

Architecture and Synthesis for Multi-Cycle On-Chip Communication (Extended Abstract)

Jason Cong, Yiping Fan, Guoling Han, Xun Yang, Zhiru Zhang

Computer Science Department, University of California, Los Angeles

Los Angeles, CA 90095, USA

{cong, fanyp, leohgl, yangxun, zhiruz}@cs.ucla.edu

1. INTRODUCTION

There are two important inflection points in the development of deep sub-micron (DSM) process technologies. The first point is when the average interconnect delay exceeds the gate delay, which happened during mid 1990s and led to the so-called timing closure problem. The second point is when single-cycle full chip synchronization is no longer possible, which is about to happen soon. It can be shown that, even with the aggressive interconnect optimization techniques (e.g., buffer insertion and wire-sizing), 5 clock cycles are still needed to go from corner-to-corner for the die of 28.3mm×28.3mm in the 0.07μm technology generation, assuming a 5.63GHz clock by 2006 predicted in ITRS'01 [5].

This clearly suggests that multi-cycle on-chip communication is a necessity in multi-gigahertz synchronous designs. However, it is not supported in the current design tools and methodologies, as most of these implicitly assume that full chip synchronization in a single clock cycle is feasible.

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

- 1) Asynchronous design: The state transitions of an asynchronous design are triggered by events instead of periodic clocks. This makes asynchronous designs operate correctly, regardless of the delays on gates and wires. However, due to the lack of design tools and performance overhead, it only applies to a very limited class of circuits. In general, it is unclear whether asynchronous designs can yield high performance in practice.
- 2) Global asynchronous locally synchronous (GALS) design: In a GALS design, all major modules are designed in accordance with proven synchronous clocking disciplines. Each module is run from its own local clock. Data exchange between any two modules strictly follows a full handshake protocol. GALS hopes to combine the advantages of synchronous and asynchronous design methodologies. However, the overhead for the “self-timed wrapper” may compromise both performance and area of the design.

- 3) Synchronous design with multi-cycle communication: Synchronous design is still by far the most popular design methodology. It is well understood and supported by the mature CAD toolset. However, the 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 multi-gigahertz design in nanometer technologies. In this paper, we present a new micro-architecture and a novel synthesis methodology for synchronous designs with multi-cycle communication.

Our contributions are as follows: (i) We propose a *Regular Distributed Register (RDR)* micro-architecture which offers high regularity and direct support of multi-cycle communication. (ii) We develop a set of novel architectural synthesis algorithms to efficiently synthesize behavior-level designs onto the RDR architecture.

2. REGULAR DISTRIBUTED REGISTER ARCHITECTURE

The RDR architecture 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. Figure 1 illustrates an example of RDR architecture.

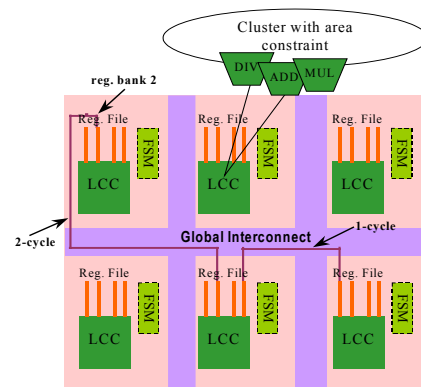


Figure 1. A 2×3 island-based RDR architecture.

Each island consists of the following components:

- 1) *Local Computational Cluster (LCC)*: The functional elements in an LCC provide the computational power of the circuit. They might be NAND gates, multiplexors (MUX), multipliers, or ALUs, etc. The size of the LCC is subject to a certain area constraint.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'03, October 1–3, 2003, Newport Beach, California, USA.

Copyright 2003 ACM 1-58113-742-7/03/0010...\$5.00.

- 2) *Register file*: The dedicated local storages reside in the register file, which can be partitioned into k banks (where k is the maximum number of cycles needed to communicate across 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) *Finite State Machine (FSM)*: Each island contains a local controller (i.e., an FSM) to control the behaviors of the computational elements and registers.

The RDR architecture is similar to recently proposed distributed-register architecture (DRA) [3] in the sense that both of these architectures distribute registers close to the local computational units, support multi-cycle communication, and allow concurrent computation and communication. However, unlike DRA, the RDR architecture is highly regular, which greatly facilitates the interconnect delay estimation. Interconnect delay can be easily estimated from the island indices of the source and the destination. Moreover, the RDR architecture has the advantage in that by varying the size of the basic island, one can easily target different clock periods and systematically explore the cycle time vs. latency tradeoff. More details of the RDR architecture are available in [1].

3. ARCHITECTURAL SYNTHESIS FOR MULTI-CYCLE COMMUNICATION

Our *architectural synthesis system for multi-cycle communication (MCAS)* is built on top of the RDR architecture. The overall design flow is shown in Figure 2.

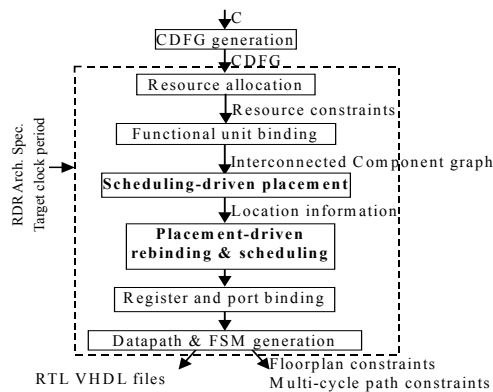


Figure 2. MCAS architectural synthesis system

Given the target clock period and the RDR architecture specification (including the island structure, functional unit library and delay table), MCAS starts with a synthesizable C description. It first generates the control data flow graph (CDFG) from the behavioral descriptions through an intermediate representation of the SUIF compiler [7].

At the front-end, MCAS performs resource allocation, followed by an initial functional unit binding. It uses the time-constrained force-directed scheduling (FDS) algorithm [4] to obtain the resource allocation. After the FDS-based resource allocation, it performs a similar algorithm in [3] to bind operation nodes to functional units for minimizing the number of potential data transfers among the components with the same types. Then an interconnected component graph (ICG) is derived from the bound CDFG.

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 architecture, 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 the placement-driven rescheduling-and-rebinding. The algorithm is based on the force-directed list-scheduling framework, and integrates simultaneous rebinding.

At the backend, MCAS performs register and port binding followed by datapath and distributed controllers generation. The final outputs of MCAS include the RT-level VHDL files for logic synthesis, floorplan and multi-cycle path constraints for place-and-rout. If the final design cannot meet the performance requirement, we can adjust the clock period and the basic island size by binary search and redo the synthesis.

The algorithmic details of the MCAS system are available in [2].

4. EXPERIMENTAL RESULTS

We implemented our MCAS system in C++/UNIX environments. Altera's Quartus II version 2.2 [6] is used to implement the datapath part into a real FPGA device, Stratix™ EP1S40F1508C5.

By comparing MCAS with the conventional architectural synthesis flow, which is based on the centralized register file architecture, we achieved a clock period improvement of 33% and a total latency improvement of 24% on average. We also did comparison with a commercial tool, Synopsys' Behavioral Compiler [8], and obtained the similar performance gain.

5. ACKNOWLEDGMENT

This research is partially sponsored by National Science Foundation under award CCR-0096383, MARCO/DARPA Gigascale Silicon Research Center, Semiconductor Research Center under 2001-TJ-910, and Altera Corporation under the California MICRO program.

6. REFERENCES

- [1] J. Cong, Y. Fan, X. Yang and Z. Zhang, "Architecture and Synthesis for Multi-Cycle Communication," in *Proceedings of 2003 International Symposium on Physical Design*, pp. 190-196, Apr. 2003.
- [2] J. Cong, Y. Fan, G. Han, X. Yang and Z. Zhang, "Architectural Synthesis Integrated with Global Placement for Multi-Cycle Communication," to appear in *Proc. of International Conference on Computer Aided Design*, Nov. 2003.
- [3] D. Kim, J. Jung, S. Lee, J. Jeon and K. Choi, "Behavior-to-Placed RTL Synthesis with Performance-Driven Placement," in *Proceedings of International Conference on Computer Aided Design*, pp. 320-326, Nov. 2001.
- [4] P. Paulin and J. Knight, "Force-Directed Scheduling for Behavioral Synthesis of ASICs," in *IEEE Trans. on CAD*, vol. 8(6), pp. 661-679, Jun. 1989.
- [5] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 2001.
- [6] Altera Web Site, <http://www.altera.com>.
- [7] SUIF Compiler, <http://suif.stanford.edu>.
- [8] Synopsys Web Site, <http://www.synopsys.com>.