EE382V-ICS: System-on-a-Chip (SoC) Design

Hardware Synthesis and Architectures


Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Outline

- 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
Hardware 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
RTL Processor 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 Processor with FSM Controller

- **Simple architecture**
- **Small number of states**
**RTL with Programmable Control**

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

**Outline**

- 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

```c
01: int OnesCounter(int Data)
02: {
03:   int Ocount = 0;
04:   int Temp, Mask = 1;
05:   while (Data > 0) {
06:     Temp = Data & Mask;
07:     Ocount = Ocount + Temp;
08:     Data >>= 1;
09:   }
10:   return Ocount;
11: }
```

<table>
<thead>
<tr>
<th>Function-based C code</th>
<th>RTL-based C code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01: while(1) {</td>
<td>01: while(1) {</td>
</tr>
<tr>
<td>02:     while (Start == 0);</td>
<td>02:     while (Start == 0);</td>
</tr>
<tr>
<td>03:       Done = 0;</td>
<td>03:       Done = 0;</td>
</tr>
<tr>
<td>04:       Data = Input;</td>
<td>04:       Data = Input;</td>
</tr>
<tr>
<td>05:       Ocount = 0;</td>
<td>05:       Ocount = 0;</td>
</tr>
<tr>
<td>06:       Mask = 1;</td>
<td>06:       Mask = 1;</td>
</tr>
<tr>
<td>07:     while (Data&gt;0) {</td>
<td>07:     while (Data&gt;0) {</td>
</tr>
<tr>
<td>08:       Temp = Data &amp; Mask;</td>
<td>08:       Temp = Data &amp; Mask;</td>
</tr>
<tr>
<td>09:       Ocount = Ocount + Temp;</td>
<td>09:       Ocount = Ocount + Temp;</td>
</tr>
<tr>
<td>10:       Data &gt;&gt;= 1;</td>
<td>10:       Data &gt;&gt;= 1;</td>
</tr>
<tr>
<td>11:   }</td>
<td>11:   }</td>
</tr>
<tr>
<td>12:     Output = Ocount;</td>
<td>12:     Output = Ocount;</td>
</tr>
<tr>
<td>13:     Done = 1;</td>
<td>13:     Done = 1;</td>
</tr>
<tr>
<td>14: }</td>
<td>14: }</td>
</tr>
</tbody>
</table>
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 statements in each state executed concurrently
  - Control signal and variable assignments executed concurrently

- **FSMD includes scheduling**
- **FSMD doesn’t specify binding or connectivity**
**CDGF and FSMD for Ones Counter**

![Diagram of 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**

```verbatim
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: // ...
14: S7: begin
15: B1 = RF[2];
16: Outport <= B1;
17: done <= 1;
18: next_state = S0;
19: end
20: endcase
21: end
22: endmodule
```

Outline

- 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)

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

**Variable usage**

<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>b</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of live variables</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**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>abs</td>
<td>min</td>
<td>max</td>
<td>&gt;&gt;</td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No. of operations</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Variable and Operation Usage**

<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></td>
<td>a</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td>=</td>
<td></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>
<td></td>
</tr>
<tr>
<td>x</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>=</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>
<td></td>
</tr>
<tr>
<td>t7</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Out</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Connectivity usage

**Operation usage**

<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>abs</td>
<td>min</td>
<td>max</td>
<td>&gt;&gt;</td>
<td>-</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>No. of operations</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Connectivity usage**

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

EE382V-ICS: SoC Design © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner
Outline

- 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
Register Sharing

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

![Diagram showing register sharing concepts](image)

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

Datapath with Shared Registers

- Variables combined into registers
- One functional unit for each operation
Functional Unit Sharing

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

---

**Operation Merging for SRA**

Initial compatibility graph

Compatibility graph after merging of min, +, and -

Compatibility graph after merging of + and -

Final graph partitions
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

Datapath with Shared Registers, FUs, 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, y]
R2 = [b, q, y, t3, y]
R3 = [x]
```

Compatibility graph

```
S0: a = In1
S1: b = In2
S2: t1 = |a|
S3: t2 = |b|
S4: t3 = x >> 3
S5: t4 = y >> 1
S6: t5 = x - t3
S7: x = max( t1, t2 )
y = min( t1, t2 )
t6 = t4 + t5
S8: t7 = max( t6, x )
Done = 1
Out = t7
```

Datapath with Shared RF

- **RF minimize connectivity cost by sharing ports**
Outline

- 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 operation in a single clock cycle**
- **Improves performance at no cost**
Datapath with Chaining and Multi-Cycling

- Multi-cycling
  - Operations that take more than one cycle
  - Allows use of slower FUs
  - Allows faster clock-cycle

Outline

- 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 (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)

In

\[ \text{Done} = 1 \]

\[ \text{Out} = t_7 \]

\[ t_1 = |a| \]

\[ x = \max(t_1, t_2) \]

\[ t_3 = \max(t_1, t_2) \gg 3 \]

\[ t_4 = \min(t_1, t_2) \gg 1 \]

\[ t_6 = t_4 + t_5 \]

\[ t_7 = \max(t_6, x) \]

\[ \text{Start} \]

\[ a = \text{In}_1 \]

\[ b = \text{In}_2 \]

\[ S_0 \]

\[ S_1 \]

\[ S_2 \]

\[ S_4 \]

\[ S_6 \]

\[ S_7 \]

\[ S_8 \]

\[ t_5 = x - t_3 \]

\[ S_5 \]

\[ t_2 = |b| \]

\[ S_3 \]

\[ S_0 \]

\[ S_1 \]

\[ S_2 \]

\[ S_4 \]

\[ S_6 \]

\[ S_7 \]

\[ S_8 \]

Timing diagram with additional NO clock cycles

<table>
<thead>
<tr>
<th>Cycles</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>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read R1</td>
<td>a</td>
<td>11</td>
<td>11</td>
<td>x</td>
<td>x</td>
<td>x</td>
<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>
<tr>
<td>Read R2</td>
<td>b</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Read R3</td>
<td>c</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>ALUIn(L)</td>
<td>a</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>ALUIn(R)</td>
<td>b</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>ALUOut</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Shifters</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Write R1</td>
<td>a</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Write R2</td>
<td>b</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Write R3</td>
<td>c</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

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 NOP cycles for the branch
- 2 NOP cycles for data dependence

Timing diagram with additional NO clock cycles

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycle</th>
<th>S</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>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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Read CWR</td>
<td>11</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Read RF(R)</td>
<td>12</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Write ALUIn(L)</td>
<td>13</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Write ALUIn(R)</td>
<td>14</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Write ALUOut</td>
<td>15</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Write RF</td>
<td>16</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Write SR</td>
<td>17</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>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>Write PC</td>
<td>18</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>
<td>j</td>
<td>k</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
RC Scheduling

ASAP schedule
ALAP schedule
Ready list with mobilities (ALAP – ASAP)
RC schedule (for single FU and 2 shifters)

Scheduling Algorithms

Perform ASAP
Perform 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

Perform ASAP
Perform ALAP
Determine mobilities ranges
Create probability distribution graphs

All ops scheduled?
yes
no

Any gain?
yes
no

Schedule op with maximum gain
Schedule op with minimum loss

EE382V-ICS: SoC Design © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schimer
TC Scheduling

Distribution Graphs for TC scheduling

Initial probability distribution graph

Graph after max, +, and – were scheduled
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 for selected bus interface
  - Bus interface
    - Implemented by bus interface component from library

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

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
Conclusions

- **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**