Regular Distributed Register Fabric and Synthesis for Multi-Cycle Communications

Jason Cong, Yiping Fan, Xun Yang and Zhiru Zhang

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

Department of Computer Science
University of California, Los Angeles

Partially supported by NSF under award CCR-0096383, MARCO/DARPA GSRC, and Altera Corp. under the California MICRO program.
Outline

◆ Needs for Multi-Cycle On-Chip Communication

◆ Contributions
  - Regular Distributed Register (RDR) Architecture
  - MCAS: Architectural Synthesis for Multi-Cycle Communication
    - Scheduling-driven placement
    - Placement-driven rescheduling & rebinding

◆ Experimental Results

◆ Conclusions & Future Work
Needs for Multi-Cycle On-Chip Communication

- Interconnect delays dominate the timing in DSM tech.
- Single-cycle full chip synchronization is no longer possible

- NTRS’97 0.07um Tech
- 5 G Hz across-chip clock
- 620 mm² (24.9mm x 24.9mm)
- IPEM BIWS estimations
  - Buffer size: 100x
  - Driver/receiver size: 100x
- From corner to corner:
  - 7 clock cycles

Multi-Cycle Interconnect Communication at Logic / Physical Level

◆ Simultaneous retiming + placement / floorplanning
  - [Cong et al, ICCAD’00] [Cong et al, DAC’03]
  - [Chong & Brayton, IWLS’01]
  - [Singh & Brown, FPGA’02]

◆ Limitation:
  - Minimum clock period can be achieved by logic optimization is bounded by max. delay-to-register (DR) ratio of the loops in the circuits

- In a loop, 4 logic cells, 2 registers
- Cell delay = 1ns
- Interconnect delay = 4ns
- DR ratio = \((D_{\text{logic}} + D_{\text{int}})/\#\text{Registers}\) = \((4+16)/2=10\text{ns}\)
- Clock cycle \(\geq 10\text{ns}\)
Our Contributions

- **Regular Distributed Register (RDR) micro-architecture**
  - Highly regular
  - Direct support of multi-cycle on-chip communication

- **MCAS: Architectural Synthesis for Multi-cycle Communication**
  - Integrated architectural synthesis (e.g. resource binding, scheduling) with physical planning
  - Target at RDR architecture
Regular Distributed Register Architecture (1)

- Distribute registers to each “island”
- Chose the island size such that local computation and communication in each island can be done in a single cycle:

\[ D_{\text{intra-island}} = D_{\text{logic}} + D_{\text{opt-int}} \leq D_{\text{logic}} + 2D_{\text{opt-int}} (W_i + H_i) \leq T \]
Regular Distributed Register Architecture (2)

- Use register banks:
  - Registers in each island are partitioned to $k$ banks for 1 cycle, 2 cycle, ... $k$ cycle interconnect communication in each island
- Highly regular
Example: Regular Distributed Register Architecture for 70nm Technology

- NTRS'97 70nm Tech
- Chip dimension: 620 mm² (24.9mm x 24.9mm)
- 5 G Hz across-chip clock
  - Can travel up to 7.52mm within 1 clock cycle under best interconnect optimization
  - Need 7 clock cycles to cross the chip

- Each island base dimension
  - \( W_i = H_i = 2.08 \text{ mm} \)
  - \( \approx \frac{1}{3} \) of distance a wire can travel in 1 clock cycle
  - Logic volume: 6.76M min-size 2-NAND gates

- 12X12 array of islands
- Local registers are partitioned to 7 banks
RDR Architecture vs. DRA

- Distributed Register File Architecture (DRA)
  - Behavior-to-Placed RTL Synthesis with Performance-Driven Placement [Kim, et al, ICCAD’01]

- Similarities:
  - Distribute registers near the local computational units
  - Supports multi-cycle communication
  - Allows concurrent computation and communication

- Distinction:
  - The RDR architecture is highly regular
    - Facilitates interconnect delay estimation
    - Enables the systematic exploration of cycle-time/latency tradeoff by varying the size of the basic island
Example: Impact of Interconnect on Scheduling

- Data flow graph extracted from discrete cosine transformation (DCT)
- The nodes with the same color are assigned to the same functional unit.

<table>
<thead>
<tr>
<th>resource</th>
<th>delay</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplier</td>
<td>2 ns</td>
<td>2</td>
</tr>
<tr>
<td>alu</td>
<td>1 ns</td>
<td>2</td>
</tr>
</tbody>
</table>

Performance-driven Placement

Long interconnect
Short interconnect
Single-cycle vs. Multi-cycle Interconnect Communication

- **Single-cycle interconnect communication**
- Scheduled in 6 clock cycles
- Clock period is 4ns
- Total latency is 24ns

- **Multi-cycle interconnect communication**
- Scheduled in 9 clock cycles
- Clock period is 2ns
- Total latency is 18ns
With placement integrated with scheduling, critical path is reduced.
The DFG can be scheduled in 8 clock cycles, with clock period of 2ns.
The total latency is 16ns.
Enhancement 2: Simultaneous Placement, Scheduling and Binding for Performance Optimization

- With placement integrated with scheduling and binding, the critical path is further reduced.
- The DFG can be scheduled in 7 clock cycles, with clock period of 2ns.
- The total latency is 14ns
MCAS: Placement-Driven Architectural Synthesis Using RDR Architecture

- CDFG generation
- Resource allocation
- Functional unit binding
  - Interconnected Component Graph (ICG)
- Scheduling-driven placement
  - Location information
- Placement-driven rebinding & scheduling
  - Register and port binding
  - Datapath & FSM generation

- RTL VHDL files
- Floorplan constraints
- Multi-cycle path constraints

Target clock period

RDR Arch. Spec.
MCAS: Scheduling-Driven Placement (1)

◆ Basic approach:
  - Integrate scheduling with an SA-based coarse placement [Chang et al, ISPD’02]
  - Hide critical data transfers into intra-island by reducing weighted wirelength.

◆ Distinction between our placement and conventional performance-driven placement
  - Problem size: Relatively small (<10^3) vs. Huge
  - Input: ICG (general graph) vs. Netlist (acyclic graph)
  - Objective: To minimize: # of Clock cycles vs. Clock period
MCAS: Scheduling-Driven Placement (2)

- **Scheduling-based timing analysis**

  - **Timing Analysis** is performed on original CDFG instead of ICG
    - A fast list scheduling is performed on CDFG instead of the classical timing analysis at every temperature during the SA process to identify critical edges in ICG, and assign higher weights to them

- **Timing Analysis by Scheduling**

  
  ![Diagram of ICG Placement](Image of ICG Placement Diagram)
MCAS: Simultaneous Rescheduling & Rebinding (1)

- Simultaneous list scheduling and binding to minimize total schedule latency

- Previous Approach [Jeon et al, ASPDAC’01]
  - \( cpl(i, j) = \text{critical path length of fanout cone rooted at node } i, \text{ when node } i \text{ is bound to functional unit } j. \)
  - Perform list scheduling using priority function \( \min_j(cpl(i, j)). \)
  - Bind node to functional unit \( j \) with the minimum \( cpl(i, j) \) at the earliest feasible control step
Our contributions

- Use force-directed list scheduling and binding with interconnect delay estimation
- Consider resource constraints
  - During scheduling (for selecting deferred nodes)
  - During binding (as part of scheduling process)
Experiment Settings

C / VHDL

CDFG generation

CDFG

Functional unit allocation & binding

Interconnected component graph

Scheduling-driven placement

Location information

Scheduling

Placement-driven scheduling

Placement-driven rebinding & rescheduling

Register and port binding

Datapath & FSM generation

RTL VHDL files;
Floorplan constraints;
Multi-cycle path constraints

Altera FPGA development system
Experimental Results (1)

- Cycle number, clock period, and overall latency comparison

<table>
<thead>
<tr>
<th></th>
<th>Flow 1</th>
<th></th>
<th>Flow 2</th>
<th></th>
<th>Flow 3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CS (ns)</td>
<td>CP (ns)</td>
<td>Lat (ns)</td>
<td>CS (ns)</td>
<td>CP (ns)</td>
<td>Lat (ns)</td>
</tr>
<tr>
<td>pr</td>
<td>27</td>
<td>5.79</td>
<td>156.33</td>
<td>29</td>
<td>3.53</td>
<td>102.37</td>
</tr>
<tr>
<td>wang</td>
<td>14</td>
<td>7.54</td>
<td>105.56</td>
<td>20</td>
<td>4.14</td>
<td>82.8</td>
</tr>
<tr>
<td>lee</td>
<td>20</td>
<td>6.25</td>
<td>125</td>
<td>27</td>
<td>3.36</td>
<td>90.72</td>
</tr>
<tr>
<td>mcm</td>
<td>34</td>
<td>7.64</td>
<td>259.76</td>
<td>39</td>
<td>4.81</td>
<td>187.59</td>
</tr>
<tr>
<td>honda</td>
<td>23</td>
<td>7.58</td>
<td>174.34</td>
<td>24</td>
<td>3.78</td>
<td>90.72</td>
</tr>
<tr>
<td>dir</td>
<td>50</td>
<td>7.03</td>
<td>351.5</td>
<td>51</td>
<td>4.41</td>
<td>224.91</td>
</tr>
<tr>
<td>chem</td>
<td>50</td>
<td>8.27</td>
<td>413.5</td>
<td>53</td>
<td>4.64</td>
<td>245.92</td>
</tr>
<tr>
<td>u5ml12</td>
<td>68</td>
<td>9.3</td>
<td>632.4</td>
<td>70</td>
<td>5.34</td>
<td>373.8</td>
</tr>
<tr>
<td>Ave Ratio</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.14</td>
<td>0.57</td>
<td>0.65</td>
</tr>
</tbody>
</table>

- Flow 1: Conventional approach
- Flow 2: Scheduling-driven placement
- Flow 3: Scheduling-driven placement + placement-driven rebinding & rescheduling
**Experimental Results (2)**

- **Total latency comparison**

![Chart showing latency comparison for different flows.](chart.png)

- **Flow 1**: Conventional approach
- **Flow 2**: Scheduling-driven placement
- **Flow 3**: Scheduling-driven placement + placement-driven rebinding & rescheduling
Synopsys Flow – Behavioral Compiler vs. MCAS

Equal high-level data flow description

- VHDL
- Behavioral Compiler
- MCAS
- C
- RTL VHDL
- Design Compiler
- Stratix-Mapped VHDL
- Quartus
- Report
- VHDL output for simulation

gcc
Experimental Results (3)

- **MCAS basic flow vs. Synopsys’ Behavioral Compiler**

<table>
<thead>
<tr>
<th>Design</th>
<th>Flow</th>
<th>Cycles</th>
<th>Reg</th>
<th>ALU</th>
<th>MULT</th>
<th>fmax (MHz)</th>
<th>LE</th>
<th>Latency (ns)</th>
<th>MCAS vs. BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr</td>
<td>Synopsys BC</td>
<td>25</td>
<td>28</td>
<td>5</td>
<td>8</td>
<td>90.31</td>
<td>2945</td>
<td>276.82</td>
<td>100.82%</td>
</tr>
<tr>
<td></td>
<td>MCAS</td>
<td>27</td>
<td>34</td>
<td>6</td>
<td>2</td>
<td>96.74</td>
<td>2476</td>
<td>279.10</td>
<td></td>
</tr>
<tr>
<td>Wang</td>
<td>Synopsys BC</td>
<td>29</td>
<td>36</td>
<td>7</td>
<td>8</td>
<td>83.61</td>
<td>3605</td>
<td>346.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCAS</td>
<td>14</td>
<td>35</td>
<td>5</td>
<td>8</td>
<td>103.76</td>
<td>4242</td>
<td>134.93</td>
<td>38.90%</td>
</tr>
<tr>
<td>MCM</td>
<td>Synopsys BC</td>
<td>43</td>
<td>142</td>
<td>23</td>
<td>7</td>
<td>79.65</td>
<td>6253</td>
<td>539.86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCAS</td>
<td>34</td>
<td>35</td>
<td>6</td>
<td>3</td>
<td>72.05</td>
<td>3876</td>
<td>471.89</td>
<td>87.41%</td>
</tr>
<tr>
<td>Honda</td>
<td>Synopsys BC</td>
<td>29</td>
<td>44</td>
<td>8</td>
<td>14</td>
<td>85.14</td>
<td>6128</td>
<td>340.62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCAS</td>
<td>23</td>
<td>42</td>
<td>6</td>
<td>8</td>
<td>87.11</td>
<td>5523</td>
<td>264.03</td>
<td>77.52%</td>
</tr>
</tbody>
</table>

- Synopsys Behavioral Compiler setting: default (optimizing latency)
- Average latency ratio of MCASA vs. BC: 76%
Conclusions

- Multi-cycle communication is needed for multi-gigahertz designs

- Regular distributed register (RDR) architecture provides high regularity and direct support of
  - Multi-cycle communication
  - Integrated resource binding, scheduling, and physical planning

- Experimental results demonstrate the effectiveness of MCAS synthesis algorithms
Future Work

Support of control-intensive applications

- Distributed controller generation
- Variable renaming
  - Static Single Assignment (SSA) form

Steering logic optimization

- Multiplexer input count minimization
- Layout-driven distributed multiplexer tree generation