Hardware Synthesis from Coarse-Grained Dataflow Specification for Fast HW/SW Cosynthesis

Hyunuk Jung and Soonhoi Ha

Seoul National University
KOREA
Contents

• Introduction
• Previous Works & Motivation
• Issues
• Schedule Information and HW Controller Generation
• Fractional Rate Dataflow (FRDF)
• Experiments
• Conclusions
Introduction

- **Hardware Synthesis from Coarse-Grained Dataflow Specification**
  - A node represents a coarse grain computation block such as FIR filter or DCT.
  - A node has complex properties such as data sample rates, I/O timings, data types, and its internal states.
  - A central controller should be generated automatically in order to control these complex coarse-grain HW library blocks and registers.
Introduction

- **Hardware Synthesis for HW/SW Cosynthesis**
  - After HW/SW partitioning of an initial dataflow specification, a partitioned subgraph mapped to hardware is automatically generated.
  - A partitioned subgraph has interfacing blocks such as SND(send) and RCV(receive) blocks for communication.
    - These interfacing blocks have internal buffers and shared memory access logics.
Coarse-Grained Dataflow Specification

- Need for higher abstraction level
- Easy and intuitive specification method
- A proven methodology for signal processing and multimedia applications
  - formality and readability
- Software synthesis from dataflow graph is viable.
- Question:
  - Can we generate HW automatically from dataflow specification?
  - Can it be applied to practical designs?
Contents

• Introduction
• Previous Works & Motivation
• Issues
• Schedule Information and HW Controller Generation
• Fractional Rate Dataflow (FRDF)
• Experiments
• Conclusions
Previous Works

- 2-dimensional DCT algorithm

- Generated Hardware Architecture from Ptolemy
Previous Works

• Generated Hardware Architecture from Meyr’s works

![Diagram of hardware architecture with MUX, DCT 1D, and controller]

• Generated Hardware Architecture from GRAPE

![Diagram of hardware architecture with FIFO, MUX, DCT 1D, and controller]

FIFO with 64 buffers

Handshaking control signals
Motivation

- In the previous works, a single execution schedule is assumed for HW implementation.
- But, proposed approach allows the designer to provide the execution schedule: a multi-rate dataflow graph can be implemented into many hardware architectures.
Motivation

• Sharing

\[ \text{input} \xrightarrow{\text{MUX}} \text{DCT1D} \xrightarrow{\text{controller}} \text{output} \]

• multiple-instantiation
Proposed System Design Procedure

Dataflow Specification

Architecture Specification

Partitioning/Scheduling

Node-PE Performance DB

HW Subgraph

HW Schedule Info.

HW VHDL code generation

VHDL Code

SW subgraph

SW SW schedule

SW C code generation

C Code

Cosimulation
Contributions

1. **We automate HW synthesis** from a functional algorithm specification in dataflow model.
   - This enables fast design space exploration.

2. By separating the scheduling and HW synthesis, we can implement diverse HW architectures from the given dataflow specification.
   - It widens the design space of hardware implementation.

3. We synthesize the glue logics and the controller to make the HW operate preserving the dataflow semantics according to the schedule information.
   - Therefore, the synthesized hardware is correct by construction.
Contents

• Introduction
• Previous Works & Motivation
• Issues
• Schedule Information and HW Controller Generation
• Fractional Rate Dataflow (FRDF)
• Experiments
• Conclusions
ISSUES : Buffer Management

- Multi-rate & initial delay
- Data types
  - Int, Macroblock(16x16), Frame(176x144)
- Register : small data type
  - I/O timing control
  - Buffer allocation
- Memory : large data type
  - Memory access logic
  - Synchronization
  - Memory allocation
ISSUES: Resource Management

- Resource sharing or Multiple instantiation
- Input multiplexing and output buffer access

Diagram with symbols for Mux select signal and Output buffer latch signal.
ISSUES : Central Controller

• Distributed control vs. Central control
  – Distributed : local controller for each node & handshaking between all nodes
  – Central : A central controller for all nodes and buffers. Area-efficient

• Central control : Scheduling & Timing management for all blocks
  – Buffer control
    • Output register enable control
  – Resource control
    • Input MUX select signal
    • start signal of resources
  – Variable execution time control
    • done signal checking logic
Contents

• Introduction
• Previous Works & Motivation
• Issues
• Schedule Information and HW Controller Generation
• Fractional Rate Dataflow (FRDF)
• Experiments
• Conclusions
Schedule Information 1

# resource allocation table
Transpose  2
DCT1D     2

# resource mapping & schedule information
# (instance name, resource number, start, duration)
# loop (loop count, start, loop period)
Transpose_0  0  0  1
Loop  8  1  2 {
   DCT1D_0  0  0  2
}
Transpose_1  1  17  1
Loop  8  18  2 {
   DCT1D_1  1  0  2
}

1-to-1 mapping of graph node $\leftrightarrow$ HW resource

controller
Schedule Information 2: Sharing

# resource allocation table
Transpose  2
DCT1D     1

# resource mapping & schedule information
# (instance name, resource number, start, duration)
# loop ( loop count, start, loop period)
Transpose_0  0  0  1
Loop  8   1   2  {
    DCT1D_0  0  0  2
  }
Transpose_1  1  17  1
Loop  8  18  2  {
    DCT1D_1  0  0  2
  }

N-to-1 mapping of graph node ↔ HW resource
Schedule Information 3: Multiple Instantiation

# resource allocation table
Transpose 2
DCT1D 4

# resource mapping & schedule information
Transpose_0 0 0 1
Loop 4 1 2 {
  DCT1D_0 0 0 2
  DCT1D_0 1 0 2
}

Transpose_1 1 9 1
Loop 4 10 2 {
  DCT1D_1 2 0 2
  DCT1D_1 3 0 2
}

1-to-N mapping of graph node ↔ HW resource
Block Types & Control Signals

Type A: combinational logic
Type B: single-cycle sequential logic
Type C: multi-cycle sequential logic with fixed execution time
Type D: multi-cycle sequential logic with variable execution time
HW Controller Generation

- Counter-based Controller
  - Buffer control, Mux control, start and done signal of block

H. Jung, K. Lee and S. Ha
IEEE Transaction on VLSI Systems
Vol.10, August 2002
Remaining Problems

• The gap between automatic and manual design still exists.
  – We cannot optimize the automatic design further because of dataflow semantic.
  – Dataflow semantic has more strict rules for firing.
  – Real design has more freedom of implementation for efficient design

• A Solution \(\rightarrow\) Fractional Rate Dataflow (FRDF)
  – FRDF makes the automatic design a little closer to the manual design.
Fractional Rate Dataflow (FRDF)

- Non-FRDF implementation
  - “Add4” block is invoked after its all inputs are valid.
  - Parallel I/O
Fractional Rate Dataflow (FRDF)

- FRDF implementation
  - The execution of block “Add4” is divided into 4 phases.
  - Serial I/O at each phase
Contents

• Introduction
• Previous Works & Motivation
• Issues
• Schedule Information and HW Controller Generation
• Fractional Rate Dataflow (FRDF)
• Experiments
• Conclusions
Experiment 1: 2-dimensional DCT Algorithm

- 16 IDCT resources
- 4 IDCT resources
- 2 IDCT resources
- 1 IDCT resources: Sharing

Graph showing throughput (ns/sample) vs. area (gates) for different configurations.
Experiment 2: Parts of H.263 Decoder

<table>
<thead>
<tr>
<th>Design Type</th>
<th>Area (gates)</th>
<th>Latency (1 Macro Block)</th>
<th>Throughput (frame/ms) (CIF format)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Clock period (ns)</td>
<td>Cycles</td>
</tr>
<tr>
<td>Manual</td>
<td>124,762</td>
<td>20</td>
<td>1170</td>
</tr>
<tr>
<td>Auto</td>
<td>186,084</td>
<td>20</td>
<td>1188</td>
</tr>
</tbody>
</table>

- Design Type: Manual, Auto
- Area: Manual 124,762 gates, Auto 186,084 gates
- Latency: Manual 20 cycles, 1170 us, Auto 20 cycles, 1188 us
- Throughput: Manual 23.40 frames/ms, Auto 23.76 frames/ms

Diagram:
- DeQ
- Skip
- IZ
- IDCT
- Mux
- Saturation
- Truncation & ADD
- Mux
- WriteBlock
- SRAM
- 16bit 8x8 Block
- 8bit 8x8 Block
- FRAME
- 8bit integer
- dx dy
- 1/(6x99)
Conclusions

• This paper addresses how to synthesize various hardware structures from a DFG specification.
  – Considering resource sharing and looped schedule
  – Maintaining the performance and area efficiency level applicable to a practical design

• Future work
  – How to share buffers to reduce additional hardware area

• Our web site : PeaCE resource Homepage
  – http://peace.snu.ac.kr/research/peace
Thank you!