Embedded System Design
Modeling, Synthesis, Verification

Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner

Chapter 6: Hardware Synthesis

Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- High-level synthesis
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions
**HW Synthesis Design Flow**

- Compilation
- Estimation
- HLS
- Model generation
- Logic synthesis
- Layout

**Hardware Synthesis**

- Design flow
  - RTL architecture
    - Input specification
    - Specification profiling
    - High-level synthesis
    - Chaining and multi-cycling
    - Data and control pipelining
    - Scheduling
    - Component interfacing
    - Conclusions
**RTL Architecture**

- **Controller**
  - FSM controller
  - Programmable controller
- **Datapath components**
  - Storage components
  - Functional units
  - Connection components
- **Pipelining**
  - Functional unit
  - Datapath
  - Control
- **Structure**
  - Chaining
  - Multicycling
  - Forwarding
  - Branch prediction
  - Caching

**RTL Architecture with FSM Controller**

- **Simple architecture**
- **Small number of states**
RTL Architecture with Programmable Controller

- Complex architecture
  - Control and datapath pipelining
  - Advanced structural features
- Large number of states (CW or IS)

Hardware Synthesis

- Design flow
- RTL architecture
  - Input specification
    - Specification profiling
    - High-level synthesis
    - Chaining and multi-cycling
    - Data and control pipelining
    - Scheduling
    - Component interfacing
    - Conclusions
Input Specification

- **Programming language (C/C++, …)**
  - Programming semantics require pre-synthesis optimization
- **System description language (SystemC, …)**
  - Simulation semantics require pre-synthesis optimization
- **Control/Data flow graph (CDFG)**
  - CDFG generation requires dependence analysis
- **Finite state machine with data (FSMD)**
  - State interpretation requires some kind of scheduling
- **RTL netlist**
  - RTL design that requires only input and output logic synthesis
- **Hardware description language (Verilog / VHDL)**
  - HDL description requires RTL library and logic synthesis

C Code for Ones Counter

- **Programming language semantics**
  - Sequential execution,
  - Coding style to minimize coding
- **HW design**
  - Parallel execution,
  - Communication through signals

```c
int OnesCounter(int Data) {
    int Ocount = 0;
    int Temp, Mask = 1;
    while (Data > 0) {
        Temp = Data & Mask;
        Ocount = Ocount + Temp;
        Data >>= 1;
    }
    return Ocount;
}
```

01: while(1) {
02:     while (Start == 0);
03:     Done = 0;
04:     Data = Input;
05:     Ocount = 0;
06:     Mask = 1;
07:     while (Data>0) {
08:         Temp = Data & Mask;
09:         Ocount = Ocount + Temp;
10:         Data >>= 1;
11:     }
12:     Output = Ocount;
13:     Done = 1;
14: }

Function-based C code

RTL-based C code
CDFG for Ones Counter

- **Control/Data flow graph**
  - Resembles programming language
    - Loops, ifs, basic blocks (BBs)
  - Explicit dependencies
    - Control dependencies between BBs
    - Data dependences inside BBs
  - Missing dependencies between BBs

---

FSMD for Ones Counter

- **FSMD more detailed than CDFG**
  - States may represent clock cycles
  - Conditionals and statements executed concurrently
    - All statement in each state executed concurrently
    - Control signal and variable assignments executed concurrently
- **FSMD includes scheduling**
  - FSMD doesn’t specify binding or connectivity

---
**Chapter 6: Hardware Synthesis**

**RTL Specification for Ones Counter**

- **RTL Specification**
  - Controller and datapath netlist
  - Input and output tables for logic synthesis
  - RTL library needed for netlist

### Input Logic Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Start</th>
<th>Data = 0</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>X</td>
<td>S0</td>
<td>X</td>
</tr>
<tr>
<td>S9</td>
<td>1</td>
<td>X</td>
<td>S1</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>X</td>
<td>S2</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>X</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>X</td>
<td>S4</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>X</td>
<td>X</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S5</td>
<td>X</td>
<td>X</td>
<td>S6</td>
<td>0</td>
</tr>
<tr>
<td>S6</td>
<td>X</td>
<td>0</td>
<td>S7</td>
<td>0</td>
</tr>
<tr>
<td>S7</td>
<td>X</td>
<td>X</td>
<td>S0</td>
<td>1</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>State</th>
<th>RF Read Port A</th>
<th>RF Read Port B</th>
<th>ALU</th>
<th>Shifter</th>
<th>RF selector</th>
<th>RF Write</th>
<th>Outport</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Import</td>
<td>RF[0]</td>
<td>Z</td>
</tr>
<tr>
<td>S6</td>
<td>RF[0]</td>
<td>X</td>
<td></td>
<td>pass</td>
<td>B3</td>
<td>RF[5]</td>
<td>Z</td>
</tr>
<tr>
<td>S7</td>
<td>RF[2]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>disable</td>
<td>enable</td>
<td></td>
</tr>
</tbody>
</table>
HDL description of Ones Counter

• HDL description
  • Same as RTL description
  • Several levels of abstraction
    • Variable binding to storage
    • Operation binding to FUs
    • Transfer binding to connections

• Partial HLS may be needed
  • Controller and datapath netlists must be generated

```vhdl
01: // –
02: always @(posedge clk)
03: begin
04:   output_logic
05:   // –
06:   case (state)
07:     S4: begin
08:       B1 = RF[0];
09:       B2 = RF[1];
10:      B3 = alu(B1, B2, l_and);
11:     RF[3] = B3;
12:     next_state = S5;
13:   end
14:   // –
15:   S7: begin
16:     B1 = RF[2];
17:     Outport <= B1;
18:     done <= 1;
19:     next_state = S0;
20:   end
21: endcase
22: end
endmodule
```

Hardware Synthesis

✓ Design flow
✓ RTL architecture
✓ Input specification
  • Specification profiling
    • High-level synthesis
    • Chaining and multi-cycling
    • Data and control pipelining
    • Scheduling
    • Component interfacing
  • Conclusions
Profiling and Estimation

- Pre-synthesis optimization
- Preliminary scheduling
  - Simple scheduling algorithm
- Profiling
  - Operation usage
  - Variable life-times
  - Connection usage
- Estimation
  - Performance
  - Cost
  - Power

Square-Root Algorithm (SRA)

- SQR = max ((0.875x + 0.5y), x)
  - x = max (|a|, |b|)
  - y = min (|a|, |b|)
Variable and Operation Usage

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t1</td>
<td>t2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t3</td>
<td>t4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No. of live variables: 2 2 2 3 3 2 1

Max. no. of units:
- abs: 2
- min: 1
- max: 1
- >>: 2
- -: 1
- +: 1

Operation usage:
- S1: t1 = |a|
- S2: t2 = |b|
- S3: t5 = x – t3
- S4: x = max( t1, t2 )
- S5: y = min( t1, t2 )
- S6: t3 = x >> 3
- S7: t4 = y >> 1

No. of operations:
- abs: 2
- min: 1
- max: 1
- >>: 2
- -: 1
- +: 1

Connectivity usage:

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>t1</td>
<td>t2</td>
<td>x</td>
<td>y</td>
<td>t3</td>
</tr>
<tr>
<td>abs1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;&gt;3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;&gt;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No. of operations:
- abs1: 1
- abs2: 1
- min: 1
- max: 1
- >>3: 1
- >>1: 1
- +: 1

Max. no. of units:
- abs: 2
- min: 1
- max: 1
- >>: 3
- -: 1
- +: 1

Operation usage:
- S1: t1 = |a|
- S2: t2 = |b|
- S3: x = max( t1, t2 )
- S4: y = min( t1, t2 )
- S5: t3 = x >> 3
- S6: t4 = y >> 1
- S7: Done = 1

Out = t7
Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
  - High-level synthesis
  - Chaining and multi-cycling
  - Data and control pipelining
  - Scheduling
  - Component interfacing
  - Conclusions

Datapath Synthesis

- Variable Merging (Storage Sharing)
- Operation Merging (FU Sharing)
- Connection Merging (Bus Sharing)
- Register merging (RF sharing)
- Chaining and Multi-Cycling
- Data and Control Pipelining
Gain in register sharing

- **Register sharing**
  - Grouping variables with non-overlapping lifetimes
  - Sharing reduces connectivity cost

General partitioning algorithm

- **Compatibility graph**
  - Compatibility:
    - Non-overlapping in time
    - Not using the same resource
  - Non-compatible:
    - Overlapping in time
    - Using the same resource

- **Priority**
  - Critical path
  - Same source, same destination
Variable Merging for SRA

(a) Initial compatibility graph

(b) Compatibility graph after merging t3, t5, and t6

(c) Compatibility graph after merging t1, x, and t7

(d) Compatibility graph after merging t2 and y

(e) Final compatibility graph

(f) Final register assignments

\[
\begin{align*}
R1 &= [ a, t1, x, t7 ] \\
R2 &= [ b, t2, y, t3, t5, t6 ] \\
R3 &= [ t4 ]
\end{align*}
\]
Variable Merging for SRA

(a) Initial compatibility graph
(b) Compatibility graph after merging t3, t5, and t6
(c) Compatibility graph after merging t1, x, and t7
(d) Compatibility graph after merging t2 and y
(e) Final compatibility graph

R1 = \{ a, t1, x, t7 \}
R2 = \{ b, t2, y, t3, t5, t6 \}
R3 = \{ t4 \}

Datapath with Shared Registers

- Variables combined into registers
- One functional unit for each operation

\begin{array}{c|c|c|c|c|c}
 a & b & \text{min} & \text{max} & \gg 1 & \gg 3 \\
\end{array}

\text{Selector} \\
\text{R1} \\
| a | \\
\text{Selector} \\
\text{R2} \\
| b |
Gain in Functional Unit Sharing

- Functional unit sharing
  - Smaller number of FUs
  - Larger connectivity cost

\[ x = a + b \]
\[ y = c - d \]

Partial FSMD
Non-shared design
Shared design

Operation Merging for SRA

Initial compatibility graph
Compatibility graph after merging of + and -
**Operation Merging for SRA**

Initial compatibility graph

Compatibility graph after merging of + and -

Compatibility graph after merging of min, + and -

Final graph partitions

**Datapath with Shared Registers and FUs**

- Variables combined into registers
- Operations combined into functional units
Connection usage for SRA

Connection Merging for SRA

• Combine connection not used at the same time
  • Priority to same source, same destination
  • Priority to maximum groups
Datapath with Shared Registers, FUs and Buses

- Minimal SRA architecture
  - 3 registers
  - 4 (2) functional units
  - 4 buses

Register Merging into RFs

- Register merging: Port sharing
  - Merge registers with non-overlapping access times
  - No of ports is equal to simultaneous read/write accesses

R1 = [a, t1, x, t7]
R2 = [b, t2, y, t3, t6]
R3 = [c]

Register assignment

Compatibility graph

Register access table

1. a = In1
2. b = In2
3. x = max(t1, t2)
4. y = min(t1, t2)
5. t3 = x >> 3
6. t4 = y >> 1
7. t5 = x – t3
8. t6 = t4 + t5
9. t7 = max(t6, x)
10. Done = 1
11. Out = t7
Datapath with Shared RF

• RF minimize connectivity cost by sharing ports

Hardware Synthesis

✓ Design flow
✓ RTL architecture
✓ Input specification
✓ Specification profiling
✓ High-level synthesis
  • Chaining and multi-cycling
  • Data and control pipelining
  • Scheduling
  • Component interfacing
  • Conclusions
Datapath with Chaining

- Chaining connects two or more FUs
- Allows execution of two or more operations in a single clock cycle
- Improves performance at no cost

Datapath with Chained and Multi-Cycled FUs

- Multi-cycling allows use of slower FUs
- Multi-cycling allows faster clock-cycle
Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- High-level synthesis
- Chaining and multi-cycling
  - Data and control pipelining
  - Scheduling
  - Component interfacing
  - Conclusions

Pipelining

- **Functional Unit pipelining**
  - Two or more operation executing at the same time
- **Datapath pipelining**
  - Two or more register transfers executing at the same time
- **Control Pipelining**
  - Two or more instructions generated at the same time
Functional Unit Pipelining (1)

- Operation delay cut in "half"
- Shorter clock cycle
- Dependencies may delay some states
- Extra NO states reduce performance gain

Timing diagram with 4 additional NO states

<table>
<thead>
<tr>
<th>States</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read R1</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>Read R2</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
</tr>
<tr>
<td>Read R3</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>ALU stage 1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>f1</td>
<td>g1</td>
<td>h1</td>
<td>i1</td>
</tr>
<tr>
<td>ALU stage 2</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e2</td>
<td>f2</td>
<td>g2</td>
<td>h2</td>
<td>i2</td>
</tr>
<tr>
<td>Shifters</td>
<td>a3</td>
<td>b3</td>
<td>c3</td>
<td>d3</td>
<td>e3</td>
<td>f3</td>
<td>g3</td>
<td>h3</td>
<td>i3</td>
</tr>
<tr>
<td>Write R1</td>
<td>a4</td>
<td>b4</td>
<td>c4</td>
<td>d4</td>
<td>e4</td>
<td>f4</td>
<td>g4</td>
<td>h4</td>
<td>i4</td>
</tr>
<tr>
<td>Write R2</td>
<td>a5</td>
<td>b5</td>
<td>c5</td>
<td>d5</td>
<td>e5</td>
<td>f5</td>
<td>g5</td>
<td>h5</td>
<td>i5</td>
</tr>
<tr>
<td>Write R3</td>
<td>a6</td>
<td>b6</td>
<td>c6</td>
<td>d6</td>
<td>e6</td>
<td>f6</td>
<td>g6</td>
<td>h6</td>
<td>i6</td>
</tr>
<tr>
<td>Write Out</td>
<td>a7</td>
<td>b7</td>
<td>c7</td>
<td>d7</td>
<td>e7</td>
<td>f7</td>
<td>g7</td>
<td>h7</td>
<td>i7</td>
</tr>
</tbody>
</table>

Functional Unit Pipelining (2)
Datapath Pipelining (1)

• Register-to-register delay cut in “equal” parts
• Much shorter clock cycle
• Dependencies may delay some states
• Extra NO states reduce performance gain

Datapath pipelining (2)

Timing diagram with additional NO clock cycles
Datapath and Control Pipelining (1)

- Fetch delay cut into several parts
- Shorter clock cycle
- Conditionals may delay some states
- Extra NO states reduce performance gain

![Datapath and Control Pipelining Diagram](image)

Data and Control Pipelining (2)

- 3 NO cycles for the branch
- 2 NO cycles for data dependence

![Data and Control Pipelining Diagram](image)

Timing diagram with additional NO clock cycles

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Operation</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Read CMem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Read RF(L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Read RF(R)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Write ALUOut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Write SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Write PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Read CMem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Read RF(L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Read RF(R)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Write ALUOut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Write SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Write PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Write ALUIn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Write ALUOut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Write SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Write PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Write ALUIn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Write ALUOut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Write SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Write PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- High-level synthesis
- Chaining and multi-cycling
- Data and control pipelining
  - Scheduling
  - Component interfacing
  - Conclusions

Scheduling

- Scheduling assigns clock cycles to register transfers
- Non-constrained scheduling
  - ASAP scheduling
  - ALAP scheduling
- Constrained scheduling
  - Resource constrained (RC) scheduling
    - Given resources, minimize metrics (time, power, ...)
  - Time constrained (TC) scheduling
    - Given time, minimize resources (FUs, storage, connections)
C and CDFG for SRA Algorithm

1. Initialize:
   - \( a = \text{In1} \)
   - \( b = \text{In2} \)

2. Calculate:
   - \( t1 = |a| \)
   - \( t2 = |b| \)
   - \( x = \max(t1,t2) \)
   - \( y = \min(t1,t2) \)
   - \( t3 = x >> 3 \)
   - \( t4 = y >> 1 \)
   - \( t5 = x - t3 \)
   - \( t6 = t4 + t5 \)
   - \( t7 = \max(t6,x) \)

3. Set Done:
   - \( \text{Done} = 1 \)

4. Output:
   - \( \text{Out} = t7 \)

ASAP and ALAP Scheduling

1. Initialize:
   - \( \text{In1, In2} \)

2. Schedule:
   - ASAP schedule
   - ALAP schedule

3. Output:
   - \( \text{Out} \)
   - \( \text{Done} \)
RC Scheduling

Perfrom ASAP
Perfrom ALAP
Determine mobilities
Create ready list
Sort ready list by mobilities
Schedule ops from ready list
Delete scheduled ops from ready list
Add new ops to ready list
Increment state index
All ops scheduled?
Yes

Create ready list
Sort ready list by mobilities
Schedule ops from ready list
Delete scheduled ops from ready list
Add new ops to ready list
Increment state index
All ops scheduled?
No

Embedded System Design © 2009: Gajski, Abdi, Gerstlauer, Schirner
Chapter 6: Hardware Synthesis
5/25/2010 53

<Diagram of RC Scheduling流程>
**TC Scheduling**

1. Perform ASAP
2. Perform ALAP
3. Determine mobilities
4. Create probability distribution graphs

Schedule op with maximum gain
Schedule op with minimum loss

Initial probability distribution graph

**Distribution Graphs for TC scheduling**

Initial probability distribution graph
Graph after max, +, and – were scheduled
Distribution Graphs for TC scheduling

<table>
<thead>
<tr>
<th>AU units</th>
<th>Probability sum/state</th>
<th>Shift units</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1.33</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1.33</td>
<td>&gt;&gt;3</td>
</tr>
<tr>
<td>S4</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

Graph after max, +, and – were scheduled

<table>
<thead>
<tr>
<th>AU units</th>
<th>Probability sum/state</th>
<th>Shift units</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

Graph after max, +, -, min, >>3, and >>1 were scheduled

Distribution graph for final schedule
TC Scheduling

Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- High-level synthesis
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
  - Component interfacing
  - Conclusions
Interface Synthesis

- Combine process and channel codes
- HW and protocol clock cycles may differ
- Insert a bus-interface component
- Communication in three parts:
  - Freely schedulable code
    - Scheduled with process code
  - Schedule constrained code
    - MAC driver from library for selected bus interface
  - Bus interface
    - Implemented by bus interface component from library

Bus Interface Controller (1)
**Bus Interface Controller (2)**

- **ready = 0**
- **ready = 1**
- **OutAddr = BusAddr**
- **OutData = BusData**
- **OutCntl = WRITE_WORD**
- **ack = 1**
- **ready = 0**
- **ack = 0**

MAC driver

---

**Transducer/ Bridge**

- Translates one protocol into another
- Controller1 receives data with protocol1 and writes into queue
- Controller2 reads from queue and sends data with protocol2

---
Conclusion

• **Synthesis techniques**
  • Variable Merging (Storage Sharing)
  • Operation Merging (FU Sharing)
  • Connection Merging (Bus Sharing)

• **Architecture techniques**
  • Chaining and Multi-Cycling
  • Data and Control Pipelining
  • Forwarding and Caching

• **Scheduling**
  • Metric constrained scheduling

• **Interfacing**
  • Part of HW component
  • Bus interface unit

• **If too complex, use partial order**