Studying a GALS FPGA Architecture Using a Parameterized Automatic Design Flow

Xin Jia*, Ranga Vemuri
University of Cincinnati
2600 Clifton Ave.
Cincinnati, OH, 45221
{jiax, ranga}@ececs.uc.edu

ABSTRACT

Routing delays dominate other delays in current FPGA designs. We have proposed a novel Globally Asynchronous Locally Synchronous (GALS) FPGA architecture called the GAPLA to deal with this problem. In the GAPLA architecture, The FPGA area is divided into locally synchronous blocks and the communications between them are through asynchronous I/O interfaces. An automatic design flow is developed for the GAPLA architecture. Starting from behavioral description, a design is partitioned into smaller modules and fit to GAPLA synchronous blocks. The asynchronous communications between modules are then sythesized. The CAD flow is parameterized in modeling the GAPLA architecture. By manipulating the parameters, we could study different factors of the designed GAPLA architecture. Our experimental results show an average of 20% performance improvement could be achieved by the GAPLA architecture.

1. INTRODUCTION

Routing delays have become a major roadblock for FPGA performance and the situation will only be worse when technology continues to scale and FPGA chips continue to grow large. Long routings not only increase the wire delay itself, but also need to go through more routing switch boxes, making the situation worse. For example, the Xilinx VirtexII xc2v8000 FPGA has a corner-to-corner interconnect delay of around 15ns [1]. Different approaches of solving this problem have been proposed. [2] and [3] pipelines the long interconnect delay and [1] proposes a synthesis flow synthesis flow to allow the long interconnect to run for several clock cycles. In those approaches, interconnects are treated as circuit components instead of conventional wires. The interconnect retiming registers can be very expensive in area which make their FPGA size several times bigger than conventional FPGAs.

*The author is currently with Mentor Graphics Co.

Using asynchronous design is another possible solution. Asynchronous design provides average-case performance. In terms of interconnect delays, performance is dictated by the average of the interconnect delays rather than the one with worst delay. Hence the use of long routings does not necessarily lead to a significant performance penalty. We designed the GAPLA: a novel Globally Asynchronous Locally Synchronous Programmable Logic Array architecture. GALS systems can be seen as synchronous logic blocks wrapped in asynchronous I/O interfaces. Interconnects inside each block is short and fast, which allows the synchronous logics to run at higher speed. Interconnects between synchronous logic blocks have longer delay, but they will not affect the clock speed of the logic blocks and only come into picture when there are communications between synchronous blocks. Therefore, performance improvement could be expected.

An automatic design flow for the GAPLA architecture has also been developed [5]. Starting from a behavioral circuit description, a design is first partitioned into smaller modules where each module can fit into one synchronous block (also called *asynchronous island*). Then the control sequences for asynchronous communications between modules are generated and put into each module. After that, a coarse-grained placer is used to place the modules to the GAPLA chip space and connections between islands are routed. Each module is then synthesized by calling existing FPGA tools. The CAD flow is designed as an auxiliary tool to study the GAPLA architecture. It is parameterized in modeling the architecture. Therefore, by changing the values of the architectural parameters, we could study the effect of these factors. In this paper, we report the results of our study of the GAPLA architecture using the above CAD flow.

2. RELATED WORK

Several asynchronous FPGA architectures have been proposed in the last decade [6, 7, 8, 9, 11]. Several of these architectures adopt the GALS concept. STACC [7] is loosely 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 micropipeline like structure. PCA [8] is a self-reconfigurable programmable logic architecture consisting of a layer of logic array and a layer of built-in-facilities. Data communications between logic blocks is realized by a wormhole message passing mechanism through the built-in-facilities which can be expensive in time. Royal et al proposes another GALS FPGA architecture [11] and the idea of using GALS architecture to limit the impact of long

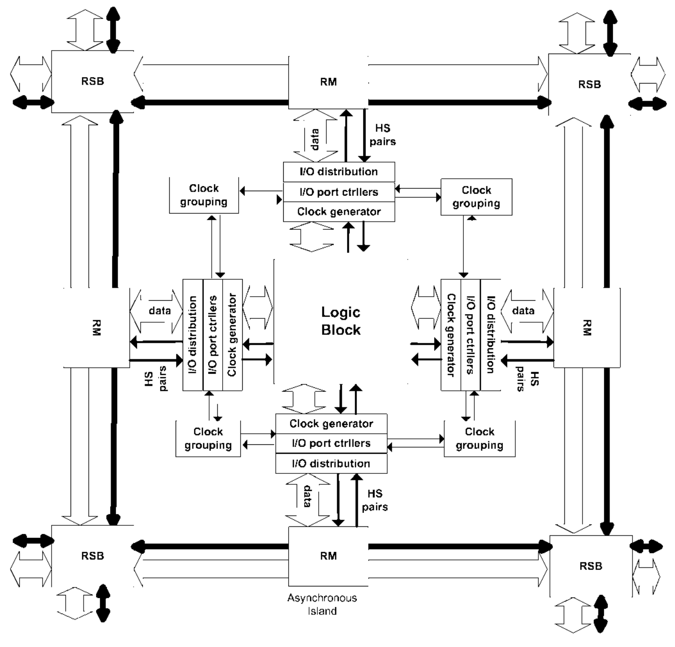

Figure 1: Block diagram of GAPLA architecture.

interconnect wire delay on the total FPGA performance. But to our knowledge, no CAD tools have been proposed for those FPGA architectures. In [13], an automatic synthesis flow is proposed for the highly-pipelined asynchronous FPGA of [9], which is a different asynchronous design style to GALS. In [12], an automatic methodology to produce GALS system is proposed. But the main focus of [12] is to automatically generated the GALS system from a higher level circuit description, while in our research, the GALS interface is a built-in feature of the GAPLA FPGA. Our research focuses on finding the optimal values for a set of architectural parameters.

3. THE GAPLA ARCHITECTURE

Figure 1 gives a basic building tile of the GAPLA architecture called an *asynchronous islands*. The GAPLA architecture is a mesh of asynchronous islands. Each island contains a synchronous logic block and 4 *asynchronous wrappers*. Each wrapper contains a local clock generator and I/O port controllers. The structure of the synchronous logic block can be any of the conventional FPGA structures. But the size of each synchronous logic block must be big enough to implement reasonable functions. In our design, we adopt the Virtex II logic array structure. The 4 clock signals generated by the clock generators are all distributed into the synchronous logic block. Logic tiles inside the synchronous block can freely choose to connect to one of these clock signals. The logics controlled by the same clock signal are called a *clock domain*. Thus, the size and shape of each clock domain of the GAPLA architecture is programmable within the limit of a synchronous logic block. 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. Please refer to [4] for details of the architecture design.

The execution time of an application mapped on the GAPLA FPGA architecture consists of two parts: computation time and communication time. Computation time is the time for synchronous logic blocks to finish the programmed computations. Communication time is the time consumed by the asynchronous communications between logic blocks. The best performance of the GAPLA architecture is the best tradeoff between communication time and computation time. The architectural parameters which affect this tradeoff are as follows.

1. The size of a synchronous logic block. A large logic block means more operations can be put into one clock domain, which generally decreases the local clock speed and increases the overall computation time. But the communication time will decrease since more communications will be done synchronous inside a clock domain.

2. The number of asynchronous I/O ports for each asynchronous island. Increasing the number of I/Os will increase the area overhead but will lessen the I/O constraints during the application partitioning process which could improve the logic usage of the logic block and benefit the system performance. Since our design of the I/O port controllers are very simple and has small layout areas, we can afford to add more I/O ports as long as it will benefit the system performance.

3. The number of global routing channels. This factor not only affect the routability of GAPLA architecture and also affect the performance since the routing might be congested and need to detour if the routing resource is limited which increases communication time.

4. CAD FLOW

The CAD flow is developed to automatically implement designs to be the GAPLA FPGA. It is also used to investigate the architecture design. As mentioned above, performance of the GAPLA is affected by three groups of parameters. The CAD flow models the architecture based on them too. By given these parameters different values, we could compare the implementation results of a set of benchmarks and get to know the effect of these parameters.

Figure 2 shows a block diagram of the overall CAD design flow. The grey boxes are modules we proposed and white boxes are modules using existing CAD tools. Partitioning is required first if the design is bigger than a given asynchronous island. Controls for asynchronous communications are then added to each module to ensure functional correctness. After that, each module is synthesized and place-and-routed using existing FPGA design tools. Also, all the modules are fed into a coarse-grained placer and router which places each module to the GAPLA chip space and finishes the global routing between modules. Finally, a simulation model of the design implemented on the GAPLA architecture is formed for performance evaluation. If all the performance constraints are met, the design is accomplished. In the following subsections, we briefly explain each functional module in the following.

4.1 Partitioning

When partitioning a design into modules, we try to minimize the communication time between partitioned modules. Also, partitioning is conducted under two constraints: the area constraint where the area of each module must be less

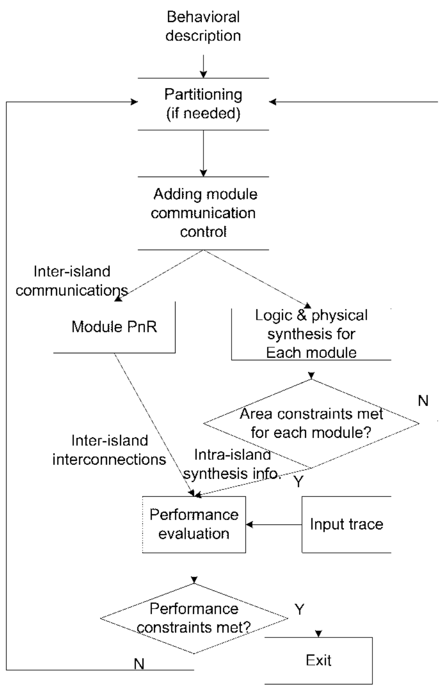
Behavioral description
Partitioning (if needed)
Adding module communication control
Inter-island communications
Module PnR
Logic & physical synthesis for Each module
Area constraints met for each module?
N
Inter-island interconnections
Intra-island synthesis info.
Y
Performance evaluation
Input trace
Performance constraints met?
Y
N
Exit

Figure 2: Block diagram of CAD flow for GAPLA.

than the given area of a synchronous block; the I/O constraint where the number of input/output ports must be less than the give number of input/output ports per asynchronous island.

To calculate the asynchronous communication time, we first build a CDFG representation of the design and edges in the CDFG are given communication weights. The communication weight of an edge consists of two parts: its "communication frequency" and its "length". The communication frequency of an edge is defined as:

$$f(e) = \begin{cases} 1 & initial\, value \\ f(e)/_{m} & if\, e\, is\, inside\, a\, branch,\ m\, is\, the \\ & number\, of\, branches \\ f(e)\times n & if\, e\, is\, inside\, a\, loop,\ n\, is\, the\, loop\, count \end{cases}$$

The length of an edge $l(e)$ is defined as follows: First, we do an As-Soon-As-Possible scheduling to the CDFG, and the control step assigned to node i is denoted as $cs(i)$. Then:

$$l(e_{ij}) = cs(i) - cs(j)$$

$l(e)$ is used to localize all the interconnects. An edge which spans more control steps may have better chance of mapped to a long interconnects. Therefore, it should be more likely to mapped to asynchronous communication channel. The final weight of an edge for the partitioning is a weighted combination of factors $f(e)$ and $l(e)$:

$$w(e) = \alpha * \frac{f(e)}{\max_e(f(e))} + \beta * \frac{\min_e(l(e))}{l(e)}$$

α and β are user defined coefficient and $\alpha+\beta = 1$. $max(f(e))$ is the maximum communication frequency of all edges, $min(l(e))$ is minimum length of all edges.

The GAPLA architecture allows multiple I/O ports of the same clock domain to be active at the same time. In this case, the time overhead for these asynchronous communications overlaps, which leads to performance benefits. The partitioning algorithm should take advantage of this and partitions the system in a way that the overlap among asynchronous communications is maximized. But because partitioning is done before the actual system timing information is obtained, an estimation method is required. We use the factor N, the number of control steps where data transmissions across partitions are required, to represent this.

Thus, the overall cost function of the partitioning algorithm is formed as:

$$F = N \times \sum w(e_{ij}),$$
$$where\ node\ i,\ j\ belong\ to\ different\ partitions.$$

A simulated annealing algorithm is used as the partitioning algorithm. After one iteration of partitioning, each partition is synthesized (logic synthesis without doing placement and routing) separately. The partitions that meets the area and I/O constraints are treated as an individual partition in the final result. The partitions that still violate the area and I/O constraints are further partitioned using the above algorithm.

4.2 Asynchronous Communication Control

To add asynchronous communications, we need to provide a proper sequence of control values for the control signals of the corresponding asynchronous I/O port controllers. The asynchronous handshaking process is automatically managed by the built-in asynchronous FSMs inside the I/O port controllers based on these signals. Therefore, we need to know at what cycle an asynchronous communication should take place. To gather this information, operations inside each module are scheduled first.

We use an As-Soon-As-Possible (ASAP) algorithm to schedule each module in order to get the best performance. Because of the architecture design, the inter-module asynchronous communications block both the sender's and receiver's operations. Thus, *deadlock* situation could occur after scheduling. Deadlock occurs when two or more communicating processes waiting for each other's data in order to continue executing. It can be solved by constructing Communication Dependency Graph (CDG) [14]. A CDG contains all the communication nodes of the system. And a directed arch between two communication nodes in a CDG iff there is a sequential dependency between the two nodes in any of the processes of the system. The dependencies of communication nodes in the constructed CDG are enforced in all the processes of the system by adding dummy control edges to the processes. After adding these edges, an ASAP scheduling algorithm is used to schedule each process and deadlock is avoided.

The outputs of the scheduling are cycle-accurate descriptions for each module. After scheduling, we know exactly at what cycle data needs to be sent to or received from other modules. Therefore, the control signals for the asynchronous communications can be added accordingly.

4.3 Module Placement and Routing

The placer will place each module to an asynchronous island. The optimization goal during placement is to minimize the total communication cost between modules. Since each communication edge carries a communication weight as explained before, the goal of the placer is thus to:

$$\min(\sum f(e_{ij}) \times D_{ij})$$

where node i, j belongs to different modules. D_{ij} is the distance between the clock domains where node i, j are placed.

Since the affect of global routing on the system performance is greatly reduced, a simple and fast line-search based router [15] is used to route the asynchronous communications to the global routing channels of GAPLA FPGA. The inter-module routing resources are represented by two matrices Horizontal Routing Sources (HRS) and Vertical Routing Sources (VRS). The two matrices can be initialized at run time to model different configurations of the GAPLA architecture. Nets are picked up once at a time from the netlist and routed. For multiple-terminal nets, the two terminals with the longest Manhattan distance are routed first.

4.4 Performance Simulation

Simulation is used to estimate the performance of the design implemented on GAPLA architecture. From the synthesis results of each module, the information about the clock frequency for each module is obtained. From the module placer and router, the information about the placement position of each module and the interconnect delays between modules are obtained. These information together with the cycle-accurate VHDL descriptions of each module is fed into our VHDL simulation model of the GAPLA. The GAPLA simulation model contains the models for the pausible local clock generators, the I/O port controllers, and the inter-island routing channels. The local clock generators are programmed to the corresponding clock frequencies of the modules. The modules are wrapped by the asynchronous interfaces and connected through the routing channels, composing a simulation model of the circuit implementation. Input traces are then read to the model and the performance can be observed.

5. ARCHITECTURE PARAMETERS

5.1 Studying Methodology

To study these parameters, we need to implement a set of benchmarks on different configurations of them. Our benchmark set consists of 12 synthetic benchmarks generated by TGFF [10]. TGFF generates Directed Acyclic Graphs (DAGs) with different number of nodes and connectivity intensities. We then assign each node an arithmetic operation and generate a VHDL description code from each graph as a benchmark. Thus, all the benchmarks are computation intensive ones with few or no control flow which are the cases for most FPGA applications. To exclude the factor of multipliers, only addition and subtraction operations are assigned. Table 1 gives the statistics for the benchmark set and their implementation results on a Virtex II FPGA.

It is time forbidden to study the three parameters at the same time. Therefore, the parameters are determined one by one. The size of a logic block is studied first since it is the single most important parameters for the GAPLA architecture. To do that, the number of I/Os per clock domain and the routing capacities between modules are assumed to be infinite. Thus, the I/O constraints during partitioning process are lifted and the routing process is not necessary since the routing distance can be estimated as the Manhattan distance between terminals. After the size of a logic block is chosen, the number of I/Os per clock domain can be determined by looking into the partitioning results since, as explained before, the number of I/Os per clock domain could be fairly large without incurring huge area overhead. After that, the global routing capacities is determined by running the router on the after-placement benchmarks with the first two parameters fixed on the GAPLA FPGA. The experimental results are explained in the following Subsections.

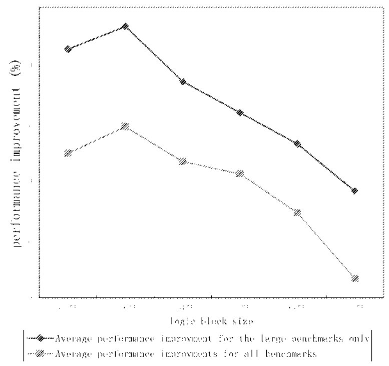

Figure 3: Average Performance improvements on the benchmark set for different logic block sizes.

Table 1: Implementation results on the Virtex II FPGA

	Nodes	Area (CLBs)	Clock (ns)	Worst wire dly(ns)	Exec. Time (ns)
ex1	150	280	7.625	4.537	99.1
ex2	400	760	9.682	6.617	203.32
ex3	500	972	11.7	8.473	386
ex4	500	856	6.51	4.023	266.9
ex5	800	1530	12.98	10.25	467.3
ex6	1000	2000	15.82	12.4	395.5
ex7	1500	2887	17.29	14.47	1072
ex8	1800	3264	14.78	12.12	1257
ex9	2000	3836	16.43	14.19	1610
ex10	2500	4492	17.02	14.20	1634
ex11	3000	5820	20.14	17.15	2296
ex12	4000	7537	19.39	17.05	3491

5.2 Size of a Synchronous Logic Block

For every benchmark, 6 different sizes for a logic block are tried. They are 36, 64, 100, 144, 256, 400 (in terms of CLBs). The performance improvement of all the 12 benchmarks are summarized in Figure 4. The average performance improvement for all 12 benchmarks is given in Figure 3. From the results, the GAPLA FPGA with logic block size 256 CLBs delivers the biggest performance improvement on average for all the 12 benchmarks. Thus, the size of a logic block is chosen to be 256 CLBs. The results also show that GAPLA FPGA could not give sound performance improvement for small applications like ex1 to ex4. If only the last 8 benchmarks are considered, the average performance improvements could be more than 28%. The Average performance improvement for the last 8 benchmarks are also shown in Figure 3.

5.3 I/Os Per Asynchronous Island

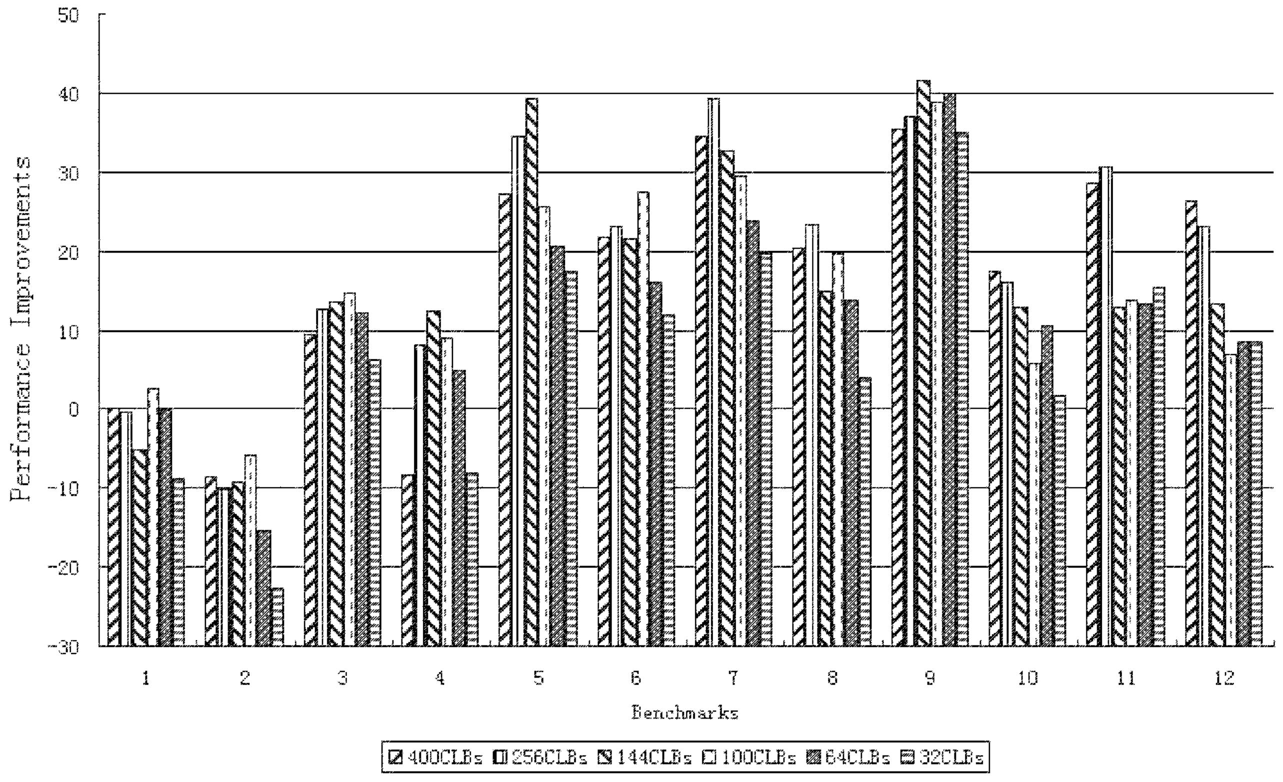

Figure 4: The effect of logic block size on performance of the GAPLA FPGA.

Table 2: The actual I/O requirements with logic size 256 CLBs

case	Max num. in port /partition	Max num. out port /partition	Avg. num. in port	Avg. num. out port
ex1	8	14	6.5	11.62
ex2	30	27	17.49	16.19
ex3	27	26	20.66	19.43
ex4	19	20	12.92	14.77
ex5	23	22	15.16	18.89
ex6	28	31	23.33	19.79
ex7	27	27	18.66	16.05
ex8	35	30	21.17	22.92
ex9	29	31	23.92	20.33
ex10	40	35	24.08	25.01
ex11	32	38	25.33	26.91
ex12	41	40	25.75	23.21
for all cases	41	40	19.58	19.59

As mentioned before, the area of a I/O port controller is relatively small. Therefore, a reasonably large number of I/Os can be integrated. In the last subsection, the size of a logic block is chosen as 256 CLBs and during the experiments, the number of I/Os per clock domain are considered to be always sufficient. The actual I/O requirements for the benchmark set with logic block size 256 CLBs are summarized in Table 2.

For the results, the average number of I/Os required per asynchronous island is around 20 and maximum is around 40. Because there are 4 asynchronous wrappers per island, the number should be divided by 4 to get the I/O requirement per asynchronous wrapper. We tried two configurations: 8 input ports, 8 output ports per wrapper and 10 input ports, 10 output ports per wrapper. In the first case, four benchmark implementations violate the I/O constraints (ex8, ex10, ex11, ex12). In the second case, only one benchmark implementation violates the I/O constraints (ex12). These benchmarks are re-synthesized under the I/O constraints and the results are given in Tabel 3. The performance improvements are compared with the implementation results in GAPLA with logic block size 256 CLBs and without the I/O constraints.

Table 3: Resynthesis results under I/O constraints for benchmarks violating these constraints

8 input ports and 8 output ports per wrapper				
case	part.	commu. cost	exec. time	Perf. Impv.
ex8	14	366	983.4	-2.16
ex10	20	761.35	1408	-2.55
ex11	24	801.5	1611	-1.13
ex12	37	1841	2882	-7.42
10 input ports and 10 output ports per wrapper				
ex12	31	1554	2719	-1.34

From the results, there are very small differences between the two configurations. Therefore, we choose the first configuration, i.e., 8 input ports and 8 output ports per asynchronous wrapper. As for the data wires, we give an aver-

Table 4: Performance evaluation after routing

	16	18	20	24	28	32
	P.I.	P.I.	P.I.	P.I.	P.I.	P.I.
ex1	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
ex2	-10	-10	-10	-10	-10	-10
ex3	12.6	12.6	12.6	12.6	12.6	12.6
ex4	8.1	8.1	8.1	8.1	8.1	8.1
ex5	34.4	34.4	34.4	34.4	34.4	34.4
ex6	21.5	21.5	21.5	21.5	21.5	21.5
ex7	39.2	39.2	39.2	39.2	39.2	39.2
ex8	21.0	21.0	21.0	21.0	21.0	21.0
ex9	-	-	33.5	33.8	37.1	37.1
ex10	-	11.3	11.6	12.6	13.8	13.8
ex11	26.6	27.5	29.3	29.3	29.3	29.3
ex12	-	-	13.0	15.6	16.5	16.5
Avg.	-	-	17.8	18.1	18.6	18.6

age of 16 data wires per communication channel as in our benchmarks. Therefore, the total number of data wires per asynchronous wrapper comes to 256.

5.4 Routings Between Asynchronous Islands

Previous experiments assume that the global asynchronous routing channels are sufficient and therefore each net can use the shorted connection route. In this Subsection, we study the impact of asynchronous routing channels on the performance of the GAPLA FPGA architecture. The first two sets of architectural parameters are fixed, namely 256CLBs per logic block and each asynchronous wrapper contains 8 input ports and 8 output ports and 256 data wires. After routing, the asynchronous communication time is more accurate based on the actual routing path. The experiments are conducted for different routing configurations (in terms of number of asynchronous communication channels). The experimental results are given in Table 4. In the table, "P.I" represents the performance improvements compared to the synchronous implementation on a Virtex II FPGA. The entries marked with "-" mean that the routing is incomplete under the corresponding configuration.

The results show that if the global asynchronous routing structure has less than 20 channels, some of the benchmarks can not be successfully routed. After that, all the benchmarks can be routed. And increasing the number of asynchronous tracks only has slight impact on the system performance (less than 1 percent on average). And increasing the number of global asynchronous routing channels will greatly increase the area overhead of the GAPLA architecture, therefore, we choose 20 channels for the global routing structure. We also assume that, on average, each asynchronous channel has 16 bits of data wires. Therefore, the total number of global data wires is 320.

5.5 Area Overhead

Haven chosen the parameters, we estimated the area overhead of the GAPLA architecture by implementing the building components of the architecture in silicon. The area overhead is estimated to be at 19.9%. (Detailed estimation is not shown due to page limitation.)

6. CONCLUSIONS

In this paper, we studied three sets of critical architectural parameters of the GAPLA FGPA, namely the size of each synchronous logic block, the number of I/Os per asynchronous island, and the number of routing channels between island, using the parameterized CAD tools. From the experimental results, the following values are chosen for these parameters: 256CLBs per logic block, 8 input ports, 8 output ports, 256 data wires per asynchronous wrapper, and 20 global routing channels and 320 global data wires. The area overhead of the GAPLA architecture using this configuration is around 19.9%. The average performance improvement for all the benchmarks is 17.8%. If only the large benchmarks, which are suitable for the GAPLA FPGA, are considered, the average performance improvement on the last 8 benchmarks is 25.4%.

7. REFERENCES

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

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

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

[4] Xin Jia, Ranga Vemuri. A novel asynchronous FPGA architecture design and its performanc evaluation. . *Proc. Int. Workshop Field Programmable Logic and Applications*, 2005.

[5] Xin Jia, Ranga Vemuri. CAD tools for a globally asynchronous locally synchronous FPGA architecture. . *Proc. 19th Int. Conf. VLSI Design, India*, 2006.

[6] S. Hauck, S. Burns, G.Borriello, C. Ebeling. A FPGA for implementing asynchronous circuits. In *IEEE Design & Test of Computers* Vol. 11, No. 3, Fall 1994.

[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] John Teifel, Rajit Manohar. Highly pipelined asynchronous FPGAs. In *Proc. Int.Symp. Field Programmable Gate Arrays* Feb. 2004.

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

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

[12] A. Girault, C. Menier. Automatic production of globally asynchronous locally synchronous systems. In *Proc. EMSOFT*, 2002.

[13] Song Peng, et al. Automated synthesis for asynchronous FPGAs. In *Proc. FCCM*, 2005.

[14] D. Filo, D. C. Ku, N. Coelho, and G. Micheli. Interface optimization for concurrent systems under timing constraints. *IEEE Trans. VLSI*, 1(3), Sept. 1993.

[15] D. Hightower. A solution to line-routing problem on the continous plane. . *Proc. Design Automation Conf.*, 1969.