Asynchronous FPGA Architecture with Distributed Control

Delong Shang, Fei Xia, Alex Yakovlev
MSD Group, School of EECE, Newcastle University
Newcastle upon Tyne, NE1 7RU, England, U.K.
{delong.shang,fei.xia,alex.yakovlev}@ncl.ac.uk

Abstract—**Asynchronous techniques have become more significant with continued scaling of VLSI technologies. This paper proposes an asynchronous FPGA architecture. Different from previous methods of introducing asynchrony into FPGAs, our method seeks to preserve the current FPGA cell structure as much as possible, whilst achieving delay insensitivity in the inter-cell interconnects. By using David Cells as the central technique in the delay insensitive clock replacement, this method is conducive to the establishment of an automatic design and synthesis flow. It also particularly caters for low power designs, where current FPGA solutions are not effective yet.**

I. Design challenges in nanometer technology

CMOS feature size shrinking in nanometre technology has enabled a wide spectrum of applications ranging from high performance computers to low power portable devices. These in turn require a variety of computation and communication architectures to optimize silicon performance. The International Technology Roadmap for Semiconductors (ITRS) [1] and research [2] predict multitudes of difficulties in scaling for wires and transistors in future technology nodes.

As the decrease in CMOS feature size keeps improving performance, programmable logic is becoming increasingly attractive. However, traditional FPGA architectures are facing challenges with the growing logic size of chips: 1) Delays of long interconnect wires can easily dominate all other delays; 2) To evenly distribute global clock signals all over the FPGA area requires great efforts because of clock skew; 3) FPGAs are more likely to contain a multitude of modules running at different clock frequencies, with data signals appearing to be asynchronous in the new clock domain when moving data across modules; 4) Increased power consumption; and 5) process variations seriously affect circuit designs.

The ITRS predicts that asynchronous circuits can be useful for these problems. As asynchronous circuits are hard to design and test, especially because of a lack of automatic design toolkits, they previously were primarily only used for the following reasons: 1) When speed requirements cannot be met with a synchronous design; 2) When power requirements cannot be met with a synchronous design; 3) When dictated by requirements; And 4) to solve special problems.

However, asynchronous logic has made enormous progress in the past decade. Complete asynchronous processors are routinely designed in the lab, and CAD toolkits are being developed. Since two of the main advantages of asynchronous logic are the absence of a clock and low power consumption, it seems natural to apply asynchronous technology to the design of FPGAs [3].

In this paper, we propose a low power asynchronous FPGA architecture. The main features of this work include: 1) Keeping the basic structures of the traditional FPGAs; 2) Designing a wrapper based on David Cells to implement a distributed control structure which replaces the clocks; and 3) proposing a design flow for asynchronous FPGAs.

The remainder of the paper is organized as follows: Sections II describes existing asynchronous FPGAs including the architectures and advantages and disadvantages; Section III introduces our new proposed low power asynchronous FPGA and a design flow; Section IV gives the conclusions and the outlooks of the future work.

II. Existing Asynchronous FPGAs

Traditional FPGA architectures normally consist of configurable logic blocks (CLB), routings (interconnect links), and IO ports. As the FPGAs are pre-designed, there should be some limitations on hardware resources (components) of the FPGAs, such as interconnect links, clock buffers, the numbers of inputs of each CLB and so on.

For example, most existing FPGAs only have four inputs for each Look Up Table (LUT). As a result, applications with more than four inputs may need long inter-block interconnect links. This will introduce long latency and may cause hazard problems. Synchronous implementations can overcome these problems by increasing clock period based on the worst case timing assumption at the cost of reduced performance and increased power consumption.

Asynchronous FPGA solutions have been presented in the literature [3-14] since 1992. Payne in his paper [4] gave a summary for MONTAGE [5], PGA-STC [6], and STACC [7].

MONTAGE from the University of Washington, was the first asynchronous FPGA, though it includes a clock signal for implementing synchronous circuits as well. It is extended from their own synchronous FPGA architecture by adding special arbitration cells, and modifying the function unit.

PGA-STC developed at U.C. Davis is targeted at the implementation of two-phase bundled-data systems such as Micropipelines [15]. The architecture is loosely based on that of

The work is supported by EPSRC projects at the School of EECE, Newcastle University, England, U.K.

the Xilinx FPGAs, with modification to the function unit, and addition of arbitration cells and programmable delay elements.

STACC is an architecture developed at the University of Edinburgh. It is a dedicated architecture for the implementation of four-phase bundled-data systems. The STACC architecture is based on that of fine-grain FPGA architectures where the global clock is replaced by an array of timing-cells that generate local register control signals.

All the above asynchronous FPGAs amend the function units to avoid hazards in signals, and use timing assumptions to guarantee the correctness of the asynchronous circuits.

Royal and Cheung proposed a Globally Asynchronous Locally Synchronous FPGA in their paper [11]. They mimic the GALS architecture used in general asynchronous circuit designs. A wrapper structure is presented in the paper. It is used to separate the synchronous blocks and asynchronous routings. The wrapper communicates with the synchronous block synchronously and provides a local clock signal to the block. It uses handshake protocols to interface with the asynchronous routings. In addition, Micropipelines are used in asynchronous routings to improve the performance.

Jia and Vemuri [12][13] proposed an asynchronous FPGA with large size of blocks using the same GALS structure as [11] and proposed a design flow for their FPGAs as well.

Teifel and Manohar [10] presented a high performance asynchronous FPGA. They used fine-grain asynchronous pipelines as the basis, and proposed a completely new logic cell for the FPGAs, in which there are Function Unit, Merge Unit, Split Unit, and Token Unit.

Fang et al in [14] presented a 3D asynchronous FPGA, which extended Teifel and Manohar's work from 2D to 3D.

Wong, Martin, et al [3] proposed a completely new asynchronous FPGA. They proposed the precharge half-buffer (PCHB) circuits for implementing logic cells and unlike the other asynchronous FPGAs, in their solution the logic cells can be Speed Independent (SI) circuits and the interconnect links are Delay Insensitive (DI) circuits. The C elements are used to implement circuits bigger than cluster logic blocks to guarantee SI as the limitations of hardware resources.

III. PROPOSED ASYNCHRONOUS FPGA

Most of these asynchronous FPGAs are proposed for improving performance to overcome the global worst case timing delays. In general, these can be classified into two types: 1) Working under timing assumptions with modification on logic cells (The GALS based FPGAs, for example, belong to this class); And 2) working under little timing assumptions with completely new logic cells.

Type 1 in general replaces the global worst case timing assumptions by the individual worst case timing assumptions. The problem of this type is that the methods mix the delays both inside blocks and between blocks. In nanometre technology, process variations will seriously affect delays, and it will be hard to set matching delays for inter-block interconnects.

Type 2 is fine from the performance point of view. However as the methods are completely new with novel cell structures, they can be impractical due to the lack of support from existing commercial EDA toolkits and testability is a problem too. In addition, as some heavy asynchronous components, such as C elements, are used, they can consume more power.

Here we propose a solution different from these two types. Our proposed asynchronous FPGA architecture is sitting in the middle. It seeks to keep the traditional logic cells (CLB level and/or no more than 4 CLBs --- cluster logic level commonly defined in FPGA industries) which potentially make it possible to use existing commercial EDA toolkits. It also employs a distributed control system to obtain DI in the interconnect links with only the delays inside logic blocks needing to be considered. Our main contribution gets rid of the delay constrains on the interconnect links, possibly very long depending on routing. This provides more process variation tolerance as delays in small local circuits (blocks) are easier to manage.

Our proposed architecture mainly focuses on low power. The new architecture avoids hazard signals, has no clock distribution problems, and provides more flexibility and good potential for automatic design and synthesis. This flexibility will benefit system designers who want to employ more sophisticated low power design techniques.

A. Architecture

Existing synchronous FPGA architectures can be simply taken as CLBs used for functionality and interconnect links used for data communications and control communications. Each CLB consists of a LUT, a D flip-flop, a Multiplexer, and other logic, for example carry logic. The LUT is used to implement the combinational computation. The output of the LUT changes with changes in the inputs. Our proposed architecture preserves this normal CLB structure to maintain usability of existing design toolkits. Basically we want to derive a technology which delivers an asynchronous equivalent to the common existing synchronous FPGA, with correct control and data path replacement techniques which get rid of the clocks.

For this purpose we need to first avoid hazard signals, which asynchronous circuits potentially have. For this we introduce a control signal, generated only when all inputs have settled. This can be done by using data encoding methods, such as dual-rail data, and completion detection logic. In the absence of a global clock, we also need to explicitly manage the delays inside CLBs. As the normal CLBs use single rail data encoding, it should use dedicated delay signals. In general a CLB is small in terms of circuit size and not distributed, making it is easy to manage the delays inside CLBs. However the delays on interconnect links are harder to manage as placement and routing may put them at arbitrary locations, allowing them to be affected more by process variations.

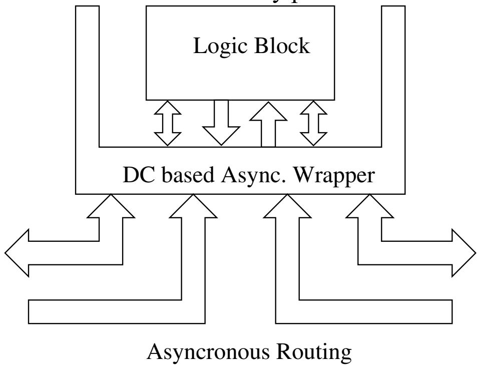

Figure 1. Proposed asynchronous FPGAs with wrapper

We propose a wrapper structure as shown in Figure 1 to connect asynchronous routings and control signals to CLBs. This wrapper uses David Cells (DC). DCs, proposed in [16], form a kind of distributed control circuit. Each DC includes an elementary two state automation. The overall system is therefore a product of such automations. Some extension work were developed in [17][18][19]. In general DCs can be defined as shown in Figure 2, in which there are a backward signal (bk), a forward signal (fw), a set of set signals (s1 to sn), and a set of reset signals (r1 to rm). A DC consists of a basic structure of three NAND gates and three logic blocks. Other implementations may be used [19].

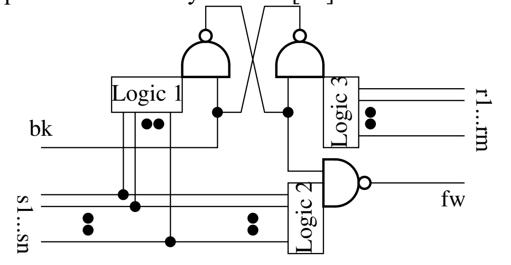

Figure 2. Definition of David Cells

The logic 1 and 2 blocks are used to implement the set and propagate functions and the logic 3 block for the reset function.

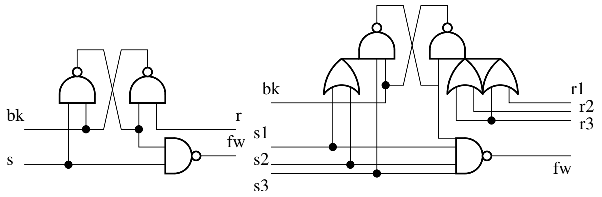

Figure 3. Examples of DCs

Two DC examples are shown in Figure 3. The left hand one is the simplest DC. Here we use the complex one on the right as an example to explain how a DC works. When both s1 and s2 are low or s3 low, the DC is set up and then the bk is set to low. The {(s1,s2), bk} or {s3, bk} forms a handshake protocol. So after bk is low, either both s1 and s2 are set to high or s3 high. After that, fw is set to low to propagate the control to the next stage. Here {fw, (r1,r3)} or {fw, (r2,r3)} forms a handshake protocol as well. This DC therefore has the set function $!s1*!s2+!s3$ and reset function $!r1*!r3+!r2*!r3$.

For an application, some of the CLBs may execute in parallel and some of them in sequence. The DC based distributed control can be used to implement any combinations and compositions of these scenarios.

This DC based distributed control is used to propagate the control path. After data is ready, it triggers the corresponding DC(s) and then the corresponding CLB(s). After finishing, it withdraws the data and propagates the control to the next stage. This way it implements an inter-block DI replacement of a clock-based control system. The proposed wrappers are used to link the distributed control and CLBs. The block diagram of a possible implementation of the wrapper is shown in Figure 4, in which the dotted box is the normal CLB and the dashed box is the proposed wrapper.

In Figure 4, P stands for programmable units; PD for programmable delays; COV for converter from single rail to dual-rail; CD for completion detection logic; DC for David Cell; s for set and r for reset.

This block diagram clearly shows that the delay on interconnect links is no longer relevant for correctness and the PDs only need to match the delays inside the CLBs. The structure is not limited to one wrapper per CLB. In practice, a wrapper can be used for a cluster of logic blocks.

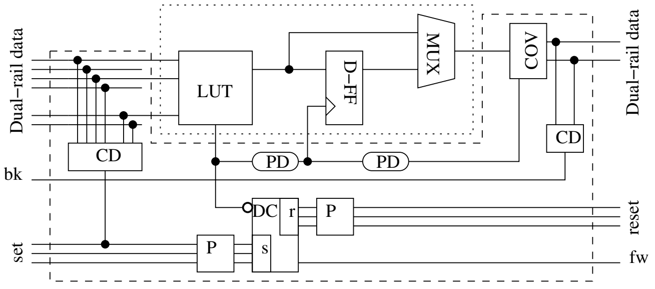

Figure 4. A possible wrapper implementation

The wrapper works as follows: All ready data pass through the left hand CD and then trigger the DC based on the requirements. After that a signal is generated to pass all ready data to the LUT and start computation. Here timing assumptions are used for the delay for latching the computation results and the delay for converting the single rail data to dual-rail data. After that the right CD generates the bk signal to allow the previous stage to start the next cycle.

For complex applications which require more than one basic block, the control signals among DCs in multiple wrappers will form set-reset signaling chains.

B. *Design Automation*

Here an EDA design flow is proposed for automatically design and synthesis systems based on the proposed asynchronous FPGA architecture. The design flow is focused on designs starting from high level specifications.

The proposed asynchronous architecture has three parts: function parts which are limited to small size circuits such as one CLB or several CLBs forming a cluster with efficient internal links, wrappers and asynchronous routings.

Existing EDA toolkits are not suitable as they do not support asynchronous designs. However as the function parts are synchronous circuits, especially as we limit the scale of each function block to one CLB or several CLBs with internal links, existing FPGA designing toolkits can be borrowed to partition functionality and map the partitioned functionality as we preserve the basic CLB structure of the traditional FPGAs.

However, the asynchronous wrapper and the asynchronous routings cannot be designed using the current most popular asynchronous EDA toolkits, such as Handshake Solutions [23] and Balsa [24], as they mix the datapath and control parts.

One asynchronous design method called *direct mapping* and related EDA toolkits [22] was developed by us. In this design flow, after partitioning and scheduling, a high level specification is divided into datapath, global control, and local control between the datapath and the global control. This matches our proposed asynchronous FPGA architecture. So the wrappers and asynchronous routings can be designed based on the direct mapping method.

To further develop the existing method, for asynchronous FPGA design, we propose that Petri nets [21] be used as an intermediate language for the above DC based distributed control as Petri net specifications can be directly mapped to DC based circuits [20]. Figure 5 shows an example Petri net model. The mapped DC circuit is shown in Figure 6. This example shows that complex control structures including a combination of sequential and parallel executions can be described in Petri net models and mapped to DC circuits in a reasonably straightforward manner.

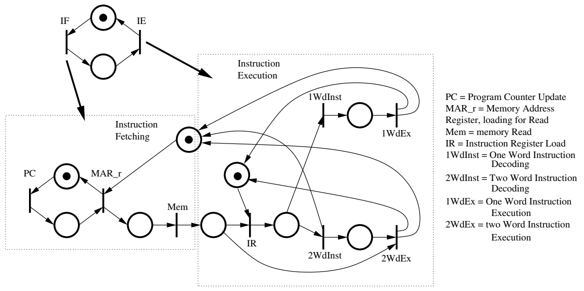
IF
IE
Instruction Execution
Instruction Fetching
PC
MAR_r
Mem
IR
1WdInst
1WdEx
2WdInst
2WdEx
PC = Program Counter Update
MAR_r = Memory Address Register, loading for Read
Mem = memory Read
IR = Instruction Register Load
1WdInst = One Word Instruction Decoding
2WdInst = Two Word Instruction Decoding
1WdEx = One Word Instruction Execution
2WdEx = two Word Instruction Execution

Figure 5. Petri net example

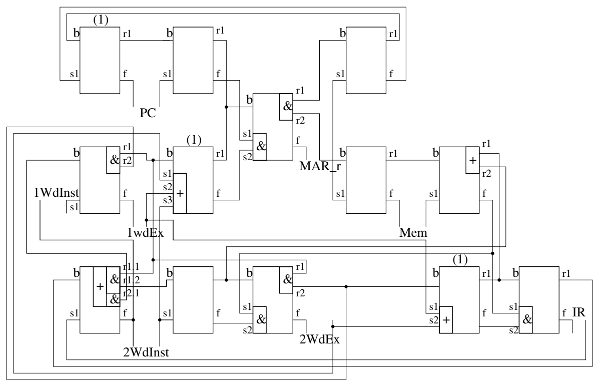

Figure 6. DC based distributed control

The proposed automatic design flow is therefore a combination of existing commercial FPGA development toolkits and direct mapping asynchronous design methods. From a high-level behaviour and functional specification, the functionality is partitioned to CLBs (clusters) using existing FPGA tools, and then the inter-block relationships are scheduled to obtain an intermediate Petri net model of control communications, which is purely data/event driven without mandating clocks. After that DC based wrappers are implemented to form an asynchronous distributed control, resulting in DI inter-block data and control links. In addition, asynchronous routings are implemented to transfer data between CLBs.

During the wrapper mapping process, the delays matched to the corresponding CLBs (clusters) should be considered.

IV. CONCLUSIONS AND OUTLOOKS

In this paper an asynchronous FPGA architecture is proposed. The fundamental features of this architecture are the preservation of current FPGA cell structures and delay insensitive inter-block data and control links. The preservation of current FPGA cell structure made it possible for the continued use of existing FPGA design toolkits and the simple DC-based distributed control made it possible for a straightforward automatic design flow to be developed. The DI across interconnect links and wrapper-based cell control also imply self-timed and data-driven operation targeting low power. Future work includes the development of a general systematic method of developing Petri net control models targeting this FPGA architecture and further development of the direct mapping method to include wrapper implementations. The David Cell technology also has significant scope for further development to suit future FPGA systems.

REFERENCES

[1] International Technology Roadmap for Semiconductores: http://public.itrs.net/.

[2] R.M. Ho, K.W. Mai, M.A. Horowitz, The Future of Wires, Proc. of IEEE, Vol 89, No 4, pp 490-504, 2001.

[3] C.G. Wong, A.J. Martin, and P. Thomas, An Architecture for Asynchronous FPGAs, Proc. of Field-Programmable Technology (FPT) 2003, pp. 170-177, 15-17 December 2003.

[4] R.E. Payne, Asynchronous FPGA Architectures, IEE Proc. on Computers and Digital Techniques, Special Issue on Asynchronous Processors, September 1996.

[5] S. Hauck, S. Burns, G. Borriello and C. Ebeling, A FPGA for Implementing Asynchronous Circuits, IEEE Design and Test of Computers, Vol. 11, No. 3, 1994.

[6] K, Maheswaran, Implementing Self-Timed Circuits in Field Programmable Gate Arrays, Master's thesis, U.C. Davis, 1995.

[7] R.E. Payne, Self-Timed FPGA Systems, Proc. of 5th international workshop on field programmable logic and applications, LNCS 975, pp 21-35, 1995.

[8] R. Konishi, H. Ito, H. Nakada, A. Nagoya, K, Oguri, etc., PCA-1: A Fully Asynchronous, Self-Reconfigurable LSI, Proc. of ASYNC 2001, March 2001.

[9] A. Rettberg and B. Kleinjohann, A Fast Asynchronous Reconfigurable Architecture for Multimedia Applications, Proc. 14th Symposium on Integrated Circuits and Systems Design, September 2001.

[10] J. Teifel and R. Manohar, Programmable Asynchronous Pipeline Arrays, Proc. 13th Intel Conference on Field Programmable Logic and Applications, September 2003.

[11] A. Royal and P.Y.K. Cheung, Globally Asynchronous Locally Synchronous FPGA Architectures, Proc. of 13th Field Programmable Logic and Application (FPL) 2003, LNCS 2778, Lisbon, Portugal, September 1-3, 2003..

[12] X. Jia and R. Vemuri, The GAPLA: A Globally Asynchronous Locally Synchronous FPGA Architecture, Proc. of 13th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) 2005, Napa, CA, USA, April 17-25, 2005.

[13] X. Jia, and R. Vemuri, Studying a GALS Architecture Using a Parameterized Automatic Design Flow, Proc. of ICCAD'06, November 5-9, 2006, San Jose, CA.

[14] D. Fang, S. Peng, C. Lafrieda, and R. Manohar, A Three Tier Asynchronous FPGA, Proc. of 23rd VLSI/ULSI Multilevel Interconnection Conference, Septmber, 2006.

[15] I.E. Sutherland, Micropipelines, Commun. ACM, Vol. 32, No. 6, pp. 720-738, 1989.

[16] R. David, Modular Design of Asynchronous Circuits Defined by Graphs. IEEE Trans. On Computers, Vol. 26, No. 8, pp. 727-737, August 1977

[17] V. Varshavsky and V. Marakhoysky, Hardware Support for Discrete Event Coordination, Proc. of WODES'96, pp. 332-340, Edinburgh, U.K., August 1996.

[18] A. Yakovlev and A.M. Koelmans, Petri nets and Digital Hardware Design, In Lectures on Petri nets II: Applications, Advances in Petri nets, Vol. 1492, pp. 154-236, 1998.

[19] D. Shang, Asynchronous Communication Circuits: Design, Test, and Synthesis, Ph.D thesis of Newcastle University, U.K., 2003.

[20] D. Shang, F. Burns, etc., Asynchronous System Synthesis Based on Direct Mapping using VHDL and Petri nets, IEE Proc. CDT, Vol. 151, No.3, 2004.

[21] J.L. Peterson, Petri net Theory and Modelling of Systems, Pretice-Hall, 1981.

[22] http://www.staff.ncl.ac.uk/alex.yakovlev/home.formal/besst/project-summary.html

[23] Handshake Solutions: TiDE:Timeless Design Environment, http://www.handshakesolutions.com/products_services/tools/Index.html

[24] Balsa: http://intranet.cs.man.ac.uk/apt/projects/tools/balsa/