Efficient Hardware Controller Synthesis for Synchronous Dataflow Graph in System Level Design

Hyunuk Jung, Kangnyoung Lee and Soonhoi Ha

Seoul National University
KOREA
Contents

• Overview
  – System Design Process
  – Hardware Synthesis
  – Communication between Modules

• Interface Problem with the Outside

• Our approach
  – Basic idea
  – Cascaded counter controller

• Experiments

• Conclusions
Overview

• **Data Flow Graph (DFG) Specification**
  – Increasing need for a design methodology of higher abstraction level
  – *Easy and intuitive* specification method
  – A proven methodology among DSP research groups
    • formality and readability
System Design Process

• **Formal System Specification using SDF**
  – algorithm simulation
  – static analysis

• HW-SW Partitioning

• Synthesis
  – Hardware, Software and Interface code

DFG specification
System Design Process

• Formal System Specification using SDF
  – algorithm simulation
  – static analysis

• HW-SW Partitioning

• Synthesis
  – Hardware, Software and Interface code

DFG specification
System Design Process

- Formal System Specification using SDF
  - algorithm simulation
  - static analysis
- HW-SW Partitioning
- **Synthesis**
  - Hardware, Software and Interface code
System Design Process

- Formal System Specification using SDF
  - algorithm simulation
  - static analysis
- HW-SW Partitioning
- **Synthesis**
  - Hardware, Software and Interface code

DFG specification

- **Interface point**
Introduction

• Automatic hardware synthesis from data flow graph (DFG) specification in system level design
• Not only hardware synthesis itself but also interface problem with the outside of hardware module
Hardware Synthesis

- VHDL Code Generation
  - A node describes its functionality with a synthesizable VHDL code in the library.

```vhdl
entity FILTER1 is
  port (output : OUT STD_LOGIC_VECTOR(15 downto 0);
    input : IN STD_LOGIC_VECTOR(15 downto 0);
  );
end FILTER1;

architecture behavior of FILTER1 is
begin
  ....
end behavior;
```
Hardware Synthesis

• The hardware synthesis problem is to stitch the library VHDL codes into a whole.
  – Automatic synthesis of interface and control codes

entity HARDWARE1 is
port (
  output : OUT STD_LOGIC_VECTOR(15 downto 0);
  input_1 : IN STD_LOGIC_VECTOR(15 downto 0);
  input_2 : IN STD_LOGIC_VECTOR(15 downto 0);
);
end HARDWARE1;

architecture structure of HARDWARE1 is

Component NodeB
Component NodeC
Component NodeD
RCV1
RCV2
SND1
MUX
Register
Counter

end structure;
Hardware Synthesis

- Distributed approach (GRAPE tool)
  - handshaking protocol between node pairs
  - local control logic for each node
- Centralized approach (Ptolemy, Meyr’s work)
  - direct or registered connection between node pairs
  - centralized controller for all nodes and registers
Centralized Approach

- **Central control logic to implement data flow semantic**
- The execution timing of each node is decided by static schedule.
- Class of node implementations
  - Combinational logic
  - Single-cycle sequential logic
  - Multi-cycle sequential logic with fixed execution time
  - Multi-cycle sequential logic with variable execution time.
Centralized Approach

**Combinational logic**

Combinational logic can be connected directly. Only its execution time (propagation delay) is considered.

**Single-cycle sequential logic**

This logic is separated into combinational logic and state and implemented as Mealy machine.

```
A  Accumulator  C
|
A  ADDER
|
B
```

```
A  ADDER  C
|
State Register
|
Enable signal
|
Clock
|
Reset
```
Centralized Approach

Multi-cycle sequential logic with fixed execution time

We should provide **start** and **done** signal.

Central Controller generates signals according to static schedule.
Centralized Approach

Delay elements

Delay elements may exist and they correspond to data registers in hardware implementation.

[Diagram showing the flow of data from A to B to C to D, with an enable signal and a delay register, and a central controller.]
Communication between Modules

- The types of communication schemes
  - Synchronous communication
    - Communication timing is predetermined.
    - **Drawback**: Tasks should be scheduled assuming the worst case execution time.
  - Batch communication
    - It is possible to emulate synchronous communication with buffers in asynchronous interface.
    - **Drawback**: It cannot be applied to DFG with global feedback.
  - **Asynchronous communication**
    - Communication timing is varied at run-time.
  - There exist many cases in which asynchronous communication scheme is an efficient or a unique solution.
Interface Problem with the Outside

• Asynchronous communication with the outside of hardware module.
  – The communication timing is non-deterministic.
• Centralized control inside of hardware module.
  – The execution timing of each node is predetermined.
  – Centralized approach gives more efficient result.

• The asynchronous communication with the outside is not considered in the previous centralized approaches.
Our Approach Overview

- Main difference from hardware-only synthesis
  - supports the communication with outside modules such as software modules or other hardware modules

Special nodes for communication
- $\text{RCV}$: receive node
- $\text{SND}$: send node
Basic Idea

- **Counter-based solution**
  - simple and intuitive
Multiple RCV nodes

- Main goal
  - Obtaining the earliest time for the readiness of output regardless of the order in which the inputs arrive

- Valid timing equation of send node

\[ VT = \max_i (RT_i + D_i) \]

\( VT \): valid timing  
\( RT_i \): receive timing of \( i \)-th receive node  
\( D_i \): critical path length from the \( i \)-th receive node

Diagram:

- RCV1
- RCV2
- RCV3
- A → B
- B → D
- C → D
- D → SND

Edge labels:
- RCV1 to A: 20
- A to B: 10
- B to D: 30
- RCV2 to C: 20
- RCV3 to C: 20

Node labels:
- D1 = 60
- D2 = 40
- D3 = 50
Multiple RCV nodes

Valid timing equation of a send node

\[ VT = \max_i (RT_i + D_i) \]

RCV1 → count : 40

RCV2 → count : 50

decreasing counter

mux

40 50

(not RCV1) or RCV2

RCV1 or RCV2

load

Assign if greater than

V1 and V2

enable signal of send buffer

RCV1 → B

RCV2 → C

RCV1 → D

SND1
Multiple RCV nodes

Valid timing equation of a send node

\[ VT = \max_i (RT_i + D_i) \]
Cascaded Counter Controller

• Efficient controller that satisfies valid timing equation

\[ D_1 = 60 \]
\[ D_2 = 40 \]
\[ D_3 = 50 \]
Multiple SND nodes

- Overlapped cascaded counter
- Separate cascaded counters

\[
\begin{align*}
D_{1,1} &= 10 \\
D_{1,2} &= 40 \\
D_{2,2} &= 50
\end{align*}
\]
Nodes with Variable Execution Time

- The cascaded counter controller provides a clean solution for this node.

An asynchronous node that takes non-deterministic time unit for its execution

Modify!

A → B → C

A → SND → B → RCV → C

start → done

clock
Nodes with Variable Execution Time

- The cascaded counter controller provides a clean solution for this node.

Asynchronous communication is used only at the points necessary. This approach exploits the static schedule information at compile time as much as possible.
Experiment 1: DES

- **Approach**: Cascaded counter, FSM
- **Distributed Area overhead (control logic + buffer)**:
  - 50ns: (32+930)cell
  - 20ns: (32+930)cell
  - 5ns: (64+930)cell

- **Centralized Area**:
  - Cascaded counter
  - FSM
  - 3259 cell
Experiment 2

<table>
<thead>
<tr>
<th>Approach</th>
<th>Distributed</th>
<th>Centralized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cascaded counter</td>
</tr>
<tr>
<td>Area overhead (control logic + buffer)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50ns (20+27)cell</td>
<td>12 cell</td>
</tr>
<tr>
<td></td>
<td>20ns (24+27)cell</td>
<td>17 cell</td>
</tr>
<tr>
<td></td>
<td>5ns (32+27)cell</td>
<td>21 cell</td>
</tr>
</tbody>
</table>
Conclusions

• Contributions
  – We suggest a solution of the interface problem while hardware synthesis from DFG specification.
  – Our approach gives more efficient implementation compared to previous approaches.
  – Performance issues of DFG approach are intensively discussed.

• Future work
  – How to determine the reasonable execution times of library blocks
Thank you!