Chapter 6: Hardware Synthesis

Hardware Synthesis

• Design flow
• RTL architecture
• Input specification
• Specification profiling
• RTL synthesis
  • Variable merging (Storage sharing)
  • Operation Merging (FU sharing)
  • Connection Merging (Bus sharing)
• Chaining and multi-cycling
• Data and control pipelining
• Scheduling
• Component interfacing
• Conclusions
HW Synthesis Design Flow

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

Hardware Synthesis

✓ Design flow
  • RTL architecture
  • Input specification
  • Specification profiling
  • RTL synthesis
    • Variable merging (Storage sharing)
    • Operation Merging (FU sharing)
    • Connection Merging (Bus sharing)
  • Chaining and multi-cycling
  • Data and control pipelining
  • Scheduling
  • Component interfacing
  • Conclusions
Chapter 6: Hardware Synthesis

**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
  - RTL synthesis
    - Variable merging (Storage sharing)
    - Operation Merging (FU sharing)
    - Connection Merging (Bus sharing)
  - Chaining and multi-cycling
  - Data and control pipelining
  - Scheduling
  - Component interfacing
  - Conclusions
Input Specification

- **Programming language (C/C++, ...)**
  - Programming semantics requires pre-synthesis optimization
- **System description language (SystemC, ...)**
  - Simulation semantics requires 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

```
01: int OnesCounter(int Data)
02: { int Ocount = 0;
03: int Temp, Mask = 1;
04: while (Data > 0) {
05: Temp = Data & Mask;
06: Ocount = Ocount + Temp;
07: Data >>= 1;
08: }
09: return Ocount;
10: }
```

```
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 dependences 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
CDFG and FSMD for Ones Counter

RTL Specification for Ones Counter

• RTL Specification
  • Controller and datapath netlist
  • Input and output tables for logic synthesis
  • RTL library needed for netlist
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
- **Netlist must be synthesized**
- **Partial HLS may be needed**

```vhdl
01:  // ...
02:  always@(posedge clk)
03:  begin : output_logic
04:  case (state)
05:    // ...
06:    S4: begin
07:      B1 = RF[0];
08:      B2 = RF[1];
09:      B3 = alu(B1, B2, l_and);
10:     RF[3] = B3;
11:     next_state = S5;
12:   end
13:  end case
14: end
```

Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
  - **Specification profiling**
  - RTL synthesis
    - Variable merging (Storage sharing)
    - Operation Merging (FU sharing)
    - Connection Merging (Bus sharing)
  - 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)

- $SQR = \max \left( (0.875x + 0.5y), x \right)$
  - $x = \max \left( |a|, |b| \right)$
  - $y = \min \left( |a|, |b| \right)$
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>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>b</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>x</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>t7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Variable 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>abs</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>min</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>max</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>&gt;&gt;3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&gt;&gt;1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation usage

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>abs1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>abs2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>min</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>max</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&gt;&gt;3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&gt;&gt;1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
  - **RTL synthesis**
    - Variable merging (Storage sharing)
    - Operation Merging (FU sharing)
    - Connection Merging (Bus sharing)
  - 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 \( t_3, t_5, \) and \( t_6 \)

(c) Compatibility graph after merging \( t_1, x, \) and \( t_7 \)

(d) Compatibility graph after merging \( t_2 \) and \( y \)

(e) Final compatibility graph

(f) Final register assignments

---

**Datapath with Shared Registers**

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

![Datapath Diagram]
Gain in Functional Unit Sharing

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

Operation Merging for SRA
Datapath with Shared Registers and FUs

- Variables combined into registers
- Operations combined into functional units

Connection usage for SRA

• Find compatible connections for merging into buses
Connection Merging for SRA

• Combine connection not used at the same time
  • Priority to same source, same destination
  • Priority to maximum groups

> Compatibility graph for input buses

> Compatibility graph for output buses

<table>
<thead>
<tr>
<th>S0</th>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>B</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>H</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>I</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>J</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>K</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>L</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>M</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>N</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>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Bus assignment:
- Bus1 = [A, C, D, E, H]
- Bus2 = [B, F, G]
- Bus3 = [I, K, M]
- Bus4 = [J, L, N]

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

Datapath with Shared RF

- RF minimize connectivity cost by sharing ports
Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- 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
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- Chaining and multi-cycling
  - Data and control pipelining
  - Scheduling
  - Component interfacing
  - Conclusions
Pipelining

- **Functional Unit pipelining**
  - Two or more operations 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

![Functional Unit Pipelining Diagram]
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)

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
Data and Control Pipelining (2)

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

Timing diagram with additional NO clock cycles

<table>
<thead>
<tr>
<th>Cycle</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>Read PC</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>Read CWR</td>
<td>S1</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>S2</td>
<td>NO</td>
<td>NO</td>
<td>S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read RF(L)</td>
<td>a</td>
<td>c</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read RF(R)</td>
<td>b</td>
<td>d</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write ALU(L)</td>
<td>a</td>
<td>c</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write ALU(R)</td>
<td>b</td>
<td>d</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write ALUOut</td>
<td>c+d</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write RF</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write PC</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10/17</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- 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

C flowchart:

\[ a = \text{In1} \]
\[ b = \text{In2} \]
\[ t_1 = |a| \]
\[ t_2 = |b| \]
\[ x = \max(t_1, t_2) \]
\[ y = \min(t_1, t_2) \]
\[ t_3 = x \gg 3 \]
\[ t_4 = y \gg 1 \]
\[ t_5 = x + t_3 \]
\[ t_6 = t_4 + t_5 \]
\[ t_7 = \max(t_6, x) \]
\[ \text{Done} = 1 \]
\[ \text{Out} = t_7 \]

CDFG:
RC Scheduling

Scheduling Algorithms
**TC Scheduling**

TC _Scheduling_ involves the scheduling of tasks based on their start and end times. The ASAP (As Soon As Possible) and ALAP (As Late As Possible) schedules are used to determine the earliest and latest start times for each task. The TC schedule combines these with the constraints of the task to determine the actual schedule.

In the diagram, tasks are represented by nodes, and the arrows indicate the precedence relationships between tasks. The ASAP schedule is shown on the left, the ALAP schedule is shown on the right, and the TC schedule is shown in the middle.

**Distribution Graphs for TC scheduling**

The distribution graphs show the initial probability distribution and the graph after max, +, and – operations were scheduled. The initial distribution graph shows the initial probability for each state, while the graph after operations show the updated probabilities after applying the operations.

- **AU units**: The allocation units for each state.
- **Probability sum/state**: The sum of probabilities for each state.
- **Shift units**: The shift units for each operation.
Distribution Graphs for TC scheduling

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

Distribution graph for final schedule

Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- 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)

- OutAddr = BusAddr
- OutData = BusData
- OutCnt = WRITE WORD
- ack = 1
- ready = 1
- ack = 0
- ready = 0

MAC driver

Bus protocol

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**