

# Architecture and Synthesis for On-Chip Multicycle Communication

Jason Cong, *Fellow, IEEE*, Yiping Fan, *Student Member, IEEE*, Guoling Han, *Student Member, IEEE*, Xun Yang, and Zhiru Zhang, *Student Member, IEEE*

**Abstract**—For multigigahertz designs in nanometer technologies, data transfers on global interconnects take multiple clock cycles. In this paper, we propose a regular distributed register (RDR) microarchitecture, which offers high regularity and direct support of multicycle on-chip communication. The RDR microarchitecture divides the entire chip into an array of islands so that all local computation and communication within an island can be performed in a single clock cycle. Each island contains a cluster of computational elements, local registers, and a local controller. On top of the RDR microarchitecture, novel layout-driven architectural synthesis algorithms have been developed for multicycle communication, including scheduling-driven placement, placement-driven simultaneous scheduling with rebinding, and distributed control generation, etc. The experimentation on a number of real-life examples demonstrates promising results. For data flow intensive examples, we obtain a 44% improvement on average in terms of the clock period and a 37% improvement on average in terms of the final latency, over the traditional flow. For designs with control flow, our approach achieves a 28% clock-period reduction and a 23% latency reduction on average.

**Index Terms**—Binding, high-level synthesis, interconnect, multicycle communication, placement, scheduling.

## I. INTRODUCTION

NANOMETER process technologies allow billions of transistors on a single die, running at multiple-gigahertz frequencies. The shrinking cycle time combined with the growing resistance–capacitance delay, die size, and average interconnect length contribute to the increasing role of the interconnect delay (especially the global interconnect delay), which does not scale well with the feature size. The graph shown in Fig. 1, taken from the 2001 SIA roadmap (ITRS'01) [33], plots this trend. The gap between the wire and the gate performance will continue to grow even with the use of new interconnect materials and aggressive interconnect optimization. As a result, the delays on wires that span the chip will exceed the clock period, and the single-cycle full chip communication will no longer be possible.

Manuscript received May 31, 2003; revised September 28, 2003 and December 5, 2003. This work was supported in part by MARCO/DARPA Gigascale Silicon Research Center (GSRC), in part by the National Science Foundation under Award CCR-0096383, in part by the Semiconductor Research Center under 2001-TJ-910, and in part by the Altera Corporation under the California MICRO program. This paper was recommended by Guest Editor P. Groeneveld.

J. Cong, Y. Fan, G. Han, and Z. Zhang are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; fanyp@cs.ucla.edu; leohgl@cs.ucla.edu; zhiruz@cs.ucla.edu).

X. Yang was with the Computer Science Department, University of California, Los Angeles, CA 90095 USA. He is now with Silvaco Data Systems, Santa Clara, CA 95054 USA (e-mail: xun.yang@silvaco.com).

Digital Object Identifier 10.1109/TCAD.2004.825872

Fig. 2<sup>1</sup> shows that even with the optimal buffer insertion and wire-sizing, five clock cycles are still needed to go from corner-to-corner for the predicted die of 28.3 × 28.3 mm in the 70-nm technology generation, assuming a 5.63-GHz clock by 2006 as projected in ITRS'01. This clearly suggests that multicycle on-chip communication is a necessity in multigigahertz synchronous designs. Unfortunately, it is not supported in the current design tools and methodologies, as most of these implicitly assume that full-chip communication in a single clock cycle is feasible.

To address the multicycle 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 [28]. However, due to the lack of design tools and possible area/performance overhead, it is unclear whether asynchronous designs can indeed yield high performance in practice.
- 2) Global asynchronous locally synchronous (GALS) design [1]: 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, but the overhead for the “self-timed wrapper” may compromise both performance and area of the design.
- 3) Synchronous design with multicycle on-chip communication: Synchronous design is still by far the most popular design methodology. It is well understood and supported by the mature computer-aided design toolset. Unfortunately, the traditional synchronous design flow assumes that the clock signal can reach the entire chip in a single clock cycle, which is no longer true for multigigahertz designs in nanometer technologies. This paper will focus on the synchronous designs and propose a way to systematically handle multicycle on-chip communication.

There have been some efforts to address the problem of multicycle on-chip communication for synchronous designs.

<sup>1</sup>This is an update of the similar figure shown in [8], which is based on NTRS'97 [32]. Optimal buffer insertion and wire sizing are performed using the IPEM [7] package with the driver, buffer, and receiver sizes being 100 × the minimum size inverter.

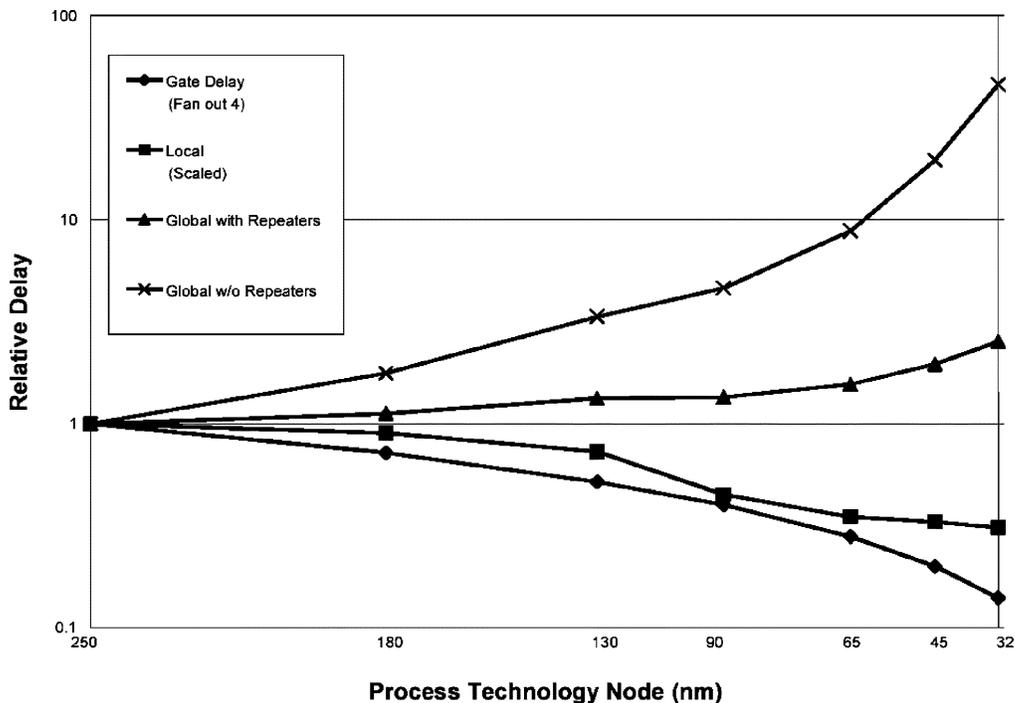


Fig. 1. Delay for local and global wiring versus feature size, from ITRS’01.

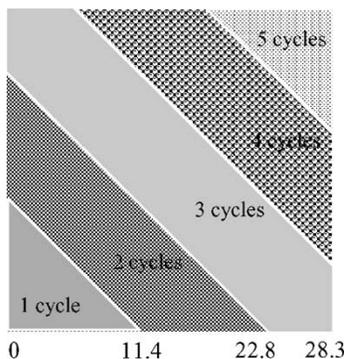
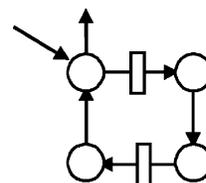


Fig. 2. Illustration of how far global signals can travel to cross the chip.



In a loop, 4 logic cells, 2 registers  
 Cell delay = 1ns  
 Interconnect delay = 2ns

Fig. 3. Limitation in exploring at multicycle communication the logic/physical level.

Most of them are at the gate level to perform retiming with placement or floorplanning [3], [5], [6], [25] to alleviate the performance degradation caused by long interconnects. Although the benefit of applying these methods can be significant, exploring multicycle communication during logic synthesis has a severe limitation, as 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 circuit [4], [21]. Fig. 3 illustrates this problem by showing a piece of circuitry with four logic cells and two registers in a loop. Assuming that cell delay is 1 ns and interconnect delay is 2 ns, the DR ratio of this loop can be calculated by

$$\begin{aligned}
 DR\_ratio &= \frac{(\text{Delay}_{\text{logic}} + \text{Delay}_{\text{interconnect}})}{\#Registers} \\
 &= \frac{(4 \times 1 + 4 \times 2)}{2} = 6 \text{ ns.}
 \end{aligned}$$

Therefore, the best possible clock period is 6 ns under any retiming solution. To further improve the clock speed above

166 MHz, one can pipeline the long wires by inserting flipflops [24]. The gains from this technique can be dramatic as the clock frequencies are no longer restricted by the interconnect speed. However, since flipflops insertion will change the cycle-level behavior of the circuit, it requires a considerable amount of manual rework in the RTL to account for the additional pipeline stages. Even worse, such rework is usually performed in *ad hoc* ways with very limited automated tool support, which seriously compromises the design productivity.

All of the aforementioned difficulties lead us to propose a methodology that addresses the multicycle communication problem at a higher-level abstraction. In this paper, we present a new microarchitecture and a novel architectural synthesis methodology to systematically handle multicycle on-chip communication for synchronous designs. Our contributions are as follows.

- 1) We propose a regular distributed register (RDR) microarchitecture which offers high regularity and direct support of multicycle communication.

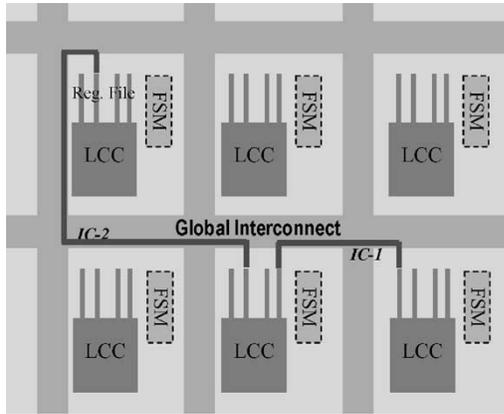


Fig. 4.  $2 \times 3$  island-based RDR microarchitecture.

- 2) We propose a synthesis methodology with novel architectural synthesis algorithms to efficiently synthesize behavior-level input onto the RDR microarchitecture.

In particular, we integrate global placement with scheduling and binding, which are two crucial steps in the high-level synthesis. We also develop a practical solution, such as the distributed control generation to handle the control flow for the RDR microarchitecture. The preliminary results of this paper were published in [9] and [10].

The remainder of the paper is organized as follows. Section II introduces the RDR microarchitecture. Section III describes our synthesis methodology and the algorithms for multicycle communication using the RDR microarchitecture. The experimental results are shown in Section IV, followed by the conclusion and future work in Section V.

## II. RDR MICROARCHITECTURE

In this section, we propose an RDR microarchitecture, which offers high regularity and direct support of multicycle communication.

### A. Structure of the RDR Microarchitecture

The RDR microarchitecture divides the entire chip into an array of islands. The registers are distributed to each island, and the island size is chosen so that all local computation and communication within an island can be performed in a single clock cycle. To support multicycle communication across the chip, the local registers in each island are divided into (up to)  $k$  banks (where  $k$  is the maximum number of cycles needed to communicate across the chip) so that registers in bank  $i$  will hold the results for  $i$  cycles for communicating with another island that is  $i$  cycles away.<sup>2</sup>

Fig. 4 illustrates a  $2 \times 3$  island-based RDR microarchitecture. Fig. 5 details the structure of a single island, which consists of the following components.

- 1) Local computational cluster (LCC): The functional elements in an LCC provide the computational power of the

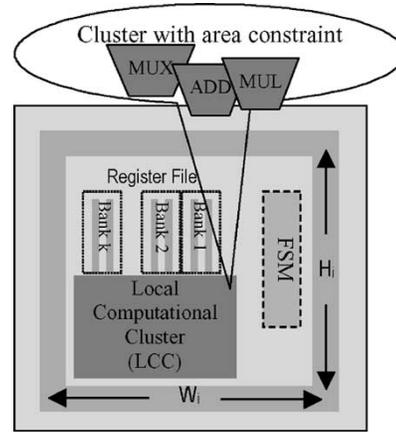


Fig. 5. Components of a single island.

circuit. They can be simple logic such as NAND gates, multiplexors (MUX), or datapath elements such as multipliers, dividers, and ALUs, etc.

- 2) Register file: The dedicated local storages reside in the register file. The registers are partitioned into (up to)  $k$  banks for 1 cycle, 2 cycle,  $\dots$ ,  $k$  cycle interconnect communication, under the assumption that we need up to  $k$  cycles to cross the chip. For example, Fig. 4 shows a short interconnect labeled as  $IC-1$  and a long interconnect labeled as  $IC-2$ . Suppose  $IC-1$  takes only one cycle and  $IC-2$  takes two cycles. Therefore, the register that drives  $IC-2$  needs to hold its value for two clock cycles when the data are transferred over  $IC-2$ . On the other hand, the register that drives  $IC-1$  only needs to hold one cycle for the communication over  $IC-1$ .
- 3) Finite state machine (FSM): Each island contains a local controller (i.e., an FSM) to control the behaviors of the computational elements and registers.

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

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

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

As an example, Fig. 6 shows an RDR microarchitecture with an  $8 \times 8$  island-based array for a 5.63-GHz design in 70-nm technology by 2006 as predicted in ITRS'01 [33]. We assume a chip dimension of  $800 \text{ mm}^2$  ( $28.3 \times 28.3 \text{ mm}$ ) in which the signal of a wire can travel up to 11.4 mm within one clock cycle under interconnect optimization. Five clock cycles are needed to cross the chip. The base dimension of each island ( $W_i = H_i = 3.94 \text{ mm}$ ) can be derived from the above formula. Each island can hold up to 19.63 M min-size two-input NAND gates.

<sup>2</sup>Interconnect pipelining can be performed to optimize the use of global interconnect. We will further discuss this issue in Section V.

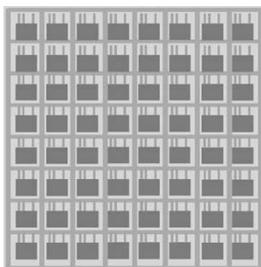


Fig. 6. RDR microarchitecture for 70-nm technology by 2006.

**B. Related Architectures**

Conventional architectures, which depend on a centralized register file and a global control, do not scale well with the large interconnect delays that have become the biggest fraction of the clock period. This motivates the architectures that exploit the physical locality by operating on data near where it is stored. Reference [13] introduced a dynamically-scheduled partitioned architecture called the multicluster architecture. It distributes functional units, dispatches queues, and registers files across multiple clusters to reduce the cycle time despite long wire delays. Reference [11] proposed a hypothetical 2009 multicomputer processor-DRAM chip model. The chip consists of 64 tiles, each of which contains a processor and a memory. The estimated round-trip communication delay in a single tile is 1 ns (two cycles), while the worst-case corner-to-corner delay is 56 cycles.

Based on a similar idea, [14] and [15] proposed a distributed-register architecture in the domain of architectural synthesis. In this architecture, registers are distributed so that each functional unit can perform a computation by reading data from the local dedicated registers and also by writing the result into the local registers. This allows for a very small interconnect delay provided that the functional unit and its dedicated registers are closely located. Data transfers between different functional units are regarded as global communications that may take multiple cycles, which decouple communication and computation. Under this distributed-register architecture, [14] first performs floorplanning to obtain physical information, then does rescheduling and rebinding to reduce the final latency, [15] performs operation binding, placement and postlayout scheduling sequentially, in which the placement is driven by the interclock slack time obtained by an initial scheduling, [29] formulates the placement problem into a linear programming model to eliminate the potential slack time violation, and resource sharing is performed after placement.

Similar to the distributed-register architecture, RDR also distributes registers close to the local computational units to achieve a short clock period, supports multicycle communication, and allows concurrent computation and communication. In addition, the RDR microarchitecture has the following advantages over the distributed-register architecture:<sup>3</sup>

- 1) RDR is highly regular, which greatly facilitates the interconnect delay estimation. Note that all of [14], [15],

<sup>3</sup>Note that these strengths are associated with high-performance designs. The RDR architecture is not suitable (or necessary) for low-frequency (and low-power) designs as due to its unnecessary area (and power) overhead.

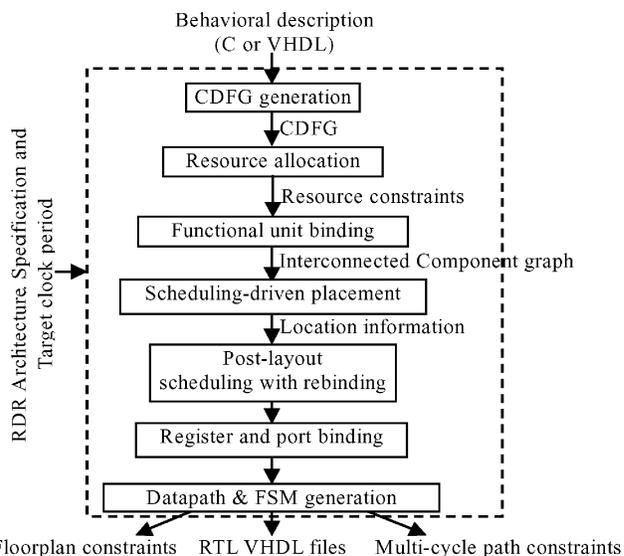


Fig. 7. MCAS architectural synthesis system.

and [29] may have difficulty with the interconnect delay estimation. Although they employ the coarse placement or floorplanning to predict the layout, the estimated interconnect delay may still be far from accurate before the final placement and routing. Generally, regular circuits and layout structures facilitate predictability and simplify the implementation process [20]. In RDR, the delay lookup table for the intra-island and the interisland interconnect can be precomputed once the island structure is specified. Therefore, interconnect delay can be easily estimated from the island indices of the source and the destination.

- 2) The RDR microarchitecture has the added advantage that by varying the size of the basic island, we can target different clock periods and systematically explore the cycle time versus latency tradeoff.

**III. ARCHITECTURAL SYNTHESIS FOR MULTICYCLE COMMUNICATION (MCAS)**

In this section, we present our architectural synthesis system for MCAS, which is built on top of the RDR microarchitecture. We will first introduce the overall design flow of MCAS in Section III-A. In Section III-B, we will show the internal representation of MCAS. The benefit of considering multicycle communication during architectural synthesis is illustrated by a motivational example in Section III-C. Then we will present the key modules of the MCAS system in Sections III-D and III-E, including the scheduling-driven placement, and the placement-driven simultaneous scheduling with rebinding. The distributed control generation algorithm will be discussed in Section III-F.

**A. Overall Design Flow**

Fig. 7 shows the overall synthesis flow of the MCAS system. The inputs of the MCAS system include the following.

- A behavioral-level description. In our case, this can be either synthesizable C or VHDL.

- A target clock period. This is optional because MCAS will try to find the best possible cycle time by binary search if the user does not provide it.
- A resource library, which defines the available functional units, on-chip memories, and registers, etc.
- The island structure and the delay lookup table of the RDR microarchitecture, which can be derived from the aforementioned formula once the target clock period is set.

At the front end, MCAS first generates the control data flow graph (CDFG) from the behavioral descriptions through the intermediate representations of the SUIF compiler infrastructure [37] and Machine-SUIF [26]. Based on the CDFG, MCAS performs resource allocation, followed by an initial functional unit binding. The objective of resource allocation is to minimize the resource usage (e.g., functional units, registers, etc.) without violating the timing constraint. It uses the time-constrained force-directed scheduling algorithm [22] to obtain the resource allocation. After the resource allocation, it employs an algorithm proposed in [15] to bind operation nodes to functional units for minimizing the potential global data transfers. An interconnected component graph (ICG) is derived from the bound CDFG. An ICG consists of a set of components (i.e., functional units) to which operation nodes are bound. They are interconnected by a set of connections that denote data transfers between components.

At the core, MCAS performs the scheduling-driven placement, which takes the ICG as input, places the components in the island structure of the RDR microarchitecture, and returns the island index of each component. After the scheduling-driven placement, both the CDFG schedule and the layout information are produced. To further minimize the schedule latency, it performs placement-driven scheduling with rebinding. The algorithm is based on the force-directed list-scheduling (FDLS) framework, and is integrated with simultaneous rebinding.

At the back end, MCAS performs register and port binding followed by datapath and distributed controller generation. The final outputs of MCAS include:

- A datapath in structural VHDL format and a set of distributed controllers in behavioral FSM style. These RT-level VHDL files will be fed into logic synthesis tools.
- Floorplan and multicycle path constraints for the downstream place-and-route tools.

### B. Internal Representation

A two-level *CDFG* representation is used in MCAS. The first-level CDFG is a Control Flow Graph (CFG). Each node corresponds to a basic block. The edges represent the control dependencies between the basic blocks. Each basic block contains, at most, one operation producing the control signal. If there are two successors, the labels on the control edges indicate which branch is the fall-through path and which one is the taken path.

At the second level, each basic block has a pure data flow graph (DFG) representation, which contains a set of operation nodes and edges that represent data dependencies among operation nodes. Fig. 8 gives an example of a two-level CDFG.

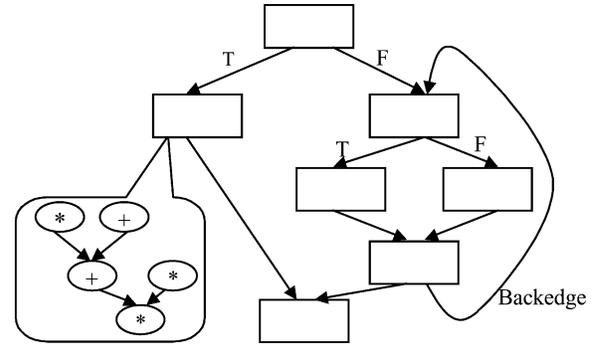


Fig. 8. Two-level CDFG.

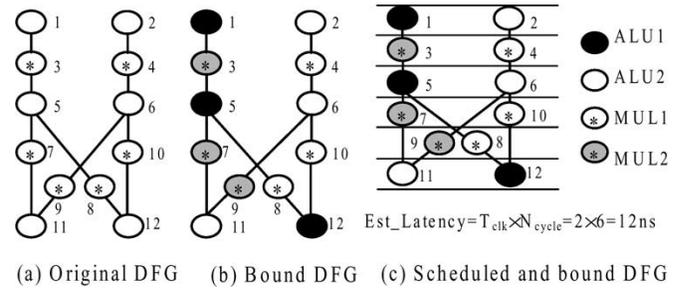


Fig. 9. DFG example.

### C. Motivational Example

In this section, we use a motivational example to illustrate the benefit of considering multicycle communication during architectural synthesis.

Fig. 9(a) shows a DFG extracted from a discrete cosine transform (DCT) algorithm [2]. For the sake of simplicity, we assume a uniform node delay ( $= 2$  ns) for all operations. Fig. 9(b) shows the bound DFG produced by the resource allocation and initial functional unit binding. Two multipliers and two ALUs are allocated. The nodes in the same pattern are bound to the same functional unit. Fig. 9(c) shows a schedule for the bound DFG. Without consideration of interconnect delay, the DFG can be scheduled in six clock cycles with an estimated clock period of 2 ns. The total latency is 12 ns.

In the traditional architectural synthesis approaches, interconnect delay is assumed to be negligible compared with the functional unit delay. However, this is no longer realistic in the deep submicron era. Interconnect may introduce extra delays after placement and routing. Suppose that a wirelength-driven placement algorithm is used and it produces the layout shown in Fig. 10. We use rectangles to represent the functional units. The numbers inside a rectangle denote the DFG nodes that are bound to the corresponding functional unit. The horizontal wires represent short interconnects with a delay of 2 ns. The vertical wires represent long interconnects with a delay of 4 ns.

Given the above layout, Fig. 11 illustrates the advantage of allowing multicycle communication when the interconnect delays are considered. We use a single line to represent a short interconnect delay and double lines to represent a long interconnect delay. In Fig. 11(a), multicycle communication is not allowed, and every communication must be done in a single

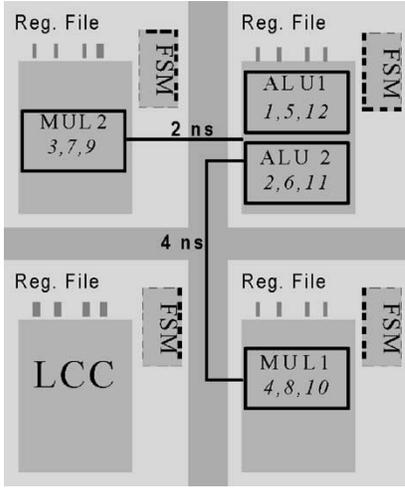


Fig. 10. Layout of a wirelength-driven placement.

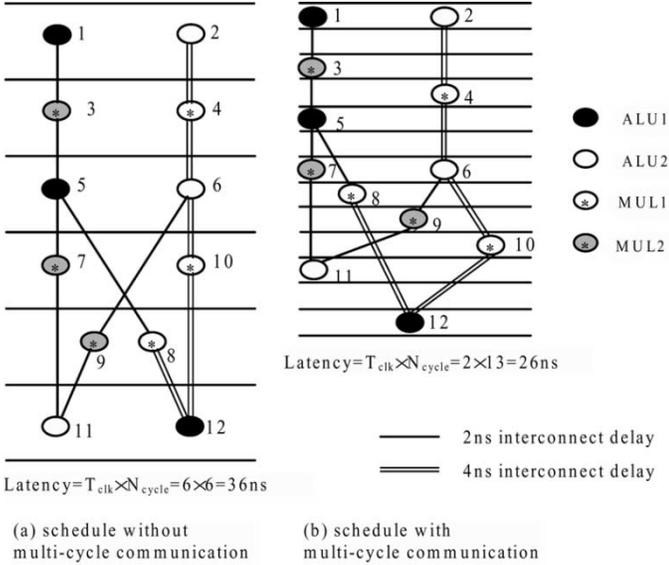


Fig. 11. Advantage of allowing multi-cycle communication.

cycle. Therefore, the final clock period has been significantly lengthened to 6 ns (2 ns logic delay plus 4 ns interconnect delay), resulting in a final schedule latency of 36 ns. In Fig. 11(b), we use RDR microarchitecture to enable multi-cycle communication. After rescheduling the DFG based on the given placement and binding, we can obtain a schedule with a 2-ns clock period. Although the cycle number increases to 13 clock cycles, the total schedule latency is reduced to 26 ns.

The above examples show the necessity for allowing multi-cycle communication during architectural synthesis. In the following sections, we will use the same DFG example to demonstrate that the latency can be further reduced by integrating physical planning with scheduling and binding for multi-cycle communication.

#### D. Scheduling-Driven Placement

Observe that long interconnect delays are on the critical path of the scheduled DFG shown in Fig. 11(b). A pure wirelength-driven placement may produce an inferior solution in terms of

the schedule latency. We can improve solution quality by integrating a high-level synthesis algorithm with the physical planning engine so that the placement or floorplanning algorithm is aware of the critical data transfers and will try to optimize the corresponding interconnects. Several existing works [12], [23], [30] fall into this track but, unfortunately, these approaches do not allow multi-cycle communication. They minimize the interconnect delay within a single clock cycle and, thus, reduce the clock period, but cannot consider multi-cycle communication to overcome long global interconnect delays.

To address this problem, we propose a novel solution called scheduling-driven placement to perform scheduling simultaneously with global placement for multi-cycle communication. The main idea of the scheduling-driven placement is to perform scheduling-based timing analysis on the CDFG to identify the critical edges and then assign a high weight to the corresponding connections (i.e., data transfer edges) in the ICG. A simulated annealing-based coarse placement engine minimizes the weighted wirelength to shorten the critical connections and, thus, potentially reduces the schedule latency.

##### 1) Problem Formulation:

**Definition 1:** A bound CDFG  $G(V, E)$  is a directed graph, where  $V$  is a set of operation nodes and  $E$  is a set of edges which denote the data dependencies. An identifier (ID)  $x(v)$  is assigned to each node in  $v \in V$ , denoting the component to which  $v$  is bound.

**Definition 2:** Given a bound CDFG  $G$ , one can construct its derivative ICG  $G'(V', E')$  as follows, where  $V'$  is a set of components to which operation nodes in  $V$  are bound to and  $E'$  is a set of connections that denote the data transfers between components. A value  $x'(v')$  is assigned to each node in  $v' \in V'$  denoting the ID of this component. Precisely,

- for each node in  $V$  there is a corresponding component in  $V'$ , i.e., there is a surjective mapping function  $\chi: V \rightarrow V'$  s.t.  $\forall v \in V, \exists v' \in V' (x(v) = x'(v'))$ .
- for each edge in  $E$  there is a corresponding connection in  $E'$ , i.e., there is a surjective mapping function  $\psi: E \rightarrow E'$  s.t.  $\forall e(s, t) \in E, \exists e'(s', t') \in E' (x(s) = x'(s') \wedge x(t) = x'(t'))$ .

The placement problem for RDR microarchitecture is formulated as follows.

**Given:** 1) Island structure  $I_x \times I_y$ , which defines the two dimensions of the island-based array. 2) The bound CDFG  $G(V, E)$  and its derivative ICG  $G'(V', E')$ . Note that ICG  $G'$  is the actual input for placement and CDFG  $G$  is only used by the timing analysis engine.

**Goal:** the placer seeks the location (in terms of island index) for each component in  $V'$  to minimize the number of clock cycles needed to schedule  $G$ .

2) *Scheduling-Based Timing Analysis:* Instead of directly operating on the ICG, which is not acyclic in general, we perform the timing analysis on the CDFG. Once the critical edges in CDFG are known, we can easily identify the corresponding critical connections in ICG and assign a high weight to them.

To determine the critical edges, one can apply as soon as possible (ASAP) scheduling and as late as possible (ALAP) scheduling on the CDFG. We compute the criticality of each edge  $e = (s, t)$  by  $\text{edge-slack}(s, t) = \text{ALAP}(s) - \text{ASAP}(t)$ ,

where  $ASAP(s)$  is the ASAP schedule of operation node  $s$  and  $ALAP(t)$  is the ALAP schedule of operation node  $t$ . The edges with zero  $edge\_slack$  are critical. Although this method is simple and intuitive, the edges of ICG can be overly constrained. Since both ASAP and ALAP ignore all resource constraints, the latencies of their schedules may be far shorter than the real ones.

In order to get a more accurate latency estimation, we perform a list-scheduling on the bound CDFG. Specifically, our scheduling algorithm picks the ready operation node with the largest critical path length, i.e., the longest path from the node to the primary outputs (POs), and schedules it to the first feasible control step (i.e., clock cycle). Given the binding and scheduling information, we can obtain the criticality of each edge in the scheduled CDFG by computing the arrival time and required time for each operation node.

We first define the following terms that will be used in the sequel.

- $succ(s)$ : the successors of node  $s$ .
- $pred(t)$ : the predecessors of node  $t$ .
- $cstep(s)$ : the control step to which node  $s$  is scheduled.
- $logic\_delay(s)$ : the logic delay of the component (i.e., functional unit) to which node  $s$  is bound.
- $int\_delay(s, t)$ : the interconnect delay of the connection between the component to which node  $s$  is bound and the one to which node  $t$  is bound.
- $conflict\_delay(t)$ : the number of control steps that  $t$  is delayed due to the resource conflicts, precisely,

$$conflict\_delay(t) = cstep(t) - \max_{s \in pred(t)} \{cstep(s) + int\_delay(s, t)\}.$$

The arrival time of node  $t$ , denoted as  $ARR(t)$ , generally refers to the data ready time when all inputs of  $t$  are available. In this case, however, even though all inputs are available,  $t$  can be deferred due to the resource conflicts. Since our scheduling algorithm always schedules the node  $t$  in the earliest feasible control step, we define the arrival time as  $ARR(t) = cstep(t)$ . The longest path delay of the scheduled CDFG is  $T = \max_{t \in PO} \{ARR(t)\}$ .

The required time of node  $t$ , denoted as  $REQ(t)$ , is the time when node  $t$  must be scheduled. Otherwise, the timing constraint could not be made. We first set the required time of all primary outputs to be  $T$ . The required time is then propagated backward in CDFG with the equation at the bottom of the page.

Required time analysis also accounts for the resource conflict during the computation of  $edge\_delay(s, t)$ . It is defined as:

$$edge\_delay(s, t) = int\_delay(s, t) + conflict\_delay(t).$$

Note that the introduction of  $conflict\_delay$  brings the consideration of resource competitions into the required time analysis.

Then we compute the slack of an edge  $(s, t)$  in CDFG by

$$EDGE\_SLACK(s, t) = REQ(t) - ARR(s) - int\_delay(s, t) - logic\_delay(s).$$

Given a scheduled and bound CDFG  $G(V, E)$ , we have the following conclusion.

*Definition 3:* An edge  $e$  is defined as *critical* if the increment on its  $int\_delay$  by  $\sigma$  will increase the total latency by  $\sigma$  as well, assuming that the same schedule is followed.

*Theorem:*  $\forall e \in E$ ,  $e$  is critical if and only if  $EDGE\_SLACK(e) = 0$ , where  $EDGE\_SLACK(e)$  is computed by the above timing analysis technique.

Since the proof is straightforward, we omit it here.

Finally, the criticality of an edge  $(s, t)$  in CDFG is defined as:

$$EDGE\_CRIT(s, t) = 1 - \frac{SLACK(s, t)}{\max_{e \in E} \{SLACK(e)\}}.$$

*3) Net Weighting:* At each temperature in the SA process, the interconnect delays are extracted from the current layout and back-annotated to the edges in the bound CDFG. Then, we determine the criticality of each edge in the ICG by the scheduling-based timing analysis. The criticalities of the edges in CDFG are obtained and transferred to the weights on the corresponding connections in ICG. We use the weighting method, which similar to that proposed in [18] for cycle time minimization, i.e.,

$$weight(e) = criticality(e')^{exponent}$$

where  $e$  is a connection in ICG,  $e'$  is the corresponding edge in CDFG, and  $exponent$  is a user-defined parameter to heavily weight connections that are critical.

Fig. 10 shows the layout at the current temperature, and Fig. 11(b) shows the schedule generated by the list-scheduling. After the scheduling-based timing analysis, we can identify that the edges (2,4), (4,6), (6,10), and (10,12) have the zero  $EDGE\_SLACK$ , i.e., they are critical. Then, we assign high weight to the connections between ALU2 and MUL1, and the ones between ALU1 and MUL1. At the next temperature, the weighted wirelength minimization engine will try to reduce the delay of these connections.

Fig. 12 shows the expected layout and schedule at the next temperature. By moving the locations of ALU2, we shorten the critical connections between ALU2 and MUL1. We can then reduce the cycle number by 2 and the schedule latency by 4 ns.

*4) SA-Based Coarse Placement Engine:* The details of the SA engine we use are described as follows.

- *Solution space:* We define the bin structure to be the same as the given island structure. Components in ICG are placed at island centers subject to the area constraint.

$$REQ(s) = \begin{cases} T, & s \in PO \\ \min_{t \in succ(s)} \{REQ(t) - edge\_delay(s, t) - logic\_delay(s)\}, & \text{otherwise} \end{cases}.$$

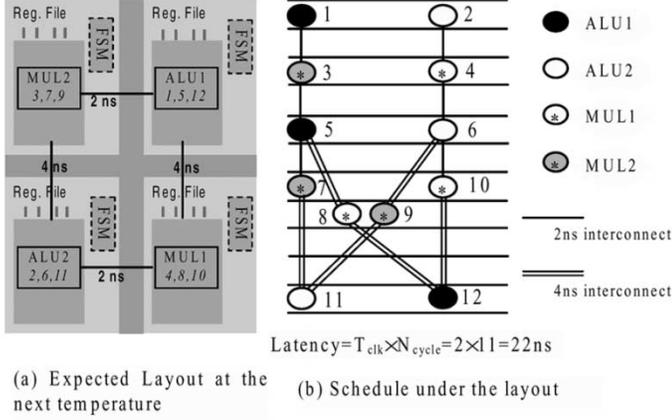


Fig. 12. Advantage of integrating scheduling with placement.

- Neighborhood structure: Two types of moves are used: 1) component move, which randomly selects a component and moves it to another island and 2) component swap, which randomly swaps two components in different islands.
- Cost function: We use the same cost function as proposed in [18]. The delay cost of an edge in ICG is the product of the edge delay and its weight, defined as the following:  $\text{delay\_cost}(e) = \text{delay}(e) \times \text{weight}(e)$ . To obtain  $\text{delay}(e)$ , where  $e$  is the connection between component  $c_s$  and  $c_t$ , we exploit the regularity of the RDR microarchitecture by computing the delay of  $e$  as a function only of the island indices of  $c_s$  and  $c_t$ . To allow an efficient assessment of the interconnect delays, we precompute a delay lookup table indexed by  $s$  and  $t$ . DELAY\_COST is the sum of all the delay costs of all the edges in the ICG. WIRE\_COST is the sum of all the bounding box lengths of the connections. The overall cost is a weighted sum of WIRE\_COST and DELAY\_COST defined as

$$\text{cost} = \alpha \times \frac{\text{DELAY\_COST}^{\text{current}}}{\text{DELAY\_COST}^{\text{previous}}} + (1 - \alpha) \times \frac{\text{WIRE\_COST}^{\text{current}}}{\text{WIRE\_COST}^{\text{previous}}}$$

where  $\alpha$  is a user-defined parameter to tradeoff the wire-length and delay.

### E. Placement-Driven Simultaneous Scheduling With Rebinding

Functional unit binding also has a considerable impact on the performance of the synthesis result. In [12], functional unit binding is integrated with floorplanning to optimize the clock period based on a given schedule. In [31], floorplanning is used to estimate layout after scheduling and allocation. The limitation of both [12] and [31] is that they do not allow multicyle communication.

A concurrent scheduling and binding algorithm based on a given floorplan is proposed in [14]. It uses the concept of dynamic critical path list-scheduling introduced by [16]. 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 the potential resource competition during scheduling and may produce suboptimal solution. Fig. 13 illustrates this limitation by a simple example.

Fig. 13(a) shows an ICG where functional units are represented by the rectangular nodes, and the edges are associated with interconnect delays. Fig. 13(b) and (c) show two different scheduling and binding solutions of a simple DFG with only four nodes. The functional unit binding is shown beside the DFG node. According to the algorithm in [14], 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 bound to ALU1. If node 3 is scheduled first, the DFG will be scheduled in three clock cycles. However, if node 4 is scheduled first and bound to ALU1, the DFG is scheduled in four clock cycles.

To overcome this deficiency, we propose a new algorithm based on the FDLS framework [22]. It is integrated with simultaneous rebinding to minimize the schedule latency with consideration of interconnect delays. FDLS is a modification of the force-directed scheduling algorithm [22]. It is used to solve the resource-constrained scheduling problem as a list-based scheduling algorithm, but by using force as the priority function. A force<sup>4</sup> is a value associated with the tentative assignment of an operation to a control step. It indicates the potential resource congestion (or resource competitions) incurred by this assignment. Since FDLS defers the operation with minimum force, it can efficiently minimize potential resource competitions.

Using the similar notations in [14], we define  $\text{cpl}_{i,j}$  as the critical path length from an operation node  $n_i$  to the POs of DFG when  $n_i$  is bound to a functional unit  $f_j$ . The critical path length is computed as

$$\text{cpl}_{i,j} = c_j + \max_{n_k \in \text{succ}_i} \left\{ \min_l \{d_{j,l} + \text{cpl}_{k,l}\} \right\}$$

where  $\text{succ}_i$  represents the set of successors of  $n_k$ , and  $c_j$  is the computation time of  $f_j$ .  $d_{j,l}$  is the delay between functional unit  $j$  and  $l$ .

Critical path length of  $n_i$  is  $\text{cpl}_i = \min_j \{\text{cpl}_{i,j}\}$ . Time constraint is calculated as  $tc = \max_i \{\text{cpl}_i\}$ . ALAP schedule time of each node is  $\text{ALAP}_i = tc - \text{cpl}_i$ . ASAP<sub>i</sub> of each operation is calculated in the similar way.

The process of the algorithm is shown in Fig. 14. The first step of the algorithm is the deferral 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

<sup>4</sup>For details of how to compute the force of an operation node, please refer to [22].

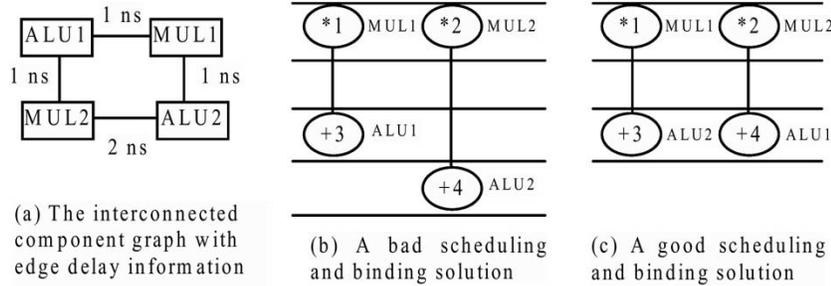


Fig. 13. Limitation of the algorithm in [14].

```

// Deferral node selection
while available resource is not enough do
  Calculate forces for ready nodes;
  Defer node with Minimum <force, cpl, est>;
  Push deferred node in Deferred_node_stack;
  if all nodes are on critical path do
    Increment timing constraint;
    Update distribution graph;
  end if
end while

// Schedule and bind ready nodes
while ready node list is not empty do
  Bind a node to a FU with Maximum <cpl, est>;
  Put non-schedulable node in wait list;
end while

// Binding deferred node if there is available resources
while FUs is available and Deferred_node_stack is not empty do
  Schedule and bind the top node of the Deferred_node_stack;
  Pop the Deferred_node_stack;
end while

```

Fig. 14. Force-directed simultaneous scheduling and binding algorithm.

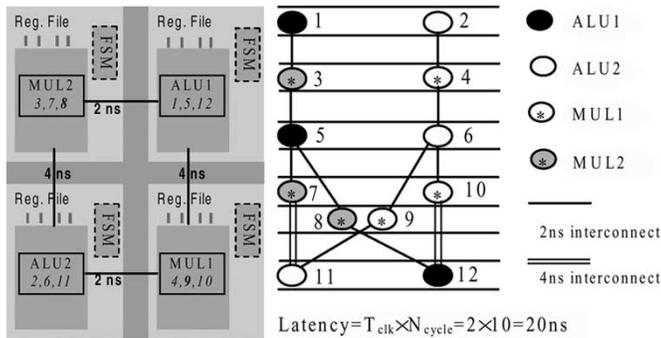


Fig. 15. Advantage of considering rescheduling and rebinding after placement.

order of deferral. After that, the algorithm will proceed to the next iteration until all nodes are scheduled and bound.

Fig. 15 illustrates the advantage of considering rescheduling and rebinding after the placement for multicycle communication. By switching the bindings of nodes 8 and 9, we can obtain a new layout shown in Fig. 15(a), and further reduce schedule latency to 20 ns as shown in Fig. 15(b).

### F. Distributed Control Generation

In this section, we present the algorithm on distributed control generation for our RDR microarchitecture. Section I describes

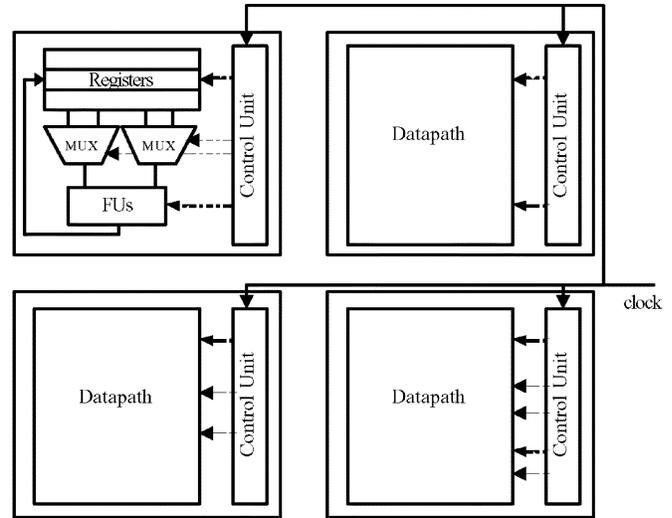


Fig. 16. Basic distributed control scheme for the RDR microarchitecture.

the basic distributed control approach for pure DFG, and Section II shows distributed control generation algorithm for the two-level CDFG.

1) *Distributed Control for DFG*: The RDR microarchitecture requires distributed control for each island. Every local control signal transmission should be made within one clock cycle. Fig. 16 illustrates a basic distributed control scheme for the RDR microarchitecture. Each island contains an FSM, which generates control signals for the datapath components in the same island. The control signals include function selections for functional units, MUX selections, and clock enables for registers and memories, etc.

For a pure DFG, each controller is essentially a linear sequence of control steps. The control step transitions are triggered by the system clock, regardless of the status of the datapath (i.e., there is no signal from the datapath to the controller). The distributed controllers in different islands are independent. They have the identical control step transition diagrams but different output signals.

#### 2) *Distributed Control for CDFG*:

a) *Preliminaries and Motivation*: A traditional approach to generate a control unit for an unbounded-latency CDFG is discussed in [19]. A state is generated for each basic block in the CDFG. It consists of a sequence of control steps, and controls the execution of the corresponding basic block. Branch signals generated by the current state activate the next state, according to the status signal generated from the datapath.

A Mealy-type FSM  $M$  is a six-tuple  $(S, X, \delta, s_0, O, \lambda)$ , where

- $S$  is a finite set of states.
- $X$  is the input alphabet (signal).
- $\delta : S \times X \rightarrow S$  is the next state transition function.
- $s_0 \in S$  is the initial state.
- $O$  is the output alphabet (signal).
- $\lambda : S \times X \rightarrow O$  is the output function.

For the RDR microarchitecture, we need to decompose the FSM into a set of distributed FSMs. Assuming that the RDR is defined as  $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_n\}$ , where  $\mathcal{I}_i$  represents an island, we should generate a local FSM  $M_{\mathcal{I}_i}$  for island  $\mathcal{I}_i$ , where

$$M_{\mathcal{I}_i} = (S_{\mathcal{I}_i}, X_{\mathcal{I}_i}, \delta_{\mathcal{I}_i}, s_{0\mathcal{I}_i}, O_{\mathcal{I}_i}, \lambda_{\mathcal{I}_i}).$$

In addition, we have the constraint that every output signal  $o \in O_{\mathcal{I}_i}$  should drive a local resource inside island  $\mathcal{I}_i$ .

One possible solution is to duplicate the FSM into every island and maintain their synchronization. However, this method is not efficient in terms of both area and delay. When one FSM makes a state transition on a trigger event, all other FSMs should wait for this event to be synchronized. Since the trigger event may take multicycles to reach the FSMs in other islands, the synchronization delay should be the delay from the location generating the trigger signal to the furthest island, i.e., for state transition  $\delta(s_j, x) = s_k$

$$\text{Delay}(\delta) = \text{MAX}_{1 \leq i \leq n} \{\text{Delay}(\mathcal{I}_{(x)}, \mathcal{I}_i)\} \quad (1)$$

where signal  $x$  is generated in island  $\mathcal{I}_{(x)}$ .

*b) Our Approach:* We take a partial state duplication approach to generate distributed controllers. We duplicate states to an island only when they are required by this island. Precisely, suppose  $s_j$  is a state for basic block  $j$  of a CDFG. If in island  $\mathcal{I}_i$  there is no resource allocated for basic block  $j$ , then  $s_j$  is not duplicated into  $\mathcal{I}_i$ . After the duplication phase, we create state transitions to combine these duplicated states together to form a local FSM.

Our distributed control generation algorithm is described in Fig. 17. The input is an FSM  $M$  generated by the traditional approach. For the local FSM  $M_{\mathcal{I}_i}$  of island  $\mathcal{I}_i$ , we first generate required states whose outputs drive the logics in this island. The physical locations of the resources are determined by the scheduling-driven placement. In the algorithm,  $s_{ji}$  denotes a state of  $M_{\mathcal{I}_i}$  and is a duplicated state of state  $s_j$  of  $M$ . We then generate the state transitions and output functions according to FSM  $M$ . Note that we also generate a new initial state and *reset* signal for every distributed FSM.

By this method, we avoid unnecessary state duplications, generate smaller distributed FSMs, and potentially reduce the interaction delay between distributed FSMs, thereby achieving more efficient area and delay cost for controllers. The delay for state transition  $\delta(s_j, x) = s_k$  is

$$\text{Delay}(\delta) = \text{MAX}_{1 \leq i \leq n} \{\text{Delay}(\mathcal{I}_{(x)}, \mathcal{I}_i) | s_{ki} \in S_{\mathcal{I}_i}\} \quad (2)$$

where signal  $x$  is generated in island  $\mathcal{I}_{(x)}$ , and  $s_{ki}$  is  $\mathcal{I}_i$ 's local duplication of  $s_k$ . On average, this delay will be less than the delay of (1).

```

for each  $\mathcal{I}_i \in \mathcal{I}$  do
     $S_{\mathcal{I}_i} = \emptyset$  /* where  $S_{\mathcal{I}_i}$  is the state set of  $M_{\mathcal{I}_i}$  */
    for each  $s_j \in S$  do
        if  $\lambda(s_j, x) = o_i, x \in X, o_i$  drives a resource in  $\mathcal{I}_i$ 
            Generate state  $s_{ji}$ 
             $S_{\mathcal{I}_i} = S_{\mathcal{I}_i} \cup s_{ji}$ 
        end if
    Local_FSM_Gen ( $\mathcal{I}_i$ )
    /* Local_FSM_Gen ( $\mathcal{I}_i$ ) generates a local FSM  $M_{\mathcal{I}_i}$  for  $\mathcal{I}_i$ :
         $M_{\mathcal{I}_i} = (S_{\mathcal{I}_i}, X_{\mathcal{I}_i}, \delta_{\mathcal{I}_i}, s_{0\mathcal{I}_i}, O_{\mathcal{I}_i}, \lambda_{\mathcal{I}_i})$ , where
         $X_{\mathcal{I}_i} = X \cup \{\text{reset}\}$ ;
         $\delta_{\mathcal{I}_i}(s_{ji}, \text{reset}) = s_{0\mathcal{I}_i} \forall s_{ji} \in S_{\mathcal{I}_i}$ ;
         $\delta_{\mathcal{I}_i}(s_{0\mathcal{I}_i}, x) = s_{ki}, \text{if } \exists j, \delta(s_j, x) = s_k$ 
         $\delta_{\mathcal{I}_i}(s_{ji}, x) = s_{ki}, \text{if } \delta(s_j, x) = s_k$ 
         $\lambda_{\mathcal{I}_i}(s_{ji}, x) = o_i, \text{if } \lambda(s_j, x) = o_i$  */
    
```

Fig. 17. Distributed FSMs generation algorithm.

Fig. 18 illustrates the distributed controller generation. In this example, we assume a two-island RDR microarchitecture, denoted by islands  $L$  and  $R$ . Suppose after the scheduling-driven placement, island  $L$  contains the computation logics for basic blocks 2, 4, and island  $R$  contains computation logics for basic blocks 1, 3, and 4. We decompose the original FSM into two FSMs as shown in Fig. 18(c). Interactions between these FSMs, represented as dotted lines, are required to maintain the synchronization. For example, the event signal that trigger the transition from state 2 to state 4 in island  $L$ , may be held for multicycles. It guarantees that the interisland communication can reach island  $R$ , and trigger the transition from the initial state to state 4 in island  $R$ .

#### IV. EXPERIMENTAL RESULTS

We implemented the MCAS system in C++/UNIX environments. For comparison, we also set up three alternative flows. Fig. 19 shows the three flows, which are traditional (Trad), placement-driven scheduling (PDS), and placement-driven scheduling with rebinding (PDS-R) flows. The leftmost branch in Fig. 19 is a traditional architectural synthesis flow, which is based on the centralized register file architecture. It performs the binding and list-scheduling algorithm sequentially without considering the layout. The placement-driven scheduling with rebinding flow, shown as the rightmost branch in Fig. 19, is our MCAS flow discussed in Section III-A. Similar to the placement-driven scheduling with rebinding flow, the placement-driven scheduling flow is also based on the RDR microarchitecture and the location information provided by the scheduling-driven placement. However, it only performs scheduling for the given binding instead of simultaneous rebinding and rescheduling. The same list-scheduling algorithm is applied for all three flows. These three scheduling flows share the same back end to generate datapath and controllers.

To obtain the final performance results, Altera's Quartus II version 2.2 [34] is used to implement the datapath part into a real FPGA device,<sup>5</sup> Stratix EP1S40F1508C5. All the pipelined

<sup>5</sup>Considering the availability of the design tools, cell libraries, and timing models, we chose FPGA platform. We would expect similar or better experimental results in application specific integrated circuit platform, in which the interconnect delay versus gate delay ratio and communication overhead are higher.

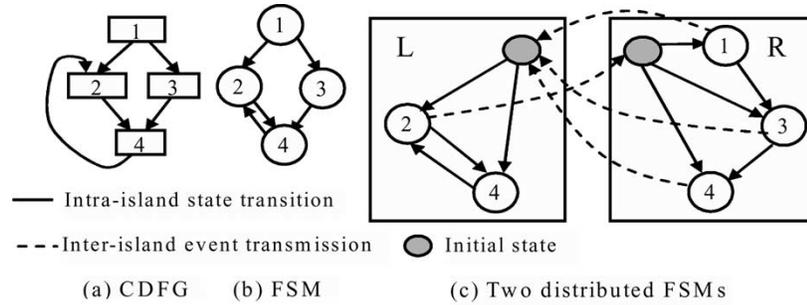


Fig. 18. Distributed control generation for CDFG based on the RDR microarchitecture.

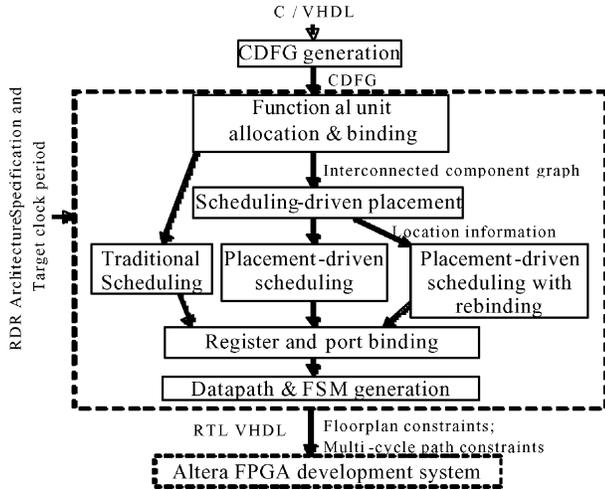


Fig. 19. Three experimental flows.

multipliers are implemented into the dedicated DSP blocks of the Stratix device. We set the target clock frequency at 200 MHz and use the default compilation options. We use LogicLock to constrain every instance into its corresponding island, and set multicyle constraints for multicyle communication paths.

The island size of the RDR microarchitecture is determined by the equation discussed in Section II. Accounting for the regularity of the target device which contains  $7 \times 2$  DSP blocks, we applied a  $7 \times 4$  RDR microarchitecture in the experiments.

A set of real-life benchmarks is used in our experiments. The pure DFG programs are from [27], including several different DCT algorithms, such as PR, WANG, LEE, and DIR, and several DSP programs, such as MCM, HONDA, CHEM, and U5ML12. Three other benchmarks with the control flow, including MATMUL, CFTMDL, and CFT1ST, are from MedaBench [17] and an FFT package [36]. All the benchmarks are data intensive applications.

#### A. Comparison for DFG Examples

Table I shows the functional unit and register binding results. The second column lists the node numbers of the DFG examples. ALU# and MUL# are the numbers of the corresponding functional unit usages after the initial binding. Although the three flows use the same number of functional units, the PDS-R flow has different binding results due to the rebinding process. The next three columns are register numbers (REG#) from the different flows. For this set of benchmarks, the PDS and PDS-R

TABLE I  
FUNCTIONAL UNIT AND REGISTER BINDING RESULTS

	Node#	ALU#	MUL#	REG#		
				Trad	PDS	PDS-R
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 II  
RELATIVE CYCLE NUMBER, CLOCK PERIOD, AND OVERALL LATENCY COMPARED WITH THE TRADITIONAL FLOW

	PDS			PDS-R		
	Clock Cycles	Clock Period	Latency	Clock Cycles	Clock Period	Latency
pr	1.07	0.61	0.65	1.07	0.63	0.68
wang	1.43	0.55	0.78	1.43	0.51	0.72
lee	1.35	0.54	0.73	1.30	0.54	0.70
mcm	1.15	0.63	0.72	1.12	0.60	0.67
honda	1.04	0.50	0.52	1.04	0.55	0.58
dir	1.02	0.63	0.64	1.02	0.62	0.63
chem	1.06	0.56	0.59	1.04	0.54	0.56
U5ml12	1.03	0.57	0.59	1.03	0.46	0.48
Ave Ratio	1.14	0.57	0.65	1.13	0.56	0.63

flows use 34% and 28% (on average) more registers than the traditional flow, respectively. PDS-R, which can refine the initial functional unit binding based on the given placement, results in a smaller register usage than PDS.

In Table II, we list the ratios of the clock cycles, the clock periods, and the total latencies produced by the placement-driven flows compared with the traditional flow. The clock periods are reported by Quartus II. Due to consideration of the interconnect delay, the placement-driven flows introduce 14% more cycles for the communication between registers, compared with the traditional flow. However, since our flows distribute registers close to functional units and apply multicyle path constraints for communications, we achieve much smaller clock periods (more than a 40% reduction).

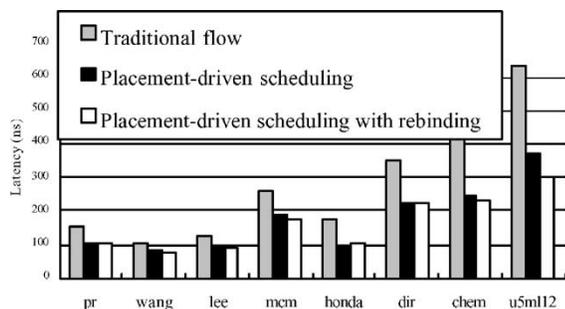


Fig. 20. Total latency comparison for the three flows.

TABLE III  
LUT AND REGISTER USAGE COMPARISON

	LUT			Register		
	Trad	PDS	PDS-R	Trad	PDS	PDS-R
pr	1	1.10	1.13	1	2.00	1.84
wang	1	1.15	0.99	1	2.83	2.92
lee	1	1.04	1.12	1	3.14	2.93
mcm	1	1.05	1.01	1	1.59	1.49
honda	1	1.20	1.36	1	1.82	1.87
dir	1	1.23	1.05	1	1.61	1.50
Chem.	1	1.31	1.36	1	2.31	2.26
u5ml12	1	1.40	1.37	1	2.41	2.03
Ave Ratio	1	1.19	1.17	1	2.21	2.10

We illustrate the total latencies in Fig. 20, where the three bars in every group represent the results from the three flows, respectively. Compared with the traditional flow, the PDS and PDS-R flows reduce the final latencies of the designs by 35% and 37%, respectively. It can be seen that PDS-R (MCAS flow) further improves the performance by simultaneous scheduling with rebinding.

Table III lists the ratios of the resources used by different design flows in terms of LUTs and registers. It can be seen that the placement-driven flows introduce less than 20% LUT overhead, but more than 100% registers as overhead. Since our RDR microarchitecture 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 MUXs, which contributes to an observable area increase in the final layout, especially for an FPGA design.

Although an SA-based placer is used, MCAS is not very slow as the scale of the ICG is typically small (number of components  $< 10^2$ ) and the list scheduling is very fast. For the largest benchmark, all the MCAS modules, including scheduling-driven placement and placement-driven scheduling with rebinding, etc., can finish within 60 s.

*B. Comparison for CDFG Examples*

For the three benchmarks containing control flow, we made comparison only between MCAS and the traditional flow. As shown in Table IV, MCAS achieves 28% clock period and 23% latency improvement over the traditional flow. Since in designs with control flow the distributed controllers become more complex than those for pure DFG designs, the resulting

TABLE IV  
MCAS VERSUS TRADITIONAL FLOW FOR CDFG EXAMPLES

Design		Function Unit Binding		MCAS / Traditional Flow				
Name	Node#	ALU#	MULT#	REG#	LE	Clock Cycle	Clock Period	Latency
matmul	63	4	9	88/70	1.19	1.11	0.71	0.78
cftmdl	199	10	7	161/152	0.92	1.02	0.76	0.77
cft1st	255	10	8	247/235	1.01	1.08	0.70	0.75
Ave Ratio	-	-	-	1.12	1.04	1.07	0.72	0.77

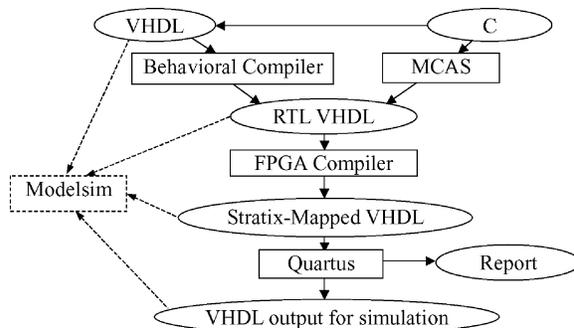


Fig. 21. BC versus MCAS flows.

clock period improvements MCAS achieves are not so large. However, since the variables' life intervals tend to be more compatible in these CDFGs, MCAS introduces only 12% more registers and 4% area overhead in terms of logic element (LE).

*C. MCAS Versus Synopsys Behavioral Compiler (BC)*

We further validate the advantage of MCAS by comparing it with a state-of-the-art commercially available behavioral synthesis tool, BC from Synopsys. The experimental flows are illustrated in Fig. 21. Since the MCAS system currently only supports C input and BC accepts VHDL format, we first transform the C descriptions into behavioral VHDL manually. We selected four representative benchmarks,<sup>6</sup> PR, WANG, MCM, and HONDA, from the benchmark set used in Section IV-A for this comparison.

The Synopsys package that we use is Version 2000.11 for spareOS5, including BC, FPGA Compiler, and DesignWare Developer. We set the default options for BC and high mapping effort for FPGA Compiler for the experiments. BC and MCAS are performed to generate RTL VHDLs using DesignWare components. FPGA Compiler then maps the RTL VHDL to Altera's Stratix device. Again, Quartus II handles the placement and routing.

Table V lists the ratios of the performance results by comparing MCAS versus Synopsys BC.<sup>7</sup> The two flows achieve the similar resource cost and scheduled clock cycles. However, MCAS flow obtains a 21% improvement in clock period and a 29% improvement in total latency on average.

<sup>6</sup>The rest benchmarks are too large for BC to run through. For example, we cannot obtain the schedule results from BC for DIR, CHEM, and U5ML12, etc., due to the long runtime.

<sup>7</sup>The number of resources allocated by two systems differ a lot as BC uses various bit-width resources whereas the current version MCAS only supports uniform bit-width resource.

TABLE V  
COMPARISON FOR MCAS VERSUS BC

Design	Synopsys BC							MCAS / Synopsys BC						
	ALU#	MUL#	REG#	LE	Clock Cycles	Clock Period	Latency	ALU#	MUL#	REG#	LE	Clock Cycles	Clock Period	Latency
pr	5	8	28	2945	25	11.07	276.82	1.20	0.25	1.25	0.87	1.16	0.81	0.94
wang	7	8	36	3605	29	11.96	346.85	0.71	1.00	1.28	1.16	0.69	0.75	0.52
mcm	23	7	142	6253	43	12.55	539.86	0.26	0.43	0.35	0.70	0.88	0.78	0.69
honda	8	14	44	6128	29	11.75	340.62	0.75	0.57	1.27	1.03	0.86	0.80	0.69
Average	-	-	-	-	-	-	-	0.73	0.56	1.04	0.94	0.90	0.79	0.71

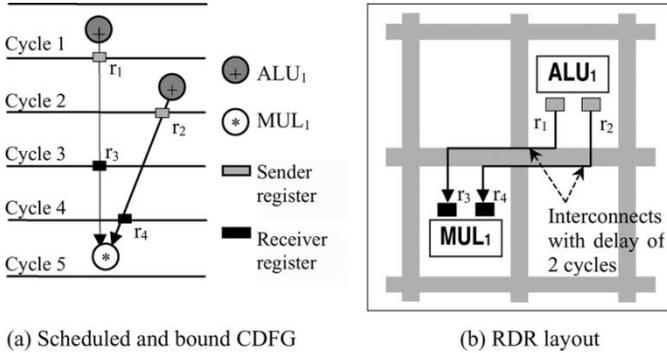


Fig. 22. Illustration of the wiring overhead in RDR.

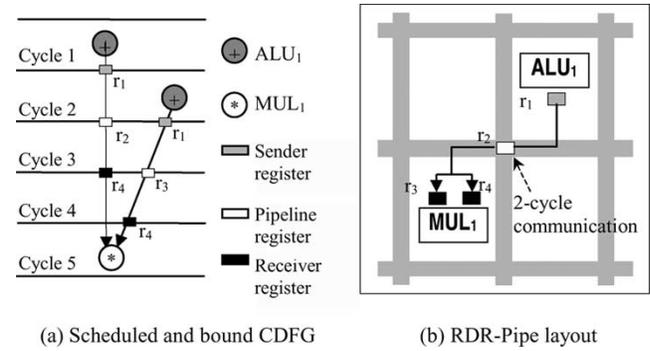


Fig. 24. Illustration of the interconnect pipelining and sharing in the RDR-Pipe.

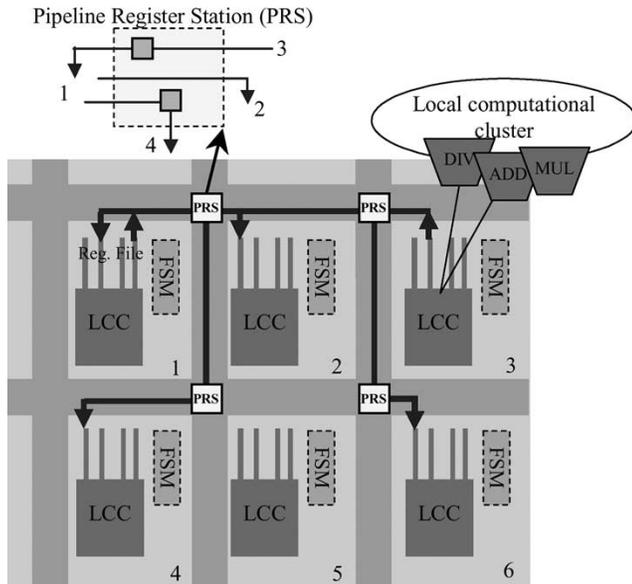


Fig. 23. RDR-Pipe: one possible extension to RDR for interconnect pipelining.

## V. CONCLUSION AND FUTURE WORK

We have proposed a novel RDR microarchitecture, which provides a regular synthesis platform for supporting multicycle on-chip communication in multigigahertz designs. The regularity of the RDR microarchitecture facilitates the predictability of interconnect delays at the higher design levels, and offers a way to systematically explore the cycle time versus latency tradeoff.

An architectural synthesis system, named MCAS, has also been developed on top of the RDR microarchitecture. Unlike

the other methodologies (e.g., asynchronous design, GALS design, etc.), our approach allows the designer to specify a system within the familiar synchronous design framework. They only need to pass the behavioral descriptions to our MCAS synthesis system. After the proper scheduling and binding by MCAS, they can obtain the RTL, and use the off-the-shelf logic synthesis and physical design tools directly. The experimental results reported by the commercial place-and-route tools have demonstrated the effectiveness of our proposed architecture, design methodology, and synthesis algorithms. For data flow intensive examples, we obtain 44% improvement on average in terms of the clock period and a 37% improvement on average in terms of the final latency, over the traditional flow. For designs with control flow, our approach achieves a 28% clock period reduction and a 23% latency reduction on average.

One limitation of the RDR microarchitecture is that extra wiring overhead may be introduced due to the possible existence of many simultaneous data transfers among the islands as each one requires a dedicated global connection. Since each signal transmission over a global wire occupies multiple cycles, sharing the wire is not possible unless they can be serialized.

Fig. 22 illustrates the problem using a very simple CDFG with 3 operation nodes. Again we assume uniform node delay that is equal to the cycle time, and the patterns on the nodes denote the functional unit binding. Given the layout shown in Fig. 22(b), the CDFG will be scheduled into five cycles. Using the RDR microarchitecture, four registers will be allocated and the sender registers  $r_1$  and  $r_2$  will hold their values for two cycles to allow the signals to reach the receiver registers  $r_3$  and  $r_4$ . Therefore, two parallel interisland wires are needed between  $ALU_1$  and  $MUL_1$  as their data transfers time overlap.

Using the same experimental settings discussed in Section IV, we can also obtain the wirelength reports from Quartus II on the same benchmark suite. The results show that, compared with traditional flow, MCAS flow increased the global wirelength by 46% and the total wirelength by 15% on average.<sup>8</sup> Clearly, more wiring overhead will be incurred as the designs grow. Since the global wires are an expensive resource, this deficiency has to be addressed.

In fact, for a  $K$ -cycle global interconnect, it is not necessary to hold the sender register constant for  $K$  cycles. Instead flip-flops can be inserted to the line to relay the signal in  $K$  cycles. In this way, although the data transfer still takes  $K$  clocks to go through the interconnect, new data can be launched every cycle. Therefore, more data transfers can share the same global wire as the minimal required launch interval is reduced from  $K$  to 1. Additionally, the throughput of a pipelined interconnect can be up to  $K$  times greater than that of the nonpipelined one in RDR.

Fig. 23 shows one possible extension to the RDR microarchitecture to enable the interconnect pipelining scheme, which is called RDR-Pipe. In this RDR-Pipe microarchitecture, pipeline registers are inserted on the global interconnects so that every  $K$ -cycle interisland communication will go through  $K - 1$  intermediate pipeline registers. The pipeline registers reside in the pipeline register stations (PRS) that are distributed along the routing channels. The incoming signals to a PRS are either relayed through a pipeline register or directly switched to different directions. Note that the pipeline registers only perform the store-and-forward function so that they are autonomous and do not need control signals.

Fig. 24 illustrates the advantages of the RDR-Pipe using the same CDFG example shown in Fig. 22. In the presence of a pipeline register  $r_2$ ,  $ALU_1$  can emit a value to  $r_1$  (denoted as  $v_1$ ) at the first cycle and emit the other value still to  $r_1$  at the second cycle as  $v_1$  has been already transferred to  $r_2$ . Therefore, the two data transfers can share the same interconnect, thus, only one global wire is needed this time between  $ALU_1$  and  $MUL_1$ . In fact, this result can be further generalized as the following. For any pair of functional units, at most one interisland wire is needed in-between under the RDR-Pipe microarchitecture.

As shown above, the RDR-Pipe microarchitecture preserves the strength of the RDR microarchitecture to achieve high performance, meanwhile, it directly supports automatic interconnect pipelining and potentially allows better wiring utilization compared with the RDR. Our future work will concentrate on the development of efficient synthesis algorithms such as functional unit binding and register allocation on top of this extended RDR microarchitecture for further wiring reduction.

Another part of enhancement will be the synthesis of the control-intensive applications. Specifically, we are developing global scheduling and resource-sharing techniques to exploit inter basic block parallelism in CDFG.

<sup>8</sup>Eight types of general wires in Stratix devices are counted:  $LL$  and  $LO$  are local wires in LABs with unit length. Wire types named  $Hn$  ( $n \in \{4, 8, 24\}$ ) and  $Vm$  ( $m \in \{4, 8, 16\}$ ) are horizontal wire length  $n$  and vertical wire length  $m$ , respectively.  $V16$  and  $H24$  are considered as global wires.

## ACKNOWLEDGMENT

The authors thank Professor M. Potkonjak of the University of California, Los Angeles, for sharing benchmarks, Professor K. Choi of Seoul National University, Seoul, Korea, for providing the CDFG toolkit [35], and Dr. V. Betz of the Altera Toronto Technology Centre, Toronto, Canada, for giving the detailed instructions of experimentation on Quartus II development system.

## REFERENCES

- [1] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1984.
- [2] W. H. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Communications*, vol. 25, pp. 1004–1009, Sept. 1977.
- [3] P. Chong and R. K. Brayton, "Characterization of feasible retimings," in *Proc. Int. Workshop Logic Synthesis*, Jun. 2001, pp. 1–6.
- [4] J. Cong and C. Wu, "FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits," in *Proc. 34th ACM/IEEE Design Automation Conf.*, Jun. 1997, pp. 644–649.
- [5] J. Cong and S. K. Lim, "Physical planning with retiming," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 2–7.
- [6] J. Cong and X. Yuan, "Multilevel global placement with retiming," in *Proc. 40th Design Automation Conf.*, Jun. 2003, pp. 208–213.
- [7] J. Cong and Z. Pan, "Interconnect performance estimation models for design planning," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 739–752, Jun. 2001.
- [8] J. Cong, "Timing closure based on physical hierarchy," in *Proc. 2002 Int. Symp. Physical Design*, Apr. 2002, pp. 170–174.
- [9] J. Cong, Y. Fan, X. Yang, and Z. Zhang, "Architecture and synthesis for multi-cycle communication," in *Proc. 2003 Int. Symp. Physical Design*, Apr. 2003, pp. 190–196.
- [10] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architectural synthesis integrated with global placement for multi-cycle communication," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2003, pp. 536–543.
- [11] W. J. Dally and S. Lacy, "VLSI architecture: Past, present and future," in *Proc. 20th Conf. Adv. Res. VLSI*, Mar. 1999, pp. 232–241.
- [12] Y. M. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 317–321.
- [13] K. I. Farkas, N. P. Jouppi, P. Chow, and Z. Vranesic, "The multicenter architecture: Reducing cycle time through partitioning," in *Proc. 30th Int. Symp. Microarchitect.*, Dec. 1997, pp. 149–159.
- [14] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," in *Proc. Asia South Pacific Design Automation Conf.*, Jan. 2001, pp. 662–667.
- [15] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2001, pp. 320–326.
- [16] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distributed Syst.*, vol. 7, pp. 506–521, May 1996.
- [17] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. Int. Symp. Microarchitect.*, Nov. 1997, pp. 330–335.
- [18] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proc. Int. Symp. Field Program. Gate Arrays*, Feb. 2000, pp. 203–213.
- [19] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.
- [20] F. Mo and R. K. Brayton, "Regular fabrics in deep sub-micron integrated-circuit design," in *Proc. 11th IEEE/ACM Int. Workshop Logic Synthesis*, Jun. 2002, pp. 7–12.
- [21] M. C. Papaefthymiou, "Understanding retiming through maximum average-delay cycles," *Math. Syst. Theory*, vol. 27, pp. 65–84, 1994.
- [22] P. Paulin and J. Knight, "Force-directed scheduling for behavioral synthesis of ASICs," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 661–679, Jun. 1989.
- [23] P. Prabhakaran and P. Banerjee, "Parallel algorithms for simultaneous scheduling, binding and floorplanning in high-level synthesis," in *Proc. Int. Symp. Circuits Syst.*, May 1998, pp. 372–376.
- [24] L. Scheffer, "Methodologies and tools for pipelined on-chip interconnect," in *Proc. Int. Conf. Computer Design*, Sept. 2002, pp. 152–157.

- [25] D. P. Singh and S. D. Brown, "Integrated retiming and placement for field programmable gate arrays," in *Proc. Int. Symp. Field Programm. Gate Arrays*, Feb. 2002, pp. 67–76.
- [26] M. D. Smith and G. Holloway, "An introduction to machine SUIF and its portable libraries for analysis and optimization," in *Division of Engineering and Applied Sciences*. Cambridge, MA: Harvard Univ., 2002.
- [27] M. B. Srivastava and M. Potkonjak, "Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 2–19, Mar. 1995.
- [28] J. T. Udding, "A formal model for defining and classifying delay-insensitive circuits and systems," *Distrib. Comput.*, vol. 1, no. 4, pp. 197–204, 1986.
- [29] J. Um, J. Kim, and T. Kim, "Layout-driven resource sharing in high-level synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2002, pp. 614–618.
- [30] J. Weng and A. Parker, "3D scheduling: High-level synthesis with floor-planning," in *Proc. Design Automation Conf.*, Jun. 1991, pp. 668–673.
- [31] M. Xu and F. J. Kurdahi, "Layout-driven RTL binding techniques for high-level synthesis," in *Proc. 9th Int. Symp. Syst. Synthesis*, Nov. 1996, pp. 33–38.
- [32] *The National Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 1997.
- [33] *International Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 2001.
- [34] Altera Web Site [Online]. Available: <http://www.altera.com>.
- [35] CDFG Toolset [Online]. Available: <http://poppy.snu.ac.kr/CDFG>.
- [36] FFT Package [Online]. Available: <http://momonga.t.u-tokyo.ac.jp/~ooura/fft.html>.
- [37] SUIF Compiler [Online]. Available: <http://suif.stanford.edu>.



**Jason Cong** (S'88–M'90–SM'96–F'00) received the B.S. degree from Peking University, Beijing, China, in 1985 and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign, in 1987 and 1990, respectively, all in computer science.

Currently, he is a Professor and Co-Director of the VLSI CAD Laboratory in the Computer Science Department, University of California, Los Angeles. He has been appointed as a Guest Professor at Peking University since 2000. He has published over 170 research papers and led over 20 research projects

supported by the Defense Advanced Research Projects Agency, the National Science Foundation, the Semiconductor Research Corporation (SRC), and a number of industrial sponsors in these areas. His research interests include layout synthesis and logic synthesis for high-performance low-power VLSI circuits, design and optimization of high-speed VLSI interconnects, field-programmable gate array (FPGA) synthesis, and reconfigurable computing.

Prof. Cong has served as the General Chair of the 1993 ACM/SIGDA Physical Design Workshop, the Program Chair and General Chair of the 1997 and 1998 International Symposia on FPGAs, respectively, and on program committees of many VLSI CAD conferences, including the Design Automation Conference, International Conference on Computer-Aided Design, and International Symposium on Circuits and Systems. He is an Associate Editor of *ACM Transactions on Design Automation of Electronic Systems* and *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*. He received the Best Graduate Award from Peking University, in 1985, and the Ross J. Martin Award for Excellence in Research from the University of Illinois, in 1989. He received the Research Initiation and Young Investigator Awards from the National Science Foundation, in 1991 and 1993, respectively. He received the Northrop Outstanding Junior Faculty Research Award from the University of California, in 1993, and the *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN* Best Paper Award in 1995. He received the ACM Recognition of Service Award in 1997, the ACM Special Interest Group on Design Automation Meritorious Service Award in 1998, and the Inventor Recognition and Technical Excellence Awards from the SRC, in 2000 and 2001, respectively.



**Yiping Fan** (S'02) received the B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 1998 and 2001, respectively. He is currently pursuing the Ph.D. degree in computer science at the University of California, Los Angeles.

His current research interests include very large scale integration system-level design, architectural synthesis, and reconfigurable computing.



**Guoling Han** (S'03) received the B.S. and M.S. degrees in computer science from Peking University, Beijing, China, in 1999 and 2002, respectively. He is currently pursuing the Ph.D. degree in computer science at the University of California, Los Angeles.

His current research interests include system-level synthesis and behavioral synthesis.



**Xun Yang** received the B.S. degree in computer science from Hefei Poly-Technique University, Hefei, China, in 1993, and the M.S. and Ph.D. degrees in computer science from Beijing Institute of Technology, Beijing, China, in 1996 and 1999, respectively.

He was with the Computer Science Department of Tsinghua University, Beijing, China, from 1999 to 2001, and with the Computer Science Department of University of California, Los Angeles, from 2001 to 2003 as a Postdoctoral Researcher. He joined

Silvaco Data Systems, Santa Clara, CA, in 2003. His research interests include VHDL simulation algorithm, high-level synthesis, and system-level design and synthesis.

Dr. Yang won Chinese National Third Class Scientific Progress Award for the "VHDL high level synthesis and multilevel simulation system for ASIC design" project that he worked on in 1999.



**Zhiru Zhang** (S'02) received the B.S. degree in computer science from Peking University, Beijing, China, in 2001 and the M.S. degree in computer science in 2003 from University of California, Los Angeles (UCLA), where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include platform-based hardware/software codesign for embedded systems, interconnect-driven high-level synthesis, and compilation techniques for reconfigurable systems.