Reading Assignment

- This lecture: 6.1, 6.2
- Next lecture: 6.12, 6.3
Outline

Simplified HLS Flow Overview

Hardware Synthesis Flow

Output Models

Input Models
Simplified HLS Setting

- Input
  - Behavioral description based on DFG
  - Ignore design constraints
- Fixed allocation for functional units
  - Functional units are combinational: no pipelining, need storage units for input/output
- Scheduling
  - Fixed clock period
  - No *chaining*: no data-dependency among activities executed in one clock cycle
- Objective: generate a hardware implementation with reasonably good performance and cost
Simplified Design Flow

1. DFG generation
2. Scheduling and functional units binding
3. Storage units allocation and binding
4. Control unit synthesis

▶ 2 and 3 are usually known as datapath synthesis
```c
double u, w, y, dx;
int i, N;

for (i = 0; i < N; ++i) {
    double u1, u2, u3, u4, u5, u6, y1;

    u1 = u * dx;
    u2 = 5 * w;
    u3 = 3 * y;
    y1 = i * dx;
    w = w + dx;
    u4 = u1 * u2;
    u5 = dx * u3;
    y = y + y1;
    u6 = u - u4;
    u = u6 - u5;
}
```

- add and sub: 1 clock cycle
- mulA and mulB: 4 clock cycles
Step 1: DFG Generation

- Based on data dependency of loop body
- Vertices are operations (activities), edges are variables
Step 2: Scheduling and Functional Units Binding

- **Control step** (cstep): usually equivalent to a clock cycle
  - Correspond to the lifetime of a single state in the FSM representing the control unit

- Scheduling and functional units binding algorithm
  - Assign operations to functional units iteratively until all operations are assigned
  - Assume external variables to loop body (e.g. u, w, y, i, dx) are ready in cstep 0 and scheduling starts in cstep 1
  - Each iteration handles one cstep
  - Data dependency: only operations with no unfinished predecessor at the beginning of the cstep can start execution in the cstep
  - Structural dependency: subject to the availability of functional units
Overall Scheduling and Functional Units Binding
Step 3: Storage Units Allocation and Binding

Storage units: registers (flip-flops)

The straight-forward approach: allocate a register to each variable
  Drawbacks: may need more than necessary number of registers, increase chip area

Solution: share registers among variables
  Variable *lifetime*: time interval between its definition to its last use
  Two variables can share a register if their lifetimes don’t overlap

Assume external variables to loop body (e.g. u, w, y, i, dx) won’t share registers with other variables
Variable Lifetimes and Register Allocations

▶ Need $5 + 3 = 8$ instead of $5 + 7 = 12$ registers
Step 4: Control Unit Synthesis

- The Diagram in the previous slide is clearly not completed
  - An input port cannot be driven by multiple signals
  - A register should hold its data until being explicitly changed
- Use a mux at each input to choose the correct signal
- Control unit synthesis: design FSMs to generate the control signals for the mux’s
  - State transition depends on scheduling and binding
Step 4: Control Unit Synthesis

- The Diagram in the previous slide is clearly not completed
  - An input port cannot driven by multiple signals
  - A register should hold its data until being explicitly changed
- Use a mux at each input to choose the correct signal
- Control unit synthesis: design FSMs to generate the control signals for the mux’s
  - State transition depends on scheduling and binding
Step 4: Control Unit Synthesis

- The Diagram in the previous slide is clearly not completed
  - An input port cannot driven by multiple signals
  - A register should hold its data until being explicitly changed
- Use a mux at each input to choose the correct signal
- Control unit synthesis: design FSMs to generate the control signals for the mux's
  - State transition depends on scheduling and binding
Generate State Transitions for FSM Design

<table>
<thead>
<tr>
<th>Port</th>
<th>csteps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1  2  3  4  5  6  7  8  9  10  11  12  13</td>
</tr>
<tr>
<td>mulA.x</td>
<td>Ru Ru Ru R1 R1 R1 R1 Rdx Rdx Rdx Rdx *</td>
</tr>
<tr>
<td>mulA.y</td>
<td>Rdx Rdx Rdx Rdx R1 R1 R1 R1 R1 mulA R1 mulA R1</td>
</tr>
<tr>
<td>Ry</td>
<td>Ry Ry Ry Ry Ry Ry Ry Ry Ry Ry Ry add</td>
</tr>
<tr>
<td>R1</td>
<td>* * * mulA R1 R1 R1 mulA sub R1 R1 R1</td>
</tr>
<tr>
<td>R2</td>
<td>* * * mulB R2 R2 R2 mulB R2 R2 R2 mulA</td>
</tr>
</tbody>
</table>

ECE 587 – Hardware/Software Co-Design
Spring 2015
15/31
Outline

Simplified HLS Flow Overview

Hardware Synthesis Flow

Output Models

Input Models
Hardware Synthesis Overview

- Synthesize HW components from specification to RTL
  - HW components as standard or custom processors or as special custom hardware units, usually known as intellectual property components (IPs).
  - Untimed specification, e.g. C
  - RTL in HDL for further synthesis

- Via high-level synthesis (HLS, C-to-RTL design)
  - Tasks: allocation/binding/scheduling
  - Also provide estimation of hardware performance metrics
  - A very difficult problem under stringent design constraints: any change in one aspect affects other aspects, trade-offs need to be explored but are not obvious

- Possible HLS flows
  - Complete the there tasks sequentially
  - Pre-allocation: define architecture for HW processor
  - Pre-binding: optimize register usage
  - Pre-scheduling: avoid structural dependency in inner loops
Hardware Synthesis Overview

- Synthesize HW components from specification to RTL
  - HW components as standard or custom processors or as special custom hardware units, usually known as intellectual property components (IPs).
  - Untimed specification, e.g. C
  - RTL in HDL for further synthesis

- Via high-level synthesis (HLS, C-to-RTL design)
  - Tasks: allocation/binding/scheduling
  - Also provide estimation of hardware performance metrics
  - A very difficult problem under stringent design constraints: any change in one aspect affects other aspects, trade-offs need to be explored but are not obvious

- Possible HLS flows
  - Complete the three tasks sequentially
  - Pre-allocation: define architecture for HW processor
  - Pre-binding: optimize register usage
  - Pre-scheduling: avoid structural dependency in inner loops
Hardware Synthesis Overview

- Synthesize HW components from specification to RTL
  - HW components as standard or custom processors or as special custom hardware units, usually known as intellectual property components (IPs).
  - Untimed specification, e.g. C
  - RTL in HDL for further synthesis
- Via high-level synthesis (HLS, C-to-RTL design)
  - Tasks: allocation/binding/scheduling
  - Also provide estimation of hardware performance metrics
  - A very difficult problem under stringent design constraints: any change in one aspect affects other aspects, trade-offs need to be explored but are not obvious
- Possible HLS flows
  - Complete the there tasks sequentially
  - Pre-allocation: define architecture for HW processor
  - Pre-binding: optimize register usage
  - Pre-scheduling: avoid structural dependency in inner loops
HW Synthesis Flow

FIGURE 6.1 HW synthesis design flow

(Gajski et al.)
Outline

Simplified HLS Flow Overview

Hardware Synthesis Flow

Output Models

Input Models
Although the output of HW synthesis is the RTL/FSM model of the component, we can divide it into pieces.

- Facilitate reasonings/communications
- Optimize with specialized algorithms

Datapath

- Perform complicate, but usually combinational, computations
- Produce status signals for decision making

Controller

- Provide control signals to the datapath, e.g. to collect input data and to distribute output data
- Interact with other components, e.g. to notify completion and to start computation once activated
Although the output of HW synthesis is the RTL/FSM model of the component, we can divide it into pieces.

- Facilitate reasonings/communications
- Optimize with specialized algorithms

Datapath
- Perform complicate, but usually combinational, computations
- Produce status signals for decision making

Controller
- Provide control signals to the datapath, e.g. to collect input data and to distribute output data
- Interact with other components, e.g. to notify completion and to start computation once activated
Although the output of HW synthesis is the RTL/FSM model of the component, we can divide it into pieces.

- Facilitate reasonings/communications
- Optimize with specialized algorithms

**Datapath**

- Perform complicate, but usually combinational, computations
- Produce status signals for decision making

**Controller**

- Provide control signals to the datapath, e.g. to collect input data and to distribute output data
- Interact with other components, e.g. to notify completion and to start computation once activated
Controller vs. Datapath

![Diagram showing the relationship between the controller and datapath.](image)

**Figure 6.2** High-level block diagram

(Gajski et al.)
Figure 6.3: RTL diagram with FSM controller

(Gajski et al.)
Controller and Datapath Implementations

- The FSM of the whole HW component can be decomposed into a FSMD.
  - The controller as a FSM (much less states compared to the FSM for the whole component)
  - The datapath as data and their operations associated with each state transition

- Controller FSM
  - Current state: stored in State Register (SR)
  - State transition: computed via input logic
  - Inputs: including original inputs and status bits returned by datapath operations
  - Outputs: computed via output logic, including original outputs and signals to activate corresponding datapath operations

- Datapath
  - Registers: for data storage and pipelining
  - Functional units: for computation
  - Interconnects: shared busses and wires, plus muxes and tri-state buffers for multiplexing
Both the controller and the datapath can be pipelined.

FIGURE 6.4 RTL diagram with programmable controller (Gajski et al.)
Outline

Simplified HLS Flow Overview

Hardware Synthesis Flow

Output Models

Input Models
LISTING 6.1  Function-based C code

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

LISTING 6.2  RTL-based C code

```c
while(1) {
    while(Start == 0);
    Done = 0;
    Data = Input;
    Ocount = 0;
    Mask = 1;
    while(Data > 0) {
        Temp = Data & Mask;
        Ocount = Ocount + Temp;
        Data >>= 1;
    }
    Output = Ocount;
    Done = 1;
}
```

(Gajski et al.)
while(1) {
    while(Start == 0);
    Done = 0;
    Data = Input;
    Ocount = 0;
    Mask = 1;
    while(Data > 0) {
        Temp = Data & Mask;
        Ocount = Ocount + Temp;
        Data >>= 1;
    }
    Output = Ocount;
    Done = 1;
}

LISTING 6.2 RTL-based C code

FIGURE 6.5 CDFG for Ones counter (Gajski et al.)
FSMs of BBs are specified and then combined.

Scheduling is explicit. Interconnects are not specified.

(Gajski et al.)
FSMs of BBs are specified and then combined.

- Scheduling is explicit. Interconnects are not specified.
HDL

```
// ...
always@(posedge clk)
begin : output_logic
  case (state)
    // ...
    S4: begin
      B1 = RF[0];
      B2 = RF[1];
      B3 = alu(B1, B2, 1_and);
      RF[3] = B3;
      next_state = S5;
    end
  endcase
end
endmodule
```

LISTING 6.3  RTL description in HDL (excerpt)

(Gajski et al.)

- **Textural**
  - May distinguish variables from signals
  - Provide hints to select components and to construct interconnects
HDL

```hdl
// ...
always @(posedge clk)
begin : output_logic
    case (state)
        // ...
        S4: begin
            B1 = RF[0];
            B2 = RF[1];
            B3 = alu(B1, B2, 1_and);
            RF[3] = B3;
            next_state = S5;
        end
    endcase
end
endmodule

// ...
S7: begin
    bus_32_0 = RF[2];
    Outport <= B3;
    Done <= 1;
    next_state = S0;
end
end
```

LISTING 6.3 RTL description in HDL (excerpt)

(Gajski et al.)

- Textural
- May distinguish variables from signals
- Provide hints to select components and to construct interconnects
HDL

// ...
always@(posedge clk)
begin : output_logic
case (state)
  // ...
  S4: begin
    B1 = RF[0];
    B2 = RF[1];
    B3 = alu(B1, B2, l_and);
    RF[3] = B3;
    next_state = S5;
  end
end
case
  // ...
  S7: begin
    bus_32_0 = RF[2];
    Outport <= B3;
    Done <= 1;
    next_state = S0;
  end
endcase
end
endmodule

LISTING 6.3 RTL description in HDL (excerpt)

(Gajski et al.)

- Textural
- May distinguish variables from signals
- Provide hints to select components and to construct interconnects
Pretty much everything is specified except state encoding for controller FSM.

(Gajski et al.)
Summary

- Inputs to HLS may provide partial information to guide the synthesis flow.
- Outputs of HLS may be further optimized.