadIndex)
    @fast        fast    = FastStatus(threadRegArrays↦triggeredThreadIndex)
    @eebleTrue   ieble   = BitSet
    @nextSr      srBits  = EightBits1ToBits32(BitClr↦BitClr↦inenb↦BitSet↦BitSet↦sink↦BitClr↦fast) // eeble=0 ieble=0 inint=1 ink=1 waiting=0
    @nextPcType  nextPc    ∈ Bits32Nat
  then
    @changePc  threadPc(triggeredThreadIndex) ≔ nextPc
    @changeReg threadRegArrays  ≔ threadRegArrays  {triggeredThreadIndex↦{SpcIndex↦spcBits}} 
                                                  {triggeredThreadIndex↦{SsrIndex↦ssrBits}} 
                                                   {triggeredThreadIndex↦{EtIndex↦etBits}} 
                                                    {triggeredThreadIndex↦{SedIndex↦sedBits}} 
                                                     {triggeredThreadIndex↦{EdIndex↦edBits}} 
                                                      {triggeredThreadIndex↦{SrIndex↦srBits}}
 end

event ResourceEvent
  any triggeredThreadIndex
      eeble ieble inenb inint ink sink waiting fast
      edBits srBits
      nextPc
  where
    @checkStat   status=RUNNING
```

```
    @thdIdxType  triggeredThreadIndex∈ThreadIndexDom
    @eeble       eeble    = EebleStatus(threadRegArrays↦triggeredThreadIndex)
    @ieble       ieble    = IebleStatus(threadRegArrays↦triggeredThreadIndex)
    @inenb       inenb    = InenbStatus(threadRegArrays↦triggeredThreadIndex)
    @inint       inint    = InintStatus(threadRegArrays↦triggeredThreadIndex)
    @ink         ink      = InkStatus(threadRegArrays↦triggeredThreadIndex)
    @sink        sink     = SinkStatus(threadRegArrays↦triggeredThreadIndex)
    @waiting     waiting  = WaitingStatus(threadRegArrays↦triggeredThreadIndex)
    @fast        fast     = FastStatus(threadRegArrays↦triggeredThreadIndex)
    @eebleTrue   eeble    = BitSet
    @edBitsType  edBits     ∈ Bits
    @nextSr      srBits   = EightBits1ToBits32(BitClr↦ieble↦BitClr↦inint↦ink↦sink↦BitClr↦fast)  // eeble=0 inenb=0 waiting=0
    @nextPcType  nextPc     ∈ Bits32Nat
  then
    @changeStatus threadStatus(triggeredThreadIndex) ≔ RUNNING
    @changeRegs   threadRegArrays ≔ threadRegArrays  {triggeredThreadIndex↦{EdIndex↦edBits}} 
                                                      {triggeredThreadIndex↦{SrIndex↦edBits}}
    @changePc     threadPc(triggeredThreadIndex) ≔ nextPc
end

event ThreadWaiting
  where
    @checkStat   status = RUNNING
    @checkStatus threadStatus(threadIndex) = WAITING
end

event ThreadMpausing
  any nextThreadStatus
  where
    @pausingCheck    MpausingCheck(status↦threadStatus↦threadIndex)=TRUE
    @nextThdStatType nextThreadStatus∈ThreadStatus
  then
    @changeStatus threadStatus(threadIndex) ≔ nextThreadStatus
end

event ThreadSpausing
  any nextThreadStatus
  where
    @pausingCheck    SpausingCheck(status↦threadStatus↦threadIndex)=TRUE
    @nextThdStatType nextThreadStatus∈ThreadStatus
  then
    @changeStatus threadStatus(threadIndex) ≔ nextThreadStatus
end
```

```
event PausingInstruction
  any instr
  where
    @pcCheck RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr   instr=GetInstr(memArray↦threadPc(threadIndex))
end

event WriteStatusInstruction
  any instr instrSize incrPc nextPc nextThreadStatus
  where
    @pcCheck         RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr           instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType       instrSize ∈ InstrSize
    @incrPc          incrPc=threadPc(threadIndex) + instrSize
    @nextPcType      nextPc=incrPc + 0
    @nextThdStatType nextThreadStatus∈ThreadStatus
  then
    @changePc     threadPc(threadIndex) ≔  nextPc
    @changeStatus threadStatus(threadIndex) ≔ nextThreadStatus
end

event WriteSrStatusInstruction
  any instr instrSize
      eeble ieble inenb inint ink sink waiting fast srBits
      incrPc nextPc nextThreadStatus
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @eeble      eeble   = EebleStatus(threadRegArrays↦threadIndex)
    @ieble      ieble   = IebleStatus(threadRegArrays↦threadIndex)
    @inenb      inenb   = InenbStatus(threadRegArrays↦threadIndex)
    @inint      inint   = InintStatus(threadRegArrays↦threadIndex)
    @ink        ink     = InkStatus(threadRegArrays↦threadIndex)
    @sink       sink    = SinkStatus(threadRegArrays↦threadIndex)
    @waiting    waiting = WaitingStatus(threadRegArrays↦threadIndex)
    @fast       fast    = FastStatus(threadRegArrays↦threadIndex)
    @srBitsType srBits∈Bits
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc=incrPc + 0
    @ntsType    nextThreadStatus∈ThreadStatus
  then
```

```
    @changeRegArray threadRegArrays ≔ threadRegArrays  {threadIndex↦{SrIndex↦srBits}}
    @changePc       threadPc(threadIndex) ≔  nextPc
    @changeStatus   threadStatus(threadIndex) ≔ nextThreadStatus
end

event Write1RegisterStatusInstruction
  any instr instrSize d dBits incrPc nextPc newThreadIndex newThreadStatus
  where
    @pcCheck       RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr         instr=GetInstr(memArray↦threadPc(threadIndex))
    @dType         d ∈ RegArrayDom
    @dBitsType     dBits ∈ Bits
    @iSizeType     instrSize ∈ InstrSize
    @incrPc        incrPc=threadPc(threadIndex) + instrSize
    @nextPcType    nextPc=incrPc + 0
    @newThdIdxType newThreadIndex∈ThreadIndexDom
    @newThdStsType newThreadStatus∈ThreadStatus
  then
    @changeStatus   threadStatus(newThreadIndex) ≔ newThreadStatus
    @changeRegArray threadRegArrays ≔ threadRegArrays  {threadIndex↦{d↦dBits}}
    @changePc       threadPc(threadIndex) ≔  nextPc
end

event Write0RegisterInstruction
  any instr instrSize incrPc nextPc
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc=incrPc + 0
  then
    @changePc   threadPc(threadIndex) ≔  nextPc
end

event Write1RegisterInstruction
  any instr instrSize d dBits incrPc nextPc
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @dType      d ∈ RegArrayDom
    @dBitsType  dBits ∈ Bits
    @iSizeType  instrSize ∈ InstrSize
```

```
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc=incrPc + 0
  then
    @changeRegArray threadRegArrays ≔ threadRegArrays  {threadIndex↦{d↦dBits}}
    @changePc       threadPc(threadIndex) ≔  nextPc
end

event Write2RegisterInstruction
  any instr instrSize d dBits e eBits incrPc nextPc
  where
    @pcCheck     RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr       instr=GetInstr(memArray↦threadPc(threadIndex))
    @dType       d ∈ RegArrayDom
    @dBitsType   dBits ∈ Bits
    @eType       e ∈ RegArrayDom
    @eBitsType   eBits ∈ Bits
    @deExclusive CompareNotEqualUnsigned(d↦e)=TRUE
    @iSizeType   instrSize ∈ InstrSize
    @incrPc      incrPc=threadPc(threadIndex) + instrSize
    @nextPcType  nextPc=incrPc + 0
  then
    @changeReg threadRegArrays  ≔ threadRegArrays  {threadIndex↦{d↦dBits}}  {threadIndex↦{e↦eBits}}
    @changePc  threadPc(threadIndex) ≔  nextPc
end

event Write1Memory0RegisterInstruction
  any instr instrSize addr memData memData0 incrPc nextPc
  where
    @pcCheck     RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr       instr=GetInstr(memArray↦threadPc(threadIndex))
    @addrType    addr∈Bits32Nat
    @addrOk      ByteAddrOkCheck(addr)=TRUE
    @memDataType memData∈Bits
    @memData0    memData0=ExtractBits8_0(memData)
    @iSizeType   instrSize ∈ InstrSize
    @incrPc      incrPc=threadPc(threadIndex) + instrSize
    @nextPcType  nextPc=incrPc + 0
  then
    @changeMem memArray  ≔ memArray  {addr↦memData0}
    @changePc  threadPc(threadIndex) ≔  nextPc
end

event Write2Memory0RegisterInstruction
```

```
  any instr instrSize addr memData memData0 memData1 incrPc nextPc
  where
    @pcCheck      RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr        instr=GetInstr(memArray↦threadPc(threadIndex))
    @addrType     addr∈Bits32Nat
    @addrOk       ShortAddrOkCheck(addr)=TRUE
    @memDataType  memData∈Bits
    @memData0     memData0=ExtractBits8_0(memData)
    @memData1     memData1=ExtractBits8_1(memData)
    @iSizeType    instrSize ∈ InstrSize
    @incrPc       incrPc=threadPc(threadIndex) + instrSize
    @nextPcType   nextPc=incrPc + 0
  then
    @changeMem memArray  := memArray  {addr↦memData0}  {addr+1↦memData1}
    @changePc  threadPc(threadIndex) :=  nextPc
end

event Write4Memory0RegisterInstruction
  any instr instrSize addr memData memData0 memData1 memData2 memData3  incrPc nextPc
  where
    @pcCheck      RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr        instr=GetInstr(memArray↦threadPc(threadIndex))
    @addrType     addr∈Bits32Nat
    @addrOk       WordAddrOkCheck(addr)=TRUE
    @memDataType  memData∈Bits
    @memData0     memData0=ExtractBits8_0(memData)
    @memData1     memData1=ExtractBits8_1(memData)
    @memData2     memData2=ExtractBits8_2(memData)
    @memData3     memData3=ExtractBits8_3(memData)
    @iSizeType    instrSize ∈ InstrSize
    @incrPc       incrPc=threadPc(threadIndex) + instrSize
    @nextPcType   nextPc=incrPc + 0
  then
    @changeMem memArray  := memArray  {addr↦memData0}  {addr+1↦memData1}  {addr+2↦memData2}  {addr+3↦memData3}
    @changePc  threadPc(threadIndex) :=  nextPc
end

event Write4Memory1RegisterInstruction
  any instr instrSize d dBits addr memData memData0 memData1 memData2 memData3  incrPc nextPc
  where
    @pcCheck      RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr        instr=GetInstr(memArray↦threadPc(threadIndex))
    @addrType     addr∈Bits32Nat
    @addrOk       WordAddrOkCheck(addr)=TRUE
```

```
    @memDataType memData∈Bits
    @memData0    memData0=ExtractBits8_0(memData)
    @memData1    memData1=ExtractBits8_1(memData)
    @memData2    memData2=ExtractBits8_2(memData)
    @memData3    memData3=ExtractBits8_3(memData)
    @dType       d ∈ RegArrayDom
    @dBitsType   dBits ∈ Bits
    @iSizeType   instrSize ∈ InstrSize
    @incrPc      incrPc=threadPc(threadIndex) + instrSize
    @nextPcType  nextPc=incrPc + 0
  then
    @changeMem      memArray  := memArray  {addr↦memData0}  {addr+1↦memData1}  {addr+2↦memData2}  {addr+3↦memData3}
    @changeRegArray threadRegArrays := threadRegArrays  {threadIndex↦{d↦dBits}}
    @changePc       threadPc(threadIndex) :=  nextPc
end

event WriteThreadRegisterInstruction
  any instr resIndex instrSize d dBits incrPc nextPc
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc=incrPc + 0
    @indexCheck resIndex∈ThreadIndexDom
    @dType      d ∈ RegArrayDom
    @dBitsType  dBits ∈ Bits
  then
    @changeRegArray threadRegArrays := threadRegArrays  {resIndex↦{d↦dBits}}
    @changePc       threadPc(threadIndex) :=  nextPc
end

event WriteThreadPcInstruction
  any instr instrSize resIdent resIndex nextThreadPc incrPc nextPc
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc=incrPc + 0
    @identType  resIdent∈Bits
    @identOk    IsThreadIdent(resIdent)=TRUE
    @index      resIndex=GetResourceIndex(resIdent)
```

```
    @indexCheck resIndex∈ThreadIndexDom
    @nextThrdPc nextThreadPc∈Bits32Nat
  then
    @changePc threadPc := threadPc  {resIndex↦nextThreadPc}  {threadIndex↦nextPc}
end

event BranchInstruction
  any instr instrSize incrPc nextPc
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc ∈ Bits32Nat
  then
    @changePc   threadPc(threadIndex) :=  nextPc
end

event DcallInstruction
  any instr instrSize index d dBits nextThreadPc incrPc nextPc
  where
    @pcCheck        RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr          instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType      instrSize ∈ InstrSize
    @incrPc         incrPc=threadPc(threadIndex) + instrSize
    @nextPcType     nextPc=incrPc + 0
    @indexType      index∈ThreadIndexDom
    @dType          d ∈ RegArrayDom
    @dBitsType      dBits ∈ Bits
    @nextThrdPcType nextThreadPc∈Bits32Nat
  then
    @changeStatus   status := DEBUGGING
    @changeRegArray threadRegArrays := threadRegArrays  {index↦{d↦dBits}}
    @changePc       threadPc := threadPc  {index↦nextThreadPc}
end

event DretInstruction
  any instr nextPc srBits
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @nextPcType nextPc ∈ Bits32Nat
    @srBitsType srBits∈Bits
```

```
  then
    @changeStatus status ≔ RUNNING
    @changePc     threadPc(threadIndex) ≔  nextPc
    @changeReg    threadRegArrays  ≔ threadRegArrays  {threadIndex↦{SrIndex↦srBits}}
end

event KretInstruction // refines BranchInstruction
  any instr instrSize incrPc nextPc edBits srBits
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc ∈ Bits32Nat
    @edBitsType edBits∈Bits
    @srBitsType srBits∈Bits
  then
    @changePc  threadPc(threadIndex) ≔  nextPc
    @changeReg threadRegArrays  ≔ threadRegArrays  {threadIndex↦{EdIndex↦edBits}}  {threadIndex↦{SrIndex↦srBits}}
end

event BranchLinkInstruction // refines BranchInstruction
  any instr instrSize incrPc nextPc lrBits
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @iSizeType  instrSize ∈ InstrSize
    @incrPc     incrPc=threadPc(threadIndex) + instrSize
    @nextPcType nextPc ∈ Bits32Nat
    @lrBits     lrBits=NatToBits32(incrPc)
  then
    @changeRegArray threadRegArrays ≔ threadRegArrays  {threadIndex↦{LrIndex↦lrBits}}
    @changePc       threadPc(threadIndex) ≔  nextPc
end

event RetSpInstruction // refines BranchInstruction
  any instr instrSize incrPc nextPc spBits lrBits
  where
    @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
    @spBitsDef  spBits∈Bits
    @lrBitsDef  lrBits∈Bits
    @iSizeType  instrSize ∈ InstrSize
```

```
      @incrPc     incrPc=threadPc(threadIndex) + instrSize
      @nextPcType nextPc ∈ Bits32Nat
    then
      @changeRegArray threadRegArrays := threadRegArrays ⩤ {threadIndex↦{SpIndex↦spBits}} ⩤ {threadIndex↦{LrIndex↦lrBits}}
      @changePc       threadPc(threadIndex) :=  nextPc
  end

end
```

14.1.2 machine XCore2

```
machine XCore2 // Extend exceptions

refines XCore1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events

  event INITIALISATION extends INITIALISATION end

  event Loading                           extends Loading                           end
  event RunExternalInputs                 extends RunExternalInputs                 end
  event SelectNextThread                  extends SelectNextThread                  end
  event ThreadWaiting                     extends ThreadWaiting                     end
  event ThreadMpausing                    extends ThreadMpausing                    end
  event ThreadSpausing                    extends ThreadSpausing                    end
  event Interrupt                         extends Interrupt                         end
  event ResourceEvent                     extends ResourceEvent                     end
  event WriteStatusInstruction            extends WriteStatusInstruction            end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  end
  event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  end
  event Write4Memory1RegisterInstruction  extends Write4Memory1RegisterInstruction  end
  event WriteThreadRegisterInstruction    extends WriteThreadRegisterInstruction    end
  event WriteThreadPcInstruction          extends WriteThreadPcInstruction          end
  event BranchInstruction                 extends BranchInstruction                 end
  event BranchLinkInstruction             extends BranchLinkInstruction             end
  event DcallInstruction                  extends DcallInstruction                  end
  event DretInstruction                   extends DretInstruction                   end
  event KretInstruction                   extends KretInstruction                   end
  event RetSpInstruction                  extends RetSpInstruction                  end
  event PausingInstruction                extends PausingInstruction                end

  event IllegalPcException extends Exception
```

```
  where
    @etBits etBits=NatToBits32(EtIllegalPc)
    @edBits edBits=NatToBits32(threadPc(threadIndex))
    @offset offset=0+0
end

event IllegalInstructionException extends Exception
  any instr
  where
    @pcCheck RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr   instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits  etBits=NatToBits32(EtIllegalInstruction)
    @edBits  edBits=NatToBits32(0)
    @offset  offset=0+0
end

event LinkErrorException extends Exception
  any instr resIdent
  where
    @pcCheck       RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr         instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits        etBits=NatToBits32(EtLinkError)
    @resIndentType resIdent∈Bits
    @edBits        edBits=ExtractBits32_0(resIdent)
    @offset        offset=0+0
end

event IllegalResourceException extends Exception
  any instr resIdent
  where
    @pcCheck       RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr         instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits        etBits=NatToBits32(EtIllegalResource)
    @resIndentType resIdent∈Bits
    @edBits        edBits=ExtractBits32_0(resIdent)
    @offset        offset=0+0
end

event LoadStoreException extends Exception
  any instr addr
  where
    @pcCheck  RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr    instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits   etBits=NatToBits32(EtLoadStore)
```

```
    @addrType addr∈Bits32Nat
    @edBits   edBits=NatToBits32(addr)
    @offset   offset=0+0
end

event IllegalPsException extends Exception
  any instr ident
  where
    @pcCheck   RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr     instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits    etBits=NatToBits32(EtIllegalPs)
    @identType ident∈Bits
    @edBits    edBits=ExtractBits32_0(ident)
    @offset    offset=0+0
end

event ArithmeticException extends Exception
  any instr
  where
    @pcCheck RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr   instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits  etBits=NatToBits32(EtArithmetic)
    @edBits  edBits=NatToBits32(0)
    @offset  offset=0+0
end

event EcallException extends Exception
  any instr
  where
    @pcCheck RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr   instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits  etBits=NatToBits32(EtEcall)
    @edBits  edBits=NatToBits32(0)
    @offset  offset=0+0
end

event ResourceDepException extends Exception
  any instr ident
  where
    @pcCheck   RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
    @instr     instr=GetInstr(memArray↦threadPc(threadIndex))
    @etBits    etBits=NatToBits32(EtResourceDep)
    @identType ident∈Bits
    @edBits    edBits=ExtractBits32_0(ident)
```

```
    @offset    offset=0+0
  end

  event KcallException extends Exception
    any instr callOperand
    where
      @pcCheck    RunningOkPcCheck(status↦threadStatus↦threadPc↦threadIndex↦interruptTriggered)=TRUE
      @instr      instr=GetInstr(memArray↦threadPc(threadIndex))
      @etBits     etBits=NatToBits32(EtKcall)
      @callOpType callOperand∈Bits
      @edBits     edBits=ExtractBits32_0(callOperand)
      @offset     offset=64+0
  end

end
```

14.1.3 machine XCoreFinal

```
machine XCoreFinal
refines XCore2
sees XCoreBasic

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events
  event INITIALISATION                    extends INITIALISATION                    end
  event Loading                           extends Loading                           end
  event RunExternalInputs                 extends RunExternalInputs                 end
  event SelectNextThread                  extends SelectNextThread                  end
  event ThreadWaiting                     extends ThreadWaiting                     end
  event ThreadMpausing                    extends ThreadMpausing                    end
  event ThreadSpausing                    extends ThreadSpausing                    end
  event LinkErrorException                extends LinkErrorException                end
  event IllegalPcException                extends IllegalPcException                end
  event IllegalInstructionException       extends IllegalInstructionException       end
  event IllegalResourceException          extends IllegalResourceException          end
  event LoadStoreException                extends LoadStoreException                end
  event IllegalPsException                extends IllegalPsException                end
  event ArithmeticException               extends ArithmeticException               end
  event EcallException                    extends EcallException                    end
  event ResourceDepException              extends ResourceDepException              end
  event KcallException                    extends KcallException                    end
  event Interrupt                         extends Interrupt                         end
  event ResourceEvent                     extends ResourceEvent                     end
  event WriteStatusInstruction            extends WriteStatusInstruction            end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  end
  event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  end
  event Write4Memory1RegisterInstruction  extends Write4Memory1RegisterInstruction  end
  event WriteThreadRegisterInstruction    extends WriteThreadRegisterInstruction    end
  event WriteThreadPcInstruction          extends WriteThreadPcInstruction          end
  event BranchInstruction                 extends BranchInstruction                 end
  event BranchLinkInstruction             extends BranchLinkInstruction             end
  event DcallInstruction                  extends DcallInstruction                  end
```

```
  event DretInstruction               extends DretInstruction               end
  event KretInstruction               extends KretInstruction               end
  event RetSpInstruction              extends RetSpInstruction              end
  event PausingInstruction            extends PausingInstruction            end
end
```

14.1.4 machine XCoreIsaArithmetic1

```
machine XCoreIsaArithmetic1 // Arithmetic instructions: functionality

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event Loading                           extends Loading                           where @false ⊥ end
  event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
  event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
  event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
  event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
  event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
  event LinkErrorException                extends LinkErrorException                where @false ⊥ end
  event IllegalPcException                extends IllegalPcException                where @false ⊥ end
  event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
  event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
  event LoadStoreException                extends LoadStoreException                where @false ⊥ end
  event IllegalPsException                extends IllegalPsException                where @false ⊥ end
  event ArithmeticException               extends ArithmeticException               where @false ⊥ end
  event EcallException                    extends EcallException                    where @false ⊥ end
  event ResourceDepException              extends ResourceDepException              where @false ⊥ end
  event KcallException                    extends KcallException                    where @false ⊥ end
  event Interrupt                         extends Interrupt                         where @false ⊥ end
  event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
  event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
```

```
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event LDC_OK extends Write1RegisterInstruction
  any x
  where
    @xType x ∈ Bits32Nat
    @dBits dBits = NatToBits32(x)
end

event ADD_OK extends Write1RegisterInstruction
  any  x y res
  where
    @xType x ∈ Bits32Nat
    @yType y ∈ Bits32Nat
    @res   res = ModuloAddUnsigned(x↦y)
    @dBits dBits = NatToBits32(res)
end

event SUB_OK extends Write1RegisterInstruction
  any x y res
  where
    @xType x ∈ Bits32Nat
    @yType y ∈ Bits32Nat
    @res   res = ModuloSubUnsigned(x↦y)
    @dBits dBits = NatToBits32(res)
end

event NEG_OK extends Write1RegisterInstruction
  any x res
  where
    @xType x ∈ Bits32Int
    @res   res = ModuloSubSigned(0↦x)
    @dBits dBits = IntToBits32(res)
```

```
end

event MUL_OK extends Write1RegisterInstruction
  any x y res
  where
    @xType x ∈ Bits32Nat
    @yType y ∈ Bits32Nat
    @res   res = ModuloMulUnsigned(x↦y)
    @dBits dBits = NatToBits32(res)
end

event DIVS_OK extends Write1RegisterInstruction
  any  x y divZero divMin res
  where
    @xType      x ∈ Bits32Int
    @yType      y ∈ Bits32Int
    @divZero    divZero=bool(y=0)
    @divZeroChk divZero=FALSE
    @divMin     divMin=bool((x=0) ∧ (y=MinBits32Int))
    @divMinChk  divMin=FALSE
    @res        res = DivSigned(x↦y)
    @regBits    dBits = IntToBits32(res)
end

event DIVS_DivZero extends ArithmeticException
  any y divZero
  where
    @yType      y ∈ Bits32Int
    @divZero    divZero=bool(y=0)
    @divZeroChk divZero=TRUE
end

event DIVS_DivMin extends ArithmeticException
  any x y divMin
  where
    @xType     x ∈ Bits32Int
    @yType     y ∈ Bits32Int
    @divMin    divMin=bool((x=0) ∧ (y=MinBits32Int))
    @divMinChk divMin=TRUE
end

event DIVU_OK extends Write1RegisterInstruction
  any  x y divZero res
  where
```

```
    @xType      x ∈ Bits32Nat
    @yType      y ∈ Bits32Nat
    @divZero    divZero=bool(y=0)
    @divZeroChk divZero=FALSE
    @res        res = DivUnsigned(x↦y)
    @dBits      dBits = NatToBits32(res)
end

event DIVU_DivZero extends ArithmeticException
  any y divZero
  where
    @yType      y ∈ Bits32Nat
    @divZero    divZero=bool(y=0)
    @divZeroChk divZero=TRUE
end

event REMS_OK extends Write1RegisterInstruction
  any  x y divZero divMin res
  where
    @xType      x ∈ Bits32Int
    @yType      y ∈ Bits32Int
    @divZero    divZero=bool(y=0)
    @divZeroChk divZero=FALSE
    @divMin     divMin=bool((x=0) ∧ (y=MinBits32Int))
    @divMinChk  divMin=FALSE
    @res        res = RemSigned(x↦y)
    @dBits      dBits = IntToBits32(res)
end

event REMS_DivZero extends ArithmeticException
  any y divZero
  where
    @yType      y ∈ Bits32Int
    @divZero    divZero=bool(y=0)
    @divZeroChk divZero=TRUE
end

event REMS_DivMin extends ArithmeticException
  any x y divMin
  where
    @xType     x ∈ Bits32Int
    @yType     y ∈ Bits32Int
    @divMin    divMin=bool((x=0) ∧ (y=MinBits32Int))
    @divMinChk divMin=TRUE
```

```
end

event REMU_OK extends Write1RegisterInstruction
  any  x y divZero res
  where
    @xType       x ∈ Bits32Nat
    @yType       y ∈ Bits32Nat
    @divZero     divZero=bool(y=0)
    @divZeroChk divZero=FALSE
    @res         res = RemUnsigned(x↦y)
    @dBits       dBits = NatToBits32(res)
end

event REMU_DivZero extends ArithmeticException
  any y divZero
  where
    @yType       y ∈ Bits32Nat
    @divZero     divZero=bool(y=0)
    @divZeroChk divZero=TRUE
end

event EQ_TRUE extends Write1RegisterInstruction
  any  x y
  where
    @xType  x ∈ Bits32Nat
    @yType  y ∈ Bits32Nat
    @eqDef  CompareEqualUnsigned(x↦y)=TRUE
    @dBits  dBits = NatToBits32(1)
end

event EQ_FALSE extends Write1RegisterInstruction
  any  x y
  where
    @xType   x ∈ Bits32Nat
    @yType   y ∈ Bits32Nat
    @eqDef   CompareEqualUnsigned(x↦y)=FALSE
    @dBits   dBits = NatToBits32(0)
end

event LSS_TRUE extends Write1RegisterInstruction
  any  x y
  where
    @xType  x ∈ Bits32Int
    @yType  y ∈ Bits32Int
```

```
    @ls      CompareLessSigned(x↦y)=TRUE
    @dBits   dBits = NatToBits32(1)
end

event LSS_FALSE extends Write1RegisterInstruction
  any  x y
  where
    @xType   x ∈ Bits32Int
    @yType   y ∈ Bits32Int
    @ls      CompareLessSigned(x↦y)=FALSE
    @dBits   dBits = NatToBits32(0)
end

event LSU_TRUE extends Write1RegisterInstruction
  any  x y
  where
    @xType   x ∈ Bits32Nat
    @yType   y ∈ Bits32Nat
    @ls      CompareLessUnsigned(x↦y)=TRUE
    @dBits   dBits = NatToBits32(1)
end

event LSU_FALSE extends Write1RegisterInstruction
  any  x y
  where
    @xType   x ∈ Bits32Nat
    @yType   y ∈ Bits32Nat
    @ls      CompareLessUnsigned(x↦y)=FALSE
    @dBits   dBits = NatToBits32(0)
end

event EXTDP_OK extends Write1RegisterInstruction
  any x y nextDp
  where
    @xType   x ∈ Bits32Nat
    @yType   y ∈ Bits32Nat
    @nextDp  nextDp = ModuloSubUnsigned(x↦ModuloMulUnsigned(y ↦ Bpw))
    @dBits   dBits = NatToBits32(nextDp)
end

event EXTSP_OK extends Write1RegisterInstruction
  any x y nextSp
  where
    @xType   x ∈ Bits32Nat
```

```
    @yType  y ∈ Bits32Nat
    @nextSp nextSp = ModuloSubUnsigned(x↦ModuloMulUnsigned(y ↦ Bpw))
    @dBits  dBits = NatToBits32(nextSp)
end

event LADD_OK extends Write2RegisterInstruction
  any carryBits carry x y res resBits
  where
    @xType     x ∈ Bits32Nat
    @yType     y ∈ Bits32Nat
    @cBitsType carryBits ∈ Bits
    @carry     carry = Bits1ToNat(ExtractBits1(carryBits↦0))
    @res       res = ModuloLaddUnsigned(AddUnsigned(x↦y)↦carry)
    @resBits   resBits = NatToBits64(res)
    @dBits     dBits = ExtractBits32_0(resBits)
    @eBits     eBits = ExtractBits32_1(resBits)
end

event LADD_IndexEqual extends IllegalInstructionException
  any d e
  where
    @dType       d∈RegArrayDom
    @eType       e∈RegArrayDom
    @deExclusive CompareEqualUnsigned(d↦e)=TRUE
end

event LSUB_OK extends Write2RegisterInstruction
  any borrowBits borrow x y res resBits
  where
    @xType     x ∈ Bits32Nat
    @yType     y ∈ Bits32Nat
    @bBitsType borrowBits∈Bits
    @borrow    borrow = Bits1ToNat(ExtractBits1(borrowBits↦0))
    @res       res = ModuloLsubUnsigned(SubUnsigned(x↦y)↦borrow)
    @resBits   resBits = NatToBits64(res)
    @dBits     dBits = ExtractBits32_0(resBits)
    @eBits     eBits = ExtractBits32_1(resBits)
end

event LSUB_IndexEqual extends IllegalInstructionException
  any d e
  where
    @dType       d∈RegArrayDom
    @eType       e∈RegArrayDom
```

```
    @deExclusive CompareEqualUnsigned(d↦e)=TRUE
end

event LMUL_OK extends Write2RegisterInstruction
  any x y v w res resBits
  where
    @xType   x ∈ Bits32Nat
    @yType   y ∈ Bits32Nat
    @vType   v ∈ Bits32Nat
    @wType   w ∈ Bits32Nat
    @res     res = ModuloLaddUnsigned(MulUnsigned(x↦y)↦AddUnsigned(v↦w))
    @resBits resBits = NatToBits64(res)
    @dBits   dBits = ExtractBits32_1(resBits)
    @eBits   eBits = ExtractBits32_0(resBits)
end

event LMUL_IndexEqual extends IllegalInstructionException
  any d e
  where
    @dType        d∈RegArrayDom
    @eType        e∈RegArrayDom
    @deExclusive CompareEqualUnsigned(d↦e)=TRUE
end

event LDIVU_OK extends Write2RegisterInstruction
  any x y v xv res resBits
  where
    @xType   x ∈ Bits
    @vType   v ∈ Bits
    @xv      xv = Bits64ToNat(TwoBits32ToBits64(x↦v))
    @yType   y ∈ Bits32Nat
    @yPos    y > 0
    @vLtY    Bits32ToNat(v) < y
    @acc     res = LdivUnsigned(xv↦y)
    @resBits resBits  = NatToBits64(res)
    @dBits   dBits = ExtractBits32_0(resBits)
    @eBits   eBits = ExtractBits32_1(resBits)
end

event LDIVU_IndexEqual extends IllegalInstructionException
  any d e
  where
    @dType        d∈RegArrayDom
    @eType        e∈RegArrayDom
```

```
    @deExclusive CompareEqualUnsigned(d↦e)=TRUE
end

event MACCS_OK extends Write2RegisterInstruction
  any accD  accE acc xBits x yBits y res resBits
  where
    @accDType  accD ∈ Bits
    @accEType  accE ∈ Bits
    @xBitsType xBits ∈ Bits
    @yBitsType yBits ∈ Bits
    @acc       acc = Bits64ToInt(TwoBits32ToBits64(accE↦accD))
    @x         x = Bits32ToInt(xBits)
    @y         y = Bits32ToInt(yBits)
    @res       res = ModuloLaddSigned(acc↦MulSigned(x↦y))
    @resBits   resBits = IntToBits64(res)
    @dBits     dBits = ExtractBits32_1(resBits)
    @eBits     eBits = ExtractBits32_0(resBits)
end

event MACCS_IndexEqual extends IllegalInstructionException
  any d e
  where
    @dType       d∈RegArrayDom
    @eType       e∈RegArrayDom
    @deExclusive CompareEqualUnsigned(d↦e)=TRUE
end

event MACCU_OK extends Write2RegisterInstruction
  any accD  accE acc xBits x yBits y res resBits
  where
    @accDType  accD ∈ Bits
    @accEType  accE ∈ Bits
    @xBitsType xBits ∈ Bits
    @yBitsType yBits ∈ Bits
    @acc       acc = Bits64ToNat(TwoBits32ToBits64(accE↦accD))
    @x         x = Bits32ToNat(xBits)
    @y         y = Bits32ToNat(yBits)
    @res       res = ModuloLaddUnsigned(acc↦MulUnsigned(x↦y))
    @resBits   resBits = NatToBits64(res)
    @dBits     dBits = ExtractBits32_1(resBits)
    @eBits     eBits = ExtractBits32_0(resBits)
end

event MACCU_IndexEqual extends IllegalInstructionException
```

```
    any d e
    where
      @dType        d∈RegArrayDom
      @eType        e∈RegArrayDom
      @deExclusive CompareEqualUnsigned(d↦e)=TRUE
  end

end
```

14.1.5 machine XCoreIsaArithmeticFinal

```
machine XCoreIsaArithmeticFinal// Arithmetic instructions: instruction encoding

refines XCoreIsaArithmetic1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event LDC_Ru6_OK extends LDC_OK
    where
      @instCheck IsRu6(instr↦13↦0)=TRUE
      @op1Fetch  d=XRu6_op1(instr)
      @op2Fetch  x=XRu6_op2(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event LDC_Lru6_OK extends LDC_OK
    where
      @instCheck IsLru6(instr↦13↦0)=TRUE
      @op1Fetch  d=XLru6_op1(instr)
      @op2Fetch  x=XLru6_op2(instr)
      @instrSize instrSize=InstSizeLong+0
  end

  event ADD_3r_OK extends ADD_OK
    any op2 op3
    where
      @instCheck Is3r(instr↦2)=TRUE
      @op1Fetch  d=X3r_op1(instr)
      @op2Fetch  op2=X3r_op2(instr)
      @op3Fetch  op3=X3r_op3(instr)
      @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
      @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
      @instrSize instrSize=InstSizeShort+0
  end
```

```
event ADD_2rus_OK extends ADD_OK
  any op2
  where
    @instCheck Is2rus(instr↦18)=TRUE
    @op1Fetch  d=X2rus_op1(instr)
    @op2Fetch  op2=X2rus_op2(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  y=X2rus_op3(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SUB_3r_OK extends SUB_OK
  any op2 op3
  where
    @instCheck Is3r(instr↦3)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end

event SUB_2rus_OK extends SUB_OK
  any op2
  where
    @instCheck Is2rus(instr↦19)=TRUE
    @op1Fetch  d=X2rus_op1(instr)
    @op2Fetch  op2=X2rus_op2(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  y=X2rus_op3(instr)
    @instrSize instrSize=InstSizeShort+0
end

event NEG_2r_OK extends NEG_OK
  any op2
  where
    @instCheck Is2r(instr↦18↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event MUL_L3r_OK extends MUL_OK
```

```
  any op2 op3
  where
    @instCheck IsL3r(instr↦7↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event DIVS_L3r_OK extends DIVS_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦8↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event DIVS_L3r_DivZero extends DIVS_DivZero
  any op3
  where
    @instCheck IsL3r(instr↦8↦12)=TRUE
    @op3Fetch  op3=XL3r_op3(instr)
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
end

event DIVS_L3r_DivMin extends DIVS_DivMin
  any op2 op3
  where
    @instCheck IsL3r(instr↦8↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
end

event DIVU_L3r_OK extends DIVU_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦9↦12)=TRUE
```

```
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event DIVU_L3r_DivZero extends DIVU_DivZero
  any op3
  where
    @instCheck IsL3r(instr↦9↦12)=TRUE
    @op3Fetch  op3=XL3r_op3(instr)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event REMS_L3r_OK extends REMS_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦24↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event REMS_L3r_DivZero extends REMS_DivZero
  any op3
  where
    @instCheck IsL3r(instr↦24↦12)=TRUE
    @op3Fetch  op3=XL3r_op3(instr)
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
end

event REMS_L3r_DivMin extends REMS_DivMin
  any op2 op3
  where
    @instCheck IsL3r(instr↦24↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
end
```

```
event REMU_L3r_OK extends REMU_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦25↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event REMU_L3r_DivZero extends REMU_DivZero
  any op3
  where
    @instCheck IsL3r(instr↦25↦12)=TRUE
    @op3Fetch  op3=XL3r_op3(instr)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event EQ_3r_TRUE extends EQ_TRUE
  any op2 op3
  where
    @instCheck Is3r(instr↦6)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end

event EQ_3r_FALSE extends EQ_FALSE
  any op2 op3
  where
    @instCheck Is3r(instr↦6)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end
```

```
event EQ_2rus_TRUE extends EQ_TRUE
  any op2
  where
    @instCheck Is2rus(instr↦22)=TRUE
    @op1Fetch  d=X2rus_op1(instr)
    @op2Fetch  op2=X2rus_op2(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  y=X2rus_op3(instr)
    @instrSize instrSize=InstSizeShort+0
end

event EQ_2rus_FALSE extends EQ_FALSE
  any op2
  where
    @instCheck Is2rus(instr↦22)=TRUE
    @op1Fetch  d=X2rus_op1(instr)
    @op2Fetch  op2=X2rus_op2(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  y=X2rus_op3(instr)
    @instrSize instrSize=InstSizeShort+0
end

event LSS_3r_TRUE extends LSS_TRUE
  any op2 op3
  where
    @instCheck Is3r(instr↦24)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end

event LSS_3r_FALSE extends LSS_FALSE
  any op2 op3
  where
    @instCheck Is3r(instr↦24)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end
```

```
event LSU_3r_TRUE extends LSU_TRUE
  any op2 op3
  where
    @instCheck Is3r(instr↦25)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end

event LSU_3r_FALSE extends LSU_FALSE
  any op2 op3
  where
    @instCheck Is3r(instr↦25)=TRUE
    @op1Fetch  d=X3r_op1(instr)
    @op2Fetch  op2=X3r_op2(instr)
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end

event EXTDP_U6_OK extends EXTDP_OK
  where
    @instCheck IsU6(instr↦14↦0↦14)=TRUE
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @op1Fetch  y=XU6_op1(instr)
    @d         d=DpIndex+0
    @instrSize instrSize=InstSizeShort+0
end

event EXTDP_Lu6_OK extends EXTDP_OK
  where
    @instCheck IsLu6(instr↦14↦0↦14)=TRUE
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @op1Fetch  y=XLu6_op1(instr)
    @d         d=DpIndex+0
    @instrSize instrSize=InstSizeLong+0
end

event EXTSP_U6_OK extends EXTSP_OK
  where
```

```
    @instCheck IsU6(instr↦14↦1↦14)=TRUE
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @op1Fetch  y=XU6_op1(instr)
    @d         d=SpIndex+0
    @instrSize instrSize=InstSizeShort+0
end

event EXTSP_Lu6_OK extends EXTSP_OK
  where
    @instCheck IsLu6(instr↦14↦1↦14)=TRUE
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @op1Fetch  y=XLu6_op1(instr)
    @d         d=SpIndex+0
    @instrSize instrSize=InstSizeLong+0
end

event LADD_L5r_OK extends LADD_OK
  any op2 op3 op5
  where
    @instCheck IsL5r(instr↦0↦1)=TRUE
    @op1Fetch  d=XL5r_op1(instr)
    @op2Fetch  op2=XL5r_op2(instr)
    @op3Fetch  op3=XL5r_op3(instr)
    @op4Fetch  e=XL5r_op4(instr)
    @op5Fetch  op5=XL5r_op5(instr)
    @carryBits carryBits=(threadRegArrays(threadIndex))(op5)
    @x         x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event LADD_L5r_IndexEqual extends LADD_IndexEqual
  where
    @instCheck IsL5r(instr↦0↦1)=TRUE
    @op1Fetch  d=XL5r_op1(instr)
    @op4Fetch  e=XL5r_op4(instr)
end

event LSUB_L5r_OK extends LSUB_OK
  any op2 op3 op5
  where
    @instCheck  IsL5r(instr↦1↦0)=TRUE
    @op1Fetch   d=XL5r_op1(instr)
    @op2Fetch   op2=XL5r_op2(instr)
    @op3Fetch   op3=XL5r_op3(instr)
```

```
    @op4Fetch   e=XL5r_op4(instr)
    @op5Fetch   op5=XL5r_op4(instr)
    @borrowBits borrowBits=(threadRegArrays(threadIndex))(op5)
    @x          x=Bits32ToInt((threadRegArrays(threadIndex))(op2))
    @y          y=Bits32ToInt((threadRegArrays(threadIndex))(op3))
    @instrSize  instrSize=InstSizeLong+0
end

event LSUB_L5r_IndexEqual extends LSUB_IndexEqual
  where
    @instCheck  IsL5r(instr↦1↦0)=TRUE
    @op1Fetch   d=XL5r_op1(instr)
    @op4Fetch   e=XL5r_op4(instr)
end

event LMUL_L6r_OK extends LMUL_OK
  any op2 op3 op5 op6
  where
    @instCheck IsL6r(instr↦0)=TRUE
    @op1Fetch  d=XL6r_op1(instr)
    @op2Fetch  op2=XL6r_op2(instr)
    @op3Fetch  op3=XL6r_op3(instr)
    @op4Fetch  e=XL6r_op4(instr)
    @op5Fetch  op5=XL6r_op5(instr)
    @op6Fetch  op6=XL6r_op6(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @v         v=Bits32ToNat((threadRegArrays(threadIndex))(op5))
    @w         w=Bits32ToNat((threadRegArrays(threadIndex))(op6))
    @instrSize instrSize=InstSizeLong+0
end

event LMUL_L6r_IndexEqual extends LMUL_IndexEqual
  where
    @instCheck IsL6r(instr↦0)=TRUE
    @op1Fetch  d=XL6r_op1(instr)
    @op4Fetch  e=XL6r_op4(instr)
end

event LDIVU_L5r_OK extends LDIVU_OK
  any op2 op3 op5
  where
    @instCheck IsL5r(instr↦0↦0)=TRUE
    @op1Fetch  d=XL5r_op1(instr)
    @op2Fetch  op2=XL5r_op2(instr)
```

```
    @op3Fetch  op3=XL5r_op3(instr)
    @op4Fetch  e=XL5r_op4(instr)
    @op5Fetch  op5=XL5r_op5(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @v         v=(threadRegArrays(threadIndex))(op5)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event LDIVU_L5r_IndexEqual extends LDIVU_IndexEqual
  where
    @instCheck IsL5r(instr↦0↦0)=TRUE
    @op1Fetch  d=XL5r_op1(instr)
    @op4Fetch  e=XL5r_op4(instr)
end

event MACCS_L4r_OK extends MACCS_OK
  any op2 op3
  where
    @instCheck IsL4r(instr↦1↦0)=TRUE
    @op1Fetch  d=XL4r_op1(instr)
    @op2Fetch  op2=XL4r_op2(instr)
    @op3Fetch  op3=XL4r_op3(instr)
    @op4Fetch  e=XL4r_op4(instr)
    @accE      accE=(threadRegArrays(threadIndex))(e)
    @accD      accD=(threadRegArrays(threadIndex))(d)
    @instrSize instrSize=InstSizeLong+0
end

event MACCS_L4r_IndexEqual extends MACCS_IndexEqual
  where
    @instCheck IsL4r(instr↦1↦0)=TRUE
    @op1Fetch  d=XL4r_op1(instr)
    @op4Fetch  e=XL4r_op4(instr)
end

event MACCU_L4r_OK extends MACCU_OK
  any op2 op3
  where
    @instCheck IsL4r(instr↦0↦1)=TRUE
    @op1Fetch  d=XL4r_op1(instr)
    @op2Fetch  op2=XL4r_op2(instr)
    @op3Fetch  op3=XL4r_op3(instr)
    @op4Fetch  e=XL4r_op4(instr)
    @accE      accE=(threadRegArrays(threadIndex))(e)
```

```
      @accD      accD=(threadRegArrays(threadIndex))(d)
      @instrSize instrSize=InstSizeLong+0
  end

  event MACCU_L4r_IndexEqual extends MACCU_IndexEqual
    where
      @instCheck IsL4r(instr↦0↦1)=TRUE
      @op1Fetch  d=XL4r_op1(instr)
      @op4Fetch  e=XL4r_op4(instr)
  end

end
```

14.1.6 machine XCoreIsaBitManipulation1

```
machine XCoreIsaBitManipulation1 // Bit-manipulation instructions: functionality

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION                   extends INITIALISATION end
  event Loading                          extends Loading                          where @false ⊥ end
  event RunExternalInputs                extends RunExternalInputs                where @false ⊥ end
  event SelectNextThread                 extends SelectNextThread                 where @false ⊥ end
  event ThreadWaiting                    extends ThreadWaiting                    where @false ⊥ end
  event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
  event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
  event LinkErrorException               extends LinkErrorException               where @false ⊥ end
  event IllegalPcException               extends IllegalPcException               where @false ⊥ end
  event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
  event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
  event LoadStoreException               extends LoadStoreException               where @false ⊥ end
  event IllegalPsException               extends IllegalPsException               where @false ⊥ end
  event ArithmeticException              extends ArithmeticException              where @false ⊥ end
  event EcallException                   extends EcallException                   where @false ⊥ end
  event ResourceDepException             extends ResourceDepException             where @false ⊥ end
  event KcallException                   extends KcallException                   where @false ⊥ end
  event Interrupt                        extends Interrupt                        where @false ⊥ end
  event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
  event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
  event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
  event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
  event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
  event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
  event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
  event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
  event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
  event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
```

```
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event AND_OK extends Write1RegisterInstruction
  any x y
  where
    @xType   x∈Bits
    @yType   y∈Bits
    @regBits dBits=AndBits(x↦y)
end

event ANDNOT_OK extends Write1RegisterInstruction
  any x y
  where
    @xType   x∈Bits
    @yType   y∈Bits
    @regBits dBits=AndNotBits(x↦y)
end

event OR_OK extends Write1RegisterInstruction
  any x y
  where
    @xType   x∈Bits
    @yType   y∈Bits
    @regBits dBits=OorBits(x↦y)
end

event XOR_OK extends Write1RegisterInstruction
  any x y
  where
    @xType   x∈Bits
    @yType   y∈Bits
    @regBits dBits=XorBits(x↦y)
end
```

```
event NOT_OK extends Write1RegisterInstruction
  any x
  where
    @xType   x∈Bits
    @regBits dBits=NegBits(x)
end

event ASHR_LowPosition_OK extends Write0RegisterInstruction
  any y
  where
    @yEq0  y<1
end

event ASHR_HighPosition_Zero_OK extends Write1RegisterInstruction
  any x y topBit
  where
    @xType     x∈Bits
    @yGt31     y>31
    @topBit    topBit=ExtractBits1(x↦31)
    @bitZero   Bits1ToNat(topBit)=0
    @regBits   dBits=ZeroExtendBits(x↦0)
end

event ASHR_HighPosition_One_OK extends Write1RegisterInstruction
  any x y topBit
  where
    @xType     x∈Bits
    @yGt31     y>31
    @topBit    topBit=ExtractBits1(x↦31)
    @bitZero   Bits1ToNat(topBit)=1
    @regBits   dBits=OneExtendBits(x↦0)
end

event ASHR_Zero_OK extends Write1RegisterInstruction
  any x y shifted topBitPos topBit
  where
    @xType     x∈Bits
    @yType     y∈Bits32Nat
    @yGt0      y≥1
    @yLt32     y≤31
    @topBit    topBit=ExtractBits1(x↦31)
    @bitZero   Bits1ToNat(topBit)=0
    @shifted   shifted=ShiftRightBits(x↦y)
    @topBitPos topBitPos=(31−y)
```

```
    @regBits   dBits=ZeroExtendBits(shifted↦topBitPos)
end

event ASHR_One_OK extends Write1RegisterInstruction
  any x y shifted topBitPos topBit
  where
    @xType     x∈Bits
    @yType     y∈Bits32Nat
    @yGt0      y≥1
    @yLt32     y≤31
    @topBit    topBit=ExtractBits1(x↦31)
    @bitZero   Bits1ToNat(topBit)=1
    @shifted   shifted=ShiftRightBits(x↦y)
    @topBitPos topBitPos=(31-y)
    @regBits   dBits=OneExtendBits(shifted↦topBitPos)
end

event SHL_OK extends Write1RegisterInstruction
  any x y
  where
    @xType   x∈Bits
    @yType   y∈Bits32Nat
    @regBits dBits=ShiftLeftBits(x↦y)
end

event SHR_OK extends Write1RegisterInstruction
  any  x y
  where
    @xType   x∈Bits
    @yType   y∈Bits32Nat
    @regBits dBits=ShiftRightBits(x↦y)
end

event BITREV_OK extends Write1RegisterInstruction
  any x
      byte0 byte1 byte2 byte3
      b0  b1  b2  b3  b4  b5  b6  b7  b8  b9  b10 b11 b12 b13 b14 b15
      b16 b17 b18 b19 b20 b21 b22 b23 b24 b25 b26 b27 b28 b29 b30 b31
      outByte0 outByte1 outByte2 outByte3
  where
    @xType     x∈Bits
    @byte0     byte0=ExtractBits8_0(x)
    @byte1     byte1=ExtractBits8_1(x)
    @byte2     byte2=ExtractBits8_2(x)
```

```
    @byte3     byte3=ExtractBits8_3(x)
    @b0        b0  =ExtractBits1_0(byte0)
    @b1        b1  =ExtractBits1_1(byte0)
    @b2        b2  =ExtractBits1_2(byte0)
    @b3        b3  =ExtractBits1_3(byte0)
    @b4        b4  =ExtractBits1_4(byte0)
    @b5        b5  =ExtractBits1_5(byte0)
    @b6        b6  =ExtractBits1_6(byte0)
    @b7        b7  =ExtractBits1_7(byte0)
    @b8        b8  =ExtractBits1_0(byte1)
    @b9        b9  =ExtractBits1_1(byte1)
    @b10       b10 =ExtractBits1_2(byte1)
    @b11       b11 =ExtractBits1_3(byte1)
    @b12       b12 =ExtractBits1_4(byte1)
    @b13       b13 =ExtractBits1_5(byte1)
    @b14       b14 =ExtractBits1_6(byte1)
    @b15       b15 =ExtractBits1_7(byte1)
    @b16       b16 =ExtractBits1_0(byte2)
    @b17       b17 =ExtractBits1_1(byte2)
    @b18       b18 =ExtractBits1_2(byte2)
    @b19       b19 =ExtractBits1_3(byte2)
    @b20       b20 =ExtractBits1_4(byte2)
    @b21       b21 =ExtractBits1_5(byte2)
    @b22       b22 =ExtractBits1_6(byte2)
    @b23       b23 =ExtractBits1_7(byte2)
    @b24       b24 =ExtractBits1_0(byte3)
    @b25       b25 =ExtractBits1_1(byte3)
    @b26       b26 =ExtractBits1_2(byte3)
    @b27       b27 =ExtractBits1_3(byte3)
    @b28       b28 =ExtractBits1_4(byte3)
    @b29       b29 =ExtractBits1_5(byte3)
    @b30       b30 =ExtractBits1_6(byte3)
    @b31       b31 =ExtractBits1_7(byte3)
    @outByte0  outByte0=EightBits1ToBits8(b31↦b30↦b29↦b28↦b27↦b26↦b25↦b24)
    @outByte1  outByte1=EightBits1ToBits8(b23↦b22↦b21↦b20↦b19↦b18↦b17↦b16)
    @outByte2  outByte2=EightBits1ToBits8(b15↦b14↦b13↦b12↦b11↦b10↦b9↦b8)
    @outByte3  outByte3=EightBits1ToBits8(b7↦b6↦b5↦b4↦b3↦b2↦b1↦b0)
    @regBits   dBits=FourBits8ToBits32(outByte0↦outByte1↦outByte2↦outByte3)
end

event BYTEREV_OK extends Write1RegisterInstruction
  any x byte0 byte1 byte2 byte3
  where
    @xType     x∈Bits
    @byte0     byte0=ExtractBits8_0(x)
```

```
    @byte1     byte1=ExtractBits8_1(x)
    @byte2     byte2=ExtractBits8_2(x)
    @byte3     byte3=ExtractBits8_3(x)
    @regBits   dBits=FourBits8ToBits32(byte3↦byte2↦byte1↦byte0)
end

event MKMSK_OK extends Write1RegisterInstruction
  any x
  where
    @xType   x∈Bits32Nat
    @regBits dBits=CreateMask(x)
end

event CLZ_OK extends Write1RegisterInstruction
  any x
  where
    @xType   x∈Bits
    @regBits dBits=NatToBits32(CountLeadingZeroes(x))
end

event CRC32_OK extends Write1RegisterInstruction
  any r dIn p
  where
    @rType   r∈Bits
    @dInType dIn∈Bits
    @pType   p∈Bits
    @regBits dBits=CrcBits(r↦dIn↦p↦32)
end

event CRC8_OK extends Write1RegisterInstruction
  any o dIn p crcBits
  where
    @oType   o∈Bits
    @dInType dIn∈Bits
    @pType   p∈Bits
    @regBits crcBits=CrcBits(o↦dIn↦p↦8)
    @crcBits dBits=ShiftRightBits(crcBits↦8)
end

event SEXT_LowPosition_OK extends Write0RegisterInstruction
  any y
  where
    @yType     y∈Bits32Nat
    @yEq0      y<1
```

```
end

event SEXT_HighPosition_OK extends Write0RegisterInstruction
  any y
  where
    @yType      y∈Bits32Nat
    @yGt31      y>31
end

event SEXT_One_OK extends Write1RegisterInstruction
  any x y bitPos bit
  where
    @xType      x∈Bits
    @yType      y∈Bits32Nat
    @yGt0       y≥1
    @yLt32      y≤31
    @bitPos     bitPos=y-1
    @bit        bit=ExtractBits1(x↦bitPos)
    @bitZero    Bits1ToNat(bit)=1
    @regBits    dBits=OneExtendBits(x↦bitPos)
end

event SEXT_Zero_OK extends Write1RegisterInstruction
  any x y bitPos bit
  where
    @xType      x∈Bits
    @yType      y∈Bits32Nat
    @yGt0       y≥1
    @yLt32      y≤31
    @bitPos     bitPos=y-1
    @bit        bit=ExtractBits1(x↦bitPos)
    @bitZero    Bits1ToNat(bit)=0
    @regBits    dBits=ZeroExtendBits(x↦bitPos)
end

event ZEXT_LowPosition_OK extends Write0RegisterInstruction
  any y
  where
    @yType      y∈Bits32Nat
    @yEq0       y<1
end

event ZEXT_HighPosition_OK extends Write0RegisterInstruction
  any y
```

```
    where
      @yType      y∈Bits32Nat
      @yGt31      y>31
  end

  event ZEXT_OK extends Write1RegisterInstruction
    any x y
    where
      @xType      x∈Bits
      @yType      y∈Bits32Nat
      @yGt0       y≥1 // Zero causes no action in spite of theoretical legal
      @yLt32      y≤31
      @regBits    dBits=ZeroExtendBits(x↦y)
  end

end
```

14.1.7 machine XCoreIsaBitManipulationFinal

```
machine XCoreIsaBitManipulationFinal // Bit-manipulation instructions: instruction encoding

refines XCoreIsaBitManipulation1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

          bitPositionMapping

invariants
  @bitPositionMapping bitPositionMapping ∈ 0‥11 → 1‥32

events

  event INITIALISATION extends INITIALISATION
    then
      @bitPositionMapping bitPositionMapping ≔  {0↦32} ⩤
                                                {1↦1} ⩤ {2↦2} ⩤ {3↦3} ⩤ {4↦4} ⩤ {5↦5} ⩤ {6↦6} ⩤ {7↦7} ⩤ {8↦8} ⩤
                                                {9↦16} ⩤ {10↦24} ⩤ {11↦32}
  end

  event AND_3r_OK extends AND_OK
    any op2 op3
    where
      @instCheck Is3r(instr↦7)=TRUE
      @op1Fetch  d=X3r_op1(instr)
      @op2Fetch  op2=X3r_op2(instr)
      @op3Fetch  op3=X3r_op3(instr)
      @x         x=(threadRegArrays(threadIndex))(op2)
      @y         y=(threadRegArrays(threadIndex))(op3)
      @instrSize instrSize=InstSizeShort+0
  end

  event OR_3r_OK extends OR_OK
    any op2 op3
    where
      @instCheck Is3r(instr↦8)=TRUE
      @op1Fetch  d=X3r_op1(instr)
      @op2Fetch  op2=X3r_op2(instr)
```

```
    @op3Fetch  op3=X3r_op3(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @y         y=(threadRegArrays(threadIndex))(op3)
    @instrSize instrSize=InstSizeShort+0
end

event XOR_L3r_OK extends XOR_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦1↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @y         y=(threadRegArrays(threadIndex))(op3)
    @instrSize instrSize=InstSizeLong+0
end

event ANDNOT_2r_OK extends ANDNOT_OK
  any op2
  where
    @instCheck Is2r(instr↦5↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(d)
    @y         y=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event NOT_2r_OK extends NOT_OK
  any op2
  where
    @instCheck Is2r(instr↦17↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event ASHR_L3r_LowPosition_OK extends ASHR_LowPosition_OK
  any op3
  where
    @instCheck IsL3r(instr↦2↦12)=TRUE
    @op3Fetch  op3=XL3r_op3(instr)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
```

```
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L3r_HighPosition_One_OK extends ASHR_HighPosition_One_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦2↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L3r_HighPosition_Zero_OK extends ASHR_HighPosition_Zero_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦2↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L3r_One_OK extends ASHR_One_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦2↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L3r_Zero_OK extends ASHR_Zero_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦2↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
```

```
    @op3Fetch  op3=XL3r_op3(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L2rus_LowPosition_OK extends ASHR_LowPosition_OK
  where
    @instCheck IsL2rus(instr↦18↦12)=TRUE
    @op3Fetch  y=XL2rus_op3(instr)
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L2rus_HighPosition_One_OK extends ASHR_HighPosition_One_OK
  any op2
  where
    @instCheck IsL2rus(instr↦18↦12)=TRUE
    @op1Fetch  d=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @op3Fetch  y=XL2rus_op3(instr)
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L2rus_HighPosition_Zero_OK extends ASHR_HighPosition_Zero_OK
  any op2
  where
    @instCheck IsL2rus(instr↦18↦12)=TRUE
    @op1Fetch  d=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @op3Fetch  y=XL2rus_op3(instr)
    @instrSize instrSize=InstSizeLong+0
end

event ASHR_L2rus_One_OK extends ASHR_One_OK
  any op2
  where
    @instCheck IsL2rus(instr↦18↦12)=TRUE
    @op1Fetch  d=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @op3Fetch  y=XL2rus_op3(instr)
    @instrSize instrSize=InstSizeLong+0
end
```

```
event ASHR_L2rus_Zero_OK extends ASHR_Zero_OK
  any op2
  where
    @instCheck IsL2rus(instr↦18↦12)=TRUE
    @op1Fetch  d=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @op3Fetch  y=XL2rus_op3(instr)
    @instrSize instrSize=InstSizeLong+0
end

event SHL_3r_OK extends SHL_OK
  any op2 op3
  where
    @instCheck Is3r(instr↦4)=TRUE
    @op1Fetch d=X3r_op1(instr)
    @op2Fetch op2=X3r_op2(instr)
    @op3Fetch op3=X3r_op3(instr)
    @x        x=(threadRegArrays(threadIndex))(op2)
    @y        y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
end

event SHL_2rus_OK extends SHL_OK
  any op2
  where
    @instCheck Is2rus(instr↦20)=TRUE
    @op1Fetch d=X2rus_op1(instr)
    @op2Fetch op2=X2rus_op2(instr)
    @x        x=(threadRegArrays(threadIndex))(op2)
    @op3Fetch y=X2rus_op3(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SHR_3r_OK extends SHR_OK
  any op2 op3
  where
    @instCheck Is3r(instr↦5)=TRUE
    @op1Fetch d=X3r_op1(instr)
    @op2Fetch op2=X3r_op2(instr)
    @op3Fetch op3=X3r_op3(instr)
    @x        x=(threadRegArrays(threadIndex))(op2)
    @y        y=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize instrSize=InstSizeShort+0
```

```
end

event SHR_2rus_OK extends SHR_OK
  any op2
  where
    @instCheck Is2rus(instr↦21)=TRUE
    @op1Fetch d=X2rus_op1(instr)
    @op2Fetch op2=X2rus_op2(instr)
    @x        x=(threadRegArrays(threadIndex))(op2)
    @op3Fetch y=X2rus_op3(instr)
    @instrSize instrSize=InstSizeShort+0
end

event BITREV_L2r_OK extends BITREV_OK
  any op2
  where
    @instCheck IsL2r(instr↦0↦0↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event BYTEREV_L2r_OK extends BYTEREV_OK
  any op2
  where
    @instCheck IsL2r(instr↦0↦1↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event MKMSK_2r_OK extends MKMSK_OK
  any op2
  where
    @instCheck Is2r(instr↦20↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @x         x=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event MKMSK_Rus_OK extends MKMSK_OK
  any op2
```

```
  where
    @instCheck IsRus(instr↦20↦1)=TRUE
    @op1Fetch  d=XRus_op1(instr)
    @op2Fetch  op2=XRus_op2(instr)
    @x         x=bitPositionMapping(op2)
    @instrSize instrSize=InstSizeShort+0
end

event CRC32_L3r_OK extends CRC32_OK
  any op1 op2 op3
  where
    @instCheck IsL3r(instr↦21↦12)=TRUE
    @op1Fetch  op1=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @r         r=(threadRegArrays(threadIndex))(op1)
    @dIn       dIn=(threadRegArrays(threadIndex))(op2)
    @p         p=(threadRegArrays(threadIndex))(op3)
    @instrSize instrSize=InstSizeLong+0
end

event CRC8_L4r_OK extends CRC8_OK
  any op1 op2 op3 op4
  where
    @instCheck IsL4r(instr↦0↦0)=TRUE
    @op1Fetch  op1=XL4r_op1(instr)
    @op2Fetch  op2=XL4r_op2(instr)
    @op3Fetch  op3=XL4r_op3(instr)
    @op4Fetch  op4=XL4r_op4(instr)
    @o         o=(threadRegArrays(threadIndex))(op1)
    @dIn       dIn=(threadRegArrays(threadIndex))(op2)
    @p         p=(threadRegArrays(threadIndex))(op3)
    @instrSize instrSize=InstSizeLong+0
end

event CLZ_L2r_OK extends CLZ_OK
  any op2
  where
    @instCheck IsL2r(instr↦1↦0↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end
```

```
event SEXT_2r_LowPosition_OK extends SEXT_LowPosition_OK
  any op2
  where
    @instCheck Is2r(instr↦6↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event SEXT_2r_HighPosition_OK extends SEXT_HighPosition_OK
  any op2
  where
    @instCheck Is2r(instr↦6↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event SEXT_2r_One_OK extends SEXT_One_OK
  any op2
  where
    @instCheck Is2r(instr↦6↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(d)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event SEXT_2r_Zero_OK extends SEXT_Zero_OK
  any op2
  where
    @instCheck Is2r(instr↦6↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @x         x=(threadRegArrays(threadIndex))(d)
    @y         y=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event SEXT_Rus_LowPosition_OK extends SEXT_LowPosition_OK
  where
    @instCheck IsRus(instr↦6↦1)=TRUE
    @op2Fetch  y=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
```

```
end

event SEXT_Rus_HighPosition_OK extends SEXT_HighPosition_OK
  where
    @instCheck IsRus(instr↦6↦1)=TRUE
    @op2Fetch  y=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SEXT_Rus_One_OK extends SEXT_One_OK
  where
    @instCheck IsRus(instr↦6↦1)=TRUE
    @op1Fetch  d=XRus_op1(instr)
    @x         x=(threadRegArrays(threadIndex))(d)
    @op2Fetch  y=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SEXT_Rus_Zero_OK extends SEXT_Zero_OK
  where
    @instCheck IsRus(instr↦6↦1)=TRUE
    @op1Fetch  d=XRus_op1(instr)
    @x         x=(threadRegArrays(threadIndex))(d)
    @op2Fetch  y=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event ZEXT_2r_LowPosition_OK extends ZEXT_LowPosition_OK
  where
    @instCheck IsRus(instr↦8↦1)=TRUE
    @op2Fetch  y=X2r_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event ZEXT_2r_HighPosition_OK extends ZEXT_HighPosition_OK
  where
    @instCheck IsRus(instr↦8↦1)=TRUE
    @op2Fetch  y=X2r_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event ZEXT_2r_OK extends ZEXT_OK
  where
    @instCheck IsRus(instr↦8↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
```

```
      @x          x=(threadRegArrays(threadIndex))(d)
      @op2Fetch   y=X2r_op2(instr)
      @instrSize  instrSize=InstSizeShort+0
  end

  event ZEXT_Rus_LowPosition_OK extends ZEXT_LowPosition_OK
    where
      @instCheck  Is2r(instr↦8↦0)=TRUE
      @op2Fetch   y=XRus_op2(instr)
      @instrSize  instrSize=InstSizeShort+0
  end

  event ZEXT_Rus_HighPosition_OK extends ZEXT_HighPosition_OK
    where
      @instCheck  Is2r(instr↦8↦0)=TRUE
      @op2Fetch   y=XRus_op2(instr)
      @instrSize  instrSize=InstSizeShort+0
  end

  event ZEXT_Rus_OK extends ZEXT_OK
    where
      @instCheck  Is2r(instr↦8↦0)=TRUE
      @op1Fetch   d=XRus_op1(instr)
      @x          x=(threadRegArrays(threadIndex))(d)
      @op2Fetch   y=XRus_op2(instr)
      @instrSize  instrSize=InstSizeShort+0
  end

end
```

14.1.8 machine XCoreIsaBranching1

```
machine XCoreIsaBranching1 // Branch instructions: functionality

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events

  event INITIALISATION extends INITIALISATION  end

  event Loading                          extends Loading                          where @false ⊥ end
  event RunExternalInputs                extends RunExternalInputs                where @false ⊥ end
  event SelectNextThread                 extends SelectNextThread                 where @false ⊥ end
  event ThreadWaiting                    extends ThreadWaiting                    where @false ⊥ end
  event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
  event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
  event LinkErrorException               extends LinkErrorException               where @false ⊥ end
  event IllegalPcException               extends IllegalPcException               where @false ⊥ end
  event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
  event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
  event LoadStoreException               extends LoadStoreException               where @false ⊥ end
  event IllegalPsException               extends IllegalPsException               where @false ⊥ end
  event ArithmeticException              extends ArithmeticException              where @false ⊥ end
  event EcallException                   extends EcallException                   where @false ⊥ end
  event ResourceDepException             extends ResourceDepException             where @false ⊥ end
  event KcallException                   extends KcallException                   where @false ⊥ end
  event Interrupt                        extends Interrupt                        where @false ⊥ end
  event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
  event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
  event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
  event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
  event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
  event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
  event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
  event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
  event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
```

```
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event ECALLT_OK extends Write0RegisterInstruction
  any c
  where
    @condTrue  c=0
end

event ECALLT_FALSE extends EcallException
  any c
  where
    @condTrue  c≠0
end

event ECALLF_OK extends Write0RegisterInstruction
  any c
  where
    @condFalse c≠0
end

event ECALLF_TRUE extends EcallException
  any c
  where
    @condFalse c=0
end

event KCALL_OK extends KcallException
end

event KRET_OK extends KretInstruction
  where
    @edBits edBits = (threadRegArrays(threadIndex))(SedIndex)
    @srBits srBits = (threadRegArrays(threadIndex))(SsrIndex)
    @nextPc nextPc = Bits32ToNat((threadRegArrays(threadIndex))(SpcIndex))
```

```
end

event BAU_OK extends BranchInstruction
end

event BLA_OK extends BranchLinkInstruction
end

event BLACP_BadAddress extends LoadStoreException
  any b i
  where
    @addr      addr  = b + (i *  Bpw)
    @addrCheck WordAddrOkCheck(addr)=FALSE
end

event BLACP_OK extends BranchLinkInstruction
  any b i addr
  where
    @addrType  addr∈Bits32Nat
    @addr      addr  = b + (i *  Bpw)
    @addrCheck WordAddrOkCheck(addr)=TRUE
    @nextPc    nextPc=Bits32ToNat(GetMemBits32(memArray↦addr))
end

event BLAT_BadAddress extends LoadStoreException
  any b i
  where
    @addr      addr = b + (i *  Bpw)
    @addrCheck WordAddrOkCheck(addr)=FALSE
end

event BLAT_OK extends BranchLinkInstruction
  any b i addr
  where
    @addrType  addr∈Bits32Nat
    @addr      addr = b + (i *  Bpw)
    @addrCheck WordAddrOkCheck(addr)=TRUE
    @nextPc    nextPc=Bits32ToNat(GetMemBits32(memArray↦addr))
end

event BLRB_OK extends BranchLinkInstruction
  any i
  where
    @nextPc nextPc = incrPc - (i * 2)
```

```
end

event BLRF_OK extends BranchLinkInstruction
  any i
  where
    @nextPc nextPc = incrPc + (i * 2)
end

event BRBF_TRUE_OK extends BranchInstruction
  any c i
  where
    @condTrue  c=0
    @nextPc    nextPc = incrPc - (i * 2)
end

event BRBF_FALSE_OK extends BranchInstruction
  any c
  where
    @condFalse c≠0
    @nextPc    nextPc = incrPc+0
end

event BRBT_TRUE_OK extends BranchInstruction
  any c i
  where
    @condTrue  c≠0
    @nextPc    nextPc = incrPc - (i * 2)
end

event BRBT_FALSE_OK extends BranchInstruction
  any c
  where
    @condFalse c=0
    @nextPc    nextPc = incrPc+0
end

event BRBU_OK extends BranchInstruction
  any i
  where
    @nextPc nextPc = incrPc - (i * 2)
end

event BRFF_TRUE_OK extends BranchInstruction
  any c i
  where
```

```
    @condTrue c=0
    @nextPc nextPc = incrPc + (i * 2)
end

event BRFF_FALSE_OK extends BranchInstruction
  any c
  where
    @condFalse c≠0
    @nextPc nextPc = incrPc+0
end

event BRFT_TRUE_OK extends BranchInstruction
  any c i
  where
    @condTrue c≠0
    @nextPc nextPc = incrPc + (i * 2)
end

event BRFT_FALSE_OK extends BranchInstruction
  any c
  where
    @condFalse c=0
    @nextPc    nextPc = incrPc+0
end

event BRFU_OK extends BranchInstruction
  any i
  where
    @nextPc nextPc = incrPc + (i * 2)
end

event BRU_OK extends BranchInstruction
  any i
  where
    @nextPc nextPc = incrPc + (i * 2)
end

event RETSP_Zero_OK extends BranchInstruction
  any i
  where
    @condFalse i≤0
    @nextPc    nextPc=Bits32ToNat((threadRegArrays(threadIndex))(LrIndex))
end

event RETSP_BadAddress extends LoadStoreException
```

```
    any b i
    where
      @condTrue  i>0
      @addr      addr = b + (i * Bpw)
      @addrCheck WordAddrOkCheck(addr)=FALSE
   end

  event RETSP_OK extends RetSpInstruction
    any addr b i lr
    where
      @condTrue  i>0
      @addrType  addr∈Bits32Nat
      @addr      addr = b + (i * Bpw)
      @addrCheck WordAddrOkCheck(addr)=TRUE
      @spBits    spBits = NatToBits32(addr)
      @lr        lr = Bits32ToNat(GetMemBits32(memArray↦addr))
      @lrBits    lrBits = NatToBits32(lr)
      @nextPc    nextPc = lr+0
   end

end
```

14.1.9 machine XCoreIsaBranchingFinal

```
machine XCoreIsaBranchingFinal  // Branch instructions: instruction encoding

refines XCoreIsaBranching1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events

  event INITIALISATION extends INITIALISATION end

  event ECALLT_1r_OK extends ECALLT_OK
    any op1
    where
      @instCheck Is1r(instr↦9↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
      @instrSize instrSize=InstSizeShort+0
  end

  event ECALLT_1r_FALSE extends ECALLT_FALSE
    any op1
    where
      @instCheck Is1r(instr↦9↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
  end

  event ECALLF_1r_OK extends ECALLF_OK
    any op1
    where
      @instCheck Is1r(instr↦9↦0)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
      @instrSize instrSize=InstSizeShort+0
  end

  event ECALLF_1r_TRUE extends ECALLF_TRUE
    any op1
    where
```

```
    @instCheck Is1r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event KCALL_1r_OK extends KCALL_OK
  any op1
  where
    @instCheck   Is1r(instr↦8↦0)=TRUE
    @op1Fetch    op1=X1r_op1(instr)
    @callOperand callOperand=(threadRegArrays(threadIndex))(op1)
end

event KCALL_U6_OK extends KCALL_OK
  any op1
  where
    @instCheck   IsU6(instr↦14↦0↦15)=TRUE
    @op1Fetch    op1=XU6_op1(instr)
    @callOperand callOperand=NatToBits32(op1)
end

event KCALL_Lu6_OK extends KCALL_OK
  any op1
  where
    @instCheck   IsLu6(instr↦14↦0↦15)=TRUE
    @op1Fetch    op1=XLu6_op1(instr)
    @callOperand callOperand=NatToBits32(op1)
end

event KRET_0r_OK extends KRET_OK
  any nextEd nextSr
  where
    @instCheck Is0r(instr↦0↦1↦13)=TRUE
    @nextEd    nextEd=(threadRegArrays(threadIndex))(SedIndex)
    @nextSr    nextSr=(threadRegArrays(threadIndex))(SsrIndex)
end

event BAU_1r_OK extends BAU_OK
  any op1
  where
    @instCheck Is1r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
    @nextPc    nextPc=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end
```

```
event BLA_1r_OK extends BLA_OK
  any op1
  where
    @instCheck Is1r(instr↦4↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
    @nextPc    nextPc=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BLACP_U10_BadAddress extends BLACP_BadAddress
  where
    @instCheck IsU10(instr↦28↦0)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=XU10_op1(instr)
end

event BLACP_U10_OK extends BLACP_OK
  where
    @instCheck IsU10(instr↦28↦0)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=XU10_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event BLACP_Lu10_BadAddress extends BLACP_BadAddress
  where
    @instCheck IsLu10(instr↦28↦0)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=XLu10_op1(instr)
end

event BLACP_Lu10_OK extends BLACP_OK
  where
    @instCheck IsLu10(instr↦28↦0)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=XLu10_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event BLAT_U6_BadAddress extends BLAT_BadAddress
  where
    @instCheck IsU6(instr↦14↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(R11Index))
    @op1Fetch  i=XU6_op1(instr)
```

```
 end

event BLAT_U6_OK extends BLAT_OK
  where
    @instCheck IsU6(instr↦14↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(R11Index))
    @op1Fetch  i=XU6_op1(instr)
    @instrSize instrSize=InstSizeShort+0
 end

event BLAT_Lu6_BadAddress extends BLAT_BadAddress
  where
    @instCheck IsLu6(instr↦14↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(R11Index))
    @op1Fetch  i=XLu6_op1(instr)
end

event BLAT_Lu6_OK extends BLAT_OK
  where
    @instCheck IsLu6(instr↦14↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(R11Index))
    @op1Fetch  i=XLu6_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event BLRB_U10_OK extends BLRB_OK
  where
    @instCheck IsU10(instr↦26↦1)=TRUE
    @op1Fetch  i=XU10_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event BLRB_Lu10_OK extends BLRB_OK
  where
    @instCheck IsLu10(instr↦26↦1)=TRUE
    @op1Fetch  i=XLu10_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event BLRF_U10_OK extends BLRF_OK
  where
    @instCheck IsU10(instr↦26↦0)=TRUE
    @op1Fetch  i=XU10_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end
```

```
event BLRF_Lu10_OK extends BLRF_OK
  where
    @instCheck IsLu10(instr↦26↦0)=TRUE
    @op1Fetch  i=XLu10_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event BRBF_Ru6_TRUE_OK extends BRBF_TRUE_OK
  any op1
  where
    @instCheck IsRu6(instr↦15↦1)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @checkOp1  op1∈0‥11
    @op2Fetch  i=XRu6_op2(instr)
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBF_Ru6_FALSE_OK extends BRBF_FALSE_OK
  any op1
  where
    @instCheck IsRu6(instr↦15↦1)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBF_Lru6_TRUE_OK extends BRBF_TRUE_OK
  any op1
  where
    @instCheck IsLru6(instr↦15↦1)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @checkOp1  op1∈0‥11
    @op2Fetch  i=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBF_Lru6_FALSE_OK extends BRBF_FALSE_OK
  any op1
  where
    @instCheck IsLru6(instr↦15↦1)=TRUE
```

```
    @op1Fetch  op1=XLru6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBT_Ru6_TRUE_OK extends BRBT_TRUE_OK
  any op1
  where
    @instCheck IsRu6(instr↦14↦1)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  i=XRu6_op2(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBT_Ru6_FALSE_OK extends BRBT_FALSE_OK
  any op1
  where
    @instCheck IsRu6(instr↦14↦1)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBT_Lru6_TRUE_OK extends BRBT_TRUE_OK
  any op1
  where
    @instCheck IsLru6(instr↦14↦1)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @checkOp1  op1∈0‥11
    @op2Fetch  i=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBT_Lru6_FALSE_OK extends BRBT_FALSE_OK
  any op1
  where
    @instCheck IsLru6(instr↦14↦1)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @checkOp1  op1∈0‥11
```

```
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRBU_U6_OK extends BRBU_OK
  where
    @instCheck IsU6(instr↦14↦1↦12)=TRUE
    @op1Fetch  i=XU6_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event BRBU_Lu6_OK extends BRBU_OK
  where
    @instCheck IsLu6(instr↦14↦1↦12)=TRUE
    @op1Fetch  i=XLu6_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event BRFF_Ru6_TRUE_OK extends BRFF_TRUE_OK
  any op1
  where
    @instCheck IsRu6(instr↦15↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  i=XRu6_op2(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFF_Ru6_FALSE_OK extends BRFF_FALSE_OK
  any op1
  where
    @instCheck IsRu6(instr↦15↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFF_Lru6_TRUE_OK extends BRFF_TRUE_OK
  any op1
  where
    @instCheck IsLru6(instr↦15↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
```

```
    @op2Fetch  i=XLru6_op2(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFF_Lru6_FALSE_OK extends BRFF_FALSE_OK
  any op1
  where
    @instCheck IsLru6(instr↦15↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFT_Ru6_TRUE_OK extends BRFT_TRUE_OK
  any op1
  where
    @instCheck IsRu6(instr↦14↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  i=XRu6_op2(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFT_Ru6_FALSE_OK extends BRFT_FALSE_OK
  any op1
  where
    @instCheck IsRu6(instr↦14↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeShort+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFT_Lru6_TRUE_OK extends BRFT_TRUE_OK
  any op1
  where
    @instCheck IsLru6(instr↦14↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  i=XLru6_op2(instr)
    @checkOp1  op1∈0‥11
```

```
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFT_Lru6_FALSE_OK extends BRFT_FALSE_OK
  any op1
  where
    @instCheck IsLru6(instr↦14↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @checkOp1  op1∈0‥11
    @instrSize instrSize=InstSizeLong+0
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event BRFU_U6_OK extends BRFU_OK
  where
    @instCheck IsU6(instr↦14↦0↦12)=TRUE
    @op1Fetch  i=XU6_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event BRFU_Lu6_OK extends BRFU_OK
  where
    @instCheck IsLu6(instr↦14↦0↦12)=TRUE
    @op1Fetch  i=XLu6_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event BRU_1r_OK extends BRU_OK
  any op1
  where
    @instCheck Is1r(instr↦5↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op1))
end

event RETSP_Zero_U6_OK extends RETSP_Zero_OK
  where
    @instCheck IsU6(instr↦14↦1↦15)=TRUE
    @op1Fetch  i=XU6_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event RETSP_Zero_Lu6_OK extends RETSP_Zero_OK
```

```
    where
      @instCheck IsLu6(instr↦14↦1↦15)=TRUE
      @op1Fetch  i=XLu6_op1(instr)
      @instrSize instrSize=InstSizeLong+0
  end

  event RETSP_U6_BadAddress extends RETSP_BadAddress
    where
      @instCheck IsU6(instr↦14↦1↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XU6_op1(instr)
  end

  event RETSP_U6_OK extends RETSP_OK
    where
      @instCheck IsU6(instr↦14↦1↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XU6_op1(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event RETSP_Lu6_BadAddress extends RETSP_BadAddress
    where
      @instCheck IsLu6(instr↦14↦1↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XLu6_op1(instr)
  end

  event RETSP_Lu6_OK extends RETSP_OK
    where
      @instCheck IsLu6(instr↦14↦1↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XLu6_op1(instr)
      @instrSize instrSize=InstSizeLong+0
  end

end
```

14.1.10machine XCoreIsaChannelEnd1

```
machine XCoreIsaChannelEnd1 // Channel-end instructions: functionality
refines XCoreFinal
sees XCoreBasic XCoreChannelEnds

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

          channelEndIndex
          channelEndStatus channelEndEventMode
          channelEndThreadIdent channelEndEnvironmentVector channelEndVector
          channelEndDataIsLocal channelEndHasDst channelEndDstIdent channelNetworkIdent
          channelEndData channelEndDataIsCtrl channelEndState channelEndDepth

invariants
  @channelEndIndex       channelEndIndex ∈ ChannelEndIndexDom
  @channelEndStatus      channelEndStatus ∈  ChannelEndIndexDom → ChannelEndStatus
  @channelEndEventMode   channelEndEventMode ∈ ChannelEndIndexDom → BOOL
  @channelEndDataIsLocal channelEndDataIsLocal ∈ ChannelEndIndexDom → BOOL
  @channelEndDstIdent    channelEndDstIdent ∈ ChannelEndIndexDom → Bits
  @channelEndHasDst      channelEndHasDst ∈ ChannelEndIndexDom → BOOL
  @channelNetworkIdent   channelNetworkIdent ∈ ChannelEndIndexDom → Bits
  @channelEndData        channelEndData ∈  ChannelEndIndexDom → ChannelEndData
  @channelEndDataIsCtrl  channelEndDataIsCtrl ∈ ChannelEndIndexDom → ChannelEndBool
  @channelEndState       channelEndState ∈ ChannelEndIndexDom → ChannelEndState
  @channelEndDepth       channelEndDepth ∈ ChannelEndIndexDom → ChannelEndDepth
  @channelEndThreadIdent channelEndThreadIdent ∈  ChannelEndIndexDom → Bits
  @channelEndEnvVec      channelEndEnvironmentVector ∈  ChannelEndIndexDom → Bits32Nat
  @channelEndVec         channelEndVector ∈  ChannelEndIndexDom → Bits32Nat

events
  event INITIALISATION extends INITIALISATION
    then
      @channelEndIdx         channelEndIndex ≔ 0
      @chanEndStatus         channelEndStatus ≔
                                    {0↦FREE}  □ {1↦FREE}  □ {2↦FREE}  □ {3↦FREE} □
                                    {4↦FREE}  □ {5↦FREE}  □ {6↦FREE}  □ {7↦FREE} □
                                    {8↦FREE}  □ {9↦FREE}  □ {10↦FREE} □ {11↦FREE} □
                                    {12↦FREE} □ {13↦FREE} □ {14↦FREE} □ {15↦FREE} □
                                    {16↦FREE} □ {17↦FREE} □ {18↦FREE} □ {19↦FREE} □
                                    {20↦FREE} □ {21↦FREE} □ {22↦FREE} □ {23↦FREE} □
```

```
                                {24↦FREE} □ {25↦FREE} □ {26↦FREE} □ {27↦FREE} □
                                {28↦FREE} □ {29↦FREE} □ {30↦FREE} □ {31↦FREE}

@chanEndEvtMde          channelEndEventMode ≔
                                {0↦TRUE}  □ {1↦TRUE}  □ {2↦TRUE}  □ {3↦TRUE} □
                                {4↦TRUE}  □ {5↦TRUE}  □ {6↦TRUE}  □ {7↦TRUE} □
                                {8↦TRUE}  □ {9↦TRUE}  □ {10↦TRUE} □ {11↦TRUE} □
                                {12↦TRUE} □ {13↦TRUE} □ {14↦TRUE} □ {15↦TRUE} □
                                {16↦TRUE} □ {17↦TRUE} □ {18↦TRUE} □ {19↦TRUE} □
                                {20↦TRUE} □ {21↦TRUE} □ {22↦TRUE} □ {23↦TRUE} □
                                {24↦TRUE} □ {25↦TRUE} □ {26↦TRUE} □ {27↦TRUE} □
                                {28↦TRUE} □ {29↦TRUE} □ {30↦TRUE} □ {31↦TRUE}

@channelEndDataIsLocal channelEndDataIsLocal    :∈ ChannelEndIndexDom → BOOL

@channelEndHasDst       channelEndHasDst ≔
                                {0↦FALSE}  □ {1↦FALSE}  □ {2↦FALSE}  □ {3↦FALSE} □
                                {4↦FALSE}  □ {5↦FALSE}  □ {6↦FALSE}  □ {7↦FALSE} □
                                {8↦FALSE}  □ {9↦FALSE}  □ {10↦FALSE} □ {11↦FALSE} □
                                {12↦FALSE} □ {13↦FALSE} □ {14↦FALSE} □ {15↦FALSE} □
                                {16↦FALSE} □ {17↦FALSE} □ {18↦FALSE} □ {19↦FALSE} □
                                {20↦FALSE} □ {21↦FALSE} □ {22↦FALSE} □ {23↦FALSE} □
                                {24↦FALSE} □ {25↦FALSE} □ {26↦FALSE} □ {27↦FALSE} □
                                {28↦FALSE} □ {29↦FALSE} □ {30↦FALSE} □ {31↦FALSE}

@channelEndDstIdent     channelEndDstIdent    :∈ ChannelEndIndexDom → Bits
@channelNetworkIdent    channelNetworkIdent   :∈ ChannelEndIndexDom → Bits
@chanEndDstData         channelEndData        :∈  ChannelEndIndexDom → ChannelEndData
@channelEndDataIsCtrl   channelEndDataIsCtrl  :∈ ChannelEndIndexDom → ChannelEndBool
@channelEndState        channelEndState ≔
                                {0↦0}  □ {1↦0}  □ {2↦0}  □ {3↦0} □
                                {4↦0}  □ {5↦0}  □ {6↦0}  □ {7↦0} □
                                {8↦0}  □ {9↦0}  □ {10↦0} □ {11↦0} □
                                {12↦0} □ {13↦0} □ {14↦0} □ {15↦0} □
                                {16↦0} □ {17↦0} □ {18↦0} □ {19↦0} □
                                {20↦0} □ {21↦0} □ {22↦0} □ {23↦0} □
                                {24↦0} □ {25↦0} □ {26↦0} □ {27↦0} □
                                {28↦0} □ {29↦0} □ {30↦0} □ {31↦0}

@channelEndDepth        channelEndDepth ≔
                                {0↦0}  □ {1↦0}  □ {2↦0}  □ {3↦0} □
```

```
                                {4↦0} □ {5↦0} □ {6↦0} □ {7↦0} □
                                {8↦0} □ {9↦0} □ {10↦0} □ {11↦0} □
                                {12↦0} □ {13↦0} □ {14↦0} □ {15↦0} □
                                {16↦0} □ {17↦0} □ {18↦0} □ {19↦0} □
                                {20↦0} □ {21↦0} □ {22↦0} □ {23↦0} □
                                {24↦0} □ {25↦0} □ {26↦0} □ {27↦0} □
                                {28↦0} □ {29↦0} □ {30↦0} □ {31↦0}

    @channelEndThreadIdent channelEndThreadIdent       :∈  ChannelEndIndexDom → Bits
    @channelEndEnvVec      channelEndEnvironmentVector :∈  ChannelEndIndexDom → Bits32Nat
    @channelEndVec         channelEndVector            :∈  ChannelEndIndexDom → Bits32Nat

end
event Loading                          extends Loading                          where @false ⊥ end
event RunExternalInputs                extends RunExternalInputs                where @false ⊥ end
event SelectNextThread                 extends SelectNextThread                 where @false ⊥ end
event ThreadWaiting                    extends ThreadWaiting                    where @false ⊥ end
event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
event LinkErrorException               extends LinkErrorException               where @false ⊥ end
event IllegalPcException               extends IllegalPcException               where @false ⊥ end
event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
event LoadStoreException               extends LoadStoreException               where @false ⊥ end
event IllegalPsException               extends IllegalPsException               where @false ⊥ end
event ArithmeticException              extends ArithmeticException              where @false ⊥ end
event EcallException                   extends EcallException                   where @false ⊥ end
event ResourceDepException             extends ResourceDepException             where @false ⊥ end
event KcallException                   extends KcallException                   where @false ⊥ end
event Interrupt                        extends Interrupt                        where @false ⊥ end
event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
```

```
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event GETR_No_ChannelEnd extends Write1RegisterInstruction
  any resType
  where
    @resTypeChck resType=ResourceChannelEnd
    @checkFree   CountFreeChannelEnds(channelEndStatus)=0 // All used up
    @dBits       dBits=NatToBits32(NoResourceId)
end

event GETR_ChannelEnd_OK extends Write1RegisterInstruction
  any resType resIndex
  where
    @resTypeChck resType=ResourceChannelEnd
    @checkFree   CountFreeChannelEnds(channelEndStatus)≠0  // Channel-end available somewhere
    @resIndex    resIndex=FindFreeChannelEnd(channelEndStatus)
    @dBits       dBits=GetChannelEndId(resIndex)
  then
    @syncStatusChange channelEndStatus(resIndex) ≔ RUNNING
end

event FREER_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event FREER_ChannelEnd_DataPending  extends IllegalResourceException
  any resIndex
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @depthCheck  channelEndDepth(resIndex)>0
end
```

```
event FREER_ChannelEnd_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType    resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
  then
    @chanStatusChange channelEndStatus(resIndex) := FREE
    @channelEndHasDst channelEndHasDst(resIndex) := FALSE
end

event SETEV_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event SETEV_ChannelEnd_OK extends Write0RegisterInstruction
  any resIdent resIndex channelEndBits
  where
    @resIdentType       resIdent∈Bits
    @typeCheck          IsChannelEndIdent(resIdent)=TRUE
    @statusCheck        IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex           resIndex=GetResourceIndex(resIdent)
    @channelEndBitsType channelEndBits∈Bits
  then
    @changeEnvVec channelEndEnvironmentVector(resIndex) :=  Bits32ToNat(channelEndBits)
end

event SETV_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event SETV_ChannelEnd_OK extends Write0RegisterInstruction
  any resIdent resIndex channelEndBits
  where
    @resIdentType  resIdent∈Bits
    @typeCheck     IsChannelEndIdent(resIdent)=TRUE
    @statusCheck   IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @channelEndBitsType channelEndBits∈Bits
```

```
  then
    @changeVect channelEndVector(resIndex) ≔  Bits32ToNat(channelEndBits)
end

event SETN_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event SETN_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event SETN_OK extends Write0RegisterInstruction
  any resIdent resIndex netIdBits
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @netIdBits   netIdBits∈Bits
  then
    @changeReg channelNetworkIdent(resIndex) ≔ netIdBits
end

event GETN_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event GETN_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @dBits       dBits=channelNetworkIdent(resIndex)
end

event EDU_ChannelEnd_NotInUse extends IllegalResourceException
```

```
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event EDU_ChannelEnd_OK extends Write0RegisterInstruction
  any resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=NatToBits32(NoResourceId)
  then
    @channelEndThreadIdent channelEndThreadIdent(resIndex) ≔  threadId
end

event EEF_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event EEF_ChannelEnd_TRUE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c=0
  then
    @changeThread channelEndThreadIdent(resIndex) ≔  threadId
end

event EEF_ChannelEnd_FALSE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=NatToBits32(NoResourceId)
```

```
    @cond          c≠0
  then
    @changeThread channelEndThreadIdent(resIndex) ≔  threadId
end

event EET_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event EET_ChannelEnd_TRUE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c≠0
  then
     @changeThread channelEndThreadIdent(resIndex) ≔  threadId
end

event EET_ChannelEnd_FALSE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c=0
  then
     @changeThread channelEndThreadIdent(resIndex) ≔  threadId
end

event EEU_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event EEU_ChannelEnd_OK extends Write0RegisterInstruction
```

```
  any resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
  then
    @changeThread channelEndThreadIdent(resIndex) ≔  threadId
end

event SETC_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event SETC_ChannelEnd_BadMode extends IllegalResourceException
  any ctrl
  where
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @ctrlType     ctrl∈Bits32Nat
    @eventCheck   ctrl≠CtrlIeModeEvent
    @intCheck     ctrl≠CtrlIeModeInterrupt
end

event SETC_ChannelEnd_Event_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
    @ctrlCheck    ctrl=CtrlIeModeEvent
  then
    @chanEndControl channelEndEventMode(resIndex) ≔ TRUE
end

event SETC_ChannelEnd_Interrupt_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsChannelEndIdent(resIdent)=TRUE
```

```
    @statusCheck  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
    @ctrlCheck    ctrl=CtrlIeModeInterrupt
  then
    @chanEndControl channelEndEventMode(resIndex) := FALSE
end

event SETD_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event SETD_OK extends Write0RegisterInstruction
  any resIdent resIndex sBits
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @sBitsType   sBits∈Bits
  then
    @changeIdent channelEndDstIdent(resIndex) :=  sBits
    @hasDst      channelEndHasDst(resIndex) :=  TRUE
end

event GETD_ChannelEnd_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event GETD_ChannelEnd_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @dBits       dBits=channelEndDstIdent(resIndex)
end

event IN_ChannelEnd_NotInUse extends IllegalResourceException
```

```
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event IN_ChannelEnd_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @available   channelEndDepth(resIndex) < Bpw
end

event IN_ChannelEnd_Ct0 extends IllegalResourceException
  any resIndex dataPos0
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @available   channelEndDepth(resIndex) ≥ Bpw
    @data0Pos    dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0     (channelEndDataIsCtrl(resIndex))(dataPos0)=TRUE
end

event IN_ChannelEnd_Ct1 extends IllegalResourceException
  any resIndex dataPos1
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @available   channelEndDepth(resIndex) ≥ Bpw
    @data1Pos    dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0     (channelEndDataIsCtrl(resIndex))(dataPos1)=TRUE
end

event IN_ChannelEnd_Ct2 extends IllegalResourceException
  any resIndex dataPos2
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
```

```
    @available   channelEndDepth(resIndex) ≥ Bpw
    @data2Pos    dataPos2=ChannelEndReadPosition2(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0     (channelEndDataIsCtrl(resIndex))(dataPos2)=TRUE
end

event IN_ChannelEnd_Ct3 extends IllegalResourceException
  any resIndex dataPos3
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @available   channelEndDepth(resIndex) ≥ Bpw
    @data3Pos    dataPos3=ChannelEndReadPosition3(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0     (channelEndDataIsCtrl(resIndex))(dataPos3)=TRUE
end

event IN_ChannelEnd_OK extends Write1RegisterInstruction
  any
    resIdent resIndex
    dataPos0 dataPos1 dataPos2 dataPos3
    data0 data1 data2 data3
    nextState
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ Bpw
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data2Pos  dataPos2=ChannelEndReadPosition2(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data3Pos  dataPos3=ChannelEndReadPosition3(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=FALSE
    @isData2   (channelEndDataIsCtrl(resIndex))(dataPos2)=FALSE
    @isData3   (channelEndDataIsCtrl(resIndex))(dataPos3)=FALSE
    @data0     data0=(channelEndData(resIndex))(dataPos0)
    @data1     data1=(channelEndData(resIndex))(dataPos1)
    @data2     data2=(channelEndData(resIndex))(dataPos2)
    @data3     data3=(channelEndData(resIndex))(dataPos3)
    @dBits     dBits=FourBits8ToBits32(data0↦data1↦data2↦data3)
    @nextState nextState=ChannelEndReadStateChange(channelEndState(resIndex)↦Bpw)
  then
    @changeDepth channelEndDepth ≔ channelEndDepth  {resIndex↦(channelEndDepth(resIndex)−Bpw)}
```

```
    @changeState channelEndState ≔ channelEndState  {resIndex↦nextState}
end

event INCT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event INCT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event INCT_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) < 1
end

event INCT_DataToken extends IllegalResourceException
  any resIndex dataPos
  where
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @data0Pos  dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isCtrl    (channelEndDataIsCtrl(resIndex))(dataPos)=FALSE
end

event INCT_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos data nextState
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @data0Pos  dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
```

```
    @isCtrl     (channelEndDataIsCtrl(resIndex))(dataPos)=TRUE
    @data       data=(channelEndData(resIndex))(dataPos)
    @dBits      dBits=OneBits8ToBits32(data)
    @nextState  nextState=ChannelEndReadStateChange(channelEndState(resIndex)↦1)
  then
    @changeDepth channelEndDepth ≔ channelEndDepth  {resIndex↦(channelEndDepth(resIndex)-1)}
    @changeState channelEndState ≔ channelEndState  {resIndex↦nextState}
end

event INT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event INT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event INT_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) < 1
end

event INT_CtrlToken extends IllegalResourceException
  any resIndex dataPos
  where
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @data0Pos  dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData    (channelEndDataIsCtrl(resIndex))(dataPos)=TRUE
end

event INT_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos data nextState
  where
```

```
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @data0Pos  dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData    (channelEndDataIsCtrl(resIndex))(dataPos)=FALSE
    @data      data=(channelEndData(resIndex))(dataPos)
    @dBits     dBits=OneBits8ToBits32(data)
    @nextState nextState=ChannelEndReadStateChange(channelEndState(resIndex)↦1)
  then
    @changeDepth channelEndDepth ≔ channelEndDepth  {resIndex↦(channelEndDepth(resIndex)-1)}
    @changeState channelEndState ≔ channelEndState  {resIndex↦nextState}
end

event CHKCT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event CHKCT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event CHKCT_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @available   channelEndDepth(resIndex) < 1
end

event CHKCT_DataToken extends IllegalResourceException
  any resIndex dataPos
  where
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @data0Pos  dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
```

```
    @isCtrl     (channelEndDataIsCtrl(resIndex))(dataPos)=FALSE
end

event CHKCT_BadCtrlToken extends IllegalResourceException
  any resIndex dataPos data comp
  where
    @resType    IsChannelEndIdent(resIdent)=TRUE
    @resStatus  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex   resIndex=GetResourceIndex(resIdent)
    @available  channelEndDepth(resIndex) ≥ 1
    @data0Pos   dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isCtrl     (channelEndDataIsCtrl(resIndex))(dataPos)=TRUE
    @data       data=(channelEndData(resIndex))(dataPos)
    @compType   comp∈Bits
    @compare    OneBits8ToBits32(data)≠comp
end

event CHKCT_OK extends Write0RegisterInstruction
  any resIdent resIndex dataPos data comp nextState
  where
    @resIdType  resIdent∈Bits
    @resType    IsChannelEndIdent(resIdent)=TRUE
    @resStatus  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex   resIndex=GetResourceIndex(resIdent)
    @available  channelEndDepth(resIndex) ≥ 1
    @data0Pos   dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isCtrl     (channelEndDataIsCtrl(resIndex))(dataPos)=TRUE
    @data       data=(channelEndData(resIndex))(dataPos)
    @compType   comp∈Bits
    @compare    OneBits8ToBits32(data)=comp
    @nextState  nextState=ChannelEndReadStateChange(channelEndState(resIndex)↦1)
  then
    @changeDepth channelEndDepth ≔ channelEndDepth  {resIndex↦(channelEndDepth(resIndex)-1)}
    @changeState channelEndState ≔ channelEndState  {resIndex↦nextState}
end

event TESTLCL_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event TESTLCL_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
```

```
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event TESTLCL_TRUE_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @isLocal   channelEndDataIsLocal(resIndex)=TRUE
    @dBits     dBits=NatToBits32(1)
end

event TESTLCL_FALSE_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @isLocal   channelEndDataIsLocal(resIndex)=FALSE
    @dBits     dBits=NatToBits32(0)
end

event TESTCT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event TESTCT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event TESTCT_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) < 1
```

```
end

event TESTCT_TRUE_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @dataPos   dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isCtrl    (channelEndDataIsCtrl(resIndex))(dataPos)=TRUE
    @dBits     dBits=NatToBits32(1)
end

event TESTCT_FALSE_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @dataPos   dataPos=ChannelEndReadPosition(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isCtrl    (channelEndDataIsCtrl(resIndex))(dataPos)=FALSE
    @dBits     dBits=NatToBits32(0)
end

event TESTWCT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event TESTWCT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event TESTWCT_Pausing_0_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
```

```
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex)=0
end

event TESTWCT_Pausing_1_OK extends PausingInstruction
  any resIdent resIndex dataPos0
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex)=1
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
end

event TESTWCT_Pausing_2_OK extends PausingInstruction
  any resIdent resIndex dataPos0 dataPos1
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex)=2
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=FALSE
end

event TESTWCT_Pausing_3_OK extends PausingInstruction
  any resIdent resIndex dataPos0 dataPos1 dataPos2
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex)=3
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data2Pos  dataPos2=ChannelEndReadPosition2(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=FALSE
```

```
    @isData2    (channelEndDataIsCtrl(resIndex))(dataPos2)=FALSE
end

event TESTWCT_NoCtrl_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos0 dataPos1 dataPos2 dataPos3
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 4
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data2Pos  dataPos2=ChannelEndReadPosition2(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data3Pos  dataPos3=ChannelEndReadPosition3(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=FALSE
    @isData2   (channelEndDataIsCtrl(resIndex))(dataPos2)=FALSE
    @isData3   (channelEndDataIsCtrl(resIndex))(dataPos3)=FALSE
    @dBits     dBits=NatToBits32(0)
end

event TESTWCT_Ct0_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos0
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 1
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=TRUE
    @dBits     dBits=NatToBits32(1)
end

event TESTWCT_Ct1_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos0 dataPos1
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 3
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
```

```
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=TRUE
    @dBits     dBits=NatToBits32(2)
end

event TESTWCT_Ct2_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos0 dataPos1 dataPos2
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 3
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data2Pos  dataPos2=ChannelEndReadPosition2(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=FALSE
    @isData2   (channelEndDataIsCtrl(resIndex))(dataPos2)=TRUE
    @dBits     dBits=NatToBits32(3)
end

event TESTWCT_Ct3_OK extends Write1RegisterInstruction
  any resIdent resIndex dataPos0 dataPos1 dataPos2 dataPos3
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @available channelEndDepth(resIndex) ≥ 4
    @data0Pos  dataPos0=ChannelEndReadPosition0(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data1Pos  dataPos1=ChannelEndReadPosition1(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data2Pos  dataPos2=ChannelEndReadPosition2(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @data3Pos  dataPos3=ChannelEndReadPosition3(channelEndState(resIndex)↦channelEndDepth(resIndex))
    @isData0   (channelEndDataIsCtrl(resIndex))(dataPos0)=FALSE
    @isData1   (channelEndDataIsCtrl(resIndex))(dataPos1)=FALSE
    @isData2   (channelEndDataIsCtrl(resIndex))(dataPos2)=FALSE
    @isData3   (channelEndDataIsCtrl(resIndex))(dataPos3)=TRUE
    @dBits     dBits=NatToBits32(4)
end

event OUT_ChannelEnd_NotInUse extends IllegalResourceException
  where
```

```
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event OUT_ChannelEnd_NoDest extends LinkErrorException
  any resIndex
  where
    @resIdentType resIdent∈Bits
    @resType      IsChannelEndIdent(resIdent)=TRUE
    @resStatus    IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @hasDst       channelEndHasDst(resIndex)=FALSE
end

event OUT_ChannelEnd_Pausing_OK extends PausingInstruction
  any resIdent resIndex dstIdent dstIndex
  where
    @resIdType  resIdent∈Bits
    @resType    IsChannelEndIdent(resIdent)=TRUE
    @resStatus  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex   resIndex=GetResourceIndex(resIdent)
    @hasDst     channelEndHasDst(resIndex)=TRUE
    @dstIdent   dstIdent=channelEndDstIdent(resIndex)
    @dstType    IsChannelEndIdent(dstIdent)=TRUE
    @dstStatus  IsChannelEndInUse(dstIdent↦channelEndStatus)=TRUE
    @dstIndex   dstIndex=GetResourceIndex(dstIdent)
    @available  ChannelEndSpace(channelEndDepth(dstIndex)) < Bpw
end

event OUT_ChannelEnd_OK extends Write0RegisterInstruction
  any
    resIdent resIndex dstIdent dstIndex
    dataPos0 dataPos1 dataPos2 dataPos3
    data data0 data1 data2 data3
    nextState
  where
    @resIdType  resIdent∈Bits
    @resType    IsChannelEndIdent(resIdent)=TRUE
    @resStatus  IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex   resIndex=GetResourceIndex(resIdent)
    @hasDst     channelEndHasDst(resIndex)=TRUE
    @dstIdent   dstIdent=channelEndDstIdent(resIndex)
    @dstType    IsChannelEndIdent(dstIdent)=TRUE
    @dstStatus  IsChannelEndInUse(dstIdent↦channelEndStatus)=TRUE
```

```
    @dstIndex  dstIndex=GetResourceIndex(dstIdent)
    @available ChannelEndSpace(channelEndDepth(dstIndex)) ≥ Bpw
    @data0Pos  dataPos0=ChannelEndWritePosition0(channelEndState(dstIndex)↦channelEndDepth(dstIndex))
    @data1Pos  dataPos1=ChannelEndWritePosition1(channelEndState(dstIndex)↦channelEndDepth(dstIndex))
    @data2Pos  dataPos2=ChannelEndWritePosition2(channelEndState(dstIndex)↦channelEndDepth(dstIndex))
    @data3Pos  dataPos3=ChannelEndWritePosition3(channelEndState(dstIndex)↦channelEndDepth(dstIndex))
    @dataType  data∈Bits
    @data0     data0=ExtractBits8_0(data)
    @data1     data1=ExtractBits8_1(data)
    @data2     data2=ExtractBits8_2(data)
    @data3     data3=ExtractBits8_3(data)
    @nextState nextState=ChannelEndWriteStateChange(channelEndState(dstIndex)↦Bpw)
  then
    @changeIsCtl channelEndDataIsCtrl ≔ channelEndDataIsCtrl  {dstIndex↦{dataPos0↦FALSE}}  {dstIndex↦{dataPos1↦FALSE}} 
                                                              {dstIndex↦{dataPos2↦FALSE}}  {dstIndex↦{dataPos3↦FALSE}}
    @changeDepth channelEndDepth ≔ channelEndDepth  {dstIndex↦(channelEndDepth(dstIndex)+Bpw)}
    @changeState channelEndState ≔ channelEndState  {dstIndex↦nextState}
    @changeData  channelEndData ≔ channelEndData  {dstIndex↦{dataPos0↦data0}}  {dstIndex↦{dataPos1↦data1}} 
                                                  {dstIndex↦{dataPos2↦data2}}  {dstIndex↦{dataPos3↦data3}}
end

event OUTCT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event OUTCT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event OUTCT_NoDest extends LinkErrorException
  any resIndex
  where
    @resIdentType resIdent∈Bits
    @resType      IsChannelEndIdent(resIdent)=TRUE
    @resStatus    IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @hasDst       channelEndHasDst(resIndex)=FALSE
end

event OUTCT_BadToken extends LinkErrorException
  any token
```

```
  where
    @resIdentType resIdent∈Bits
    @identOk      IsChannelEndIdent(resIdent)=TRUE
    @tokenType    token∈Bits
    @tokenChk     IsPrivilegedCtrlToken(token)=TRUE
end

event OUTCT_Pausing_OK extends PausingInstruction
  any resIdent resIndex dstIdent dstIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @hasDst    channelEndHasDst(resIndex)=TRUE
    @dstIdent  dstIdent=channelEndDstIdent(resIndex)
    @dstType   IsChannelEndIdent(dstIdent)=TRUE
    @dstStatus IsChannelEndInUse(dstIdent↦channelEndStatus)=TRUE
    @dstIndex  dstIndex=GetResourceIndex(dstIdent)
    @available ChannelEndSpace(channelEndDepth(dstIndex)) < 1
end

event OUTCT_OK extends Write0RegisterInstruction
  any resIdent resIndex dstIdent dstIndex dataPos token nextState
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @hasDst    channelEndHasDst(resIndex)=TRUE
    @dstIdent  dstIdent=channelEndDstIdent(resIndex)
    @dstType   IsChannelEndIdent(dstIdent)=TRUE
    @dstStatus IsChannelEndInUse(dstIdent↦channelEndStatus)=TRUE
    @dstIndex  dstIndex=GetResourceIndex(dstIdent)
    @available ChannelEndSpace(channelEndDepth(dstIndex)) ≥ 1
    @dataPos   dataPos=ChannelEndWritePosition(channelEndState(dstIndex)↦channelEndDepth(dstIndex))
    @tokenType token∈Bits
    @tokenChk  IsPrivilegedCtrlToken(token)=FALSE
    @nextState nextState=ChannelEndWriteStateChange(channelEndState(dstIndex)↦1)
  then
    @changeIsCtl channelEndDataIsCtrl ≔ channelEndDataIsCtrl  {dstIndex↦{dataPos↦TRUE}}
    @changeDepth channelEndDepth ≔ channelEndDepth  {dstIndex↦(channelEndDepth(dstIndex)+1)}
    @changeState channelEndState ≔ channelEndState  {dstIndex↦nextState}
    @changeData  channelEndData ≔ channelEndData  {dstIndex↦{dataPos↦token}}
```

```
end

event OUTT_BadIdent extends IllegalResourceException
  where
    @identOk IsChannelEndIdent(resIdent)=FALSE
end

event OUTT_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsChannelEndIdent(resIdent)=TRUE
    @statusCheck IsChannelEndInUse(resIdent↦channelEndStatus)=FALSE
end

event OUTT_NoDest extends LinkErrorException
  any resIndex
  where
    @resIdentType resIdent∈Bits
    @resType      IsChannelEndIdent(resIdent)=TRUE
    @resStatus    IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @hasDst       channelEndHasDst(resIndex)=FALSE
end

event OUTT_Pausing_OK extends PausingInstruction
  any resIdent resIndex dstIdent dstIndex
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @hasDst    channelEndHasDst(resIndex)=TRUE
    @dstIdent  dstIdent=channelEndDstIdent(resIndex)
    @dstType   IsChannelEndIdent(dstIdent)=TRUE
    @dstStatus IsChannelEndInUse(dstIdent↦channelEndStatus)=TRUE
    @dstIndex  dstIndex=GetResourceIndex(dstIdent)
    @available ChannelEndSpace(channelEndDepth(dstIndex)) < 1
end

event OUTT_OK extends Write0RegisterInstruction
  any resIdent resIndex dstIdent dstIndex dataPos token nextState
  where
    @resIdType resIdent∈Bits
    @resType   IsChannelEndIdent(resIdent)=TRUE
    @resStatus IsChannelEndInUse(resIdent↦channelEndStatus)=TRUE
```

```
      @resIndex  resIndex=GetResourceIndex(resIdent)
      @hasDst    channelEndHasDst(resIndex)=TRUE
      @dstIdent  dstIdent=channelEndDstIdent(resIndex)
      @dstType   IsChannelEndIdent(dstIdent)=TRUE
      @dstStatus IsChannelEndInUse(dstIdent↦channelEndStatus)=TRUE
      @dstIndex  dstIndex=GetResourceIndex(dstIdent)
      @available ChannelEndSpace(channelEndDepth(dstIndex)) ≥ 1
      @dataPos   dataPos=ChannelEndWritePosition(channelEndState(dstIndex)↦channelEndDepth(dstIndex))
      @tokenType token∈Bits
      @nextState nextState=ChannelEndWriteStateChange(channelEndState(dstIndex)↦1)
    then
      @changeIsCtl channelEndDataIsCtrl ≔ channelEndDataIsCtrl  {dstIndex↦{dataPos↦FALSE}}
      @changeDepth channelEndDepth ≔ channelEndDepth  {dstIndex↦(channelEndDepth(dstIndex)+1)}
      @changeState channelEndState ≔ channelEndState  {dstIndex↦nextState}
      @changeData  channelEndData ≔ channelEndData  {dstIndex↦{dataPos↦token}}
  end

  event SelectNextChannelEnd
    then
      @threadIndex channelEndIndex ≔  (channelEndIndex + 1) mod MaxChannelEnds
  end

  event ChannelEndEvent extends ResourceEvent
    where
      @chanStatus  channelEndStatus(channelEndIndex)=RUNNING
      @chanEvent   channelEndEventMode(channelEndIndex)=TRUE
      @valueCheck  channelEndDepth(channelEndIndex)>0
      @threadIdent channelEndThreadIdent(channelEndIndex)≠NatToBits32(NoResourceId)
      @threadIndex triggeredThreadIndex=GetResourceIndex(channelEndThreadIdent(channelEndIndex))
      @edBits      edBits=NatToBits32(channelEndEnvironmentVector(channelEndIndex))
      @nextPc      nextPc=channelEndVector(channelEndIndex)
  end

  event ChannelEndInterrupt extends Interrupt
    where
      @chanStatus  channelEndStatus(channelEndIndex)=RUNNING
      @chanEvent   channelEndEventMode(channelEndIndex)=FALSE
      @valueCheck  channelEndDepth(channelEndIndex)>0
      @threadIdent channelEndThreadIdent(channelEndIndex)≠NatToBits32(NoResourceId)
      @threadIndex triggeredThreadIndex=GetResourceIndex(channelEndThreadIdent(channelEndIndex))
      @edBits      edBits=NatToBits32(channelEndEnvironmentVector(channelEndIndex))
      @nextPc      nextPc=channelEndVector(channelEndIndex)
  end
end
```

14.1.11machine XCoreIsaChannelEndFinal

```
machine XCoreIsaChannelEndFinal // Channel-end instructions: instruction decoding

refines XCoreIsaChannelEnd1

sees XCoreBasic XCoreChannelEnds

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

          channelEndIndex
          channelEndStatus channelEndEventMode
          channelEndThreadIdent channelEndEnvironmentVector channelEndVector
          channelEndDataIsLocal channelEndHasDst channelEndDstIdent channelNetworkIdent
          channelEndData channelEndDataIsCtrl channelEndState channelEndDepth

events

  event INITIALISATION extends INITIALISATION end

  event GETR_Rus_No_ChannelEnd extends GETR_No_ChannelEnd
    where
      @instCheck IsRus(instr↦16↦0)=TRUE
      @op1Fetch  d=XRus_op1(instr)
      @op2Fetch  resType=XRus_op2(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event GETR_Rus_ChannelEnd_OK extends GETR_ChannelEnd_OK
    where
      @instCheck IsRus(instr↦16↦0)=TRUE
      @op1Fetch  d=XRus_op1(instr)
      @op2Fetch  resType=XRus_op2(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event FREER_1r_ChannelEnd_NotInUse extends FREER_ChannelEnd_NotInUse
    any op1
    where
      @instCheck  Is1r(instr↦2↦0)=TRUE
      @op1Fetch   op1=X1r_op1(instr)
      @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
```

```
end

event FREER_1r_ChannelEnd_DataPending extends FREER_ChannelEnd_DataPending
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
end

event FREER_1r_ChannelEnd_OK extends FREER_ChannelEnd_OK
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize  instrSize=InstSizeShort+0
end

event SETEV_1r_ChannelEnd_NotInUse extends SETEV_ChannelEnd_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦7↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETEV_1r_ChannelEnd_OK extends SETEV_ChannelEnd_OK
  any op1
  where
    @instCheck Is1r(instr↦7↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @chanBits  channelEndBits=(threadRegArrays(threadIndex))(R11Index)
    @instrSize instrSize=InstSizeShort+0
end

event SETV_1r_ChannelEnd_NotInUse extends SETV_ChannelEnd_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦8↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end
```

```
event SETV_1r_ChannelEnd_OK extends SETV_ChannelEnd_OK
  any op1
  where
    @instCheck Is1r(instr↦8↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @chanBits  channelEndBits=(threadRegArrays(threadIndex))(R11Index)
    @instrSize instrSize=InstSizeShort+0
end

event SETN_Lr2r_BadIdent extends SETN_BadIdent
  any op1
  where
    @instCheck IsLr2r(instr↦6↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETN_Lr2r_NotInUse extends SETN_NotInUse
  any op1
  where
    @instCheck IsLr2r(instr↦6↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETN_Lr2r_OK extends SETN_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦6↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @dBits     netIdBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event GETN_L2r_NotInUse extends GETN_NotInUse
  any op2
  where
    @instCheck IsL2r(instr↦6↦1↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end
```

```
event GETN_L2r_OK extends GETN_OK
  any op2
  where
    @instCheck IsL2r(instr↦6↦1↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Ru6_ChannelEnd_NotInUse extends SETC_ChannelEnd_NotInUse
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Ru6_ChannelEnd_BadMode extends SETC_ChannelEnd_BadMode
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Ru6_ChannelEnd_Event_OK extends SETC_ChannelEnd_Event_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_ChannelEnd_Interrupt_OK extends SETC_ChannelEnd_Interrupt_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
```

```
end

event SETC_Lru6_ChannelEnd_NotInUse extends SETC_ChannelEnd_NotInUse
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Lru6_ChannelEnd_BadMode extends SETC_ChannelEnd_BadMode
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Lru6_ChannelEnd_Event_OK extends SETC_ChannelEnd_Event_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_ChannelEnd_Interrupt_OK extends SETC_ChannelEnd_Interrupt_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_ChannelEnd_NotInUse extends SETC_ChannelEnd_NotInUse
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_L2r_ChannelEnd_BadMode extends SETC_ChannelEnd_BadMode
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
end

event SETC_L2r_ChannelEnd_Event_OK extends SETC_ChannelEnd_Event_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_ChannelEnd_Interrupt_OK extends SETC_ChannelEnd_Interrupt_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETD_R2r_ChannelEnd_NotInUse extends SETD_ChannelEnd_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETD_R2r_ChannelEnd_OK extends SETD_OK
  any op1 op2
```

```
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @sBits     sBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event GETD_L2r_ChannelEnd_NotInUse extends GETD_ChannelEnd_NotInUse
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event GETD_L2r_ChannelEnd_OK extends GETD_ChannelEnd_OK
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event EDU_1r_ChannelEnd_NotInUse extends EDU_ChannelEnd_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦0↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EDU_1r_ChannelEnd_OK extends EDU_ChannelEnd_OK
  any op1
  where
    @instCheck Is1r(instr↦0↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event EEF_2r_ChannelEnd_NotInUse extends EEF_ChannelEnd_NotInUse
```

```
  any op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event EEF_2r_ChannelEnd_TRUE_OK extends EEF_ChannelEnd_TRUE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EEF_2r_ChannelEnd_FALSE_OK extends EEF_ChannelEnd_FALSE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EET_2r_ChannelEnd_NotInUse extends EET_ChannelEnd_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event EET_2r_ChannelEnd_TRUE_OK extends EET_ChannelEnd_TRUE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
```

```
    @instrSize instrSize=InstSizeShort+0
end

event EET_2r_ChannelEnd_FALSE_OK extends EET_ChannelEnd_FALSE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EEU_1r_ChannelEnd_NotInUse extends EEU_ChannelEnd_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦0↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EEU_1r_ChannelEnd_OK extends EEU_ChannelEnd_OK
  any op1
  where
    @instCheck Is1r(instr↦0↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event IN_2r_ChannelEnd_NotInUse extends IN_ChannelEnd_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_ChannelEnd_Ct0 extends IN_ChannelEnd_Ct0
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
```

```
end

event IN_2r_ChannelEnd_Ct1 extends IN_ChannelEnd_Ct1
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_ChannelEnd_Ct2 extends IN_ChannelEnd_Ct2
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_ChannelEnd_Ct3 extends IN_ChannelEnd_Ct3
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_ChannelEnd_Pausing_OK extends IN_ChannelEnd_Pausing_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_ChannelEnd_OK extends IN_ChannelEnd_OK
  any
    op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end
```

```
event INCT_2r_BadIdent extends INCT_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦16↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INCT_2r_NotInUse extends INCT_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦16↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INCT_2r_Pausing_OK extends INCT_Pausing_OK
  any op2
  where
    @instCheck Is2r(instr↦16↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INCT_2r_DataToken extends INCT_DataToken
  any op2
  where
    @instCheck Is2r(instr↦16↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INCT_2r_OK extends INCT_OK
  any op2
  where
    @instCheck Is2r(instr↦16↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event INT_2r_BadIdent extends INT_BadIdent
  any op2
```

```
  where
    @instCheck Is2r(instr↦17↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INT_2r_NotInUse extends INT_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦17↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INT_2r_Pausing_OK extends INT_Pausing_OK
  any op2
  where
    @instCheck Is2r(instr↦17↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INT_2r_CtrlToken extends INT_CtrlToken
  any op2
  where
    @instCheck Is2r(instr↦17↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INT_2r_OK extends INT_OK
  any op2
  where
    @instCheck Is2r(instr↦17↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event CHKCT_2r_BadIdent extends CHKCT_BadIdent
  any op1
  where
    @instCheck Is2r(instr↦25↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_2r_NotInUse extends CHKCT_NotInUse
  any op1
  where
    @instCheck Is2r(instr↦25↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_2r_Pausing_OK extends CHKCT_Pausing_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦25↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_2r_OK extends CHKCT_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦25↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @comp      comp=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event CHKCT_2r_DataToken extends CHKCT_DataToken
  any op1
  where
    @instCheck Is2r(instr↦25↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_2r_BadCtrlToken extends CHKCT_BadCtrlToken
  any op1 op2
  where
    @instCheck Is2r(instr↦25↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @comp      comp=(threadRegArrays(threadIndex))(op2)
end

event CHKCT_Rus_BadIdent extends CHKCT_BadIdent
  any op1
  where
    @instCheck IsRus(instr↦25↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_Rus_NotInUse extends CHKCT_NotInUse
  any op1
  where
    @instCheck IsRus(instr↦25↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_Rus_Pausing_OK extends CHKCT_Pausing_OK
  any op1 op2
  where
    @instCheck IsRus(instr↦25↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @op2Fetch  op2=XRus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_Rus_OK extends CHKCT_OK
  any op1 op2
  where
    @instCheck IsRus(instr↦25↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @op2Fetch  op2=XRus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @comp      comp=NatToBits32(op2)
    @instrSize instrSize=InstSizeShort+0
end

event CHKCT_Rus_DataToken extends CHKCT_DataToken
  any op1
  where
    @instCheck IsRus(instr↦25↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CHKCT_Rus_BadCtrlToken extends CHKCT_BadCtrlToken
  any op1 op2
  where
    @instCheck IsRus(instr↦25↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @op2Fetch  op2=XRus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @comp      comp=NatToBits32(op2)
end

event TESTLCL_L2r_BadIdent extends TESTLCL_BadIdent
  any op2
  where
    @instCheck IsL2r(instr↦4↦0↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
end

event TESTLCL_L2r_NotInUse extends TESTLCL_NotInUse
  any op2
  where
    @instCheck IsL2r(instr↦4↦0↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
end

event TESTLCL_L2r_TRUE_OK extends TESTLCL_TRUE_OK
  any op2
  where
    @instCheck IsL2r(instr↦4↦0↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event TESTLCL_L2r_FALSE_OK extends TESTLCL_FALSE_OK
  any op2
  where
    @instCheck IsL2r(instr↦4↦0↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
```

```
end

event TESTCT_2r_BadIdent extends TESTCT_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦23↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TESTCT_2r_NotInUse extends TESTCT_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦23↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TESTCT_2r_Pausing_OK extends TESTCT_Pausing_OK
  any op2
  where
    @instCheck Is2r(instr↦23↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TESTCT_2r_TRUE_OK extends TESTCT_TRUE_OK
  any op2
  where
    @instCheck Is2r(instr↦23↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event TESTCT_2r_FALSE_OK extends TESTCT_FALSE_OK
  any op2
  where
    @instCheck Is2r(instr↦23↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end
```

```
event TESTWCT_2r_BadIdent extends TESTWCT_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TESTWCT_2r_NotInUse extends TESTWCT_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TESTWCT_2r_Pausing_0_OK extends TESTWCT_Pausing_0_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event TESTWCT_2r_Pausing_1_OK extends TESTWCT_Pausing_1_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event TESTWCT_2r_Pausing_2_OK extends TESTWCT_Pausing_2_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event TESTWCT_2r_Pausing_3_OK extends TESTWCT_Pausing_3_OK
```

```
  any op1 op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event TESTWCT_2r_NoCtrl_OK extends TESTWCT_NoCtrl_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event TESTWCT_2r_Ct0_OK extends TESTWCT_Ct0_OK
  any op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event TESTWCT_2r_Ct1_OK extends TESTWCT_Ct1_OK
  any op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event TESTWCT_2r_Ct2_OK extends TESTWCT_Ct2_OK
  any op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
```

```
    @instrSize instrSize=InstSizeShort+0
end

event TESTWCT_2r_Ct3_OK extends TESTWCT_Ct3_OK
  any op2
  where
    @instCheck Is2r(instr↦24↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event OUT_R2r_ChannelEnd_NotInUse extends OUT_ChannelEnd_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUT_R2r_ChannelEnd_NoDest extends OUT_ChannelEnd_NoDest
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUT_R2r_ChannelEnd_Pausing_OK extends OUT_ChannelEnd_Pausing_OK
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUT_R2r_ChannelEnd_OK extends OUT_ChannelEnd_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @data      data=(threadRegArrays(threadIndex))(op2)
```

```
    @instrSize instrSize=InstSizeShort+0
end

event OUTCT_2r_BadIdent extends OUTCT_BadIdent
  any op1
  where
    @instCheck Is2r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_2r_NotInUse extends OUTCT_NotInUse
  any op1
  where
    @instCheck Is2r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_2r_NoDest extends OUTCT_NoDest
  any op1
  where
    @instCheck Is2r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_2r_BadToken extends OUTCT_BadToken
  any op1 op2
  where
    @instCheck Is2r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @token     token=ExtractBits8_0((threadRegArrays(threadIndex))(op2))
end

event OUTCT_2r_Pausing_OK extends OUTCT_Pausing_OK
  any op1
  where
    @instCheck Is2r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end
```

```
event OUTCT_2r_OK extends OUTCT_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦9↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @token     token=ExtractBits8_0((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeShort+0
end

event OUTCT_Rus_BadIdent extends OUTCT_BadIdent
  any op1
  where
    @instCheck IsRus(instr↦9↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_Rus_NotInUse extends OUTCT_NotInUse
  any op1
  where
    @instCheck IsRus(instr↦9↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_Rus_NoDest extends OUTCT_NoDest
  any op1
  where
    @instCheck IsRus(instr↦9↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_Rus_BadToken extends OUTCT_BadToken
  any op1 op2
  where
    @instCheck IsRus(instr↦9↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @op2Fetch  op2=XRus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @token     token=ExtractBits8_0(NatToBits32(op2))
end
```

```
event OUTCT_Rus_Pausing_OK extends OUTCT_Pausing_OK
  any op1
  where
    @instCheck IsRus(instr↦9↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTCT_Rus_OK extends OUTCT_OK
  any op1 op2
  where
    @instCheck IsRus(instr↦9↦1)=TRUE
    @op1Fetch  op1=XRus_op1(instr)
    @op2Fetch  op2=XRus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @token     token=ExtractBits8_0(NatToBits32(op2))
    @instrSize instrSize=InstSizeShort+0
end

event OUTT_R2r_BadIdent extends OUTT_BadIdent
  any op1
  where
    @instCheck IsR2r(instr↦1↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTT_R2r_NotInUse extends OUTT_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦1↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTT_R2r_NoDest extends OUTT_NoDest
  any op1
  where
    @instCheck IsR2r(instr↦1↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTT_R2r_Pausing_OK extends OUTT_Pausing_OK
  any op1
```

```
    where
      @instCheck IsR2r(instr↦1↦1)=TRUE
      @op1Fetch  op1=XR2r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event OUTT_R2r_OK extends OUTT_OK
    any op1 op2
    where
      @instCheck IsR2r(instr↦1↦1)=TRUE
      @op1Fetch  op1=XR2r_op1(instr)
      @op2Fetch  op2=XR2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
      @token     token=ExtractBits8_0((threadRegArrays(threadIndex))(op2))
      @instrSize instrSize=InstSizeShort+0
  end

  event SelectNextChannelEnd extends SelectNextChannelEnd end
  event ChannelEndEvent      extends ChannelEndEvent end
  event ChannelEndInterrupt  extends ChannelEndInterrupt end

end
```

14.1.12machine XCoreIsaClock1

```
machine XCoreIsaClock1

refines XCoreFinal

sees XCoreBasic XCoreClocks XCorePorts

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events

  event INITIALISATION                   extends INITIALISATION end
  event Loading                          extends Loading                          where @false ⊥ end
  event RunExternalInputs                extends RunExternalInputs                where @false ⊥ end
  event SelectNextThread                 extends SelectNextThread                 where @false ⊥ end
  event ThreadWaiting                    extends ThreadWaiting                    where @false ⊥ end
  event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
  event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
  event LinkErrorException               extends LinkErrorException               where @false ⊥ end
  event IllegalPcException               extends IllegalPcException               where @false ⊥ end
  event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
  event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
  event LoadStoreException               extends LoadStoreException               where @false ⊥ end
  event IllegalPsException               extends IllegalPsException               where @false ⊥ end
  event ArithmeticException              extends ArithmeticException              where @false ⊥ end
  event EcallException                   extends EcallException                   where @false ⊥ end
  event ResourceDepException             extends ResourceDepException             where @false ⊥ end
  event KcallException                   extends KcallException                   where @false ⊥ end
  event Interrupt                        extends Interrupt                        where @false ⊥ end
  event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
  event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
  event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
  event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
  event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
  event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
  event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
  event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
  event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
  event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
  event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
```

```
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event SETCLK_Clock_Error extends IllegalResourceException
  any resIndex
  where
    @identOk  IsClockIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
 end

event SETCLK_Clock_OK extends Write0RegisterInstruction
  any resIdent resIndex srcBits
  where
    @resIdType   resIdent∈Bits
    @identOk     IsClockIdent(resIdent)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @srcBitsType srcBits∈Bits
 end

event SETRDY_Clock_Error extends IllegalResourceException
  any resIndex
  where
    @identOk  IsClockIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
 end

event SETRDY_Clock_OK extends Write0RegisterInstruction
  any resIdent resIndex srcIdent
  where
    @resIdType   resIdent∈Bits
    @identOk     IsClockIdent(resIdent)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @srcIdType   srcIdent∈Bits
 end

event SETC_Clock_Error extends IllegalResourceException
  any resIndex ctrl
```

```
    where
      @resIdType  resIdent ∈Bits
      @isClock    IsClockIdent(resIdent)=TRUE
      @resIndex   resIndex=GetResourceIndex(resIdent)
      @ctrlType   ctrl∈Bits32Nat
  end

  event SETC_Clock_OK extends Write0RegisterInstruction
    any resIdent resIndex ctrl
    where
      @resIdType  resIdent ∈Bits
      @isClock    IsClockIdent(resIdent)=TRUE
      @resIndex   resIndex=GetResourceIndex(resIdent)
      @ctrlType   ctrl∈Bits32Nat
  end

end
```

14.1.13machine XCoreIsaClock2

```
machine XCoreIsaClock2

refines XCoreIsaClock1

sees XCoreBasic XCoreClocks XCorePorts

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray
          clockEnabled clockRunning clockReadyPort

invariants
  @clockReadyPort clockReadyPort   ∈  ClockIndexDom → Bits
  @clockEnabled   clockEnabled     ∈  ClockIndexDom → BOOL
  @clockRunning   clockRunning     ∈  ClockIndexDom → BOOL

events

  event INITIALISATION extends INITIALISATION
    then
      @clockEnabled   clockEnabled ≔ {0↦FALSE} ⩤ {1↦FALSE} ⩤ {2↦FALSE} ⩤ {3↦FALSE}
      @clockRunning   clockRunning ≔ {0↦FALSE} ⩤ {1↦FALSE} ⩤ {2↦FALSE} ⩤ {3↦FALSE}
      @clockReadyPort clockReadyPort :∈  ClockIndexDom → Bits
  end

  event SETCLK_Clock_RunningClock extends SETCLK_Clock_Error
    where
      @isRunning clockRunning(resIndex)=TRUE
  end

  event SETCLK_Clock_BadIdent extends SETCLK_Clock_Error
    any srcBits
    where
      @isRunning   clockRunning(resIndex)=FALSE
      @srcBitsType srcBits∈Bits
      @clkOk       IsPortIdent(srcBits)=FALSE
      @refOk       IsRefClockIdent(srcBits)=FALSE
  end

  event SETCLK_Clock_Port_OK extends SETCLK_Clock_OK
    where
      @clkOk     IsPortIdent(srcBits)=TRUE
      @isRunning clockRunning(resIndex)=FALSE
```

```
end

event SETCLK_Clock_RefClock_OK extends SETCLK_Clock_OK
  where
    @clkOk     IsRefClockIdent(srcBits)=TRUE
    @isRunning clockRunning(resIndex)=FALSE
end

event SETRDY_Clock_NotInUse extends SETRDY_Clock_Error
  where
    @clockEnabled clockEnabled(resIndex)=FALSE
end

event SETRDY_Clock_OK extends SETRDY_Clock_OK
  where
    @clockEnabled clockEnabled(resIndex)=TRUE
    @isRunning    clockRunning(resIndex)=FALSE
  then
    @clockReadyPort  clockReadyPort(resIndex) ≔ srcIdent
end

event SETC_Clock_BadMode extends SETC_Clock_Error
  where
    @ctrlInUseOn    ctrl≠CtrlInUseOn
    @ctrlInUseOff   ctrl≠CtrlInUseOff
    @ctrlRunStartR ctrl≠CtrlRunStartR
    @ctrlRunStopR   ctrl≠CtrlRunStopR
end

event SETC_Clock_On_OK extends SETC_Clock_OK
  where
    @ctrlChk CompareEqualUnsigned(ctrl↦CtrlInUseOn)=TRUE
  then
    @clockEnabled clockEnabled(resIndex) ≔ TRUE
end

event SETC_Clock_Off_OK extends SETC_Clock_OK
  where
    @ctrlChk CompareEqualUnsigned(ctrl↦CtrlInUseOff)=TRUE
  then
    @clockEnabled clockEnabled(resIndex) ≔ FALSE
    @clockRunning clockRunning(resIndex) ≔ FALSE
end

event SETC_Clock_StartWhenOff extends SETC_Clock_Error
```

```
    where
      @ctrlChk CompareEqualUnsigned(ctrl↦CtrlRunStartR)=TRUE
      @clockEnabled clockEnabled(resIndex) = FALSE
  end

  event SETC_Clock_Start_OK extends SETC_Clock_OK
    where
      @ctrlChk CompareEqualUnsigned(ctrl↦CtrlRunStartR)=TRUE
      @clockEnabled clockEnabled(resIndex) = TRUE
    then
      @clockControl clockRunning(resIndex) ≔ TRUE
  end

  event SETC_Clock_Stop_OK extends SETC_Clock_OK
    where
      @ctrlChk CompareEqualUnsigned(ctrl↦CtrlRunStopR)=TRUE
    then
      @clockControl clockRunning(resIndex) ≔ FALSE
  end

end
```

14.1.14machine XCoreIsaClockFinal

```
machine XCoreIsaClockFinal // Add instruction decode methods

refines XCoreIsaClock2

sees XCoreBasic XCoreClocks XCorePorts

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

          clockEnabled clockRunning clockReadyPort

events

  event INITIALISATION  extends INITIALISATION end

  event SETCLK_Lr2r_Clock_RunningClock extends SETCLK_Clock_RunningClock
    any op1
    where
      @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
      @op1Fetch  op1=XLr2r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
   end

  event SETCLK_Lr2r_Clock_BadIdent extends SETCLK_Clock_BadIdent
    any op1 op2
    where
      @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
      @op1Fetch  op1=XLr2r_op1(instr)
      @op2Fetch  op2=XLr2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
      @srcBits   srcBits=(threadRegArrays(threadIndex))(op2)
   end

  event SETCLK_Lr2r_Clock_Port_OK extends SETCLK_Clock_Port_OK
    any op1 op2
    where
      @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
      @op1Fetch  op1=XLr2r_op1(instr)
      @op2Fetch  op2=XLr2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
      @srcBits   srcBits=(threadRegArrays(threadIndex))(op2)
      @instrSize instrSize=InstSizeLong+0
```

```
 end

event SETCLK_Lr2r_Clock_RefClock_OK extends SETCLK_Clock_RefClock_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @srcBits   srcBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
 end

event SETRDY_Lr2r_Clock_NotInUse extends SETRDY_Clock_NotInUse
  any op1
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETRDY_Lr2r_Clock_OK extends SETRDY_Clock_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @srcIdent  srcIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Ru6_Clock_BadMode extends SETC_Clock_BadMode
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XRu6_op2(instr)
end

event SETC_Ru6_Clock_On_OK extends SETC_Clock_On_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
```

```
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Clock_Off_OK extends SETC_Clock_Off_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Lru6_Clock_BadMode extends SETC_Clock_BadMode
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XLru6_op2(instr)
end

event SETC_Lru6_Clock_On_OK extends SETC_Clock_On_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Clock_Off_OK extends SETC_Clock_Off_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
end
```

```
event SETC_L2r_Clock_BadMode extends SETC_Clock_BadMode
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  op2=XL2r_op2(instr)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
end

event SETC_L2r_Clock_On_OK extends SETC_Clock_On_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  op2=XL2r_op2(instr)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Clock_Off_OK extends SETC_Clock_Off_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  op2=XL2r_op2(instr)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Ru6_Clock_StartWhenOff extends SETC_Clock_StartWhenOff
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XRu6_op2(instr)
end

event SETC_Ru6_Clock_Start_OK extends SETC_Clock_Start_OK
  any op1
   where
    @instCheck IsRu6(instr↦29↦0)=TRUE
```

```
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Clock_Stop_OK extends SETC_Clock_Stop_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Lru6_Clock_StartWhenOff extends SETC_Clock_StartWhenOff
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
end

event SETC_Lru6_Clock_Start_OK extends SETC_Clock_Start_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Clock_Stop_OK extends SETC_Clock_Stop_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @control   ctrl=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Clock_StartWhenOff extends SETC_Clock_StartWhenOff
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
```

```
      @op1Fetch  op1=XL2r_op1(instr)
      @op2Fetch  op2=XL2r_op2(instr)
      @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
  end

  event SETC_L2r_Clock_Start_OK extends SETC_Clock_Start_OK
    any op1 op2
    where
      @instCheck IsL2r(instr↦5↦1↦12)=TRUE
      @op1Fetch  op1=XL2r_op1(instr)
      @op2Fetch  op2=XL2r_op2(instr)
      @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
      @instrSize instrSize=InstSizeLong+0
  end

  event SETC_L2r_Clock_Stop_OK extends SETC_Clock_Stop_OK
    any op1 op2
    where
      @instCheck IsL2r(instr↦5↦1↦12)=TRUE
      @op1Fetch  op1=XL2r_op1(instr)
      @op2Fetch  op2=XL2r_op2(instr)
      @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
      @instrSize instrSize=InstSizeLong+0
  end

end
```

14.1.15machine XCoreIsaLoadAddress1

```
machine XCoreIsaLoadAddress1 // Load-address instructions: functionality

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION  end

  event Loading                           extends Loading                           where @false ⊥ end
  event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
  event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
  event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
  event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
  event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
  event LinkErrorException                extends LinkErrorException                where @false ⊥ end
  event IllegalPcException                extends IllegalPcException                where @false ⊥ end
  event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
  event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
  event LoadStoreException                extends LoadStoreException                where @false ⊥ end
  event IllegalPsException                extends IllegalPsException                where @false ⊥ end
  event ArithmeticException               extends ArithmeticException               where @false ⊥ end
  event EcallException                    extends EcallException                    where @false ⊥ end
  event ResourceDepException              extends ResourceDepException              where @false ⊥ end
  event KcallException                    extends KcallException                    where @false ⊥ end
  event Interrupt                         extends Interrupt                         where @false ⊥ end
  event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
  event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
  event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  where @false ⊥ end
```

```
  event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
  event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
  event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
  event BranchInstruction                extends BranchInstruction                where @false ⊥ end
  event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
  event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
  event DretInstruction                  extends DretInstruction                  where @false ⊥ end
  event KretInstruction                  extends KretInstruction                  where @false ⊥ end
  event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
  event PausingInstruction               extends PausingInstruction               where @false ⊥ end

  event LDAB_OK extends Write1RegisterInstruction
    any b i m res
    where
      @bType   b∈Bits32Nat
      @iType   i∈Bits32Nat
      @mType   m∈Bits32Nat
      @resDef  res=ModuloSubUnsigned(b↦ModuloMulUnsigned(i↦m))
      @regBits dBits=NatToBits32(res)
  end

  event LDAF_OK extends Write1RegisterInstruction
    any b i m res
    where
      @bType   b∈Bits32Nat
      @iType   i∈Bits32Nat
      @mType   m∈Bits32Nat
      @resDef  res=ModuloAddUnsigned(b↦ModuloMulUnsigned(i↦m))
      @regBits dBits=NatToBits32(res)
  end

end
```

14.1.16machine XCoreIsaLoadAddressFinal

```
machine XCoreIsaLoadAddressFinal // Load-address instructions: instruction encoding

refines XCoreIsaLoadAddress1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end
  event Write2RegisterInstruction extends Write2RegisterInstruction end
  event PausingInstruction extends PausingInstruction end

  event LDA16B_L3r_OK extends LDAB_OK
    any op2 op3
    where
      @instCheck IsL3r(instr↦6↦12)=TRUE
      @op1Fetch  d=XL3r_op1(instr)
      @op2Fetch  op2=XL3r_op2(instr)
      @op3Fetch  op3=XL3r_op3(instr)
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
      @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
      @m         m=2+0
      @instrSize instrSize=InstSizeLong+0
  end

  event LDA16F_L3r_OK extends LDAF_OK
    any op2 op3
    where
      @instCheck IsL3r(instr↦5↦12)=TRUE
      @op1Fetch  d=XL3r_op1(instr)
      @op2Fetch  op2=XL3r_op2(instr)
      @op3Fetch  op3=XL3r_op3(instr)
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
      @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
      @m         m=2+0
      @instrSize instrSize=InstSizeLong+0
  end

  event LDAPB_U10_OK extends LDAB_OK
    any op1
```

```
  where
    @instCheck IsU10(instr↦27↦1)=TRUE
    @op1Fetch  op1=XU10_op1(instr)
    @d         d=R11Index+0
    @b         b=nextPc+0
    @i         i=op1+0
    @m         m=2+0
    @instrSize instrSize=InstSizeShort+0
end

event LDAPB_Lu10_OK extends LDAB_OK
  any op1
  where
    @instCheck IsLu10(instr↦27↦1)=TRUE
    @op1Fetch  op1=XLu10_op1(instr)
    @d         d=R11Index+0
    @b         b=nextPc+0
    @i         i=op1+0
    @m         m=2+0
    @instrSize instrSize=InstSizeLong+0
end

event LDAPF_U10_OK extends LDAF_OK
  any op1
  where
    @instCheck IsU10(instr↦27↦0)=TRUE
    @op1Fetch  op1=XU10_op1(instr)
    @d         d=R11Index+0
    @b         b=nextPc+0
    @i         i=op1+0
    @m         m=2+0
    @instrSize instrSize=InstSizeShort+0
end

event LDAPF_Lu10_OK extends LDAF_OK
  any op1
  where
    @instCheck IsLu10(instr↦27↦0)=TRUE
    @op1Fetch  op1=XLu10_op1(instr)
    @d         d=R11Index+0
    @b         b=nextPc+0
    @i         i=op1+0
    @m         m=2+0
    @instrSize instrSize=InstSizeLong+0
end
```

```
event LDAWB_L3r_OK extends LDAB_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦4↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @m         m=Bpw+0
    @instrSize instrSize=InstSizeLong+0
end

event LDAWB_L2rus_OK extends LDAB_OK
  any op2 op3
  where
    @instCheck IsL2rus(instr↦20↦12)=TRUE
    @op1Fetch  d=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @op3Fetch  op3=XL2rus_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=op3+0
    @m         m=Bpw+0
    @instrSize instrSize=InstSizeLong+0
end

event LDAWCP_U6_OK extends LDAF_OK
  any op1
  where
    @instCheck IsU6(instr↦15↦1↦13)=TRUE
    @op1Fetch  op1=XU6_op1(instr)
    @d         d=R11Index+0
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op1+0
    @m         m=Bpw+0
    @instrSize instrSize=InstSizeShort+0
end

event LDAWCP_Lu6_OK extends LDAF_OK
  any op1
  where
    @instCheck IsLu6(instr↦15↦1↦13)=TRUE
    @op1Fetch  op1=XLu6_op1(instr)
    @d         d=R11Index+0
```

```
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i          i=op1+0
    @m          m=Bpw+0
    @instrSize instrSize=InstSizeLong+0
end

event LDAWDP_Ru6_OK extends LDAF_OK
  any op2
  where
    @instCheck IsRu6(instr↦12↦0)=TRUE
    @op1Fetch  d=XRu6_op1(instr)
    @op2Fetch  op2=XRu6_op2(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i          i=op2+0
    @m          m=Bpw+0
    @instrSize instrSize=InstSizeShort+0
end

event LDAWDP_Lru6_OK extends LDAF_OK
  any op2
  where
    @instCheck IsLru6(instr↦12↦0)=TRUE
    @op1Fetch  d=XLru6_op1(instr)
    @op2Fetch  op2=XLru6_op2(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i          i=op2+0
    @m          m=Bpw+0
    @instrSize instrSize=InstSizeLong+0
end

event LDAWF_L3r_OK extends LDAF_OK
  any op2 op3
  where
    @instCheck IsL3r(instr↦3↦12)=TRUE
    @op1Fetch  d=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @m          m=Bpw+0
    @instrSize instrSize=InstSizeLong+0
end

event LDAWF_L2rus_OK extends LDAF_OK
  any op2 op3
```

```
    where
      @instCheck IsL2rus(instr↦19↦12)=TRUE
      @op1Fetch  d=XL2rus_op1(instr)
      @op2Fetch  op2=XL2rus_op2(instr)
      @op3Fetch  op3=XL2rus_op3(instr)
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
      @i         i=op3+0
      @m         m=Bpw+0
      @instrSize instrSize=InstSizeLong+0
  end

  event LDAWSP_Ru6_OK extends LDAF_OK
    any op2
    where
      @instCheck IsRu6(instr↦12↦1)=TRUE
      @op1Fetch  d=XRu6_op1(instr)
      @op2Fetch  op2=XRu6_op2(instr)
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=op2+0
      @m         m=Bpw+0
      @instrSize instrSize=InstSizeShort+0
  end

  event LDAWSP_Lru6_OK extends LDAF_OK
    any op2
    where
      @instCheck IsLru6(instr↦12↦1)=TRUE
      @op1Fetch  d=XLru6_op1(instr)
      @op2Fetch  op2=XLru6_op2(instr)
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=op2+0
      @m         m=Bpw+0
      @instrSize instrSize=InstSizeLong+0
  end

end
```

14.1.17machine XCoreIsaLock1

```
machine XCoreIsaLock1  // Lock-resource control instructions

refines XCoreFinal

sees XCoreBasic XCoreLocks

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

          lockStatus lockLocked

invariants
    @lockStatus lockStatus ∈LockIndexDom → LockStatus
    @lockLocked lockLocked ∈LockIndexDom → BOOL

events

  event INITIALISATION extends INITIALISATION
    then
    @lockStatus lockStatus ≔  {0↦FREE} ⩤ {1↦FREE} ⩤ {2↦FREE} ⩤ {3↦FREE}
    @lockLocked lockLocked :∈LockIndexDom → BOOL
  end

  event Loading                        extends Loading                        where @false ⊥ end
  event RunExternalInputs              extends RunExternalInputs              where @false ⊥ end
  event SelectNextThread               extends SelectNextThread               where @false ⊥ end
  event ThreadWaiting                  extends ThreadWaiting                  where @false ⊥ end
  event ThreadMpausing                 extends ThreadMpausing                 where @false ⊥ end
  event ThreadSpausing                 extends ThreadSpausing                 where @false ⊥ end
  event LinkErrorException             extends LinkErrorException             where @false ⊥ end
  event IllegalPcException             extends IllegalPcException             where @false ⊥ end
  event IllegalInstructionException    extends IllegalInstructionException    where @false ⊥ end
  event IllegalResourceException       extends IllegalResourceException       where @false ⊥ end
  event LoadStoreException             extends LoadStoreException             where @false ⊥ end
  event IllegalPsException             extends IllegalPsException             where @false ⊥ end
  event ArithmeticException            extends ArithmeticException            where @false ⊥ end
  event EcallException                 extends EcallException                 where @false ⊥ end
  event ResourceDepException           extends ResourceDepException           where @false ⊥ end
  event KcallException                 extends KcallException                 where @false ⊥ end
  event Interrupt                      extends Interrupt                      where @false ⊥ end
  event ResourceEvent                  extends ResourceEvent                  where @false ⊥ end
```

```
event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event GETR_No_Lock extends Write1RegisterInstruction
  any resType
  where
    @resTypeType resType=ResourceLock
    @checkFree   CountFreeLocks(lockStatus)=0 // All used up
    @dBits       dBits=NatToBits32(NoResourceId)
end

event GETR_Lock_OK extends Write1RegisterInstruction
  any resType resIndex
  where
    @resTypeType resType=ResourceLock
    @checkFree   CountFreeLocks(lockStatus)≠0  // Lock available somewhere
    @resIndex    resIndex=FindFreeLock(lockStatus)
    @dBits       dBits=GetLockIdent(resIndex)
  then
    @lockStatus lockStatus(resIndex) ≔ RUNNING
    @lockLocked lockLocked(resIndex) ≔  FALSE
end

event FREER_Lock_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsLockIdent(resIdent)=TRUE
```

```
    @statusCheck IsLockInUse(resIdent↦lockStatus)=FALSE
end

event FREER_Lock_Locked extends IllegalResourceException
  any resIndex
  where
    @typeCheck   IsLockIdent(resIdent)=TRUE
    @statusCheck IsLockInUse(resIdent↦lockStatus)=TRUE
    @synchIndex  resIndex=GetResourceIndex(resIdent)
    @lockedCheck lockLocked(resIndex)=TRUE
end

event FREER_Lock_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsLockIdent(resIdent)=TRUE
    @statusCheck IsLockInUse(resIdent↦lockStatus)=TRUE
    @synchIndex  resIndex=GetResourceIndex(resIdent)
    @lockedCheck lockLocked(resIndex)=FALSE
  then
    @syncStatusChange lockStatus(resIndex) ≔ FREE
end

event IN_Lock_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsLockIdent(resIdent)=TRUE
    @statusCheck IsLockInUse(resIdent↦lockStatus)=FALSE
end

event IN_Lock_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsLockIdent(resIdent)=TRUE
    @statusCheck IsLockInUse(resIdent↦lockStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @available   lockLocked(resIndex)=TRUE
end

event IN_Lock_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
```

```
      @typeCheck   IsLockIdent(resIdent)=TRUE
      @statusCheck IsLockInUse(resIdent↦lockStatus)=TRUE
      @resIndex    resIndex=GetResourceIndex(resIdent)
      @available   lockLocked(resIndex)=FALSE
    then
      @changeAvail lockLocked(resIndex) ≔ TRUE
  end

  event OUT_Lock_NotInUse extends IllegalResourceException
    where
      @typeCheck   IsLockIdent(resIdent)=TRUE
      @statusCheck IsLockInUse(resIdent↦lockStatus)=FALSE
  end

  event OUT_Lock_OK extends Write0RegisterInstruction
    any resIdent resIndex
    where
      @resIdType   resIdent∈Bits
      @typeCheck   IsLockIdent(resIdent)=TRUE
      @statusCheck IsLockInUse(resIdent↦lockStatus)=TRUE
      @resIndex    resIndex=GetResourceIndex(resIdent)
      @available   lockLocked(resIndex)=TRUE
    then
      @changeDst lockLocked(resIndex) ≔ FALSE
  end

end
```

14.1.18machine XCoreIsaLockFinal

```
machine XCoreIsaLockFinal // Lock control instructions, instruction encoding
refines XCoreIsaLock1
sees XCoreBasic XCoreLocks

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

          lockStatus lockLocked

events
  event INITIALISATION extends INITIALISATION end
  event GETR_Rus_No_Lock extends GETR_No_Lock
    where
      @instCheck IsRus(instr↦16↦0)=TRUE
      @op1Fetch  d=XRus_op1(instr)
      @op2Fetch  resType=XRus_op2(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event GETR_Rus_Lock_OK extends GETR_Lock_OK
    where
      @instCheck   IsRus(instr↦16↦0)=TRUE
      @op1Fetch    d=XRus_op1(instr)
      @op2Fetch    resType=XRus_op2(instr)
      @instrSize   instrSize=InstSizeShort+0
  end

  event FREER_1r_Lock_NotInUse extends FREER_Lock_NotInUse
    any op1
    where
      @instCheck   Is1r(instr↦2↦0)=TRUE
      @op1Fetch    op1=X1r_op1(instr)
      @resIdent    resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event FREER_1r_Lock_Locked extends FREER_Lock_Locked
    any op1
    where
      @instCheck   Is1r(instr↦2↦0)=TRUE
      @op1Fetch    op1=X1r_op1(instr)
      @resIdent    resIdent=(threadRegArrays(threadIndex))(op1)
  end
```

```
event FREER_1r_Lock_OK extends FREER_Lock_OK
  any op1
  where
    @instCheck   Is1r(instr↦2↦0)=TRUE
    @op1Fetch    op1=X1r_op1(instr)
    @resIdent    resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize   instrSize=InstSizeShort+0
end

event IN_2r_Lock_NotInUse extends IN_Lock_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_Lock_Pausing_OK extends IN_Lock_Pausing_OK
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_Lock_OK extends IN_Lock_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event OUT_R2r_NotInUse extends OUT_Lock_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUT_R2r_Lock_OK extends OUT_Lock_OK
```

```
    any op1 op2
    where
      @instCheck IsR2r(instr↦21↦0)=TRUE
      @op1Fetch  op1=XR2r_op1(instr)
      @op2Fetch  op2=XR2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
      @instrSize instrSize=InstSizeShort+0
  end
end
```

14.1.19machine XCoreIsaMemoryRead1

```
machine XCoreIsaMemoryRead1 // Memory-read instructions: functionality

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event Loading                           extends Loading                           where @false ⊥ end
  event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
  event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
  event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
  event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
  event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
  event LinkErrorException                extends LinkErrorException                where @false ⊥ end
  event IllegalPcException                extends IllegalPcException                where @false ⊥ end
  event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
  event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
  event LoadStoreException                extends LoadStoreException                where @false ⊥ end
  event IllegalPsException                extends IllegalPsException                where @false ⊥ end
  event ArithmeticException               extends ArithmeticException               where @false ⊥ end
  event EcallException                    extends EcallException                    where @false ⊥ end
  event ResourceDepException              extends ResourceDepException              where @false ⊥ end
  event KcallException                    extends KcallException                    where @false ⊥ end
  event Interrupt                         extends Interrupt                         where @false ⊥ end
  event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
  event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
  event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  where @false ⊥ end
```

```
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event KRESTSP_OutOfRangeAddress extends LoadStoreException
  any b i
  where
    @bType   b∈Bits32Nat
    @iType   i∈Bits32Nat
    @nextKsp addr=b+(i* Bpw)
    @addrOk  WordAddrRangeOk(addr)=FALSE
end

event KRESTSP_MisalignedAddress extends LoadStoreException
  any b i
  where
    @bType   b∈Bits32Nat
    @iType   i∈Bits32Nat
    @nextKsp addr=b+(i* Bpw)
    @addrOk  WordAddrAlignedOk(addr)=FALSE
end

event KRESTSP_OK extends Write2RegisterInstruction
  any b i addr
  where
    @bType     b∈Bits32Nat
    @iType     i∈Bits32Nat
    @nextKsp   addr=b+(i* Bpw)
    @addrOk    WordAddrOkCheck(addr)=TRUE
    @nextKspBt dBits=NatToBits32(addr)
    @regBits   eBits=GetMemBits32(memArray↦addr)
end

event LD8U_OutOfRangeAddress extends LoadStoreException
  any b i
  where
```

```
    @addr        addr = b+i
    @addrCheck   ByteAddrRangeOk(addr)=FALSE
end

event LD8U_OK extends Write1RegisterInstruction
  any b i addr memData0
  where
    @bType       b∈Bits32Nat
    @iType       i∈Bits32Nat
    @addr        addr = b+i
    @addrCheck   ByteAddrOkCheck(addr)=TRUE
    @memData0Def memData0=memArray(addr)
    @resRegData  dBits=OneBits8ToBits32(memData0)
end

event LD16S_OutOfRangeAddress extends LoadStoreException
  any b i
  where
    @bType     b∈Bits32Nat
    @iType     i∈Bits32Nat
    @addr      addr = b+(i*2)
    @addrCheck ShortAddrRangeOk(addr)=FALSE
end

event LD16S_MisalignedAddress extends LoadStoreException
  any b i
  where
    @bType     b∈Bits32Nat
    @iType     i∈Bits32Nat
    @addr      addr = b+(i*2)
    @addrCheck ShortAddrAlignedOk(addr)=FALSE
end

event LD16S_OK extends Write1RegisterInstruction
  any b i addr
  where
    @bType      b∈Bits32Nat
    @iType      i∈Bits32Nat
    @addr       addr = b+(i*2)
    @addrCheck  ShortAddrOkCheck(addr)=TRUE
    @resRegData dBits=GetMemBits16(memArray↦addr)
end

event LDW_OutOfRangeAddress extends LoadStoreException
```

```
    any b i
    where
      @bType    b∈Bits32Nat
      @iType    i∈Bits32Nat
      @addrDef  addr=b+(i* Bpw)
      @addrOk   WordAddrRangeOk(addr)=FALSE
  end

  event LDW_MisalignedAddress extends LoadStoreException
    any b i
    where
      @bType    b∈Bits32Nat
      @iType    i∈Bits32Nat
      @addrDef  addr=b+(i* Bpw)
      @addrOk   WordAddrAlignedOk(addr)=FALSE
  end

  event LDW_OK extends Write1RegisterInstruction
    any b i addr
    where
      @bType       b∈Bits32Nat
      @iType       i∈Bits32Nat
      @addrDef     addr=b+(i* Bpw)
      @addrOk      WordAddrOkCheck(addr)=TRUE
      @regBitsDef dBits = GetMemBits32(memArray↦addr)
  end

end
```

14.1.20machine XCoreIsaMemoryReadFinal

```
machine XCoreIsaMemoryReadFinal  // Memory-read instructions: instruction encoding

refines XCoreIsaMemoryRead1

sees XCoreBasic

variables status interruptTriggered resourceTriggered memArray threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event KRESTSP_U6_OutOfRangeAddress extends KRESTSP_OutOfRangeAddress
    where
      @instCheck IsU6(instr↦15↦0↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XU6_op1(instr)
  end

  event KRESTSP_U6_MisalignedAddress extends KRESTSP_MisalignedAddress
    where
      @instCheck IsU6(instr↦15↦0↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XU6_op1(instr)
  end

  event KRESTSP_U6_OK extends KRESTSP_OK
    where
      @instCheck IsU6(instr↦15↦0↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XU6_op1(instr)
      @d         d=KspIndex+0
      @e         e=SpIndex+0
      @instrSize instrSize=InstSizeShort+0
  end

  event KRESTSP_Lu6_OutOfRangeAddress extends KRESTSP_OutOfRangeAddress
    where
      @instCheck IsLu6(instr↦15↦0↦15)=TRUE
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=XLu6_op1(instr)
  end
```

```
event KRESTSP_Lu6_MisalignedAddress extends KRESTSP_MisalignedAddress
  where
    @instCheck IsLu6(instr↦15↦0↦15)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=XLu6_op1(instr)
end

event KRESTSP_Lu6_OK extends KRESTSP_OK
  where
    @instCheck IsLu6(instr↦15↦0↦15)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=XLu6_op1(instr)
    @d         d=KspIndex+0
    @e         e=SpIndex+0
    @instrSize instrSize=InstSizeLong+0
end

event LD8U_3r_OutOfRangeAddress extends LD8U_OutOfRangeAddress
  any op2 op3
  where
    @instCheck   Is3r(instr↦17)=TRUE
    @op2Fetch    op2=X3r_op2(instr)
    @op3Fetch    op3=X3r_op3(instr)
    @b           b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i           i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event LD8U_3r_OK extends LD8U_OK
  any op2 op3
  where
    @instCheck   Is3r(instr↦17)=TRUE
    @op1Fetch    d=X3r_op1(instr)
    @op2Fetch    op2=X3r_op2(instr)
    @op3Fetch    op3=X3r_op3(instr)
    @b           b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i           i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize   instrSize=InstSizeShort+0
end

event LD16S_3r_OutOfRangeAddress extends LD16S_OutOfRangeAddress
  any op2 op3
  where
    @instCheck  Is3r(instr↦16)=TRUE
    @op2Fetch   op2=X3r_op2(instr)
    @op3Fetch   op3=X3r_op3(instr)
```

```
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event LD16S_3r_MisalignedAddress extends LD16S_MisalignedAddress
  any op2 op3
  where
    @instCheck  Is3r(instr↦16)=TRUE
    @op2Fetch   op2=X3r_op2(instr)
    @op3Fetch   op3=X3r_op3(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event LD16S_3r_OK extends LD16S_OK
  any op2 op3
  where
    @instCheck  Is3r(instr↦16)=TRUE
    @op1Fetch   d=X3r_op1(instr)
    @op2Fetch   op2=X3r_op2(instr)
    @op3Fetch   op3=X3r_op3(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize  instrSize=InstSizeShort+0
end

event LDET_0r_OutOfRangeAddress extends LDW_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦2↦1↦14)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=4+0
end

event LDET_0r_MisalignedAddress extends LDW_MisalignedAddress
  where
    @instCheck Is0r(instr↦2↦1↦14)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=4+0
end

event LDET_0r_OK extends LDW_OK
  where
    @instCheck Is0r(instr↦2↦1↦14)=TRUE
    @d         d=EtIndex+0
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
```

```
    @i          i=4+0
    @instrSize instrSize=InstSizeShort+0
end

event LDSED_0r_OutOfRangeAddress extends LDW_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦2↦1↦13)=TRUE
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=3+0
end

event LDSED_0r_MisalignedAddress extends LDW_MisalignedAddress
  where
    @instCheck Is0r(instr↦2↦1↦13)=TRUE
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=3+0
end

event LDSED_0r_OK extends LDW_OK
  where
    @instCheck Is0r(instr↦2↦1↦13)=TRUE
    @d          d=SedIndex+0
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=3+0
    @instrSize instrSize=InstSizeShort+0
end

event LDSPC_0r_OutOfRangeAddress extends LDW_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦1↦0↦12)=TRUE
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=1+0
end

event LDSPC_0r_MisalignedAddress extends LDW_MisalignedAddress
  where
    @instCheck Is0r(instr↦1↦0↦12)=TRUE
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=1+0
end

event LDSPC_0r_OK extends LDW_OK
  where
    @instCheck Is0r(instr↦1↦0↦12)=TRUE
    @d          d=SpcIndex+0
```

```
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=1+0
    @instrSize instrSize=InstSizeShort+0
end

event LDSSR_0r_OutOfRangeAddress extends LDW_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦1↦0↦14)=TRUE
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=2+0
end

event LDSSR_0r_MisalignedAddress extends LDW_MisalignedAddress
  where
    @instCheck Is0r(instr↦1↦0↦14)=TRUE
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=2+0
end

event LDSSR_0r_OK extends LDW_OK
  where
    @instCheck Is0r(instr↦1↦0↦14)=TRUE
    @d          d=SsrIndex+0
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i          i=2+0
    @instrSize instrSize=InstSizeShort+0
end

event LDW_3r_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2 op3
  where
    @instCheck  Is3r(instr↦9)=TRUE
    @op2Fetch   op2=X3r_op2(instr)
    @op3Fetch   op3=X3r_op3(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event LDW_3r_MisalignedAddress extends LDW_MisalignedAddress
  any op2 op3
  where
    @instCheck  Is3r(instr↦9)=TRUE
    @op2Fetch   op2=X3r_op2(instr)
    @op3Fetch   op3=X3r_op3(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
```

```
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event LDW_3r_OK extends LDW_OK
  any op2 op3
  where
    @instCheck  Is3r(instr↦9)=TRUE
    @op1Fetch   d=X3r_op1(instr)
    @op2Fetch   op2=X3r_op2(instr)
    @op3Fetch   op3=X3r_op3(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i          i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
    @instrSize  instrSize=InstSizeShort+0
end

event LDW_2rus_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2 op3
  where
    @instCheck Is2rus(instr↦1)=TRUE
    @op2Fetch  op2=X2rus_op2(instr)
    @op3Fetch  op3=X2rus_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=op3+0
end

event LDW_2rus_MisalignedAddress extends LDW_MisalignedAddress
  any op2 op3
  where
    @instCheck Is2rus(instr↦1)=TRUE
    @op2Fetch  op2=X2rus_op2(instr)
    @op3Fetch  op3=X2rus_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=op3+0
end

event LDW_2rus_OK extends LDW_OK
  any op2 op3
  where
    @instCheck Is2rus(instr↦1)=TRUE
    @op1Fetch  d=X2rus_op1(instr)
    @op2Fetch  op2=X2rus_op2(instr)
    @op3Fetch  op3=X2rus_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=op3+0
    @instrSize instrSize=InstSizeShort+0
```

```
end

event LDWCP_Ru6_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2
  where
    @instCheck IsRu6(instr↦13↦1)=TRUE
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op2+0
end

event LDWCP_Ru6_MisalignedAddress extends LDW_MisalignedAddress
  any op2
  where
    @instCheck IsRu6(instr↦13↦1)=TRUE
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op2+0
end

event LDWCP_Ru6_OK extends LDW_OK
  any op2
  where
    @instCheck IsRu6(instr↦13↦1)=TRUE
    @op1Fetch  d=XRu6_op1(instr)
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op2+0
    @instrSize instrSize=InstSizeShort+0
end

event LDWCP_Lru6_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2
  where
    @instCheck IsLru6(instr↦13↦1)=TRUE
    @op2Fetch  op2=XLru6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op2+0
end

event LDWCP_Lru6_MisalignedAddress extends LDW_MisalignedAddress
  any op2
  where
    @instCheck IsLru6(instr↦13↦1)=TRUE
    @op2Fetch  op2=XLru6_op2(instr)
```

```
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i          i=op2+0
end

event LDWCP_Lru6_OK extends LDW_OK
  any op2
  where
    @instCheck IsLru6(instr↦13↦1)=TRUE
    @op1Fetch  d=XLru6_op1(instr)
    @op2Fetch  op2=XLru6_op2(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i          i=op2+0
    @instrSize instrSize=InstSizeLong+0
end

event LDWCPL_U10_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op1
  where
    @instCheck IsU10(instr↦28↦1)=TRUE
    @op1Fetch  op1=XU10_op1(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i          i=op1+0
end

event LDWCPL_U10_MisalignedAddress extends LDW_MisalignedAddress
  any op1
  where
    @instCheck IsU10(instr↦28↦1)=TRUE
    @op1Fetch  op1=XU10_op1(instr)
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i          i=op1+0
end

event LDWCPL_U10_OK extends LDW_OK
  any op1
  where
    @instCheck IsU10(instr↦28↦1)=TRUE
    @op1Fetch  op1=XU10_op1(instr)
    @d          d=R11Index+0
    @b          b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i          i=op1+0
    @instrSize instrSize=InstSizeShort+0
end

event LDWCPL_Lu10_OutOfRangeAddress extends LDW_OutOfRangeAddress
```

```
  any op1
  where
    @instCheck IsLu10(instr↦28↦1)=TRUE
    @op1Fetch  op1=XLu10_op1(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op1+0
end

event LDWCPL_Lu10_MisalignedAddress extends LDW_MisalignedAddress
  any op1
  where
    @instCheck IsLu10(instr↦28↦1)=TRUE
    @op1Fetch  op1=XLu10_op1(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op1+0
end

event LDWCPL_Lu10_OK extends LDW_OK
  any op1
  where
    @instCheck IsLu10(instr↦28↦1)=TRUE
    @op1Fetch  op1=XLu10_op1(instr)
    @d         d=R11Index+0
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(CpIndex))
    @i         i=op1+0
    @instrSize instrSize=InstSizeLong+0
end

event LDWDP_Ru6_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2
  where
    @instCheck IsRu6(instr↦11↦0)=TRUE
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i         i=op2+0
end

event LDWDP_Ru6_MisalignedAddress extends LDW_MisalignedAddress
  any op2
  where
    @instCheck IsRu6(instr↦11↦0)=TRUE
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i         i=op2+0
end
```

```
event LDWDP_Ru6_OK extends LDW_OK
  any op2
  where
    @instCheck IsRu6(instr↦11↦0)=TRUE
    @op1Fetch  d=XRu6_op1(instr)
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i         i=op2+0
    @instrSize instrSize=InstSizeShort+0
end

event LDWDP_Lru6_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2
  where
    @instCheck IsLru6(instr↦11↦0)=TRUE
    @op2Fetch  op2=XLru6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i         i=op2+0
end

event LDWDP_Lru6_MisalignedAddress extends LDW_MisalignedAddress
  any op2
  where
    @instCheck IsLru6(instr↦11↦0)=TRUE
    @op2Fetch  op2=XLru6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i         i=op2+0
end

event LDWDP_Lru6_OK extends LDW_OK
  any op2
  where
    @instCheck IsLru6(instr↦11↦0)=TRUE
    @op1Fetch  d=XLru6_op1(instr)
    @op2Fetch  op2=XLru6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))
    @i         i=op2+0
    @instrSize instrSize=InstSizeLong+0
end

event LDWSP_Ru6_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2
  where
    @instCheck IsRu6(instr↦11↦1)=TRUE
```

```
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=op2+0
end

event LDWSP_Ru6_MisalignedAddress extends LDW_MisalignedAddress
  any op2
  where
    @instCheck IsRu6(instr↦11↦1)=TRUE
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=op2+0
end

event LDWSP_Ru6_OK extends LDW_OK
  any op2
  where
    @instCheck IsRu6(instr↦11↦1)=TRUE
    @op1Fetch  d=XRu6_op1(instr)
    @op2Fetch  op2=XRu6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=op2+0
    @instrSize instrSize=InstSizeShort+0
end

event LDWSP_Lru6_OutOfRangeAddress extends LDW_OutOfRangeAddress
  any op2
  where
    @instCheck IsLru6(instr↦11↦1)=TRUE
    @op2Fetch  op2=XLru6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=op2+0
end

event LDWSP_Lru6_MisalignedAddress extends LDW_MisalignedAddress
  any op2
  where
    @instCheck IsLru6(instr↦11↦1)=TRUE
    @op2Fetch  op2=XLru6_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @i         i=op2+0
end

event LDWSP_Lru6_OK extends LDW_OK
  any op2
```

```
    where
      @instCheck IsLru6(instr↦11↦1)=TRUE
      @op1Fetch  d=XLru6_op1(instr)
      @op2Fetch  op2=XLru6_op2(instr)
      @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
      @i         i=op2+0
      @instrSize instrSize=InstSizeLong+0
  end

end
```

14.1.21machine XCoreIsaMemoryWrite1

```
machine XCoreIsaMemoryWrite1 // Memory-write instructions

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event Loading                           extends Loading                           where @false ⊥ end
  event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
  event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
  event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
  event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
  event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
  event LinkErrorException                extends LinkErrorException                where @false ⊥ end
  event IllegalPcException                extends IllegalPcException                where @false ⊥ end
  event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
  event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
  event LoadStoreException                extends LoadStoreException                where @false ⊥ end
  event IllegalPsException                extends IllegalPsException                where @false ⊥ end
  event ArithmeticException               extends ArithmeticException               where @false ⊥ end
  event EcallException                    extends EcallException                    where @false ⊥ end
  event ResourceDepException              extends ResourceDepException              where @false ⊥ end
  event KcallException                    extends KcallException                    where @false ⊥ end
  event Interrupt                         extends Interrupt                         where @false ⊥ end
  event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
  event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
  event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  where @false ⊥ end
```

```
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event ENTSP_OutOfRangeAddress extends LoadStoreException
  any x
  where
    @condTrue  x>0
    @addr      addr=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @addrOk    WordAddrRangeOk(addr)=FALSE
end

event ENTSP_MisalignedAddress extends LoadStoreException
  any x
  where
    @condTrue  x>0
    @addr      addr=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @addrOk    WordAddrAlignedOk(addr)=FALSE
end

event ENTSP_OK extends Write4Memory1RegisterInstruction
  any x
  where
    @condTrue  x>0
    @memData   memData=(threadRegArrays(threadIndex))(LrIndex)
    @addr      addr=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @d         d=SpIndex+0
    @xType     x∈Bits32Nat
    @dBits     dBits=NatToBits32(ModuloSubUnsigned(addr↦ModuloMulUnsigned(x↦Bpw)))
end

event ENTSP_Zero_OK extends Write0RegisterInstruction
  any x
  where
    @condFalse x≤0
end
```

```
event KENTSP_OutOfRangeAddress extends LoadStoreException
  where
    @ksp      addr=Bits32ToNat((threadRegArrays(threadIndex))(KspIndex))
    @addrOk   WordAddrRangeOk(addr)=FALSE
end

event KENTSP_MisalignedAddress extends LoadStoreException
  where
    @ksp      addr=Bits32ToNat((threadRegArrays(threadIndex))(KspIndex))
    @addrOk   WordAddrAlignedOk(addr)=FALSE
end

event KENTSP_OK extends Write4Memory1RegisterInstruction
  any x
  where
    @memData memData=(threadRegArrays(threadIndex))(SpIndex)
    @ksp      addr=Bits32ToNat((threadRegArrays(threadIndex))(KspIndex))
    @d        d=SpIndex+0
    @xType    x∈Bits32Nat
    @dBits    dBits=NatToBits32(ModuloSubUnsigned(addr↦ModuloMulUnsigned(x↦Bpw)))
end

event ST8_OutOfRangeAddress extends LoadStoreException
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
    @addr     addr=b+i
    @addrOk   ByteAddrRangeOk(addr)=FALSE
end

event ST8_OK extends Write1Memory0RegisterInstruction
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
    @addr     addr=b+i
end

event ST16_OutOfRangeAddress extends LoadStoreException
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
```

```
    @addr     addr=b+(i*2)
    @addrOk   ShortAddrRangeOk(addr)=FALSE
end

event ST16_MisalignedAddress extends LoadStoreException
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
    @addr     addr=b+(i*2)
    @addrOk   ShortAddrAlignedOk(addr)=FALSE
end

event ST16_OK extends Write2Memory0RegisterInstruction
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
    @addr     addr=b+(i*2)
end

event STW_OutOfRangeAddress extends LoadStoreException
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
    @addr     addr=b+(i*Bpw)
    @addrOk   WordAddrRangeOk(addr)=FALSE
end

event STW_MisalignedAddress extends LoadStoreException
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
    @addr     addr=b+(i*Bpw)
    @addrOk   WordAddrAlignedOk(addr)=FALSE
end

event STW_OK extends Write4Memory0RegisterInstruction
  any b i
  where
    @bType    b∈Bits32Nat
    @iType    i∈Bits32Nat
```

```
    @addr     addr=b+(i*Bpw)
end

event STET_OutOfRangeAddress extends LoadStoreException
  any b
  where
    @bType    b∈Bits32Nat
    @addr     addr=b+(4*Bpw)
    @addrOk   WordAddrRangeOk(addr)=FALSE
end

event STET_MisalignedAddress extends LoadStoreException
  any b
  where
    @bType    b∈Bits32Nat
    @addr     addr=b+(4*Bpw)
    @addrOk   WordAddrAlignedOk(addr)=FALSE
end

event STET_OK extends Write4Memory0RegisterInstruction
  any b
  where
    @memData memData=(threadRegArrays(threadIndex))(EtIndex)
    @bType    b∈Bits32Nat
    @addr     addr=b+(4*Bpw)
end

event STSED_OutOfRangeAddress extends LoadStoreException
  any b
  where
    @bType    b∈Bits32Nat
    @addr     addr=b+(3*Bpw)
    @addrOk   WordAddrRangeOk(addr)=FALSE
end

event STSED_MisalignedAddress extends LoadStoreException
  any b
  where
    @bType    b∈Bits32Nat
    @addr     addr=b+(3*Bpw)
    @addrOk   WordAddrAlignedOk(addr)=FALSE
end

event STSED_OK extends Write4Memory0RegisterInstruction
```

```
  any b
  where
    @memData memData=(threadRegArrays(threadIndex))(SedIndex)
    @bType   b∈Bits32Nat
    @addr    addr=b+(3*Bpw)
end

event STSPC_OutOfRangeAddress extends LoadStoreException
  any b
  where
    @bType   b∈Bits32Nat
    @addr    addr=b+(1*Bpw)
    @addrOk  WordAddrRangeOk(addr)=FALSE
end

event STSPC_MisalignedAddress extends LoadStoreException
  any b
  where
    @bType   b∈Bits32Nat
    @addr    addr=b+(1*Bpw)
    @addrOk  WordAddrAlignedOk(addr)=FALSE
end

event STSPC_OK extends Write4Memory0RegisterInstruction
  any b
  where
    @memData memData=(threadRegArrays(threadIndex))(SpcIndex)
    @bType   b∈Bits32Nat
    @addr    addr=b+(1*Bpw)
end

event STSSR_OutOfRangeAddress extends LoadStoreException
  any b
  where
    @bType   b∈Bits32Nat
    @addr    addr=b+(2*Bpw)
    @addrOk  WordAddrRangeOk(addr)=FALSE
end

event STSSR_MisalignedAddress extends LoadStoreException
  any b
  where
    @bType   b∈Bits32Nat
    @addr    addr=b+(2*Bpw)
```

```
    @addrOk  WordAddrAlignedOk(addr)=FALSE
end

event STSSR_OK extends Write4Memory0RegisterInstruction
  any b
  where
    @memData memData=(threadRegArrays(threadIndex))(SsrIndex)
    @bType   b∈Bits32Nat
    @addr    addr=b+(2*Bpw)
end

event STWDP_OutOfRangeAddress extends LoadStoreException
  any i
  where
    @addrDef addr=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))+(i * Bpw)
    @addrOk  WordAddrRangeOk(addr)=FALSE
end

event STWDP_MisalignedAddress extends LoadStoreException
  any i
  where
    @addrDef addr=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))+(i * Bpw)
    @addrOk  WordAddrAlignedOk(addr)=FALSE
end

event STWDP_OK extends Write4Memory0RegisterInstruction
  any i
  where
    @addrDef addr=Bits32ToNat((threadRegArrays(threadIndex))(DpIndex))+(i * Bpw)
end

event STWSP_OutOfRangeAddress extends LoadStoreException
  any i
  where
    @addrDef addr=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))+(i * Bpw)
    @addrOk  WordAddrRangeOk(addr)=FALSE
end

event STWSP_MisalignedAddress extends LoadStoreException
  any i
  where
    @addrDef addr=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))+(i * Bpw)
    @addrOk  WordAddrAlignedOk(addr)=FALSE
end
```

```
  event STWSP_OK extends Write4Memory0RegisterInstruction
    any i
    where
      @addrDef addr=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))+(i * Bpw)
  end

end
```

14.1.22machine XCoreIsaMemoryWriteFinal

```
machine XCoreIsaMemoryWriteFinal

refines XCoreIsaMemoryWrite1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event ENTSP_U6_OutOfRangeAddress extends ENTSP_OutOfRangeAddress
    where
      @instCheck IsU6(instr↦14↦1↦13)=TRUE
      @op1Fetch  x=XU6_op1(instr)
  end

  event ENTSP_U6_MisalignedAddress extends ENTSP_MisalignedAddress
    where
      @instCheck IsU6(instr↦14↦1↦13)=TRUE
      @op1Fetch  x=XU6_op1(instr)
  end

  event ENTSP_U6_OK extends ENTSP_OK
    where
      @instCheck IsU6(instr↦14↦1↦13)=TRUE
      @op1Fetch  x=XU6_op1(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event ENTSP_Lu6_OutOfRangeAddress extends ENTSP_OutOfRangeAddress
    where
      @instCheck IsLu6(instr↦14↦1↦13)=TRUE
      @op1Fetch  x=XLu6_op1(instr)
  end

  event ENTSP_Lu6_MisalignedAddress extends ENTSP_MisalignedAddress
    where
      @instCheck IsLu6(instr↦14↦1↦13)=TRUE
      @op1Fetch  x=XLu6_op1(instr)
```

```
end

event ENTSP_Lu6_OK extends ENTSP_OK
  where
    @instCheck IsLu6(instr↦14↦1↦13)=TRUE
    @op1Fetch  x=XLu6_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event ENTSP_Zero_U6_OK extends ENTSP_Zero_OK
  where
    @instCheck IsU6(instr↦14↦1↦13)=TRUE
    @op1Fetch  x=XU6_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event ENTSP_Zero_Lu6_OK extends ENTSP_Zero_OK
  where
    @instCheck IsLu6(instr↦14↦1↦13)=TRUE
    @op1Fetch  x=XLu6_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event KENTSP_U6_OutOfRangeAddress extends KENTSP_OutOfRangeAddress
  where
    @instCheck IsU6(instr↦15↦0↦14)=TRUE
end

event KENTSP_U6_MisalignedAddress extends KENTSP_MisalignedAddress
  where
    @instCheck IsU6(instr↦15↦0↦14)=TRUE
end

event KENTSP_U6_OK extends KENTSP_OK
  where
    @instCheck IsU6(instr↦15↦0↦14)=TRUE
    @op1Fetch  x=XU6_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event KENTSP_Lu6_OutOfRangeAddress extends KENTSP_OutOfRangeAddress
  where
    @instCheck IsLu6(instr↦15↦0↦14)=TRUE
end
```

```
event KENTSP_Lu6_MisalignedAddress extends KENTSP_MisalignedAddress
  where
    @instCheck IsLu6(instr↦15↦0↦14)=TRUE
end

event KENTSP_Lu6_OK extends KENTSP_OK
  where
    @instCheck IsLu6(instr↦15↦0↦14)=TRUE
    @op1Fetch  x=XLu6_op1(instr)
    @instrSize instrSize=InstSizeLong+0
end

event ST8_L3r_OutOfRangeAddress extends ST8_OutOfRangeAddress
  any op2 op3
  where
    @instCheck IsL3r(instr↦17↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event ST8_L3r_OK extends ST8_OK
  any op1 op2 op3
  where
    @instCheck IsL3r(instr↦17↦12)=TRUE
    @op1Fetch  op1=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @instrSize instrSize=InstSizeLong+0
    @memData   memData=(threadRegArrays(threadIndex))(op1)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2)) // Note different to published specification
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3)) // Note different to published specification
end

event ST16_L3r_OutOfRangeAddress extends ST16_OutOfRangeAddress
  any op2 op3
  where
    @instCheck IsL3r(instr↦16↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end
```

```
event ST16_L3r_MisalignedAddress extends ST16_MisalignedAddress
  any op2 op3
  where
    @instCheck IsL3r(instr↦16↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event ST16_L3r_OK extends ST16_OK
  any op1 op2 op3
  where
    @instCheck IsL3r(instr↦16↦12)=TRUE
    @op1Fetch  op1=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @instrSize instrSize=InstSizeLong+0
    @memData   memData=(threadRegArrays(threadIndex))(op1)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2)) // Note different to published specification
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3)) // Note different to published specification
end

event STW_L3r_OutOfRangeAddress extends STW_OutOfRangeAddress
  any op2 op3
  where
    @instCheck IsL3r(instr↦0↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event STW_L3r_MisalignedAddress extends STW_MisalignedAddress
  any op2 op3
  where
    @instCheck IsL3r(instr↦0↦12)=TRUE
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event STW_L3r_OK extends STW_OK
  any op1 op2 op3
```

```
  where
    @instCheck IsL3r(instr↦0↦12)=TRUE
    @op1Fetch  op1=XL3r_op1(instr)
    @op2Fetch  op2=XL3r_op2(instr)
    @op3Fetch  op3=XL3r_op3(instr)
    @instrSize instrSize=InstSizeLong+0
    @memData   memData=(threadRegArrays(threadIndex))(op1)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @i         i=Bits32ToNat((threadRegArrays(threadIndex))(op3))
end

event STW_2rus_OutOfRangeAddress extends STW_OutOfRangeAddress
  any op2
  where
    @instCheck Is2rus(instr↦0)=TRUE
    @op2Fetch  op2=X2rus_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  i=X2rus_op3(instr)
end

event STW_2rus_MisalignedAddress extends STW_MisalignedAddress
  any op2
  where
    @instCheck Is2rus(instr↦0)=TRUE
    @op2Fetch  op2=X2rus_op2(instr)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  i=X2rus_op3(instr)
end

event STW_2rus_OK extends STW_OK
  any op1 op2
  where
    @instCheck Is2rus(instr↦0)=TRUE
    @op1Fetch  op1=X2rus_op1(instr)
    @op2Fetch  op2=X2rus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
    @memData   memData=(threadRegArrays(threadIndex))(op1)
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @op3Fetch  i=X2rus_op3(instr)
end

event STET_0r_OutOfRangeAddress extends STET_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦1↦1↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
```

```
end

event STET_0r_MisalignedAddress extends STET_MisalignedAddress
  where
    @instCheck Is0r(instr↦1↦1↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end

event STET_0r_OK extends STET_OK
  where
    @instCheck Is0r(instr↦1↦1↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @instrSize instrSize=InstSizeShort+0
end

event STSED_0r_OutOfRangeAddress extends STSED_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦1↦1↦12)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end

event STSED_0r_MisalignedAddress extends STSED_MisalignedAddress
  where
    @instCheck Is0r(instr↦1↦1↦12)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end

event STSED_0r_OK extends STSED_OK
  where
    @instCheck Is0r(instr↦1↦1↦12)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @instrSize instrSize=InstSizeShort+0
end

event STSPC_0r_OutOfRangeAddress extends STSPC_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦1↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end

event STSPC_0r_MisalignedAddress extends STSPC_MisalignedAddress
  where
    @instCheck Is0r(instr↦1↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end
```

```
event STSPC_0r_OK extends STSPC_OK
  where
    @instCheck Is0r(instr↦1↦0↦13)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @instrSize instrSize=InstSizeShort+0
end

event STSSR_0r_OutOfRangeAddress extends STSSR_OutOfRangeAddress
  where
    @instCheck Is0r(instr↦1↦0↦15)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end

event STSSR_0r_MisalignedAddress extends STSSR_MisalignedAddress
  where
    @instCheck Is0r(instr↦1↦0↦15)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
end

event STSSR_0r_OK extends STSSR_OK
  where
    @instCheck Is0r(instr↦1↦0↦15)=TRUE
    @b         b=Bits32ToNat((threadRegArrays(threadIndex))(SpIndex))
    @instrSize instrSize=InstSizeShort+0
end

event STWDP_Ru6_OutOfRangeAddress extends STWDP_OutOfRangeAddress
  where
    @instCheck IsRu6(instr↦10↦0)=TRUE
    @op2Fetch  i=XRu6_op2(instr)
end

event STWDP_Ru6_MisalignedAddress extends STWDP_MisalignedAddress
  where
    @instCheck IsRu6(instr↦10↦0)=TRUE
    @op2Fetch  i=XRu6_op2(instr)
end

event STWDP_Ru6_OK extends STWDP_OK
  any op1
  where
    @instCheck IsRu6(instr↦10↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @memData   memData=(threadRegArrays(threadIndex))(op1)
```

```
    @op2Fetch  i=XRu6_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event STWDP_Lru6_OutOfRangeAddress extends STWDP_OutOfRangeAddress
  where
    @instCheck IsLru6(instr↦10↦0)=TRUE
    @op2Fetch  i=XLru6_op2(instr)
end

event STWDP_Lru6_MisalignedAddress extends STWDP_MisalignedAddress
  where
    @instCheck IsLru6(instr↦10↦0)=TRUE
    @op2Fetch  i=XLru6_op2(instr)
end

event STWDP_Lru6_OK extends STWDP_OK
  any op1
  where
    @instCheck IsLru6(instr↦10↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @memData   memData=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  i=XLru6_op2(instr)
    @instrSize instrSize=InstSizeLong+0
end

event STWSP_Ru6_OutOfRangeAddress extends STWSP_OutOfRangeAddress
  where
    @instCheck IsRu6(instr↦10↦1)=TRUE
    @op2Fetch  i=XRu6_op2(instr)
end

event STWSP_Ru6_MisalignedAddress extends STWSP_MisalignedAddress
  where
    @instCheck IsRu6(instr↦10↦1)=TRUE
    @op2Fetch  i=XRu6_op2(instr)
end

event STWSP_Ru6_OK extends STWSP_OK
  any op1
  where
    @instCheck IsRu6(instr↦10↦1)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @memData   memData=(threadRegArrays(threadIndex))(op1)
    @op2Fetch  i=XRu6_op2(instr)
```

```
      @instrSize instrSize=InstSizeShort+0
  end

  event STWSP_Lru6_OutOfRangeAddress extends STWSP_OutOfRangeAddress
    where
      @instCheck IsLru6(instr↦10↦1)=TRUE
      @op2Fetch  i=XLru6_op2(instr)
  end

  event STWSP_Lru6_MisalignedAddress extends STWSP_MisalignedAddress
    where
      @instCheck IsLru6(instr↦10↦1)=TRUE
      @op2Fetch  i=XLru6_op2(instr)
  end

  event STWSP_Lru6_OK extends STWSP_OK
    any op1
    where
      @instCheck IsLru6(instr↦10↦1)=TRUE
      @op1Fetch  op1=XLru6_op1(instr)
      @memData   memData=(threadRegArrays(threadIndex))(op1)
      @op2Fetch  i=XLru6_op2(instr)
      @instrSize instrSize=InstSizeLong+0
  end

end
```

14.1.23machine XCoreIsaPort1

```
machine XCoreIsaPort1 // Port control instructions

refines XCoreFinal

sees XCoreBasic XCorePorts XCoreClocks

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

          portIndex portStatus
          portEventMode portInMode portBufferedMode
          portTriggered portThreadIdent portEnvironmentVector portVector
          portTransferData portTransferDataFilled portClockIdent
          portTimerValue portTimerTriggerValue portTimestamp
          incomplete

invariants
  @incomplete             incomplete ∈ BOOL
  @portIndex              portIndex ∈ PortIndexDom
  @portStatus             portStatus ∈ PortIndexDom → PortStatus
  @portEventMode          portEventMode ∈ PortIndexDom → BOOL
  @portInMode             portInMode ∈ PortIndexDom → BOOL
  @portBufferedMode       portBufferedMode ∈ PortIndexDom → BOOL
  @portTransferData       portTransferData ∈  PortIndexDom → Bits
  @portTransferDataFilled portTransferDataFilled ∈ PortIndexDom → BOOL
  @portClockIdent         portClockIdent ∈  PortIndexDom → Bits
  @portTimerValue         portTimerValue ∈PortIndexDom → Bits16Nat
  @portTimerTriggerValue  portTimerTriggerValue ∈PortIndexDom → Bits16Nat
  @portTimestamp          portTimestamp ∈PortIndexDom → Bits16Nat
  @portTriggered          portTriggered ∈  PortIndexDom → BOOL
  @portThreadIdent        portThreadIdent ∈  PortIndexDom → Bits
  @portEnvVec             portEnvironmentVector ∈  PortIndexDom → Bits32Nat
  @portVec                portVector ∈  PortIndexDom → Bits32Nat

events

  event INITIALISATION extends INITIALISATION
    then
      @incomplete             incomplete ≔ FALSE
      @portIndex              portIndex ≔0
      @portStatus             portStatus ≔  {0↦FREE} ⩤ {1↦FREE} ⩤ {2↦FREE} ⩤ {3↦FREE}
```

```
    @portEventMode          portEventMode ≔  {0↦TRUE} □ {1↦TRUE} □ {2↦TRUE} □ {3↦TRUE}
    @portInMode             portInMode :∈ PortIndexDom → BOOL
    @portBufferedMode       portBufferedMode :∈ PortIndexDom → BOOL
    @portTransferData       portTransferData :∈  PortIndexDom → Bits
    @portTransferDataFilled portTransferDataFilled ≔  {0↦FALSE} □ {1↦FALSE} □ {2↦FALSE} □ {3↦FALSE}
    @portClockIdent         portClockIdent :∈  PortIndexDom → Bits
    @portTimerValue         portTimerValue :∈PortIndexDom → Bits16Nat
    @portTimerTriggerValue  portTimerTriggerValue :∈PortIndexDom → Bits16Nat
    @portTimestamp          portTimestamp :∈PortIndexDom → Bits16Nat
    @portTriggered          portTriggered ≔  {0↦FALSE} □ {1↦FALSE} □ {2↦FALSE} □ {3↦FALSE}
    @portThreadIdent        portThreadIdent :∈  PortIndexDom → Bits
    @portEnvVec             portEnvironmentVector :∈  PortIndexDom → Bits32Nat
    @portVec                portVector :∈  PortIndexDom → Bits32Nat
end

event Loading                          extends Loading                          where @false ⊥ end
event RunExternalInputs                extends RunExternalInputs                where @false ⊥ end
event SelectNextThread                 extends SelectNextThread                 where @false ⊥ end
event ThreadWaiting                    extends ThreadWaiting                    where @false ⊥ end
event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
event LinkErrorException               extends LinkErrorException               where @false ⊥ end
event IllegalPcException               extends IllegalPcException               where @false ⊥ end
event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
event LoadStoreException               extends LoadStoreException               where @false ⊥ end
event IllegalPsException               extends IllegalPsException               where @false ⊥ end
event ArithmeticException              extends ArithmeticException              where @false ⊥ end
event EcallException                   extends EcallException                   where @false ⊥ end
event ResourceDepException             extends ResourceDepException             where @false ⊥ end
event KcallException                   extends KcallException                   where @false ⊥ end
event Interrupt                        extends Interrupt                        where @false ⊥ end
event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
```

```
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event SETCLK_Port_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
 end

event SETCLK_Port_BadIdent extends IllegalResourceException
  any clkBits
  where
    @identOk     IsPortIdent(resIdent)=TRUE
    @inUse       IsPortInUse(resIdent↦portStatus)=TRUE
    @clkBitsType clkBits∈Bits
    @clkOk       IsClockIdent(clkBits)=FALSE
 end

event SETCLK_Port_OK extends Write0RegisterInstruction
  any resIdent resIndex clkBits
  where
    @resIdType   resIdent∈Bits
    @identOk     IsPortIdent(resIdent)=TRUE
    @inUse       IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex    resIndex=GetResourceIndex(resIdent)
    @clkBitsType clkBits∈Bits
    @clkOk       IsClockIdent(clkBits)=TRUE
  then
    @changeClk portClockIdent(resIndex) ≔ clkBits
 end

event OUT_Port_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
```

```
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event OUT_Port_Pausing_OK extends PausingInstruction
  any
    resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @filled    portTransferDataFilled(resIndex)=TRUE
end

event OUT_Port_OK extends Write0RegisterInstruction
  any resIdent resIndex portBits
  where
    @resIdType     resIdent∈Bits
    @identOk       IsPortIdent(resIdent)=TRUE
    @inUse         IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @filled        portTransferDataFilled(resIndex)=FALSE
    @portBitsType portBits∈Bits
  then
    @inMode        portInMode(resIndex) ≔  FALSE
    @changeData    portTransferData(resIndex) ≔ portBits
    @changeFilled portTransferDataFilled(resIndex) ≔  TRUE
end

event IN_Port_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event IN_Port_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @filled    portTransferDataFilled(resIndex)=FALSE
end
```

```
event IN_Port_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @filled    portTransferDataFilled(resIndex)=TRUE
    @dBits     dBits=portTransferData(resIndex)
  then
    @inMode       portInMode(resIndex) ≔  TRUE
    @triggered    portTriggered(resIndex) ≔ FALSE
    @changeFilled portTransferDataFilled(resIndex) ≔  FALSE
end

event ENDIN_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event ENDIN_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event ENDIN_NotInMode extends IllegalResourceException
  any resIndex
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @inMode   portInMode(resIndex)=FALSE
end

event ENDIN_NotBufferedMode extends IllegalResourceException
  any resIndex
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @bufMode  portBufferedMode(resIndex)=FALSE
end
```

```
event ENDIN_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @inMode    portInMode(resIndex)=TRUE
    @bufMode   portBufferedMode(resIndex)=TRUE
    @dBits     dBits=NatToBits32(0)
  then
    @setError incomplete ≔ TRUE
end

event INPW_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event INPW_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event INPW_NotBufferedMode extends IllegalResourceException
  any resIndex
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @bufMode  portBufferedMode(resIndex)=FALSE
end

event INPW_UnsupportedWidth extends IllegalResourceException
  any widthOk
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @widthOk  widthOk=FALSE
end

event INPW_OK extends Write1RegisterInstruction
  any resIdent resIndex widthOk
```

```
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @bufMode   portBufferedMode(resIndex)=TRUE
    @widthOk   widthOk=TRUE
    @dBits     dBits=NatToBits32(0)
  then
    @setError  incomplete ≔ TRUE
end

event INSHR_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event INSHR_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event INSHR_OK extends Write1RegisterInstruction
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @dBits     dBits=NatToBits32(0)
  then
    @setError incomplete ≔ TRUE
end

event OUTPW_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event OUTPW_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end
```

```
event OUTPW_NotBufferedMode extends IllegalResourceException
  any resIndex
  where
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @bufMode   portBufferedMode(resIndex)=FALSE
end

event OUTPW_UnsupportedWidth extends IllegalResourceException
  any widthOk
  where
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @widthOk   widthOk=FALSE
end

event OUTPW_OK extends Write0RegisterInstruction
  any resIdent resIndex widthOk
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @bufMode   portBufferedMode(resIndex)=TRUE
    @widthOk   widthOk=TRUE
  then
    @setError incomplete ≔ TRUE
end

event OUTSHR_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event OUTSHR_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event OUTSHR_OK extends Write0RegisterInstruction
  any resIdent
  where
```

```
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
  then
    @setError incomplete ≔ TRUE
end

event PEEK_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event PEEK_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event PEEK_OK extends Write1RegisterInstruction
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @dBits     dBits=NatToBits32(0)
  then
    @setError incomplete ≔ TRUE
end

event SETC_Port_BadCtrl extends IllegalResourceException
  any ctrl
  where
    @identOk   IsPortIdent(resIdent)=TRUE
    @ctrlType  ctrl∈Bits32Nat
    @ctrlCheck IsPortControl(ctrl)=FALSE
end

event SETC_Port_InUseOn_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
```

```
    @ctrlCheck    ctrl=CtrlInUseOn
  then
    @portControl portStatus(resIndex) ≔ RUNNING
end

event SETC_Port_InUseOff_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
    @ctrlCheck    ctrl=CtrlInUseOff
  then
    @portControl portStatus(resIndex) ≔ FREE
end

event SETC_Port_Event_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
    @ctrlCheck    ctrl=CtrlIeModeEvent
  then
    @portControl portEventMode(resIndex) ≔ TRUE
end

event SETC_Port_Interrupt_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
    @ctrlCheck    ctrl=CtrlIeModeInterrupt
  then
    @portControl portEventMode(resIndex) ≔ FALSE
end

event SETC_Port_Other_OK extends Write0RegisterInstruction
```

```
  any resIdent ctrl
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @ctrlType  ctrl∈Bits32Nat
    @ctrlCheck IsPortControl(ctrl)=TRUE
    @InUseOn   ctrl≠CtrlInUseOn
    @InUseOff  ctrl≠CtrlInUseOff
    @IeModeEvt ctrl≠CtrlIeModeEvent
    @IeModeInt ctrl≠CtrlIeModeInterrupt
end

event SETD_Port_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETD_Port_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
  then
    @setError incomplete ≔ TRUE
end

event SETEV_Port_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsPortIdent(resIdent)=TRUE
    @statusCheck IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETEV_Port_OK extends Write0RegisterInstruction
  any resIdent resIndex portBits
  where
    @resIdentType  resIdent∈Bits
    @typeCheck     IsPortIdent(resIdent)=TRUE
    @statusCheck   IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @portBitsType  portBits∈Bits
  then
```

```
    @changeEnvVec portEnvironmentVector(resIndex) ≔  Bits32ToNat(portBits)
end

event SETV_Port_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsPortIdent(resIdent)=TRUE
    @statusCheck IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETV_Port_OK extends Write0RegisterInstruction
  any resIdent resIndex portBits
  where
    @resIdentType  resIdent∈Bits
    @typeCheck     IsPortIdent(resIdent)=TRUE
    @statusCheck   IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @portBitsType  portBits∈Bits
  then
    @changeVect portVector(resIndex) ≔  Bits32ToNat(portBits)
end

event GETD_Port_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event GETD_Port_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdentType resIdent∈Bits
    @identOk      IsPortIdent(resIdent)=TRUE
    @inUse        IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @dBits        dBits=NatToBits32(0)
  then
    @setError incomplete ≔ TRUE
end

event SETPSC_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end
```

```
event SETPSC_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETPSC_NotBufferedMode extends IllegalResourceException
  any resIndex
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @bufMode  portBufferedMode(resIndex)=FALSE
end

event SETPSC_UnsupportedCount extends IllegalResourceException
  any countOk
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @widthOk  countOk=FALSE
end

event SETPSC_OK extends Write0RegisterInstruction
  any resIdent resIndex countOk
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @bufMode   portBufferedMode(resIndex)=TRUE
    @widthOk   countOk=TRUE
  then
    @setError incomplete ≔ TRUE
end

event CLRPT_BadIdent extends IllegalResourceException
  where
    @identOk   IsPortIdent(resIdent)=FALSE
end

event CLRPT_NotInUse extends IllegalResourceException
  where
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=FALSE
```

```
end

event CLRPT_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
  then
    @portTimerTrigValue portTimerTriggerValue(resIndex) ≔  0
end

event SETPT_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event SETPT_NotInUse extends IllegalResourceException
  where
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETPT_OK extends Write0RegisterInstruction
  any resIdent resIndex portBits
  where
    @resIdType    resIdent∈Bits
    @identOk      IsPortIdent(resIdent)=TRUE
    @inUse        IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @portBitsType portBits∈Bits
  then
    @portTimerTrigValue portTimerTriggerValue(resIndex) ≔  Bits16ToNat(portBits)
end

event GETTS_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event GETTS_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
```

```
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event GETTS_OK extends Write1RegisterInstruction
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
  then
    @setError  incomplete ≔ TRUE
end

event SETRDY_Port_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETRDY_Port_OK extends Write0RegisterInstruction
  any resIdent srcIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @srcIdType srcIdent∈Bits
  then
    @setError  incomplete ≔ TRUE
end

event SETTW_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event SETTW_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event SETTW_NotBufferedMode extends IllegalResourceException
  any resIndex
  where
```

```
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @bufMode  portBufferedMode(resIndex)=FALSE
end

event SETTW_UnsupportedWidth extends IllegalResourceException
  any widthOk
  where
    @identOk  IsPortIdent(resIdent)=TRUE
    @inUse    IsPortInUse(resIdent↦portStatus)=TRUE
    @widthOk  widthOk=FALSE
end

event SETTW_OK extends Write0RegisterInstruction
  any resIdent resIndex widthOk
  where
    @resIdType resIdent ∈ Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @bufMode   portBufferedMode(resIndex)=TRUE
    @widthOk   widthOk=TRUE
  then
    @setError  incomplete ≔ TRUE
end

event SYNCR_BadIdent extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=FALSE
end

event SYNCR_NotInUse extends IllegalResourceException
  where
    @identOk IsPortIdent(resIdent)=TRUE
    @inUse   IsPortInUse(resIdent↦portStatus)=FALSE
end

event EDU_Port_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsPortIdent(resIdent)=TRUE
    @statusCheck IsPortInUse(resIdent↦portStatus)=FALSE
end

event EDU_Port_OK extends Write0RegisterInstruction
```

```
  any resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=NatToBits32(NoResourceId)
  then
    @portThreadIdent portThreadIdent(resIndex) ≔  threadId
end

event EEF_Port_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsPortIdent(resIdent)=TRUE
    @statusCheck IsPortInUse(resIdent↦portStatus)=FALSE
end

event EEF_Port_TRUE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c=0
  then
    @changeThread portThreadIdent(resIndex) ≔  threadId
end

event EEF_Port_FALSE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=NatToBits32(NoResourceId)
    @cond         c≠0
  then
    @changeThread portThreadIdent(resIndex) ≔  threadId
end

event EET_Port_NotInUse extends IllegalResourceException
```

```
  where
    @typeCheck   IsPortIdent(resIdent)=TRUE
    @statusCheck IsPortInUse(resIdent↦portStatus)=FALSE
end

event EET_Port_TRUE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c≠0
  then
     @changeThread portThreadIdent(resIndex) ≔  threadId
end

event EET_Port_FALSE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c=0
  then
     @changeThread portThreadIdent(resIndex) ≔  threadId
end

event EEU_Port_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsPortIdent(resIdent)=TRUE
    @statusCheck IsPortInUse(resIdent↦portStatus)=FALSE
end

event EEU_Port_OK extends Write0RegisterInstruction
  any resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsPortIdent(resIdent)=TRUE
    @statusCheck  IsPortInUse(resIdent↦portStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
```

```
    @threadId     threadId=GetThreadIdent(threadIndex)
  then
    @changeThread portThreadIdent(resIndex) :=  threadId
end

event SYNCR_OK extends Write0RegisterInstruction
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsPortIdent(resIdent)=TRUE
    @inUse     IsPortInUse(resIdent↦portStatus)=TRUE
  then
    @setError  incomplete := TRUE
end

event SelectNextPort
  then
    @portIndex portIndex :=  (portIndex + 1) mod MaxPorts
end

event PortEvent extends ResourceEvent
  where
    @portStatus  portStatus(portIndex)=RUNNING
    @portEvent   portEventMode(portIndex)=TRUE
    @valueCheck  portTriggered(portIndex)=TRUE
    @threadIdent portThreadIdent(portIndex)≠NatToBits32(NoResourceId)
    @threadIndex triggeredThreadIndex=GetResourceIndex(portThreadIdent(portIndex))
    @edBits      edBits=NatToBits32(portEnvironmentVector(portIndex))
    @nextPc      nextPc=portVector(portIndex)
end

event PortInterrupt extends Interrupt
  where
    @chanStatus  portStatus(portIndex)=RUNNING
    @portEvent   portEventMode(portIndex)=FALSE
    @valueCheck  portTriggered(portIndex)=TRUE
    @threadIdent portThreadIdent(portIndex)≠NatToBits32(NoResourceId)
    @threadIndex triggeredThreadIndex=GetResourceIndex(portThreadIdent(portIndex))
    @edBits      edBits=NatToBits32(portEnvironmentVector(portIndex))
    @nextPc      nextPc=portVector(portIndex)
end

event TransferPort
  any txPort rxPort
  where
```

```
      @txPortType txPort∈PortIndexDom
      @rxPortType rxPort∈PortIndexDom
        @txFilled   portTransferDataFilled(txPort)=TRUE
        @rxFilled   portTransferDataFilled(rxPort)=FALSE
      then
        @changeData   portTransferData(rxPort) ≔  portTransferData(txPort)
        @changeTrig   portTriggered(rxPort) ≔ TRUE
        @changeFilled portTransferDataFilled ≔ portTransferDataFilled ⩤ {txPort↦FALSE} ⩤ {rxPort↦TRUE}
  end

end
```

14.1.24machine XCoreIsaPortFinal

```
machine XCoreIsaPortFinal

refines XCoreIsaPort1

sees XCoreBasic XCorePorts XCoreClocks

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

          portIndex portStatus
          portEventMode portInMode portBufferedMode
          portTriggered portThreadIdent portEnvironmentVector portVector
          portTransferData portTransferDataFilled portClockIdent
          portTimerValue portTimerTriggerValue portTimestamp
          incomplete

events

  event INITIALISATION extends INITIALISATION end

  event SETCLK_Lr2r_Port_BadIdent extends SETCLK_Port_BadIdent
    any op1 op2
    where
      @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
      @op1Fetch  op1=XLr2r_op1(instr)
      @op2Fetch  op2=XLr2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
      @regBits   clkBits=(threadRegArrays(threadIndex))(op2)
   end

  event SETCLK_Lr2r_Port_NotInUse extends SETCLK_Port_NotInUse
    any op1
    where
      @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
      @op1Fetch  op1=XLr2r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
   end

  event SETCLK_Lr2r_Port_OK extends SETCLK_Port_OK
    any op1 op2
    where
```

```
    @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @regBits   clkBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
 end

event OUT_R2r_Port_NotInUse extends OUT_Port_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUT_R2r_Port_Pausing_OK extends OUT_Port_Pausing_OK
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUT_R2r_Port_OK extends OUT_Port_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @regBits   portBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event IN_2r_Port_NotInUse extends IN_Port_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_Port_Pausing_OK extends IN_Port_Pausing_OK
  any op2
```

```
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_Port_OK extends IN_Port_OK
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event ENDIN_2r_BadIdent extends ENDIN_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦18↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event ENDIN_2r_NotInUse extends ENDIN_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦18↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event ENDIN_2r_NotInMode extends ENDIN_NotInMode
  any op2
  where
    @instCheck Is2r(instr↦18↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event ENDIN_2r_NotBufferedMode extends ENDIN_NotBufferedMode
  any op2
  where
    @instCheck Is2r(instr↦18↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event ENDIN_2r_OK extends ENDIN_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦18↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event INPW_L2rus_BadIdent extends INPW_BadIdent
  any op2
  where
    @instCheck IsL2rus(instr↦18↦14)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INPW_L2rus_NotInUse extends INPW_NotInUse
  any op2
  where
    @instCheck IsL2rus(instr↦18↦14)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INPW_L2rus_NotBufferedMode extends INPW_NotBufferedMode
  any op2
  where
    @instCheck IsL2rus(instr↦18↦14)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INPW_L2rus_UnsupportedWidth extends INPW_UnsupportedWidth
  any op2
  where
    @instCheck IsL2rus(instr↦18↦14)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end
```

```
event INPW_L2rus_OK extends INPW_OK
  any op1 op2 op3
  where
    @instCheck IsL2rus(instr↦18↦14)=TRUE
    @op1Fetch  op1=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @op3Fetch  op3=XL2rus_op3(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event INSHR_2r_BadIdent extends INSHR_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦22↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INSHR_2r_NotInUse extends INSHR_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦22↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event INSHR_2r_OK extends INSHR_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦22↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event OUTPW_L2rus_BadIdent extends OUTPW_BadIdent
  any op2
  where
    @instCheck IsL2rus(instr↦18↦13)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end
```

```
event OUTPW_L2rus_NotInUse extends OUTPW_NotInUse
  any op2
  where
    @instCheck IsL2rus(instr↦18↦13)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event OUTPW_L2rus_NotBufferedMode extends OUTPW_NotBufferedMode
  any op2
  where
    @instCheck IsL2rus(instr↦18↦13)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event OUTPW_L2rus_UnsupportedWidth extends OUTPW_UnsupportedWidth
  any op2
  where
    @instCheck IsL2rus(instr↦18↦13)=TRUE
    @op2Fetch  op2=XL2rus_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event OUTPW_L2rus_OK extends OUTPW_OK
  any op1 op2 op3
  where
    @instCheck IsL2rus(instr↦18↦13)=TRUE
    @op1Fetch  op1=XL2rus_op1(instr)
    @op2Fetch  op2=XL2rus_op2(instr)
    @op3Fetch  op3=XL2rus_op3(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event OUTSHR_R2r_BadIdent extends OUTSHR_BadIdent
  any op1
  where
    @instCheck IsR2r(instr↦21↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTSHR_R2r_NotInUse extends OUTSHR_NotInUse
  any op1
```

```
  where
    @instCheck IsR2r(instr↦21↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event OUTSHR_R2r_OK extends OUTSHR_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦21↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event PEEK_2r_BadIdent extends PEEK_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦23↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event PEEK_2r_NotInUse extends PEEK_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦23↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event PEEK_2r_OK extends PEEK_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦23↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Port_BadCtrl extends SETC_Port_BadCtrl
  any op1
  where
```

```
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Ru6_Port_InUseOn_OK extends SETC_Port_InUseOn_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Port_InUseOff_OK extends SETC_Port_InUseOff_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Port_Event_OK extends SETC_Port_Event_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Port_Interrupt_OK extends SETC_Port_Interrupt_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end
```

```
event SETC_Ru6_Port_Other_OK extends SETC_Port_Other_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Lru6_Port_BadCtrl extends SETC_Port_BadCtrl
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Lru6_Port_InUseOn_OK extends SETC_Port_InUseOn_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Port_InUseOff_OK extends SETC_Port_InUseOff_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Port_Event_OK extends SETC_Port_Event_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
```

```
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Port_Interrupt_OK extends SETC_Port_Interrupt_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Port_Other_OK extends SETC_Port_Other_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Port_BadCtrl extends SETC_Port_BadCtrl
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
end

event SETC_L2r_Port_InUseOn_OK extends SETC_Port_InUseOn_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end
```

```
event SETC_L2r_Port_InUseOff_OK extends SETC_Port_InUseOff_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Port_Event_OK extends SETC_Port_Event_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Port_Interrupt_OK extends SETC_Port_Interrupt_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Port_Other_OK extends SETC_Port_Other_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end
```

```
event SETD_R2r_Port_NotInUse extends SETD_Port_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETD_R2r_Port_OK extends SETD_Port_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETEV_1r_Port_NotInUse extends SETEV_Port_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦7↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETEV_1r_Port_OK extends SETEV_Port_OK
  any op1
  where
    @instCheck Is1r(instr↦7↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @portBits  portBits=(threadRegArrays(threadIndex))(R11Index)
    @instrSize instrSize=InstSizeShort+0
end

event SETV_1r_Port_NotInUse extends SETV_Port_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦8↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETV_1r_Port_OK extends SETV_Port_OK
```

```
  any op1
  where
    @instCheck Is1r(instr↦8↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @portBits  portBits=(threadRegArrays(threadIndex))(R11Index)
    @instrSize instrSize=InstSizeShort+0
end

event GETD_L2r_Port_NotInUse extends GETD_Port_NotInUse
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event GETD_L2r_Port_OK extends GETD_Port_OK
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event SETPSC_R2r_BadIdent extends SETPSC_BadIdent
  any op1
  where
    @instCheck IsR2r(instr↦24↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETPSC_R2r_NotInUse extends SETPSC_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦24↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETPSC_R2r_NotBufferedMode extends SETPSC_NotBufferedMode
  any op1
```

```
  where
    @instCheck IsR2r(instr↦24↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETPSC_R2r_UnsupportedCount extends SETPSC_UnsupportedCount
  any op1
  where
    @instCheck IsR2r(instr↦24↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETPSC_R2r_OK extends SETPSC_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦24↦0)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event CLRPT_1r_BadIdent extends CLRPT_BadIdent
  any op1
  where
    @instCheck Is1r(instr↦16↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CLRPT_1r_NotInUse extends CLRPT_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦16↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event CLRPT_1r_OK extends CLRPT_OK
  any op1
  where
    @instCheck Is1r(instr↦16↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETPT_R2r_BadIdent extends SETPT_BadIdent
  any op1
  where
    @instCheck IsR2r(instr↦7↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETPT_R2r_NotInUse extends SETPT_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦7↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETPT_R2r_OK extends SETPT_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦7↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @portBits  portBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event GETTS_2r_BadIdent extends GETTS_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦7↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event GETTS_2r_NotInUse extends GETTS_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦7↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
```

```
end

event GETTS_2r_OK extends GETTS_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦7↦0)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event SETRDY_Lr2r_Port_NotInUse extends SETRDY_Port_NotInUse
  any op1
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETRDY_Lr2r_Port_OK extends SETRDY_Port_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @srcIdent  srcIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event SETTW_Lr2r_BadIdent extends SETTW_BadIdent
  any op1
  where
    @instCheck IsLr2r(instr↦4↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETTW_Lr2r_NotInUse extends SETTW_NotInUse
  any op1
  where
    @instCheck IsLr2r(instr↦4↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
```

```
end

event SETTW_Lr2r_NotBufferedMode extends SETTW_NotBufferedMode
  any op1
  where
    @instCheck IsLr2r(instr↦4↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETTW_Lr2r_UnsupportedWidth extends SETTW_UnsupportedWidth
  any op1
  where
    @instCheck IsLr2r(instr↦4↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETTW_Lr2r_OK extends SETTW_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦4↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event EDU_1r_Port_NotInUse extends EDU_Port_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦0↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EDU_1r_Port_OK extends EDU_Port_OK
  any op1
  where
    @instCheck Is1r(instr↦0↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end
```

```
event EEF_2r_Port_NotInUse extends EEF_Port_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event EEF_2r_Port_TRUE_OK extends EEF_Port_TRUE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EEF_2r_Port_FALSE_OK extends EEF_Port_FALSE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EET_2r_Port_NotInUse extends EET_Port_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event EET_2r_Port_TRUE_OK extends EET_Port_TRUE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
```

```
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EET_2r_Port_FALSE_OK extends EET_Port_FALSE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EEU_1r_Port_NotInUse extends EEU_Port_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦0↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EEU_1r_Port_OK extends EEU_Port_OK
  any op1
  where
    @instCheck Is1r(instr↦0↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SYNCR_1r_BadIdent extends SYNCR_BadIdent
  any op1
  where
    @instCheck Is1r(instr↦16↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SYNCR_1r_NotInUse extends SYNCR_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦16↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
```

```
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event SYNCR_1r_OK extends SYNCR_OK
    any op1
    where
      @instCheck Is1r(instr↦16↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
      @instrSize instrSize=InstSizeShort+0
  end

  event SelectNextPort extends SelectNextPort end
  event PortEvent      extends PortEvent end
  event PortInterrupt  extends PortInterrupt end
  event TransferPort   extends TransferPort end

end
```

14.1.25machine XCoreIsaPortConnections

```
machine XCoreIsaPortConnections // Specify example port loop-backs
refines XCoreIsaPortFinal
sees XCoreBasic XCorePorts XCoreClocks

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

          portIndex portStatus
          portEventMode portInMode portBufferedMode
          portTriggered portThreadIdent portEnvironmentVector portVector
          portTransferData portTransferDataFilled portClockIdent
          portTimerValue portTimerTriggerValue portTimestamp
          incomplete

events

  event INITIALISATION extends INITIALISATION end
  event SETCLK_Lr2r_Port_BadIdent extends SETCLK_Lr2r_Port_BadIdent end
  event SETCLK_Lr2r_Port_NotInUse extends SETCLK_Lr2r_Port_NotInUse end
  event SETCLK_Lr2r_OK extends SETCLK_Lr2r_Port_OK end
  event OUT_R2r_Port_NotInUse extends OUT_R2r_Port_NotInUse end
  event OUT_R2r_Port_Pausing_OK extends OUT_R2r_Port_Pausing_OK end
  event OUT_R2r_Port_OK extends OUT_R2r_Port_OK end
  event IN_2r_Port_NotInUse extends IN_2r_Port_NotInUse end
  event IN_2r_Port_Pausing_OK extends IN_2r_Port_Pausing_OK end
  event IN_2r_Port_OK extends IN_2r_Port_OK end
  event CLRPT_1r_BadIdent extends CLRPT_1r_BadIdent end
  event CLRPT_1r_NotInUse extends CLRPT_1r_NotInUse end
  event CLRPT_1r_OK extends CLRPT_1r_OK end
  event ENDIN_2r_BadIdent extends ENDIN_2r_BadIdent end
  event ENDIN_2r_NotInUse extends ENDIN_2r_NotInUse end
  event ENDIN_2r_NotInMode extends ENDIN_2r_NotInMode end
  event ENDIN_2r_NotBufferedMode extends ENDIN_2r_NotBufferedMode end
  event ENDIN_2r_OK extends ENDIN_2r_OK end
  event GETTS_2r_BadIdent extends GETTS_2r_BadIdent end
  event GETTS_2r_NotInUse extends GETTS_2r_NotInUse end
  event GETTS_2r_OK extends GETTS_2r_OK end
  event INPW_L2rus_BadIdent extends INPW_L2rus_BadIdent end
  event INPW_L2rus_NotInUse extends INPW_L2rus_NotInUse end
  event INPW_L2rus_NotBufferedMode extends INPW_L2rus_NotBufferedMode end
  event INPW_L2rus_UnsupportedWidth extends INPW_L2rus_UnsupportedWidth end
  event INPW_L2rus_OK extends INPW_L2rus_OK end
  event INSHR_2r_BadIdent extends INSHR_2r_BadIdent end
```

```
event INSHR_2r_NotInUse extends INSHR_2r_NotInUse end
event INSHR_2r_OK extends INSHR_2r_OK end
event OUTPW_L2rus_BadIdent extends OUTPW_L2rus_BadIdent end
event OUTPW_L2rus_NotInUse extends OUTPW_L2rus_NotInUse end
event OUTPW_L2rus_NotBufferedMode extends OUTPW_L2rus_NotBufferedMode end
event OUTPW_L2rus_UnsupportedWidth extends OUTPW_L2rus_UnsupportedWidth end
event OUTPW_L2rus_OK extends OUTPW_L2rus_OK end
event OUTSHR_R2r_BadIdent extends OUTSHR_R2r_BadIdent end
event OUTSHR_R2r_NotInUse extends OUTSHR_R2r_NotInUse end
event OUTSHR_R2r_OK extends OUTSHR_R2r_OK end
event PEEK_2r_BadIdent extends PEEK_2r_BadIdent end
event PEEK_2r_NotInUse extends PEEK_2r_NotInUse end
event PEEK_2r_OK extends PEEK_2r_OK end
event SETPSC_R2r_BadIdent extends SETPSC_R2r_BadIdent end
event SETPSC_R2r_NotInUse extends SETPSC_R2r_NotInUse end
event SETPSC_R2r_NotBufferedMode extends SETPSC_R2r_NotBufferedMode end
event SETPSC_R2r_UnsupportedCount extends SETPSC_R2r_UnsupportedCount end
event SETPSC_R2r_OK extends SETPSC_R2r_OK end
event SETPT_R2r_BadIdent extends SETPT_R2r_BadIdent end
event SETPT_R2r_NotInUse extends SETPT_R2r_NotInUse end
event SETPT_R2r_OK extends SETPT_R2r_OK end
event SETRDY_Lr2r_Port_NotInUse extends SETRDY_Lr2r_Port_NotInUse end
event SETRDY_Lr2r_Port_OK extends SETRDY_Lr2r_Port_OK end
event SETTW_Lr2r_BadIdent extends SETTW_Lr2r_BadIdent end
event SETTW_Lr2r_NotInUse extends SETTW_Lr2r_NotInUse end
event SETTW_Lr2r_NotBufferedMode extends SETTW_Lr2r_NotBufferedMode end
event SETTW_Lr2r_UnsupportedWidth extends SETTW_Lr2r_UnsupportedWidth end
event SETTW_Lr2r_OK extends SETTW_Lr2r_OK end
event SYNCR_1r_BadIdent extends SYNCR_1r_BadIdent end
event SYNCR_1r_NotInUse extends SYNCR_1r_NotInUse end
event SYNCR_1r_OK extends SYNCR_1r_OK end
event SETC_Ru6_Port_BadCtrl extends SETC_Ru6_Port_BadCtrl end
event SETC_Ru6_Port_InUseOn_OK extends SETC_Ru6_Port_InUseOn_OK end
event SETC_Ru6_Port_InUseOff_OK extends SETC_Ru6_Port_InUseOff_OK end
event SETC_Ru6_Port_Event_OK extends SETC_Ru6_Port_Event_OK end
event SETC_Ru6_Port_Interrupt_OK extends SETC_Ru6_Port_Interrupt_OK end
event SETC_Ru6_Port_Other_OK extends SETC_Ru6_Port_Other_OK end
event SETC_Lru6_Port_BadCtrl extends SETC_Lru6_Port_BadCtrl end
event SETC_Lru6_Port_InUseOn_OK extends SETC_Lru6_Port_InUseOn_OK end
event SETC_Lru6_Port_InUseOff_OK extends SETC_Lru6_Port_InUseOff_OK end
event SETC_Lru6_Port_Event_OK extends SETC_Lru6_Port_Event_OK end
event SETC_Lru6_Port_Interrupt_OK extends SETC_Lru6_Port_Interrupt_OK end
event SETC_Lru6_Port_Other_OK extends SETC_Lru6_Port_Other_OK end
event SETC_L2r_Port_BadCtrl extends SETC_L2r_Port_BadCtrl end
event SETC_L2r_Port_InUseOn_OK extends SETC_L2r_Port_InUseOn_OK end
```

```
  event SETC_L2r_Port_InUseOff_OK extends SETC_L2r_Port_InUseOff_OK end
  event SETC_L2r_Port_Event_OK extends SETC_L2r_Port_Event_OK end
  event SETC_L2r_Port_Interrupt_OK extends SETC_L2r_Port_Interrupt_OK end
  event SETC_L2r_Port_Other_OK extends SETC_L2r_Port_Other_OK end
  event SETD_R2r_Port_NotInUse extends SETD_R2r_Port_NotInUse end
  event SETD_R2r_Port_OK extends SETD_R2r_Port_OK end
  event SETEV_1r_Port_NotInUse extends SETEV_1r_Port_NotInUse end
  event SETEV_1r_Port_OK extends SETEV_1r_Port_OK end
  event SETV_1r_Port_NotInUse extends SETV_1r_Port_NotInUse end
  event SETV_1r_Port_OK extends SETV_1r_Port_OK end
  event GETD_L2r_Port_NotInUse extends GETD_L2r_Port_NotInUse end
  event GETD_L2r_Port_OK extends GETD_L2r_Port_OK end
  event EDU_1r_Port_NotInUse extends EDU_1r_Port_NotInUse end
  event EDU_1r_Port_OK       extends EDU_1r_Port_OK end
  event EEF_2r_Port_NotInUse extends EEF_2r_Port_NotInUse end
  event EEF_2r_Port_TRUE_OK  extends EEF_2r_Port_TRUE_OK end
  event EEF_2r_Port_FALSE_OK extends EEF_2r_Port_FALSE_OK end
  event EET_2r_Port_NotInUse extends EET_2r_Port_NotInUse end
  event EET_2r_Port_TRUE_OK  extends EET_2r_Port_TRUE_OK end
  event EET_2r_Port_FALSE_OK extends EET_2r_Port_FALSE_OK end
  event EEU_1r_Port_NotInUse extends EEU_1r_Port_NotInUse end
  event EEU_1r_Port_OK       extends EEU_1r_Port_OK end
  event SelectNextPort extends SelectNextPort end
  event PortEvent      extends PortEvent end
  event PortInterrupt  extends PortInterrupt end

  event TxPort0to1 extends TransferPort
    where
        @txPort txPort=0+0
        @rxPort rxPort=1+0
  end

  event TxPort2to3 extends TransferPort
    where
        @txPort txPort=2+0
        @rxPort rxPort=3+0
  end

end
```

14.1.26machine XCoreIsaProcessorState1

```
machine XCoreIsaProcessorState1
refines XCoreFinal
sees XCoreBasic XCoreProcessorState

variables status interruptTriggered resourceTriggered
          procState memArray
          threadIndex threadStatus threadRegArrays threadPc

invariants
  @procStateType procState ∈ ProcStateDom→Bits

events

  event INITIALISATION extends INITIALISATION
    then
      @procState procState  ≔ {PsRamBaseIndex↦NatToBits32(MinMemIndex)} ⩤ {PsDbgTypeIndex↦NatToBits32(0)}
  end

  event Loading                           extends Loading                           where @false ⊥ end
  event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
  event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
  event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
  event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
  event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
  event LinkErrorException                extends LinkErrorException                where @false ⊥ end
  event IllegalPcException                extends IllegalPcException                where @false ⊥ end
  event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
  event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
  event LoadStoreException                extends LoadStoreException                where @false ⊥ end
  event IllegalPsException                extends IllegalPsException                where @false ⊥ end
  event ArithmeticException               extends ArithmeticException               where @false ⊥ end
  event EcallException                    extends EcallException                    where @false ⊥ end
  event ResourceDepException              extends ResourceDepException              where @false ⊥ end
  event KcallException                    extends KcallException                    where @false ⊥ end
  event Interrupt                         extends Interrupt                         where @false ⊥ end
  event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
  event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
```

```
event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event GETPS_BadType extends IllegalPsException
  where
    @identChk  IsProcessorStateIdent(ident)=FALSE
end

event GETPS_BadIndex extends IllegalPsException
  any index
  where
    @debugMode status≠DEBUGGING
    @identChk  IsProcessorStateIdent(ident)=TRUE
    @idxFetch  index=GetProcessorStateIndex(ident)
    @indexOk   IsRunningReadProcessorStateIndex(index)=FALSE
end

event GETPS_OK extends Write1RegisterInstruction
  any ident index
  where
    @debugMode status≠DEBUGGING
    @identType ident∈Bits
    @identChk  IsProcessorStateIdent(ident)=TRUE
    @idxFetch  index=GetProcessorStateIndex(ident)
    @indexOk   IsRunningReadProcessorStateIndex(index)=TRUE
    @dBits     dBits=procState(index)
end

event GETPS_Debug_BadIndex extends IllegalPsException
  any index
  where
    @debugMode status=DEBUGGING
```

```
    @identChk  IsProcessorStateIdent(ident)=TRUE
    @idxFetch  index=GetProcessorStateIndex(ident)
    @indexOk   IsDebuggingReadProcessorStateIndex(index)=FALSE
end

event GETPS_Debug_OK extends Write1RegisterInstruction
  any ident index
  where
    @debugMode status=DEBUGGING
    @identType ident∈Bits
    @identChk  IsProcessorStateIdent(ident)=TRUE
    @idxFetch  index=GetProcessorStateIndex(ident)
    @indexOk   IsDebuggingReadProcessorStateIndex(index)=TRUE
    @dBits     dBits=procState(index)
end

event SETPS_BadType extends IllegalPsException
  where
    @identChk  IsProcessorStateIdent(ident)=FALSE
end

event SETPS_BadWrite extends IllegalPsException
  any index procBits badWrite
  where
    @identChk      IsProcessorStateIdent(ident)=TRUE
    @idxFetch      index=GetProcessorStateIndex(ident)
    @procBitsType  procBits∈Bits
    @badWrite      badWrite=bool(Bits32ToNat(procBits)=MinMemIndex ∧ index=PsRamBaseIndex)
    @badWriteChk   badWrite=TRUE
end

event SETPS_BadIndex extends IllegalPsException
  any index
  where
    @debugMode status≠DEBUGGING
    @identChk  IsProcessorStateIdent(ident)=TRUE
    @idxFetch  index=GetProcessorStateIndex(ident)
    @indexOk   IsRunningWriteProcessorStateIndex(index)=FALSE
end

event SETPS_OK extends Write0RegisterInstruction
  any ident index procBits badWrite
  where
    @debugMode    status≠DEBUGGING
    @identType    ident∈Bits
```

```
    @idxFetch     index=GetProcessorStateIndex(ident)
    @indexOk      IsRunningWriteProcessorStateIndex(index)=TRUE
    @procBitsType procBits∈Bits
    @badWrite     badWrite=bool(Bits32ToNat(procBits)=MinMemIndex ∧ index=PsRamBaseIndex)
    @badWriteChk  badWrite=FALSE
  then
    @changePs procState(index) :=  procBits
end

event SETPS_Debug_BadIndex extends IllegalPsException
  any index
  where
    @debugMode status≠DEBUGGING
    @identChk  IsProcessorStateIdent(ident)=TRUE
    @idxFetch  index=GetProcessorStateIndex(ident)
    @indexOk   IsDebuggingWriteProcessorStateIndex(index)=FALSE
end

event SETPS_Debug_OK extends Write0RegisterInstruction
  any ident index procBits badWrite
  where
    @debugMode    status≠DEBUGGING
    @identType    ident∈Bits
    @idxFetch     index=GetProcessorStateIndex(ident)
    @indexOk      IsDebuggingWriteProcessorStateIndex(index)=TRUE
    @procBitsType procBits∈Bits
    @badWrite     badWrite=bool(Bits32ToNat(procBits)=MinMemIndex ∧ index=PsRamBaseIndex)
    @badWriteChk  badWrite=FALSE
  then
    @changePs procState(index) :=  procBits
end

event DCALL_Running_OK extends DcallInstruction
  any sr sink fast nextDspc nextDssr
  where
    @checkStatus status=RUNNING
    @index       index=0+0
    @nextDspc    nextDspc=NatToBits32(incrPc)
    @nextDssr    nextDssr=(threadRegArrays(threadIndex))(SrIndex)
    @sr          sr=(threadRegArrays(index))(SrIndex)
    @sink        sink=ExtractBits1_5(sr)
    @fast        fast=ExtractBits1_7(sr)
    @dBits       dBits=EightBits1ToBits32(BitClr↦BitClr↦BitClr↦BitSet↦BitSet↦sink↦BitClr↦fast)
    @d           d=SrIndex+0
```

```
    @nextPc       nextThreadPc=Bits32ToNat(procState(PsDbgEntryIndex))
  then
    @changeProc   procState  := procState ⩤ {PsDbgSpcIndex↦nextDspc} ⩤ {PsDbgSsrIndex↦nextDssr} ⩤ {PsDbgTypeIndex↦NatToBits32(DcallCause)}
end

event DCALL_Debugging_OK extends Write0RegisterInstruction
  where
    @checkStatus status=DEBUGGING
    @checkThd0   threadIndex=0
end

event DRET_NoDebugMode extends IllegalInstructionException
  where
    @checkDebugging status≠DEBUGGING
end

event DRET_OK extends DretInstruction
  where
    @checkStatus   status=DEBUGGING
    @checkThread0  threadIndex=0
    @srBits        srBits=procState(PsDbgSsrIndex)
    @nextPcTypeDef nextPc=Bits32ToNat(procState(PsDbgSpcIndex))
end

event DENTSP_NoDebugMode extends IllegalInstructionException
  where
    @checkDebugging status≠DEBUGGING
end

event DENTSP_OK extends Write1RegisterInstruction
  any procBits
  where
    @checkStatus status=DEBUGGING
    @checkThrd0  threadIndex=0
    @d           d=SpIndex+0
    @dBits       dBits=NatToBits32(MaxWordIndex)
    @procBits    procBits=(threadRegArrays(threadIndex))(SpIndex)
  then
    @changeProc  procState  := procState ⩤ {PsDbgSpcIndex↦procBits}
end

event DRESTSP_NoDebugMode extends IllegalInstructionException
  where
    @checkDebugging status≠DEBUGGING
end
```

```
  event DRESTSP_OK extends Write1RegisterInstruction
    where
      @checkStatus status=DEBUGGING
      @checkThrd0  threadIndex=0
      @d           d=SpIndex+0
      @dBits       dBits=procState(PsDbgSspIndex)
  end

  event DGETREG_NoDebugMode extends IllegalInstructionException
    where
      @checkDebugging status≠DEBUGGING
  end

  event DGETREG_BadThreadIndex extends IllegalPsException
    any dThreadIndex
    where
      @checkDebug  status=DEBUGGING
      @checkThrd0  threadIndex=0
      @threadIndex dThreadIndex=GetResourceIndex(procState(PsDbgTnumIndex))
      @checkThread dThreadIndex ∈ ThreadIndexDom
  end

  event DGETREG_BadRegisterIndex extends IllegalPsException
    any dRegIndex
    where
      @checkDebug  status=DEBUGGING
      @checkThrd0  threadIndex=0
      @regIndex    dRegIndex=Bits32ToNat(procState(PsDbgTregIndex))
      @checkReg    dRegIndex ∈ RegArrayDom
  end

  event DGETREG_OK extends Write1RegisterInstruction
    any dThreadIndex dRegIndex
    where
      @checkStatus status=DEBUGGING
      @checkThrd0  threadIndex=0
      @threadIndex dThreadIndex=GetResourceIndex(procState(PsDbgTnumIndex))
      @checkThread dThreadIndex ∈ ThreadIndexDom
      @regIndex    dRegIndex=Bits32ToNat(procState(PsDbgTregIndex))
      @checkReg    dRegIndex ∈ RegArrayDom
      @dBits       dBits=(threadRegArrays(dThreadIndex))(dRegIndex)
  end
end
```

14.1.27machine XCoreIsaProcessorStateFinal

```
machine XCoreIsaProcessorStateFinal

refines XCoreIsaProcessorState1

sees XCoreBasic XCoreProcessorState

variables status interruptTriggered resourceTriggered
          procState memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event GETPS_L2r_BadType extends GETPS_BadType
    any op2
    where
      @instCheck IsL2r(instr↦2↦1↦12)=TRUE
      @op2Fetch  op2=XL2r_op2(instr)
      @regBirs   ident=(threadRegArrays(threadIndex))(op2)
  end

  event GETPS_L2r_BadIndex extends GETPS_BadIndex
    any op2
    where
      @instCheck IsL2r(instr↦2↦1↦12)=TRUE
      @op2Fetch  op2=XL2r_op2(instr)
      @regBirs   ident=(threadRegArrays(threadIndex))(op2)
  end

  event GETPS_L2r_OK extends GETPS_OK
    any op2
    where
      @instCheck IsL2r(instr↦2↦1↦12)=TRUE
      @op1Fetch  d=XL2r_op1(instr)
      @op2Fetch  op2=XL2r_op2(instr)
      @regBirs   ident=(threadRegArrays(threadIndex))(op2)
      @instrSize instrSize=InstSizeLong+0
  end

  event GETPS_Debug_L2r_BadIndex extends GETPS_Debug_BadIndex
    any op2
    where
```

```
    @instCheck IsL2r(instr↦2↦1↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
    @regBirs   ident=(threadRegArrays(threadIndex))(op2)
end

event GETPS_Debug_L2r_OK extends GETPS_Debug_OK
  any op2
  where
    @instCheck IsL2r(instr↦2↦1↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @regBirs   ident=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event SETPS_Lr2r_BadType extends SETPS_BadType
  any op1
  where
    @instCheck IsLr2r(instr↦3↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @ident     ident=(threadRegArrays(threadIndex))(op1)
end

event SETPS_Lr2r_BadWrite extends SETPS_BadWrite
  any op1 op2
  where
    @instCheck IsLr2r(instr↦3↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @ident     ident=(threadRegArrays(threadIndex))(op1)
    @procBits  procBits=(threadRegArrays(threadIndex))(op2)
end

event SETPS_Lr2r_BadIndex extends SETPS_BadIndex
  any op1
  where
    @instCheck IsLr2r(instr↦3↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @ident     ident=(threadRegArrays(threadIndex))(op1)
end

event SETPS_Lr2r_OK extends SETPS_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦3↦0↦12)=TRUE
```

```
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @ident     ident=(threadRegArrays(threadIndex))(op1)
    @procBits  procBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event SETPS_Debug_Lr2r_BadIndex extends SETPS_Debug_BadIndex
  any op1
  where
    @instCheck IsLr2r(instr↦3↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @ident     ident=(threadRegArrays(threadIndex))(op1)
end

event SETPS_Debug_Lr2r_OK extends SETPS_Debug_OK
  any op1 op2
  where
    @instCheck IsLr2r(instr↦3↦0↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @op2Fetch  op2=XLr2r_op2(instr)
    @ident     ident=(threadRegArrays(threadIndex))(op1)
    @procBits  procBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end

event DCALL_0r_Running_OK extends DCALL_Running_OK
  where
    @instCheck Is0r(instr↦0↦1↦12)=TRUE
    @instrSize instrSize=InstSizeShort+0
end

event DCALL_0r_Debugging_OK extends DCALL_Debugging_OK
  where
    @instCheck Is0r(instr↦0↦1↦12)=TRUE
    @instrSize instrSize=InstSizeShort+0
end

event DRET_0r_NoDebugMode extends DRET_NoDebugMode
  where
    @instCheck Is0r(instr↦0↦1↦14)=TRUE
end

event DRET_0r_OK extends DRET_OK
  where
```

```
    @instCheck Is0r(instr↦0↦1↦14)=TRUE
end

event DENTSP_0r_NoDebugMode extends DENTSP_NoDebugMode
  where
    @instCheck Is0r(instr↦2↦0↦12)=TRUE
end

event DENTSP_0r_OK extends DENTSP_OK
  where
    @instCheck Is0r(instr↦2↦0↦12)=TRUE
    @instrSize instrSize=InstSizeShort+0
end

event DRESTSP_0r_NoDebugMode extends DRESTSP_NoDebugMode
  where
    @instCheck Is0r(instr↦2↦0↦13)=TRUE
end

event DRESTSP_0r_OK extends DRESTSP_OK
  where
    @instCheck Is0r(instr↦2↦0↦13)=TRUE
    @instrSize instrSize=InstSizeShort+0
end

event DGETREG_1r_NoDebugMode extends DGETREG_NoDebugMode
  where
    @instCheck Is1r(instr↦7↦0)=TRUE
end

event DGETREG_1r_BadThreadIndex extends DGETREG_BadThreadIndex
  where
    @instCheck Is1r(instr↦7↦0)=TRUE
end

event DGETREG_1r_BadRegisterIndex extends DGETREG_BadRegisterIndex
  where
    @instCheck Is1r(instr↦7↦0)=TRUE
end

event DGETREG_1r_OK extends DGETREG_OK
  where
    @instCheck Is1r(instr↦7↦0)=TRUE
    @op1Fetch  d=X1r_op1(instr)
```

```
      @instrSize instrSize=InstSizeShort+0
  end

end
```

14.1.28machine XCoreIsaRegisterSet1

```
machine XCoreIsaRegisterSet1 // Register-set instructions functionality

refines XCoreFinal

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event Loading                           extends Loading                           where @false ⊥ end
  event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
  event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
  event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
  event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
  event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
  event LinkErrorException                extends LinkErrorException                where @false ⊥ end
  event IllegalPcException                extends IllegalPcException                where @false ⊥ end
  event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
  event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
  event LoadStoreException                extends LoadStoreException                where @false ⊥ end
  event IllegalPsException                extends IllegalPsException                where @false ⊥ end
  event ArithmeticException               extends ArithmeticException               where @false ⊥ end
  event EcallException                    extends EcallException                    where @false ⊥ end
  event ResourceDepException              extends ResourceDepException              where @false ⊥ end
  event KcallException                    extends KcallException                    where @false ⊥ end
  event Interrupt                         extends Interrupt                         where @false ⊥ end
  event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
  event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
  event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
  event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
  event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
  event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
  event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
  event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
  event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
  event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  where @false ⊥ end
```

```
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event CLRSR_OK extends Write1RegisterInstruction
  any mask
  where
    @maskType   mask∈Bits
    @dBits      dBits=AndNotBits((threadRegArrays(threadIndex))(SrIndex)↦mask)
    @d          d=SrIndex+0
end

event SETSR_OK extends Write1RegisterInstruction
  any mask
  where
    @maskType mask∈Bits
    @dBits    dBits=OorBits((threadRegArrays(threadIndex))(SrIndex)↦mask)
    @d        d=SrIndex+0
end

event GETED_OK extends Write1RegisterInstruction
  where
    @dBits dBits=(threadRegArrays(threadIndex))(EdIndex)
end

event GETET_OK extends Write1RegisterInstruction
  where
    @dBits dBits=(threadRegArrays(threadIndex))(EtIndex)
end

event GETKEP_OK extends Write1RegisterInstruction
  where
    @dBits dBits=(threadRegArrays(threadIndex))(KepIndex)
end

event GETKSP_OK extends Write1RegisterInstruction
  where
```

```
    @dBits dBits=(threadRegArrays(threadIndex))(KspIndex)
end

event GETSR_OK extends Write1RegisterInstruction
  any mask
  where
    @maskType mask∈Bits
    @dBits    dBits=AndBits((threadRegArrays(threadIndex))(SrIndex)↦mask)
end

event SETCP_OK extends Write1RegisterInstruction
  where
    @d d=CpIndex+0
end

event SETDP_OK extends Write1RegisterInstruction
  where
    @d d=DpIndex+0
end

event SETKEP_OK extends Write1RegisterInstruction
  where
    @d d=KepIndex+0
end

event SETSP_OK extends Write1RegisterInstruction
  where
    @d d=SpIndex+0
end

event TINITCP_BadType extends IllegalResourceException
  where
    @identOk IsThreadIdent(resIdent)=FALSE
end

event TINITCP_NotSyncing extends IllegalResourceException
  any resIndex
  where
    @identOk  IsThreadIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @resFree  threadStatus(resIndex)≠SPAUSING
end

event TINITCP_OK extends WriteThreadRegisterInstruction
  any resIdent
```

```
  where
    @resIdType resIdent∈Bits
    @identOk   IsThreadIdent(resIdent)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @resFree   threadStatus(resIndex)=SPAUSING
    @d         d=CpIndex+0
end

event TINITDP_BadType extends IllegalResourceException
  where
    @identOk IsThreadIdent(resIdent)=FALSE
end

event TINITDP_NotSyncing extends IllegalResourceException
  any resIndex
  where
    @identOk  IsThreadIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @resFree  threadStatus(resIndex)≠SPAUSING
end

event TINITDP_OK extends WriteThreadRegisterInstruction
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsThreadIdent(resIdent)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @resFree   threadStatus(resIndex)=SPAUSING
    @d         d=DpIndex+0
end

event TINITLR_BadType extends IllegalResourceException
  where
    @identOk IsThreadIdent(resIdent)=FALSE
end

event TINITLR_NotSyncing extends IllegalResourceException
  any resIndex
  where
    @identOk  IsThreadIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @resFree  threadStatus(resIndex)≠SPAUSING
end

event TINITLR_OK extends WriteThreadRegisterInstruction
```

```
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsThreadIdent(resIdent)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @resFree   threadStatus(resIndex)=SPAUSING
    @d         d=LrIndex+0
end

event TINITSP_BadType extends IllegalResourceException
  where
    @identOk IsThreadIdent(resIdent)=FALSE
end

event TINITSP_NotSyncing extends IllegalResourceException
  any resIndex
  where
    @identOk  IsThreadIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @resFree  threadStatus(resIndex)≠SPAUSING
end

event TINITSP_OK extends WriteThreadRegisterInstruction
  any resIdent
  where
    @resIdType resIdent∈Bits
    @identOk   IsThreadIdent(resIdent)=TRUE
    @resIndex  resIndex=GetResourceIndex(resIdent)
    @resFree   threadStatus(resIndex)=SPAUSING
    @d         d=SpIndex+0
end

event TINITPC_BadType extends IllegalResourceException
  where
    @identOk IsThreadIdent(resIdent)=FALSE
end

event TINITPC_NotSyncing extends IllegalResourceException
  any resIndex
  where
    @identOk  IsThreadIdent(resIdent)=TRUE
    @resIndex resIndex=GetResourceIndex(resIdent)
    @resFree  threadStatus(resIndex)≠SPAUSING
end
```

```
  event TINITPC_OK extends WriteThreadPcInstruction
    any pcBits
    where
      @resFree      threadStatus(resIndex)=SPAUSING
      @pcBitsType   pcBits∈Bits
      @nextThreadPc nextThreadPc=Bits32ToNat(pcBits)
  end

  event TSETR_BadType extends IllegalResourceException
    where
      @identOk IsThreadIdent(resIdent)=FALSE
  end

  event TSETR_NotInUse extends IllegalResourceException
    any resIndex
    where
      @identOk  IsThreadIdent(resIdent)=TRUE
      @resIndex resIndex=GetResourceIndex(resIdent)
      @resFree  threadStatus(resIndex)=FREE
  end

  event TSETR_OK extends WriteThreadRegisterInstruction
    any resIdent
    where
      @resIdType resIdent∈Bits
      @identOk   IsThreadIdent(resIdent)=TRUE
      @resIndex  resIndex=GetResourceIndex(resIdent)
      @resFree   threadStatus(resIndex)≠FREE
      @dRange         d∈RegArrayDom
  end

end
```

14.1.29machine XCoreIsaRegisterSetFinal

```
machine XCoreIsaRegisterSetFinal // Register-set instructions instruction encoding

refines XCoreIsaRegisterSet1

sees XCoreBasic

variables status interruptTriggered resourceTriggered
          memArray
          threadIndex threadStatus threadRegArrays threadPc

events

  event INITIALISATION extends INITIALISATION end

  event CLRSR_U6_OK extends CLRSR_OK
    any op1
    where
      @instCheck  IsU6(instr↦15↦0↦12)=TRUE
      @op1Fetch   op1=XU6_op1(instr)
      @mask       mask=NatToBits32(op1)
      @instrSize  instrSize=InstSizeShort+0
  end

  event CLRSR_Lu6_OK extends CLRSR_OK
    any op1
    where
      @instCheck IsLu6(instr↦15↦0↦12)=TRUE
      @op1Fetch  op1=XLu6_op1(instr)
      @mask      mask=NatToBits32(op1)
      @instrSize instrSize=InstSizeLong+0
  end

  event SETSR_U6_OK extends SETSR_OK
    any op1
    where
      @instCheck IsU6(instr↦15↦0↦13)=TRUE
      @op1Fetch  op1=XU6_op1(instr)
      @mask      mask=NatToBits32(op1)
      @instrSize instrSize=InstSizeShort+0
  end

  event SETSR_Lu6_OK extends SETSR_OK
    any op1
```

```
  where
    @instCheck IsLu6(instr↦15↦0↦13)=TRUE
    @op1Fetch  op1=XLu6_op1(instr)
    @mask      mask=NatToBits32(op1)
    @instrSize instrSize=InstSizeLong+0
end

event GETED_0r_OK extends GETED_OK
  where
    @instCheck Is0r(instr↦1↦1↦14)=TRUE
    @instrSize instrSize=InstSizeShort+0
    @d         d=R11Index+0
end

event GETET_0r_OK extends GETET_OK
  where
    @instCheck Is0r(instr↦1↦1↦15)=TRUE
    @instrSize instrSize=InstSizeShort+0
    @d         d=R11Index+0
end

event GETKEP_0r_OK extends GETKEP_OK
  where
    @instCheck Is0r(instr↦2↦0↦15)=TRUE
    @instrSize instrSize=InstSizeShort+0
    @d         d=R11Index+0
end

event GETKSP_0r_OK extends GETKSP_OK
  where
    @instCheck Is0r(instr↦2↦1↦12)=TRUE
    @instrSize instrSize=InstSizeShort+0
    @d         d=R11Index+0
end

event GETSR_U6_OK extends GETSR_OK
  any op1
  where
    @instCheck IsU6(instr↦15↦1↦12)=TRUE
    @op1Fetch  op1=XU6_op1(instr)
    @mask      mask=NatToBits32(op1)
    @instrSize instrSize=InstSizeShort+0
end

event GETSR_Lu6_OK extends GETSR_OK
```

```
  any op1
  where
    @instCheck IsLu6(instr↦15↦1↦12)=TRUE
    @op1Fetch  op1=XLu6_op1(instr)
    @mask      mask=NatToBits32(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETCP_1r_OK extends SETCP_OK
  any s
  where
    @instCheck Is1r(instr↦6↦1)=TRUE
    @op1Fetch  s=X1r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
    @dBits     dBits=(threadRegArrays(threadIndex))(s)
end

event SETDP_1r_OK extends SETDP_OK
  any s
  where
    @instCheck Is1r(instr↦6↦0)=TRUE
    @op1Fetch  s=X1r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
    @dBits     dBits=(threadRegArrays(threadIndex))(s)
end

event SETKEP_0r_OK extends SETKEP_OK
  where
    @instCheck Is0r(instr↦0↦1↦15)=TRUE
    @instrSize instrSize=InstSizeShort+0
    @dBits     dBits=(threadRegArrays(threadIndex))(R11Index)
end

event SETSP_1r_OK extends SETSP_OK
  any s
  where
    @instCheck Is1r(instr↦5↦1)=TRUE
    @op1Fetch  s=X1r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
    @dBits     dBits=(threadRegArrays(threadIndex))(s)
end

event TINITCP_2r_BadType extends TINITCP_BadType
  any op2
  where
```

```
    @instCheck  Is2r(instr↦3↦0)=TRUE
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITCP_2r_NotSyncing extends TINITCP_NotSyncing
  any op2
  where
    @instCheck  Is2r(instr↦3↦0)=TRUE
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITCP_2r_OK extends TINITCP_OK
  any s op2
  where
    @instCheck  Is2r(instr↦3↦0)=TRUE
    @op1Fetch   s=X2r_op1(instr)
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
    @dBits      dBits=(threadRegArrays(threadIndex))(s)
    @instrSize  instrSize=InstSizeShort+0
end

event TINITDP_2r_BadType extends TINITDP_BadType
  any op2
  where
    @instCheck  Is2r(instr↦1↦0)=TRUE
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITDP_2r_NotSyncing extends TINITDP_NotSyncing
  any op2
  where
    @instCheck  Is2r(instr↦1↦0)=TRUE
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITDP_2r_OK extends TINITDP_OK
  any s op2
  where
    @instCheck  Is2r(instr↦1↦0)=TRUE
    @op1Fetch   s=X2r_op1(instr)
```

```
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
    @dBits      dBits=(threadRegArrays(threadIndex))(s)
    @instrSize  instrSize=InstSizeShort+0
end

event TINITLR_L2r_BadType extends TINITLR_BadType
  any op2
  where
    @instCheck  IsL2r(instr↦2↦0↦12)=TRUE
    @op2Fetch   op2=XL2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITLR_L2r_NotSyncing extends TINITLR_NotSyncing
  any op2
  where
    @instCheck  IsL2r(instr↦2↦0↦12)=TRUE
    @op2Fetch   op2=XL2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITLR_L2r_OK extends TINITLR_OK
  any s op2
  where
    @instCheck  IsL2r(instr↦2↦0↦12)=TRUE
    @op1Fetch   s=XL2r_op1(instr)
    @op2Fetch   op2=XL2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
    @dBits      dBits=(threadRegArrays(threadIndex))(s)
    @instrSize  instrSize=InstSizeLong+0
end

event TINITSP_2r_BadType extends TINITSP_BadType
  any op2
  where
    @instCheck  Is2r(instr↦2↦0)=TRUE
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITSP_2r_NotSyncing extends TINITSP_NotSyncing
  any op2
  where
    @instCheck  Is2r(instr↦2↦0)=TRUE
```

```
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITSP_2r_OK extends TINITSP_OK
  any s op2
  where
    @instCheck  Is2r(instr↦2↦0)=TRUE
    @op1Fetch   s=X2r_op1(instr)
    @op2Fetch   op2=X2r_op2(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op2)
    @dBits      dBits=(threadRegArrays(threadIndex))(s)
    @instrSize  instrSize=InstSizeShort+0
end

event TINITPC_2r_BadType extends TINITPC_BadType
  any op2
  where
    @instCheck Is2r(instr↦0↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITPC_2r_NotSyncing extends TINITPC_NotSyncing
  any op2
  where
    @instCheck Is2r(instr↦0↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event TINITPC_2r_OK extends TINITPC_OK
  any s op2
  where
    @instCheck Is2r(instr↦0↦0)=TRUE
    @op1Fetch  s=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @pcBits    pcBits=(threadRegArrays(threadIndex))(s)
    @instrSize instrSize=InstSizeShort+0
end

event TSETR_3r_BadType extends TSETR_BadType
  any op3
  where
```

```
      @instCheck Is3r(instr↦23)=TRUE
      @op3Fetch  op3=X3r_op3(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op3)
  end

  event TSETR_3r_NotInUse extends TSETR_NotInUse
    any op3
    where
      @instCheck Is3r(instr↦23)=TRUE
      @op3Fetch  op3=X3r_op3(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op3)
  end

  event TSETR_3r_OK extends TSETR_OK
    any s op3
    where
      @instCheck Is3r(instr↦23)=TRUE
      @op1Fetch  d=X3r_op1(instr)
      @op2Fetch  s=X3r_op2(instr)
      @op3Fetch  op3=X3r_op3(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op3)
      @dBits     dBits=(threadRegArrays(threadIndex))(s)
      @instrSize instrSize=InstSizeShort+0
  end

end
```

14.1.30machine XCoreIsaResource1

```
machine XCoreIsaResource1 // Instruction events common to any resource

refines XCoreFinal

sees XCoreBasic XCoreResourceClassifications

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events

  event INITIALISATION extends INITIALISATION end

  event Loading                          extends Loading                          where @false ⊥ end
  event RunExternalInputs                extends RunExternalInputs                where @false ⊥ end
  event SelectNextThread                 extends SelectNextThread                 where @false ⊥ end
  event ThreadWaiting                    extends ThreadWaiting                    where @false ⊥ end
  event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
  event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
  event LinkErrorException               extends LinkErrorException               where @false ⊥ end
  event IllegalPcException               extends IllegalPcException               where @false ⊥ end
  event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
  event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
  event LoadStoreException               extends LoadStoreException               where @false ⊥ end
  event IllegalPsException               extends IllegalPsException               where @false ⊥ end
  event ArithmeticException              extends ArithmeticException              where @false ⊥ end
  event EcallException                   extends EcallException                   where @false ⊥ end
  event ResourceDepException             extends ResourceDepException             where @false ⊥ end
  event KcallException                   extends KcallException                   where @false ⊥ end
  event Interrupt                        extends Interrupt                        where @false ⊥ end
  event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
  event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
  event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
  event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
  event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
  event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
  event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
  event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
  event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
  event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
```

```
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event GETR_BadType extends IllegalInstructionException
  any resType
  where
    @resTypeType  resType∈Bits8Nat
    @isValidCheck IsGetrResourceType(resType)=FALSE
end

event FREER_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsFreerResourceIdent(resIdent)=FALSE
end

event SETEV_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsSetevResourceIdent(resIdent)=FALSE
end

event SETV_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsSetvResourceIdent(resIdent)=FALSE
end

event SETD_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsSetdResourceIdent(resIdent)=FALSE
end

event GETD_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsGetdResourceIdent(resIdent)=FALSE
end

event SETC_BadIdent extends IllegalResourceException
```

```
  where
    @isValidCheck IsSetcResourceIdent(resIdent)=FALSE
end

event SETCLK_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsSetclkResourceIdent(resIdent)=FALSE
end

event SETRDY_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsSetrdyResourceIdent(resIdent)=FALSE
end

event SETRDY_NotPort extends IllegalResourceException
  any srcIdent
  where
    @srcIdentType srcIdent∈Bits
    @isPort       IsPortIdent(srcIdent)=FALSE
end

event SETRDY_NotOneBitPort extends IllegalResourceException
  any srcIdent
  where
    @srcIdentType srcIdent∈Bits
    @isPort       IsPortIdent(srcIdent)=TRUE
    @isOneBit     IsOneBitPortIdent(srcIdent)=FALSE
end

event IN_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsInResourceIdent(resIdent)=FALSE
end

event OUT_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsOutResourceIdent(resIdent)=FALSE
end

event EDU_BadIdent extends IllegalResourceException
  where
    @isValidCheck IsEventResourceIdent(resIdent)=FALSE
end

event EEF_BadIdent extends IllegalResourceException
```

```
    where
      @isValidCheck IsEventResourceIdent(resIdent)=FALSE
  end

  event EET_BadIdent extends IllegalResourceException
    where
      @isValidCheck IsEventResourceIdent(resIdent)=FALSE
  end

  event EEU_BadIdent extends IllegalResourceException
    where
      @isValidCheck IsEventResourceIdent(resIdent)=FALSE
  end

end
```

14.1.31machine XCoreIsaResourceFinal

```
machine XCoreIsaResourceFinal // Instruction events common to any resource

refines XCoreIsaResource1

sees XCoreBasic XCoreResourceClassifications

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

events

  event INITIALISATION extends INITIALISATION end

  event GETR_Rus_BadType extends GETR_BadType
    where
      @instCheck IsRus(instr↦16↦0)=TRUE
      @op2Fetch  resType=XRus_op2(instr)
  end

  event FREER_1r_BadIdent extends FREER_BadIdent
    any op1
    where
      @instCheck  Is1r(instr↦2↦0)=TRUE
      @op1Fetch   op1=X1r_op1(instr)
      @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event SETEV_1r_BadIdent extends SETEV_BadIdent
    any op1
    where
      @instCheck Is1r(instr↦7↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event SETV_1r_BadIdent extends SETV_BadIdent
    any op1
    where
      @instCheck Is1r(instr↦8↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
  end
```

```
event SETD_R2r_BadIdent extends SETD_BadIdent
  any op1
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Ru6_BadIdent extends SETC_BadIdent
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Lru6_BadIdent extends SETC_BadIdent
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_L2r_BadIdent extends SETC_BadIdent
  any op1
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETCLK_Lr2r_BadIdent extends SETCLK_BadIdent
  any op1
  where
    @instCheck IsLr2r(instr↦1↦1↦12)=TRUE
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
 end

event SETRDY_Lr2r_BadIdent extends SETRDY_BadIdent
  any op1
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
```

```
    @op1Fetch  op1=XLr2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETRDY_Lr2r_NotPort extends SETRDY_NotPort
  any op2
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
    @op2Fetch  op2=XLr2r_op2(instr)
    @srcIdent  srcIdent=(threadRegArrays(threadIndex))(op2)
end

event SETRDY_Lr2r_NotOneBitPort extends SETRDY_NotOneBitPort
  any op2
  where
    @instCheck IsLr2r(instr↦5↦0↦12)=TRUE
    @op2Fetch  op2=XLr2r_op2(instr)
    @srcIdent  srcIdent=(threadRegArrays(threadIndex))(op2)
end

event GETD_L2r_BadIdent extends GETD_BadIdent
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event IN_2r_BadIdent extends IN_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event OUT_R2r_BadIdent extends OUT_BadIdent
  any op1
  where
    @instCheck IsR2r(instr↦21↦0)=TRUE
    @op2Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EDU_1r_BadIdent extends EDU_BadIdent
```

```
    any op1
    where
      @instCheck Is1r(instr↦0↦0)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event EEF_2r_BadIdent extends EEF_BadIdent
    any op2
    where
      @instCheck Is2r(instr↦5↦1)=TRUE
      @op2Fetch  op2=X2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
  end

  event EET_2r_BadIdent extends EET_BadIdent
    any op2
    where
      @instCheck Is2r(instr↦4↦1)=TRUE
      @op2Fetch  op2=X2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
  end

  event EEU_1r_BadIdent extends EEU_BadIdent
    any op1
    where
      @instCheck Is1r(instr↦0↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
  end

end
```

14.1.32machine XCoreIsaThread1

```
machine XCoreIsaThread1  // Thread control instructions
refines XCoreFinal
sees XCoreBasic XCoreSynchronisers

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

          threadIsSynchronous threadSynchroniserIndex mPausingSynchroniserIndex
          synchroniserStatus synchroniserMasterIndex synchroniserSlaves synchroniserSlavesStatus

          threadEeble threadInenb

invariants
  @threadIsSynchronousType threadIsSynchronous ∈ ThreadIndexDom → BOOL
  @threadSyncIndexType     threadSynchroniserIndex ∈ ThreadIndexDom → SynchroniserIndexDom
  @mPausingSyncIndexType   mPausingSynchroniserIndex ∈ ThreadIndexDom → SynchroniserIndexDom
  @syncStatus              synchroniserStatus ∈ SynchroniserIndexDom → SynchroniserStatus
  @syncMasterIndex         synchroniserMasterIndex ∈ SynchroniserIndexDom → ThreadIndexDom
  @synchSlaves             synchroniserSlaves ∈ SynchroniserIndexDom → ThreadCount
  @synchSlavesStatus       synchroniserSlavesStatus ∈ SynchroniserIndexDom → ThreadStatusArray
  @threadEeble             threadEeble ∈ ThreadIndexDom → BOOL
  @threadInenb             threadInenb ∈ ThreadIndexDom → BOOL

events

  event INITIALISATION extends INITIALISATION
    then

      @threadIsSynch     threadIsSynchronous :∈ ThreadIndexDom → BOOL
      @mPausingSyncIndex mPausingSynchroniserIndex :∈ ThreadIndexDom → SynchroniserIndexDom
      @threadSyncIndex   threadSynchroniserIndex :∈ ThreadIndexDom → SynchroniserIndexDom
      @synchsStatus      synchroniserStatus ≔
                                 {0↦FREE} □ {1↦FREE} □ {2↦FREE} □ {3↦FREE} □
                                 {4↦FREE} □ {5↦FREE} □ {6↦FREE}
      @syncMasterIndex   synchroniserMasterIndex :∈ SynchroniserIndexDom → ThreadIndexDom
      @synchSlaves       synchroniserSlaves :∈ SynchroniserIndexDom → ThreadCount
      @syncSlaveStat     synchroniserSlavesStatus ≔
                                 {0↦{0↦FREE}} □ {0↦{1↦FREE}} □ {0↦{2↦FREE}} □ {0↦{3↦FREE}} □
                                 {0↦{4↦FREE}} □ {0↦{5↦FREE}} □ {0↦{6↦FREE}} □ {0↦{7↦FREE}} □

                                 {1↦{0↦FREE}} □ {1↦{1↦FREE}} □ {1↦{2↦FREE}} □ {1↦{3↦FREE}} □
```

```
                                {1↦{4↦FREE}} □ {1↦{5↦FREE}} □ {1↦{6↦FREE}} □ {1↦{7↦FREE}} □

                                {2↦{0↦FREE}} □ {2↦{1↦FREE}} □ {2↦{2↦FREE}} □ {2↦{3↦FREE}} □
                                {2↦{4↦FREE}} □ {2↦{5↦FREE}} □ {2↦{6↦FREE}} □ {2↦{7↦FREE}} □

                                {3↦{0↦FREE}} □ {3↦{1↦FREE}} □ {3↦{2↦FREE}} □ {3↦{3↦FREE}} □
                                {3↦{4↦FREE}} □ {3↦{5↦FREE}} □ {3↦{6↦FREE}} □ {3↦{7↦FREE}} □

                                {4↦{0↦FREE}} □ {4↦{1↦FREE}} □ {4↦{2↦FREE}} □ {4↦{3↦FREE}} □
                                {4↦{4↦FREE}} □ {4↦{5↦FREE}} □ {4↦{6↦FREE}} □ {4↦{7↦FREE}} □

                                {5↦{0↦FREE}} □ {5↦{1↦FREE}} □ {5↦{2↦FREE}} □ {5↦{3↦FREE}} □
                                {5↦{4↦FREE}} □ {5↦{5↦FREE}} □ {5↦{6↦FREE}} □ {5↦{7↦FREE}} □

                                {6↦{0↦FREE}} □ {6↦{1↦FREE}} □ {6↦{2↦FREE}} □ {6↦{3↦FREE}} □
                                {6↦{4↦FREE}} □ {6↦{5↦FREE}} □ {6↦{6↦FREE}} □ {6↦{7↦FREE}}

    @threadEeble        threadEeble :∈ ThreadIndexDom → BOOL
    @threadInenb        threadInenb :∈ ThreadIndexDom → BOOL

end

event Loading                           extends Loading                           where @false ⊥ end
event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
event LinkErrorException                extends LinkErrorException                where @false ⊥ end
event IllegalPcException                extends IllegalPcException                where @false ⊥ end
event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
event LoadStoreException                extends LoadStoreException                where @false ⊥ end
event IllegalPsException                extends IllegalPsException                where @false ⊥ end
event ArithmeticException               extends ArithmeticException               where @false ⊥ end
event EcallException                    extends EcallException                    where @false ⊥ end
event ResourceDepException              extends ResourceDepException              where @false ⊥ end
event KcallException                    extends KcallException                    where @false ⊥ end
event Interrupt                         extends Interrupt                         where @false ⊥ end
event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
```

```
event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
event WriteSrStatusInstruction         extends WriteSrStatusInstruction         where @false ⊥ end
event Write1Memory0RegisterInstruction extends Write1Memory0RegisterInstruction where @false ⊥ end
event Write2Memory0RegisterInstruction extends Write2Memory0RegisterInstruction where @false ⊥ end
event Write4Memory0RegisterInstruction extends Write4Memory0RegisterInstruction where @false ⊥ end
event Write4Memory1RegisterInstruction extends Write4Memory1RegisterInstruction where @false ⊥ end
event WriteThreadRegisterInstruction   extends WriteThreadRegisterInstruction   where @false ⊥ end
event WriteThreadPcInstruction         extends WriteThreadPcInstruction         where @false ⊥ end
event BranchInstruction                extends BranchInstruction                where @false ⊥ end
event BranchLinkInstruction            extends BranchLinkInstruction            where @false ⊥ end
event DcallInstruction                 extends DcallInstruction                 where @false ⊥ end
event DretInstruction                  extends DretInstruction                  where @false ⊥ end
event KretInstruction                  extends KretInstruction                  where @false ⊥ end
event RetSpInstruction                 extends RetSpInstruction                 where @false ⊥ end
event PausingInstruction               extends PausingInstruction               where @false ⊥ end

event CLRE_OK extends Write0RegisterInstruction
  then
    @threadEeble threadEeble(threadIndex):=FALSE
    @threadInenb threadInenb(threadIndex):=FALSE
end

event FREET_OK extends WriteStatusInstruction
  where
    @nextThreadStatus nextThreadStatus=FREE+0  // Free thread
end

event GETID_OK extends Write1RegisterInstruction
   where
    @dBits dBits=GetThreadIdent(threadIndex)
end

event GETR_No_Threads extends Write1RegisterInstruction
  any resType
   where
    @resIdType  resType=ResourceThread
    @checkFree  CountFreeThreads(threadStatus)=0 // All used up
    @dits       dBits=NatToBits32(NoResourceId)
end

event GETR_Threads_OK extends Write1RegisterInstruction
```

```
  any resType resIndex
  where
    @resIdType  resType=ResourceThread
    @checkFree  CountFreeThreads(threadStatus)≠0  // Thread available somewhere
    @resIndex   resIndex=FindFreeThread(threadStatus)
    @dBits      dBits=GetThreadIdent(resIndex)
  then
    @threadEeble  threadEeble(resIndex):=FALSE
    @threadInenb  threadInenb(resIndex):=FALSE
    @threadIsSync threadIsSynchronous(resIndex):=FALSE
end

event FREER_Thread_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsThreadIdent(resIdent)=TRUE
    @statusCheck IsThreadInUse(resIdent↦threadStatus)=FALSE
end

event FREER_Thread_Running extends IllegalResourceException
  where
    @typeCheck   IsThreadIdent(resIdent)=TRUE
    @statusCheck IsThreadRunning(resIdent↦threadStatus)=TRUE
end

event FREER_Thread_IsSynchronous extends IllegalResourceException
  where
    @typeCheck   IsThreadIdent(resIdent)=TRUE
    @statusCheck IsThreadSynchronous(resIdent↦threadIsSynchronous)=TRUE
end

event FREER_Thread_OK extends WriteStatusInstruction
  any resIdent resIndex
  where
    @resIdType        resIdent∈Bits
    @typeCheck        IsThreadIdent(resIdent)=TRUE
    @statusCheck      IsThreadRunning(resIdent↦threadStatus)=FALSE
    @syncCheck        IsThreadSynchronous(resIdent↦threadIsSynchronous)=FALSE
    @resIndex         resIndex=GetResourceIndex(resIdent)
    @nextThreadStatus nextThreadStatus=FREE+0
end

event GETR_No_Synchronisers extends Write1RegisterInstruction
  any resType
  where
```

```
    @resType   resType=ResourceSynchroniser
    @checkFree CountFreeSynchronisers(synchroniserStatus)=0 // All used up
    @dBits     dBits=NatToBits32(NoResourceId)
end

event GETR_Synchroniser_OK extends Write1RegisterInstruction
  any resType resIndex
  where
    @resType     resType=ResourceSynchroniser
    @checkFree   CountFreeSynchronisers(synchroniserStatus)≠0  // Synchroniser available somewhere
    @resIndex    resIndex=FindFreeSynchroniser(synchroniserStatus)
    @dBits       dBits=GetSynchroniserIdent(resIndex)
  then
    @syncStatus  synchroniserStatus(resIndex):=SPAUSING
    @syncMaster  synchroniserMasterIndex(resIndex):=threadIndex
    @synchSlaves synchroniserSlaves(resIndex):=0
end

event FREER_Synchroniser_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=FALSE
end

event FREER_Synchroniser_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @synchIndex  resIndex=GetResourceIndex(resIdent)
  then
    @syncStatusChange synchroniserStatus(resIndex):=FREE
end

event GETST_BadIdent extends IllegalResourceException
  where
    @syncOk IsSynchroniserIdent(resIdent)=FALSE
end

event GETST_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=FALSE
```

```
end

event GETST_NoThreads extends Write1RegisterInstruction
  where
    @freeCheck CountFreeThreads(threadStatus)=0 // All used up
    @dBits     dBits=NatToBits32(NoResourceId)
end

event GETST_OK extends Write1RegisterStatusInstruction
  any resIdent resIndex syncSlaves
  where
    @resIdType     resIdent∈Bits
    @typeCheck     IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck   IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @syncSlaves    syncSlaves=synchroniserSlaves(resIndex)
    @slavesLimit   syncSlaves < MaxThreads
    @freeCheck     CountFreeThreads(threadStatus)≠0
    @threadIndex   newThreadIndex=FindFreeThread(threadStatus)
    @threadStatus  newThreadStatus=SPAUSING+0
    @dBits         dBits=GetThreadIdent(newThreadIndex)
  then
    @threadIsSync       threadIsSynchronous(newThreadIndex):=TRUE
    @threadSyncIndex    threadSynchroniserIndex(newThreadIndex):=resIndex                // Bind new thread to synchroniser
    @changeSyncSlaves   synchroniserSlaves:=synchroniserSlaves  {resIndex↦(syncSlaves+1)}
    @changeSyncStatus   synchroniserSlavesStatus:=synchroniserSlavesStatus  {resIndex↦{newThreadIndex↦SPAUSING}}
end

event MJOIN_BadIdent extends IllegalResourceException
  where
    @syncOk IsSynchroniserIdent(resIdent)=FALSE
end

event MJOIN_NotInUse extends IllegalResourceException
   where
    @typeCheck   IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=FALSE
end

event MJOIN_No_Slaves_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @typeCheck   IsSynchroniserIdent(resIdent)=TRUE
```

```
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @checkIndex  resIndex=GetResourceIndex(resIdent)
    @zeroSlaves  synchroniserSlaves(resIndex)=0
end

event MJOIN_All_Paused_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType     resIdent∈Bits
    @typeCheck     IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck   IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @checkIndex    resIndex=GetResourceIndex(resIdent)
    @zeroSlaves    synchroniserSlaves(resIndex)≠0
    @checkSlaves   CountSpausedThreads(synchroniserSlavesStatus(resIndex))=synchroniserSlaves(resIndex)   // All slaves paused
  then
    @changeSyncStatus synchroniserStatus(resIndex):=CLOSING
end

event MJOIN_Some_Paused_OK extends WriteSrStatusInstruction
  any resIdent resIndex
  where
    @resIdType     resIdent∈Bits
    @typeCheck     IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck   IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @checkIndex    resIndex=GetResourceIndex(resIdent)
    @zeroSlaves    synchroniserSlaves(resIndex)≠0
    @checkSlaves   CountSpausedThreads(synchroniserSlavesStatus(resIndex))≠synchroniserSlaves(resIndex)    // Some slaves paused
    @srBits        srBits  = EightBits1ToBits32(BitClr↦ieble↦inenb↦inint↦ink↦sink↦waiting↦fast) // eeble=0
    @inintFalse    inint   = BitClr
    @nts           nextThreadStatus=MPAUSING+0
  then
    @changeSyncStatus   synchroniserStatus(resIndex):=CLOSING
    @mPausingSyncIndex  mPausingSynchroniserIndex(threadIndex):=resIndex
end

event Mpausing_Closing_OK extends ThreadMpausing
  any resIndex
  where
    @resIndex      resIndex=mPausingSynchroniserIndex(threadIndex)
    @statusCheck   synchroniserStatus(resIndex)=CLOSING
    @checkSlaves   CountFreeThreads(synchroniserSlavesStatus(resIndex))≠MaxThreads   // Some threads not yet freed
    @nextThdStat   nextThreadStatus=MPAUSING+0
end
```

```
event Mpausing_Closed_OK extends ThreadMpausing
  any resIndex
  where
    @resIndex     resIndex=mPausingSynchroniserIndex(threadIndex)
    @statusCheck  synchroniserStatus(resIndex)=CLOSING
    @checkSlaves  CountFreeThreads(synchroniserSlavesStatus(resIndex))=MaxThreads // All threads freed
    @nextThdStat  nextThreadStatus=RUNNING+0
  then
    @changeSyncStatus   synchroniserStatus(resIndex):=SPAUSING
end

event MSYNC_BadIdent extends IllegalResourceException
  where
    @syncOk IsSynchroniserIdent(resIdent)=FALSE
end

event MSYNC_NotInUse extends IllegalResourceException
  where
    @syncOk      IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=FALSE
end

event MSYNC_No_Slaves_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @syncOk      IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @checkIndex  resIndex=GetResourceIndex(resIdent)
    @zeroSlaves  synchroniserSlaves(resIndex)=0
end

event MSYNC_All_Paused_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdType   resIdent∈Bits
    @syncOk      IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @checkIndex  resIndex=GetResourceIndex(resIdent)
    @zeroSlaves  synchroniserSlaves(resIndex)≠0
    @checkSlaves CountSpausedThreads(synchroniserSlavesStatus(resIndex))=synchroniserSlaves(resIndex)  // All slaves paused
  then
    @changeSyncStatus synchroniserStatus(resIndex):=RUNNING
end
```

```
event MSYNC_Some_Paused_OK extends WriteStatusInstruction
  any resIdent resIndex
  where
    @resIdType      resIdent∈Bits
    @syncOk         IsSynchroniserIdent(resIdent)=TRUE
    @statusCheck    IsSynchroniserInUse(resIdent↦synchroniserStatus)=TRUE
    @checkIndex     resIndex=GetResourceIndex(resIdent)
    @zeroSlaves     synchroniserSlaves(resIndex)≠0
    @checkSlaves    CountSpausedThreads(synchroniserSlavesStatus(resIndex))≠synchroniserSlaves(resIndex)   // Some slaves paused
    @nextThrdStatus nextThreadStatus=MPAUSING+0
  then
     @changeSyncStatus   synchroniserStatus(resIndex):=RUNNING
     @mPausingSyncIndex  mPausingSynchroniserIndex(threadIndex):=resIndex
end

event Mpausing_Starting_OK extends ThreadMpausing
  any resIndex
  where
    @resIndex      resIndex=mPausingSynchroniserIndex(threadIndex)
    @statusCheck   synchroniserStatus(resIndex)=RUNNING
    @checkSlaves   CountRunningThreads(synchroniserSlavesStatus(resIndex))≠synchroniserSlaves(resIndex)   // Some threads not running
    @nextThdStat   nextThreadStatus=MPAUSING+0
end

event Mpausing_Started_OK extends ThreadMpausing
  any resIndex
  where
    @resIndex      resIndex=mPausingSynchroniserIndex(threadIndex)
    @statusCheck   synchroniserStatus(resIndex)=RUNNING
    @checkSlaves   CountRunningThreads(synchroniserSlavesStatus(resIndex))=synchroniserSlaves(resIndex)  // All slaves running
    @nextThdStat   nextThreadStatus=RUNNING+0
  then
    @changeSyncStatus   synchroniserStatus(resIndex):=SPAUSING
end

event SSYNC_NotSynchronous extends IllegalResourceException
  where
    @threadIsSync threadIsSynchronous(threadIndex)=FALSE
end

event SSYNC_Close_OK extends WriteStatusInstruction
  any resIndex
  where
    @threadIsSync threadIsSynchronous(threadIndex)=TRUE
    @syncIndex    resIndex=threadSynchroniserIndex(threadIndex)
```

```
    @statusCheck  synchroniserStatus(resIndex)=CLOSING
    @nextThdStat  nextThreadStatus=FREE+0
  then
    @changeSyncStatus synchroniserSlavesStatus:=synchroniserSlavesStatus  {resIndex↦{threadIndex↦FREE}}
end

event SSYNC_Pause_OK extends WriteStatusInstruction
  any resIndex
  where
    @threadIsSync threadIsSynchronous(threadIndex)=TRUE
    @syncIndex    resIndex=threadSynchroniserIndex(threadIndex)
    @statusCheck  synchroniserStatus(resIndex)=SPAUSING
    @nextThdStat  nextThreadStatus=SPAUSING+0
  then
    @changeSyncStatus synchroniserSlavesStatus:=synchroniserSlavesStatus  {resIndex↦{threadIndex↦SPAUSING}}
end

event SSYNC_Run_OK extends Write0RegisterInstruction
  any resIndex
  where
    @threadIsSync threadIsSynchronous(threadIndex)=TRUE
    @syncIndex    resIndex=threadSynchroniserIndex(threadIndex)
    @statusCheck  synchroniserStatus(resIndex)=RUNNING
end

event Spausing_OK extends ThreadSpausing
  any resIndex
  where
    @resIndex     resIndex=threadSynchroniserIndex(threadIndex)
    @statusCheck  synchroniserStatus(resIndex)=SPAUSING
    @nextThdStat  nextThreadStatus=SPAUSING+0
 end

event Spausing_Continue_OK extends ThreadSpausing
  any resIndex
  where
    @resIndex      resIndex=threadSynchroniserIndex(threadIndex)
    @statusCheck   synchroniserStatus(resIndex)=RUNNING
    @nextThdStat   nextThreadStatus=RUNNING+0
  then
    @changeSyncStatus   synchroniserSlavesStatus:=synchroniserSlavesStatus  {resIndex↦{threadIndex↦RUNNING}}
end

event Spausing_Close_OK extends ThreadSpausing
  any resIndex syncSlaves
```

```
  where
    @resIndex      resIndex=threadSynchroniserIndex(threadIndex)
    @statusCheck   synchroniserStatus(resIndex)=CLOSING
    @syncSlaves    syncSlaves=synchroniserSlaves(resIndex)
    @slavesGtZero  syncSlaves>0
    @nextThdStat   nextThreadStatus=SPAUSING+0
  then
    @changeSyncSlaves   synchroniserSlaves≔synchroniserSlaves  {resIndex↦(syncSlaves-1)}
    @changeSyncStatus   synchroniserSlavesStatus≔synchroniserSlavesStatus  {resIndex↦{threadIndex↦FREE}}  // Signal thread freed
end

event Spausing_Closed_OK extends ThreadSpausing
  any resIndex syncSlaves
  where
    @resIndex      resIndex=threadSynchroniserIndex(threadIndex)
    @statusCheck   synchroniserStatus(resIndex)=CLOSING
    @syncSlaves    syncSlaves=synchroniserSlaves(resIndex)
    @slavesGtZero  syncSlaves=0
    @nextThdStat   nextThreadStatus=FREE+0
end

event TSETMR_NotSynchronous extends IllegalResourceException
  where
    @threadIsSync threadIsSynchronous(threadIndex)=FALSE
end

event TSETMR_MasterNotInUse extends IllegalResourceException
  any syncIndex resIndex
  where
    @threadIsSync threadIsSynchronous(threadIndex)=TRUE
    @syncIndex    syncIndex=threadSynchroniserIndex(threadIndex)
    @resIndex     resIndex=synchroniserMasterIndex(syncIndex) // Look up master via synchroniser
    @statusCheck  threadStatus(resIndex)=FREE
end

event TSETMR_OK extends WriteThreadRegisterInstruction
  any syncIndex
  where
    @threadIsSync threadIsSynchronous(threadIndex)=TRUE
    @syncIndex    syncIndex=threadSynchroniserIndex(threadIndex)
    @resIndex     resIndex=synchroniserMasterIndex(syncIndex) // Look up master via synchroniser
    @checkThread  threadStatus(resIndex)≠FREE
end

event TSTART_BadIdent extends IllegalResourceException
```

```
  where
    @syncOk IsThreadIdent(resIdent)=FALSE
end

event TSTART_NotSpaused extends IllegalResourceException
  where
    @identOk  IsThreadIdent(resIdent)=TRUE
    @statusOk IsThreadSpausing(resIdent↦threadStatus)=FALSE
end

event TSTART_OK extends WriteStatusInstruction
  any resIdent resIndex
  where
    @tIdentType   resIdent∈Bits
    @identOk      IsThreadIdent(resIdent)=TRUE
    @statusOk     IsThreadSpausing(resIdent↦threadStatus)=TRUE
    @indexOk      resIndex ∈ ThreadIndexDom
    @nextThdStat  nextThreadStatus=RUNNING+0
end

event WAITEF_TRUE_OK extends WriteSrStatusInstruction
  any c
  where
    @cType    c∈Bits32Nat
    @cond     c=0
    @srBits   srBits=EightBits1ToBits32(BitSet↦ieble↦inenb↦inint↦ink↦sink↦waiting↦fast) // eeble=0
    @nts      nextThreadStatus=WAITING+0
end

event WAITEF_FALSE_OK extends Write0RegisterInstruction
  any c
  where
    @cType c∈Bits32Nat
    @cond  c≠0
end

event WAITET_TRUE_OK extends WriteSrStatusInstruction
  any c
  where
    @cType    c∈Bits32Nat
    @cond     c≠0
    @srBits   srBits=EightBits1ToBits32(BitSet↦ieble↦inenb↦inint↦ink↦sink↦waiting↦fast) // eeble=0
    @nts      nextThreadStatus=WAITING+0
end
```

```
  event WAITET_FALSE_OK extends Write0RegisterInstruction
    any c
    where
      @cType c∈Bits32Nat
      @cond  c=0
  end

  event WAITEU_OK extends WriteSrStatusInstruction
    where
      @srBits srBits=EightBits1ToBits32(BitSet↦ieble↦inenb↦inint↦ink↦sink↦waiting↦fast) // eeble=0
      @nts    nextThreadStatus=WAITING+0
  end

end
```

14.1.33machine XCoreIsaThreadFinal

```
machine XCoreIsaThreadFinal  // Thread control instructions
refines XCoreIsaThread1
sees XCoreBasic XCoreSynchronisers

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray
          threadIsSynchronous threadSynchroniserIndex mPausingSynchroniserIndex
          synchroniserStatus synchroniserMasterIndex synchroniserSlaves synchroniserSlavesStatus
          threadEeble threadInenb

events

  event INITIALISATION extends INITIALISATION end

  event CLRE_0r_OK extends CLRE_OK
    where
      @instCheck Is0r(instr↦0↦0↦13)=TRUE
      @instrSize instrSize=InstSizeShort+0
  end

  event FREET_0r_OK extends FREET_OK
    where
      @instCheck Is0r(instr↦0↦0↦15)=TRUE
  end

  event GETID_0r_OK extends GETID_OK
     where
      @instCheck Is0r(instr↦2↦0↦14)=TRUE
      @d         d=R11Index+0
      @instrSize instrSize=InstSizeShort+0
  end

  event GETR_Rus_No_Threads extends GETR_No_Threads
    where
      @instCheck IsRus(instr↦16↦0)=TRUE
      @op1Fetch  d=XRus_op1(instr)
      @op2Fetch  resType=XRus_op2(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event GETR_Rus_Threads_OK extends GETR_Threads_OK
    where
```

```
    @instCheck IsRus(instr↦16↦0)=TRUE
    @op1Fetch  d=XRus_op1(instr)
    @op2Fetch  resType=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event FREER_1r_Thread_NotInUse extends FREER_Thread_NotInUse
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
end

event FREER_1r_Thread_Running extends FREER_Thread_Running
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
end

event FREER_1r_Thread_IsSynchronous extends FREER_Thread_IsSynchronous
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
end

event FREER_1r_Thread_OK extends FREER_Thread_OK
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize  instrSize=InstSizeShort+0
end

event GETR_Rus_No_Synchronisers extends GETR_No_Synchronisers
  where
    @instCheck IsRus(instr↦16↦0)=TRUE
    @op1Fetch  d=XRus_op1(instr)
    @op2Fetch  resType=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
```

```
end

event GETR_Rus_Synchroniser_OK extends GETR_Synchroniser_OK
  where
    @instCheck IsRus(instr↦16↦0)=TRUE
    @op1Fetch  d=XRus_op1(instr)
    @op2Fetch  resType=XRus_op2(instr)
    @instrSize instrSize=InstSizeShort+0
end

event FREER_1r_Synchroniser_NotInUse extends FREER_Synchroniser_NotInUse
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
end

event FREER_1r_Synchroniser_OK extends FREER_Synchroniser_OK
  any op1
  where
    @instCheck  Is1r(instr↦2↦0)=TRUE
    @op1Fetch   op1=X1r_op1(instr)
    @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize  instrSize=InstSizeShort+0
end

event GETST_2r_BadIdent extends GETST_BadIdent
  any op2
  where
    @instCheck Is2r(instr↦0↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event GETST_2r_NotInUse extends GETST_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦0↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event GETST_2r_NoThreads extends GETST_NoThreads
  where
```

```
    @instCheck Is2r(instr↦0↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @instrSize instrSize=InstSizeShort+0
end

event GETST_2r_OK extends GETST_OK
  any op2
  where
    @instCheck Is2r(instr↦0↦1)=TRUE
    @op1Fetch  d=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event MJOIN_1r_BadIdent extends MJOIN_BadIdent
  any op1
  where
    @instCheck   Is1r(instr↦2↦1)=TRUE
    @op1Fetch    op1=X1r_op1(instr)
    @resIdent    resIdent=(threadRegArrays(threadIndex))(op1)
end

event MJOIN_1r_NotInUse extends MJOIN_NotInUse
  any op1
  where
    @instCheck   Is1r(instr↦2↦1)=TRUE
    @op1Fetch    op1=X1r_op1(instr)
    @resIdent    resIdent=(threadRegArrays(threadIndex))(op1)
end

event MJOIN_1r_No_Slaves_OK extends MJOIN_No_Slaves_OK
  any op1
  where
    @instCheck   Is1r(instr↦2↦1)=TRUE
    @op1Fetch    op1=X1r_op1(instr)
    @resIdent    resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize   instrSize=InstSizeShort+0
end

event MJOIN_1r_All_Paused_OK extends MJOIN_All_Paused_OK
  any op1
  where
    @instCheck    Is1r(instr↦2↦1)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
```

```
    @resIdent     resIdent=(threadRegArrays(threadIndex))(op1)
end

event MJOIN_1r_Some_Paused_OK extends MJOIN_Some_Paused_OK
  any op1
  where
    @instCheck    Is1r(instr↦2↦1)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
    @resIdent     resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize    instrSize=InstSizeShort+0
end

event Mpausing_Closing_OK extends Mpausing_Closing_OK end
event Mpausing_Closed_OK extends Mpausing_Closed_OK end

event MSYNC_1r_BadIdent extends MSYNC_BadIdent
  any op1
  where
    @instCheck Is1r(instr↦3↦1)=TRUE
    @op1Fetch op1=X1r_op1(instr)
    @resIdent resIdent=(threadRegArrays(threadIndex))(op1)
end

event MSYNC_1r_NotInUse extends MSYNC_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦3↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event MSYNC_1r_No_Slaves_OK extends MSYNC_No_Slaves_OK
  any op1
  where
    @instCheck Is1r(instr↦3↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event MSYNC_1r_All_Paused_OK extends MSYNC_All_Paused_OK
  any op1
  where
    @instCheck    Is1r(instr↦3↦1)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
```

```
    @resIdent     resIdent=(threadRegArrays(threadIndex))(op1)
end

event MSYNC_1r_Some_Paused_OK extends MSYNC_Some_Paused_OK
  any op1
  where
    @instCheck    Is1r(instr↦3↦1)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
    @resIdent     resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize    instrSize=InstSizeShort+0
end

event Mpausing_Starting_OK extends Mpausing_Starting_OK end
event Mpausing_Started_OK extends Mpausing_Started_OK end

event SSYNC_0r_NotSynchronous extends SSYNC_NotSynchronous
  where
    @instCheck  Is0r(instr↦0↦0↦14)=TRUE
end

event SSYNC_0r_Close_OK extends SSYNC_Close_OK
  where
    @instCheck     Is0r(instr↦0↦0↦14)=TRUE
    @instrSize     instrSize=InstSizeShort+0
end

event SSYNC_0r_Pause_OK extends SSYNC_Pause_OK
  where
    @instCheck  Is0r(instr↦0↦0↦14)=TRUE
    @instrSize  instrSize=InstSizeShort+0
end

event SSYNC_0r_Run_OK extends SSYNC_Run_OK
  where
    @instCheck  Is0r(instr↦0↦0↦14)=TRUE
    @instrSize  instrSize=InstSizeShort+0
end

event Spausing_OK extends Spausing_OK end
event Spausing_Continue_OK extends Spausing_Continue_OK end
event Spausing_Close_OK extends Spausing_Close_OK end
event Spausing_Closed_OK extends Spausing_Closed_OK end

event TSETMR_2r_NotSynchronous extends TSETMR_NotSynchronous
  where
```

```
    @instCheck Is2r(instr↦3↦1)=TRUE
end

event TSETMR_2r_MasterNotInUse extends TSETMR_MasterNotInUse
  where
    @instCheck Is2r(instr↦3↦1)=TRUE
end

event TSETMR_2r_OK extends TSETMR_OK
  any op2
  where
    @instCheck   Is2r(instr↦3↦1)=TRUE
    @op1Fetch    d=X2r_op1(instr)
    @op2Fetch    op2=X2r_op2(instr)
    @dBits       dBits=(threadRegArrays(resIndex))(op2)
    @instrSize   instrSize=InstSizeShort+0
end

event TSTART_1r_BadIdent extends TSTART_BadIdent
  any op1
  where
    @instCheck    Is1r(instr↦3↦0)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
    @threadId     resIdent=(threadRegArrays(threadIndex))(op1)
end

event TSTART_1r_NotSpaused extends TSTART_NotSpaused
  any op1
  where
    @instCheck    Is1r(instr↦3↦0)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
    @threadId     resIdent=(threadRegArrays(threadIndex))(op1)
end

event TSTART_1r_OK extends TSTART_OK
  any op1
  where
    @instCheck    Is1r(instr↦3↦0)=TRUE
    @op1Fetch     op1=X1r_op1(instr)
    @threadId     resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize    instrSize=InstSizeShort+0
end

event WAITEF_1r_TRUE_OK extends WAITEF_TRUE_OK
  any op1
```

```
    where
      @instCheck Is1r(instr↦1↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
  end

  event WAITEF_1r_FALSE_OK extends WAITEF_FALSE_OK
    any op1
    where
      @instCheck Is1r(instr↦1↦1)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
      @instrSize instrSize=InstSizeShort+0
  end

  event WAITET_1r_TRUE_OK extends WAITET_TRUE_OK
    any op1
    where
      @instCheck Is1r(instr↦1↦0)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
  end

  event WAITET_1r_FALSE_OK extends WAITET_FALSE_OK
    any op1
    where
      @instCheck Is1r(instr↦1↦0)=TRUE
      @op1Fetch  op1=X1r_op1(instr)
      @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
      @instrSize instrSize=InstSizeShort+0
  end

  event WAITEU_0r_OK extends WAITEU_OK
    where
      @instCheck Is0r(instr↦0↦0↦12)=TRUE
      @instrSize instrSize=InstSizeShort+0
  end
end
```

14.1.34machine XCoreIsaTimer1

```
machine XCoreIsaTimer1

refines XCoreFinal

sees XCoreBasic XCoreTimers XCoreClocks

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

          timerIndex
          timerStatus timerBlockingMode timerEventMode timerThreadIdent timerEnvironmentVector timerVector
          timerClockIdent timerValue timerTriggerValue

invariants
  @timerIndex        timerIndex ∈ TimerIndexDom
  @timerStatus       timerStatus ∈  TimerIndexDom → TimerStatus
  @timerBlockingMode timerBlockingMode ∈  TimerIndexDom → BOOL
  @timerEventMode    timerEventMode ∈  TimerIndexDom → BOOL
  @timerThreadIdent  timerThreadIdent ∈  TimerIndexDom → Bits
  @timerEnvVec       timerEnvironmentVector ∈  TimerIndexDom → Bits32Nat
  @timerVec          timerVector ∈  TimerIndexDom → Bits32Nat
  @timerValue        timerValue ∈  TimerIndexDom → Bits32Nat
  @timerTriggerValue timerTriggerValue ∈  TimerIndexDom → Bits32Nat
  @timerClockIdent   timerClockIdent ∈  TimerIndexDom → Bits

events

  event INITIALISATION extends INITIALISATION
    then
     @timerIndex        timerIndex ≔ 0
     @timerStatus       timerStatus ≔ {0↦FREE} ⩤ {1↦FREE} ⩤ {2↦FREE} ⩤ {3↦FREE}
     @timerBlockingMode timerBlockingMode :∈  TimerIndexDom → BOOL
     @timerEventMode    timerEventMode ≔ {0↦TRUE} ⩤ {1↦TRUE} ⩤ {2↦TRUE} ⩤ {3↦TRUE}
     @timerThreadIdent  timerThreadIdent :∈  TimerIndexDom → Bits
     @timerEnvVec       timerEnvironmentVector :∈  TimerIndexDom → Bits32Nat
     @timerVec          timerVector :∈  TimerIndexDom → Bits32Nat
     @timerValue        timerValue :∈  TimerIndexDom → Bits32Nat
     @timerTriggerValue timerTriggerValue :∈  TimerIndexDom → Bits32Nat
     @timerClockIdent   timerClockIdent :∈  TimerIndexDom → Bits
  end
```

```
event Loading                           extends Loading                           where @false ⊥ end
event RunExternalInputs                 extends RunExternalInputs                 where @false ⊥ end
event SelectNextThread                  extends SelectNextThread                  where @false ⊥ end
event ThreadWaiting                     extends ThreadWaiting                     where @false ⊥ end
event ThreadMpausing                    extends ThreadMpausing                    where @false ⊥ end
event ThreadSpausing                    extends ThreadSpausing                    where @false ⊥ end
event LinkErrorException                extends LinkErrorException                where @false ⊥ end
event IllegalPcException                extends IllegalPcException                where @false ⊥ end
event IllegalInstructionException       extends IllegalInstructionException       where @false ⊥ end
event IllegalResourceException          extends IllegalResourceException          where @false ⊥ end
event LoadStoreException                extends LoadStoreException                where @false ⊥ end
event IllegalPsException                extends IllegalPsException                where @false ⊥ end
event ArithmeticException               extends ArithmeticException               where @false ⊥ end
event EcallException                    extends EcallException                    where @false ⊥ end
event ResourceDepException              extends ResourceDepException              where @false ⊥ end
event KcallException                    extends KcallException                    where @false ⊥ end
event Interrupt                         extends Interrupt                         where @false ⊥ end
event ResourceEvent                     extends ResourceEvent                     where @false ⊥ end
event WriteStatusInstruction            extends WriteStatusInstruction            where @false ⊥ end
event Write1RegisterStatusInstruction   extends Write1RegisterStatusInstruction   where @false ⊥ end
event Write0RegisterInstruction         extends Write0RegisterInstruction         where @false ⊥ end
event Write1RegisterInstruction         extends Write1RegisterInstruction         where @false ⊥ end
event Write2RegisterInstruction         extends Write2RegisterInstruction         where @false ⊥ end
event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  where @false ⊥ end
event Write4Memory1RegisterInstruction  extends Write4Memory1RegisterInstruction  where @false ⊥ end
event WriteThreadRegisterInstruction    extends WriteThreadRegisterInstruction    where @false ⊥ end
event WriteThreadPcInstruction          extends WriteThreadPcInstruction          where @false ⊥ end
event BranchInstruction                 extends BranchInstruction                 where @false ⊥ end
event BranchLinkInstruction             extends BranchLinkInstruction             where @false ⊥ end
event DcallInstruction                  extends DcallInstruction                  where @false ⊥ end
event DretInstruction                   extends DretInstruction                   where @false ⊥ end
event KretInstruction                   extends KretInstruction                   where @false ⊥ end
event RetSpInstruction                  extends RetSpInstruction                  where @false ⊥ end
event PausingInstruction                extends PausingInstruction                where @false ⊥ end

event GETR_No_Timers extends Write1RegisterInstruction
  any resType
  where
```

```
    @resTypeChck resType=ResourceTimer
    @checkFree   CountFreeTimers(timerStatus)=0 // All used up
    @dBits       dBits=NatToBits32(NoResourceId)
end

event GETR_Timers_OK extends Write1RegisterInstruction
  any resType resIndex
  where
    @resTypeChck resType=ResourceTimer
    @checkFree   CountFreeTimers(timerStatus)≠0  // Timer available somewhere
    @resIndex    resIndex=FindFreeTimer(timerStatus)
    @dBits       dBits=GetTimerIdent(resIndex)
  then
    @timerStatus      timerStatus(resIndex) ≔ RUNNING
    @timerThreadIdent timerThreadIdent(resIndex) ≔  NatToBits32(NoResourceId)
end

event FREER_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event FREER_Timer_OK extends Write0RegisterInstruction
  any resIdent resIndex
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @resType      IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
  then
    @statusChange timerStatus(resIndex) ≔ FREE
end

event SETEV_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event SETEV_Timer_OK extends Write0RegisterInstruction
  any resIdent resIndex timerBits
  where
    @resIdentType  resIdent∈Bits
```

```
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @timerBitsType timerBits∈Bits
  then
    @changeEnvVec timerEnvironmentVector(resIndex) :=  Bits32ToNat(timerBits)
end

event SETV_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event SETV_Timer_OK extends Write0RegisterInstruction
  any resIdent resIndex timerBits
  where
    @resIdentType  resIdent∈Bits
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @timerBitsType timerBits∈Bits
  then
    @changeVect timerVector(resIndex) :=  Bits32ToNat(timerBits)
end

event EDU_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event EDU_Timer_OK extends Write0RegisterInstruction
  any resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=NatToBits32(NoResourceId)
  then
    @timerThreadIdent timerThreadIdent(resIndex) :=  threadId
end
```

```
event EEF_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event EEF_Timer_TRUE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c=0
  then
    @changeThread timerThreadIdent(resIndex) ≔  threadId
end

event EEF_Timer_FALSE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=NatToBits32(NoResourceId)
    @cond         c≠0
  then
    @changeThread timerThreadIdent(resIndex) ≔  threadId
end

event EET_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event EET_Timer_TRUE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
```

```
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @threadId      threadId=GetThreadIdent(threadIndex)
    @cond          c≠0
  then
     @changeThread timerThreadIdent(resIndex) ≔  threadId
end

event EET_Timer_FALSE_OK extends Write0RegisterInstruction
  any c resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
    @cond         c=0
  then
     @changeThread timerThreadIdent(resIndex) ≔  threadId
end

event EEU_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event EEU_Timer_OK extends Write0RegisterInstruction
  any resIdent resIndex threadId
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @threadId     threadId=GetThreadIdent(threadIndex)
  then
    @changeThread timerThreadIdent(resIndex) ≔  threadId
end

event SETC_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end
```

```
event SETC_Timer_BadMode extends IllegalResourceException
  any ctrl
  where
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @ctrlType      ctrl∈Bits32Nat
    @afterCheck    ctrl≠CtrlCondAfter
    @noneCheck     ctrl≠CtrlCondNone
    @eventCheck    ctrl≠CtrlIeModeEvent
    @intCheck      ctrl≠CtrlIeModeInterrupt
end

event SETC_Timer_AfterMode_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @ctrlType      ctrl∈Bits32Nat
    @ctrlCheck     ctrl=CtrlCondAfter
  then
    @timerControl timerBlockingMode(resIndex) ≔ TRUE
end

event SETC_Timer_NoneMode_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @ctrlType      ctrl∈Bits32Nat
    @ctrlCheck     ctrl=CtrlCondNone
  then
    @timerControl timerBlockingMode(resIndex) ≔ FALSE
end

event SETC_Timer_Event_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
```

```
    @ctrlType      ctrl∈Bits32Nat
    @ctrlCheck     ctrl=CtrlIeModeEvent
  then
    @timerControl timerEventMode(resIndex) ≔ TRUE
end

event SETC_Timer_Interrupt_OK extends Write0RegisterInstruction
  any resIdent resIndex ctrl
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @ctrlType     ctrl∈Bits32Nat
    @ctrlCheck    ctrl=CtrlIeModeInterrupt
  then
    @timerControl timerEventMode(resIndex) ≔ FALSE
end

event SETD_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event SETD_Timer_OK extends Write0RegisterInstruction
  any resIdent resIndex timerBits
  where
    @resIdentType  resIdent∈Bits
    @typeCheck     IsTimerIdent(resIdent)=TRUE
    @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex      resIndex=GetResourceIndex(resIdent)
    @timerBitsType timerBits∈Bits
  then
    @changeTimeout   timerTriggerValue(resIndex) ≔  Bits32ToNat(timerBits)
    @changeValue     timerValue(resIndex) ≔  0
    @changeStatus    timerStatus(resIndex) ≔ RUNNING
end

event GETD_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end
```

```
event GETD_Timer_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @dBits        dBits=NatToBits32(timerTriggerValue(resIndex))
end

event IN_Timer_NotInUse extends IllegalResourceException
  where
    @typeCheck   IsTimerIdent(resIdent)=TRUE
    @statusCheck IsTimerInUse(resIdent↦timerStatus)=FALSE
end

event IN_Timer_AfterMode_Pausing_OK extends PausingInstruction
  any resIdent resIndex
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @after        timerBlockingMode(timerIndex)=TRUE
    @timing       timerValue(timerIndex)<timerTriggerValue(timerIndex)
end

event IN_Timer_AfterMode_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdentType resIdent∈Bits
    @typeCheck    IsTimerIdent(resIdent)=TRUE
    @statusCheck  IsTimerInUse(resIdent↦timerStatus)=TRUE
    @resIndex     resIndex=GetResourceIndex(resIdent)
    @after        timerBlockingMode(timerIndex)=TRUE
    @timing       timerValue(timerIndex)≥timerTriggerValue(timerIndex)
    @dBits        dBits=NatToBits32(timerValue(resIndex))
end

event IN_Timer_NoneMode_OK extends Write1RegisterInstruction
  any resIdent resIndex
  where
    @resIdentType resIdent∈Bits
```

```
      @typeCheck     IsTimerIdent(resIdent)=TRUE
      @statusCheck   IsTimerInUse(resIdent↦timerStatus)=TRUE
      @resIndex      resIndex=GetResourceIndex(resIdent)
      @after         timerBlockingMode(timerIndex)=FALSE
      @dBits         dBits=NatToBits32(timerValue(resIndex))
  end

  event SelectNextTimer
    then
      @threadIndex timerIndex ≔  (timerIndex + 1) mod MaxTimers
  end

  event IncrementTimer
    where
      @timerStatus timerStatus(timerIndex)=RUNNING
    then
      @changeTimer timerValue(timerIndex) ≔  ModuloAddUnsigned(timerValue(timerIndex)↦1)
  end

  event TimerEvent extends ResourceEvent
    where
      @timerStatus timerStatus(timerIndex)=RUNNING
      @timerEvent  timerEventMode(timerIndex)=TRUE
      @valueCheck  timerValue(timerIndex)=timerTriggerValue(timerIndex)
      @threadIdent timerThreadIdent(timerIndex)≠NatToBits32(NoResourceId)
      @threadIndex triggeredThreadIndex=GetResourceIndex(timerThreadIdent(timerIndex))
      @edBits      edBits=NatToBits32(timerEnvironmentVector(timerIndex))
      @nextPc      nextPc=timerVector(timerIndex)
  end

  event TimerInterrupt extends Interrupt
    where
      @timerStatus timerStatus(timerIndex)=RUNNING
      @timerEvent  timerEventMode(timerIndex)=FALSE
      @valueCheck  timerValue(timerIndex)=timerTriggerValue(timerIndex)
      @threadIdent timerThreadIdent(timerIndex)≠NatToBits32(NoResourceId)
      @threadIndex triggeredThreadIndex=GetResourceIndex(timerThreadIdent(timerIndex))
      @edBits      edBits=NatToBits32(timerEnvironmentVector(timerIndex))
      @nextPc      nextPc=timerVector(timerIndex)
  end

end
```

14.1.35machine XCoreIsaTimerFinal

```
machine XCoreIsaTimerFinal // Introduce instruction decoding

refines XCoreIsaTimer1

sees XCoreBasic XCoreTimers XCoreClocks

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray

          timerIndex
          timerStatus timerBlockingMode timerEventMode timerThreadIdent timerEnvironmentVector timerVector
          timerClockIdent timerValue timerTriggerValue

events

  event INITIALISATION extends INITIALISATION end

  event GETR_Rus_No_Timers extends GETR_No_Timers
    where
      @instCheck IsRus(instr↦16↦0)=TRUE
      @op1Fetch  d=XRus_op1(instr)
      @op2Fetch  resType=XRus_op2(instr)
      @instrSize instrSize=InstSizeShort+0
  end

  event GETR_Rus_Timers_OK extends GETR_Timers_OK
    where
      @instCheck   IsRus(instr↦16↦0)=TRUE
      @op1Fetch    d=XRus_op1(instr)
      @op2Fetch    resType=XRus_op2(instr)
      @instrSize   instrSize=InstSizeShort+0
  end

  event FREER_1r_Timer_NotInUse extends FREER_Timer_NotInUse
    any op1
    where
      @instCheck  Is1r(instr↦2↦0)=TRUE
      @op1Fetch   op1=X1r_op1(instr)
      @resIdent   resIdent=(threadRegArrays(threadIndex))(op1)
  end

  event FREER_1r_Timer_OK extends FREER_Timer_OK
```

```
  any op1
  where
    @instCheck Is1r(instr↦2↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event EDU_1r_Timer_NotInUse extends EDU_Timer_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦0↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EDU_1r_Timer_OK extends EDU_Timer_OK
  any op1
  where
    @instCheck Is1r(instr↦0↦0)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event EEF_2r_Timer_NotInUse extends EEF_Timer_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event EEF_2r_Timer_TRUE_OK extends EEF_Timer_TRUE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EEF_2r_Timer_FALSE_OK extends EEF_Timer_FALSE_OK
```

```
  any op1 op2
  where
    @instCheck Is2r(instr↦5↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EET_2r_Timer_NotInUse extends EET_Timer_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event EET_2r_Timer_TRUE_OK extends EET_Timer_TRUE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EET_2r_Timer_FALSE_OK extends EET_Timer_FALSE_OK
  any op1 op2
  where
    @instCheck Is2r(instr↦4↦1)=TRUE
    @op1Fetch  op1=X2r_op1(instr)
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @c         c=Bits32ToNat((threadRegArrays(threadIndex))(op1))
    @instrSize instrSize=InstSizeShort+0
end

event EEU_1r_Timer_NotInUse extends EEU_Timer_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦0↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event EEU_1r_Timer_OK extends EEU_Timer_OK
  any op1
  where
    @instCheck Is1r(instr↦0↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Timer_NotInUse extends SETC_Timer_NotInUse
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Ru6_Timer_BadMode extends SETC_Timer_BadMode
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Ru6_Timer_AfterMode_OK extends SETC_Timer_AfterMode_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Timer_NoneMode_OK extends SETC_Timer_NoneMode_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
```

```
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Timer_Event_OK extends SETC_Timer_Event_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Ru6_Timer_Interrupt_OK extends SETC_Timer_Interrupt_OK
  any op1
  where
    @instCheck IsRu6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XRu6_op1(instr)
    @op2Fetch  ctrl=XRu6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeShort+0
end

event SETC_Lru6_Timer_NotInUse extends SETC_Timer_NotInUse
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Lru6_Timer_BadMode extends SETC_Timer_BadMode
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETC_Lru6_Timer_AfterMode_OK extends SETC_Timer_AfterMode_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
```

```
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Timer_NoneMode_OK extends SETC_Timer_NoneMode_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Timer_Event_OK extends SETC_Timer_Event_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_Lru6_Timer_Interrupt_OK extends SETC_Timer_Interrupt_OK
  any op1
  where
    @instCheck IsLru6(instr↦29↦0)=TRUE
    @op1Fetch  op1=XLru6_op1(instr)
    @op2Fetch  ctrl=XLru6_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Timer_NotInUse extends SETC_Timer_NotInUse
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end
```

```
event SETC_L2r_Timer_BadMode extends SETC_Timer_BadMode
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
end

event SETC_L2r_Timer_AfterMode_OK extends SETC_Timer_AfterMode_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Timer_NoneMode_OK extends SETC_Timer_NoneMode_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Timer_Event_OK extends SETC_Timer_Event_OK
  any op1 op2
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETC_L2r_Timer_Interrupt_OK extends SETC_Timer_Interrupt_OK
  any op1 op2
```

```
  where
    @instCheck IsL2r(instr↦5↦1↦12)=TRUE
    @op1Fetch  op1=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @ctrl      ctrl=Bits32ToNat((threadRegArrays(threadIndex))(op2))
    @instrSize instrSize=InstSizeLong+0
end

event SETD_R2r_Timer_NotInUse extends SETD_Timer_NotInUse
  any op1
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETD_R2r_Timer_OK extends SETD_Timer_OK
  any op1 op2
  where
    @instCheck IsR2r(instr↦2↦1)=TRUE
    @op1Fetch  op1=XR2r_op1(instr)
    @op2Fetch  op2=XR2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @timerBits timerBits=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeShort+0
end

event GETD_L2r_Timer_NotInUse extends GETD_Timer_NotInUse
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end

event GETD_L2r_Timer_OK extends GETD_Timer_OK
  any op2
  where
    @instCheck IsL2r(instr↦3↦1↦12)=TRUE
    @op1Fetch  d=XL2r_op1(instr)
    @op2Fetch  op2=XL2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
    @instrSize instrSize=InstSizeLong+0
end
```

```
event SETEV_1r_Timer_NotInUse extends SETEV_Timer_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦7↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETEV_1r_Timer_OK extends SETEV_Timer_OK
  any op1
  where
    @instCheck Is1r(instr↦7↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @timerBits timerBits=(threadRegArrays(threadIndex))(R11Index)
    @instrSize instrSize=InstSizeShort+0
end

event SETV_1r_Timer_NotInUse extends SETV_Timer_NotInUse
  any op1
  where
    @instCheck Is1r(instr↦8↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
end

event SETV_1r_Timer_OK extends SETV_Timer_OK
  any op1
  where
    @instCheck Is1r(instr↦8↦1)=TRUE
    @op1Fetch  op1=X1r_op1(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op1)
    @timerBits timerBits=(threadRegArrays(threadIndex))(R11Index)
    @instrSize instrSize=InstSizeShort+0
end

event IN_2r_Timer_NotInUse extends IN_Timer_NotInUse
  any op2
  where
    @instCheck Is2r(instr↦22↦0)=TRUE
    @op2Fetch  op2=X2r_op2(instr)
    @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
end
```

```
  event IN_2r_Timer_AfterMode_Pausing_OK extends IN_Timer_AfterMode_Pausing_OK
    any op2
    where
      @instCheck Is2r(instr↦22↦0)=TRUE
      @op2Fetch  op2=X2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
  end

  event IN_2r_Timer_AfterMode_OK extends IN_Timer_AfterMode_OK
    any op2
    where
      @instCheck Is2r(instr↦22↦0)=TRUE
      @op1Fetch  d=X2r_op1(instr)
      @op2Fetch  op2=X2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
      @instrSize instrSize=InstSizeShort+0
  end

  event IN_2r_Timer_NoneMode_OK extends IN_Timer_NoneMode_OK
    any op2
    where
      @instCheck Is2r(instr↦22↦0)=TRUE
      @op1Fetch  d=X2r_op1(instr)
      @op2Fetch  op2=X2r_op2(instr)
      @resIdent  resIdent=(threadRegArrays(threadIndex))(op2)
      @instrSize instrSize=InstSizeShort+0
  end

  event SelectNextTimer extends SelectNextTimer end
  event IncrementTimer  extends IncrementTimer end
  event TimerEvent      extends TimerEvent end
  event TimerInterrupt  extends TimerInterrupt end

end
```

14.1.36machine XCoreSupportMch

```
machine XCoreSupportMch // Handling of events outside decoded instructions

refines XCoreFinal

sees XCoreBasic Loader

variables status interruptTriggered resourceTriggered
          threadIndex threadStatus threadRegArrays threadPc
          memArray
          loadPtr

invariants
  @loadPtrType loadPtr ∈ 0‥MemArraySize

events

  event INITIALISATION extends INITIALISATION
    then
      @loadPtr loadPtr ≔ 0
  end

  event ThreadMpausing                   extends ThreadMpausing                   where @false ⊥ end
  event ThreadSpausing                   extends ThreadSpausing                   where @false ⊥ end
  event LinkErrorException               extends LinkErrorException               where @false ⊥ end
  event IllegalPcException               extends IllegalPcException               where @false ⊥ end
  event IllegalInstructionException      extends IllegalInstructionException      where @false ⊥ end
  event IllegalResourceException         extends IllegalResourceException         where @false ⊥ end
  event LoadStoreException               extends LoadStoreException               where @false ⊥ end
  event IllegalPsException               extends IllegalPsException               where @false ⊥ end
  event ArithmeticException              extends ArithmeticException              where @false ⊥ end
  event EcallException                   extends EcallException                   where @false ⊥ end
  event ResourceDepException             extends ResourceDepException             where @false ⊥ end
  event KcallException                   extends KcallException                   where @false ⊥ end
  event Interrupt                        extends Interrupt                        where @false ⊥ end
  event ResourceEvent                    extends ResourceEvent                    where @false ⊥ end
  event WriteStatusInstruction           extends WriteStatusInstruction           where @false ⊥ end
  event Write1RegisterStatusInstruction  extends Write1RegisterStatusInstruction  where @false ⊥ end
  event Write0RegisterInstruction        extends Write0RegisterInstruction        where @false ⊥ end
  event Write1RegisterInstruction        extends Write1RegisterInstruction        where @false ⊥ end
  event Write2RegisterInstruction        extends Write2RegisterInstruction        where @false ⊥ end
```

```
event WriteSrStatusInstruction          extends WriteSrStatusInstruction          where @false ⊥ end
event Write1Memory0RegisterInstruction  extends Write1Memory0RegisterInstruction  where @false ⊥ end
event Write2Memory0RegisterInstruction  extends Write2Memory0RegisterInstruction  where @false ⊥ end
event Write4Memory0RegisterInstruction  extends Write4Memory0RegisterInstruction  where @false ⊥ end
event Write4Memory1RegisterInstruction  extends Write4Memory1RegisterInstruction  where @false ⊥ end
event WriteThreadRegisterInstruction    extends WriteThreadRegisterInstruction    where @false ⊥ end
event WriteThreadPcInstruction          extends WriteThreadPcInstruction          where @false ⊥ end
event BranchInstruction                 extends BranchInstruction                 where @false ⊥ end
event BranchLinkInstruction             extends BranchLinkInstruction             where @false ⊥ end
event DcallInstruction                  extends DcallInstruction                  where @false ⊥ end
event DretInstruction                   extends DretInstruction                   where @false ⊥ end
event KretInstruction                   extends KretInstruction                   where @false ⊥ end
event RetSpInstruction                  extends RetSpInstruction                  where @false ⊥ end
event PausingInstruction                extends PausingInstruction                where @false ⊥ end

event Loading refines Loading
  where
    @statusCheck status = LOADING
    @loadPtrCheck loadPtr ∈ 0‥(MemArraySize-1)
  then
    @loadMem memArray(MinMemIndex + loadPtr) ≔ FetchImageData(loadPtr)
    @incLoadPtr loadPtr ≔ loadPtr + 1
end

event LoadComplete refines Loading
  where
    @statusCheck status = LOADING
    @loadPtrCheck loadPtr ∉ 0‥(MemArraySize-1)
  then
    @incStatus status ≔ RUNNING
end

event RunExternalInputs extends RunExternalInputs
end

event SelectNextThread refines SelectNextThread
  where
    @checkStat status = RUNNING
  then
    @threadIndex threadIndex ≔  (threadIndex + 1) mod MaxThreads
end

event ThreadWaiting extends ThreadWaiting
```

```
  end

  event OutOfRangePc extends IllegalPcException
    where
      @pcRangeCheck threadPc(threadIndex) ∉ XCorePcDom
  end

  event MisalignedPc extends IllegalPcException
    where
      @pcAlignCheck threadPc(threadIndex) mod 2 ≠ 0
  end

  event UnknownInstruction extends IllegalInstructionException
    where
      @instCheck  Is0r(instr↦0↦0↦0)=TRUE        // Example - to be extended to all undecoded instructions
  end

end
```

14.2 Contexts

14.2.1 context ArithmeticTheorems

```
context ArithmeticTheorems // Container for arithmetic axioms that should be theorems discharged by provers

axioms

    @minusGtz        ∀x,y·x∈ℤ ∧ y∈ℤ ∧ y<x ⇒  (x−y)>0
    @minusNat        ∀x,y·x∈ℤ ∧ y∈ℤ ∧ y<x ⇒  (x−y)∈ℕ
    @minusInSet  ∀S,x,y·S⊆ℕ ∧ x∈S ∧ y∈S ∧y≤x ⇒(x−y)∈S

    @rangeHasRange ∀A,a1,a2,B·A=a1‥a2 ∧ a1∈B ∧  a2∈B ⇒ A⊆B

end
```

14.2.2 context OvrTheorems

```
context OvrTheorems // Container for override axioms that should be theorems discharged by provers

extends ThreadArray RegArray MemArrayDomE4

axioms

   theorem
   @ovrNat1  ∀ a,I,D,i1,d1 ·
               I⊆ℕ ∧ D⊆ℕ ∧
               a∈I → D ∧ i1∈I ∧ d1∈D ⇒
               a□{i1 ↦ d1}∈I → D

   theorem
   @ovrNat2  ∀ a,I,D,i1,d1,i2,d2 ·
               I⊆ℕ ∧ D⊆ℕ ∧
               a∈I → D ∧
               i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ⇒
               a□{i1 ↦ d1}□{i2 ↦ d2}∈I → D

   @ovrNat4  ∀ a,I,D,i1,d1,i2,d2,i3,d3,i4,d4 ·
              I⊆ℕ ∧ D⊆ℕ ∧
              a∈I → D ∧
              i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
              i3∈I ∧ d3∈D ∧ i4∈I ∧ d4∈D
               ⇒
              a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}∈I → D

    @ovrNat8  ∀ a,I,D,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6,i7,d7,i8,d8 ·
              I⊆ℕ ∧ D⊆ℕ ∧
              a∈I → D ∧
              i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
              i3∈I ∧ d3∈D ∧ i4∈I ∧ d4∈D ∧
              i5∈I ∧ d5∈D ∧ i6∈I ∧ d6∈D ∧
              i7∈I ∧ d7∈D ∧ i8∈I ∧ d8∈D
              ⇒
              a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□{i7 ↦ d7}□{i8 ↦ d8}∈I → D

    @ovrNat12   ∀ a,I,D,i1,d1,i2,d2,i3,d3,i4,d4, i5,d5,i6,d6,i7,d7,i8,d8, i9,d9,i10,d10,i11,d11,i12,d12 ·
               I⊆ℕ ∧ D⊆ℕ ∧
```

```
            a∈I → D ∧
            i1∈I  ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
            i3∈I  ∧ d3∈D ∧ i4∈I ∧ d4∈D ∧
            i5∈I  ∧ d5∈D ∧ i6∈I ∧ d6∈D ∧
            i7∈I  ∧ d7∈D ∧ i8∈I  ∧ d8∈D ∧
            i9∈I  ∧ d9∈D ∧ i10∈I ∧ d10∈D ∧
            i11∈I ∧ d11∈D ∧ i12∈I ∧ d12∈D
                  ⇒
            a⨆{i1 ↦ d1}⨆{i2 ↦ d2}⨆{i3 ↦ d3}⨆{i4 ↦ d4}⨆{i5 ↦ d5}⨆{i6 ↦ d6}⨆
               {i7 ↦ d7}⨆{i8 ↦ d8}⨆{i9 ↦ d9}⨆{i10 ↦ d10}⨆{i11 ↦ d11}⨆{i12 ↦ d12}∈I → D

@ovrENat1   ∀ I,D,i1,d1 ·
            I⊆ℕ ∧ D⊆ℕ ∧
            i1∈I ∧ d1∈D
            ⇒
            {i1 ↦ d1}∈I → D

@ovrENat4   ∀ I,D,i1,d1,i2,d2,i3,d3,i4,d4 ·
            I⊆ℕ ∧ D⊆ℕ ∧
            i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
            i3∈I ∧ d3∈D ∧ i4∈I ∧ d4∈D
            ⇒
            {i1 ↦ d1}⨆{i2 ↦ d2}⨆{i3 ↦ d3}⨆{i4 ↦ d4}∈I → D

@ovrENat6   ∀ I,D,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6 ·
            I⊆ℕ ∧ D⊆ℕ ∧
            i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
            i3∈I ∧ d3∈D ∧ i4∈I ∧ d4∈D ∧
            i5∈I ∧ d5∈D ∧ i6∈I ∧ d6∈D
            ⇒
            {i1 ↦ d1}⨆{i2 ↦ d2}⨆{i3 ↦ d3}⨆{i4 ↦ d4}⨆{i5 ↦ d5}⨆{i6 ↦ d6}∈I → D

@ovrENat7    ∀ I,D,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6,i7,d7,i8,d8 ·
             I⊆ℕ ∧ D⊆ℕ ∧
             i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
             i3∈I ∧ d3∈D ∧ i4∈I ∧ d4∈D ∧
             i5∈I ∧ d5∈D ∧ i6∈I ∧ d6∈D ∧
             i7∈I ∧ d7∈D ∧ i8∈I ∧ d8∈D
             ⇒
             {i1 ↦ d1}⨆{i2 ↦ d2}⨆{i3 ↦ d3}⨆{i4 ↦ d4}⨆{i5 ↦ d5}⨆{i6 ↦ d6}⨆{i7 ↦ d7}⨆{i8 ↦ d8}∈I → D

@ovrENat8    ∀ I,D,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6,i7,d7 ·
```

```
            I⊆ℕ ∧ D⊆ℕ ∧
            i1∈I ∧ d1∈D ∧ i2∈I ∧ d2∈D ∧
            i3∈I ∧ d3∈D ∧ i4∈I ∧ d4∈D ∧
            i5∈I ∧ d5∈D ∧ i6∈I ∧ d6∈D ∧
            i7∈I ∧ d7∈D
            ⇒
            {i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□{i7 ↦ d7}∈I → D

@ovrENat12  ∀ I,D,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6,i7,d7,i8,d8,i9,d9,i10,d10,i11,d11,i12,d12 ·
            I⊆ℕ ∧ D⊆ℕ ∧
            i1∈I    ∧ d1∈D   ∧ i2∈I   ∧ d2∈D ∧
            i3∈I    ∧ d3∈D   ∧ i4∈I   ∧ d4∈D ∧
            i5∈I    ∧ d5∈D   ∧ i6∈I   ∧ d6∈D ∧
            i7∈I    ∧ d7∈D   ∧ i8∈I   ∧ d8∈D ∧
            i9∈I    ∧ d9∈D   ∧ i10∈I ∧ d10∈D ∧
            i11∈I ∧ d11∈D ∧ i12∈I ∧ d12∈D
            ⇒
           {i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□
           {i7 ↦ d7}□{i8 ↦ d8}□{i9 ↦ d9}□{i10 ↦ d10}□{i11 ↦ d11}□{i12 ↦ d12}∈I → D

@ovrENat32  ∀ D,d ·
            D⊆ℕ ∧d∈D
            ⇒
           {0 ↦ d}□{1 ↦ d}□{2↦ d}□{3↦ d}□{4↦ d}□{5↦ d}□{6↦ d}□{7↦ d}□
           {8↦ d}□{9↦ d}□{10↦ d}□{11↦ d}□{12↦ d}□{13↦ d}□{14↦ d}□{15↦ d}□
           {16↦ d}□{17↦ d}□{18↦ d}□{19↦ d}□{20↦ d}□{21↦ d}□{22↦ d}□{23↦ d}□
           {24↦ d}□{25↦ d}□{26↦ d}□{27↦ d}□{28↦ d}□{29↦ d}□{30↦ d}□{31↦ d}∈0‥31 → D

@natArraysOvr1 ∀I,J,T,N,i,a,a1·
               I⊆ℕ ∧J⊆ℕ ∧N⊆ℕ ∧ T=(J→N) ∧
               i∈I ∧ a∈I→T ∧  a1∈T
               ⇒a□{i↦a1}∈I→T

@natArraysOvr7 ∀I,J,T,N,i,a,a1,a2,a3,a4,a5,a6,a7·
               I⊆ℕ ∧J⊆ℕ ∧N⊆ℕ ∧ T=(J→N) ∧
               i∈I ∧ a∈I→T ∧  a1∈T ∧  a2∈T ∧  a3∈T ∧  a4∈T ∧  a5∈T ∧  a6∈T ∧  a7∈T
               ⇒
               a□{i↦a1}□{i↦a2}□{i↦a3}□{i↦a4}□{i↦a5}□{i↦a6}□{i↦a7}∈I→T

@natArraysOvr7by8  ∀J,N,T,a1,a2,a3,a4,a5,a6,a7,a8·
                   J⊆ℕ ∧N⊆ℕ ∧ T=(J→N) ∧
                   a1∈T ∧  a2∈T ∧  a3∈T ∧  a4∈T ∧  a5∈T ∧  a6∈T ∧  a7∈T ∧  a8∈T
```

```
⇒
{0 ↦ a1}⩌{0 ↦ a2}⩌{0 ↦ a3}⩌{0 ↦ a4}⩌{0 ↦ a5}⩌{0 ↦ a6}⩌{0 ↦ a7}⩌{0 ↦ a8}⩌
{1 ↦ a1}⩌{1 ↦ a2}⩌{1 ↦ a3}⩌{1 ↦ a4}⩌{1 ↦ a5}⩌{1 ↦ a6}⩌{1 ↦ a7}⩌{1 ↦ a8}⩌
{2 ↦ a1}⩌{2 ↦ a2}⩌{2 ↦ a3}⩌{2 ↦ a4}⩌{2 ↦ a5}⩌{2 ↦ a6}⩌{2 ↦ a7}⩌{2 ↦ a8}⩌
{3 ↦ a1}⩌{3 ↦ a2}⩌{3 ↦ a3}⩌{3 ↦ a4}⩌{3 ↦ a5}⩌{3 ↦ a6}⩌{3 ↦ a7}⩌{3 ↦ a8}⩌
{4 ↦ a1}⩌{4 ↦ a2}⩌{4 ↦ a3}⩌{4 ↦ a4}⩌{4 ↦ a5}⩌{4 ↦ a6}⩌{4 ↦ a7}⩌{4 ↦ a8}⩌
{5 ↦ a1}⩌{5 ↦ a2}⩌{5 ↦ a3}⩌{5 ↦ a4}⩌{5 ↦ a5}⩌{5 ↦ a6}⩌{5 ↦ a7}⩌{5 ↦ a8}⩌
{6 ↦ a1}⩌{6 ↦ a2}⩌{6 ↦ a3}⩌{6 ↦ a4}⩌{6 ↦ a5}⩌{6 ↦ a6}⩌{6 ↦ a7}⩌{6 ↦ a8}∈0‥6→T
```

theorem

@ovrBool1
```
∀ a,I,i1,d1 ·
I⊆ℕ ∧
a∈I → BOOL ∧
i1∈I ∧ d1∈BOOL
⇒
a⩌{i1 ↦ d1}∈I → BOOL
```

theorem

@ovrBool2
```
∀ a,I,i1,d1,i2,d2 ·
I⊆ℕ ∧
a∈I → BOOL ∧
i1∈I ∧ d1∈BOOL ∧ i2∈I ∧ d2∈BOOL
⇒
a⩌{i1 ↦ d1}⩌{i2 ↦ d2}∈I → BOOL
```

@ovrBool4
```
∀ a,I,i1,d1,i2,d2,i3,d3,i4,d4 ·
I⊆ℕ ∧
a∈I → BOOL ∧
i1∈I ∧ d1∈BOOL ∧ i2∈I ∧ d2∈BOOL ∧
i3∈I ∧ d3∈BOOL ∧ i4∈I ∧ d4∈BOOL
⇒
a⩌{i1 ↦ d1}⩌{i2 ↦ d2}⩌{i3 ↦ d3}⩌{i4 ↦ d4}∈I → BOOL
```

@ovrBool8
```
∀ a,I,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6,i7,d7,i8,d8 ·
I⊆ℕ ∧
a∈I → BOOL ∧
i1∈I ∧ d1∈BOOL ∧ i2∈I ∧ d2∈BOOL ∧
i3∈I ∧ d3∈BOOL ∧ i4∈I ∧ d4∈BOOL ∧
i5∈I ∧ d5∈BOOL ∧ i6∈I ∧ d6∈BOOL ∧
i7∈I ∧ d7∈BOOL ∧ i8∈I ∧ d8∈BOOL
⇒
```

```
            a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□{i7 ↦ d7}□{i8 ↦ d8}∈I → BOOL

@ovrEBool1  ∀ I,i1,d1 ·
             I⊆ℕ ∧
             i1∈I ∧ d1∈BOOL
             ⇒
             {i1 ↦ d1}∈I → BOOL

@ovrEBool2   ∀ I,i1,d1,i2,d2 ·
             I⊆ℕ ∧
             i1∈I ∧ d1∈BOOL ∧ i2∈I ∧ d2∈BOOL
             ⇒
             {i1 ↦ d1}□{i2 ↦ d2}∈I → BOOL

@ovrEBool4   ∀ I,i1,d1,i2,d2,i3,d3,i4,d4 ·
             I⊆ℕ ∧
             i1∈I ∧ d1∈BOOL ∧ i2∈I ∧ d2∈BOOL ∧
             i3∈I ∧ d3∈BOOL ∧ i4∈I ∧ d4∈BOOL
             ⇒
             {i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}∈I → BOOL

@ovrEBool32  ∀ d ·
             d∈BOOL
             ⇒
             {0 ↦ d}□{1 ↦ d}□{2↦ d}□{3↦ d}□{4↦ d}□{5↦ d}□{6↦ d}□{7↦ d}□
             {8↦ d}□{9↦ d}□{10↦ d}□{11↦ d}□{12↦ d}□{13↦ d}□{14↦ d}□{15↦ d}□
             {16↦ d}□{17↦ d}□{18↦ d}□{19↦ d}□{20↦ d}□{21↦ d}□{22↦ d}□{23↦ d}□
             {24↦ d}□{25↦ d}□{26↦ d}□{27↦ d}□{28↦ d}□{29↦ d}□{30↦ d}□{31↦ d}∈0‥31 → BOOL

@boolArraysOvr1 ∀I,J,T,i,a,a1·
                I⊆ℕ ∧J⊆ℕ ∧T=(J→BOOL) ∧
                i∈I ∧ a∈I→T ∧  a1∈T
                ⇒a□{i↦a1}∈I→T

@boolArraysOvr4 ∀I,J,T,i,a,a1,a2,a3,a4·
                I⊆ℕ ∧J⊆ℕ ∧T=(J→BOOL) ∧
                i∈I ∧ a∈I→T ∧  a1∈T ∧  a2∈T ∧  a3∈T ∧  a4∈T
                ⇒a□{i↦a1}□{i↦a2}□{i↦a3}□{i↦a4}∈I→T


@ovrBits1    ∀ a,I,i1,d1 ·
             I⊆ℕ ∧
```

```
            a∈I → Bits ∧
            i1∈I ∧ d1∈Bits
            ⇒
            a□{i1 ↦ d1}∈I → Bits

@ovrBits2   ∀ I,a,i1,d1,i2,d2 ·
            I⊆ℕ ∧
            a∈I → Bits ∧
            i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits
            ⇒
            a□{i1 ↦ d1}□{i2 ↦ d2}∈I → Bits

@ovrBits3   ∀ I,a,i1,d1,i2,d2,i3,d3 ·
            I⊆ℕ ∧
            a∈I → Bits ∧
            i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits ∧ i3∈I ∧ d3∈Bits
            ⇒
            a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}∈I → Bits

@ovrBits4   ∀ I,a,i1,d1,i2,d2,i3,d3,i4,d4 ·
            I⊆ℕ ∧
            a∈I → Bits ∧
            i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits ∧
            i3∈I ∧ d3∈Bits ∧ i4∈I ∧ d4∈Bits
            ⇒
            a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}∈I → Bits

@ovrBits6   ∀ I,a,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6 ·
            I⊆ℕ ∧
            a∈I → Bits ∧
            i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits ∧
            i3∈I ∧ d3∈Bits ∧ i4∈I ∧ d4∈Bits ∧
            i5∈I ∧ d5∈Bits ∧ i6∈I ∧ d6∈Bits
            ⇒
            a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}∈I → Bits

@ovrEBits32  ∀ d ·
             d∈Bits
             ⇒
            {0 ↦ d}□{1 ↦ d}□{2↦ d}□{3↦ d}□{4↦ d}□{5↦ d}□{6↦ d}□{7↦ d}□
            {8↦ d}□{9↦ d}□{10↦ d}□{11↦ d}□{12↦ d}□{13↦ d}□{14↦ d}□{15↦ d}□
            {16↦ d}□{17↦ d}□{18↦ d}□{19↦ d}□{20↦ d}□{21↦ d}□{22↦ d}□{23↦ d}□
```

```
            {24↦ d}⩀{25↦ d}⩀{26↦ d}⩀{27↦ d}⩀{28↦ d}⩀{29↦ d}⩀{30↦ d}⩀{31↦ d}∈0‥31 → Bits

@bitArraysOvr1 ∀I,J,T,i,a,a1·
               I⊆ℕ ∧J⊆ℕ ∧T=(J→Bits) ∧
               i∈I ∧ a∈I→T ∧  a1∈T
               ⇒a⩀{i↦a1}∈I→T

@bitArraysOvr2 ∀I,J,T,i,a,a1,a2·
               I⊆ℕ ∧J⊆ℕ ∧T=(J→Bits) ∧
               i∈I ∧ a∈I→T ∧  a1∈T ∧  a2∈T
               ⇒a⩀{i↦a1}⩀{i↦a2}∈I→T

@bitArraysOvr4 ∀I,J,T,i,a,a1,a2,a3,a4·
               I⊆ℕ ∧J⊆ℕ ∧T=(J→Bits) ∧
               i∈I ∧ a∈I→T ∧  a1∈T ∧  a2∈T ∧  a3∈T ∧  a4∈T
               ⇒a⩀{i↦a1}⩀{i↦a2}⩀{i↦a3}⩀{i↦a4}∈I→T

@bitArraysOvr6 ∀I,J,T,i,a,a1,a2,a3,a4,a5,a6·
               I⊆ℕ ∧J⊆ℕ ∧T=(J→Bits) ∧
               i∈I ∧ a∈I→T ∧  a1∈T ∧  a2∈T ∧  a3∈T ∧  a4∈T ∧  a5∈T ∧  a6∈T
               ⇒a⩀{i↦a1}⩀{i↦a2}⩀{i↦a3}⩀{i↦a4}⩀{i↦a5}⩀{i↦a6}∈I→T

@ovrEBits1    ∀ I,i1,d1 ·
              I⊆ℕ ∧
              i1∈I ∧ d1∈Bits
              ⇒
             {i1 ↦ d1}∈I → Bits

@ovrEBits2    ∀ I,i1,d1,i2,d2 ·
              I⊆ℕ ∧
              i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits
              ⇒
              {i1 ↦ d1}⩀{i2 ↦ d2}∈I → Bits

@ovrEBits4    ∀ I,i1,d1,i2,d2,i3,d3,i4,d4 ·
              I⊆ℕ ∧
              i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits ∧
              i3∈I ∧ d3∈Bits ∧ i4∈I ∧ d4∈Bits
              ⇒
             {i1 ↦ d1}⩀{i2 ↦ d2}⩀{i3 ↦ d3}⩀{i4 ↦ d4}∈I → Bits

@ovrEBits6    ∀ I,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6 ·
```

```
                I⊆ℕ ∧
                i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits ∧
                i3∈I ∧ d3∈Bits ∧ i4∈I ∧ d4∈Bits ∧
                i5∈I ∧ d5∈Bits ∧ i6∈I ∧ d6∈Bits
                ⇒
                {i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}∈I → Bits

  @ovrEBits8    ∀ I,i1,d1,i2,d2,i3,d3,i4,d4,i5,d5,i6,d6,i7,d7,i8,d8 ·
                I⊆ℕ ∧
                i1∈I ∧ d1∈Bits ∧ i2∈I ∧ d2∈Bits ∧
                i3∈I ∧ d3∈Bits ∧ i4∈I ∧ d4∈Bits ∧
                i5∈I ∧ d5∈Bits ∧ i6∈I ∧ d6∈Bits ∧
                i7∈I ∧ d7∈Bits ∧ i8∈I ∧ d8∈Bits
                ⇒
               {i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□{i7 ↦ d7}□{i8 ↦ d8}∈I → Bits

theorem @memArrayOvr1 ∀a,i1,d1·a∈(ExtMemArrayDom→Bits) ∧ i1∈ExtMemArrayDom ∧ d1∈Bits
                       ⇒ a□{i1 ↦ d1}∈ExtMemArrayDom → Bits

theorem @memArrayOvr2 ∀a,i1,i2,d1,d2·a∈(ExtMemArrayDom→Bits) ∧ i1∈ExtMemArrayDom ∧ d1∈Bits ∧ i2∈ExtMemArrayDom ∧ d2∈Bits
                       ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}∈ExtMemArrayDom → Bits

theorem @memArrayOvr4 ∀a,i1,i2,i3,i4,d1,d2,d3,d4·a∈(ExtMemArrayDom→Bits) ∧
                        i1∈ExtMemArrayDom ∧ d1∈Bits ∧
                        i2∈ExtMemArrayDom ∧ d2∈Bits ∧
                        i3∈ExtMemArrayDom ∧ d3∈Bits ∧
                        i4∈ExtMemArrayDom ∧ d4∈Bits
                        ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}∈ExtMemArrayDom → Bits

theorem
@regArrayOvr1 ∀ti,a,a1·
              ti∈ThreadIndexDom ∧
              a∈(ThreadIndexDom→RegArray) ∧
              a1∈RegArray
              ⇒a□{ti↦a1}∈ThreadIndexDom→RegArray

theorem
@regArrayOvr2 ∀ti,a,a1,a2·
              ti∈ThreadIndexDom ∧
              a∈(ThreadIndexDom→RegArray) ∧
              a1∈RegArray ∧ a2∈RegArray
              ⇒a□{ti↦a1}□{ti↦a2}∈ThreadIndexDom→RegArray
```

```
theorem
@regArrayOvr4 ∀ti,a,a1,a2,a3,a4·
              ti∈ThreadIndexDom ∧
              a∈(ThreadIndexDom→RegArray) ∧
              a1∈RegArray ∧ a2∈RegArray ∧
              a3∈RegArray ∧ a4∈RegArray
              ⇒a□{ti↦a1}□{ti↦a2}□{ti↦a3}□{ti↦a4}∈ThreadIndexDom→RegArray

theorem
@regArrayOvr6 ∀ti,a,a1,a2,a3,a4,a5,a6·
              ti∈ThreadIndexDom ∧
              a∈(ThreadIndexDom→RegArray) ∧
              a1∈RegArray ∧ a2∈RegArray ∧ a3∈RegArray ∧
              a4∈RegArray ∧ a5∈RegArray ∧ a6∈RegArray
              ⇒a□{ti↦a1}□{ti↦a2}□{ti↦a3}□{ti↦a4}□{ti↦a5}□{ti↦a6}∈ThreadIndexDom→RegArray

theorem
@threadStatusOvr1 ∀a,i1,d1·a∈(ThreadIndexDom→ThreadStatus) ∧
                  i1∈ThreadIndexDom ∧ d1∈ThreadStatus
                   ⇒ a□{i1 ↦ d1}∈ThreadIndexDom → ThreadStatus

theorem
@threadStatusOvr2 ∀a,i1,i2,d1,d2·a∈(ThreadIndexDom→ThreadStatus) ∧
                        i1∈ThreadIndexDom ∧ d1∈ThreadStatus ∧ i2∈ThreadIndexDom ∧ d2∈ThreadStatus
                        ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}∈ThreadIndexDom → ThreadStatus

theorem @threadStatusOvr4 ∀a,i1,i2,i3,i4,d1,d2,d3,d4·a∈(ThreadIndexDom→ThreadStatus) ∧
                          i1∈ThreadIndexDom ∧ d1∈ThreadStatus ∧
                          i2∈ThreadIndexDom ∧ d2∈ThreadStatus ∧
                          i3∈ThreadIndexDom ∧ d3∈ThreadStatus ∧
                          i4∈ThreadIndexDom ∧ d4∈ThreadStatus
                          ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}∈ThreadIndexDom → ThreadStatus

theorem
@threadStatusOvr8 ∀a,i1,i2,i3,i4,i5,i6,i7,i8,d1,d2,d3,d4,d5,d6,d7,d8·a∈(ThreadIndexDom→ThreadStatus) ∧
                  i1∈ThreadIndexDom ∧ d1∈ThreadStatus ∧
                  i2∈ThreadIndexDom ∧ d2∈ThreadStatus ∧
                  i3∈ThreadIndexDom ∧ d3∈ThreadStatus ∧
                  i4∈ThreadIndexDom ∧ d4∈ThreadStatus ∧
                  i5∈ThreadIndexDom ∧ d5∈ThreadStatus ∧
                  i6∈ThreadIndexDom ∧ d6∈ThreadStatus ∧
```

```
                    i7∈ThreadIndexDom ∧ d7∈ThreadStatus ∧
                    i8∈ThreadIndexDom ∧ d8∈ThreadStatus
                  ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□{i7 ↦ d7}□{i8 ↦ d8}∈ThreadIndexDom → ThreadStatus

  theorem @threadPcOvr1 ∀a,i1,d1·a∈(ThreadIndexDom→Bits32Nat) ∧
                          i1∈ThreadIndexDom ∧ d1∈Bits32Nat
                          ⇒ a□{i1 ↦ d1}∈ThreadIndexDom → Bits32Nat

  theorem @threadPcOvr2 ∀a,i1,i2,d1,d2·a∈(ThreadIndexDom→Bits32Nat) ∧
                          i1∈ThreadIndexDom ∧ d1∈Bits32Nat ∧ i2∈ThreadIndexDom ∧ d2∈Bits32Nat
                          ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}∈ThreadIndexDom → Bits32Nat

  theorem @threadPcOvr4 ∀a,i1,i2,i3,i4,d1,d2,d3,d4·a∈(ThreadIndexDom→Bits32Nat) ∧
                          i1∈ThreadIndexDom ∧ d1∈Bits32Nat ∧
                          i2∈ThreadIndexDom ∧ d2∈Bits32Nat ∧
                          i3∈ThreadIndexDom ∧ d3∈Bits32Nat ∧
                          i4∈ThreadIndexDom ∧ d4∈Bits32Nat
                          ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}∈ThreadIndexDom → Bits32Nat

  theorem @threadPcOvr8 ∀a,i1,i2,i3,i4,i5,i6,i7,i8,d1,d2,d3,d4,d5,d6,d7,d8·a∈(ThreadIndexDom→Bits32Nat) ∧
                          i1∈ThreadIndexDom ∧ d1∈Bits32Nat ∧
                          i2∈ThreadIndexDom ∧ d2∈Bits32Nat ∧
                          i3∈ThreadIndexDom ∧ d3∈Bits32Nat ∧
                          i4∈ThreadIndexDom ∧ d4∈Bits32Nat ∧
                          i5∈ThreadIndexDom ∧ d5∈Bits32Nat ∧
                          i6∈ThreadIndexDom ∧ d6∈Bits32Nat ∧
                          i7∈ThreadIndexDom ∧ d7∈Bits32Nat ∧
                          i8∈ThreadIndexDom ∧ d8∈Bits32Nat
                        ⇒ a□{i1 ↦ d1}□{i2 ↦ d2}□{i3 ↦ d3}□{i4 ↦ d4}□{i5 ↦ d5}□{i6 ↦ d6}□{i7 ↦ d7}□{i8 ↦ d8}∈ThreadIndexDom → Bits32Nat

end
```

14.2.3 context Bit

```
context Bit // Define fundemental bit - distinct from BOOL

constants Set Clr

sets Bit

axioms

    @bitDef Bit = {Set,Clr}
    @setClrExcl Set ≠ Clr
    theorem @bitNotEmpty Bit ≠ ∅

end
```

14.2.4 context Bits

```
context Bits // Define bit-structures

extends Bit

constants MaxBitsSize MaxBitsIndex Bits

axioms

    @maxBitsSize MaxBitsSize = 64 // Up to 64 bits is supported
    @maxBitsIndex MaxBitsIndex = MaxBitsSize-1
    @bitsDef Bits = (0‥MaxBitsIndex) → Bit

    theorem @maxBitsIndexVal MaxBitsIndex = 63
    theorem @bits32Exists Bits ≠  ∅

end
```

14.2.5 context BitsToNat

```
context BitsToNat // Define translations from bit-structures to numbers

extends Bits

constants BitsToNat NatToBits BitsToInt IntToBits

          MaxBits32Nat MinBits32Int MaxBits32Int
          MaxBits64Nat MinBits64Int MaxBits64Int

          Bits1Nat
          Bits2Nat
          Bits4Nat
          Bits5Nat
          Bits6Nat
          Bits8Nat
          Bits10Nat
          Bits16Nat
          Bits20Nat
          Bits32Nat Bits32Int
          Bits64Nat Bits64Int

          Bits1ToNat NatToBits1
          Bits2ToNat NatToBits2
          Bits4ToNat
          Bits5ToNat
          Bits6ToNat
          Bits8ToNat  NatToBits8
          Bits10ToNat
          Bits16ToNat
          Bits20ToNat
          Bits32ToNat NatToBits32
          Bits32ToInt IntToBits32
          Bits64ToNat NatToBits64
          Bits64ToInt IntToBits64

axioms

    @bitsToNat BitsToNat ∈ (Bits × ℕ) → ℕ      // n = BitsToNat(b,size), zero extends from nth bit
    @natToBits NatToBits ∈  (ℕ × ℕ) → Bits     // b = BitsToNat(n,size)
    @bitsToInt BitsToInt ∈ (Bits × ℕ) → ℤ     // i = BitsToNat(b,size), signed extends from nth bit
    @intToBits IntToBits ∈  (ℤ × ℕ) → Bits    // b = BitsToNat(i,size)
```

```
@maxBits32Nat MaxBits32Nat = ((2147483647*2)+1) // Rodin cannot support 4294967295
@minBits32Int MinBits32Int = -(2147483647)
@maxBits32Int MaxBits32Int =  2147483647
@maxBits64Nat MaxBits64Nat = ((2147483647*2) * (2147483647*2))
@minBits64Int MinBits64Int = -(2147483647 * 2147483647)
@maxBits64Int MaxBits64Int =  (2147483647 * 2147483647)
@nat64GtNat32 MaxBits32Nat < MaxBits64Nat // Should be a theorem but prover cannot handle 64-bit

@bits1Nat      Bits1Nat = 0‥1
@bits1ToNat    Bits1ToNat ∈ Bits → Bits1Nat
@bits1ToNatDef  ∀b·b∈Bits ⇒  Bits1ToNat(b)=BitsToNat(b↦1)
@natToBits1    NatToBits1 ∈ Bits1Nat → Bits
@natToBits1Def  ∀n·n∈Bits1Nat ⇒  NatToBits1(n)=NatToBits(n↦1)

@bits2Nat Bits2Nat = 0‥3
@bits2ToNat Bits2ToNat ∈ Bits → Bits2Nat
@bits2ToNatDef ∀b·b∈Bits ⇒  Bits2ToNat(b)=BitsToNat(b↦2)
@natToBits2 NatToBits2 ∈ Bits2Nat → Bits
@natToBits2Def ∀n·n∈Bits2Nat ⇒  NatToBits2(n)=NatToBits(n↦2)

@bits4Nat Bits4Nat = 0‥15
@bits4ToNat Bits4ToNat ∈ Bits → Bits4Nat
@bits4ToNatDef ∀b·b∈Bits ⇒  Bits4ToNat(b)=BitsToNat(b↦4)

@bits5Nat Bits5Nat = 0‥31
@bits5ToNat Bits5ToNat ∈ Bits → Bits5Nat
@bits5ToNatDef ∀b·b∈Bits ⇒  Bits5ToNat(b)=BitsToNat(b↦5)

@bits6Nat Bits6Nat = 0‥63
@bits6ToNat Bits6ToNat ∈ Bits → Bits6Nat
@bits6ToNatDef ∀b·b∈Bits ⇒  Bits6ToNat(b)=BitsToNat(b↦6)

@bits8Nat Bits8Nat = 0‥255
@bits8ToNat Bits8ToNat ∈ Bits → Bits8Nat
@bits8ToNatDef ∀b·b∈Bits ⇒  Bits8ToNat(b)=BitsToNat(b↦8)
@natToBits8 NatToBits8 ∈ Bits8Nat → Bits
@natToBits8Def ∀n·n∈Bits8Nat ⇒  NatToBits8(n)=NatToBits(n↦8)

@bits10Nat Bits10Nat = 0‥1023
@bits10ToNat Bits10ToNat ∈ Bits → Bits10Nat
@bits10ToNatDef ∀b·b∈Bits ⇒  Bits10ToNat(b)=BitsToNat(b↦10)
```

```
@bits16Nat Bits16Nat = 0‥65535
@bits16ToNat Bits16ToNat ∈ Bits → Bits16Nat
@bits16ToNatDef ∀b·b∈Bits ⇒  Bits16ToNat(b)=BitsToNat(b↦16)

@bits20Nat Bits20Nat = 0‥1048575
@bits20ToNat Bits20ToNat ∈ Bits → Bits20Nat
@bits20ToNatDef ∀b·b∈Bits ⇒  Bits20ToNat(b)=BitsToNat(b↦20)

@bits32Nat Bits32Nat = 0‥MaxBits32Nat
@bits32ToNat Bits32ToNat ∈ Bits → Bits32Nat
theorem @bits32ToNatRaw Bits32ToNat∈ℙ(ℤ × Bit) ⇸ ℤ
@bits32ToNatDef ∀b·b∈Bits ⇒  Bits32ToNat(b)=BitsToNat(b↦32)
@natToBits32 NatToBits32 ∈ Bits32Nat → Bits
@natToBits32Def ∀n·n∈Bits32Nat ⇒  NatToBits32(n)=NatToBits(n↦32)

@bits32Int Bits32Int = MinBits32Int‥MaxBits32Int
@bits32ToInt Bits32ToInt ∈ Bits → Bits32Int
theorem @bits32ToIntRaw Bits32ToInt∈ℙ(ℤ × Bit) ⇸ ℤ
@bits32ToIntDef ∀b·b∈Bits ⇒  Bits32ToInt(b)=BitsToInt(b↦32)
@intToBits32 IntToBits32 ∈ Bits32Int → Bits
@intToBits32Def ∀i·i∈Bits32Int ⇒  IntToBits32(i)=IntToBits(i↦32)

@bits64Nat Bits64Nat = 0‥MaxBits64Nat
@bits64ToNat Bits64ToNat ∈ Bits → Bits64Nat
@bits64ToNatDef ∀b·b∈Bits ⇒  Bits64ToNat(b)=BitsToNat(b↦64)
@natToBits64 NatToBits64 ∈ Bits64Nat → Bits
@natToBits64Def ∀n·n∈Bits64Nat ⇒  NatToBits64(n)=NatToBits(n↦64)

@bits64Int Bits64Int = MinBits64Int‥MaxBits64Int
@bits64ToInt Bits64ToInt ∈ Bits → Bits64Int
@bits64ToIntDef ∀b·b∈Bits ⇒  Bits64ToInt(b)=BitsToInt(b↦64)
@intToBits64 IntToBits64 ∈ Bits64Int → Bits
@intToBits64Def ∀i·i∈Bits64Int ⇒  IntToBits64(i)=IntToBits(i↦64)

@bits32NatHas1million ∀x·x∈0‥2000000000 ⇒ x∈Bits32Nat

theorem @bits1NatInBits32Nat  Bits1Nat ⊆  Bits32Nat
theorem @bits2NatInBits32Nat  Bits2Nat ⊆  Bits32Nat
theorem @bits4NatInBits32Nat  Bits4Nat ⊆  Bits32Nat
theorem @bits5NatInBits32Nat  Bits5Nat ⊆  Bits32Nat
theorem @bits6NatInBits32Nat  Bits6Nat ⊆  Bits32Nat
```

```
theorem @bits16NatInBits32Nat Bits16Nat ⊆  Bits32Nat
theorem @bits20NatInBits32Nat Bits20Nat ⊆  Bits32Nat

theorem @bits32NatInBits64Nat Bits32Nat ⊆  Bits64Nat

theorem @bits1NatHas0   0∈Bits1Nat
theorem @bits1NatHas1   1∈Bits1Nat

theorem @bits4NatHas00  0∈Bits4Nat
theorem @bits4NatHas01  1∈Bits4Nat
theorem @bits4NatHas02  2∈Bits4Nat
theorem @bits4NatHas03  3∈Bits4Nat
theorem @bits4NatHas04  4∈Bits4Nat
theorem @bits4NatHas05  5∈Bits4Nat
theorem @bits4NatHas06  6∈Bits4Nat
theorem @bits4NatHas07  7∈Bits4Nat
theorem @bits4NatHas08  8∈Bits4Nat
theorem @bits4NatHas09  9∈Bits4Nat
theorem @bits4NatHas10 10∈Bits4Nat
theorem @bits4NatHas11 11∈Bits4Nat
theorem @bits4NatHas12 12∈Bits4Nat
theorem @bits4NatHas13 13∈Bits4Nat
theorem @bits4NatHas14 14∈Bits4Nat
theorem @bits4NatHas15 15∈Bits4Nat

theorem @bits5NatHas00  0∈Bits5Nat
theorem @bits5NatHas01  1∈Bits5Nat
theorem @bits5NatHas02  2∈Bits5Nat
theorem @bits5NatHas03  3∈Bits5Nat
theorem @bits5NatHas04  4∈Bits5Nat
theorem @bits5NatHas05  5∈Bits5Nat
theorem @bits5NatHas06  6∈Bits5Nat
theorem @bits5NatHas07  7∈Bits5Nat
theorem @bits5NatHas08  8∈Bits5Nat
theorem @bits5NatHas09  9∈Bits5Nat
theorem @bits5NatHas10 10∈Bits5Nat
theorem @bits5NatHas11 11∈Bits5Nat
theorem @bits5NatHas12 12∈Bits5Nat
theorem @bits5NatHas13 13∈Bits5Nat
theorem @bits5NatHas14 14∈Bits5Nat
theorem @bits5NatHas15 15∈Bits5Nat
```

```
    theorem @bits5NatHas16 16∈Bits5Nat
    theorem @bits5NatHas17 17∈Bits5Nat
    theorem @bits5NatHas18 18∈Bits5Nat
    theorem @bits5NatHas19 19∈Bits5Nat
    theorem @bits5NatHas20 20∈Bits5Nat
    theorem @bits5NatHas21 21∈Bits5Nat
    theorem @bits5NatHas22 22∈Bits5Nat
    theorem @bits5NatHas23 23∈Bits5Nat
    theorem @bits5NatHas24 24∈Bits5Nat
    theorem @bits5NatHas25 25∈Bits5Nat
    theorem @bits5NatHas26 26∈Bits5Nat
    theorem @bits5NatHas27 27∈Bits5Nat
    theorem @bits5NatHas28 28∈Bits5Nat
    theorem @bits5NatHas29 29∈Bits5Nat
    theorem @bits5NatHas30 30∈Bits5Nat
    theorem @bits5NatHas31 31∈Bits5Nat

    theorem @bits32NatHas0   0∈Bits32Nat
    theorem @bits32NatHas1   1∈Bits32Nat
    theorem @bits32NatHas2   2∈Bits32Nat
    theorem @bits32NatHas4   4∈Bits32Nat
    theorem @bits32NatHas8   8∈Bits32Nat
    theorem @bits32NatHas15  15∈Bits32Nat
    theorem @bits32NatHas16  16∈Bits32Nat
    theorem @bits32NatHas32  32∈Bits32Nat
    theorem @bits32NatHas64  64∈Bits32Nat
    theorem @bits32NatHas128 128∈Bits32Nat

    theorem @zeroNatToBits32IsBits NatToBits32(0)∈Bits

end
```

14.2.6 context BitsConcatenation

```
context BitsConcatenation

extends BitsToNat

constants ConcatenateBits
          TwoBits1ToBits2
          TwoBits2ToBits4
          TwoBits4ToBits8
          TwoBits32ToBits64
          TwoBits8ToBits16
          TwoBits16ToBits32
          FourBits8ToBits32
          EightBits1ToBits8
          EightBits1ToBits32

          OneBits32ToBits64Sext OneBits32ToBits64Zext
          OneBits8ToBits32 TwoBits8ToBits32

axioms

  @concatenateBits     ConcatenateBits ∈ (Bits× ℕ × Bits × ℕ) → Bits  // op = ConcatenateBits(ip1,size1,ip2,size2)

  @twoBits1ToBits2TypeDef   TwoBits1ToBits2   ∈ (Bits × Bits) → Bits
  @twoBits1ToBits2Def               ∀b0,b1 ·b0∈Bits ∧ b1∈Bits ⇒ TwoBits1ToBits2(b0↦b1) = ConcatenateBits(b0↦1↦b1↦1)

  @twoBits2ToBits4TypeDef   TwoBits2ToBits4   ∈ (Bits × Bits) → Bits
  @twoBits2ToBits4Def               ∀b0,b1 ·b0∈Bits ∧ b1∈Bits ⇒ TwoBits2ToBits4(b0↦b1) = ConcatenateBits(b0↦2↦b1↦2)

  @twoBits4ToBits8TypeDef   TwoBits4ToBits8   ∈ (Bits × Bits) → Bits
  @twoBits4ToBits8Def               ∀b0,b1 ·b0∈Bits ∧ b1∈Bits ⇒ TwoBits4ToBits8(b0↦b1) = ConcatenateBits(b0↦4↦b1↦4)

  @twoOneBits32ToBits64TypeDef   TwoBits32ToBits64   ∈ (Bits × Bits) → Bits
  @twoOneBits32ToBits64Def                ∀b0,b1 ·b0∈Bits ∧ b1∈Bits ⇒ TwoBits32ToBits64(b0↦b1) = ConcatenateBits(b0↦32↦b1↦32)

  @twoBits8ToBits16TypeDef TwoBits8ToBits16 ∈ (Bits × Bits) → Bits
  @twoBits8ToBits16Def           ∀b0,b1 ·b0∈Bits ∧ b1∈Bits ⇒ TwoBits8ToBits16(b0↦b1) = ConcatenateBits(b0↦8↦b1↦8)

  @twoBits16ToBits32TypeDef TwoBits16ToBits32 ∈ (Bits × Bits) → Bits
  @twoBits16ToBits32Def           ∀b0,b1 ·b0∈Bits ∧ b1∈Bits ⇒ TwoBits16ToBits32(b0↦b1) = ConcatenateBits(b0↦16↦b1↦16)

  @fourBits8ToBits32TypeDef FourBits8ToBits32 ∈ (Bits × Bits × Bits × Bits) → Bits
```

```
  @fourBits8ToBits32Def            ∀b0,b1,b2,b3·b0∈Bits ∧ b1∈Bits ∧b2∈Bits ∧ b3∈Bits ⇒
                                  FourBits8ToBits32(b0↦b1↦b2↦b3) = TwoBits16ToBits32(TwoBits8ToBits16(b0↦b1)↦TwoBits8ToBits16(b2↦b3))

  @eightBits1ToBits8       EightBits1ToBits8 ∈ (Bits × Bits × Bits × Bits × Bits × Bits × Bits × Bits) → Bits
  @eightBits1ToBits8Def    ∀b0,b1,b2,b3,b4,b5,b6,b7·b0∈Bits ∧ b1∈Bits ∧ b2∈Bits ∧ b3∈Bits ∧ b4∈Bits ∧ b5∈Bits ∧ b6∈Bits ∧ b7∈Bits ⇒
                              EightBits1ToBits8(b0↦b1↦b2↦b3↦b4↦b5↦b6↦b7)=
                               TwoBits4ToBits8(TwoBits2ToBits4(TwoBits1ToBits2(b0↦b1)↦TwoBits1ToBits2(b2↦b3))↦
                                TwoBits2ToBits4(TwoBits1ToBits2(b4↦b5)↦ TwoBits1ToBits2(b6↦b7)))

  @eightBits1ToBits32      EightBits1ToBits32 ∈ (Bits × Bits × Bits × Bits × Bits × Bits × Bits × Bits) → Bits
  @eightBits1ToBits32Def   ∀b0,b1,b2,b3,b4,b5,b6,b7·b0∈Bits ∧ b1∈Bits ∧ b2∈Bits ∧ b3∈Bits ∧ b4∈Bits ∧ b5∈Bits ∧ b6∈Bits ∧ b7∈Bits ⇒
                           EightBits1ToBits32(b0↦b1↦b2↦b3↦b4↦b5↦b6↦b7)=
                           FourBits8ToBits32(EightBits1ToBits8(b0↦b1↦b2↦b3↦b4↦b5↦b6↦b7)↦NatToBits8(0)↦NatToBits8(0)↦NatToBits8(0))

  @bits32ToBits64Sext        OneBits32ToBits64Sext ∈ Bits → Bits // Uninterpreted

  @bits32ToBits64ZextDef     OneBits32ToBits64Zext ∈ Bits → Bits
  @bits32ToBits64Zext        ∀b32·b32∈Bits ⇒ OneBits32ToBits64Zext(b32) = TwoBits32ToBits64(b32↦NatToBits32(0))

  @oneBits8ToBits32Type      OneBits8ToBits32 ∈ Bits → Bits
  @oneBits8ToBits32Def       ∀b0·b0∈Bits ⇒ OneBits8ToBits32(b0)=FourBits8ToBits32(b0↦NatToBits8(0)↦NatToBits8(0)↦NatToBits8(0))

  @twoBits8ToBits32Type      TwoBits8ToBits32  ∈ (Bits × Bits) → Bits
  @twoBits8ToBits32Def       ∀b0,b1·b0∈Bits ∧ b1∈Bits ⇒ TwoBits8ToBits32(b0↦b1)=FourBits8ToBits32(b0↦b1↦NatToBits8(0)↦NatToBits8(0))

end
```

14.2.7 context BitsExtraction

```
context BitsExtraction // Define extraction of bit-structures from Bit32

extends BitsToNat

constants ExtractBits
          ExtractBits1  ExtractBits1Nat
          ExtractBits2  ExtractBits2Nat
          ExtractBits4  ExtractBits4Nat
          ExtractBits5  ExtractBits5Nat
          ExtractBits6  ExtractBits6Nat
          ExtractBits8_0  ExtractBits8_1  ExtractBits8_2 ExtractBits8_3
          ExtractBits1_0  ExtractBits1_1  ExtractBits1_2 ExtractBits1_3 ExtractBits1_4  ExtractBits1_5  ExtractBits1_6 ExtractBits1_7
          ExtractBits32_0 ExtractBits32_1
          ExtractBits10 ExtractBits10Nat
          ExtractBits16 ExtractBits16Nat
          ExtractBits20 ExtractBits20Nat

axioms

    @extractType ExtractBits ∈ (Bits × ℕ × ℕ) → Bits     // op = ExtractBits(ip,pos,size)
    @extractDef ∀ip,pos,size,op,ipSize ·
                 ipSize∈0‥MaxBitsSize  ∧
                 ip∈Bits ∧
                 size∈0‥MaxBitsSize  ∧
                 pos∈(0‥(ipSize−size)) ∧
                 op∈Bits ∧
                 op=ExtractBits(ip↦pos↦size)
                                  ⇒ (∀i·i∈(0‥(size−1)) ⇒  op(i)=ip(i+pos))

    @extract1Type ExtractBits1 ∈ (Bits × (0‥31)) → Bits
    @extract1Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥31) ⇒  ExtractBits1(b32↦pos)=ExtractBits(b32↦pos↦1)

    @extract2Type ExtractBits2 ∈ (Bits × (0‥30)) → Bits
    @extract2Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥30) ⇒ ExtractBits2(b32↦pos)=ExtractBits(b32↦pos↦2)

    @extract4Type ExtractBits4 ∈ (Bits × (0‥28)) → Bits
    @extract4Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥28) ⇒ ExtractBits4(b32↦pos)=ExtractBits(b32↦pos↦4)

    @extract5Type ExtractBits5 ∈ (Bits × (0‥27)) → Bits
    @extract5Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥27) ⇒ ExtractBits5(b32↦pos)=ExtractBits(b32↦pos↦5)
```

```
@extract6Type ExtractBits6 ∈ (Bits × (0‥26)) → Bits
@extract6Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥26) ⇒ ExtractBits6(b32↦pos)=ExtractBits(b32↦pos↦6)

@extractBits8_0   ExtractBits8_0 ∈ Bits → Bits
@extractBits8_0Def ∀b32·b32∈Bits ⇒ ExtractBits8_0(b32)=ExtractBits(b32↦0↦8)

@extractBits8_1   ExtractBits8_1 ∈ Bits → Bits
@extractBits8_1Def ∀b32·b32∈Bits ⇒ ExtractBits8_1(b32)=ExtractBits(b32↦8↦8)

@extractBits8_2   ExtractBits8_2 ∈ Bits → Bits
@extractBits8_2Def ∀b32·b32∈Bits ⇒ ExtractBits8_2(b32)=ExtractBits(b32↦16↦8)

@extractBits8_3   ExtractBits8_3 ∈ Bits → Bits
@extractBits8_3Def ∀b32·b32∈Bits ⇒ ExtractBits8_3(b32)=ExtractBits(b32↦24↦8)

@extractBits1_0   ExtractBits1_0 ∈ Bits → Bits
@extractBits1_0Def ∀b32·b32∈Bits ⇒ ExtractBits1_0(b32)=ExtractBits(b32↦0↦1)

@extractBits1_1   ExtractBits1_1 ∈ Bits → Bits
@extractBits1_1Def ∀b32·b32∈Bits ⇒ ExtractBits1_1(b32)=ExtractBits(b32↦1↦1)

@extractBits1_2   ExtractBits1_2 ∈ Bits → Bits
@extractBits1_2Def ∀b32·b32∈Bits ⇒ ExtractBits1_2(b32)=ExtractBits(b32↦2↦1)

@extractBits1_3   ExtractBits1_3 ∈ Bits → Bits
@extractBits1_3Def ∀b32·b32∈Bits ⇒ ExtractBits1_3(b32)=ExtractBits(b32↦3↦1)

@extractBits1_4   ExtractBits1_4 ∈ Bits → Bits
@extractBits1_4Def ∀b32·b32∈Bits ⇒ ExtractBits1_4(b32)=ExtractBits(b32↦4↦1)

@extractBits1_5   ExtractBits1_5 ∈ Bits → Bits
@extractBits1_5Def ∀b32·b32∈Bits ⇒ ExtractBits1_5(b32)=ExtractBits(b32↦5↦1)

@extractBits1_6   ExtractBits1_6 ∈ Bits → Bits
@extractBits1_6Def ∀b32·b32∈Bits ⇒ ExtractBits1_6(b32)=ExtractBits(b32↦6↦1)

@extractBits1_7   ExtractBits1_7 ∈ Bits → Bits
@extractBits1_7Def ∀b32·b32∈Bits ⇒ ExtractBits1_7(b32)=ExtractBits(b32↦7↦1)

@extractBits32_0   ExtractBits32_0 ∈ Bits → Bits
@extractBits32_0Def ∀b64·b64∈Bits ⇒ ExtractBits32_0(b64)=ExtractBits(b64↦0↦32)
```

```
    @extractBits32_1   ExtractBits32_1 ∈ Bits → Bits
    @extractBits32_1Def ∀b64·b64∈Bits ⇒ ExtractBits32_1(b64)=ExtractBits(b64↦32↦32)

    @extract10Type ExtractBits10 ∈ (Bits × (0‥22)) → Bits
    @extract10Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥22) ⇒ ExtractBits10(b32↦pos)=ExtractBits(b32↦pos↦10)

    @extract16Type ExtractBits16 ∈ (Bits × (0‥16)) → Bits
    @extract16Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥16) ⇒  ExtractBits16(b32↦pos)=ExtractBits(b32↦pos↦16)

    @extract20Type ExtractBits20 ∈ (Bits × (0‥12)) → Bits
    @extract20Def ∀b32,pos·b32∈Bits ∧ pos∈(0‥12) ⇒  ExtractBits20(b32↦pos)=ExtractBits(b32↦pos↦32)


    @extract1NatType ExtractBits1Nat ∈ (Bits × (0‥31)) → Bits1Nat
    @extract1NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥31) ⇒ ExtractBits1Nat(b32↦pos) = Bits1ToNat(ExtractBits1(b32 ↦ pos))

    @extract2NatType ExtractBits2Nat ∈ (Bits × (0‥30)) → Bits2Nat
    @extract2NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥30) ⇒ ExtractBits2Nat(b32↦pos) = Bits2ToNat(ExtractBits2(b32 ↦ pos))

    @extract4NatType ExtractBits4Nat ∈ (Bits × (0‥28)) → Bits4Nat
    @extract4NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥28) ⇒ ExtractBits4Nat(b32↦pos) = Bits4ToNat(ExtractBits4(b32 ↦ pos))

    @extract5NatType ExtractBits5Nat ∈ (Bits × (0‥27)) → Bits5Nat
    @extract5NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥27) ⇒ ExtractBits5Nat(b32↦pos) = Bits5ToNat(ExtractBits5(b32 ↦ pos))

    @extract6NatType ExtractBits6Nat ∈ (Bits × (0‥26)) → Bits6Nat
    @extract6NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥26) ⇒ ExtractBits6Nat(b32↦pos) = Bits6ToNat(ExtractBits6(b32 ↦ pos))

    @extract10NatType ExtractBits10Nat ∈ (Bits × (0‥22)) → Bits10Nat
    @extract10NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥22) ⇒ ExtractBits10Nat(b32↦pos) = Bits10ToNat(ExtractBits10(b32 ↦ pos))

    @extract16NatType ExtractBits16Nat ∈ (Bits × (0‥16)) → Bits16Nat
    @extract16NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥16) ⇒ ExtractBits16Nat(b32↦pos) = Bits16ToNat(ExtractBits16(b32 ↦ pos))

    @extract20NatType ExtractBits20Nat ∈ (Bits × (0‥12)) → Bits20Nat
    @extract20NatDef ∀b32,pos · b32∈Bits ∧ pos∈(0‥12) ⇒ ExtractBits20Nat(b32↦pos) = Bits20ToNat(ExtractBits20(b32 ↦ pos))

end
```

14.2.8 context BitsManipulation

```
context BitsManipulation

extends BitsConcatenation BitsExtraction
        ArithmeticTheorems

constants CountLeadingZeroes NandBits CrcBits

          BitSet BitClr BitsSet BitsClr
          CreateMask ZeroExtendBits OneExtendBits
          ShiftRightBits ShiftLeftBits
          OorBits AndBits XorBits AndNotBits NegBits

axioms

  @countLeadingZs CountLeadingZeroes ∈Bits → Bits32Nat // op = CountLeadingZeroes(ip)
  @nandBits       NandBits  ∈ (Bits × Bits) → Bits // op = NandBits(ip1,ip2)
  @crcBits        CrcBits ∈ (Bits × Bits × Bits × Bits32Nat) → Bits  // op = CrcBits(ip,data,polynomial,size)

  @negBits      NegBits ∈ Bits → Bits
  @negBitsDef       ∀a·a∈Bits ⇒ NegBits(a) = NandBits(a↦a)

  @andBits      AndBits ∈ (Bits × Bits) → Bits
  @andBitsDef       ∀a,b·a∈Bits ∧ b∈Bits ⇒ AndBits(a↦b) = NegBits(NandBits(a↦b))

  @oorBits      OorBits ∈ (Bits × Bits) → Bits
  @oorBitsDef       ∀a,b·a∈Bits ∧ b∈Bits ⇒ OorBits(a↦b) = NandBits(NegBits(a)↦NegBits(b))

  @xorBits      XorBits ∈ (Bits × Bits) → Bits
  @xorBitsDef       ∀a,b·a∈Bits ∧ b∈Bits ⇒ XorBits(a↦b) = NandBits(NandBits(a↦NandBits(a↦b))↦NandBits(NandBits(a↦b)↦b))

  @andNotBits   AndNotBits ∈ (Bits × Bits) → Bits // op = AndNotBits(ip1,ip2)
  @andNotBitsDef ∀a,b·a∈Bits ∧ b∈Bits ⇒ AndNotBits(a↦b) = AndBits(a↦NegBits(b))

  @bitClr       BitClr ∈ Bits
  @bitClrDef    BitClr = NatToBits1(0)
  @bitSet       BitSet ∈ Bits
  @bitSetDef    BitSet = NatToBits1(1)

  @bitsClr      BitsClr ∈ Bits
  @bitsClrDef   BitsClr = NatToBits32(0)
  @bitsSet      BitsSet ∈ Bits
```

```
  @bitsSetDef    BitsSet = NegBits(BitsClr)

  @zeroExtBits    ZeroExtendBits ∈ (Bits × Bits32Nat) → Bits // op = ZeroExtBits(ip,size)
  @zeroExtBitsDef ∀a,b·a∈Bits ∧ b∈Bits32Nat ∧ b<32 ⇒ ZeroExtendBits(a↦b) = ConcatenateBits(ExtractBits(a↦0↦b)↦b↦BitsClr↦(32-b))

  @oneExtBits     OneExtendBits ∈ (Bits × Bits32Nat) → Bits // op = OneExtendBits(ip,size)
  @oneExtBitsDef ∀a,b·a∈Bits ∧ b∈Bits32Nat ∧ b<32 ⇒ OneExtendBits(a↦b)    = ConcatenateBits(ExtractBits(a↦0↦b)↦b↦BitsSet↦(32-b))

  @createMask    CreateMask ∈ Bits32Nat → Bits // op = CreateMask(size)
  @createMaskDef ∀a·a∈Bits32Nat ⇒ CreateMask(a) = ZeroExtendBits(BitsSet↦a)

  @shiftRightBits    ShiftRightBits ∈ (Bits × Bits32Nat) → Bits // op = ShiftRightBits(ip,size)
  @shiftRightBitsDef ∀a,b·a∈Bits ∧ b∈Bits32Nat ∧ b<32 ⇒ ShiftRightBits(a↦b) = ZeroExtendBits(ExtractBits(a↦b↦(32-b))↦(32-b))

  @shiftLeftBits    ShiftLeftBits ∈ (Bits × Bits32Nat) → Bits // op = ShiftLeftBits(ip,size)
  @shiftLeftBitsDef ∀a,b·a∈Bits ∧ b∈Bits32Nat ∧ b<32 ⇒
                    ShiftLeftBits(a↦b) = ConcatenateBits(ExtractBits(BitsClr↦0↦b)↦b↦ExtractBits(a↦0↦(32-b))↦(32-b))

end
```

14.2.9 context Arithmetic

```
context Arithmetic

extends BitsExtraction

constants WrapBits32Unsigned WrapBits64Unsigned
          WrapBits32Signed WrapBits64Signed
          AddUnsigned ModuloAddUnsigned ModuloLaddUnsigned
          AddSigned ModuloAddSigned ModuloLaddSigned
          SubUnsigned ModuloSubUnsigned ModuloLsubUnsigned
          SubSigned ModuloSubSigned
          MulUnsigned ModuloMulUnsigned
          MulSigned ModuloMulSigned
          DivUnsigned LdivUnsigned
          DivSigned
          RemUnsigned
          RemSigned
          RoundUnsigned
          CompareEqualUnsigned
          CompareNotEqualUnsigned
          CompareLessUnsigned CompareLessSigned

axioms

    @wrapBits32Unsigned     WrapBits32Unsigned ∈ ℤ → Bits32Nat
    @wrapBits32DefUnsigned     ∀a·a∈ℕ ⇒  WrapBits32Unsigned(a) = Bits32ToNat(ExtractBits(NatToBits(a↦64)↦0↦32))

    @wrapBits64Unsigned     WrapBits64Unsigned ∈ ℤ → Bits64Nat
    @wrapBits64DefUnsigned     ∀a·a∈ℕ ⇒  WrapBits64Unsigned(a) = Bits64ToNat(ExtractBits(NatToBits(a↦64)↦0↦64))

    @wrapBits32Signed       WrapBits32Signed ∈ ℤ → Bits32Int
    @wrapBits32DefSigned         ∀a·a∈ℤ ⇒  WrapBits32Signed(a) = Bits32ToInt(ExtractBits(IntToBits(a↦64)↦0↦32))

    @wrapBits64Signed       WrapBits64Signed ∈ ℤ → Bits64Int
    @wrapBits64DefSigned         ∀a·a∈ℤ ⇒  WrapBits64Signed(a) = Bits64ToInt(ExtractBits(IntToBits(a↦64)↦0↦64))

    @addUnsigned          AddUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits64Nat // 32-bit add with 64bit result
    @addUnsignedDef                ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒ AddUnsigned(a↦b) = (a+b)

    @moduloAddUnsigned    ModuloAddUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits32Nat // 32-bit add with wrap
    @moduloAddUnsignedDef ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒ ModuloAddUnsigned(a↦b) = WrapBits32Unsigned(a+b)
```

```
@moduloLaddUnsigned    ModuloLaddUnsigned ∈ (Bits64Nat × Bits64Nat) → Bits64Nat // 64-bit add with wrap
@moduloLaddUnsignedDef ∀a,b·a∈Bits64Nat ∧ b∈Bits64Nat ⇒ ModuloLaddUnsigned(a↦b) = WrapBits64Unsigned(a+b)

@addSigned             AddSigned ∈ (Bits32Int × Bits32Int) → Bits64Int // 32-bit add with 64bit result
@addSignedDef                      ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒ AddSigned(a↦b) = (a+b)

@moduloAddSigned       ModuloAddSigned ∈ (Bits32Int × Bits32Int) → Bits32Int // 32-bit add with wrap
@moduloAddSignedDef       ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒ ModuloAddSigned(a↦b) = WrapBits32Signed(a+b)

@moduloLaddSigned      ModuloLaddSigned ∈ (Bits64Int × Bits64Int) → Bits64Int // 64-bit add with wrap
@moduloLaddSignedDef     ∀a,b·a∈Bits64Int ∧ b∈Bits64Int ⇒ ModuloLaddSigned(a↦b) = WrapBits64Signed(a+b)

@subUnsigned           SubUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits64Nat // 32-bit subtract with 64bit result
@subUnsignedDef                ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒ SubUnsigned(a↦b) = (a-b)

@moduloSubUnsigned     ModuloSubUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits32Nat // 32-bit subtract with wrap
@moduloSubUnsignedDef ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒ ModuloSubUnsigned(a↦b) = WrapBits32Unsigned(a-b)

@moduloLsubUnsigned    ModuloLsubUnsigned ∈ (Bits64Nat × Bits64Nat) → Bits64Nat // 64-bit add with wrap
@moduloLsubUnsignedDef ∀a,b·a∈Bits64Nat ∧ b∈Bits64Nat ⇒ ModuloLsubUnsigned(a↦b) = WrapBits64Unsigned(a-b)

@subSigned             SubSigned ∈ (Bits32Int × Bits32Int) → Bits64Int // 32-bit subtract with 64bit result
@subSignedDef                  ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒ SubSigned(a↦b) = (a-b)

@moduloSubSigned       ModuloSubSigned ∈ (Bits32Int × Bits32Int) → Bits32Int // 32-bit subtract with wrap
@moduloSubSignedDef       ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒ ModuloSubSigned(a↦b) = WrapBits32Signed(a-b)

@mulUnsigned           MulUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits64Nat // 32-bit multiply with 64bit result
@mulUnsignedDef                ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒ MulUnsigned(a↦b) = (a*b)

@moduloMulUnsigned     ModuloMulUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits32Nat // 32-bit multiply with wrap
@moduloMulUnsignedDef ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒ ModuloMulUnsigned(a↦b) = WrapBits32Unsigned(a*b)

@mulSigned             MulSigned ∈ (Bits32Int × Bits32Int) → Bits64Int // 32-bit multiply with 64bit result
@mulSignedDef                  ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒ MulSigned(a↦b) = (a*b)

@moduloMulSigned       ModuloMulSigned ∈ (Bits32Int × Bits32Int) → Bits32Int // 32-bit multiply with wrap
@moduloMulSignedDef       ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒ ModuloMulSigned(a↦b) = WrapBits32Signed(a*b)

@divUnsigned           DivUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits32Nat // 32-bit divide with wrap
@divUnsignedDef                ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ∧ b≠0⇒  DivUnsigned(a↦b) = (a÷b)
```

```
    @ldivUnsigned          LdivUnsigned ∈ (Bits64Nat × Bits32Nat) → Bits64Nat
    @ldivUnsignedDef              ∀a,b·a∈Bits64Nat ∧ b∈Bits32Nat ∧ b≠0 ⇒  LdivUnsigned(a↦b) = (a÷b)

    @divSigned             DivSigned      ∈ (Bits32Int × Bits32Int) → Bits32Int // 32-bit divide with wrap
    @divSignedDef                   ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ∧ b≠0 ⇒  DivSigned(a↦b) = (a÷b)

    @remUnsigned        RemUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits32Nat // 32-bit remainder
    @remUnsignedDef          ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ∧ b≠0 ⇒  RemUnsigned(a↦b) = (a mod b)

    @remSigned          RemSigned ∈ (Bits32Int × Bits32Int) → Bits32Int // 32-bit remainder
    @remSignedDef       ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ∧ b≠0 ⇒  RemSigned(a↦b) = ModuloSubSigned(a↦ModuloMulSigned(b↦DivSigned(a↦b)))

    @roundUnsigned      RoundUnsigned ∈ (Bits32Nat × Bits32Nat) → Bits32Nat // 32-bit round-down
    @roundUnsignedDef     ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ∧ b≠0 ⇒ RoundUnsigned(a↦b)=MulUnsigned(DivUnsigned(a↦b)↦b)

    @compareEqualUnsigned       CompareEqualUnsigned      ∈ (Bits32Nat × Bits32Nat) → BOOL
    @compareEqualUnsignedDef         ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒  CompareEqualUnsigned(a↦b)=bool(a=b)

    @compareNotEqualUnsigned    CompareNotEqualUnsigned      ∈ (Bits32Nat × Bits32Nat) → BOOL
    @compareNotEqualUnsignedDef ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒  CompareNotEqualUnsigned(a↦b)=bool(a≠b)

    @compareLessUnsigned        CompareLessUnsigned      ∈ (Bits32Nat × Bits32Nat) → BOOL
    @compareLessUnsignedDef           ∀a,b·a∈Bits32Nat ∧ b∈Bits32Nat ⇒  CompareLessUnsigned(a↦b)=bool(a<b)

    @compareLessSigned          CompareLessSigned      ∈ (Bits32Int × Bits32Int) → BOOL
    @compareLessSignedDef             ∀a,b·a∈Bits32Int ∧ b∈Bits32Int ⇒  CompareLessSigned(a↦b)=bool(a<b)

end
```

14.2.10context Fifo

```
context Fifo // Defines fifo as circular buffer

constants FifoSize FifoIndexDom FifoSizeDom
          FifoIncrement
          FifoWriteIndex FifoReadIndex

axioms

  @fifoSize         FifoSize ∈ ℕ1 // Must be at least one
  @fifoIndexDomDef FifoIndexDom = 0‥(FifoSize−1)
  @fifoSizeDomDef   FifoSizeDom =  0‥FifoSize

  @fifoIncrement     FifoIncrement ∈ FifoIndexDom × FifoSizeDom → FifoIndexDom
  @fifoIncrementDef ∀index,incr·index∈FifoIndexDom ∧ incr∈FifoSizeDom ⇒ (FifoIncrement(index↦incr)=((index+incr) mod FifoSize))

  @fifoWriteIndex     FifoWriteIndex ∈ FifoIndexDom × FifoSizeDom → FifoIndexDom
  @fifoWriteIndexDef ∀readIndex,depth·readIndex∈FifoIndexDom ∧ depth∈FifoSizeDom ⇒
(FifoWriteIndex(readIndex↦depth)=FifoIncrement(readIndex↦depth))

  @fifoReadIndex     FifoReadIndex ∈ (FifoIndexDom × (0‥FifoSize)) → FifoIndexDom
  @fifoReadIndexDef ∀readIndex,depth·readIndex∈FifoIndexDom ∧ depth∈FifoSizeDom ⇒ (FifoReadIndex(readIndex↦depth)=readIndex)

end
```

14.2.11 context ArraySearch

```
context ArraySearch

extends BitsToNat

constants FindInArray CountInArray

axioms

  @findInArray   ∀size·size∈1‥1000 ⇒ FindInArray ∈ ((0‥(size−1)→Bits32Nat) × 1‥size × Bits32Nat) → (0‥(size−1))
  @countInArray ∀size·size∈1‥1000 ⇒ CountInArray ∈ ((0‥(size−1)→Bits32Nat) × 1‥size × Bits32Nat) → (0‥size)

end
```

14.2.12context Status

```
context Status // Define enumerations of machine status.

extends BitsToNat

constants FREE STARTING LOADING RUNNING DEBUGGING WAITING MPAUSING SPAUSING CLOSING

axioms

  @freeDef      FREE      = 1
  @startingDef  STARTING  = 2
  @loadingDef   LOADING   = 3
  @runningDef   RUNNING   = 4
  @debuggingDef DEBUGGING = 5
  @waitingDef   WAITING   = 6    // Waiting for an event
  @mpausingDef  MPAUSING  = 7    // Master pausing for slaves
  @spausingDef  SPAUSING  = 8    // Slaves pausing for master
  @closingDef   CLOSING   = 9

  theorem @freeInBit32Nat      FREE ∈Bits32Nat
  theorem @staringInBit32Nat   STARTING∈Bits32Nat
  theorem @loadingInBit32Nat   LOADING∈Bits32Nat
  theorem @runningInBit32Nat   RUNNING∈Bits32Nat
  theorem @debuggingInBit32Nat DEBUGGING∈Bits32Nat
  theorem @waitingInBit32Nat   WAITING∈Bits32Nat
  theorem @mpausingInBit32Nat  MPAUSING∈Bits32Nat
  theorem @spausingInBit32Nat  SPAUSING∈Bits32Nat
  theorem @closingInBit32Nat   CLOSING∈Bits32Nat

end
```

14.2.13context XCoreStatus

```
context XCoreStatus

extends Status

constants CoreStatus

axioms

  @coreStatusDef CoreStatus = {LOADING,RUNNING,DEBUGGING}
  theorem @coreStatusType     CoreStatus ⊆ℕ
  theorem @coreStatusBit32Nat CoreStatus ⊆Bits32Nat

end
```

14.2.14context ThreadState

```
context ThreadState

extends Status

constants ThreadStatus

axioms

  @threadStatusDef ThreadStatus = {FREE,RUNNING,WAITING,MPAUSING,SPAUSING}
  theorem @threadStatusNat      ThreadStatus ⊆ℕ
  theorem @threadStatusBit32Nat ThreadStatus ⊆Bits32Nat

end
```

14.2.15context ThreadDom

```
context ThreadDom

constants MaxThreads MaxThreadIndex ThreadIndexDom

axioms

  @maxThreads MaxThreads ∈ ℕ1 // Must be at least one thread
  @maxThreadIndex MaxThreadIndex = MaxThreads-1
  @threadIndexDom ThreadIndexDom = 0‥(MaxThreads-1)
  theorem @threadIndexDomType ThreadIndexDom ⊆ℕ

end
```

14.2.16context ThreadArray

```
context ThreadArray

extends ThreadDom ThreadState

end
```

14.2.17context RegArrayDom

```
context RegArrayDom // Define domain for register array at most abstract level

constants RegArraySize RegArrayDom PcIndex

axioms

  @regArraySizeType          RegArraySize ∈ ℕ1                  // Contains at least one element
  @regArrayDomIsZeroArray    RegArrayDom = 0‥(RegArraySize−1) // Zero-indexed array
  @pcIndexInRegArrayDom      PcIndex ∈ RegArrayDom
  theorem @regArrayDomExists RegArrayDom ≠ ∅
  theorem @regArrayDomIsNat  RegArrayDom⊆ℕ

end
```

14.2.18context RegArrayDomE1

```
context RegArrayDomE1 // Define register array as containing multiple pages, one for each thread

extends RegArrayDom ThreadDom

end
```

14.2.19context RegArray

```
context RegArray // Define register array as zero-indexed array of Bits

extends RegArrayDom Bits

constants RegArray

axioms

  @regArrayType             RegArray = RegArrayDom → Bits
  theorem @regDomBitsExists RegArrayDom → Bits ≠ ∅
  theorem @regArrayExists   RegArray ≠ ∅

end
```

14.2.20context RegArrayE1

```
context RegArrayE1 // Define register array as containing multiple pages, one per thread

extends RegArray RegArrayDomE1

end
```

14.2.21context ProcessorState

```
context ProcessorState

extends Bits

constants ProcStateSize ProcStateDom MaxProcStateIndex

axioms

  @size ProcStateSize ∈ ℕ1 // Array is at least 1 in size
  @max  MaxProcStateIndex = (ProcStateSize-1)
  @dom  ProcStateDom = 0‥MaxProcStateIndex
  theorem @procStateDomExists ProcStateDom ≠ ∅

  theorem @procStateExists ProcStateDom → Bits ≠ ∅

  theorem @procStateOvr1 ∀a,i1,d1·a∈(ProcStateDom→Bits) ∧ i1∈ProcStateDom ∧ d1∈Bits ⇒ a□{i1 ↦ d1}∈ProcStateDom → Bits

end
```

14.2.22context PcDom

```
context PcDom // Define basic instruction array domain
constants PcDom MaxVector MinVector

axioms
  @axm2 MaxVector ∈ ℕ // MaxVector is zero or greater
  @axm3 MaxVector ≥ MinVector // Address space is one element or greater
  @axm1 MinVector ∈ ℕ // MinVector is zero or greater
  @axm4 PcDom = MinVector‥MaxVector
  theorem @thm1 PcDom ≠  ∅ // Confirm that domain exists
end
```

14.2.23context PcArray

```
context PcArray
extends ThreadDom BitsToNat

end
```

14.2.24context MemArrayDom

```
context MemArrayDom // Define domain for memory data array

constants MinMemIndex MemArraySize MaxMemIndex MaxWordIndex
          MemArrayDom

axioms
  @axm1 MinMemIndex ∈ ℕ
  @axm2 MemArraySize ≥ 4 // Data array is at least 1 word in size
  @axm3 MaxMemIndex = MinMemIndex + (MemArraySize − 1)
  theorem @minLtMax MinMemIndex ≤ MaxMemIndex

  @axm4 MaxWordIndex = (MinMemIndex + MemArraySize)−4

  @axm5 MemArrayDom = MinMemIndex‥MaxMemIndex
  theorem @memArrayDomIsNat MemArrayDom ⊆ ℕ

end
```

14.2.25context MemArrayDomE1

```
context MemArrayDomE1 // Define instruction array within memory array
extends MemArrayDom PcDom
axioms
  @pcDomInMemDom PcDom ⊆ MemArrayDom
End
```

14.2.26context MemArrayDomE2

```
context MemArrayDomE2
// Define domain of extended, 8-bit memory in terms of MemArrayDom
extends MemArrayDomE1

constants ExtMemArrayDom
axioms
  @pcDomInMemDom ExtMemArrayDom = MinMemIndex‥MaxMemIndex+3 // 3 incessible bytes in order to simplify gluing invariant
  theorem @pc0InExtMemDom ∀x·x∈MemArrayDom ⇒ x     ∈ExtMemArrayDom
  theorem @pc1InExtMemDom ∀x·x∈MemArrayDom ⇒ x+1∈ExtMemArrayDom
  theorem @pc2InExtMemDom ∀x·x∈MemArrayDom ⇒ x+2∈ExtMemArrayDom
  theorem @pc3InExtMemDom ∀x·x∈MemArrayDom ⇒ x+3∈ExtMemArrayDom
  theorem @memArrayDomInExt MemArrayDom ⊆ ExtMemArrayDom
end
```

14.2.27context MemArrayDomE3

```
context MemArrayDomE3 // Fix PcDom for XCore

extends MemArrayDomE2

constants XCorePcDom XCoreBootVector

axioms

  @minVectorDef MinVector = MinMemIndex
  @maxVectorDef MaxVector = MaxMemIndex-1
  @pcDomDef XCorePcDom = MinMemIndex..MaxMemIndex-1
  theorem @xcorePcDomInPcDom XCorePcDom = PcDom
  @xcorePcDomExists XCorePcDom ≠ ∅  // Confirm that domain exists
  @xCoreBootVectorType XCoreBootVector ∈ XCorePcDom

end
```

14.2.28context MemArrayDomE4

```
context MemArrayDomE4
// Define memory address as within Bits32
extends MemArrayDomE3 BitsToNat

axioms

  @memArrayDomInBits32 MemArrayDom ⊆ Bits32Nat
  theorem @xCorePcDomInBits32 XCorePcDom ⊆ Bits32Nat

end
```

14.2.29context MemArray

```
context MemArray
// Define basic memory 32-bit data array
extends MemArrayDom Bits

axioms
  theorem @memArrayExists MemArrayDom → Bits ≠ ∅
end
```

14.2.30context MemArrayE1

```
context MemArrayE1
// Define InstArray as within MemArray
extends MemArray InstArray MemArrayDomE1
end
```

14.2.31context MemArrayE2

```
context MemArrayE2
/* Define practical memory 8-bit data array
   Define mapping between 32-bit and 8-bit arrays */
extends MemArrayE1 MemArrayDomE2 BitsConcatenation
constants ExtMemArray
axioms

  @extMemArrayDef ExtMemArray = ExtMemArrayDom → Bits

  theorem @extMemArrayExists ExtMemArrayDom → Bits ≠ ∅

  @arraysMapping ∀i,x,y1,y2,a32,a8 · y2≥(y1+3) ∧ i∈x··y1 ∧ a32∈x··y1→Bits ∧ a8∈x··y2→Bits
                 ⇒a32(i)=FourBits8ToBits32(a8(i)↦a8(i+1)↦a8(i+2)↦a8(i+3))

  @arraysMappinggInstantiate ∀i,a32,a8 · i∈MemArrayDom ∧ a32∈MemArrayDom→Bits ∧ a8∈ExtMemArrayDom → Bits
                             ⇒a32(i)=FourBits8ToBits32(a8(i)↦a8(i+1)↦a8(i+2)↦a8(i+3))
End
```

14.2.32context MemArrayE3

```
context MemArrayE3 // Add MemArrayDomE3
extends MemArrayE2 MemArrayDomE3
end
```

14.2.33context MemAlignment

```
context MemAlignment
// Define basic memory alignment set
constants Alignments

axioms
  @axm1 Alignments ⊆  ℕ1 // Alignments must be 1 or more
end
```

14.2.34context MemAlignmentE1

```
context MemAlignmentE1
// Define memory alignments for practical memory
extends MemAlignment

constants MemAlignments

axioms
  @axm2 MemAlignments = {1,2,4}
  theorem @oneIsMemAlignment  1∈MemAlignments
  theorem @twoIsMemAlignment  2∈MemAlignments
  theorem @fourIsMemAlignment 4∈MemAlignments
end
```

14.2.35context Macros

```
context Macros // Macro to hide compound expressions
extends MemArrayE3
        XCoreStatus XCoreThreads PcArray

constants PcOk
          ThreadRunning ThreadDebugging ThreadActive
          RunningOkPcCheck RunningPcCheck
          MpausingCheck SpausingCheck
          GetInstr GetMemBits32 GetMemBits16
          WordAddrRangeOk ShortAddrRangeOk ByteAddrRangeOk
          WordAddrAlignedOk ShortAddrAlignedOk
          WordAddrOkCheck ShortAddrOkCheck ByteAddrOkCheck

axioms

  @pcOk          PcOk ∈ (ThreadIndexDom → Bits32Nat) × ThreadIndexDom → BOOL
  @pcOkDef               ∀p,t · p∈(ThreadIndexDom → Bits32Nat) ∧ t∈ThreadIndexDom  ⇒
                 PcOk(p↦t)= bool((p(t)∈XCorePcDom) ∧ (p(t) mod 2=0))

  @threadRun     ThreadRunning ∈ CoreStatus × (ThreadIndexDom → ThreadStatus) × ThreadIndexDom → BOOL
  @threadRunDef    ∀s,ts,t · s∈CoreStatus ∧ ts∈(ThreadIndexDom → ThreadStatus) ∧  t∈ThreadIndexDom  ⇒
                   ThreadRunning(s↦ts↦t)= bool(s=RUNNING ∧ ts(t)=RUNNING)

  @threadDeb     ThreadDebugging ∈ CoreStatus × (ThreadIndexDom → ThreadStatus) × ThreadIndexDom → BOOL
  @threadDebDef    ∀s,ts,t · s∈CoreStatus ∧ ts∈(ThreadIndexDom → ThreadStatus) ∧  t∈ThreadIndexDom  ⇒
                   ThreadDebugging(s↦ts↦t)= bool(t=0 ∧ s=DEBUGGING ∧ ts(t)=RUNNING)

  @threadAct     ThreadActive ∈ CoreStatus × (ThreadIndexDom → ThreadStatus) × ThreadIndexDom → BOOL
  @threadActDef    ∀s,ts,t · s∈CoreStatus ∧ ts∈(ThreadIndexDom → ThreadStatus) ∧  t∈ThreadIndexDom  ⇒
                   ThreadActive(s↦ts↦t)= bool(ThreadRunning(s↦ts↦t)=TRUE ∨ ThreadDebugging(s↦ts↦t)=TRUE)

  @pcCheckOkType RunningOkPcCheck∈(CoreStatus×(ThreadIndexDom→ThreadStatus)×(ThreadIndexDom→Bits32Nat)×ThreadIndexDom×BOOL)→BOOL
  @pcCheckOkDef  ∀s,ts,p,t,i·(s∈CoreStatus)∧(ts∈(ThreadIndexDom→ThreadStatus)∧p∈(ThreadIndexDom→Bits32Nat)∧t∈ThreadIndexDom∧i∈BOOL)⇒
                 RunningOkPcCheck(s↦ts↦p↦t↦i)= bool(ThreadActive(s↦ts↦t)=TRUE ∧ PcOk(p↦t)=TRUE ∧ (i=FALSE))

  @rpcCheckType RunningPcCheck ∈ (CoreStatus × (ThreadIndexDom → ThreadStatus) × (ThreadIndexDom → Bits32Nat) × ThreadIndexDom) → BOOL
  @rpcCheckDef  ∀s,ts,p,t·(s∈CoreStatus)∧(ts∈(ThreadIndexDom→ThreadStatus)∧p∈(ThreadIndexDom→Bits32Nat)∧t∈ThreadIndexDom)⇒
              RunningPcCheck(s↦ts↦p↦t) = bool(ThreadActive(s↦ts↦t)=TRUE ∧ PcOk(p↦t)=TRUE)

  @mpcCheckType MpausingCheck ∈ (CoreStatus × (ThreadIndexDom → ThreadStatus) × ThreadIndexDom) → BOOL
  @mpcCheckDef ∀s,ts,t · (s∈CoreStatus) ∧ (ts∈(ThreadIndexDom → ThreadStatus) ∧  t∈ThreadIndexDom)  ⇒
               MpausingCheck(s↦ts↦t) = bool((s=RUNNING) ∧ (ts(t)=MPAUSING))
```

```
  @spcCheckType SpausingCheck ∈ (CoreStatus × (ThreadIndexDom → ThreadStatus) × ThreadIndexDom) → BOOL
  @spcCheckDef ∀s,ts,t · (s∈CoreStatus) ∧ (ts∈(ThreadIndexDom → ThreadStatus) ∧  t∈ThreadIndexDom)  ⇒
               SpausingCheck(s↦ts↦t) = bool((s=RUNNING) ∧ (ts(t)=SPAUSING))

  @getInstrType  GetInstr ∈ ((ExtMemArrayDom→Bits) × PcDom) → Inst
  @getInstrDef   ∀m,pc · (m∈(ExtMemArrayDom→Bits))∧ pc∈PcDom  ⇒ GetInstr(m↦pc)=FourBits8ToBits32(m(pc+2)↦m(pc+3)↦m(pc)↦m(pc+1))
  @getInstrBasic GetInstr∈ℙ(ℤ × ℙ(ℤ × Bit)) × ℤ ⇸ ℙ(ℤ × Bit)

  @getMemBits16Type GetMemBits16 ∈ ((ExtMemArrayDom→Bits)×MemArrayDom) → Bits
  @getMemBits16Def ∀m,a · (m∈(ExtMemArrayDom→Bits)) ∧ (a∈MemArrayDom)⇒  (GetMemBits16(m↦a)=TwoBits8ToBits32(m(a+0)↦m(a+1)))

  @getMemBits32Type GetMemBits32 ∈ ((ExtMemArrayDom→Bits)×MemArrayDom) → Bits
  @getMemBits32Def   ∀m,a (m∈(ExtMemArrayDom→Bits)) ∧ (a∈MemArrayDom)⇒
                     GetMemBits32(m↦a)=FourBits8ToBits32(m(a+0)↦m(a+1)↦m(a+2)↦m(a+3)))

  @wordAddrRangeOk WordAddrRangeOk ∈ ℕ → BOOL
  @wordAddrRangeOkDef ∀a · a∈ℕ ⇒  WordAddrRangeOk(a)=
                       bool(a∈MemArrayDom ∧ a+1∈MemArrayDom ∧ a+2∈MemArrayDom ∧ a+3∈MemArrayDom)

  @wordAddrAlignedOk WordAddrAlignedOk ∈ ℕ → BOOL
  @wordAddrAlignedOkDef ∀a · a∈ℕ ⇒  WordAddrAlignedOk(a)=bool(a mod 4=0)

  @wordAddrCheckOkType WordAddrOkCheck ∈ ℕ → BOOL
  @wordAddrCheckOkDef ∀a · a∈ℕ ⇒  WordAddrOkCheck(a)=bool(WordAddrRangeOk(a)=TRUE ∧ WordAddrAlignedOk(a)=TRUE)

  @shortAddrRangeOk ShortAddrRangeOk ∈ ℕ → BOOL
  @shortAddrRangeOkDef ∀a · a∈ℕ ⇒  ShortAddrRangeOk(a)=bool(a∈MemArrayDom ∧ a+1∈MemArrayDom)

  @shortAddrAlignedOk ShortAddrAlignedOk ∈ ℕ → BOOL
  @shortAddrAlignedOkDef ∀a · a∈ℕ ⇒  ShortAddrAlignedOk(a)=bool(a mod 2=0)

  @shortAddrCheckOkType ShortAddrOkCheck ∈ ℕ → BOOL
  @shortAddrCheckOkDef ∀a · a∈ℕ ⇒  ShortAddrOkCheck(a)=bool(ShortAddrRangeOk(a)=TRUE ∧ ShortAddrAlignedOk(a)=TRUE)

  @byteAddrRangeOk   ByteAddrRangeOk ∈ ℕ → BOOL
  @byteAddrRangeOkDef ∀a · a∈ℕ ⇒  ByteAddrRangeOk(a)=bool(a∈MemArrayDom)

  @byteAddrCheckOkType ByteAddrOkCheck ∈ ℕ → BOOL
  @byteAddrCheckOkDef ∀a · a∈ℕ ⇒  ByteAddrOkCheck(a)=bool(ByteAddrRangeOk(a)=TRUE)

end
```

14.2.36context InstSize

```
context InstSize

constants InstrSize

axioms
  @instrSizeDef InstrSize ⊆ ℕ1  // Instruction size must be greater than zero
end
```

14.2.37context InstArray

```
context InstArray // Define register array type
extends PcDom XCoreFields

axioms
  theorem @InstArrayExists PcDom → Inst ≠ ∅
end
```

14.2.38context Loader

```
context Loader

extends XCoreMemArrayDom

constants FetchImageData

axioms
   @fetchImageDataType FetchImageData ∈ 0‥(MemArraySize-1) → Bits
end
```

14.2.39context XCoreBasic

```
context XCoreBasic // Container for all basic XCore contexts

extends OvrTheorems Macros BitsManipulation Arithmetic
        XCoreMemArray XCoreDataSize XCoreInstSize XCoreEncodings XCoreStatusCodes

end
```

14.2.40context XCoreChannelEnds

```
context XCoreChannelEnds

extends XCoreResources ArraySearch Fifo

constants ChannelEndStatus
          MaxChannelEnds MaxChannelEndIndex ChannelEndIndexDom MaxChannelEndDepth
          GetChannelEndId
          IsChannelEndIdent IsChannelEndInUse
          ChannelEndData ChannelEndBool ChannelEndState ChannelEndDepth
          FindFreeChannelEnd CountFreeChannelEnds

          ChannelEndReadStateChange ChannelEndWriteStateChange
          ChannelEndReadPosition  ChannelEndReadPosition0  ChannelEndReadPosition1  ChannelEndReadPosition2  ChannelEndReadPosition3
          ChannelEndWritePosition ChannelEndWritePosition0 ChannelEndWritePosition1 ChannelEndWritePosition2 ChannelEndWritePosition3
          ChannelEndSpace

          IsApplicationCtrlToken IsSpecialCtrlToken IsPrivilegedCtrlToken

axioms

  @channelEndStatusDef   ChannelEndStatus = {FREE,RUNNING}
  theorem @channelEndStatusType  ChannelEndStatus ⊆ℕ

  @maxChannelEnds MaxChannelEnds ∈ ℕ1 // Must be at least one
  @maxChannelEndIndex MaxChannelEndIndex = MaxChannelEnds-1
  @synchroniserIndexDom ChannelEndIndexDom = 0‥MaxChannelEndIndex

  @maxChannelEndsSeven MaxChannelEnds = 32
  @fifoSize            FifoSize = MaxChannelEndDepth
  @maxChannelEndDepth  MaxChannelEndDepth = 8

  @getChannelEndId      GetChannelEndId ∈ ChannelEndIndexDom → ResourceId
  @getChannelEndIdDef       ∀i·i∈ChannelEndIndexDom ⇒ GetChannelEndId(i)=GetResourceId(i↦ResourceChannelEnd)

  @isChannelEndIdentType IsChannelEndIdent ∈ Bits → BOOL
  @isChannelEndIdentDef  ∀i·i∈Bits  ⇒ IsChannelEndIdent(i)=
                           bool(GetResourceType(i)=ResourceChannelEnd ∧ GetResourceIndex(i)∈ChannelEndIndexDom)

  @isChannelEndInUseType IsChannelEndInUse ∈ (Bits × (ChannelEndIndexDom → ChannelEndStatus)) → BOOL
  @isChannelEndInUseDef  ∀i,s·i∈Bits ∧ s∈ChannelEndIndexDom → ChannelEndStatus  ⇒ IsChannelEndInUse(i↦s)=
                           bool(GetResourceIndex(i)∈ChannelEndIndexDom ∧ s(GetResourceIndex(i))≠FREE)
```

```
@findFreeType FindFreeChannelEnd ∈ (ChannelEndIndexDom→ChannelEndStatus) → ChannelEndIndexDom
@findFreeDef  ∀s·(s∈ChannelEndIndexDom→ChannelEndStatus) ⇒ FindFreeChannelEnd(s)=FindInArray(s↦MaxChannelEnds↦FREE)

@countFreeType CountFreeChannelEnds ∈ (ChannelEndIndexDom→ChannelEndStatus) → 0‥MaxChannelEnds
@countFreeDef  ∀s·(s∈ChannelEndIndexDom→ChannelEndStatus) ⇒ CountFreeChannelEnds(s)=CountInArray(s↦MaxChannelEnds↦FREE)

@channelEndData  ChannelEndData  = FifoIndexDom → Bits
@channelEndBool  ChannelEndBool  = FifoIndexDom → BOOL
@channelEndState ChannelEndState = FifoIndexDom
@channelEndDepth ChannelEndDepth = FifoSizeDom

@channelEndReadState    ChannelEndReadStateChange ∈ ChannelEndState × (0‥MaxChannelEndDepth) → ChannelEndState
@channelEndReadStateDef  ∀s,i·s∈ChannelEndState ∧ i∈FifoSizeDom ⇒
                         ChannelEndReadStateChange(s↦i)=FifoIncrement(s↦i) // Only read increments state

@channelEndWrite        ChannelEndWriteStateChange ∈ ChannelEndState × (0‥MaxChannelEndDepth) → ChannelEndState
@channelEndWriteDef           ∀s,i·s∈ChannelEndState ∧ i∈FifoSizeDom ⇒
                         ChannelEndWriteStateChange(s↦i)=FifoIncrement(s↦0) // Only read increments state

@channelEndReadPos     ChannelEndReadPosition ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
@channelEndReadPosDef    ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                       ChannelEndReadPosition(state↦depth)=FifoIncrement(FifoReadIndex(state↦depth)↦0)

@channelEndReadPos0    ChannelEndReadPosition0 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
@channelEndReadPos0Def ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                       ChannelEndReadPosition0(state↦depth)=FifoIncrement(FifoReadIndex(state↦depth)↦3) // MSB first

@channelEndReadPos1    ChannelEndReadPosition1 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
@channelEndReadPos1Def ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                       ChannelEndReadPosition1(state↦depth)=FifoIncrement(FifoReadIndex(state↦depth)↦2) // MSB first

@channelEndReadPos2    ChannelEndReadPosition2 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
@channelEndReadPos2Def ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                       ChannelEndReadPosition2(state↦depth)=FifoIncrement(FifoReadIndex(state↦depth)↦1) // MSB first

@channelEndReadPos3    ChannelEndReadPosition3 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
@channelEndReadPos3Def ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                       ChannelEndReadPosition3(state↦depth)=FifoIncrement(FifoReadIndex(state↦depth)↦0) // MSB first

@channelEndWritePos     ChannelEndWritePosition ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
@channelEndWritePosDef    ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                        ChannelEndWritePosition(state↦depth)=FifoIncrement(FifoWriteIndex(state↦depth)↦0)
```

```
  @channelEndWritePos0     ChannelEndWritePosition0 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
  @channelEndWritePos0Def  ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                           ChannelEndWritePosition0(state↦depth)=FifoIncrement(FifoWriteIndex(state↦depth)↦3) // MSB first

  @channelEndWritePos1     ChannelEndWritePosition1 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
  @channelEndWritePos1Def  ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                           ChannelEndWritePosition1(state↦depth)=FifoIncrement(FifoWriteIndex(state↦depth)↦2) // MSB first

  @channelEndWritePos2     ChannelEndWritePosition2 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
  @channelEndWritePos2Def  ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                           ChannelEndWritePosition2(state↦depth)=FifoIncrement(FifoWriteIndex(state↦depth)↦1) // MSB first

  @channelEndWritePos3     ChannelEndWritePosition3 ∈ ChannelEndState × ChannelEndDepth → FifoIndexDom
  @channelEndWritePos3Def  ∀state,depth·state∈ChannelEndState ∧ depth∈ChannelEndDepth ⇒
                           ChannelEndWritePosition3(state↦depth)=FifoIncrement(FifoWriteIndex(state↦depth)↦0) // MSB first

  @channelEndSpace         ChannelEndSpace ∈ChannelEndDepth → FifoIndexDom
  @channelEndSpaceDef             ∀depth·depth∈ChannelEndDepth ⇒  ChannelEndSpace(depth)=(FifoSize−depth)

  @isApplicationCtrlTokenType IsApplicationCtrlToken ∈ Bits → BOOL
  @isApplicationCtrlToken           ∀t·t∈Bits  ⇒ IsApplicationCtrlToken(t)=bool(Bits8ToNat(t)≥0 ∧ Bits8ToNat(t)≤127)

  @isSpecialCtrlTokenType     IsSpecialCtrlToken ∈ Bits → BOOL
  @isSpecialCtrlToken                 ∀t·t∈Bits  ⇒ IsSpecialCtrlToken(t)=bool(Bits8ToNat(t)≥128 ∧ Bits8ToNat(t)≤191)

  @isPrivilegedCtrlTokenType  IsPrivilegedCtrlToken ∈ Bits → BOOL
  @isPrivilegedCtrlToken            ∀t·t∈Bits  ⇒ IsPrivilegedCtrlToken(t)=bool(Bits8ToNat(t)≥192 ∧ Bits8ToNat(t)≤223)

end
```

14.2.41context XCoreClocks

```
context XCoreClocks

extends XCoreResources ArraySearch

constants ClockStatus
          MaxClocks MaxClockIndex ClockIndexDom
          GetClockId
          FindFreeClock CountFreeClocks
          IsClockIdent IsRefClockIdent

axioms

  @clockStatusDef ClockStatus = {FREE,RUNNING}
  theorem @clockStatusInNat       ClockStatus ⊆ℕ
  theorem @clockStatusInBits32Nat ClockStatus ⊆Bits32Nat

  @maxClocks MaxClocks = 6
  @maxClockIndex MaxClockIndex = MaxClocks−1
  @synchroniserIndexDom ClockIndexDom = 0‥(MaxClocks−1)

  @getClockId GetClockId ∈ ClockIndexDom → ResourceId
  @getClockIdDef ∀i·i∈ClockIndexDom ⇒ GetClockId(i) = GetResourceId(i↦ResourceClock)

  @findFreeStatusType FindFreeClock ∈ (ClockIndexDom→ClockStatus) → ClockIndexDom
  @findFreeStatusDef  ∀s·(s∈ClockIndexDom→ClockStatus) ⇒ FindFreeClock(s)=FindInArray(s↦MaxClocks↦FREE)

  @countFreeType CountFreeClocks ∈ (ClockIndexDom→ClockStatus) → 0‥MaxClocks
  @countFreeDef  ∀t·(t∈ClockIndexDom→ClockStatus) ⇒ CountFreeClocks(t)=CountInArray(t↦MaxClocks↦FREE)

  @isClockIdentType IsClockIdent ∈ Bits → BOOL
  @isClockIdentDef  ∀i·i∈Bits  ⇒ IsClockIdent(i)=bool(GetResourceType(i)=ResourceClock ∧ GetResourceIndex(i)∈ClockIndexDom)

  @isRefClockType IsRefClockIdent ∈ Bits → BOOL
  @isRefClockDef     ∀i·i∈Bits  ⇒ IsRefClockIdent(i)=bool(Bits32ToNat(i)=1)

end
```

14.2.42context XCoreDataSize

```
context XCoreDataSize

constants Bpw

axioms
  @bpwDef Bpw = 4
end
```

14.2.43context XCoreFields

```
context XCoreFields // Define XCore bottom-level format fields, with accessor functions

extends BitsExtraction

constants Inst
          XOPC XLOPC XUOPC XLUOPC XSUBOPC XLSUBOPC
          XBIT4_OPC XLBIT4_OPC
          X3R_DST X3R_SRC1 X3R_SRC2
          XBIT5 XRBIT5 XLBIT5
          X2R_REG Xr2R_REG X2R_SRC Xr2R_SRC XR_REG XLR_REG
          XRU6_REG XLRU6_REG
          XU6 XLU6 XU10 XLU10 XLIMM XU16 XU20
          XL4R_DST2
          XL5R_DST2 XL5R_SRC3
          XL6R_DST2 XL6R_SRC3 XL6R_SRC4

          XOPC_Nat XLOPC_Nat XUOPC_Nat XLUOPC_Nat XSUBOPC_Nat XLSUBOPC_Nat
          XBIT4_OPC_Nat XLBIT4_OPC_Nat
          X3R_DST_Nat X3R_SRC1_Nat X3R_SRC2_Nat
          XBIT5_Nat XRBIT5_Nat XLBIT5_Nat
          X2R_REG_Nat Xr2R_REG_Nat X2R_SRC_Nat Xr2R_SRC_Nat XR_REG_Nat XLR_REG_Nat
          XRU6_REG_Nat XLRU6_REG_Nat
          XU6_Nat XLU6_Nat XU10_Nat XLU10_Nat XLIMM_Nat XU16_Nat XU20_Nat
          XL4R_DST2_Nat
          XL5R_DST2_Nat XL5R_SRC3_Nat
          XL6R_DST2_Nat XL6R_SRC3_Nat XL6R_SRC4_Nat

axioms

    @instructionDef Inst = Bits

    @xOpcType XOPC ∈ Inst → Bits
    @xOpcDef    ∀i·i∈Inst ⇒ XOPC(i)=ExtractBits5(i↦27)

    @xLopcType XLOPC ∈ Inst → Bits
    @xLopcDef    ∀i·i∈Inst ⇒ XLOPC(i)=ExtractBits5(i↦11)

    @xUopcType XUOPC ∈ Inst → Bits
    @xUopcDef    ∀i·i∈Inst ⇒ XUOPC(i)=ExtractBits1(i↦26)

    @xLuopcType XLUOPC ∈ Inst → Bits
    @xLuopcDef    ∀i·i∈Inst ⇒ XLUOPC(i)=ExtractBits1(i↦10)
```

```
@xSubOpcType XSUBOPC ∈ Inst → Bits
@xSubOpcDef    ∀i·i∈Inst  ⇒ XSUBOPC(i)=ExtractBits(i↦22↦5)

@xLsubOpcType XLSUBOPC ∈ Inst → Bits
@xLsubOpcDef    ∀i·i∈Inst  ⇒ XLSUBOPC(i)=ExtractBits5(i↦6)

@xBit4OpcType XBIT4_OPC ∈ Inst → Bits
@xBit4OpcDef    ∀i·i∈Inst ⇒ XBIT4_OPC(i)=ExtractBits1(i↦20)

@xLbit4OpcType XLBIT4_OPC ∈ Inst → Bits
@xLbit4OpcDef    ∀i·i∈Inst ⇒ XLBIT4_OPC(i)=ExtractBits1(i↦4)

@x3rDstType X3R_DST ∈ Inst → Bits
@x3rDstDef    ∀i·i∈Inst ⇒ X3R_DST(i)=ExtractBits2(i↦20)

@x3rSrc1Type X3R_SRC1 ∈ Inst → Bits
@x3rSrc1Def    ∀i·i∈Inst ⇒ X3R_SRC1(i)=ExtractBits2(i↦18)

@x3rSrc2Type X3R_SRC2 ∈ Inst → Bits
@x3rSrc2Def    ∀i·i∈Inst ⇒ X3R_SRC2(i)=ExtractBits2(i↦16)

@xBit5Type XBIT5 ∈ Inst → Bits
@xBit5Def    ∀i·i∈Inst  ⇒ XBIT5(i)=ExtractBits1(i↦21)

@xRbit5Type XRBIT5 ∈ Inst → Bits
@xRbit5Def    ∀i·i∈Inst  ⇒ XRBIT5(i)=ExtractBits1(i↦21)

@xLbit5Type XLBIT5 ∈ Inst → Bits
@xLbit5Def    ∀i·i∈Inst  ⇒ XLBIT5(i)=ExtractBits1(i↦5)

@x2rRegType X2R_REG ∈ Inst → Bits
@x2rRegDef    ∀i·i∈Inst ⇒ X2R_REG(i)=ExtractBits2(i↦18)

@xr2rRegType Xr2R_REG ∈ Inst → Bits
@xr2rRegDef    ∀i·i∈Inst ⇒ Xr2R_REG(i)=ExtractBits2(i↦16)

@x2rSrcType X2R_SRC ∈ Inst → Bits
@x2rSrcDef    ∀i·i∈Inst ⇒ X2R_SRC(i)=ExtractBits2(i↦16)

@xr2rSrcType Xr2R_SRC ∈ Inst → Bits
@xr2rSrcDef    ∀i·i∈Inst ⇒ Xr2R_SRC(i)=ExtractBits2(i↦18)
```

```
@xRRegType XR_REG ∈ Inst → Bits
@xRRegDef    ∀i·i∈Inst ⇒ XR_REG(i)=ExtractBits4(i↦16)

@xLrRegType XLR_REG ∈ Inst → Bits
@xLrRegDef    ∀i·i∈Inst ⇒ XLR_REG(i)=ExtractBits4(i↦0)

@xRu6RegType XRU6_REG ∈ Inst → Bits
@xRu6RegDef    ∀i·i∈Inst ⇒ XRU6_REG(i)=ExtractBits4(i↦22)

@xLru6RegType XLRU6_REG ∈ Inst → Bits
@xLru6RegDef    ∀i·i∈Inst ⇒ XLRU6_REG(i)=ExtractBits4(i↦6)

@xU6Type XU6 ∈ Inst → Bits
@xU6Def    ∀i·i∈Inst ⇒ XU6(i)=ExtractBits6(i↦16)

@xLu6Type XLU6 ∈ Inst → Bits
@xLu6Def    ∀i·i∈Inst ⇒ XLU6(i)=ExtractBits6(i↦0)

@xU10Type XU10 ∈ Inst → Bits
@xU10Def    ∀i·i∈Inst ⇒ XU10(i)=ExtractBits10(i↦16)

@xLu10Type XLU10 ∈ Inst → Bits
@xLu10Def    ∀i·i∈Inst ⇒ XLU10(i)=ExtractBits10(i↦0)

@xLimmType XLIMM ∈ Inst → Bits
@xLimmDef    ∀i·i∈Inst ⇒ XLIMM(i)=ExtractBits10(i↦16)

@xU16Type XU16 ∈ Inst → Bits
@xU16Def    ∀i·i∈Inst ⇒ XU16(i)=ExtractBits16(i↦0)

@xU20Type XU20∈ Inst → Bits
@xU20Def    ∀i·i∈Inst ⇒ XU20(i)=ExtractBits20(i↦0)

@xL4R_Dst2Type XL4R_DST2 ∈ Inst → Bits
@xL4R_Dst2Def    ∀i·i∈Inst ⇒ XL4R_DST2(i)=ExtractBits4(i↦0)

@xL5R_Dst2Type XL5R_DST2 ∈ Inst → Bits
@xL5R_Dst2Def    ∀i·i∈Inst ⇒ XL5R_DST2(i)=ExtractBits2(i↦2)

@xL5R_Src3Type XL5R_SRC3 ∈ Inst → Bits
@xL5R_Src3Def    ∀i·i∈Inst ⇒ XL5R_SRC3(i)=ExtractBits2(i↦0)
```

```
@xL6R_Dst2Type XL6R_DST2 ∈ Inst → Bits
@xL6R_Dst2Def    ∀i·i∈Inst ⇒ XL6R_DST2(i)=ExtractBits2(i↦4)

@xL6R_Src3Type XL6R_SRC3 ∈ Inst → Bits
@xL6R_Src3Def    ∀i·i∈Inst ⇒ XL6R_SRC3(i)=ExtractBits2(i↦2)

@xL6R_Src4Type XL6R_SRC4 ∈ Inst → Bits
@xL6R_Src4Def    ∀i·i∈Inst ⇒ XL6R_SRC4(i)=ExtractBits2(i↦0)


@xOpcNatType XOPC_Nat ∈ Inst → Bits5Nat
@xOpcNatDef    ∀i·i∈Inst ⇒ XOPC_Nat(i)=Bits5ToNat(XOPC(i))

@xLopcNatType XLOPC_Nat ∈ Inst → Bits5Nat
@xLopcNatDef    ∀i·i∈Inst ⇒ XLOPC_Nat(i)=Bits5ToNat(XLOPC(i))

@xUopcNatType XUOPC_Nat ∈ Inst → Bits1Nat
@xUopcNatDef    ∀i·i∈Inst ⇒ XUOPC_Nat(i)=Bits1ToNat( XUOPC(i))

@xLuopcNatType XLUOPC_Nat ∈ Inst → Bits1Nat
@xLuopcNatDef    ∀i·i∈Inst ⇒ XLUOPC_Nat(i)=Bits1ToNat(XLUOPC(i))

@xSubOpcNatType XSUBOPC_Nat ∈ Inst → Bits5Nat
@xSubOpcNatDef    ∀i·i∈Inst  ⇒ XSUBOPC_Nat(i)=Bits5ToNat(XSUBOPC(i))

@xLsubOpcNatType XLSUBOPC_Nat ∈ Inst → Bits5Nat
@xLsubOpcNatDef    ∀i·i∈Inst  ⇒ XLSUBOPC_Nat(i)=Bits5ToNat(XLSUBOPC(i))

@xBit4OpcNatType XBIT4_OPC_Nat ∈ Inst → Bits1Nat
@xBit4OpcNatDef    ∀i·i∈Inst ⇒ XBIT4_OPC_Nat(i)=Bits1ToNat(XBIT4_OPC(i))

@xLbit4OpcNatType XLBIT4_OPC_Nat ∈ Inst → Bits1Nat
@xLbit4OpcNatDef    ∀i·i∈Inst ⇒ XLBIT4_OPC_Nat(i)=Bits1ToNat(XLBIT4_OPC(i))

@x3rDstNatType X3R_DST_Nat ∈ Inst → Bits2Nat
@x3rDstNatDef    ∀i·i∈Inst ⇒ X3R_DST_Nat(i)=Bits2ToNat(X3R_DST(i))

@x3rSrc1NatType X3R_SRC1_Nat ∈ Inst → Bits2Nat
@x3rSrc1NatDef    ∀i·i∈Inst ⇒ X3R_SRC1_Nat(i)=Bits2ToNat(X3R_SRC1(i))

@x3rSrc2NatType X3R_SRC2_Nat ∈ Inst → Bits2Nat
```

```
@x3rSrc2NatDef    ∀i·i∈Inst ⇒ X3R_SRC2_Nat(i)=Bits2ToNat(X3R_SRC2(i))

@xBit5NatType XBIT5_Nat ∈ Inst → Bits1Nat
@xBit5NatDef    ∀i·i∈Inst  ⇒ XBIT5_Nat(i)=Bits1ToNat(XBIT5(i))

@xRbit5NatType XRBIT5_Nat ∈ Inst → Bits1Nat
@xRbit5NatDef    ∀i·i∈Inst  ⇒ XRBIT5_Nat(i)=Bits1ToNat( XRBIT5(i))

@xLbit5NatType XLBIT5_Nat ∈ Inst → Bits1Nat
@xLbit5NatDef    ∀i·i∈Inst  ⇒ XLBIT5_Nat(i)=Bits1ToNat(XLBIT5(i))

@x2rRegNatType X2R_REG_Nat ∈ Inst → Bits2Nat
@x2rRegNatDef    ∀i·i∈Inst ⇒ X2R_REG_Nat(i)=Bits2ToNat(X2R_REG(i))

@xr2rRegNatType Xr2R_REG_Nat ∈ Inst → Bits2Nat
@xr2rRegNatDef    ∀i·i∈Inst ⇒ Xr2R_REG_Nat(i)=Bits2ToNat( Xr2R_REG(i))

@x2rSrcNatType X2R_SRC_Nat ∈ Inst → Bits2Nat
@x2rSrcNatDef    ∀i·i∈Inst ⇒ X2R_SRC_Nat(i)=Bits2ToNat(X2R_SRC(i))

@xr2rSrcNatType Xr2R_SRC_Nat ∈ Inst → Bits2Nat
@xr2rSrcNatDef    ∀i·i∈Inst ⇒ Xr2R_SRC_Nat(i)=Bits2ToNat(Xr2R_SRC(i))

@xRRegNatType XR_REG_Nat ∈ Inst → Bits4Nat
@xRRegNatDef    ∀i·i∈Inst ⇒ XR_REG_Nat(i)=Bits4ToNat(XR_REG(i))

@xLrRegNatType XLR_REG_Nat ∈ Inst → Bits4Nat
@xLrRegNatDef    ∀i·i∈Inst ⇒ XLR_REG_Nat(i)=Bits4ToNat(XLR_REG(i))

@xRu6RegNatType XRU6_REG_Nat ∈ Inst → Bits4Nat
@xRu6RegNatDef    ∀i·i∈Inst ⇒ XRU6_REG_Nat(i)=Bits4ToNat(XRU6_REG(i))

@xLru6RegNatType XLRU6_REG_Nat ∈ Inst → Bits4Nat
@xLru6RegNatDef    ∀i·i∈Inst ⇒ XLRU6_REG_Nat(i)=Bits4ToNat(XLRU6_REG(i))

@xU6NatType XU6_Nat ∈ Inst → Bits6Nat
@xU6NatDef    ∀i·i∈Inst ⇒ XU6_Nat(i)=Bits6ToNat(XU6(i))

@xLu6NatType XLU6_Nat ∈ Inst → Bits6Nat
@xLu6NatDef    ∀i·i∈Inst ⇒ XLU6_Nat(i)=Bits6ToNat( XLU6(i))

@xU10NatType XU10_Nat ∈ Inst → Bits10Nat
```

```
    @xU10NatDef     ∀i·i∈Inst ⇒ XU10_Nat(i)=Bits10ToNat(XU10(i))

    @xLu10NatType XLU10_Nat ∈ Inst → Bits10Nat
    @xLu10NatDef     ∀i·i∈Inst ⇒ XLU10_Nat(i)=Bits10ToNat(XLU10(i))

    @xLimmNatType XLIMM_Nat ∈ Inst → Bits10Nat
    @xLimmNatDef     ∀i·i∈Inst ⇒ XLIMM_Nat(i)=Bits10ToNat(XLIMM(i))

    @xU16NatType XU16_Nat ∈ Inst → Bits16Nat
    @xU16NatDef     ∀i·i∈Inst ⇒ XU16_Nat(i)=Bits16ToNat(XU16(i))

    @xU20NatType XU20_Nat ∈ Inst → Bits20Nat
    @xU20NatDef     ∀i·i∈Inst ⇒ XU20_Nat(i)=Bits20ToNat(XU20(i))

    @xL4R_Dst2NatType XL4R_DST2_Nat ∈ Inst → Bits4Nat
    @xL4R_Dst2NatDef     ∀i·i∈Inst ⇒ XL4R_DST2_Nat(i)=Bits2ToNat(XL4R_DST2(i))

    @xL5R_Dst2NatType XL5R_DST2_Nat ∈ Inst → Bits2Nat
    @xL5R_Dst2NatDef     ∀i·i∈Inst ⇒ XL5R_DST2_Nat(i)=Bits2ToNat(XL5R_DST2(i))

    @xL5R_Src3NatType XL5R_SRC3_Nat ∈ Inst → Bits2Nat
    @xL5R_Src3NatDef     ∀i·i∈Inst ⇒ XL5R_SRC3_Nat(i)=Bits2ToNat(XL5R_SRC3(i))

    @xL6R_Dst2NatType XL6R_DST2_Nat ∈ Inst → Bits2Nat
    @xL6R_Dst2NatDef     ∀i·i∈Inst ⇒ XL6R_DST2_Nat(i)=Bits2ToNat(XL6R_DST2(i))

    @xL6R_Src3NatType XL6R_SRC3_Nat ∈ Inst → Bits2Nat
    @xL6R_Src3NatDef     ∀i·i∈Inst ⇒ XL6R_SRC3_Nat(i)=Bits2ToNat(XL6R_SRC3(i))

    @xL6R_Src4NatType XL6R_SRC4_Nat ∈ Inst → Bits2Nat
    @xL6R_Src4NatDef     ∀i·i∈Inst ⇒ XL6R_SRC4_Nat(i)=Bits2ToNat(XL6R_SRC4(i))

end
```

14.2.44context XCoreFormats

```
context XCoreFormats // Define XCore instruction formats in terms of fields, accessed via their accessor functions
 extends XCoreOperands

constants Is3r IsL3r Is2rus IsL2rus IsRu6 IsLru6 IsU6 IsLu6 IsU10 IsLu10 Is2r IsR2r IsL2r IsLr2r IsRus Is1r Is0r IsL4r IsL5r IsL6r

axioms
  @is3r Is3r ∈ (Inst × Bits5Nat) → BOOL
  @is3rDef ∀inst,opc·inst∈Inst ∧ opc∈Bits5Nat
           ⇒ Is3r(inst↦opc) =
           bool(XOPC_Nat(inst)=opc ∧ XSUBOPC_Nat(inst)∈0‥26 ∧
           X3r_op1(inst)∈0‥11 ∧ X3r_op2(inst)∈0‥11 ∧ X3r_op3(inst)∈0‥11)

  @isL3r IsL3r ∈ (Inst × Bits5Nat × Bits4Nat) → BOOL
  @isL3rDef ∀inst,lopc,lr_reg·inst∈Inst ∧ lopc∈Bits5Nat ∧ lr_reg∈12‥15
           ⇒ IsL3r(inst↦lopc↦lr_reg) =
            bool(XOPC_Nat(inst)=31 ∧ XSUBOPC_Nat(inst)∈0‥26 ∧ XLOPC_Nat(inst)=lopc ∧ XLSUBOPC_Nat(inst)=31 ∧
            XLBIT5_Nat(inst)=1 ∧ XLBIT4_OPC_Nat(inst)=0 ∧ XLR_REG_Nat(inst)=lr_reg ∧
            XL3r_op1(inst)∈0‥11 ∧ XL3r_op2(inst)∈0‥11 ∧ XL3r_op3(inst)∈0‥11)

  @is2rus Is2rus ∈ (Inst × Bits5Nat) → BOOL
  @is2rusDef ∀inst,opc·inst∈Inst ∧ opc∈Bits5Nat ⇒ Is2rus(inst↦opc) =
             bool(XOPC_Nat(inst)=opc ∧ XSUBOPC_Nat(inst)∈0‥26 ∧
             X2rus_op1(inst)∈0‥11 ∧ X2rus_op2(inst)∈0‥11 ∧ X2rus_op3(inst)∈0‥11)

  @isL2rus IsL2rus ∈ (Inst × Bits5Nat × Bits4Nat) → BOOL
  @isL2rusDef ∀inst,lopc, lr_reg·inst∈Inst ∧ lopc∈Bits5Nat ∧  lr_reg∈12‥15
            ⇒ IsL2rus(inst↦lopc↦ lr_reg) =
              bool(XOPC_Nat(inst)=31 ∧ XSUBOPC_Nat(inst)∈0‥26 ∧ XLOPC_Nat(inst)=lopc ∧ XLSUBOPC_Nat(inst)=31 ∧
              XLBIT5_Nat(inst)=1 ∧ XLBIT4_OPC_Nat(inst)=0 ∧ XLR_REG_Nat(inst)= lr_reg ∧
              XL2rus_op1(inst)∈0‥11 ∧ XL2rus_op2(inst)∈0‥11 ∧ XL2rus_op3(inst)∈0‥11)

  @isRu6 IsRu6 ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
  @isRu6Def ∀inst,opc,uopc·inst∈Inst ∧ opc∈Bits5Nat ∧ uopc∈Bits1Nat
           ⇒ IsRu6(inst↦opc↦uopc) =
            bool(XOPC_Nat(inst)=opc ∧ XUOPC_Nat(inst)=uopc)

  @isLru6 IsLru6 ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
  @isLru6Def ∀inst,lopc,luopc·inst∈Inst ∧ lopc∈Bits5Nat ∧ luopc∈Bits1Nat
           ⇒ IsLru6(inst↦lopc↦luopc) =
             bool(XOPC_Nat(inst)=30 ∧ XUOPC_Nat(inst)=0 ∧ XLOPC_Nat(inst)=lopc ∧ XLUOPC_Nat(inst)=luopc)
```

```
@isU6 IsU6 ∈ (Inst × Bits5Nat × Bits1Nat × Bits4Nat) → BOOL
@isU6Def ∀inst,opc,uopc,ru6_reg·inst∈Inst ∧ opc∈14‥15 ∧ uopc∈Bits1Nat ∧ ru6_reg∈Bits4Nat
         ⇒ IsU6(inst↦opc↦uopc↦ru6_reg) =
         bool(XOPC_Nat(inst)=opc ∧ XUOPC_Nat(inst)=uopc ∧ XRU6_REG_Nat(inst)=ru6_reg)

@isLu6 IsLu6 ∈ (Inst × Bits5Nat × Bits1Nat × Bits4Nat) → BOOL
@isLu6Def ∀inst,lopc,luopc,lru6_reg·inst∈Inst ∧ lopc∈Bits5Nat ∧ luopc∈Bits1Nat ∧ lru6_reg∈Bits4Nat
          ⇒ IsLu6(inst↦lopc↦luopc↦lru6_reg) =
          bool(XOPC_Nat(inst)=30 ∧ XUOPC_Nat(inst)=0 ∧ XLOPC_Nat(inst)=lopc ∧ XLRU6_REG_Nat(inst)=lru6_reg ∧ XLUOPC_Nat(inst)=luopc)

@isU10 IsU10 ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
@isU10Def ∀inst,opc,uopc·inst∈Inst ∧ opc∈Bits5Nat ∧ uopc∈Bits1Nat
          ⇒ IsU10(inst↦opc↦uopc) = bool(XOPC_Nat(inst)=opc ∧ XUOPC_Nat(inst)=uopc)

@isLu10 IsLu10 ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
@isLu10Def ∀inst,lopc,luopc·inst∈Inst ∧ lopc∈Bits5Nat ∧ luopc∈Bits1Nat ⇒ IsLu10(inst↦lopc↦luopc) =
           bool(XOPC_Nat(inst)=30 ∧ XUOPC_Nat(inst)=0 ∧ XLOPC_Nat(inst)=lopc ∧ XLUOPC_Nat(inst)=luopc)

@is2r Is2r ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
@is2rDef ∀inst,opc,bit4_opc ·inst∈Inst ∧ opc∈Bits5Nat ∧ bit4_opc∈Bits1Nat
         ⇒ Is2r(inst↦opc↦bit4_opc) =
         bool(XOPC_Nat(inst)=opc∧XSUBOPC_Nat(inst)∈27‥31∧XBIT4_OPC_Nat(inst)=bit4_opc∧X2r_op1(inst)∈0‥11∧X2r_op2(inst)∈0‥11)

@isR2r IsR2r ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
@isR2rDef ∀inst,opc,bit4_opc ·inst∈Inst ∧ opc∈Bits5Nat ∧ bit4_opc∈Bits1Nat
          ⇒ IsR2r(inst↦opc↦bit4_opc) =
          bool(XOPC_Nat(inst)=opc∧XSUBOPC_Nat(inst)∈27‥31∧XBIT4_OPC_Nat(inst)=bit4_opc∧XR2r_op1(inst)∈0‥11∧XR2r_op2(inst)∈0‥11)

@isL2r IsL2r ∈ (Inst × Bits5Nat × Bits1Nat × Bits4Nat) → BOOL
@isL2rDef ∀inst,lopc,bit4_opc,lr_reg·inst∈Inst ∧ lopc∈Bits5Nat ∧ bit4_opc∈Bits1Nat ∧ lr_reg∈12‥15
          ⇒ IsL2r(inst↦lopc↦bit4_opc↦lr_reg) =
          bool(XOPC_Nat(inst)=31∧XSUBOPC_Nat(inst)∈27‥31∧XLOPC_Nat(inst)=lopc∧XBIT4_OPC_Nat(inst)=bit4_opc∧XLR_REG_Nat(inst)=lr_reg∧
               XL2r_op1(inst)∈0‥11 ∧ XL2r_op2(inst)∈0‥11)

@isLr2r IsLr2r ∈ (Inst × Bits5Nat × Bits1Nat × Bits4Nat) → BOOL
@isLr2rDef ∀inst,lopc,bit4_opc,lr_reg·inst∈Inst ∧ lopc∈Bits5Nat ∧ bit4_opc∈Bits1Nat ∧lr_reg∈12‥15
           ⇒ IsLr2r(inst↦lopc↦bit4_opc↦lr_reg) =
              bool(XOPC_Nat(inst)=31 ∧ XSUBOPC_Nat(inst)∈27‥31 ∧ XLOPC_Nat(inst)=lopc ∧
                   XBIT4_OPC_Nat(inst)=bit4_opc ∧ XLR_REG_Nat(inst)=lr_reg)
```

```
  @isRus IsRus ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
  @isRusDef ∀inst,opc,bit4_opc·inst∈Inst ∧ opc∈Bits5Nat ∧ bit4_opc∈Bits1Nat
             ⇒ IsRus(inst↦opc↦bit4_opc) =
                 bool(XOPC_Nat(inst)=opc ∧ XSUBOPC_Nat(inst)∈27‥31 ∧ XBIT4_OPC_Nat(inst)=bit4_opc ∧ XRus_op2(inst)∈0‥11)

  @is1r Is1r ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
  @is1rDef ∀inst,opc,bit4_opc·inst∈Inst ∧ opc∈Bits5Nat ∧ bit4_opc∈Bits1Nat
             ⇒ Is1r(inst↦opc↦bit4_opc) =
                bool(XOPC_Nat(inst)=opc ∧ XSUBOPC_Nat(inst)=31 ∧ XBIT5_Nat(inst)=1 ∧
                       XBIT4_OPC_Nat(inst)=bit4_opc ∧ XR_REG_Nat(inst)∈0‥11)

  @is0r Is0r ∈ (Inst × Bits5Nat × Bits1Nat × Bits4Nat) → BOOL
  @is0rDef ∀inst,opc,bit4_opc,r_reg·inst∈Inst ∧ opc∈Bits5Nat ∧ bit4_opc∈Bits1Nat ∧ r_reg∈12‥15
            ⇒ Is0r(inst↦opc↦bit4_opc↦r_reg) =
           bool(XOPC_Nat(inst)=opc ∧ XSUBOPC_Nat(inst)=31 ∧ XBIT5_Nat(inst)=1 ∧ XBIT4_OPC_Nat(inst)=bit4_opc ∧ XR_REG_Nat(inst)=r_reg)

  @isL4r IsL4r ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
  @isL4rDef ∀inst,lopc,lbit4_opc·inst∈Inst ∧ lopc∈Bits5Nat ∧lbit4_opc∈Bits1Nat
             ⇒ IsL4r(inst↦lopc↦lbit4_opc) =
             bool(XOPC_Nat(inst)=31 ∧ XSUBOPC_Nat(inst)∈0‥26 ∧
                  XLOPC_Nat(inst)=lopc ∧ XLSUBOPC_Nat(inst)=31 ∧
                  XLBIT5_Nat(inst)=1 ∧ XLBIT4_OPC_Nat(inst)=lbit4_opc ∧
                  XLR_REG_Nat(inst)∈0‥11)

  @isL5r IsL5r ∈ (Inst × Bits5Nat × Bits1Nat) → BOOL
  @isL5rDef ∀inst,lopc,lbit4_opc·inst∈Inst ∧ lopc∈Bits5Nat ∧lbit4_opc∈Bits1Nat
             ⇒ IsL5r(inst↦lopc↦lbit4_opc) =
             bool(XOPC_Nat(inst)=31 ∧ XSUBOPC_Nat(inst)∈0‥26 ∧
                  XLOPC_Nat(inst)=lopc ∧ XLSUBOPC_Nat(inst)∈27‥31 ∧
                  XLBIT5_Nat(inst)=1 ∧ XLBIT4_OPC_Nat(inst)=lbit4_opc ∧
                  XL5r_op4(inst)∈0‥11 ∧ XL5r_op5(inst)∈0‥11)

  @isL6r IsL6r ∈ (Inst × Bits5Nat) → BOOL
  @isL6rDef ∀inst,lopc·inst∈Inst ∧ lopc∈Bits5Nat
             ⇒ IsL6r(inst↦lopc) =
             bool(XOPC_Nat(inst)=31 ∧ XSUBOPC_Nat(inst)∈0‥26 ∧ XLOPC_Nat(inst)=lopc ∧ XLSUBOPC_Nat(inst)∈0‥26)

end
```

14.2.45context XCoreOperands

```
context XCoreOperands
extends XCoreSubOperands

constants X3r_op1 X3r_op2 X3r_op3
          XL3r_op1 XL3r_op2 XL3r_op3
          X2rus_op1 X2rus_op2 X2rus_op3
          XL2rus_op1 XL2rus_op2 XL2rus_op3
          XRu6_op1 XRu6_op2 XLru6_op1 XLru6_op2
          XU6_op1
          XLu6_op1
          XU10_op1
          XLu10_op1
          X2r_op1 X2r_op2
          XR2r_op1 XR2r_op2
          XL2r_op1 XL2r_op2
          XLr2r_op1 XLr2r_op2
          XRus_op1 XRus_op2
          X1r_op1
          XL4r_op1 XL4r_op2 XL4r_op3 XL4r_op4
          XL5r_op1 XL5r_op2 XL5r_op3 XL5r_op4 XL5r_op5
          XL6r_op1 XL6r_op2 XL6r_op3 XL6r_op4 XL6r_op5 XL6r_op6

axioms

  @x3r_op1 X3r_op1 ∈ Inst → Bits4Nat
  @x3r_op1Def ∀i·i∈Inst ⇒ X3r_op1(i)=Bits4ToNat(TwoBits2ToBits4(X3R_DST(i)↦Decode3rSubOpc_op1(i)))

  @x3r_op2 X3r_op2 ∈ Inst → Bits4Nat
  @x3r_op2Def ∀i·i∈Inst ⇒ X3r_op2(i)=Bits4ToNat(TwoBits2ToBits4(X3R_SRC1(i)↦Decode3rSubOpc_op2(i)))

  @x3r_op3 X3r_op3 ∈ Inst → Bits4Nat
  @x3r_op3Def ∀i·i∈Inst ⇒ X3r_op3(i)=Bits4ToNat(TwoBits2ToBits4(X3R_SRC2(i)↦Decode3rSubOpc_op3(i)))

  @xL3r_op1 XL3r_op1 ∈ Inst → Bits4Nat
  @xL3r_op1Def ∀i·i∈Inst ⇒ XL3r_op1(i)=X3r_op1(i) // Same decoding as short 3r format

  @xL3r_op2 XL3r_op2 ∈ Inst → Bits4Nat
  @xL3r_op2Def ∀i·i∈Inst ⇒ XL3r_op2(i)=X3r_op2(i) // Same decoding as short 3r format

  @xL3r_op3 XL3r_op3 ∈ Inst → Bits4Nat
  @xL3r_op3Def ∀i·i∈Inst ⇒ XL3r_op3(i)=X3r_op3(i) // Same decoding as short 3r format
```

```
@x2rus_op1 X2rus_op1 ∈ Inst → Bits4Nat
@x2rus_op1Def ∀i·i∈Inst ⇒ X2rus_op1(i)=X3r_op1(i) // Same decoding as short 3r format

@x2rus_op2 X2rus_op2 ∈ Inst → Bits4Nat
@x2rus_op2Def ∀i·i∈Inst ⇒ X2rus_op2(i)=X3r_op2(i) // Same decoding as short 3r format

@x2rus_op3 X2rus_op3 ∈ Inst → Bits4Nat
@x2rus_op3Def ∀i·i∈Inst ⇒ X2rus_op3(i)=X3r_op3(i) // Same decoding as short 3r format

@xL2rus_op1 XL2rus_op1 ∈ Inst → Bits4Nat
@xL2rus_op1Def ∀i·i∈Inst ⇒ XL2rus_op1(i)=X3r_op1(i) // Same decoding as short 3r format

@xL2rus_op2 XL2rus_op2 ∈ Inst → Bits4Nat
@xL2rus_op2Def ∀i·i∈Inst ⇒ XL2rus_op2(i)=X3r_op2(i) // Same decoding as short 3r format

@xL2rus_op3 XL2rus_op3 ∈ Inst → Bits4Nat
@xL2rus_op3Def ∀i·i∈Inst ⇒ XL2rus_op3(i)=X3r_op3(i) // Same decoding as short 3r format

@xRu6_op1 XRu6_op1 ∈ Inst → Bits4Nat
@xRu6_op1Def ∀i·i∈Inst ⇒ XRu6_op1(i)=XRU6_REG_Nat(i)

@xRu6_op2 XRu6_op2 ∈ Inst → Bits4Nat
@xRu6_op2Def ∀i·i∈Inst ⇒ XRu6_op2(i)=XU6_Nat(i)

@xLru6_op1 XLru6_op1 ∈ Inst → Bits4Nat
@xLru6_op1Def ∀i·i∈Inst ⇒ XLru6_op1(i)=XLRU6_REG_Nat(i)

@xLru6_op2 XLru6_op2 ∈ Inst → Bits16Nat
@xLru6_op2Def ∀i·i∈Inst ⇒ XLru6_op2(i)=Bits16ToNat(ConcatenateBits(XLU6(i)↦6↦XLIMM(i)↦10))

@xu6_op1 XU6_op1 ∈ Inst → Bits6Nat
@xu6_op1Def ∀i·i∈Inst ⇒ XU6_op1(i)=XU6_Nat(i)

@xLu6_op1 XLu6_op1 ∈ Inst → Bits6Nat
@xLu6_op1Def ∀i·i∈Inst ⇒ XLu6_op1(i)=XLru6_op2(i) // Same decoding as Lru6 op2

@xu10_op1 XU10_op1 ∈ Inst → Bits10Nat
@xu10_op1Def ∀i·i∈Inst ⇒ XU10_op1(i)=XU10_Nat(i)

@xLu10_op1 XLu10_op1 ∈ Inst → Bits10Nat
@xLu10_op1Def ∀i·i∈Inst ⇒ XLu10_op1(i)=Bits20ToNat(ConcatenateBits(XLU10(i)↦10↦XLIMM(i)↦10))
```

```
@x2r_op1 X2r_op1 ∈ Inst → Bits4Nat
@x2r_op1Def ∀i·i∈Inst ⇒ X2r_op1(i)=Bits4ToNat(TwoBits2ToBits4(X2R_REG(i)↦Decode2rSubOpc_op1(i)))

@x2r_op2 X2r_op2 ∈ Inst → Bits4Nat
@x2r_op2Def ∀i·i∈Inst ⇒ X2r_op2(i)=Bits4ToNat(TwoBits2ToBits4(X2R_SRC(i)↦Decode2rSubOpc_op2(i)))

@xL2r_op1 XL2r_op1 ∈ Inst → Bits4Nat
@xL2r_op1Def ∀i·i∈Inst ⇒ XL2r_op1(i)=Bits4ToNat(TwoBits2ToBits4(X2R_REG(i)↦Decode2rSubOpc_op1(i)))

@xL2r_op2 XL2r_op2 ∈ Inst → Bits4Nat
@xL2r_op2Def ∀i·i∈Inst ⇒ XL2r_op2(i)=Bits4ToNat(TwoBits2ToBits4(X2R_SRC(i)↦Decode2rSubOpc_op2(i)))

@xR2r_op1 XR2r_op1 ∈ Inst → Bits4Nat
@xR2r_op1Def ∀i·i∈Inst ⇒ XR2r_op1(i)=Bits4ToNat(TwoBits2ToBits4(Xr2R_REG(i)↦Decode2rSubOpc_op1(i))) // Note Decode2rSubOpc_op1 not reversed

@xR2r_op2 XR2r_op2 ∈ Inst → Bits4Nat
@xR2r_op2Def ∀i·i∈Inst ⇒ XR2r_op2(i)=Bits4ToNat(TwoBits2ToBits4(Xr2R_SRC(i)↦Decode2rSubOpc_op2(i))) // Note Decode2rSubOpc_op1 not reversed

@xLr2r_op1 XLr2r_op1 ∈ Inst → Bits4Nat
@xLr2r_op1Def ∀i·i∈Inst ⇒ XLr2r_op1(i)=XL2r_op2(i) // Simple reversal of l2r

@xLr2r_op2 XLr2r_op2 ∈ Inst → Bits4Nat
@xLr2r_op2Def ∀i·i∈Inst ⇒ XLr2r_op2(i)=XL2r_op1(i) // Simple reversal of l2r

@xRus_op1 XRus_op1 ∈ Inst → Bits4Nat
@xRus_op1Def ∀i·i∈Inst ⇒ XRus_op1(i)=Bits4ToNat(TwoBits2ToBits4(X2R_REG(i)↦Decode2rSubOpc_op1(i)))

@xRus_op2 XRus_op2 ∈ Inst → Bits4Nat
@xRus_op2Def ∀i·i∈Inst ⇒ XRus_op2(i)=Bits4ToNat(TwoBits2ToBits4(X2R_SRC(i)↦Decode2rSubOpc_op2(i)))

@x1r_op1 X1r_op1 ∈ Inst → Bits4Nat
@x1r_op1Def ∀i·i∈Inst ⇒ X1r_op1(i)=Bits4ToNat(XR_REG(i))

@xL4r_op1 XL4r_op1 ∈ Inst → Bits4Nat
@xL4r_op1Def ∀i·i∈Inst ⇒ XL4r_op1(i)=X3r_op1(i)

@xL4r_op2 XL4r_op2 ∈ Inst → Bits4Nat
@xL4r_op2Def ∀i·i∈Inst ⇒ XL4r_op2(i)=X3r_op2(i)

@xL4r_op3 XL4r_op3 ∈ Inst → Bits4Nat
@xL4r_op3Def ∀i·i∈Inst ⇒ XL4r_op3(i)=X3r_op3(i)
```

```
  @xL4r_op4 XL4r_op4 ∈ Inst → Bits4Nat
  @xL4r_op4Def ∀i·i∈Inst ⇒ XL4r_op4(i)=XL4R_DST2_Nat(i)

  @xL5r_op1 XL5r_op1 ∈ Inst → Bits4Nat
  @xL5r_op1Def ∀i·i∈Inst ⇒ XL5r_op1(i)=X3r_op1(i)

  @xL5r_op2 XL5r_op2 ∈ Inst → Bits4Nat
  @xL5r_op2Def ∀i·i∈Inst ⇒ XL5r_op2(i)=X3r_op2(i)

  @xL5r_op3 XL5r_op3 ∈ Inst → Bits4Nat
  @xL5r_op3Def ∀i·i∈Inst ⇒ XL5r_op3(i)=X3r_op3(i)

  @xL5r_op4 XL5r_op4 ∈ Inst → Bits4Nat
  @xL5r_op4Def ∀i·i∈Inst ⇒ XL5r_op4(i)=Bits4ToNat(TwoBits2ToBits4(XL5R_DST2(i)↦DecodeL5rSubOpc_op4(i)))

  @xL5r_op5 XL5r_op5 ∈ Inst → Bits4Nat
  @xL5r_op5Def ∀i·i∈Inst ⇒ XL5r_op5(i)=Bits4ToNat(TwoBits2ToBits4(XL5R_SRC3(i)↦DecodeL5rSubOpc_op5(i)))

  @xL6r_op1 XL6r_op1 ∈ Inst → Bits4Nat
  @xL6r_op1Def ∀i·i∈Inst ⇒ XL6r_op1(i)=XL3r_op1(i)

  @xL6r_op2 XL6r_op2 ∈ Inst → Bits4Nat
  @xL6r_op2Def ∀i·i∈Inst ⇒ XL6r_op2(i)=XL3r_op2(i)

  @xL6r_op3 XL6r_op3 ∈ Inst → Bits4Nat
  @xL6r_op3Def ∀i·i∈Inst ⇒ XL6r_op3(i)=XL3r_op3(i)

  @xL6r_op4 XL6r_op4 ∈ Inst → Bits4Nat
  @xL6r_op4Def ∀i·i∈Inst ⇒ XL6r_op4(i)=Bits4ToNat(TwoBits2ToBits4(XL6R_DST2(i)↦DecodeL6rSubOpc_op4(i)))

  @xL6r_op5 XL6r_op5 ∈ Inst → Bits4Nat
  @xL6r_op5Def ∀i·i∈Inst ⇒ XL6r_op5(i)=Bits4ToNat(TwoBits2ToBits4(XL6R_SRC3(i)↦DecodeL6rSubOpc_op5(i)))

  @xL6r_op6 XL6r_op6 ∈ Inst → Bits4Nat
  @xL6r_op6Def ∀i·i∈Inst ⇒ XL6r_op6(i)=Bits4ToNat(TwoBits2ToBits4(XL6R_SRC4(i)↦DecodeL6rSubOpc_op6(i)))

end
```

14.2.46context XCoreSubOperands

```
context XCoreSubOperands

extends XCoreFields BitsConcatenation

constants Decode3r_op1 Decode3r_op2 Decode3r_op3
          Decode3rSubOpc_op1   Decode3rSubOpc_op2   Decode3rSubOpc_op3
          Decode2rusSubOpc_op1 Decode2rusSubOpc_op2 Decode2rusSubOpc_op3

          Decode2r_op1 Decode2r_op2
          Decode2rSubOpc_op1 Decode2rSubOpc_op2

          DecodeL5rSubOpc_op4 DecodeL5rSubOpc_op5

          DecodeL6rSubOpc_op4 DecodeL6rSubOpc_op5 DecodeL6rSubOpc_op6

axioms

    @decode3rSubOp1    Decode3r_op1 ∈ Bits5Nat→Bits2Nat
    @decode3rSubOp1Def ∀n·n∈Bits5Nat ⇒ Decode3r_op1(n) = (n mod 3)// Note swapped with Decode3r_op3 in published spec
    @decode3rSubOp2    Decode3r_op2 ∈ Bits5Nat→Bits2Nat
    @decode3rSubOp2Def ∀n·n∈Bits5Nat ⇒ Decode3r_op2(n) = ((n÷3) mod 3)
    @decode3rSubOp3    Decode3r_op3 ∈ Bits5Nat→Bits2Nat
    @decode3rSubOp3Def ∀n·n∈Bits5Nat ⇒ Decode3r_op3(n) = (n÷9)

    @decode3rSubOpc_op1   Decode3rSubOpc_op1 ∈ Inst → Bits
    @decode3rSubOpc_op1Def ∀i·i∈Inst ⇒ Decode3rSubOpc_op1(i)=NatToBits2(Decode3r_op1(XSUBOPC_Nat(i)))
    @decode3rSubOpc_op2   Decode3rSubOpc_op2 ∈ Inst → Bits
    @decode3rSubOpc_op2Def ∀i·i∈Inst ⇒ Decode3rSubOpc_op2(i)=NatToBits2(Decode3r_op2(XSUBOPC_Nat(i)))
    @decode3rSubOpc_op3   Decode3rSubOpc_op3 ∈ Inst → Bits
    @decode3rSubOpc_op3Def ∀i·i∈Inst ⇒ Decode3rSubOpc_op3(i)=NatToBits2(Decode3r_op3(XSUBOPC_Nat(i)))

    @decode2rusSubOpc_op1   Decode2rusSubOpc_op1 ∈ Inst → Bits
    @decode2rusSubOpc_op1Def ∀i·i∈Inst ⇒ Decode2rusSubOpc_op1(i)=NatToBits2(Decode3r_op3(XSUBOPC_Nat(i)))
    @decode2rusSubOpc_op2   Decode2rusSubOpc_op2 ∈ Inst → Bits
    @decode2rusSubOpc_op2Def ∀i·i∈Inst ⇒ Decode2rusSubOpc_op2(i)=NatToBits2(Decode3r_op2(XSUBOPC_Nat(i)))
    @decode2rusSubOpc_op3   Decode2rusSubOpc_op3 ∈ Inst → Bits
    @decode2rusSubOpc_op3Def ∀i·i∈Inst ⇒ Decode2rusSubOpc_op3(i)=NatToBits2(Decode3r_op1(XSUBOPC_Nat(i)))

    @decode2rSubOp1     Decode2r_op1 ∈ Bits6Nat→Bits2Nat
    @decode2rSubOp1Def  ∀n·n∈Bits6Nat ∧ n≥27 ⇒ Decode2r_op1(n) = (((n-27) mod 27) mod 3)
    @decode2rSubOp2     Decode2r_op2 ∈ Bits6Nat→Bits2Nat
    @decode2rSubOp2Def  ∀n·n∈Bits6Nat ∧ n≥27 ⇒ Decode2r_op2(n) = (((n-27) mod 27) ÷ 3)
```

```
    @decode2rSubOpc_op1     Decode2rSubOpc_op1 ∈ Inst → Bits
    @decode2rSubOpc_op1Def ∀i·i∈Inst
                           ⇒ Decode2rSubOpc_op1(i)=NatToBits2(Decode2r_op1(Bits6ToNat(ConcatenateBits(XSUBOPC(i)↦5↦XBIT5(i)↦1))))
    @decode2rSubOpc_op2     Decode2rSubOpc_op2 ∈ Inst → Bits
    @decode2rSubOpc_op2Def ∀i·i∈Inst
                           ⇒ Decode2rSubOpc_op2(i)=NatToBits2(Decode2r_op2(Bits6ToNat(ConcatenateBits(XSUBOPC(i)↦5↦XBIT5(i)↦1))))

    @decodeL5rSubOpc_op1    DecodeL5rSubOpc_op4 ∈ Inst → Bits
    @decodeL5rSubOpc_op1Def ∀i·i∈Inst
                           ⇒ DecodeL5rSubOpc_op4(i)=NatToBits2(Decode2r_op1(Bits6ToNat(ConcatenateBits(XLSUBOPC(i)↦5↦XLBIT5(i)↦1))))
    @decodeL5rSubOpc_op2    DecodeL5rSubOpc_op5 ∈ Inst → Bits
    @decodeL5rSubOpc_op2Def ∀i·i∈Inst
                           ⇒ DecodeL5rSubOpc_op5(i)=NatToBits2(Decode2r_op2(Bits6ToNat(ConcatenateBits(XLSUBOPC(i)↦5↦XLBIT5(i)↦1))))

    @decodeL6rSubOpc_op4    DecodeL6rSubOpc_op4 ∈ Inst → Bits
    @decodeL6rSubOpc_op4Def ∀i·i∈Inst ⇒ DecodeL6rSubOpc_op4(i)=NatToBits2(Decode3r_op1(XLSUBOPC_Nat(i)))
    @decodeL6rSubOpc_op5    DecodeL6rSubOpc_op5 ∈ Inst → Bits
    @decodeL6rSubOpc_op5Def ∀i·i∈Inst ⇒ DecodeL6rSubOpc_op5(i)=NatToBits2(Decode3r_op2(XLSUBOPC_Nat(i)))
    @decodeL6rSubOpc_op6    DecodeL6rSubOpc_op6 ∈ Inst → Bits
    @decodeL6rSubOpc_op6Def ∀i·i∈Inst ⇒ DecodeL6rSubOpc_op6(i)=NatToBits2(Decode3r_op3(XLSUBOPC_Nat(i)))

end
```

14.2.47context XCoreInstSize

```
context XCoreInstSize // Instruction size function for XCore

extends XCoreFields

constants InstrSize InstToSize InstSizeShort InstSizeLong

axioms

  @instSizeShort InstSizeShort = 2
  @instSizeLong  InstSizeLong  = 4

  @instSizeDef InstrSize = {2,4}
  @instSizeTypeDef InstToSize ∈ Bits → InstrSize
  @instSizeDef2 ∀i·i∈Inst ⇒ (XOPC_Nat(i)=31 ⇒ InstToSize(i)=InstSizeLong)
  @instSizeDef3 ∀i·i∈Inst ⇒ (XOPC_Nat(i)≠31 ⇒ InstToSize(i)=InstSizeShort)

end
```

14.2.48context XCoreLocks

```
context XCoreLocks

extends XCoreResources ArraySearch

constants LockStatus
          MaxLocks MaxLockIndex LockIndexDom
          GetLockIdent IsLockIdent
          IsLockInUse
          FindFreeLock CountFreeLocks

axioms

  @lockStatusDef LockStatus = {FREE,RUNNING}
  theorem @lockStatusType LockStatus ⊆ℕ

  @maxLocks MaxLocks ∈ ℕ1 // Must be at least one
  @maxLockIndex MaxLockIndex = MaxLocks−1
  @synchroniserIndexDom LockIndexDom = 0‥(MaxLocks−1)

  @maxLocksSeven MaxLocks = 4

  @getLockIdType   GetLockIdent ∈ Bits8Nat → Bits
  @getLockIdDef           ∀i·i∈Bits8Nat ⇒ GetLockIdent(i)=GetResourceId(i↦ResourceLock)

  @isLockIdentType IsLockIdent ∈ Bits → BOOL
  @isLockIdentDef  ∀i·i∈Bits  ⇒ IsLockIdent(i)=
                          bool(GetResourceType(i)=ResourceLock ∧ GetResourceIndex(i)∈LockIndexDom)

  @isLockInUseType IsLockInUse ∈ (Bits × (LockIndexDom → LockStatus)) → BOOL
  @isLockInUseDef  ∀i,s·i∈Bits ∧ s∈LockIndexDom → LockStatus  ⇒ IsLockInUse(i↦s)=
                          bool(GetResourceIndex(i)∈LockIndexDom ∧ s(GetResourceIndex(i))=RUNNING)

  @findFreeType FindFreeLock ∈ (LockIndexDom→LockStatus) → LockIndexDom
  @findFreeDef  ∀s·(s∈LockIndexDom→LockStatus) ⇒ FindFreeLock(s)=FindInArray(s↦MaxLocks↦FREE)

  @countFreeType CountFreeLocks ∈ (LockIndexDom→LockStatus) → 0‥MaxLocks
  @countFreeDef  ∀s·(s∈LockIndexDom→LockStatus) ⇒ CountFreeLocks(s)=CountInArray(s↦MaxLocks↦FREE)

end
```

14.2.49context XCoreMemArrayDom

```
context XCoreMemArrayDom
extends MemArrayDomE4

axioms
  @minMemIndex MinMemIndex = 65536 // 0x10000
  @memArraySize MemArraySize = 65536 // 0x10000
  @xCoreBootVectorDef XCoreBootVector = MinMemIndex //
End
```

14.2.50context XCoreMemArray

```
context XCoreMemArray

extends ArithmeticTheorems
        MemArrayE3 XCoreMemArrayDom

axioms
  theorem @memDomInBits32Nats MemArrayDom⊆Bits32Nat
end
```

14.2.51context XCorePorts

```
context XCorePorts
extends XCoreResources ArraySearch Fifo
constants PortStatus
          MaxPorts MaxPortIndex PortIndexDom
          IsPortIdent IsOneBitPortIdent IsPortInUse
          IsPortControl

axioms
  @timerStatusDef PortStatus = {FREE,RUNNING}
  theorem @portStatusType PortStatus ⊆ℕ

  @maxPorts MaxPorts ∈ ℕ1 // Must be at least one
  @maxPortIndex MaxPortIndex = MaxPorts-1
  @synchroniserIndexDom PortIndexDom = 0‥(MaxPorts-1)

  @maxPortsSeven MaxPorts = 4

  @isPortIdentType IsPortIdent ∈ Bits → BOOL
  @isPortIdentDef  ∀i·i∈Bits  ⇒ IsPortIdent(i)=
                          bool(GetResourceType(i)=ResourcePort ∧ GetResourceIndex(i)∈PortIndexDom)

  @isOneBitPortIdentType IsOneBitPortIdent ∈ Bits → BOOL
  @isOneBitPortIdentDef  ∀i·i∈Bits  ⇒ IsOneBitPortIdent(i)=bool(GetResourceIndex(i)=1) // Arbitrary place holder

  @isPortInUseType IsPortInUse ∈ (Bits × (PortIndexDom → PortStatus)) → BOOL
  @isPortInUseDef  ∀i,s·i∈Bits ∧ s∈PortIndexDom → PortStatus
                   ⇒ IsPortInUse(i↦s)=bool(GetResourceIndex(i)∈PortIndexDom ∧ s(GetResourceIndex(i))≠FREE)

  @isPortControlType IsPortControl ∈ Bits32Nat → BOOL
  @isPortControlDef  ∀c·c∈Bits32Nat ⇒ IsPortControl(c)=
                     bool(c=CtrlInUseOn ∨ c=CtrlInUseOff ∨
                          c=CtrlCondNone ∨ c=CtrlCondAfter ∨
                          c=CtrlIeModeEvent ∨ c=CtrlIeModeInterrupt ∨
                          c=CtrlDriveDrive ∨ c=CtrlDrivePullDown ∨ c=CtrlDrivePullUp ∨
                          c=CtrlDriveRunStopR ∨ c=CtrlDriveRunStartR ∨ c=CtrlDriveRunClrBuf ∨
                          c=CtrlRunStopR ∨ c=CtrlRunStartR ∨ c=CtrlRunClrBuf ∨
                          c=CtrlMsMaster ∨ c=CtrlMsSlave ∨
                          c=CtrlBufNoBuffers ∨ c=CtrlBufBuffers ∨
                          c=CtrlRdyNoReady ∨ c=CtrlRdyStrobed ∨ c=CtrlRdyHandshake ∨
                          c=CtrlSdelayNoSdelay ∨ c=CtrlSdelaySdelay ∨
                          c=CtrlPortDataport ∨ c=CtrlPortClockport ∨ c=CtrlPortReadyport ∨
```

```
                    c=CtrlInvNoInvert V c=CtrlInvInvert)
end
```

14.2.52context XCoreProcessorState

```
context XCoreProcessorState

extends XCoreResources ProcessorState BitsManipulation

constants IsProcessorStateIdent GetProcessorStateIndex
          PsRamBaseIndex PsVectorBaseIndex PsXCoreCtrl0Index
          PsBootModePins PsReserved PsOtpSecurity PsWoOscillator PsCoci PsCowi PsPeci PsPewi PsPwrWakeCount PsPwrWake
          PsDbgSsrIndex PsDbgSpcIndex PsDbgSspIndex PsDbgTnumIndex PsDbgTregIndex PsDbgTypeIndex PsDbgDataIndex
          PsDbgRunCtrl
          PsDbgScratch0 PsDbgScratch1 PsDbgScratch2 PsDbgScratch3 PsDbgScratch4 PsDbgScratch5 PsDbgScratch6 PsDbgScratch7
          PsDbgIbreakAddr0 PsDbgIbreakAddr1 PsDbgIbreakAddr2 PsDbgIbreakAddr3
          PsDbgIbreakCtrl0 PsDbgIbreakCtrl1 PsDbgIbreakCtrl2 PsDbgIbreakCtrl3
          PsDbgDwatchAddr10 PsDbgDwatchAddr11 PsDbgDwatchAddr12 PsDbgDwatchAddr13
          PsDbgDwatchAddr20 PsDbgDwatchAddr21 PsDbgDwatchAddr22 PsDbgDwatchAddr23
          PsDbgDwatchCtrl0  PsDbgDwatchCtrl1  PsDbgDwatchCtrl2  PsDbgDwatchCtrl3
          PsDbgRwatchAddr10 PsDbgRwatchAddr11 PsDbgRwatchAddr12 PsDbgRwatchAddr13
          PsDbgRwatchAddr20 PsDbgRwatchAddr21 PsDbgRwatchAddr22 PsDbgRwatchAddr23
          PsDbgRwatchCtrl0  PsDbgRwatchCtrl1  PsDbgRwatchCtrl2  PsDbgRwatchCtrl3
          PsDbgEntryIndex
          IsNormalReadProcessorStateIndex    IsNormalWriteProcessorStateIndex
          IsDebugReadProcessorStateIndex     IsDebugWriteProcessorStateIndex
          IsRunningReadProcessorStateIndex   IsRunningWriteProcessorStateIndex
          IsDebuggingReadProcessorStateIndex IsDebuggingWriteProcessorStateIndex

axioms

  @ramBase     PsRamBaseIndex        = 0  // 0x00
  @vectorBase  PsVectorBaseIndex     = 1  // 0x01
  @ctrl0Base   PsXCoreCtrl0Index     = 2  // 0x02
  @btModePins  PsBootModePins        = 3  // 0x03
  @reserved    PsReserved            = 4  // 0x04
  @otpSecurity PsOtpSecurity         = 5  // 0x05
  @woOscill    PsWoOscillator        = 6  // 0x06
  @coci        PsCoci                = 7  // 0x07
  @cowi        PsCowi                = 8  // 0x08
  @peci        PsPeci                = 9  // 0x09
  @pewi        PsPewi                = 10 // 0x0a
  @pwrWakeCt   PsPwrWakeCount        = 11 // 0x0b
  @pwrWake     PsPwrWake             = 12 // 0x0c

  @dssrIndex   PsDbgSsrIndex         = 16 // 0x10
  @dspcIndex   PsDbgSpcIndex         = 17 // 0x11
  @dsspIndex   PsDbgSspIndex         = 18 // 0x12
```

```
@dtidIndex    PsDbgTnumIndex        = 19 // 0x13
@dtregIndex   PsDbgTregIndex        = 20 // 0x14
@dtypeIndex   PsDbgTypeIndex        = 21 // 0x15
@dDataIndex   PsDbgDataIndex        = 22 // 0x16
@dbgRunCtrl   PsDbgRunCtrl          = 24 // 0x18

@dbgScratch0 PsDbgScratch0          = 32 // 0x20
@dbgScratch1 PsDbgScratch1          = 33 // 0x21
@dbgScratch2 PsDbgScratch2          = 34 // 0x22
@dbgScratch3 PsDbgScratch3          = 35 // 0x23
@dbgScratch4 PsDbgScratch4          = 36 // 0x24
@dbgScratch5 PsDbgScratch5          = 37 // 0x25
@dbgScratch6 PsDbgScratch6          = 38 // 0x26
@dbgScratch7 PsDbgScratch7          = 39 // 0x27

@dbgIbkAddr0 PsDbgIbreakAddr0       = 48 // 0x30
@dbgIbkAddr1 PsDbgIbreakAddr1       = 49 // 0x31
@dbgIbkAddr2 PsDbgIbreakAddr2       = 50 // 0x32
@dbgIbkAddr3 PsDbgIbreakAddr3       = 51 // 0x33

@dbgIbkCtrl0 PsDbgIbreakCtrl0       = 64 // 0x40
@dbgIbkCtrl1 PsDbgIbreakCtrl1       = 65 // 0x41
@dbgIbkCtrl2 PsDbgIbreakCtrl2       = 66 // 0x42
@dbgIbkCtrl3 PsDbgIbreakCtrl3       = 67 // 0x43

@dbgDwAddr10 PsDbgDwatchAddr10       = 80 // 0x50
@dbgDwAddr11 PsDbgDwatchAddr11       = 81 // 0x51
@dbgDwAddr12 PsDbgDwatchAddr12       = 82 // 0x52
@dbgDwAddr13 PsDbgDwatchAddr13       = 83 // 0x53

@dbgDwAddr20 PsDbgDwatchAddr20       = 96 // 0x60
@dbgDwAddr21 PsDbgDwatchAddr21       = 97 // 0x61
@dbgDwAddr22 PsDbgDwatchAddr22       = 98 // 0x62
@dbgDwAddr23 PsDbgDwatchAddr23       = 99 // 0x63

@dbgDwCtrl0  PsDbgDwatchCtrl0        = 112 // 0x70
@dbgDwCtrl1  PsDbgDwatchCtrl1        = 113 // 0x71
@dbgDwCtrl2  PsDbgDwatchCtrl2        = 114 // 0x72
@dbgDwCtrl3  PsDbgDwatchCtrl3        = 115 // 0x73

@dbgRwAddr10 PsDbgRwatchAddr10       = 128 // 0x80
@dbgRwAddr11 PsDbgRwatchAddr11       = 129 // 0x81
@dbgRwAddr12 PsDbgRwatchAddr12       = 130 // 0x82
@dbgRwAddr13 PsDbgRwatchAddr13       = 131 // 0x83
```

```
@dbgRwAddr20 PsDbgRwatchAddr20      = 144 // 0x90
@dbgRwAddr21 PsDbgRwatchAddr21      = 145 // 0x91
@dbgRwAddr22 PsDbgRwatchAddr22      = 146 // 0x92
@dbgRwAddr23 PsDbgRwatchAddr23      = 147 // 0x93

@dbgRwCtrl0  PsDbgRwatchCtrl0       = 160 // 0xa0
@dbgRwCtrl1  PsDbgRwatchCtrl1       = 161 // 0xa1
@dbgRwCtrl2  PsDbgRwatchCtrl2       = 162 // 0xa2
@dbgRwCtrl3  PsDbgRwatchCtrl3       = 163 // 0xa3

@dentIndex   PsDbgEntryIndex       = 32 // 0x20

@maxIndex   ProcStateSize = 164 // 0xa3 + 1

theorem @procStateDom ProcStateDom = 0‥163
theorem @ramBaseInPsDom     PsRamBaseIndex    ∈ ProcStateDom
theorem @vectorBaseInPsDom  PsVectorBaseIndex ∈ ProcStateDom
theorem @ctrl0BaseInPsDom   PsXCoreCtrl0Index ∈ ProcStateDom
theorem @btModePinsInPsDom  PsBootModePins    ∈ ProcStateDom
theorem @reservedInPsDom    PsReserved        ∈ ProcStateDom
theorem @otpSecurityInPsDom PsOtpSecurity     ∈ ProcStateDom
theorem @woOscillInPsDom    PsWoOscillator    ∈ ProcStateDom
theorem @cociInPsDom        PsCoci            ∈ ProcStateDom
theorem @cowiInPsDom        PsCowi            ∈ ProcStateDom
theorem @peciInPsDom        PsPeci            ∈ ProcStateDom
theorem @pewiInPsDom        PsPewi            ∈ ProcStateDom
theorem @pwrWakeCtInPsDom   PsPwrWakeCount    ∈ ProcStateDom
theorem @pwrWakeInPsDom     PsPwrWake         ∈ ProcStateDom

theorem @dssrIndexInPsDom   PsDbgSsrIndex     ∈ ProcStateDom
theorem @dspcIndexInPsDom   PsDbgSpcIndex     ∈ ProcStateDom
theorem @dsspIndexInPsDom   PsDbgSspIndex     ∈ ProcStateDom
theorem @dtidIndexInPsDom   PsDbgTnumIndex    ∈ ProcStateDom
theorem @dtregIndexInPsDom  PsDbgTregIndex    ∈ ProcStateDom
theorem @dtypeIndexInPsDom  PsDbgTypeIndex    ∈ ProcStateDom
theorem @dDataIndexInPsDom  PsDbgDataIndex    ∈ ProcStateDom
theorem @dbgRunCtrlInPsDom  PsDbgRunCtrl      ∈ ProcStateDom

theorem @dbgScratch0InPsDom PsDbgScratch0     ∈ ProcStateDom
theorem @dbgScratch1InPsDom PsDbgScratch1     ∈ ProcStateDom
theorem @dbgScratch2InPsDom PsDbgScratch2     ∈ ProcStateDom
theorem @dbgScratch3InPsDom PsDbgScratch3     ∈ ProcStateDom
```

```
theorem @dbgScratch4InPsDom PsDbgScratch4          ∈ ProcStateDom
theorem @dbgScratch5InPsDom PsDbgScratch5          ∈ ProcStateDom
theorem @dbgScratch6InPsDom PsDbgScratch6          ∈ ProcStateDom
theorem @dbgScratch7InPsDom PsDbgScratch7          ∈ ProcStateDom

theorem @dbgIbkAddr0InPsDom PsDbgIbreakAddr0       ∈ ProcStateDom
theorem @dbgIbkAddr1InPsDom PsDbgIbreakAddr1       ∈ ProcStateDom
theorem @dbgIbkAddr2InPsDom PsDbgIbreakAddr2       ∈ ProcStateDom
theorem @dbgIbkAddr3InPsDom PsDbgIbreakAddr3       ∈ ProcStateDom

theorem @dbgIbkCtrl0InPsDom PsDbgIbreakCtrl0       ∈ ProcStateDom
theorem @dbgIbkCtrl1InPsDom PsDbgIbreakCtrl1       ∈ ProcStateDom
theorem @dbgIbkCtrl2InPsDom PsDbgIbreakCtrl2       ∈ ProcStateDom
theorem @dbgIbkCtrl3InPsDom PsDbgIbreakCtrl3       ∈ ProcStateDom

theorem @dbgDwAddr10InPsDom PsDbgDwatchAddr10       ∈ ProcStateDom
theorem @dbgDwAddr11InPsDom PsDbgDwatchAddr11       ∈ ProcStateDom
theorem @dbgDwAddr12InPsDom PsDbgDwatchAddr12       ∈ ProcStateDom
theorem @dbgDwAddr13InPsDom PsDbgDwatchAddr13       ∈ ProcStateDom

theorem @dbgDwAddr20InPsDom PsDbgDwatchAddr20       ∈ ProcStateDom
theorem @dbgDwAddr21InPsDom PsDbgDwatchAddr21       ∈ ProcStateDom
theorem @dbgDwAddr22InPsDom PsDbgDwatchAddr22       ∈ ProcStateDom
theorem @dbgDwAddr23InPsDom PsDbgDwatchAddr23       ∈ ProcStateDom

theorem @dbgDwCtrl0InPsDom  PsDbgDwatchCtrl0        ∈ ProcStateDom
theorem @dbgDwCtrl1InPsDom  PsDbgDwatchCtrl1        ∈ ProcStateDom
theorem @dbgDwCtrl2InPsDom  PsDbgDwatchCtrl2        ∈ ProcStateDom
theorem @dbgDwCtrl3InPsDom  PsDbgDwatchCtrl3        ∈ ProcStateDom

theorem @dbgRwAddr10InPsDom PsDbgRwatchAddr10       ∈ ProcStateDom
theorem @dbgRwAddr11InPsDom PsDbgRwatchAddr11       ∈ ProcStateDom
theorem @dbgRwAddr12InPsDom PsDbgRwatchAddr12       ∈ ProcStateDom
theorem @dbgRwAddr13InPsDom PsDbgRwatchAddr13       ∈ ProcStateDom

theorem @dbgRwAddr20InPsDom PsDbgRwatchAddr20       ∈ ProcStateDom
theorem @dbgRwAddr21InPsDom PsDbgRwatchAddr21       ∈ ProcStateDom
theorem @dbgRwAddr22InPsDom PsDbgRwatchAddr22       ∈ ProcStateDom
theorem @dbgRwAddr23InPsDom PsDbgRwatchAddr23       ∈ ProcStateDom

theorem @dbgRwCtrl0InPsDom  PsDbgRwatchCtrl0        ∈ ProcStateDom
theorem @dbgRwCtrl1InPsDom  PsDbgRwatchCtrl1        ∈ ProcStateDom
```

```
theorem @dbgRwCtrl2InPsDom  PsDbgRwatchCtrl2       ∈ ProcStateDom
theorem @dbgRwCtrl3InPsDom  PsDbgRwatchCtrl3       ∈ ProcStateDom


@getProcessorStateIndex GetProcessorStateIndex ∈ ResourceId → Bits8Nat
@getProcessorStateIndexDef ∀i·i∈Bits ⇒ GetProcessorStateIndex(i) = (Bits8ToNat(ExtractBits8_1(i))) // Different to standard resource indexing

@isProcessorStateIdentType IsProcessorStateIdent ∈ (Bits) → BOOL
@isProcessorStateIdentDef  ∀i·i∈Bits ⇒ IsProcessorStateIdent(i)=
                               bool(GetResourceType(i)=ResourceProcState ∧ GetProcessorStateIndex(i)∈ProcStateDom)

@isNormReadProcStateIndexType IsNormalReadProcessorStateIndex ∈ (Bits8Nat) → BOOL
@isNormReadProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsNormalReadProcessorStateIndex(i)=
                               bool((i=PsRamBaseIndex)     ∨ (i=PsVectorBaseIndex) ∨
                                    (i=PsXCoreCtrl0Index) ∨ (i=PsBootModePins)     ∨
                                    (i=PsReserved)         ∨ (i=PsOtpSecurity)      ∨
                                    (i=PsCoci) ∨ (i=PsCowi) ∨ (i=PsPeci) ∨ (i=PsPewi) ∨
                                    (i=PsPwrWakeCount) ∨ (i=PsPwrWake))

@isNormWriteProcStateIndexType IsNormalWriteProcessorStateIndex ∈ (Bits8Nat) → BOOL
@isNormWriteProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsNormalWriteProcessorStateIndex(i)=
                               bool((i=PsRamBaseIndex)     ∨ (i=PsVectorBaseIndex)    ∨
                                    (i=PsXCoreCtrl0Index) ∨
                                    (i=PsReserved)         ∨ (i=PsOtpSecurity)∨
                                    (i=PsWoOscillator)     ∨
                                    (i=PsPwrWakeCount)     ∨ (i=PsPwrWake))

@isDbgReadProcStateIndexType IsDebugReadProcessorStateIndex ∈ (Bits8Nat) → BOOL
@isDbgReadProcStateIndexDef      ∀i·i∈Bits8Nat ⇒ IsDebugReadProcessorStateIndex(i)=
                               bool((i=PsDbgSsrIndex)   ∨ (i=PsDbgSpcIndex) ∨
                                    (i=PsDbgSspIndex)   ∨ (i=PsDbgTnumIndex)∨
                                    (i=PsDbgTregIndex) ∨ (i=PsDbgTypeIndex)∨
                                    (i=PsDbgDataIndex) ∨ (i=PsDbgRunCtrl)   ∨
                                    (i=PsDbgScratch0)   ∨ (i=PsDbgScratch1) ∨ (i=PsDbgScratch2) ∨ (i=PsDbgScratch3) ∨
                                    (i=PsDbgScratch4)   ∨ (i=PsDbgScratch5) ∨ (i=PsDbgScratch6) ∨ (i=PsDbgScratch7) ∨
                                    (i=PsDbgIbreakAddr0)   ∨ (i=PsDbgIbreakAddr1)   ∨ (i=PsDbgIbreakAddr2)   ∨ (i=PsDbgIbreakAddr3)   ∨
                                    (i=PsDbgIbreakCtrl0)   ∨ (i=PsDbgIbreakCtrl1)   ∨ (i=PsDbgIbreakCtrl2)   ∨ (i=PsDbgIbreakCtrl3)   ∨
                                    (i=PsDbgDwatchAddr10) ∨ (i=PsDbgDwatchAddr11) ∨ (i=PsDbgDwatchAddr12) ∨ (i=PsDbgDwatchAddr13) ∨
                                    (i=PsDbgDwatchAddr20) ∨ (i=PsDbgDwatchAddr21) ∨ (i=PsDbgDwatchAddr22) ∨ (i=PsDbgDwatchAddr23) ∨
                                    (i=PsDbgDwatchCtrl0)   ∨ (i=PsDbgDwatchCtrl1)   ∨ (i=PsDbgDwatchCtrl2)   ∨ (i=PsDbgDwatchCtrl3)   ∨
                                    (i=PsDbgRwatchAddr10) ∨ (i=PsDbgRwatchAddr11) ∨ (i=PsDbgRwatchAddr12) ∨ (i=PsDbgRwatchAddr13) ∨
                                    (i=PsDbgRwatchAddr20) ∨ (i=PsDbgRwatchAddr21) ∨ (i=PsDbgRwatchAddr22) ∨ (i=PsDbgRwatchAddr23) ∨
```

```
                                    (i=PsDbgRwatchCtrl0)  ∨ (i=PsDbgRwatchCtrl1)  ∨ (i=PsDbgRwatchCtrl2)  ∨ (i=PsDbgRwatchCtrl3))

  @isDbgWriteProcStateIndexType IsDebugWriteProcessorStateIndex ∈ (Bits8Nat) → BOOL
  @isDbgWriteProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsDebugWriteProcessorStateIndex(i)=IsDebugReadProcessorStateIndex(i)


  @isRunReadProcStateIndexType IsRunningReadProcessorStateIndex ∈ (Bits8Nat) → BOOL
  @isRunReadProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsRunningReadProcessorStateIndex(i)=IsNormalReadProcessorStateIndex(i)

  @isRunWriteProcStateIndexType IsRunningWriteProcessorStateIndex ∈ (Bits8Nat) → BOOL
  @isRunWriteProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsRunningWriteProcessorStateIndex(i)=IsNormalReadProcessorStateIndex(i)

  @isDbgingReadProcStateIndexType IsDebuggingReadProcessorStateIndex ∈ (Bits8Nat) → BOOL
  @isDbgingReadProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsDebuggingReadProcessorStateIndex(i)=
                               bool(IsNormalReadProcessorStateIndex(i)=TRUE ∨ IsDebugReadProcessorStateIndex(i)=TRUE)

  @isDbgingWriteProcStateIndexType IsDebuggingWriteProcessorStateIndex ∈ (Bits8Nat) → BOOL
  @isDbgingWriteProcStateIndexDef     ∀i·i∈Bits8Nat ⇒ IsDebuggingWriteProcessorStateIndex(i)=
                                bool(IsNormalWriteProcessorStateIndex(i)=TRUE ∨ IsDebugWriteProcessorStateIndex(i)=TRUE)

end
```

14.2.53context XCoreRegArray

```
context XCoreRegArray // Define registers within XCore register array

extends ArithmeticTheorems
        RegArrayE1 BitsToNat

constants KspIndex KepIndex VectIndex CpIndex DpIndex SpIndex LrIndex R11Index EdIndex EtIndex SrIndex
          SedIndex SpcIndex SsrIndex
axioms

  @regArraySize  RegArraySize = 25 // Operand and fixed registers

  @r11IndexDef  R11Index = 11 // Fixed by spec
  @cpIndexDef   CpIndex = 12 // Fixed by spec
  @dpIndexDef   DpIndex = 13 // Fixed by spec
  @spIndexDef   SpIndex = 14 // Fixed by spec
  @lrIndexDef   LrIndex = 15 // Fixed by spec
  @kspIndexDef  KspIndex = 16
  @kepIndexDef  KepIndex = 17
  @vectIndexDef VectIndex = 18
  @edIndexDef   EdIndex = 19
  @etIndexDef   EtIndex = 20
  @srIndexDef   SrIndex = 21
  @sedIndexDef  SedIndex = 22
  @spcIndexDef  SpcIndex = 23
  @ssrIndexDef  SsrIndex = 24

  theorem @kspIndexType  KspIndex ∈ RegArrayDom
  theorem @kepIndexType  KepIndex ∈ RegArrayDom
  theorem @vectIndexType VectIndex ∈ RegArrayDom
  theorem @cpIndexType   CpIndex ∈ RegArrayDom
  theorem @dpIndexType   DpIndex ∈ RegArrayDom
  theorem @spIndexType   SpIndex ∈ RegArrayDom
  theorem @lrIndexType   LrIndex ∈ RegArrayDom
  theorem @r11IndexType  R11Index ∈ RegArrayDom
  theorem @edIndexType   EdIndex ∈ RegArrayDom
  theorem @etIndexType   EtIndex ∈ RegArrayDom
  theorem @srIndexType   SrIndex ∈ RegArrayDom
  theorem @sedIndexType  SedIndex ∈ RegArrayDom
  theorem @spcIndexType  SpcIndex ∈ RegArrayDom
  theorem @ssrIndexType  SsrIndex ∈ RegArrayDom
```

```
  theorem @regArrayDom RegArrayDom = 0‥24
  theorem @0_11inRegArrayDom 0‥11 ⊆  RegArrayDom
  theorem @0_15inRegArrayDom 0‥15 ⊆  RegArrayDom
  theorem @regDomInBits32Nat RegArrayDom⊆Bits32Nat

end
```

14.2.54context XCoreResourceClassifications

```
context XCoreResourceClassifications

extends XCoreTimers XCoreChannelEnds XCoreSynchronisers XCoreClocks XCoreLocks XCorePorts XCoreProcessorState XCoreThreads

constants
          IsGetrResourceType  IsFreerResourceIdent
          IsSetevResourceIdent IsSetvResourceIdent
          IsSetdResourceIdent  IsGetdResourceIdent
          IsSetcResourceIdent
          IsSetclkResourceIdent
          IsSetrdyResourceIdent
          IsInResourceIdent    IsOutResourceIdent
          IsEventResourceIdent

axioms

  @isGetrResourceType     IsGetrResourceType ∈ Bits8Nat → BOOL
  @isGetrResourceTypeDef    ∀t·t∈Bits8Nat ⇒ IsGetrResourceType(t) =
                                  bool((t=ResourceTimer)         ∨  (t=ResourceChannelEnd) ∨
                                       (t=ResourceSynchroniser) ∨  (t=ResourceThread)     ∨
                                       (t=ResourceLock))

  @isFreerResourceIdent     IsFreerResourceIdent ∈ Bits → BOOL
  @isFreerResourceIdentDef    ∀i·i∈Bits ⇒ IsFreerResourceIdent(i) =
                                  bool(IsTimerIdent(i)=TRUE         ∨  IsChannelEndIdent(i)=TRUE ∨
                                       IsSynchroniserIdent(i)=TRUE ∨  IsThreadIdent(i)=TRUE     ∨
                                       IsLockIdent(i)=TRUE)

  @isSetevResourceIdent     IsSetevResourceIdent ∈ Bits → BOOL
  @isSetevResourceIdentDef    ∀i·i∈Bits ⇒ IsSetevResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE ∨ IsTimerIdent(i)=TRUE ∨  IsChannelEndIdent(i)=TRUE)

  @isSetvResourceIdent      IsSetvResourceIdent ∈ Bits → BOOL
  @isSetvResourceIdentDef     ∀i·i∈Bits ⇒ IsSetvResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE ∨  IsTimerIdent(i)=TRUE ∨  IsChannelEndIdent(i)=TRUE)

  @isSetdResourceIdent      IsSetdResourceIdent ∈ Bits → BOOL
  @isSetdResourceIdentDef      ∀i·i∈Bits ⇒ IsSetdResourceIdent(i) =
                                  bool(IsTimerIdent(i)=TRUE         ∨  IsChannelEndIdent(i)=TRUE ∨
                                       IsSynchroniserIdent(i)=TRUE ∨  IsClockIdent(i)=TRUE)
```

```
@isSetcResourceIdent      IsSetcResourceIdent ∈ Bits → BOOL
@isSetcResourceIdentDef       ∀i·i∈Bits ⇒ IsSetcResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE ∨  IsTimerIdent(i)=TRUE ∨
                                       IsChannelEndIdent(i)=TRUE ∨ IsClockIdent(i)=TRUE)

@isSetclkResourceIdent     IsSetclkResourceIdent ∈ Bits → BOOL
@isSetclkResourceIdentDef      ∀i·i∈Bits ⇒ IsSetclkResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE ∨  IsClockIdent(i)=TRUE)

@isSetrdyResourceIdent     IsSetrdyResourceIdent ∈ Bits → BOOL
@isSetrdyResourceIdentDef      ∀i·i∈Bits ⇒ IsSetrdyResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE ∨  IsClockIdent(i)=TRUE)

@isGetdResourceIdent      IsGetdResourceIdent ∈ Bits → BOOL
@isGetdResourceIdentDef       ∀i·i∈Bits ⇒ IsGetdResourceIdent(i) = IsSetdResourceIdent(i)

@isInResourceIdent        IsInResourceIdent ∈ Bits → BOOL
@isInResourceIdentDef          ∀i·i∈Bits ⇒ IsInResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE       ∨  IsTimerIdent(i)=TRUE ∨
                                       IsChannelEndIdent(i)=TRUE ∨  IsLockIdent(i)=TRUE)

@isOutResourceIdent       IsOutResourceIdent ∈ Bits → BOOL
@isOutResourceIdentDef         ∀i·i∈Bits ⇒ IsOutResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE ∨ IsChannelEndIdent(i)=TRUE ∨  IsLockIdent(i)=TRUE)

@isEventResourceIdent     IsEventResourceIdent ∈ Bits → BOOL
@isEventResourceIdentDef      ∀i·i∈Bits ⇒ IsEventResourceIdent(i) =
                                  bool(IsPortIdent(i)=TRUE  ∨ IsTimerIdent(i)=TRUE ∨ IsChannelEndIdent(i)=TRUE)

end
```

14.2.55context XCoreResources

```
context XCoreResources

extends Status BitsExtraction BitsConcatenation

constants ResourceId NoResourceId GetResourceId GetResourceIndex GetResourceType
          ResourcePort ResourceTimer ResourceChannelEnd ResourceSynchroniser
          ResourceThread ResourceLock ResourceClock ResourceProcState ResourceConMessages

          CtrlInUseOff CtrlInUseOn
          CtrlCondNone CtrlCondFull CtrlCondAfter CtrlCondEq CtrlCondNeq CtrlCondGreater CtrlCondLess
          CtrlIeModeEvent CtrlIeModeInterrupt
          CtrlDriveDrive CtrlDrivePullDown CtrlDrivePullUp
          CtrlDriveRunStopR CtrlDriveRunStartR CtrlDriveRunClrBuf
          CtrlRunStopR CtrlRunStartR CtrlRunClrBuf
          CtrlMsMaster CtrlMsSlave
          CtrlBufNoBuffers CtrlBufBuffers
          CtrlRdyNoReady CtrlRdyStrobed CtrlRdyHandshake
          CtrlSdelayNoSdelay CtrlSdelaySdelay
          CtrlPortDataport CtrlPortClockport CtrlPortReadyport
          CtrlInvNoInvert CtrlInvInvert

axioms

  @resourceId   ResourceId = Bits
  @noResourceId NoResourceId = 0

  @getResourceId GetResourceId ∈ Bits8Nat × Bits8Nat → ResourceId
  @getResourceIdDef ∀i,t·i∈Bits8Nat ∧ t∈Bits8Nat ⇒ GetResourceId(i↦t) =
FourBits8ToBits32(NatToBits8(t)↦NatToBits8(i)↦NatToBits8(0)↦NatToBits8(0))

  @getResourceIndex GetResourceIndex ∈ ResourceId → Bits8Nat
  @getResourceIndexDef ∀i·i∈Bits ⇒ GetResourceIndex(i) = Bits8ToNat(ExtractBits8_1(i))

  @getResourceType GetResourceType ∈ ResourceId → Bits8Nat
  @getResourceTypeDef  ∀i·i∈Bits ⇒ GetResourceType(i) = Bits8ToNat(ExtractBits8_0(i))

  @resourcePort         ResourcePort         = 0
  @resourceTimer        ResourceTimer        = 1
  @resourceChannelEnd   ResourceChannelEnd   = 2
  @resourceSynchroniser ResourceSynchroniser = 3
  @resourceThread       ResourceThread       = 4
  @resourceLock         ResourceLock         = 5
  @resourceClock        ResourceClock        = 6
```

```
  @resourceProcState    ResourceProcState    = 11
  @resourceConMessages  ResourceConMessages  = 12

  @ctrlInUseOff         CtrlInUseOff         =     0 // 0x0000
  @ctrlInUseOn          CtrlInUseOn          =     8 // 0x0008
  @ctrlCondNone         CtrlCondNone         =     1 // 0x0001
  @ctrlCondFull         CtrlCondFull         =     1 // 0x0001
  @ctrlCondAfter        CtrlCondAfter        =     9 // 0x0009
  @ctrlCondEq           CtrlCondEq           =    17 // 0x0011
  @ctrlCondNeq          CtrlCondNeq          =    25 // 0x0019
  @ctrlCondGreater      CtrlCondGreater      =    33 // 0x0021
  @ctrlCondLess         CtrlCondLess         =    41 // 0x0029
  @ctrlIeModeEvent      CtrlIeModeEvent      =     2 // 0x0002
  @ctrlIeModeInterrupt  CtrlIeModeInterrupt  =    10 // 0x000A
  @ctrlDriveDrive       CtrlDriveDrive       =     3 // 0x0003
  @ctrlDrivePullDown    CtrlDrivePullDown    =    11 // 0x000B
  @ctrlDrivePullUp      CtrlDrivePullUp      =    19 // 0x0013
  @ctrlDriveRunStopR    CtrlDriveRunStopR    =     7 // 0x0007
  @ctrlDriveRunStartR   CtrlDriveRunStartR   =    15 // 0x000F
  @ctrlDriveRunClrBuf   CtrlDriveRunClrBuf   =    23 // 0x0017
  @ctrlRunStopR         CtrlRunStopR         =     7 // 0x0007
  @ctrlRunStartR        CtrlRunStartR        =    15 // 0x000F
  @ctrlRunClrBuf        CtrlRunClrBuf        =    23 // 0x0017
  @ctrlMsMaster         CtrlMsMaster         =  4103 // 0x1007
  @ctrlMsSlave          CtrlMsSlave          =  4111 // 0x100F
  @ctrlBufNoBuffers     CtrlBufNoBuffers     =  8199 // 0x2007
  @ctrlBufBuffers       CtrlBufBuffers       =  8207 // 0x200F
  @ctrlRdyNoReady       CtrlRdyNoReady       = 12295 // 0x3007
  @ctrlRdyStrobed       CtrlRdyStrobed       = 12303 // 0x300F
  @ctrlRdyHandshake     CtrlRdyHandshake     = 12311 // 0x3017
  @ctrlSdelayNoSdelay   CtrlSdelayNoSdelay   = 16391 // 0x4007
  @ctrlSdelaySdelay     CtrlSdelaySdelay     = 16399 // 0x400F
  @ctrlPortDataport     CtrlPortDataport     = 20487 // 0x5007
  @ctrlPortClockport    CtrlPortClockport    = 20495 // 0x500F
  @ctrlPortReadyport    CtrlPortReadyport    = 20503 // 0x5017
  @ctrlInvNoInvert      CtrlInvNoInvert      = 24583 // 0x6007
  @ctrlInvInvert        CtrlInvInvert        = 24591 // 0x600F

end
```

14.2.56context XCoreStatusCodes

```
context XCoreStatusCodes

extends BitsExtraction XCoreRegArray

constants EtLinkError EtIllegalPc EtIllegalInstruction EtIllegalResource EtLoadStore EtIllegalPs EtArithmetic
          EtResourceDep
          EtEcall EtKcall
          DcallCause
          EebleStatus IebleStatus InenbStatus InintStatus InkStatus SinkStatus WaitingStatus FastStatus

axioms

  @etLinkError   EtLinkError           = 1
  @etIllegalPc   EtIllegalPc           = 2
  @etIllegalInst EtIllegalInstruction  = 3
  @etIllegalRes  EtIllegalResource     = 4
  @etLoadStore   EtLoadStore           = 5
  @etIllegalPs   EtIllegalPs           = 6
  @EtArithmetic  EtArithmetic          = 7
  @etEcall       EtEcall               = 8
  @EtResDep      EtResourceDep         = 9
  @etKcall       EtKcall               = 15
  @dcallCause    DcallCause            = 0

  @eebleStatusType EebleStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @eebleStatus            ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                   EebleStatus(r↦t)=ExtractBits1_0((r(t))(SrIndex))

  @iebleStatusType IebleStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @iebleStatus            ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                   IebleStatus(r↦t)=ExtractBits1_1((r(t))(SrIndex))

  @inenbStatusType InenbStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @inenbStatus            ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                   InenbStatus(r↦t)=ExtractBits1_2((r(t))(SrIndex))

  @inintStatusType InintStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @inintStatus           ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                   InintStatus(r↦t)=ExtractBits1_3((r(t))(SrIndex))

  @inkStatusType   InkStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @inkStatus               ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
```

```
                    InkStatus(r↦t)=ExtractBits1_4((r(t))(SrIndex))

  @sinkStatusType SinkStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @sinkStatus              ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                    SinkStatus(r↦t)=ExtractBits1_5((r(t))(SrIndex))

  @waitingStatusType WaitingStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @waitingStatus            ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                     WaitingStatus(r↦t)=ExtractBits1_6((r(t))(SrIndex))

  @fastStatusType FastStatus ∈ (ThreadIndexDom → RegArray) × ThreadIndexDom → Bits
  @fastStatus              ∀r,t · r∈(ThreadIndexDom → RegArray) ∧ t∈ThreadIndexDom  ⇒
                    FastStatus(r↦t)=ExtractBits1_7((r(t))(SrIndex))

end
```

14.2.57context XCoreSynchronisers

```
context XCoreSynchronisers

extends XCoreResources ArraySearch

constants SynchroniserStatus
          MaxSynchronisers MaxSynchroniserIndex SynchroniserIndexDom
          GetSynchroniserIdent IsSynchroniserIdent IsSynchroniserInUse
          FindFreeSynchroniser CountFreeSynchronisers

axioms

  @synchroniserStatusDef SynchroniserStatus = {FREE,RUNNING,MPAUSING,SPAUSING,CLOSING}
  theorem @synchroniserStatusType SynchroniserStatus ⊆ℕ

  @maxSynchronisers MaxSynchronisers ∈ ℕ1 // Must be at least one
  @maxSynchroniserIndex MaxSynchroniserIndex = MaxSynchronisers−1
  @synchroniserIndexDom SynchroniserIndexDom = 0‥(MaxSynchronisers−1)

  @maxSynchronisersSeven MaxSynchronisers = 7

  @getSynchroniserIdType   GetSynchroniserIdent ∈ Bits8Nat → Bits
  @getSynchroniserIdDef           ∀i·i∈Bits8Nat ⇒ GetSynchroniserIdent(i)=GetResourceId(i↦ResourceSynchroniser)

  @isSynchroniserIdentType IsSynchroniserIdent ∈ Bits → BOOL
  @isSynchroniserIdentDef     ∀i·i∈Bits ⇒ IsSynchroniserIdent(i)=bool(GetResourceType(i)=ResourceSynchroniser ∧
GetResourceIndex(i)∈SynchroniserIndexDom)

  @isSynchroniserInUseType IsSynchroniserInUse ∈ (Bits × (SynchroniserIndexDom → SynchroniserStatus)) → BOOL
  @isSynchroniserInUseDef     ∀i,s·i∈Bits ∧ s∈SynchroniserIndexDom → SynchroniserStatus  ⇒ IsSynchroniserInUse(i↦s)=
                           bool(GetResourceIndex(i)∈SynchroniserIndexDom ∧ s(GetResourceIndex(i))≠FREE)

  @findFreeType FindFreeSynchroniser ∈ (SynchroniserIndexDom→SynchroniserStatus) → SynchroniserIndexDom
  @findFreeDef  ∀s·(s∈SynchroniserIndexDom→SynchroniserStatus) ⇒ FindFreeSynchroniser(s)=FindInArray(s↦MaxSynchronisers↦FREE)

  @countFreeType CountFreeSynchronisers ∈ (SynchroniserIndexDom→SynchroniserStatus) → 0‥MaxSynchronisers
  @countFreeDef  ∀s·(s∈SynchroniserIndexDom→SynchroniserStatus) ⇒ CountFreeSynchronisers(s)=CountInArray(s↦MaxSynchronisers↦FREE)

end
```

14.2.58context XCoreThreads

```
context XCoreThreads

extends XCoreResources ThreadArray ArraySearch

constants ThreadStatusArray ThreadCount
          GetThreadIdent
          IsThreadIdent IsThreadSynchronous IsThreadInUse IsThreadRunning IsThreadSpausing
          FindFreeThread CountFreeThreads CountRunningThreads CountSpausedThreads

axioms

  @maxThreads MaxThreads = 8

  @threadCount ThreadCount = 0‥MaxThreads

  @threadStatusArray ThreadStatusArray = ThreadIndexDom → ThreadStatus

  @getThreadIdType GetThreadIdent ∈ Bits8Nat → Bits
  @getThreadIdDef ∀i·i∈Bits8Nat ⇒ GetThreadIdent(i)=GetResourceId(i↦ResourceThread)

  @isThreadIdentType IsThreadIdent ∈ Bits → BOOL
  @isThreadIdentDef ∀i·i∈Bits ⇒ IsThreadIdent(i)=bool(GetResourceType(i)=ResourceThread ∧ GetResourceIndex(i)∈ThreadIndexDom)

  @isThreadSynchronousType IsThreadSynchronous ∈ (Bits × (ThreadIndexDom → BOOL)) → BOOL
  @isThreadSynchronousDef ∀i,s·i∈Bits ∧ s∈ThreadIndexDom → BOOL  ⇒ IsThreadSynchronous(i↦s)=
                          bool(GetResourceIndex(i)∈ThreadIndexDom ∧ s(GetResourceIndex(i))=TRUE)

  @isThreadInUseType IsThreadInUse ∈ (Bits × (ThreadIndexDom → ThreadStatus)) → BOOL
  @isThreadInUseDef ∀i,s·i∈Bits ∧ s∈ThreadIndexDom → ThreadStatus  ⇒ IsThreadInUse(i↦s)=
                    bool(GetResourceIndex(i)∈ThreadIndexDom ∧ s(GetResourceIndex(i))≠FREE)

  @isThreadRunningType IsThreadRunning ∈ (Bits × (ThreadIndexDom → ThreadStatus)) → BOOL
  @isThreadRunningDef ∀i,s·i∈Bits ∧ s∈ThreadIndexDom → ThreadStatus  ⇒ IsThreadRunning(i↦s)=
                      bool(GetResourceIndex(i)∈ThreadIndexDom ∧ s(GetResourceIndex(i))=RUNNING)

  @isThreadSpausingType IsThreadSpausing ∈ (Bits × (ThreadIndexDom → ThreadStatus)) → BOOL
  @isThreadSpausingDef ∀i,s·i∈Bits ∧ s∈ThreadIndexDom → ThreadStatus  ⇒ IsThreadSpausing(i↦s)=
                       bool(GetResourceIndex(i)∈ThreadIndexDom ∧ s(GetResourceIndex(i))=SPAUSING)

  @findFreeStatusType FindFreeThread ∈ (ThreadIndexDom→ThreadStatus) → ThreadIndexDom
  @findFreeStatusDef ∀s·(s∈ThreadIndexDom→ThreadStatus) ⇒ FindFreeThread(s)=FindInArray(s↦MaxThreads↦FREE)
```

```
  @countFreeType CountFreeThreads ∈ (ThreadIndexDom→ThreadStatus) → 0‥MaxThreads
  @countFreeDef ∀t·(t∈ThreadIndexDom→ThreadStatus) ⇒ CountFreeThreads(t)=CountInArray(t↦MaxThreads↦FREE)

  @countRunningType CountRunningThreads ∈ (ThreadIndexDom→ThreadStatus) → 0‥MaxThreads
  @countRunningDef ∀t·(t∈ThreadIndexDom→ThreadStatus) ⇒ CountRunningThreads(t)=CountInArray(t↦MaxThreads↦RUNNING)

  @countSpausedType CountSpausedThreads ∈ (ThreadIndexDom→ThreadStatus) → 0‥MaxThreads
  @countSpausedDef ∀t·(t∈ThreadIndexDom→ThreadStatus) ⇒ CountSpausedThreads(t)=CountInArray(t↦MaxThreads↦SPAUSING)

end
```

14.2.59context XCoreTimers

```
context XCoreTimers

extends XCoreResources ArraySearch

constants TimerStatus
          MaxTimers MaxTimerIndex TimerIndexDom
          GetTimerIdent
          IsTimerIdent
          IsTimerInUse
          FindFreeTimer CountFreeTimers

axioms

  @timerStatusDef TimerStatus = {FREE,RUNNING}
  theorem @timerStatusType  TimerStatus ⊆ℕ

  @maxTimers MaxTimers = 4
  @maxTimerIndex MaxTimerIndex = MaxTimers−1
  @synchroniserIndexDom TimerIndexDom = 0‥(MaxTimers−1)

  @getTimerIdent GetTimerIdent ∈ TimerIndexDom → ResourceId
  @getTimerIdentDef ∀i·i∈TimerIndexDom ⇒ GetTimerIdent(i) = GetResourceId(i↦ResourceTimer)

  @isTimerIdentType IsTimerIdent ∈ Bits → BOOL
  @isTimerIdentDef  ∀i·i∈Bits ⇒ IsTimerIdent(i)=
                    bool(GetResourceType(i)=ResourceTimer ∧ GetResourceIndex(i)∈TimerIndexDom)

  @isTimerInUseType IsTimerInUse ∈ (Bits × (TimerIndexDom → TimerStatus)) → BOOL
  @isTimerInUseDef  ∀i,s·i∈Bits ∧ s∈TimerIndexDom → TimerStatus  ⇒ IsTimerInUse(i↦s)=
                    bool(GetResourceIndex(i)∈TimerIndexDom ∧ s(GetResourceIndex(i))≠FREE)

  @findFreeStatusType FindFreeTimer ∈ (TimerIndexDom→TimerStatus) → TimerIndexDom
  @findFreeStatusDef  ∀s·(s∈TimerIndexDom→TimerStatus) ⇒ FindFreeTimer(s)=FindInArray(s↦MaxTimers↦FREE)

  @countFreeType CountFreeTimers ∈ (TimerIndexDom→TimerStatus) → 0‥MaxTimers
  @countFreeDef  ∀t·(t∈TimerIndexDom→TimerStatus) ⇒ CountFreeTimers(t)=CountInArray(t↦MaxTimers↦FREE)

end
```