Architecture-Level Synthesis for Automatic Interconnect Pipelining

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

ABSTRACT
For multi-gigahertz synchronous designs in nanometer technologies, multiple clock cycles are needed to cross the global interconnects, thus making it necessary to have pipelined global interconnects. In this paper we present an architecture-level synthesis solution to support automatic pipelining of on-chip interconnects. Specifically, we extend the recently proposed Regular Distributed Register (RDR) micro-architecture to support interconnect pipelining. We formulate a novel global interconnect sharing problem for global wiring minimization and show that it is polynomial time solvable by transformation to a special case of the real-time scheduling problem. Experimental results show that our approach matches or exceeds the RDR-based approach in performance, with a significant wiring reduction of 15% to 21%.

Categories and Subject Descriptors
B.5.2 [**Hardware**]: Design Aids – *automatic synthesis*

General Terms
Algorithms, Design, Experimentation

Keywords
High-level synthesis, multi-cycle communication, interconnect pipelining, scheduling, register binding

1. INTRODUCTION
Nanometer process technologies enable gigascale integration with multiple-gigahertz operating 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. According to the predictions by SIA ITRS roadmaps [14], 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.

Since the clock period often represents a fixed constraint in high-performance designs, it is not acceptable to simply degrade the entire design to the speed of the slowest global interconnect. Although integration of retiming with placement or floorplanning [2][12][4] can help to alleviate the problem, it cannot derive a circuit whose clock period works less than the lower bound defined by the maximum delay-to-register ratio of the loop in the circuit [10].

To further improve the clock speed, one can pipeline the long wires by inserting clocked and enabled elements such as latches and flip-flops. The gains from this technique can be dramatic, as the clock frequencies are no longer restricted by the interconnect speed. In one reported case, Intel inserted thousands of flip-flops on the global wires of the Itanium™ processor and achieved up to 1.7 GHz operating frequency even under 0.18um technology [9]. ITRS [14] has also acknowledged this strategy by removing global clock cycle times from its 2001 and later roadmaps. Some recent works [8][3] combined buffer and flip-flop insertion with the simple assumption that flip-flops can be inserted at will. However, they did not address the intrinsic difficulties of wire pipelining under the RT level. Flip-flop insertion may change the cycle-level behavior of the circuit [11]; this requires a considerable amount of manual rework to the RTL design. Even worse, such rework is usually performed in ad hoc ways with no, or very limited, automated tool support, which seriously compromises the design productivity.

Because of all these aforementioned difficulties, new design methodologies are required for coping with the increasingly important on-chip communication design at a higher-level abstraction. The recently proposed Regular Distributed Register (RDR) micro-architecture [5] provides a promising way to address this problem. It offers high regularity and direct support of the multi-cycle on-chip communication. However, the RDR micro-architecture may introduce extra global wiring overhead in the presence of many simultaneous data transfers, as each one requires a dedicated global connection.

In this paper we present an architecture-level synthesis solution to support automatic interconnect pipelining. The main contributions of this work are as follows: (i) We propose an extension to the RDR micro-architecture, called RDR-Pipe, to efficiently support the multi-cycle on-chip communication with interconnect pipelining. (ii) We formulate a novel global interconnect sharing problem for global wiring minimization, and show that it is polynomial time solvable by transformation to a special case of the real-time scheduling problem.

The remainder of the paper is organized as follows. Section 2 reviews the RDR micro-architecture and discusses its limitation. Section 3 presents the RDR-Pipe micro-architecture, an extension

DAC 2004, June 7–11, 2004, San Diego, California, USA

to the RDR micro-architecture for automatic interconnect pipelining. Section 4 describes our proposed architectural synthesis methodology. In particular, we will focus on the global interconnect sharing algorithm. The experimental results are shown in Section 5, followed by conclusions in Section 6.

2. REVIEW OF THE RDR MICRO-ARCHITECTURE

The RDR micro-architecture [5] divides the entire chip into an array of islands. The registers are distributed to each island, and the size of each island is chosen such that all intra-island computation and communication can be performed in a single clock cycle. The inter-island communications can take multiple cycles.

Each island consists of the following components: (1) A local computational cluster (LCC) that contains functional elements of the circuit, such as multiplexors (MUX), multipliers, ALUs, etc. (2) A local register file that represents dedicated local storages. It can be partitioned into K banks (assuming that up to K cycles are needed to cross the chip), such that registers in bank i will hold the results for i cycles for communicating with another island that is i cycles away. (3) A finite state machine (FSM) that controls the behaviors of the computational elements and registers.

The RDR micro-architecture provides a regular synthesis platform for supporting multi-cycle on-chip communication. Its regularity greatly facilitates the predictability of interconnect delays at early design stages. Additionally, it offers a way to systematically explore the cycle time vs. latency tradeoff. According to the studies in [5][6][7], RDR exhibits a 31% better clock period and a 24% better total latency compared to the conventional approach.

However, the RDR micro-architecture may introduce a considerable amount of wiring overhead 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 the transmissions can be serialized.

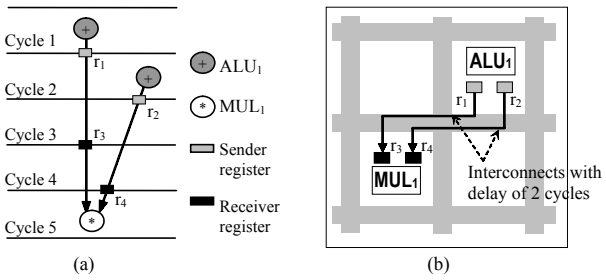

Figure 1. Illustration of the wiring overhead in RDR (a) Scheduled and bound CDFG, (b) RDR layout.

Figure 1 illustrates the problem using a very simple control data flow graph (CDFG) with three operation nodes. We assume a uniform node delay that is equal to the cycle time. Given the layout shown in Figure 1 (b), the CDFG will be scheduled into five cycles. Using the RDR micro-architecture, 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 inter-island wires are needed between ALU_1 and MUL_1 as their data transfer times overlap.

Clearly, more wiring overhead may be incurred as the designs grow. Since the global wires are an expensive resource, this deficiency has to be addressed.

3. RDR-PIPE MICRO-ARCHITECTURE FOR INTERCONNECT PIPELINING

For a K-cycle global interconnect, we observe that it is not necessary to hold the sender register constantly for K cycles. Instead, flip-flops can be inserted to the wire 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, the throughput of a pipelined interconnect can be up to K times greater than that of the non-pipelined one in RDR. In addition, more data transfers can share the same global wire, as the minimal launch interval is reduced from K to 1.

Based on the above consideration, we propose the *RDR-Pipe* micro-architecture which extends the RDR to enable an interconnect-pipelining scheme for the on-chip communication design.

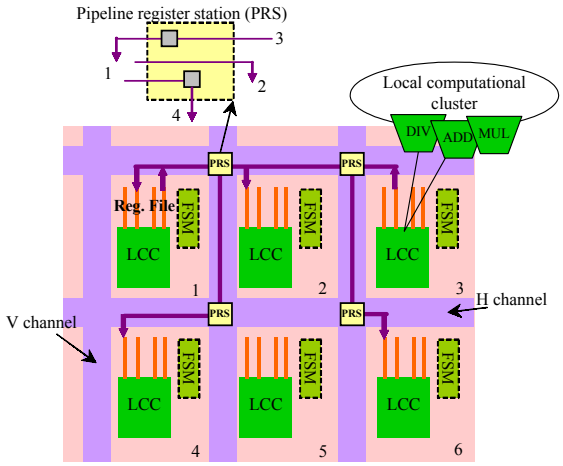

Figure 2. A 2×3 island-based RDR-Pipe micro-architecture.

Figure 2 illustrates a 2×3 island-based RDR-Pipe micro-architecture. RDR-Pipe also consists of an array of islands surrounded by the horizontal (H) and vertical (V) routing channels. The key difference between RDR-Pipe versus RDR is that pipeline registers are inserted on the global interconnects so that every K-cycle inter-island 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.

Figure 3 illustrates the advantages of the RDR-Pipe micro-architecture using the same CDFG example shown in Section 2. With 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 still emit the other value to r_1 at the second cycle, as v_1 has already been transferred to r_2. Therefore, the two data transfers can share the same interconnect, and only one global wire is needed this time between ALU_1 and

MUL_1. In fact, this result can be further generalized as follows: Under the RDR-Pipe micro-architecture, at most one inter-island global wire is needed between any pair of functional units,

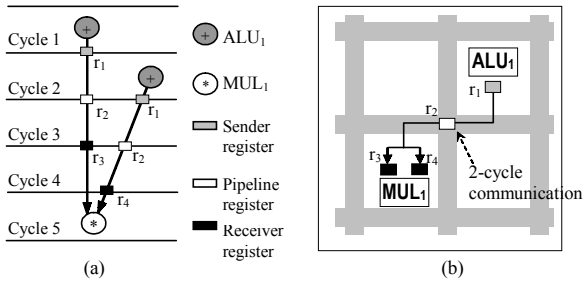

Figure 3. Illustration of the interconnect pipelining and sharing in RDR-Pipe (a) Scheduled and bound CDFG, (b) RDR-Pipe layout.

As shown above, the RDR-Pipe micro-architecture preserves the strength of the RDR micro-architecture to achieve high performance; meanwhile, it directly supports automatic interconnect pipelining and potentially allows better wiring utilization compared to the RDR.

4. PROPOSED ARCHITECTURAL SYNTHESIS METHODOLOGY

In this section we describe our proposed architectural synthesis methodology. In particular, we will focus on the global interconnect sharing algorithm.

4.1 Overall Design Flow

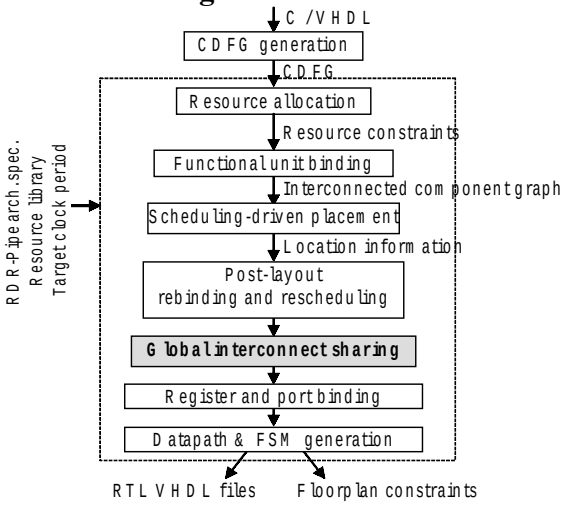

Figure 4. MCAS-Pipe design flow.

To efficiently synthesize the behavioral descriptions onto the RDR-Pipe micro-architecture, we extend the MCAS architectural synthesis system [6] to support the interconnect pipelining. Figure 4 shows the overall synthesis flow of the extended MCAS system called MCAS-Pipe.

MCAS-Pipe takes a behavioral-level description as input. In our case, this can be either synthesizable C or VHDL. The original MCAS modules, as shown in Figure 4, are used to derive a scheduled and bound CDFG and also the physical locations for the functional units. The common steps include CDFG resource allocation, initial functional unit binding, scheduling-driven placement, and post-layout rebinding and rescheduling.

A key module called global interconnect sharing is then performed to minimize the number of global wires, followed by the register allocation and port binding. This step will be discussed in detail in Section 4.2.

At the backend, MCAS-Pipe generates a datapath and distributed controller generation. In the same way as MCAS, the final outputs of MCAS-Pipe include RT-level VHDL files for logic synthesis tools and floorplan constraints for physical design tools. However, no multi-cycle path constraints will be generated by MCAS-Pipe due to the regular pipelining of all interconnects. Note that multi-cycle path constraints are used extensively in MCAS for multi-cycle on-chip communication.

4.2 Global Interconnect Sharing

This section describes the procedures used for global interconnect sharing, which aims to minimize the number of global wires.

In RDR-Pipe, the value of a variable may be transmitted from its producer island to several consumer islands, taking different numbers of clock cycles. The starting point of a data transfer may be flexible due to the possible slack between the actual transfer latency and the arrival-to-deadline interval. We can make use of this flexibility to schedule the data transfers to further reduce the global wires and pipeline registers needed for data communications among the islands.

4.2.1 Motivation

The input to the global interconnect sharing problem is a scheduled and bound CDFG where every operation is bound to a certain functional unit and is scheduled to a certain control step. Hereafter, we use *T(op)* to denote the control step where operation *op* is scheduled and *FU(op)* to denote the functional unit to which it is bound.

In a CDFG, a variable is produced by an operation node and consumed by one or more operation nodes. Every data edge between two operations represents a *data transfer* (or *transfer,* for short) from the *producer* operation to the *consumer* operation. We will not distinguish an edge and its corresponding data transfer hereafter. Let e be an edge (or a transfer) in a CDFG, and let p_e be the producer and c_e be the consumer of the transfer, $T(p_e)$ and $T(c_e)$ are the control steps in which the producer and consumer are scheduled respectively. The *active-interval* of data transfer e is the time period from $T(p_e)+1$ to $T(c_e)-1$, denoted as $[T(p_e)+1, T(c_e)-1]$. A transfer schedule is said to be feasible if it starts and finishes within its active-interval.

Under the RDR-Pipe micro-architecture, operation *op* is performed in island A (denoted as $op \in A$) if and only if *FU(op)* is located in A. Data communications from one island to another are through a *channel*, which is a set of *data links* implemented in pipelined global interconnects between the islands. The channel from islands A to B is denoted as a pair *(A, B)*, which is associated with a channel latency of *D(A, B)* cycles; i.e., the data links between A and B have *D(A, B)-1* pipeline stages. Therefore, transfer e should be issued in channel *(A, B)* with the latency of *D(A, B)* cycles, if and only if $p_e \in A$ and $c_e \in B$. A channel should accommodate all the data transfers $\{e \mid p_e \in A \text{ and } c_e \in B\}$.

Every cycle a data link can issue at most one transfer. Since every link is fully pipelined, as long as a transfer is issued to the first

pipeline stage on the link, it will be automatically forwarded through the subsequent pipeline stages. In other words, one transfer can be issued on a link in each clock cycle (i.e., the throughput of a link is one transfer per cycle). Therefore, the *effective occupancy time* of a transfer on a data link is exactly one cycle.

The *width of a channel* is defined as the number of links used to accomplish the data transfers on the channel. It is determined by the maximum number of simultaneous issues of the transfers.

Figure 5 shows a scheduled and bound CDFG and the corresponding RDR-Pipe layout. There are two edges e and g representing the transfers from p_e to c_e and from p_g to c_g, respectively. According to the operation schedule, the active-interval of transfer e is $[4, 6]$, and that of g is $[2, 6]$. Both producers belong to island A, and both consumers belong to island B. Suppose the channel latency $D(A, B)$ is two clock cycles, the transfers of e and g will then take two clock cycles. In this design, both data transfers occur on cycle 4; two data links are required to accomplish these transfers on channel (A, B).

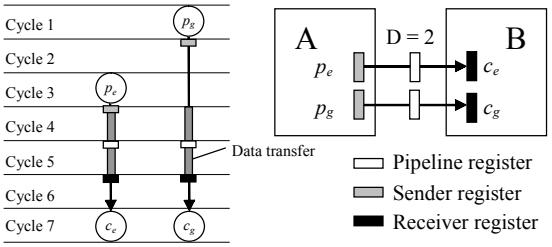

Figure 5. Simultaneous transfers using two data links.

However, if transfer g is issued on cycle 3 as shown in Figure 6, a shared data link is enough to perform transfers g and e in a pipelined manner, as illustrated by the RDR-Pipe layout on the right-hand side of Figure 5. Note that a multiplexor is introduced to share the data link, assuming the sender registers remain the same.

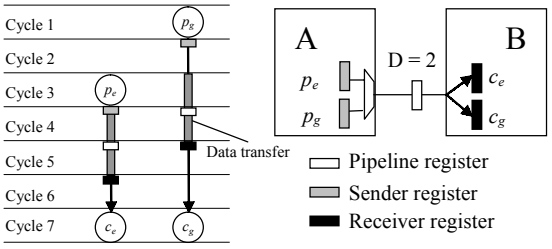

Figure 6. Scheduled transfers using only one data link.

4.2.2 Channel Width Minimization

Given the definitions of transfer and channel, we have the following observation: For every two channels (A_1, B_1) and (A_2, B_2), if $A_1 \neq A_2$ or $B_1 \neq B_2$, the channels accommodate disjoint sets of transfers. This is because that every operation can only belong to one island, thus every transfer can only belong to one channel.

Furthermore, since there are no steering logics and controls for pipeline registers outside of islands in the proposed RDR-Pipe micro-architecture, sharing data links between two channels is not allowed. Therefore, all channels are independent of each other.

THEOREM 1. Global pipelined interconnects are minimized if and only if the width of every channel is minimized.

Since data links are implemented by global pipelined interconnect, the total width of channels directly reflects the cost of global wiring and pipeline registers. The problem of minimizing the total global interconnects and pipeline registers can be transformed into a sequence of problems for minimizing the width of each individual channel.

4.2.3 Data Transfer Scheduling

The example shown Section 4.2.1 indicates that data link sharing can be realized by data transfer scheduling.

The transfer e with producer p_e and consumer c_e can be issued within $[T(p_e)+1, T(c_e)-D(A, B)]$, where $p_e \in A$ and $c_e \in B$, and the effective occupancy time of a transfer on a data link is only one cycle. Therefore, channel width minimization can be formulated as a transfer scheduling problem:

PROBLEM. Transfer Scheduling

Given: (1) A channel (A, B) of width m. (2) A data transfer set $\{e \mid p_e \in A \text{ and } c_e \in B\}$, where each transfer e is associated with an arrival time $T(p_e)+1$, a deadline $T(c_e)-D(A, B)$, and the unit effective occupancy time.

Assumption: For every time slot, at most one transfer can be issued on a data link.

Objective: To find a feasible transfer schedule on these data links.

If we view a data link as a processor and a transfer as a task with unit execution time, this problem is a special case of the deadline scheduling of tasks with ready times on processors, which is solvable in polynomial time by an earliest deadline first (EDF) algorithm [1].

```
Algorithm: Transfer Scheduling (X, m)
Objective:
  Check whether transfer set X can be scheduled to m data links.
Assumption:
  1) Every x_i of X has an arrival time a_i and a deadline d_i
  2) The minimal arrival time is T_0, and maximal deadline is T_m
Begin
  Sort X by the non-descending order of their deadline
  t := T_0
  while t ≤ T_m do
    A := ∅; k := 0
    for each unassigned x_i of X
        if d_i < t then return fail
        if a_i ≥ t then A = A ∪ {x_i}
    while A is not empty and k < m do
        Remove the first x_i from A
        Assign x_i to the next free data link
        k := k + 1
    t := t + 1
  return success
```

Figure 7. Transfer scheduling algorithm.

The pseudo-code of the algorithm is shown in Figure 7. First, the transfers are sorted by the non-descending order of their deadlines. Then repeatedly in every time slot, active unassigned tasks are scheduled to the data links according to this order. Notice A denotes the active unassigned task set. The algorithm will return a feasible schedule if one exists.

THEOREM 2. The algorithm is optimal for the transfer scheduling problem, with a run time complexity of $O(nlogn)$, where n is the number of the given transfers.

Table 1. Performance comparison of three alternative flows.

Designs	CONV / MCAS			MCAS			MCAS-Pipe / MCAS		
	CP	CS	LAT	CP (ns)	CS	LAT (ns)	CP	CS	LAT
PR	1.67	0.90	1.51	5.86	21	123.06	1.00	1.00	1.00
WANG	1.63	0.89	1.46	5.40	19	102.56	0.89	1.00	0.89
LEE	1.57	0.91	1.43	5.91	34	200.77	1.04	1.00	1.04
MCM	1.52	0.94	1.43	7.13	32	228.19	0.88	1.00	0.88
HONDA	1.43	0.88	1.26	7.35	50	367.65	1.00	1.00	1.00
DIR	1.52	0.91	1.38	7.12	55	391.38	0.97	1.00	0.97
Average	1.56	0.91	1.41	6.46	35.17	235.60	0.96	1.00	0.96

It is easy to show the equivalence of the transfer scheduling and the deadline scheduling of tasks with ready times. The original optimality proof of the latter problem can be found in [1]. The complexity is dominated by the sorting, which is $O(nlogn)$.

The solution to the whole global interconnect sharing problem is straightforward. First, we will construct the transfer tasks for each channel. For each channel, since the upper bound of its width is the number of transfers on it, we perform a binary search for the minimum number of data links required for the transfer set, using the transfer scheduling algorithm.

COROLLARY 1. The pipelined interconnect sharing problem can be solved optimally with run time complexity $O(Cnlog^2n)$, where C is the total number of channels, and n is the maximum number of transfers of all the channels.

4.3 Register Allocation Based on Minimized Channels

After channel minimization, pipeline registers are allocated for every data link according to its latency. At last, sender and receiver register sharing are performed, according to the variables' new lifetimes determined by operation scheduling, as well as transfer scheduling.

Since a variable's sender and receiver registers are located in the producer and consumer islands respectively, their lifetimes are split into several segments, one for sending in its producer island and the others for receiving in different consumer islands. For the example in Figure 6, the sending lifetime of a variable produced by p_g is *[2, 3]* and its receiving lifetime is *[5, 7]*. Similarly, the sending and receiving lifetimes for p_e are *[4, 4]* and *[6, 7]* respectively. In this example, variables produced by p_e and p_e can actually share one sender register since their sending lifetimes, *[4, 4]* and *[2, 3]*, are compatible. The multiplexor is also reduced due to the register sharing.

5. EXPERIMENTAL RESULTS

We implemented the MCAS-Pipe synthesis system in C++/UNIX environments. For comparison, we set up three alternative flows, which are conventional, MCAS, and MCAS-Pipe flows. The conventional high-level synthesis flow is based on the conventional architecture with a centralized register file and a global control. It performs the binding and list-scheduling algorithm sequentially without considering the layout. The MCAS flow, which is presented in [6], is built on top of the RDR micro-architecture, and the physical information is provided by the scheduling-driven placement. The MCAS-Pipe flow is based on the RDR-Pipe micro-architecture and is an extension to MCAS flow. It performs the global interconnect sharing algorithm to minimize the global wiring and number of pipeline registers. All three flows share the same backend to generate datapath and controllers. For MCAS flow, multi-cycle path constraints are also generated by the backend.

To obtain the final performance, wirelength, and area results, Altera's Quartus II version 3.0 [15] is used to implement the datapath and controllers into a real FPGA device,[1] Stratix™ EP1S40F1508C5. All the pipelined 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. In consistent with [7], we applied a 7×4 RDR-Pipe micro-architecture, and used LogicLock™ to constrain every instance into its corresponding island.

We use a set of data-intensive benchmarks to test our architectural synthesis system. All of them are from [13], including several DCT algorithms, such as PR, WANG, LEE, and DIR, and two DSP programs, MCM and HONDA.

The performance comparison results are shown in Table 1, where the results from MCAS flow, including clock period (CP), clock cycles (CS), and latency (LAT, the product of CP and CS), are listed in absolute values. The clock periods are reported by Quartus II, and clock cycles are determined by the CDFG scheduling algorithm. For comparison, the relative numbers of the conventional flow and MCAS-Pipe flow over MCAS results are listed. We can see that MCAS-Pipe matches or exceeds the MCAS flow in performance. Compared to the conventional flow, MCAS-Pipe achieves a 38% improvement in terms of clock period and a 30% reduction in total latency on average.

We collect the wirelength results for these designs after place-and-route. Eight types of general wires in Stratix devices are considered: *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. Table 2 shows the wirelength comparison of the MCAS-Pipe versus MCAS. In this table, absolute numbers of the total wirelengths for four wire type groups are listed for MCAS flow, followed by ratios of MCAS-Pipe flow over MCAS flow. Since the RDR-Pipe micro-architecture allows the automatic on-chip interconnect pipelining and the global interconnect sharing to further minimize the global wiring, we are able to share more data transfers on the same interconnect. Consequently, MCAS-Pipe

[1] Considering the availability of design tools, cell libraries and timing models, we chose the FPGA platform. We would expect similar or better experimental results in the ASIC platform, in which the interconnect delay versus gate delay ratio and communication overhead are higher.

Table 2. Wirelength comparison of MCAS-Pipe vs. MCAS.

Designs	MCAS					MCAS-Pipe / MCAS				
	LL+LO	H4+V4	H8+V8	V16+H24	Total	LL+LO	H4+V4	H8+V8	V16+H24	Total
PR	3304	14900	8472	26808	53484	0.95	0.84	0.97	0.69	0.79
WANG	3882	14880	11088	14368	44218	0.84	0.82	0.91	0.69	0.80
LEE	2649	11772	6904	17896	39221	0.89	0.80	0.62	0.89	0.82
MCM	7173	34076	20672	32240	94161	1.15	0.90	1.07	0.60	0.85
HONDA	7076	28560	18232	17312	71180	0.99	0.80	0.77	0.76	0.80
DIR	6138	26044	17904	20264	70350	1.06	0.94	0.85	0.64	0.84
Average	-	-	-	-	-	0.98	0.85	0.86	0.71	0.82

reduces 28.8% global wirelength (total length of wire type *V16* and *H24*) and 19.3% total wirelength on average.

Table 3 also lists the ratios of the resource used by different design flows in terms of logic elements (LEs) and registers. The MCAS-Pipe uses 9% more registers than MCAS flow, while slightly less LEs on average. The transfer scheduling changed the allocation of sender and receiver registers so that the multiplexor network structures are changed, which could affect the final area.

Table 3. Logic utilizations of three alternative flows.

Designs	CONV / MCAS		MCAS		MCAS-Pipe /	
	Reg#	LE	Reg#	LE	Reg#	LE
PR	0.71	0.95	31	1181	1.19	0.95
WANG	0.63	0.81	40	1435	1.20	0.85
LEE	0.76	0.96	29	988	1.00	0.84
MCM	0.75	1.00	57	2467	1.05	1.19
HONDA	0.83	0.90	41	2542	1.05	1.01
DIR	0.75	0.95	44	2260	1.05	1.01
Average	0.74	0.93	-	-	1.09	0.98

6. CONCLUSIONS

In this paper we present a high-level solution to automatic on-chip interconnect pipelining during the architectural synthesis stage. We propose the RDR-Pipe micro-architecture to extend the RDR micro-architecture for automatic interconnect pipelining. Also, we develop a novel global interconnect sharing algorithm to effectively reduce the global wiring. Experimental results show that our approach matches or exceeds the RDR-based approach in performance, with a greatly reduced wiring demand.

ACKNOWLEDGEMENTS

This research is partially funded by MARCO/DARPA Gigascale Silicon Research Center (GSRC), National Science Foundation under award CCR-0096383, and Altera Corporation under the California MICRO program. The authors thank the anonymous reviewers of [6] and [7] for pointing out the wiring overhead problem associated with the RDR architecture, and Dr. Vaughn Betz of Altera Corporation for providing the detailed instructions of experimentation on the Quartus II development system.

REFERENCES

[1] J. Blazewicz, "Deadline Scheduling of Tasks with Ready Times and Resource Constraints," *Information Processing Letters*, vol. 8(2), pp. 60-63, 1979.

[2] P. Chong and R. K. Brayton, "Characterization of Feasible Retimings," *Proc. of International Workshop on Logic and Synthesis,* pp. 1-6, Jun. 2001.

[3] P. Cocchini, "Concurrent Flip-Flop and Repeater Insertion for High Performance Integrated Circuits," *Proc. of International Conference on Computer Aided Design,* pp. 268-273, Nov. 2002.

[4] J. Cong and X. Yuan, "Multilevel Global Placement with Retiming," *Proc. of 40th Design Automation Conference*, pp. 208-213, Jun. 2003.

[5] J. Cong, Y. Fan, X. Yang and Z. Zhang, "Architecture and Synthesis for Multi-Cycle Communication," *Proc. of 2003 International Symposium on Physical Design*, pp. 190-196, Apr. 2003.

[6] J. Cong, Y. Fan, G. Han, X. Yang and Z. Zhang, "Architectural Synthesis Integrated with Global Placement for Multi-Cycle Communication," Proc. *of International Conference on Computer Aided Design*, pp. 536-543, Nov. 2003.

[7] J. Cong, Y. Fan, G. Han, X. Yang and Z. Zhang, "Architecture and Synthesis for On-Chip Multicycle Communication," to appear in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*.

[8] R. Lu, G. Zhong, C. Koh and K. Chao, "Flip-Flop and Repeater Insertion for Early Interconnect Planning," *Proc. of Design Automation and Test in European Conference*, pp. 690-695, Mar. 2002.

[9] R. McInerney, K. Leeper, T. Hill, H. Chan, B. Basaran and L. McQuiddy, "Methodology for Repeater Insertion Management in the RTL, Layout, Floorplan and Fullchip Timing Databases of the Itanium™ Microprocessor," *Proc. of 2000 International Symposium on Physical Design,* pp. 99-104, Apr. 2000.

[10] M. C. Papaefthymiou, "Understanding Retiming Through Maximum Average-Delay Cycles," *Mathematical Systems Theory*, vol. 27, pp. 65-84, 1994.

[11] L. Scheffer, "Methodologies and Tools for Pipelined On-Chip Interconnect," *Proc. of International Conference on Computer Design*, pp. 152-157, Sep. 2002.

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

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

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

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