A NOVEL ASYNCHRONOUS FPGA ARCHITECTURE DESIGN AND ITS PERFORMANCE EVALUATION

Xin Jia, Ranga Vemuri

University of Cincinnati
Cincinnati, OH, USA 45221
email:{jiax,ranga}@ececs.uc.edu

ABSTRACT

This paper proposes GAPLA: a Globally Asynchronous Locally Synchronous Programmable Logic Array architecture. The whole FPGA area is divided into locally synchronous blocks wrapped with asynchronous I/O interfaces. Data communications between synchronous blocks are controlled by 2-phase handshaking signals under bundled-data delay assumption. The size and shape of each locally synchronous block are programmable so that different modules in a design can be effectively implemented. By dividing the FPGA area into smaller blocks, the delays of long interconnect wires, which could easily dominate other delays in conventional FPGAs, only come into picture when there are communications between blocks. Therefore, each block could run at higher speed. The area overhead of adopting the GALS style in GAPLA architecture is estimated to be very small (about 7%). Experimental results show an up to 55% performance improvement compared to the conventional FPGAs.

1. INTRODUCTION

As the logic size of FPGA grows, there are some challenges facing by conventional FPGA architectures. First, the delays of the long interconnect wires can easily dominate all other delays. Several researchers address this problem by either pipelining the long interconnect delay or modifying the synthesis flow to allow the long interconnect to run for several clock cycles [1, 2, 3]. In those approaches, interconnects are treated as circuit components instead of conventional wires which makes a high performance solution hard to achieve. Also, these solutions are area-expensive. Second, to evenly distribute the global clock signals all over the FPGA area requires great efforts because of the clock skew. Third, FPGAs are more likely to contain a multitude of modules running at different clock frequencies since they have grown to sufficient sizes. Data signals appear to be asynchronous in the new clock domain when moving data across modules. Current FPGA architectures and CAD tools provide very few or even no support for asynchronous communications.

Introducing asynchronous concept into the FPGA architecture is a possible solution to the named challenges. In terms of interconnect delays, performance is dictated by the average of the interconnect delays rather than the worst-case delay. Hence use of long wires does not necessarily lead to a significant performance penalty. Clock distribution is no longer a problem since the global clock signals are removed. By adopting asynchronous design, FPGAs can provide architectural supports for communications across different clock domains. Different modules running at different clock frequencies can be easily glued together.

This paper presents GAPLA: a novel GALS Programmable Logic Array architecture. GAPLA architecture is based on our asynchronous wrapper design which allows the synchronous logic blocks to communicate with each other through a 2-phase handshaking protocol under bundled data delay assumption. The width of each data bundle and the size and shape of each synchronous logic block are all programmable. Thus, the borders of synchronous logic blocks are determined by the circuit functions to be implemented instead of the architecture, which provides the most effective bundling of data and prevents the waste of logic and communication resources.

2. RELATED WORK

There are several GALS systems in the literature. Bormann et al [4], Moore et al [5] and Muttersbach et al [6] invent different asynchronous wrapper designs which support 4-phase handshaking data transfer. In a GALS FPGA design, communications between synchronous logic blocks are through long interconnect wires with long delays. A 4-phase handshaking needs to go through the long interconnect wires 4 times to complete one data transfer and therefore produces huge delays. In our GAPLA architecture, we propose a simple and efficient asynchronous wrapper design which supports 2-phase handshaking protocol.

Quite a few asynchronous FPGA architectures have been proposed in the last decade [7, 8, 9]. Several of these architectures adopt the GALS concept. STACC [7] is loosely

0-7803-9362-7/05/$20.00 ©2005 IEEE

based on Sutherland's Micropipeline design. The clock signal of the data array is replaced by the handshaking control signals of the timing array in a mircopipeline like structure. PCA [8] is a self-reconfigurable programmable logic architecture. Data communications between logic blocks is realized by a wormhole message passing mechanism through the built-in-facilities. Royal et al proposes another GALS FPGA architecture [9] and the idea of using GALS architecture to limit the impact of long interconnect wire delay on the total FPGA performance. One common problem of the above architectures is that the size and shape of each synchronous logic block are fixed. Thus, the architectures put very strict constraints on the way applications are mapped. In our GAPLA architecture, we allow the size and shape of each synchronous logic block and also the data width of each I/O port to be highly programmable. Therefore, a design can be partitioned into modules of different sizes according to their functions and all the modules can be efficiently implemented.

3. THE GAPLA ARCHITECTURE

3.1. Architecture Overview

Figure 1 shows a block diagram of the GAPLA architecture which contains *asynchronous islands* connected in a mesh structure. Each asynchronous island contains a synchronous logic block and 4 asynchronous wrappers. Each asynchronous wrapper contains a local clock generator and I/O port controllers. Logic tiles inside the island can freely choose its clock from the four local clock signals. The clock grouping module is used to merge two asynchronous wrappers into one big wrapper to expand the I/O capacities of a clock domain. The routing resources between asynchronous islands contain horizontal and vertical routing channels for both data and handshaking control signals. Adjacent asynchronous islands are also directly connected to enable fast communications. The boxes marked "RSB" are the Routing Switch Boxes. The RSBs route both the data and the handshaking control signals.

3.2. Asynchronous Island Structure

3.2.1. Asynchronous Wrapper

An asynchronous wrapper contains a local clock generator and I/O port controllers. The local clock generator is a ring oscillator with programmable period. It can be paused by requests from the port controllers. A port controller is an asynchronous FSM working as a converter from the asynchronous domain to the synchronous domain. It controls the generation of the local clock to make sure that all the timing constraints are met for correct data communication. Metastability then can never cause the system to fail.

Some GALS local clock generators have been developed [5, 10] which can provide very stable and accurate clock signals. In the GAPLA design, we adopt the clock generator design of [5]. Figure 2 shows the clock generation and pause module proposed in [5]. The gate marked with a "C" is a Muller C-element. The ME block is a mutual exclusive element. The RC_i signal is the request signal to pause the next clock generation from port controller i. The next rising clock edge is delayed as long as one of RC_i is high. This scheme allows multiple ports to be active at the same time.

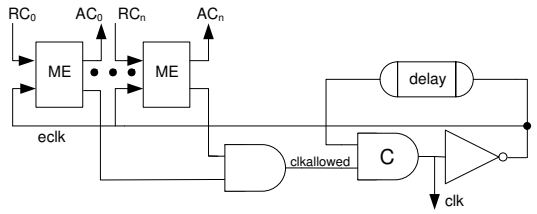

Fig. 2. The local clock generation and pause scheme [5].

Figure 3 shows our port controller design which supports 2-phase asynchronous handshaking communication to reduce the communication time. For a 2-phase asynchronous communication, a full handshaking cycle contains either Req+, Ack+ or Req-, Ack-. The "+" and "-" represent the rising and falling of the signals respectively. The output port controller works as follows. Initially both the Req and Ack signals are low. The EN signal is set to high by the logic block to indicate that data is ready to be sent out in the next clock cycle. After clk+, the Req+ and the data are sent out. Req+ causes signal RC to high which stretches the next clk+. The signal RC only returns to low after AC+ and Ack+. Ack+ indicates the asynchronous communication is accomplished and data has been received by the next stage logic. After RC returns low, the clock generator is freed. In the next clock cycle, if the signal EN remains high, data is sent out again along with Req-. The circuit then waits for Ack- to complete the second data transmission. The input port controller works in a similar way. A high EN signal, which causes RC to be high, indicates that new data is required for the logic to continue operation. A change in the Req signal, which means new data has arrived, pulls down RC and frees the clock signal to allow the incoming data to be latched. An event in the Ack signal represents the successfully receiving of the incoming data. Both the input and the output port controllers are demand-type, which means that once an asynchronous communication is issued (by the EN signal), the synchronous logic is stopped until the communication is finished.

Bundled-data delay assumption is adopted for the data signals of the asynchronous communications. A pair of handshaking control signals (Req and Ack) together with the data signals it controls is called a *communication channel*. The

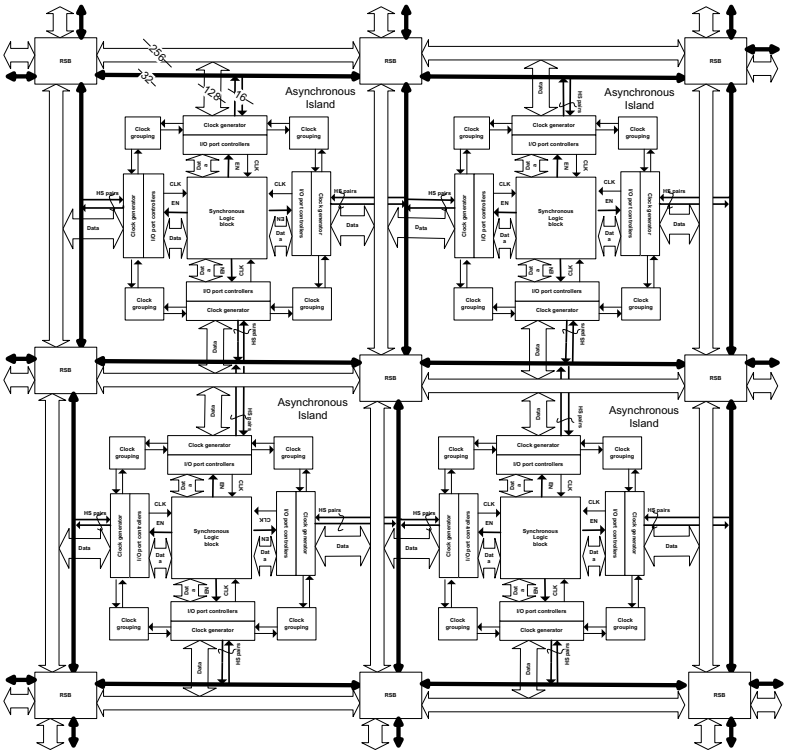

Fig. 1. Block diagram of GAPLA architecture.

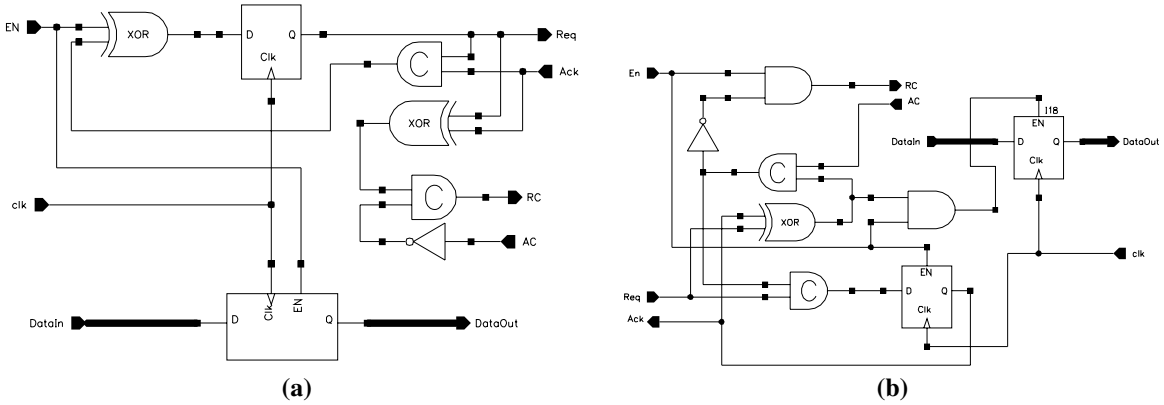

Fig. 3. Circuit design of an (a)output and an (b)input port controller.

number of data wires for each communication channel is programmable in the GAPLA architecture. This is realized by allowing the data width of each I/O port to be programmable as shown in Figure 4. The I/O port controllers control the I/O data registers by providing the "Enable" signals to the registers. We allow the "Enable" signals generated from the I/O port controllers to be connected to the bidirectional data registers through a connection matrix. Thus, each I/O port controller can freely choose how many bits of data are required for a communication. To simplify the connection matrix design, each I/O port can only control up to 64 bits of data. In GAPLA architecture, an asynchronous wrapper contains 8 input ports and 8 output ports. The total data signal wires for all the I/O ports are 128 bits.

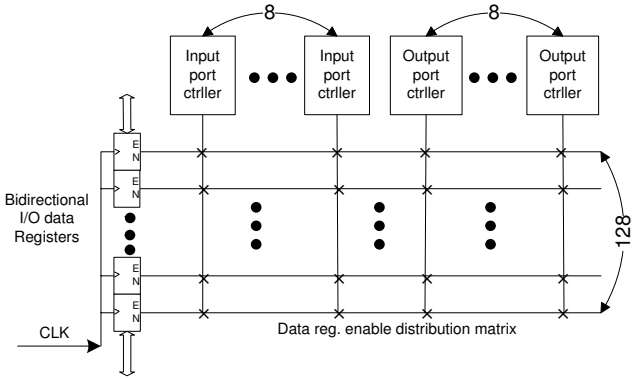

Fig. 4. The programmable I/O port data width.

3.2.2. *Synchronous Logic Block Structure*

The synchronous logic block can adopt any existing synchronous FGPA structure, but the size of each synchronous logic block must be big enough to implement reasonable functions. In the GAPLA design, we use the Xilinx Virtex II FPGA structure. Each synchronous logic block contains 24×20 Virtex II CLBs and 24 multiplier blocks. The 4 clock signals generated from the local clock generators are all distributed over the whole logic block. The logic controlled by a clock signal is called a *clock domain*. The logic resources are divided into 16 *Clock Distribution Units* (CDUs). The clock of a CDU can be freely chosen from the 4 clock signals. Therefore, the size and shape of each local clock domain is programmable within the size of a synchronous logic block. Figure 5 illustrates the clock distribution inside a synchronous block.

3.2.3. *Clock Grouping Module*

The idea of grouping two asynchronous clock domains into one clock domain is proposed in [11]. In GAPLA design,

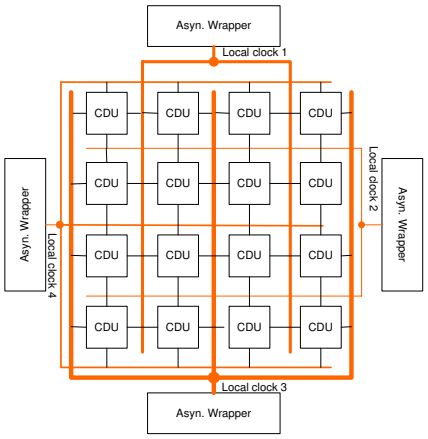

Fig. 5. Clock distribution of a synchronous block.

we adopt this idea to expand the I/O capacity of a clock domain. A C element can be used to group two clock generators to create a common clock signal [11]. The I/O capacity of a clock domain is thus doubled using one clock grouping module.

3.3. Inter-Island Routing Structure

The routing resources between asynchronous islands consist of two types of interconnections. The first is the direct and fast interconnections between adjacent islands. The second is the horizontal and vertical global routing channels to connect remote islands. The direct connections between adjacent islands consist of 8 handshaking pairs and 64-bit data and the global routing channels consist of 32 handshaking pairs and 256-bit data as shown in Figure 1. To make the bundled-data delay assumption valid, a predefined small delay is added to each Req signal in the routing structure.

A *Routing Switch Box* (RSB) is used to connect the horizontal and vertical routing channels at every corner. A RSB contains two parts: one part to provide routings for handshaking control signals called *CRSB* and one part to provide routings for logic called *LRSB*. Besides the switching activities, the CRSB needs to provide three more additional functions to the handshaking control signals, namely fanin/fanout and arbitration and merge. Fanin/fanout function is required when data communication is one-to-many or many-to-one. Arbitration is needed when data from different sources compete to communicate with one destination. Merge function is needed when data is broadcasting. A programmable fanin/fanout module is proposed in [11]. Figure 6 shows the structure of a CRSB. The central switch matrix is used to route the handshaking pairs to different directions. Four 4-input fanin/fanout modules and two 4-input arbiter and merge modules are put along the routing channels. They can also be clustered to form modules with more inputs by pro-

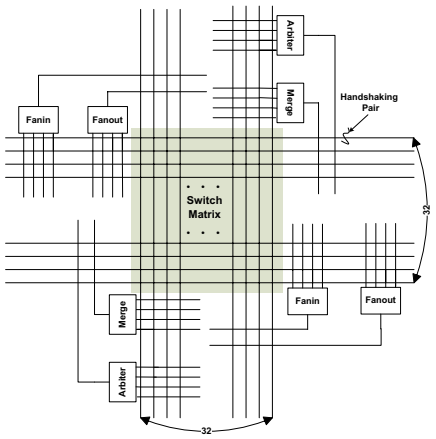

Fig. 6. The CRSB structure.

gramming the connections. The LRSB can adopt the routing switch box design of the conventional FPGA architecture.

4. EVALUATING THE GAPLA ARCHITECTURE

4.1. Area Estimation

We have constructed the layout for some of the basic building blocks of the GAPLA architecture including the I/O port controller, the fanin/fanout module and the arbiter etc in a $0.18\mu m$ process to estimate the area cost of the proposed architecture. The GAPLA adopts the local clock generator for GALS system proposed in [5] which occupies the area of approximately $2M\lambda^2$. The area of each I/O port controller is around $15K\lambda^2$ which is considerably small. The 16 I/O port controllers, the 128 I/O data registers and the distribution matrix for the enable signals of the I/O data registers add up to around $6.5M\lambda^2$. Most of this areas are occupied by the distribution matrix. Thus, we estimate an asynchronous wrapper could occupy the area of around $8.5M\lambda^2$ which is less than the area of a Virtex II CLB. For an asynchronous island which contains 4 asynchronous wrappers and 576 CLBs, the total area cost for the asynchronous wrappers is around $35M\lambda^2$ which is less than 1%. In the routing structure, 320 wires (32 handshaking pairs and 256 data wires) and the RSBs are added between every two adjacent asynchronous islands. In a Virtex II FPGA, there are about 200 routing wires between every two column or row of CLBs and those routing wires together with the routing switch box occupy around 90% of the total FPGA area. In the GAPLA architecture, the extra routing wires are added for every 24 CLB columns or rows. Therefore, the area overhead of the routing resources in GAPLA is estimated as

$$\frac{320}{200 \times 24} \times 0.9 = 6\%$$

Thus, in total, the area cost of adopting a GALS style in GAPLA architecture is estimated to be slightly less than 7%.

4.2. Performance Evaluation

When doing performance evaluation, the interconnect delay between different clock domains are calculated as follows: Communication overhead of the direct interconnections between adjacent asynchronous wrappers is one local clock cycle since the direct interconnects are short and fast and therefore won't cause the clock signal to stretch. The wires of the global routing channels have the delay of 2ns for the extension of each asynchronous island based on the interconnect wire delay of the Virtex II FPGA. Therefore, for two asynchronous wrappers that are one routing switch box away to communicate with each other, the data will reach the receiver in 2ns after it is sent out and the sender will get the acknowledge signal in 4ns. If this interconnect delay is less than one local clock cycle, then the communication overhead is one local clock cycle, otherwise, the local clock is stretched and the communication overhead is determined by the interconnect delay. We have developed a systematic CAD flow for the GAPLA architecture [13], and the following are some experimental results using the CAD tools.

4.2.1. Circuits using both local and long interconnections

We implement several synthetic circuit examples generated from a modified graph generator TGFF [12] program. The program generates directed acyclic graphs (DAGs) and the arcs between nodes in the graph are generated randomly. Therefore, when synthesized into a FPGA, both local and long interconnections are required. We randomly assign each node in the generated graphs an arithmetic or logic operation, and then the DAGs are synthesized with the priority set to performance. The results are given in Table 1 and Table 2.

The results show that significant performance improvement is achieved by adopting the GAPLA architecture when implementing large designs like *ex3*. When implementing the relatively smaller designs like *ex1*, the GAPLA architecture shows a comparable performance to its synchronous counterparts.

4.2.2. Systolic circuits

Systolic circuits primarily use local connections. For this type of circuits, the GAPLA architecture is expected to give comparable performance to conventional FPGA architectures. As an example, we implement a fully pipelined DES (data encryption standard) algorithm. The DES algorithm contains two basic modules which can run in parallel: the data encryption module which contains 16 encryption rounds and the subkey computation module. The subkey computation

Table 1. Implementation results on the Virtex II FPGA

	Nodes	Mults	Clock (ns)	Worst wire delay(ns)	Area (CLBs)	perf. (ns)
ex1	100	26	7.19	4.62	691	194
ex2	576	102	12.96	6.24	2421	933
ex3	1160	216	16.55	9.279	5007	2383

Table 2. Implementation results on the GAPLA architecture

	Part.	Avg. clk	Worst intra-clk domain wire dly	Area	perf. (ns)	Perf.Improv. (%)
ex1	2	6.23	4.06	960	184	5.1
ex2	8	6.14	4.72	3840	517	44.6
ex3	16	6.37	4.91	7680	1083	54.55

module only needs to run once for one key. The data encryption module is partitioned into 3 parts. The throughput of the implementation is 213M/s and the pipeline latency is 84.51 ns. To compare, we also implement this algorithm into the Virtex II FPGA. The pipeline throughput and latency of the Virtex II implementation are 208M/s and 75.02ns. The pipeline latency of the GAPLA implementation is higher due to the asynchronous communication overhead.

5. CONCLUSIONS

This paper proposes the GAPLA, a globally asynchronous locally synchronous FGPA architecture. The GAPLA architecture is based on our asynchronous wrapper design which supports 2-phase handshaking communication protocol with bundled-data delay assumption. The logic size and shape of each clock domain in the GAPLA architecture are programmable and the data width of each asynchronous bundled-data communication channel is also programmable. These features give more flexibility to the GAPLA architecture. The GAPLA architecture is suitable to implement designs with multiple modules working under different clock frequencies like the system-on-a-chip design. It also shows significant performance advantages when implementing large designs where the long interconnect wire delay dominates other delays. The area overhead of adopting a GALS design style in the GAPLA architecture is considerably small.

6. REFERENCES

[1] William Tsu, Andre Dehon, et al. High-speed, hierarchical synchronous reconfigurable array. In *Proc. Int. Symp. Field Programmable Gate Arrays*, 1999.

[2] Akshay Sharma, Katherine Compton, et al. Exploration of pipelined FPGA interconnect structures. In *Proc. Int. Symp. Field Programmable Gate Arrays*, 2004.

[3] Jason Cong, Y. Fan, et al. Architecture and synthesis for multi-cycle communications. In *Proc. Int. Symp. Physical Design*, Apr. 2003.

[4] David Bormann, Peter Y.K. Cheung. Asynchronous wrapper for heterogeneous systems. In *Proc. ICCD* 1997.

[5] S.W.Moore, G.S.Taylor P.A.Cunningham, R.D.Mullins, P.Robinson. Self calibrating clocks for globally asynchronous locally synchronous systems. In *Proc. ICCD* 2000.

[6] J.Muttersbach, T.Villiger, W.Fichtner. Practical design of globally asynchronous locally synchronous systems. In *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems* 2000.

[7] Robert Payne. Self-timed FPGA systems. In *Int. Workshop Field Programmable Logic and Applications* 1995.

[8] R. Konishi, I. Hideyuki, et al. PCA-1: a fully asynchronous self-reconfigurable LSI. In *Int. Symp. Asynchronous Circuits and Systems* Mar. 2001.

[9] Andrew Royal, Peter Cheung. Golbally asynchronous locally synchronous FPGA architectures. In *Int. Workshop Field Programmable Logic and Applications* 2003.

[10] T. Olsson, P. Nilsson, T. Meincke, A. Hemani, M. Torkelson. A digitally controlled low-power clock multiplier for globally asynchronous locally synchronous designs. In *Proc. IEEE Int. Symp. Circuits and Systems* May 2000.

[11] Robert Payne. Self-timed field programmable gate array architecture. Ph.D Thesis, University of Edinburgh, 1997.

[12] R.P.Dick, D.L.Rhodes, W.Wolf. TGFF:task graphs for free. In *Proc. Int. Workshop Hardware/Software Codesign* Mar. 1998.

[13] Xin Jia, Ranga Vemuri. The CAD tools for a globally asynchronous locally synchronous FPGA architectures. *submitted to ICCD'05 for review*.