Architecture and Synthesis for Multi-Cycle Communication

Jason Cong, Yiping Fan, Xun Yang, Zhiru Zhang
Computer Science Department
University of California, Los Angeles
Los Angeles CA 90095 USA
{cong, fanyp, yangxun, zhiruz}@cs.ucla.edu

ABSTRACT

For multi-gigahertz designs in nanometer technologies, data transfers on global interconnects take multiple clock cycles. In this paper, we propose a *regular distributed register* (RDR) micro-architecture for multi-cycle on-chip communication. An RDR architecture structurally consists of a two-dimensional array of islands, each of which contains a cluster of computational logic and local register files. We also propose a new synthesis methodology based on the RDR architecture. Novel layout-driven architectural synthesis algorithms have been developed for RDR. Application of these algorithms to several real-life benchmarks demonstrates 44% improvement on average in terms of the clock period and 37% improvement on average in terms of the final latency.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS – Design Aids

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

RDR, multi-cycle communication, deep sub-micron, timing closure, scheduling, binding, placement, interconnect

1. INTRODUCTION

There are two important inflection points in the development of deep sub-micron (DSM) process technologies. One is when the average interconnect delay exceeds the gate delay, which happened during mid 1990's and led to the timing closure problem. The other is when we cannot reach every part of the chip in a single clock cycle, which is happening now. It has been shown in [1] that, even with the aggressive interconnect optimization techniques (such as buffer insertion and wire-sizing), 7 clock cycles are still needed to go from corner-to-corner for the predicted die-size in the 0.07μm technology generation, assuming a 5GHz clock, based on NTRS'97 [2]. Although the exact clock cycles may vary given the recent update of the roadmap [3], this still clearly suggests that multi-cycle on-chip communication is a necessity in multi-gigahertz synchronous designs. However, given the fact that most existing design tools only deal with the first problem but completely lack consideration of multi-cycle communication, further system performance increase is at risk.

To address the multi-cycle communication problem, one can explore the following design methodologies:

1. Asynchronous design: The state transitions of an asynchronous design are triggered by events instead of periodic clocks. This makes asynchronous designs operate correctly, regardless of the delays on gates and wires [4]. However, due to the lack of design tools and performance overhead, it only applies to a very limited class of circuits. In general, it is unclear whether asynchronous designs can yield high performance in practice.
2. Global asynchronous locally synchronous (GALS) design [5]: In a GALS design, all major modules are designed in accordance with proven synchronous clocking disciplines. Each module is run from its own local clock. Data exchange between any two modules strictly follows a full handshake protocol. GALS hopes to combine the advantages of synchronous and asynchronous design methodologies. However, the overhead for the "self-timed wrapper" may compromise both performance and area of the design.
3. Synchronous design with multi-cycle communication: Synchronous design is still by far the most popular design methodology. It is well understood and supported by the mature CAD toolset. However, the ever increasing gap between gate delays and interconnect delays requires the handling of multi-cycle communication that cannot be overcome by the traditional synchronous design flow. This paper will focus on the synchronous designs and propose a way to systematically handle multi-cycle communication.

In the context of the synchronous designs, several layout-driven synthesis approaches have been proposed in recent literature to alleviate the problem introduced by the long interconnect. At the gate-level, simultaneous retiming and placement or floorplanning is performed in [6][7] to optimize the register-to-register delay (i.e., the clock period). Unfortunately, exploring multi-cycle communication during logic synthesis has a big limitation. As discussed in [8], the minimum clock period that can be achieved by logic optimization is bounded by the maximum delay-to-register (DR) ratio of the loops in the circuits. This requires the consideration of multi-cycle communication during architectural & behavioral synthesis. [10] proposed an architectural synthesis approach which incorporates a performance-driven placement to guide the post-layout scheduling. The target architecture they proposed is the distributed-register architecture which helps to explicitly separate the long interconnect delays from logic delays. Under this architecture, [11] performed an integrated resource sharing and placement to eliminate the slack time violation due to the interconnect delays. Note that the irregular structure used by both [10] and [11] may cause difficulty for interconnect delay estimation. Regular circuit and layout structures [12] can be employed to avoid this problem. Generally, regular structure facilitates predictability and simplifies the implementation process.

In this paper, we present a new synthesis methodology for synchronous designs with multi-cycle communication. Our contributions are as follows (i) we propose a *regular distributed register* (RDR) micro-architecture which offers high regularity and direct support of multi-cycle communication; (ii) we propose synthesis methodology and develop novel architectural synthesis algorithms which efficiently synthesize behavior-level input onto the RDR architecture.

The remainder of the paper is organized as follows. Section 2 introduces the RDR architecture. Section 3 sketches our synthesis methodology and the algorithms for RDR architecture. The experimental results are shown in Section 4, followed by the conclusions and future work in Section 5.

2. REGULAR DISTRIBUTED REGISTER ARCHITECTURE

In this section, we propose a regular distributed register (RDR) micro-architecture. It provides high regularity and direct support of multi-cycle communication over global interconnects.

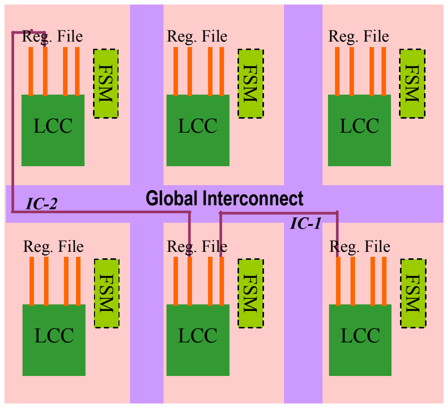


Figure 1. A 2×3 island-based RDR architecture

An RDR architecture consists of a two-dimensional array of islands. The size of each island is chosen such that intra-island computation and communication can be done in a single clock cycle. In other words, the data obtained from a local register can be processed by a certain functional unit, and then be stored to a local register within only one cycle.

Figure 1 illustrates a 2×3 island-based RDR architecture. Figure 2 details the structure of a single island, which consists of the following components:

1. *Local computational cluster (LCC)*: It contains the computational elements of an island such as adders, multipliers and dividers, etc. The size of the LCC is subject to certain area constraints and a target clock period.
2. *Register file*: The dedicated local storages resides in the register file. The registers are partitioned into k banks for 1 cycle, 2 cycle, … k cycle interconnect communication, under the assumption that we need up to k cycles to cross the chip. For example, Figure 1 shows a short interconnect labeled as *IC-1* and a long interconnect labeled as *IC-2*. *IC-1* takes only 1 cycle and *IC-2* takes 2 cycles. Therefore, the register that drives *IC-2* needs to hold its value for 2 clock cycles when we transfer the data over *IC-2*. On the other hand, the register that drives *IC-1* only needs to hold 1 cycle for the communication over *IC-1*.
3. *Finite state Machine (FSM)*: The controller for the computational elements is implemented as an FSM.

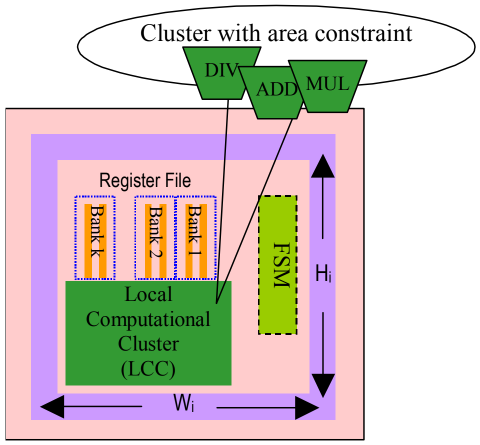

Cluster with area constraint
DIV
ADD
MUL
Register File
Bank k
Bank 2
Bank 1
FSM
Local Computational Cluster (LCC)
Hi
Wi

Figure 2. Components of a single island

As discussed in [10], one of the advantages of distributed register architecture over centralized register architecture is that it can achieve a short clock period and effectively reduce the overall performance degradation due to the interconnect delay.

Since we distribute registers to each island, the delays of long wires do not lengthen the clock period. The potential drawback of this approach is that it may demand extra communication cycles for inter-island data transfer. Fortunately, this can be properly harnessed by a smart coarse placement to hide as many critical data transfers as possible. The regularity from RDR architecture ensures the placement a meaningful delay estimation on interconnects.

The RDR architecture has the added advantage that by varying the size of the basic island, we can target at different clock periods and systematically explore the cycle time vs. latency tradeoff.

Given a target clock period, the following formula shows how to compute the geometrical dimension of a basic island:

$$D_{intra\text{-}island} \le D_{logic} + 2 \times D_{opt\text{-}int}\,(W_i + H_i) \le T_{clk}$$

where T_{clk} is the target clock period, D_{logic} is the largest logic delay, $D_{opt\text{-}int}(x)$ is a function which estimates the interconnect delay over a certain distance x, W_i is the island width, H_i is the island height, and $D_{intra\text{-}island}$ is the average intra-island delay. The average intra-island delay should be no greater than the largest logic delay D_{logic} plus the worst-case interconnect delay, which approximates to $2{\times}D_{opt\text{-}int}(W_i{+}H_i)$ (i.e., the estimated interconnect delay over a corner-to-corner round trip within an island).

Figure 3 shows an RDR architecture with a 12×12 island-based array for a 5GHz design in 70nm technology by 2008 [2]. We assume a chip dimension of 620 mm^2 (24.9mm x 24.9mm) in which the signal of a wire can travel up to 7.52mm within 1 clock cycle under interconnect optimization. We need a total of 7 clock cycles to cross the chip. Based on the above formula, we can derive the base dimension of each island $W_i=H_i=2.08mm$.

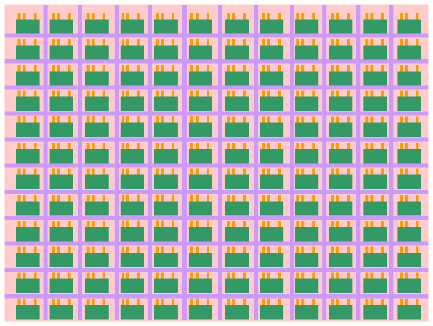


Figure 3. Example: An RDR architecture for 70nm technology

3. PLACEMENT-DRIVEN ARCHITECTURAL SYNTHESIS USING RDR ARCHITECTURE

In this section, we present our architectural synthesis system for RDR architecture, named MCAS. We will first introduce the overall design flow in Section 3.1, followed by a motivational example in Section 3.2. Then we will present the key modules of the MCAS system, including the scheduling-driven placement, the placement-driven simultaneous rescheduling and rebinding, and the datapath & FSM generation.

3.1 Overall Design Flow

Figure 4 shows the overall synthesis flow of the MCAS system. MCAS starts with a synthesizable behavioral C or VHDL description. RDR architecture specification is needed (including the island structure, functional unit library and delay table). The target clock period is also given and used in the followed synthesis steps. If the final design cannot meet the clock period requirement, we can adjust the island size of the RDR architecture and perform another iteration by binary search of clock period.

We first generate the control data flow graph (CDFG) from the behavioral descriptions. In the next step, we obtain the resource allocation from a force-directed scheduling algorithm [13] using the critical path length as the timing constraint. Then we perform an initial functional unit binding and derive an interconnected component graph from the bound CDFG.

After that, the interconnected component graph is fed to the scheduling-driven placement to provide location information (i.e., island index) of each functional unit. The scheduling-driven placement algorithm will be discussed in Section 3.3. Based on the physical information, we perform simultaneous rescheduling and rebinding on the CDFG. This algorithm will be presented in Section 3.4. At the backend, all of the scheduling and binding information is back-annotated to the CDFG and fed to the datapath & FSM generation module. A datapath in structural VHDL format and controllers in behavioral FSM style are generated. This module will be discussed in Section 3.5

The synthesis system finally generates RT-level VHDL files for logic synthesis and outputs floorplan constraints and multi-cycle path constraints for placement & routing.

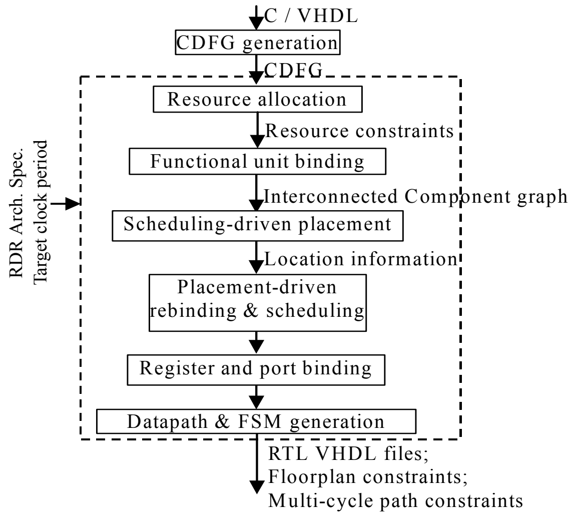


Figure 4. MCAS architectural synthesis system

3.2 A Motivational Example

In this subsection, we use a motivational example to illustrate the advantage of using multi-cycle communication and the need for the consideration of multi-cycle communication during architectural synthesis.

Figure 5 is a data flow graph (DFG) extracted from a discrete cosine transform (DCT) algorithm [14]. In this DFG, nodes 1, 2, 5, 6, 9 and 10 are addition or subtraction operations, and nodes 3, 4, 7, 8, 11 and 12 are multiplication operations. In this example, we assume that the delay of a multiplication operation is 2 ns and that of an addition or a subtraction operation is 1 ns.

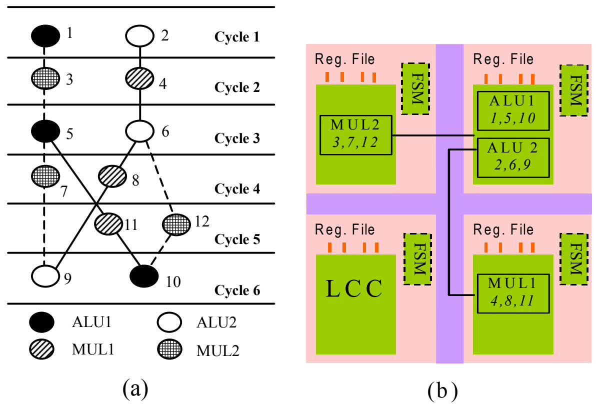


Figure 5. (a) Schedule and binding without consideration of interconnect delays; (b) Layout of wirelength-driven placement

In the traditional architectural synthesis approaches, interconnect delay is assumed to be negligible compared with the functional unit delay, which is not realistic anymore in DSM era. Without considerations of interconnect delays, the DFG is scheduled in 6 clock cycles with an estimated clock period of 2 ns. The total schedule latency is 12 ns. Two multipliers and two ALUs are allocated. The nodes in the same pattern are bound to the same functional unit.

However, interconnect may introduce extra delays on the DFG edges after place & route. Figure 5 (b) shows the layout produced by a wirelength-driven placement. Each box represents a functional unit, and the numbers inside the box denote the DFG nodes bound to the functional unit. The horizontal wires represent short interconnects with a delay of 1 ns. The vertical wires represent long interconnects with a delay of 2 ns. The interconnect delays are back-annotated to the DFG edges. On the DFG edges in Figure 5 (a), a solid line represents a long interconnect delay, and a dash line represents a short interconnect delay. The introduction of interconnect delay has lengthened the actual clock period to 4 ns, resulting in 24 ns of the final schedule latency.

Observe that in Figure 5, the interconnect delay has significantly compromised the final latency. We can minimize the negative impact of interconnect delay by using our RDR architecture to allow multi-cycle communication. Figure 6 shows the rescheduled result based on fixed placement and binding under the assumption that interconnect delays can be more than one clock cycle. The resulting clock period is 2 ns. Although the cycle number increases to 9 clock cycles, the total schedule latency is reduced to 18 ns. Note that in the figure, a short line on a dash edge indicates to merge a 1 ns interconnect delay to a 1 ns operation.

The following subsections will demonstrate that the latency can be further reduced if we consider the multi-cycle communication during scheduling and binding, which are two crucial steps of architectural synthesis.

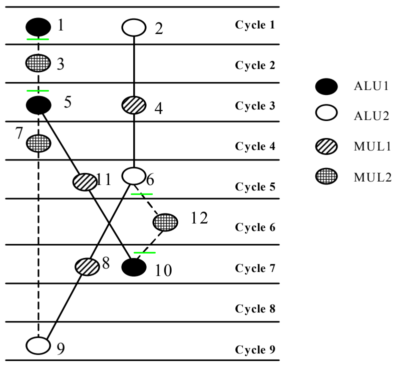


Figure 6. Schedule with multi-cycle interconnect delay

3.3 Scheduling-Driven Placement

In previous work, rescheduling based on fixed binding and placement is used to reduce scheduling latency [10]. However, the effect of scheduling on placement has been rarely studied.

In Figure 6, we have seen that long interconnect delays are on the critical path of DFG. A pure wirelength-driven placement may produce a poor solution with long critical path. To address this problem, we propose a scheduling-driven placement algorithm, in which scheduling guides the placement to find a placement solution with a minimal total schedule latency.

Using the same example from Figure 6, Figure 7 shows that by applying a scheduling-driven placement with the critical path awareness, the DFG can be scheduled in 8 clock cycles and the total schedule latency can be reduced 16 ns.

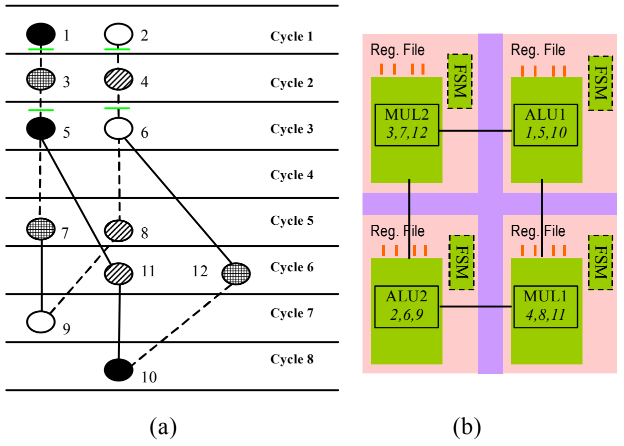


Figure 7. (a) Schedule of scheduling-driven placement; (b) Layout of scheduling-driven placement

Our scheduling-driven placement is formulated as follows. The inputs to the placer are the following: (1) *Target clock period*: T_{clk} (2) *Original CDFG*: $G=(N, E)$. (3) *Interconnected Component graph*: $G^*=(N^*, E^*)$, which is derived from the bound CDFG. Nodes in N^* represent the functional units to which operation nodes in G are bound such as ALUs, multipliers, dividers, etc. Edges in E^* represent the data transfers between these nodes. These edges are annotated with a delay $D(e)$ corresponding to the physical delay between the functional units. The goal is to place the nodes of N^* so that the total schedule latency of G is minimized.

We integrate scheduling with an SA-based coarse placement algorithm [15]. A fast list scheduling is performed on G instead of the classical timing analysis at every temperature during the SA process to identify critical edges in E^*, and assign higher weights to them. By reducing the weighted wirelength, we try to hide as many critical data transfers into intra-island communication as possible, and make the uncritical data transfers go through the inter-island, multi-cycle communication over global interconnect.

Initially, we define the bin structure of the coarse placement to be the given island structure. The criticalities of the corresponding nets are obtained and converted to weight on the nets at each temperature during the SA process (once scheduling-based timing analysis is performed). Our net weighting method is similar to [16]. The criticality of an edge is defined to be

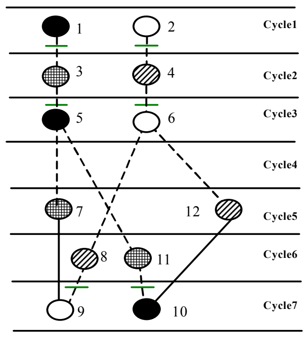
$$crit(e)=1-slack(e)/L$$

where L is the schedule latency and $slack(e)$ is the edge slack produced by the list-scheduling algorithm.

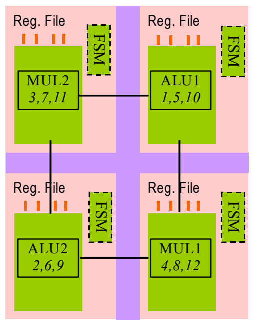
After the placement, the functional units that are placed in the same bin will be clustered into the LCC of the corresponding island.

3.4 Placement-Driven Simultaneous Rescheduling and Rebinding

In [17], functional unit binding is performed simultaneously with a floorplanning to estimate the quality of the floorplan. In [18], floorplanning is used to estimate layout after scheduling and allocation. The limitation of both [17] and [18] is that they only optimize the clock period without performing rescheduling to reduce the clock cycle number.



(a)

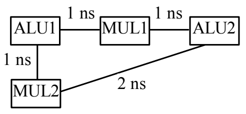


(b)

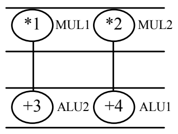
Figure 8. (a) Schedule of placement-driven simultaneous scheduling and binding; (b) Layout of placement-driven simultaneous scheduling and binding

From Figure 8 (a), it can be seen that the schedule latency can be further reduced to 14 ns if we apply simultaneous rescheduling and rebinding based on the given placement of Figure 8 (b).

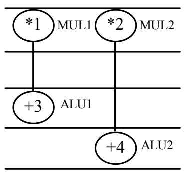
A concurrent scheduling and binding algorithm based on a given floorplan is proposed in [9]. It uses the concept of dynamic critical path list-scheduling (CPLS) introduced by [19]. The algorithm schedules the ready operations in descending order of the critical path length, and simultaneously binds the operations to functional units in such a way that the binding incurs the least increase of total schedule latency. However, this algorithm does not consider potential resource competition during scheduling and may produce suboptimal solution. Figure 9 illustrates this limitation.



(a)



(b)



(c)

Figure 9. Algorithm in [9] may result in a bad solution (a) The interconnected component graph with edge delay information; (b) A good scheduling and binding solution; (c) A bad scheduling and binding solution

Figure 9 (a) shows an interconnected component graph where functional units are represented by the rectangular nodes, and the edges are associated with interconnect delays. Figure 9 (b) and (c) shows two different scheduling and binding solutions of a simple DFG with only 4 nodes. The functional unit binding is shown beside the DFG node. According to the algorithm in [9], both nodes 3 and 4 will be ready and compete for ALU1 in clock cycle 3. Since they have the same priority (i.e., critical path length), either one of them may be chosen and be bound to ALU1. If node 3 is scheduled first, the DFG will be scheduled in 3 clock cycles. However, if node 4 is scheduled first and bound to ALU1, we will end up with a DFG scheduled in 4 clock cycles.

To overcome this problem, we propose a new algorithm based on a force-directed list-scheduling framework [13]. It integrates with simultaneous rebinding, and tries to minimize the schedule latency with consideration of interconnect delays.

The first step of our algorithm is to defer the node selection. The node with the least force is deferred. The critical path length (CPL) and the earliest start time (EST) are used as the secondary and tertiary priority functions to break ties. The nodes are deferred one-by-one until enough functional units are available. In the second step, the remaining ready nodes are scheduled and bound in decreasing order of CPL, and EST is used to break ties. It is possible that some nodes cannot be scheduled to the earliest clock cycle due to resource competitions. In the third step, if there are spare resources available, the previously deferred nodes will be explored and scheduled to the current clock cycle in the reverse order of deferral. After that, the algorithm will proceed to next iteration until all nodes are scheduled and bound.

3.5 Datapath & FSM Generation

After the previous phases, the binding and scheduling information is back-annotated to the CDFG's edges and nodes. The backend of our architectural synthesis system will extract this information to construct datapath and controllers. The datapath, including instances of functional units, registers and steering logic, is generated as a structural VHDL file. This step also generates floorplan information and multi-cycle constraints for RDR synthesis flows. The floorplan information is used to constrain the placement location for every instance in the datapath. The multi-cycle constraints correspond to the multi-cycle communication paths between registers, and are used to guide the physical design tools to optimize the clock period.

In each island, an FSM controller is generated to control the instances inside the island. These distributed controllers of different islands have identical state transition diagrams, but different output signals. The VHDL files for the datapath and the controllers, the floorplan and multi-cycle paths constraints, are fed into the logic synthesis and physical design tools to produce the final design layout.

4. EXPERIMENTAL RESULTS

We implemented our MCAS system in C++/UNIX environments. To obtain the final performance results, Altera's Quartus II version 2.2 [20] is used to implement the datapath part into a real FPGA device, Stratix™ EP1S40F1508C5. All of the pipelined multipliers are implemented into the dedicated DSP blocks in the Stratix™ device. We set the target clock frequency at 200 MHz and use the default compilation options. We impose LogicLock™ to constrain every instance into its corresponding island, and set multi-cycle path constraints for multi-cycle communication paths.

For comparison, we also set up two alternative flows. Figure 10 shows the three flows labeled as 1, 2 and 3. Flow 3 is our MCAS flow discussed in Section 3.1. The simplest flow (flow 1 in Figure 10) uses the traditional scheduling algorithm based on fixed binding information. Similar to flow 3, flow 2 is also based on the RDR architecture and the location information provided by the scheduling-driven placement. However, flow 2 only performs scheduling for the given binding instead of simultaneous rebinding and scheduling in flow 3. The same list-scheduling algorithm is applied for all three flows. These three scheduling flows are converged in later synthesis phases.

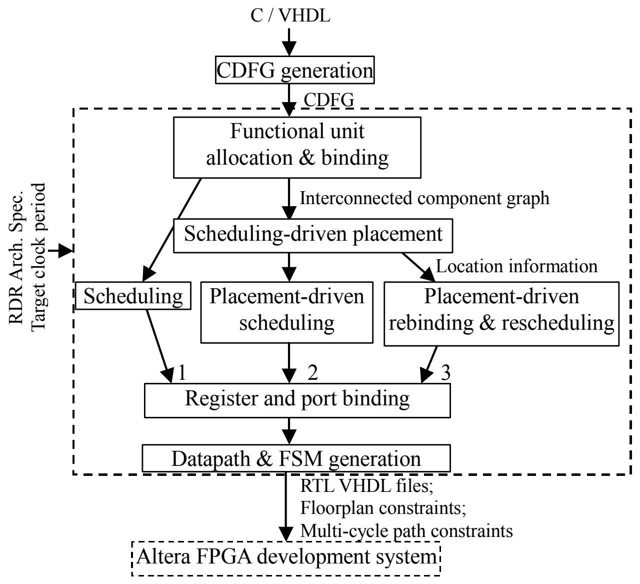

C / VHDL
CDFG generation
CDFG
Functional unit allocation & binding
Interconnected component graph
Scheduling-driven placement
Location information
RDR Arch. Spec.
Target clock period
Scheduling
Placement-driven scheduling
Placement-driven rebinding & rescheduling
1
2
3
Register and port binding
Datapath & FSM generation
RTL VHDL files;
Floorplan constraints;
Multi-cycle path constraints
Altera FPGA development system

Figure 10. Three experimental flows

We have tested the three different flows for a set of real-life benchmarks, which include several different DCT algorithms, such as Planar Rotation (PR), WANG, LEE and DIR, and several DSP programs such as MCM, HONDA, CHEM, and U5ML12. All of the benchmarks are from [21]. In the experiments, we applied 7×4 RDR architecture for small designs (PR, WANG, LEE, DIR, MCM, and HONDA), and 4×4 architecture for CHEM and U5ML12.

Table 1. Functional unit and register binding results

	Node#	ALU#	MULT#	Register#		
				F 1	F 2	F 3
PR	46	6	2	34	38	35
WANG	52	5	8	35	46	46
LEE	53	8	4	36	40	41
MCM	98	6	3	35	53	50
HONDA	101	6	8	42	55	56
DIR	152	7	4	61	68	66
CHEM.	351	13	11	69	103	101
U5ML12	551	18	13	89	153	131
Ave Ratio	-	-	-	1.00	1.34	1.28

Table 1 shows the binding results, which are from the functional unit binding and the register binding. The second column lists the node numbers of the CDFG examples. ALU and MULT are the numbers of the corresponding functional unit usages after the initial binding. Although the three flows generate the same functional unit usages, flow 3 has different binding results due to the rebinding process. The next three columns are register usage numbers from the different flows. On average for this set of benchmarks, flows 2 and 3 use 34% and 28% more registers than flow 1 respectively. Flow 3, which has permuted the functional unit binding, results in a smaller register usage than flow 2.

In Table 2, we list the control step numbers (CS), clock periods (CP) reported by QuartusII, and total latencies (Lat, the product of CS and CP) produced by the three flows. Considering the interconnect delay, flows 2 and 3 introduce more cycles for the communication between registers. Compared with flow 1, flows 2 and 3 produce 14% more cycles. However, since flows 2 and 3 separate the communications from the computations and even apply multi-cycle path constraints for communications, the individual paths in the final layout are reduced, resulting in much smaller clock periods (more than a 40% reduction).

We also illustrate the total latencies in Figure 11, where the three bars in every group represent the results from flows 1, 2 and 3 respectively. Compared to the traditional flow, our architectural synthesis based on RDR approaches (flows 2 and 3) reduces the final latencies of the designs by 35% and 37% respectively. It can also be seen that flow 3 has better latency than flow 2. It proves our conviction that scheduling-driven placement can reduce schedule latency and simultaneous rescheduling and rebinding can further improve design performance.

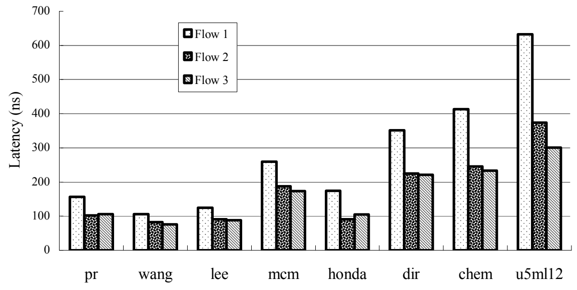


Figure 11. Total latency comparison for the three flows

Table 3 lists the resources used by different design flows in terms of LUT and register. It can be seen that flows 2 and 3 introduce less than 20% LUT overhead, but more than 100% registers as overhead. Since our RDR architecture uses more registers than the traditional approach, the register usage is increased. The increased register number also increases the complexity of the steering logic structure, such as multiplexors, which then contributes an observable portion of the area in the final layout, especially for an FPGA design.

Table 2. Cycle number, clock period, and overall latency comparison for the three flows

	Flow 1			Flow 2			Flow 3		
	CS	CP (ns)	Lat (ns)	CS	CP (ns)	Lat (ns)	CS	CP (ns)	Lat (ns)
PR	27	5.79	156.33	29	3.53	102.37	29	3.66	106.14
WANG	14	7.54	105.56	20	4.14	82.80	20	3.81	76.20
LEE	20	6.25	125.00	27	3.36	90.72	26	3.38	87.88
MCM	34	7.64	259.76	39	4.81	187.59	38	4.57	173.66
HONDA	23	7.58	174.34	24	3.78	90.72	24	4.18	100.32
DIR	50	7.03	351.50	51	4.41	224.91	51	4.33	220.83
CHEM	50	8.27	413.50	53	4.64	245.92	52	4.49	233.48
U5ML12	68	9.30	632.40	70	5.34	373.80	70	4.30	301.00
Ave Ratio	1.00	1.00	1.00	1.14	0.57	0.65	1.13	0.56	0.63

Table 3. LUT and register usage comparison

	LUT			Register		
	F 1	F 2	F 3	F 1	F 2	F 3
PR	787	867	891	436	870	804
WANG	945	1089	931	272	769	793
LEE	709	738	793	223	700	653
MCM	1959	2055	1983	735	1167	1095
HONDA	1076	1292	1460	554	1010	1034
DIR	1913	2351	2018	956	1536	1433
CHEM	3597	4720	4901	933	2155	2107
U5ML12	5676	7953	7786	1358	3278	2750
Ave Ratio	1.00	1.19	1.17	1.00	2.21	2.10

5. CONCLUSIONS & FUTURE WORK

We have proposed a novel RDR architecture to support multi-cycle on-chip communication in multi-gigahertz designs. Compared with several existing methodologies, the regularity of RDR architecture facilitates the predictability of interconnect delays at the higher design levels. An architectural synthesis system using the RDR architecture has been developed. The experimental results on Altera's Startix™ device have demonstrated the effectiveness of our proposed architecture, design methodology, and synthesis algorithms.

In the future, we will extend our architectural synthesis system to support control-intensive applications. Many problems, such as variable renaming and allocation, distributed controller generation, etc., will be further studied. In addition, we have observed that the steering logic has a great impact on the performance and area of the final layout, and we will consider optimizing them in the future synthesis flow.

6. ACKNOWLEDGEMENTS

This research is partially supported by National Science Foundation under award CCR-0096383, MARCO/DARPA Gigascale Silicon Research Center (GSRC), and Altera Corporation under the California MICRO program. The authors would like to thank Professor Miodrag Potkonjak of UCLA for his contributions of the benchmarks, and Professor Kiyoung Choi of Seoul National University, Korea for providing the CDFG packages.

7. REFERENCES

[1] J. Cong, "Timing Closure Based on Physical Hierarchy," *in Proceedings of 2002 International Symposium on Physical Design*, pp. 170-174, 2002.

[2] Semiconductor Industry Association, The National Technology Roadmap for Semiconductors, 1997.

[3] Semiconductor Industry Association, International Technology Roadmap for Semiconductors, 2001.

[4] J. T. Udding, "A Formal Model for Defining and Classifying Delay-Insensitive Circuits," *Distributed Computing*, vol. 1(4), pp. 197-204, 1986.

[5] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," *PhD thesis*, Stanford University, Oct. 1984.

[6] J. Cong and S. K. Lim, "Physical Planning with Retiming," *in Proceedings of International Conference on Computer Aided Design*, pp. 2-7, 2000.

[7] D. P. Singh and S. D. Brown, "Integrated Retiming and Placement for Field Programmable Gate Arrays," *in Proceedings of International Symposium on Field Programmable Gate Arrays*, pp. 67-76, 2002.

[8] J. Cong and C. Wu, "FPGA Synthesis with Retiming and Pipelining for Clock Period Minimization of Sequential Circuits," *in Proceedings of the 34th ACM/IEEE Design Automation Conference*, pp. 644-649, 1997.

[9] J. Jeon, D. Kim, D. Shin and K. Choi, "High-level Synthesis under Multi-Cycle Interconnect Delay," *in Proceedings of Asia and South Pacific Design Automation Conference*, pp. 662-667, 2001.

[10] D. Kim, J. Jung, S. Lee, J. Jeon and K. Choi, "Behavior-to-Placed RTL Synthesis with Performance-Driven Placement," *in Proceedings of International Conference on Computer Aided Design*, pp. 320-326, 2001.

[11] J. Um, J. Kim and T. Kim, "Layout-Driven Resource Sharing in High-Level Synthesis," *in Proceedings of International Conference on Computer Aided Design*, pp. 614-618, 2002.

[12] F. Mo and R. K. Brayton, "Regular Fabrics in Deep Sub-Micron Integrated-Circuit Design," *11th IEEE/ACM International Workshop on Logic & Synthesis*, pp. 7-12, 2002.

[13] P. Paulin and J. Knight, "Force-Directed Scheduling for Behavioral Synthesis of ASICs," *in IEEE Trans. on CAD*, vol. 8(6), pp. 661-679, June 1989.

[14] W. H. Chen, C. Smith and S. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," in *IEEE Trans. on Communications*, vol. 25(9), pp. 1004-1009, Sept. 1977.

[15] C. C. Chang, J. Cong, Z. Pan and X. Yuan, "Physical Hierarchy Generation with Routing Congestion Control," *in Proceedings of 2002 International Symposium on Physical Design*, pp. 36-41, 2002.

[16] A. Marquardt, V. Betz and J. Rose, "Timing-driven placement for FPGAs," *in Proceedings of International Symposium on Field Programmable Gate Arrays,* pp. 203-213, 2000.

[17] Y. M. Fang and D. F. Wong, "Simultaneous Functional-Unit Binding and Floorplanning," *in Proceedings of International Conference on Computer Aided Design*, pp. 317-321, 1994.

[18] M. Xu and F. J. Kurdahi, "Layout-Driven RTL Binding Techniques for High-Level Synthesis," *in Proceedings of 9th International Symposium on System Synthesis*, pp. 33-38, 1996.

[19] Y. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *in IEEE Trans. on Parallel and Distributed Systems*, vol. 7(5), pp. 506 -521, May 1996.

[20] Altera Web Site, http://www.altera.com/.

[21] M. B. Srivastava and M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," in *IEEE Trans. on VLSI Systems*, vol. 3(1), pp. 2-19, March 1995.