2011 17th IEEE International Symposium on Asynchronous Circuits and Systems

# Variation Tolerant AFPGA Architecture

Hock Soon Low, Delong Shang, Fei Xia, Alex Yakovlev
MSD Group, School of EECE
Newcastle University
Newcastle upon Tyne, UK
{h.s.low1,delong.shang,fei.xia,alex.yakovlev}@ncl.ac.uk

**Abstract — This paper describes the realization of an interconnect Delay Insensitive (DI) FPGA architecture with distributed asynchronous control. This architecture maintains the basic block structure of traditional FPGAs allowing the potential use of existing FPGA design tools in block design. This asynchronous FPGA architecture is mainly aimed at tolerating the unpredictable delay variations caused by process and environment variations in current and future VLSI technology nodes and also targets power supply variations, including modes such as dynamic voltage scaling and variable Vdd, such as in applications featuring energy harvesting. This is achieved by making the longer inter-block interconnects DI, keeping the computational logic single-rail, and removing global clocks.**

***Keywords-FPGA; Variation; Asynchronous; Architecture.***

## I. INTRODUCTION

Feature size shrinking in CMOS transistors leads to high density in VLSI chips and also causes variations to become a key factor affecting system behaviors. Two sources of variations, environmental factors such as changes in temperature due to changes in environment temperature or chip activity and physical factors during the manufacturing process [1], can affect circuit performance and functionality.

Power supply variation can be considered an environmental factor. Energy harvesting devices, for example, tend to provide variable levels of power and voltage at run time.

CMOS process scaling continues to increase the difficulty and cost of the fabrication process to achieve uniformity of die production. This limitation will result in random and spatially varying deviations from intended design parameters, a phenomenon generally known as process parametric variation. The International Technology Roadmap for Semiconductors (ITRS) states that parametric variation is becoming a greater concern. This is because variation and margins for logic and interconnects are not scaling at the same rate [2].

The ITRS also predicts that chip designs are more likely to have multiple modules running different clocks (GALS-orientated). Up to 40% of the chip will be using asynchronous techniques to solve clock distribution problem by 2020 [2]. Asynchronous circuits, such as Delay Insensitive (DI) and Speed Independent (SI or QDI) circuits can in general maintain correctness in the face of uncertain delays caused by parametric or environmental variations and are more likely to function correctly under variable Vdd.

Field-Programmable Gate Array (FPGA) is one of the main VLSI methods in current digital system designs. Variation tolerance is an important issue for future FPGA technologies.

The work was supported by EPSRC DTA studentship and grant Holistic (EP/G066728) at the School of EECE, Newcastle University, UK.

This paper focuses on dealing with the effects of process and environment variations on the behaviors of high-performance modern FPGAs. One of the commonest manifestations of process variation is in the delays or latencies of circuits. Here we investigate techniques with which systems can be made tolerant to such effects.

In modern FPGAs, interconnects occupy up to 80% of the chip area [3]. Although the latest nanometer technology trends to increase the proportions of the logic array, interconnects still occupy a large part of the entire chip. They play an important role in linking together all the functional blocks including configurable logic blocks, memory blocks and multiplexers. Long interconnect wires can easily dominate delay if not managed properly. Similarly, with the increasing density of FPGAs in the latest technology, it is getting more challenging to distribute global clocks across wide areas evenly. Extra effort is required to deal with such issues as clock skew.

Asynchronous design techniques may avoid the need for a global clock to control circuit activities; instead activities may be locally controlled with for example handshaking protocols [4]. Since asynchronous circuits potentially offer global clock absence and low power, applying asynchronous design techniques to FPGAs is attractive. Asynchronous circuits can be said to be self-timed, Speed-Independent (SI) or Delay-Insensitive (DI) depending on the delay assumptions that are made [4]. SI circuits consider only the gate delay and neglect the wire delay when analyzing circuit correctness. The most robust class is DI where circuits will operate correctly without any assumption on delays either in gates or interconnect wires. SI and DI circuits are particularly useful to cope with process and environmental variations affecting latency behavior. However, asynchronous circuits are more difficult to design and test compared to synchronous ones because of the wide variety of possible signaling protocols and a broad spectrum of the degree of delay insensitivity from bounded delay to full DI. Partly because of this, asynchronous design suffers from a lack of automatic design tools especially those combining all possible techniques in a single suite. These issues have been particularly impeding the progress of asynchronous techniques in FPGA, because the latter is intrinsically less customizable.

### *A. Contributions and Paper Organization*

An FPGA architecture which achieves DI interconnectivity between logic clusters has been developed. The long interconnects are made DI with a new distributed control, based on David Cells, replacing clock trees. Clusters can be made close to SI because local wire lengths can be easily managed. Programmable delay elements are also introduced in

1522-8681/11 $26.00 © 2011 IEEE
DOI 10.1109/ASYNC.2011.17

wrapper circuits to allow local logic cell delay matching after fabrication, and potentially support low Vdd operations. The architecture preserves the block structure of current technology FPGA with its single-rail combinational logic arrangement, making the system design flow simpler and more accessible as well as keeping size and power consumption under control. In effect the new architecture seeks to achieve *DI in the large* for long inter-cluster wires and approximate *SI in the small* within clusters. The result is a balance between the desire of using asynchrony for tolerating the effects of variations and retaining the major part of current design flow.

The rest of the paper is organized as follows. Section II describes existing attempts at introducing asynchrony into FPGAs. Section III describes our asynchronous FPGA architecture and its essential components in detail. Section 4 describes the proposed system design flow for this type of FPGA. Section V contains further investigations through example system designs. Section 6 concludes with discussions and future work outlook.

## II. Related Work

Various statistical design techniques have been proposed to deal with the variation problem in CMOS design [14, 15]. However, these techniques, proposed for custom VLSI and ASICs, cannot be directly applied to FPGAs in which the circuit mapping varies depending on the user design after fabrication. A few papers suggested chip-wise placement based on traditional FPGA architecture [16, 17] but this requires extensive efforts to determine the variation map of each chip.

Different styles of asynchronous FPGA architectures were presented since 1992 [5, 18-24] motivated mainly by either low power and/or high speed performance. From the point of view of implementation techniques, these architectures can be classified into three types.

Type-1 (e.g. [18] and [19]) rely on significant elements of timing assumption to guarantee the correctness of the asynchronous logic. In general, the global worst case timing assumption is replaced with local worst case timing assumptions. This method mixes the delays for both FPGA intra-block and inter-block connections. It is hard to match delays for FPGA inter-block interconnects as the wire lengths can be non-deterministic at design time. Process and environmental variations in general will make the delay matching problem worse.

Type-2 (e.g. [20] and [21]) focuses on high performance. They are also somewhat tolerant to operational variations (delay, voltage, temperature) through SI/DI with none or minimal reliance on timing assumptions. On the other hand, they modify the entire FPGA fabric and replace the fundamental basic logic block of current FPGA technology. This makes the system design process less accessible by significantly reducing the usefulness of existing design tools. Also, some of these architectures are fine-grained and C-element intensive implying high area and power overhead.

Type-3, proposed in [27] and [5], is to some degree a compromise between types (1) and (2). These proposed architectures keep the traditional logic block structure and introduces the concept of an asynchronous wrapper around it. This type of design has two potential advantages: 1) maintaining the existing FPGA block structure to provide the possibility of using existing commercial EDA tools for in-block design and 2) potentially [5] making global interconnects DI for latency variation tolerance across long wires where it is most needed. Such architectures have yet to be fully explored.

This paper presents the implementation of a type-3 architecture [5] with detailed circuit level designs of all essential elements and proposes a standard design flow for this architecture.

## III. Design and Implementation

Fully SI, or better still, fully DI systems are tolerant to latency variations caused by process and environmental variations. However, these types of systems in general require a completely different design process for which few mature EDA toolkits exist. This problem is even more acute for FPGAs than for general ASIC. As a result, the design process for fully SI/DI FPGAs is far from straightforward and usually presents a steep learning curve to the designer. This difficulty for system design on an industrial scale weakens the main attractiveness of SI/DI FPGAs in prototyping and time to market.

In addition, SI/DI circuits tend to be based on relatively complex coding methods, such as dual-rail or in general m-of-n, implying large area overheads. Asynchronous circuits are supposedly low power as they can be made to work only when necessary and do not require clock trees. However, complex coding may result in an increase of switching in, for example, combinational logic leading to more dynamic power consumption, potentially negating savings from removing clock signals. The larger circuit size may also lead to greater leakage power. The design goal on power in this work is thus to keep the power consumption under control at a comparable level to synchronous designs.

### A. Architecture

Figure 1 shows the schematic view of the basic island-style FPGA architecture. The routing channels consist of wire segments, switch boxes and connection boxes surrounding the logic cluster. Ignoring the clock circuits, the conventional FPGA structure has two types of circuits, 1) small logic islands (logic clusters), and 2) large wires (interconnect channel). If the entire structure is not implemented fully in SI/DI circuits, the large wires need more efforts on timing than the small clusters. Process variations will also cause comparatively more significant latency problems in the large wires.

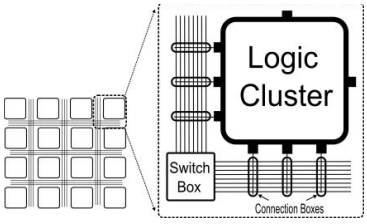


**Figure 1: Basic Island Style FPGA Architecture.**

In other words, tolerance for latency variations is most important for long interconnects. Within local areas of limited size, latency variations tend to be easier to manage. The

approach of our chosen architecture exploits this possible tradeoff by using DI inter-block connections and leaving the basic conventional FPGA block/cluster relatively unchanged. The overall system will in general consist of a set of blocks on a DI interconnect fabric. By completely removing the clock tree and replacing it with a distributed David Cell based control, this method makes each cluster functionally behave as a self-timed block to its environment.

**Table 1: Architecture Overview**

| Architecture Overview | | |
|---|---|---|
| ***Parameter*** | ***value*** | ***Description*** |
| Architecture | Island style | [6] |
| LUT Size (K) | 4 * Inputs | [7] |
| Cluster Size (N) | 4 * PLE | [7] |
| Cluster input Channels (i) | 16 | [7] |
| Channel Type | Dual- rail/Channel | 1-of-2 coding |
| Switch Box | Universal | - |
| Connection Box | Normal | - |
| Process Technology | UMC-90nm CMOS | - |
| Handshake Protocol | 4-Phase Dual-Rail | [4] |

Table 1 shows the structural choices made for our asynchronous FPGA architecture implementation. The granularity of the FPGA architectures is usually based on three main logic cluster parameters. These are the lookup table (LUT) size (*K*), cluster size (*N*) or the number of LUTs in a cluster, and the number of inputs per cluster (*I*). The granularity will affect the area, speed and power. In general, increasing the sizes of *K* and *N* will increase the functionality and performance but also increase the area exponentially. Cluster input size, *I*, should be kept as small as possible, however if *I* is too small, many logic elements in the cluster may be unusable [8, 9]. The traditional FPGA architecture uses four LUTs per cluster and four inputs for each LUT. Here we choose these as default values.

The basic building block in an FPGA comprises a lookup table (LUT), a register (DFF) and a multiplexer (MUX). It is normally called logic cell (LC) in Xilinx the equivalent from Altera is called logic element (LE).

### *B. Programmable Logic Element (PLE)*

In this paper, we call our basic building block *programmable logic element* (PLE). The architecture does not include a global clock system. At the level of the basic building block PLE, the functionality of clock signals in conventional FPGAs is now performed by a newly designed wrapper set around a conventional LE. This is shown in Figure 2.

The wrapper consists of the following circuits:

- Programmable completion detection, PCD
- Trigger Selection switch, SW
- Programmable delay, PD
- Single- to dual-rail converter, CONV
- Completion detection, CD

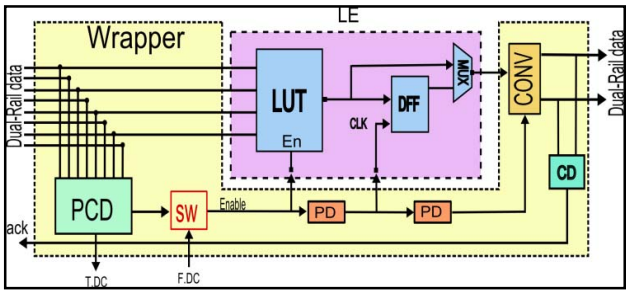


**Figure 2: Programmable Logic Element**

PCD is used to detect that all input data is ready. There are two possible uses for this input data when it is ready: 1) the data is used for a concurrent operation with other PLEs which may or may not be in the same cluster; 2) the data is used for an operation involving only the local PLE. For the former case, the ready signal goes to the David Cell (DC) based distributed control to synchronize with the input data for other PLEs (via T.DC – "to DC") and when the synchronization is complete, SW will be enabled by the F.DC ("from DC") signal from the DC circuit. For the latter case, the ready signal goes directly to SW to enable the local LUT to start data processing.

SW is implemented with a single bit SRAM cell to register the LUT enable signal, which can be either directly from PCD or from the distributed DC control. The enable signal is used to guarantee that the LUT is activated only when the data is ready for the entire operation. This minimizes the number of switching activities in the combinational logic.

PD is used to set up delays to latch data and to indicate the "ready" of the single-rail data from MUX. This is programmable delay bundling and will be discussed later in more detail. CONV is used to convert the conventional single-rail data in PLE to dual-rail data for data propagation via the DI interconnects. CD is the completion detection indicating this dual-rail output.

The wrapper works as follows: The "all data ready" state is detected by the left-hand side PCD which generates a trigger signal. The trigger signal (after synchronizing with trigger signals from other blocks in an operation group, if needed) will then be fed to LUT to start the computation. PDs are used to control the timing of latching the result of LUT computation and starting the conversion of single rail data to dual-rail data. The right-hand side CD will then generate the "done" signal when the conversion is complete.

For complex concurrent operations requiring more than one PLE, the trigger signals from PCDs can be collected for group control using DC control. SW can be configured to select trigger signal between PCD and DC. This will be discussed later in more detail.

#### *1) PCD Design*

PCD is the completion detection circuit used to detect the readiness of all input data signals and, upon this detection, generate a trigger signal. The input data from the DI

interconnects is in dual-rail, 1-of-2 coding format. This format allows the straightforward detection of valid signal from spacer or empty codewords. A commonly used dual-rail code is

{0,0}=spacer,
{0,1}=0,
{1,0}=1,
{1,1}=not valid.

Because the codeword {1,1} is illegal and does not occur, an OR gate is sufficient to safely indicate a single channel as being "valid" or "empty".

The C-element is one of the fundamental and commonly used components in asynchronous circuits [4], it provides the hysteresis in the empty-to-valid and valid-to-empty transition required for transparent handshaking. It waits for all its inputs to be valid to set the output to high and waits for all its inputs to become empty to set the output to low. For other input combinations, the output does not change. C-elements are thus ideal for collecting the states of multiple channels.

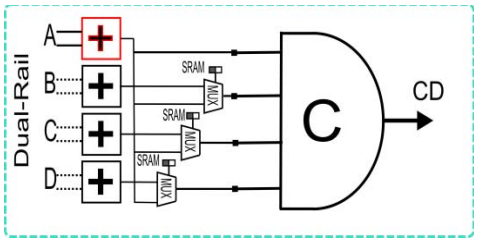


**Figure 3: Programmable CD Circuit.**

In our PLE design, a C-element, four 2-inputs OR gates and three multiplexers are use to form the PCD circuit as shown in Figure 3. The C-element used has four fixed inputs; every input must be able to produce a "valid" signal for the circuit to function. Depending on system design and configuration, some channels may not be used. Flexible channel selection is required for the C-element in such cases. This is solved by using multiplexers to link an unused channel (B, C or D) to Channel A, which is assumed to be always in use if the PLE participates in the computation. Whether a channel among B, C and D is used or not in the application can be set by enabling or disabling the relevant MUX at program time.

*2) SW Box*

The SW trigger switch box is also programmable. The trigger signal can be programmed to be either from the local PCD or from DC control via the F.DC signal. The F.DC signal is passed from the PLE below and to the PLE above in multi-PLE blocks. This is shown in Figure 4.

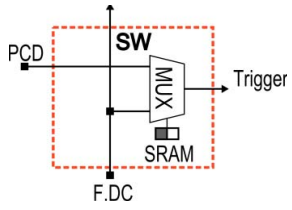


**Figure 4: SW box circuit.**

*3) PD unit*

The conventional single-rail LE needs an equivalent local clock to function properly in the absence of a global clock system. One of the simplest approaches to building an asynchronous system using conventional data path elements from synchronous systems is known as "bundled data". In this, delay elements are used to bundle the data path elements temporally and local control signals performing the equivalent functions of clocks are generated from the outputs of these delays. Correct bundling means that the delay element will have a good matching for the latency of the data path element being bundled. Bundling local delays this way provides more flexibility than the global worst case assumptions needed for a global synchronous clock system, but the temporal control principles are the same. Here we use the delay bundling approach to control the LE within a PLE.

The most basic delay circuit is typically made out of a chain of inverters. Because of variability, either environmental or physical, the latency characteristics of similar LEs in different parts of a chip may be different. A PD element has been designed here to cope with variations. An example with four selectable ranges is shown in Figure 5. A MUX is used to provide the programmable selection of different points in the inverter chain to maintain the correct delay given specific operating conditions.

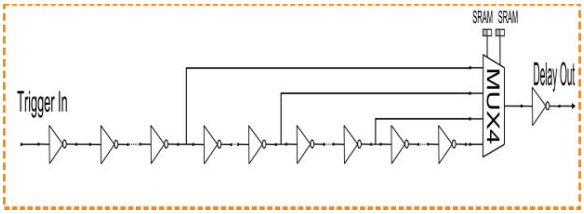


**Figure 5: Programmable Delay circuit.**

This programmability in delay is useful provided information on variations is available at program time. Such information can be about the environment and power supply. It can even be about process variations post fabrication, providing a facility for chip-wise programming or late binding [26] techniques. PDs in different blocks on a chip may be set differently based on information collected from a characterization process.

There is a tradeoff between delay programming precision and PD overhead costs. In order to maintain average and not worst case performance under a wide variation range, higher delay programming precisions may be needed, resulting in larger area overheads in PDs. The dynamic power cost of delay bundling does not change with or without programmability.

*4) Single-rail to Dual-Rail Conversion Circuit*

Converting the single-rail data output from LE to dual-rail format for the DI interconnects is the responsibility of CONV in Figure 2. This simple circuit is shown in Figure 6.

There is no need to perform overt dual-rail to single rail conversion at the input of LE because one of the data wires directly gives the single-rail binary value of valid code words. Once it has been made sure that spacers do not propagate to LE this wire can be directly used as an input.

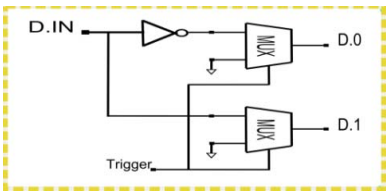


Figure 6: Converter circuit.

*5) Gate Count/Area*

Asynchronous circuits tend to be bigger than synchronous ones. In the case of full SI/DI there are overheads in both the control circuits replacing clock systems and data path circuits because of the complex coding in such systems, e.g. dual-rail. Even bundled data systems tend to have a clock replacement consisting of delay units which are much more substantial than the clock wires they replace.

With this additional investment, many advantages can be obtained including better variation tolerance, avoiding the inevitable problems clock systems face with technology scaling, lower power consumption in low duty-cycle cases, etc.

Table 2: PLE size in terms of Number of Transistors

| BOX | BLOCK | Parts Included | Total |
|---|---|---|---|
| ***Logic Element*** | LUT | SRAM * 16 | 96 |
| | | Mux Tree (K = 4) $\sum_{i=1}^{k} 2^i$ | 30 |
| | | Buffer * 30 | 60 |
| | DFF | | 24 |
| | 2:1 MUX | | 4 |
| ***Total:*** | | | 214 |
| ***Wrapper*** | PCD | SRAM * 3 | 18 |
| | | 2:1MUX * 3 | 12 |
| | | Gates | 60 |
| | SW | 2:1MUX * 1 | 4 |
| | | SRAM * 1 | 6 |
| | (PD)*2 | (Buffer * 10) * 2 | 40 |
| | | (4:1MUX ) * 2 | 20 |
| | | (SRAM * 2)*2 | 24 |
| | CONV | 2:1MUX * 2 | 8 |
| | | Buffer | 2 |
| | CD | OR2 | 6 |
| ***Total:*** | | | 200 |

Table 2 shows the complexity of our PLE. It is worth noting that the wrapper contains a similar number of circuit elements to LE itself. This means that our PLE is almost twice as big in terms of size. However the following subsection shows that power does not increase significantly through power analysis.

*6) Power Comparison*

The proposed PLE is roughly twice the size of the normal PLE. Normally, larger circuits consume more power. Our new PLE uses asynchronous techniques to replace global clock tree. This should reduce power. The combined effect is uncertain. Even though power is not the main motivation for this work, it needs to be studied to see if there are radical changes to power consumption.

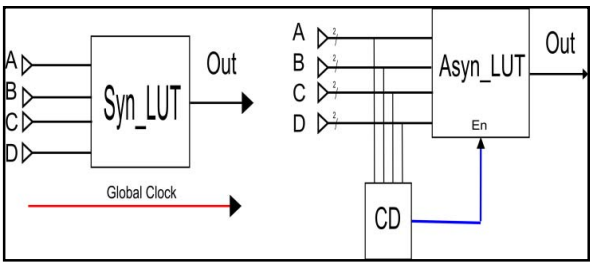


Figure 7: Synchronous LUT and CD LUT.

We use one PLE as an example. As variations are introduced, it is hard to guarantee that the data will come at the same time. Here we assume the worst case where all four data bits come at different moments of time. This has no effect on synchronous FPGA as correct operations are guaranteed by the global clock signals. Only during the clock rising edge, the stable data is required. But this is obtained by spending power on clocks. In our structure, the data computation starts only when all data has come.

The operation power consumption of conventional SRAM based LUT and CD based LUT are investigated in a comparative study using the following setup (Figure 7).

Experiment setup:

- Four signals, namely A, B, C and D, arrive at the input of LUT with different timing.
- Signal A is assumed to arrive last and other signals were changing before it become stable. (16 transitions between "0000" to "1111", changing 1 bit at a time).
- When every signal including 'A' eventually settles, output '1' will be produced.

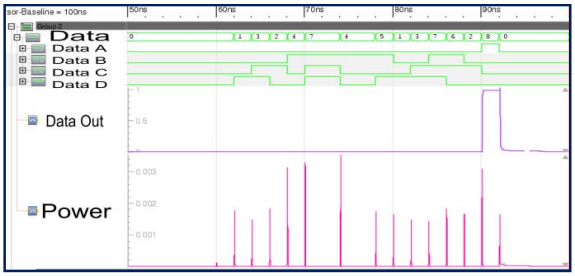


Figure 8: Synchronous LUT operation power.

The simulation result is shown in Figure 8. When there is a transition in an input signal, the power line will spike. In the synchronous circuit, every change of data between valid clock signals changed the state of the LUT address and consumes energy. On the other hand, in the asynchronous design, data will not be read out from the SRAM until it received the enable signal (En) from CD as shown in Figure 9. The spikes of power in Figure 9 are therefore from CD.

This simulation shows that although we have increased the size in the PLE circuit, it consumes roughly the same power and energy as the normal clock based LUT. If taking account of the clock tree circuit and dynamic clock transition power, the proposed asynchronous LUT with CD may produce an

overall better power consumption characteristic. The power figures can be found in Table 3. The simulation is relevant to a high duty-cycle situation where the unit is in operation most of the time with the asynchronous arrival of inputs representing a small proportion of the total time. In low duty-cycle situations when the unit is not active for more significant lengths of time, the asynchronous solution can be more advantageous.

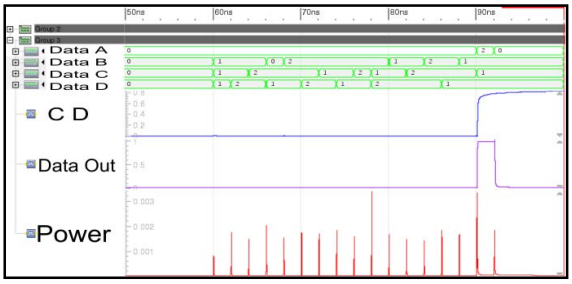


**Figure 9: Completion Detection LUT operation power.**

**Table 3: Power and Energy Comparison between synchronous and Asynchronous PLEs**

| Circuit | Operation Energy | Single operation Power | Operating Voltage |
|---|---|---|---|
| Synchronous LUT | 1.118pJ | 3.102mW | 1.0V |
| Completion Detection LUT | 1.152pJ | 3.355mW | 1.0V |

*7) PLE Behavior and Performance*

This PLE implementation was constructed through the Cadence design process on UMC 90nm CMOS technology. Analog simulations show that the circuit works correctly as designed without logic errors within the Vdd range of 0.45V~1V. The overall latency and energy per operation performance is shown in Figure 10.

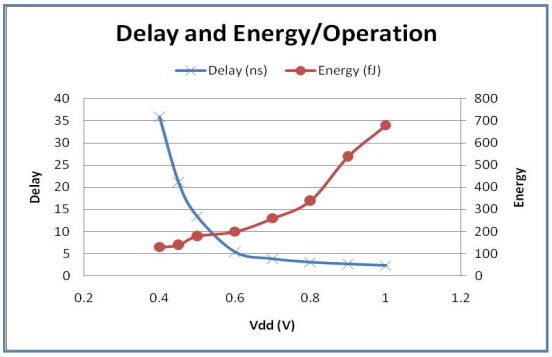


**Figure 10: Energy and Latency vs Vdd.**

The circuit stops functioning correctly when Vdd is further reduced below 0.45V. On closer observation, it was noted that the bundling PD delays become faster than the units (LUT etc.) they are bundled with. This is an interesting phenomenon (noticed earlier in e.g. [25]) where usual delay elements based on inverter chains cannot maintain correct temporal bundling for memory type circuits (e.g. the SRAM cells here in the LUT) when Vdd is lowered towards the subthreshold region. This is because of a mismatch between the rates of slowing down when Vdd is reduced for memory and inverter circuits.

As long as the bundling was still working correctly, however, PLE works fine under different, but constant Vdds, and the energy and latency behaviors are not surprising. It is worth noting that there is not much of an increase in latency (2×) by lowering Vdd to 0.6V but the energy savings can be quite significant.

An investigation was then carried out to see how PLE behaves under continuous varying Vdd over the range in which it is know to work correctly with constant Vdd. In this experiment, a relatively slow sinusoid signal between 1V and 0.45V was set as the Vdd pattern and PLE subjected to that. The LE was configured as an adder to calculate the sum of A, B, and C without carry, with the D data channel unused. All three used data lines are given the value of 1 when valid and they are switched out of spacer as soon as the previous round's completion detection is generated. In other words, in this experiment PLE works in a self-looping environment with the completion of one round providing the ready signal to trigger the next round. The simulation result is shown in Figure 11.

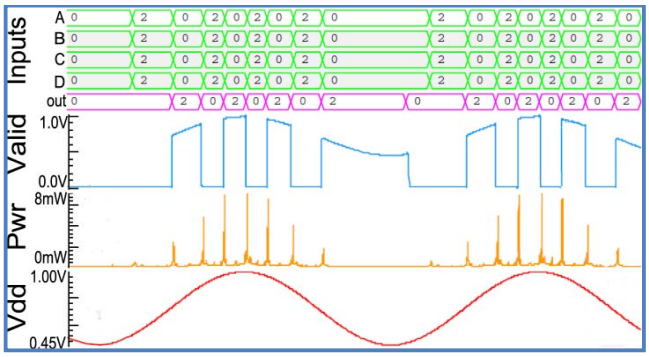


**Figure 11: PLE working under variable Vdd.**

From these results it can be seen that the circuit works under a continuously varying Vdd pattern smoothly without errors. The circuit is slower with lower power consumption when Vdd is reduced and faster with higher power consumption when Vdd is increased. The signal Valid is from the output CD block indicating that the data output lines have valid data values and not spacers. The data value "2" indicates binary value 1 because of the dual-rail coding ({1,0}=1) on the input and output signals. The data computation was correct (without carry, 1+1+1=1).

## C. *Cluster Design*

The *cluster*, consisting of a group of PLEs, is the next level in the hierarchy of this FPGA architecture. The units in the same hierarchical layer in conventional FPGAs is known as the configurable logic block (CLB) in Xilinx terminology and logic array block (LAB) in that of Altera.

Similar to most commercial FPGAs, our cluster (Figure 12) consists of four PLEs with the addition of one David Cell (DC) which forms part of a distributed intra-cluster and inter-cluster control. The general cluster structure is shown in Figure 12.

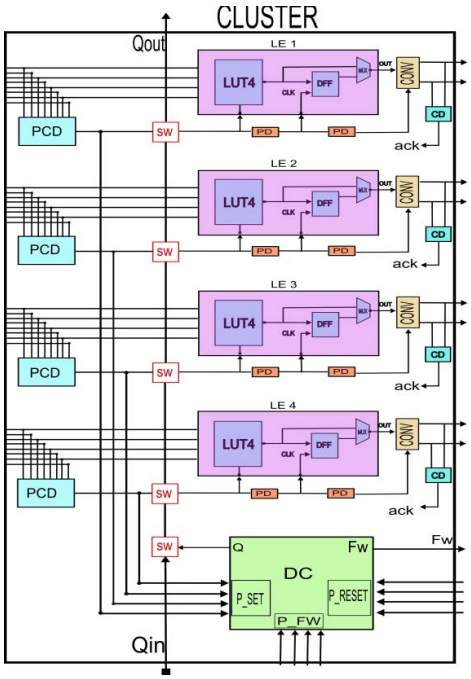


Figure 12: Cluster with DC.

The DC based distributed control in the cluster is used to propagate the control path. When input data is ready, PCDs in the PLEs in the cluster will generate trigger signals which can be collected by DC control for group triggering. Some PLEs may need to execute concurrently and others sequentially. SW between PCD and LE allows either the selection of self-trigger directly from its corresponding PCD for sequential operation or group-trigger from DC for concurrent operation. After computation is completed, DC withdraws the data and propagates the control signal to the next stage. This structure also allows data feedback channels from the output of each PLE to the input PCDs of other PLEs. This will be discussed in more detail later.

### *D. Programmable DC Circuit*

Distribution control using DCs was proposed in [10, 11]. Some extension work on direct mapping of asynchronous control circuits from Petri Nets to DCs were also reported in [12, 13].

A DC can be simply defined as shown in Figure 13(A) with set function (logic1 and 2), reset function (logic 3) and a control signal (Q or Qb at the SR latch). When SET function is activated, the control is passed to the DC.

When RESET function is activated, the control is removed from the DC and passed on to the next stage. Basically DCs are used to build distributed controllers as all DCs work based on handshake protocols.

Each DC includes an elementary two stage automaton. The overall system of a network of DCs is therefore a product of such automatons.

However the DC used here is not the same as conventional DCs as flexibility is needed. This is because all four input signals are not necessarily always used in the DC as some PLEs may be involved in sequential operations. All depends on the configuration required. For example, if only two PLEs are run concurrently, only the two corresponding ready signals are used to configure the DC to propagate control and so on.

Here the set and reset functions all need to be programmable. The basic structure of DC consists of three NAND gates and three logic blocks. A set of SET signals (s1 – sn) will trigger signal Q and the forward signal (Fw) based on the programmable logic blocks "Logic 1" and "Logic 2". The RESET signals (r1 – r n) will reset DC back to its default state through programmable logic block Logic 3. This is schematically shown in Figure13 (A).

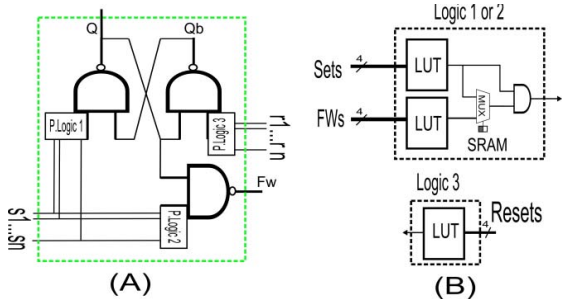


Figure 13: (A) DC Structure. (B) SET & RESET logic Boxes.

The programmable logic SET and RESET blocks are implemented with LUTs, which can be programmed to cover all possible combinational relations of their inputs. In this case each logic block has four inputs because our cluster structure has $n$=4. This is shown in Figure 13 (B) in more detail.

In order to keep these SET and RESET programmable logic blocks to a practical scale, basic timing assumptions can be used within them. Although this makes these blocks not strictly SI, the delays within such small and local areas can be more easily and reliably managed. Based on this argument we have gone for small scale timing assumptions in this tradeoff.

## IV. Design Flow

Existing asynchronous FPGAs either need a completely new design flow to implement their fully SI/DI solutions or are designed based on desynchronization methods.

By retaining the LE structure of conventional FPGAs and having a similar organization at the cluster level, our asynchronous FPGA architecture allows the synthesis of PLEs or clusters through the existing FPGA design flow to a large degree. Because of the replacement of clocks with the DC based distributed control, not all parts of the existing LE and LAB design flow can be directly applied. However, the basic mapping method should be directly applicable in principle, although modifications are needed to accommodate the new control structure.

As for the DC based distributed control, direct mapping for asynchronous circuits provides a suitable solution for us. This is based on Petri net specifications of the control path which can be directly mapped onto a DC based control structure. Here we propose the design flow for systems using this asynchronous FPGA architecture in Figure 14.

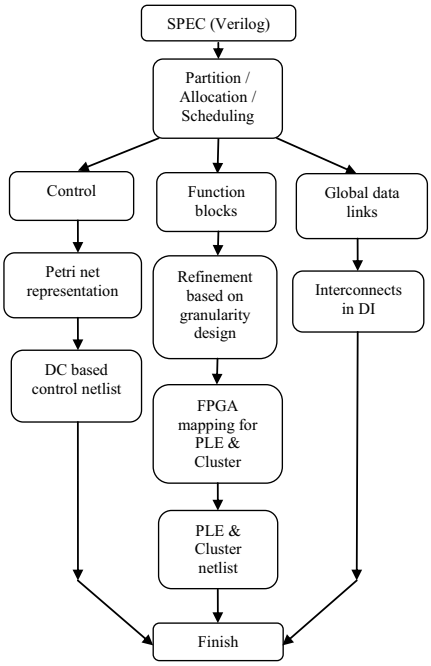


**Figure 14: System design flow.**

The system specification is assumed to be in a design language such as Verilog. This Verilog specification is passed to the next step where after partition, allocation, and/or scheduling, the design is divided into control, data path function blocks, and global data links. This step is similar to the automatic division of control and data paths in the process described in [12] and the techniques there can be re-used with minimal modification.

Control is represented in Petri net models, which can describe all types of control flow found in a Verilog system specification. For instance, common control elements such as fork, join, and arbitration can be represented by the Petri net models in Figure 15 which is taken from [12].

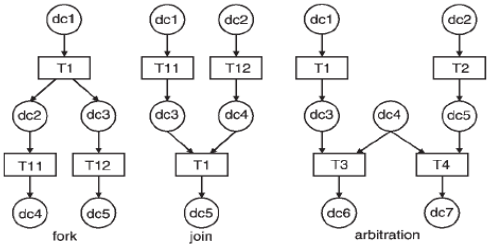


**Figure 15: Petri net models of control elements.**

Such a Petri net control model can then be used to generate the DC based distributed control with the direct mapping techniques described in [12]. For example, as shown in Figure 15, the places in this Petri net model directly indicate DCs. In other words, for each place in the control Petri net model, a DC is specified in the final implementation. The transitions and connection topology among the DC places are implemented through the SET and RESET logic of the DCs and interconnections between them.

The data path function blocks can be similarly derived through a step of colored Petri net modeling [12]. Once the general function blocks have been synthesized, they need to be refined based on the FPGA granularity for partitioning, depending on PLE and cluster sizes. This step is not available directly from [12]. However, this same step exists in converting a general VLSI design to FPGA implementation so the same methods can be applied. By keeping the PLE and cluster sizes of conventional FPGA we have made this step straightforward. This is then followed by obtaining PLE and cluster circuits using existing FPGA mapping techniques.

The global data interconnect fabric mainly consists of the channels for data communication. In our design flow it is implemented in dual-rail DI circuits directly. Their generation is a straightforward process.

## V. DESIGN EXAMPLES

This section describes example system designs which demonstrate possible ways of configuring systems on the architecture described in the preceding sections as well as the flexibility and features of this architecture.

A four bit ripple-carry full adder shows the flexibility of intra-cluster operation organization and independent DC control of PLEs. The second example of a 3-to-8 decoder circuit uses a single DC to control units in multiple clusters.

### *A. Full Adder Implementation*

Figure 16 shows the implementation of a four bit ripple carry adder using two logic clusters. This is to demonstrate the typical behavior of the ripple carry adder where each stage of the adder wait until its previous stage has completed computation and propagated its carry output signal.

Signals A0-A3, B0-B3 and CIN are assumed to be from the previous stage. The arrival of the signals can be in any order because of variable interconnect lengths and computation latencies, based on the overall DI inter-cluster communication assumption. When some of the inputs from the previous stage A0-1, B0-1, CIN have arrived in Cluster 1, some of the PLEs in this cluster can start computing. For example, LE2 may start its computation to generate its carry out signal C1. The trigger signal for LE2 was generated from its corresponding PCD once valid A0, B0 and CIN signals are detected without any intervention from DC control.

The C1 signal generated by LE2 is fed via an internal feedback channel (such channels are not shown in Figure 12 but are allowed in the architecture) to satisfy the PCD conditions of LE3 and LE4. Their PCDs produce two trigger signals to DC, which is waiting to collect these along with the PCD Signal from PLE1. Once all three of these signals have been collected by DC, it generates a merged trigger signal for the parallel triggering of LE4, LE3 and LE1. This merged trigger signal is in fact passed through all four PLEs through a chain consisting of all four SWs. The SW in PLE2 will not generate a second trigger locally for LE2 because it is programmed to respond to its local PCD instead of the DC control. The resulting concurrent action among three PLEs (4, 3 and 1) generates three latched outputs S0, S1 and C2.

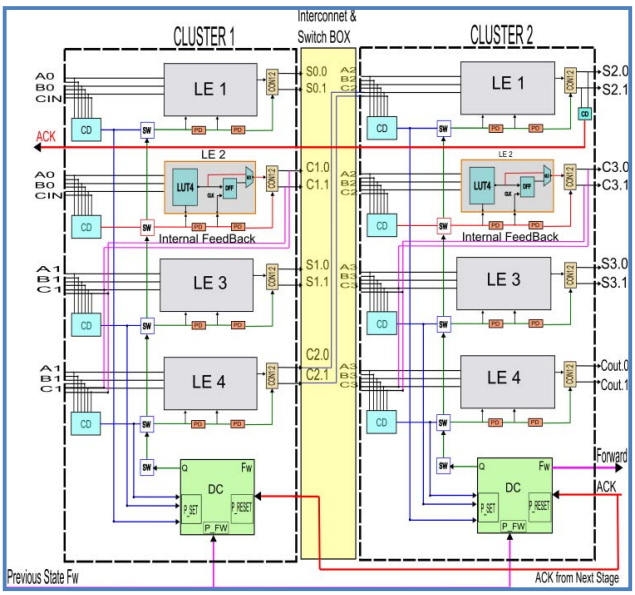


**Figure 16: Four bit full Adder example.**

After both clusters have completed their computations, the acknowledgement signal ACK will be generated by the output CD together with the output data to allow the previous stage to clear its data. There may not be a need to collect CDs from all PLEs for this acknowledgement as the designer may view the cluster as a small enough block so that internal timing assumptions can be made. In this case a subset of CDs may suffice to acknowledge the previous stage.

After this, the forward signal (Fw) will then be generated once the previous stage releases its data and forwards the control to the next stage to start a new round of operation. The same operation happens at the next stage interface and once the data has been consumed, the ACK coming from the right hand side will reset the DCs in both Cluster 1 and Cluster 2.

The circuits in this example demonstrate that not all PLEs must have all the components included in Figure 2 in use. For instance, only when an acknowledgement signal is needed from a PLE will it use its right hand side CD block. It also demonstrates the flexibility and programmability of DC set and reset blocks. In this case the DC setting is not directly related to the PCD of PLE2.

### *B. 3-to-8 Decoder Implementation*

The example in this section demonstrates that units in multiple clusters can be controlled with a single DC. Figure 17 shows an implementation of a 3-to-8 decoder with two clusters. It can be used as an address decoder. The combination of A, B and C inputs causes one of the 8 output lines to be selected.

DC functions are configurable, for this example DC in Cluster 1 is configured as an AND gate. When A, B and C signals are detected in Cluster 1, the forward signal Fw will be sent to Cluster 2. As shown in Figure 17, the trigger line of DC 2 is linked with Cluster 1. This allows parallel outputs to be generated from both Clusters. Only one ACK signal is required to acknowledge the previous stage, in this case it is from the right-hand side CD of PLE1. Again one RESET signal to DC will reset the logic in both clusters.

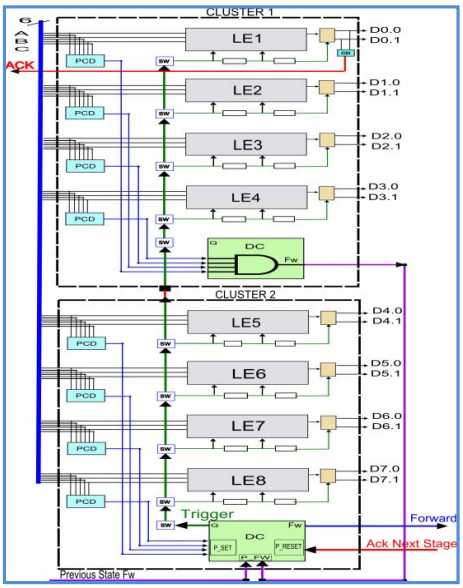


**Figure 17: 3-to-8 decoder example.**

## VI. CONCLUSIONS AND OUTLOOK

This paper describes a detailed low level circuit realization of a previously proposed asynchronous FPGA architecture. Different from existing asynchronous FPGA architectures, it tries to strike a sensible balance between maintaining similarities to current synchronous FPGA structures whilst achieving full asynchrony in places where it matters most, i.e. in long interconnect links. We envisage that the proposed approach will allow more flexibility in adjusting possible levels of DI, according to the needs of reconfigurable logic applications, building on the strengths of type-3 architecture compared with those of types-1 and -2.

By keeping the single-rail data representation of current FPGAs "in the small" in combinational logic computation units, this architecture makes it possible for designers to use existing FPGA logic mapping tools in block design. By introducing delay-insensitivity "in the large" into the inter-block long data links, it provides the advantages inherent to asynchrony to these interconnects, namely variation tolerance and latency robustness.

Other advantages of such a hybrid structure include the lower size and power requirements for combinational logic circuits using single-rail data representation, and robust distributed asynchronous control and DI inter-block communication, which allow efficient computation and correct operation across a wide Vdd range.

In order to realize such architecture, a number of structural choices were made and detailed hardware elements have been designed. It was decided that the granularity and block structures should broadly follow the practice in current commercial FPGAs.

The basic building block of the architecture, the programmable logic element (PLE), has been designed in detail. It includes a number of finer grain components, including a logic element (LE) inherited directly from current commercial FPGA, completion detection circuits and delay matching / data bundling circuits.

Some of the completion detection and the bundling delays in the PLE are programmable to cater for functional configurability and variation tolerance concerns.

This type of PLE has been demonstrated to work under variable Vdd across a wide range, with reasonable latency and energy behavior.

On the level above PLEs are the clusters. The standard cluster has been designed with a David Cell (DC) distributed control unit managing the operations of the PLEs in a cluster. This asynchronous control fully replaces the intra-cluster clock system in current commercial FPGAs providing the complete equivalent functional set, which includes both parallel and sequential operations of the PLEs in any possible arrangement.

A design flow for systems in this architecture has been proposed which makes maximal use of existing asynchronous and FPGA synthesis methods.

Two case studies have been carried out to demonstrate the functional capabilities of the architecture. A four-bit ripple carry full adder showcases the flexibility of intra-cluster DC control. A 3-to-8 decoder additionally demonstrates inter-cluster control from a single DC.

It is our plan to carry out comprehensive comparative studies between systems constructed in our architecture and in other types of asynchronous FPGA architectures. We also plan to further develop and complete the design and synthesis flow, including a fully automatic synthesis method, for the architecture studied in this paper.

## ACKNOWLEDGMENT

This work was supported by EPSRC DTA Studentship and grant Holistic (EP/G066728). Special thanks for Dr Stephen Bell from Rutherford Appleton Laboratory for his support in using Cadence tool kits and simulation techniques.

## REFERENCES

[1] D. Boning, and S. Nassif, "Models of Process Variations in Device and Interconnect". In Design of high-performance microprocessor circuits, A. Chandrakasan, W.J. Bowhill, F. Fox Eds. IEEE Press: New York. pp 98 – 116, 2001.

[2] International Technology Roadmap for Semiconductors. Available from: http://www.itrs.net/.

[3] A. Singh, and M. Marek-Sadowska, "FPGA interconnect planning". In Proc. 2002 international workshop on System-level interconnect prediction. ACM: San Diego, Cal, USA, 2002.

[4] J. Sparsø, and S. Furber, "Principles of asynchronous circuit design - A systems perspective". Kluwer Academic Publishers, 2001.

[5] D. Shang, F. Xia, and A. Yakovlev, "Asynchronous FPGA architecture with distributed control". In Circuits and Systems (ISCAS), Proceedings of IEEE International Symposium, pp 1436 – 1439, 2010.

[6] J. Rose, A. El Gamal and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays". In Proceedings of the IEEE. Vol 81, pp 1013-1029, 1993.

[7] V. Betz, and J. Rose, "How much logic should go in an FPGA logic block". In Design & Test of Computers. Vol 15, pp 10-15, 1998.

[8] E. Ahmed, and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density". In IEEE Trans. VLSI. Vol 12, pp 288-298, 2004.

[9] I. Koun, T. Russell, and J. Rose, "FPGA Architecture: Survey and Challenges". In Found. and Trends in Electron. Des. Autom. Vol 2, pp 135-253, 2007.

[10] R. David, "Modular Design of Asynchronous Circuits Defined by Graphs". In IEEE Trans. Computers. Vol 26 , pp 727-737, 1977.

[11] V. Varshavsky, and V. Marakhovsky, "Support for discrete event coordination". In Proc. Int. Workshop on Discrete-event Systems (WODES), pp 332–340, August 1996.

[12] D. Shang, F. Burns, A. Koelmans, A. Yakovlev, and F. Xia, "Asynchronous system synthesis based on direct mapping using VHDL and Petri nets". In IEE Proceedings – CDT. Vol 151, pp 209-220, 2004.

[13] D. Shang, "Asynchronous communication circuits: Design, test, and synthesis". PhD thesis. University of Newcastle upon Tyne, 2003.

[14] S. H. Kulkarni, D. Sylvester, and D. Blaauw, "A Statistical Framework for Post-Silicon Tuning through Body Bias Clustering". In Proc. IEEE/ACM ICCAD, San Jose, CA, USA, pp 39-46, 2006.

[15] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester, "Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation". In Proc. IEEE/ACM ICCAD, Ann Arbor, MI, USA, pp 1023-1028, Nov 2005.

[16] L. Cheng, J. Xiong, L. He, and M. Hutton, "FPGA Performance Optimization Via Chipwise Placement Considering Process Variations". In Proc. FPL, Aug 2006.

[17] S. Sivaswamy, and K. Bazargan, "Statistical Analysis and Process Variation-Aware Routing and Skew Assignment for FPGAs". In ACM Trans. Reconfigurable Technol. Syst. Vol 1, pp 1-35, 2008.

[18] R. Payne, "Asynchronous FPGA architectures". In IEE Proceedings – CDT. Vol 143, pp 282-286, 1996.

[19] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for implementing asynchronous circuits". In Design & Test of Computers, 11, IEEE, Autumn 1994.

[20] J. Teifel, and R. Manohar, "An asynchronous dataflow FPGA architecture". In IEEE Trans. Computers. Vol 53, pp 1376-1392, Nov 2004.

[21] Achronix Semiconductor Corporation. Available from: http://www.achronix.com/.

[22] A. Royal and P. Cheung, "Globally Asynchronous Locally Synchronous FPGA Architectures", In Proc. FPL 2003. Springer Berlin / Heidelberg. pp 355-364, 2003.

[23] M. Najibi, K. Saleh, M. Naderi, H. Pedram, and M. Sedighi, "Prototyping globally asynchronous locally synchronous circuits on commercial synchronous FPGAs". In Proc. RSP 2005, Montreal, Canada, June 08 – June 10, 2005.

[24] X. Jia, and R.Vemuri, "Using GALS architecture to reduce the impact of long wire delay on FPGA performance". In Proc. ASP-DAC 2005, pp 1260-1263, Jan 18-21, 2005.

[25] A. Baz, D. Shang, Fei. Xia, and A. Yakovlev, "Self-timed SRAM for energy harvesting systems". In Proc. PATMOS 2010, LNCS 6448, pp 105-115, Grenoble, France, Sept 7-10, 2010.

[26] P. Sedcole, and P. Cheung, "Within-die variations in FPGAs: disaster or opportunity?" Poster, Imperial College London. Available from: http://www3.imperial.ac.uk/pls/portallive/docs/1/41113698.PDF.

[27] C. Traver, R.B. Reese, and M.A. Thornton, "Cell designs for self-timed FPGAs". In ASIC/SOC Conference, 2001. Proceedings. 14th Annual IEEE International. 2001.