Digital Circuit Design and Language

RTL Design
(Using ASM/SM Chart)

Chang, Ik Joon
Kyunghee University
Process of Logic Simulation and Synthesis

Design Entry
- Develop specification

HDL Description
- Develop/edit HDL description
- Simulate/verify HDL description
  - Correct?
    - Yes → Logic Synthesis (Static Timing Analysis)
    - No → Functional Verification

Logic Simulation
- Test bench

Functional Verification
- Develop (manually) gate-level model
- Compare simulation results

Logic Synthesis (Static Timing Analysis)
- Synthesize netlist
  - Synthesis tools?
    - Yes → Gate-level Netlist
    - No → Logic Verification (Function + Timing)
- Simulate netlist/model

Gate-level Netlist
- Create production masks for ICs
  - Match?
    - Yes → Logic Verification (Function + Timing)
    - No → Logic Verification (Function + Timing)

Large Digital System Design

- Will you still use a state table technique?
  - No, time consuming

- Modular design approach
  - Partitioned into modular sub-system
  - Various modules are interconnected with data-path and control signals
Binary information stored in a digital system can be classified as either data or control information.

- **Data**: Information that is manipulated to perform data processing tasks. For example, data operations by adder, decoder, multiplexer, counter, shift register, ...
- **Control**: Command signals that supervise the data operations.

=> Logic design

1. Design of *data processing circuits* (datapath)
2. Design of *control circuits* for the control of action sequence
Digital System Architecture (Cont.)

- Datapath units consist of:
  - Arithmetic units:
    - Arithmetic and logic units (ALU)
    - Storage registers
  - Logic for moving data:
    - through the system
    - between the computation units and internal registers
    - to and from the external environments

- Control units are commonly modeled by:
  - State Diagram
  - Algorithm state machine (ASM) charts for FSM

- A combined control-dataflow sequential machine is modeled by ASMD charts
Algorithmic State Machine Chart (ASM CHARTS)

+ State Machine Charts (SM Charts)
  * Resemble Software Flow Charts
  * Substitute for State Graph

+ Advantages of SM Charts over State Graphs
  * Easier to Understand Operation of Digital System
  * Necessary Conditions for Transitions Automatically Satisfied
  * Directly Leads to Behavior Models and Hardware Realization
ASM Chart Basic Unit

- **Basic Unit**
  - State Box (Moore output and register operation)
  - Decision Box
  - Conditional Box (Mealy output and register operation)
Mealy and Moore Models of FSM

- Moore model: outputs are synchronized with clock
- Mealy model: value presented immediately before the active edge of the clock

FIGURE 5.21  Block diagrams of Mealy and Moore state machines
ASM Chart Example

Control + Datapath

Control
\textbf{The state of the system during one clock interval}
EQUIVALENT ASM BLOCKS

(a) S1 / Z1 → X1
    0 → 0
    1 → Z2
    Z2 → 0
    1 → X2
    0 → S2 /
    1 → S3 /

(b) S1 / Z1 → X2
    0 → 0
    1 → X1
    0 → S2 /
    1 → Z2
    Z2 → S3 /
SM BLOCK WITH FEEDBACK

+ No Internal Feedback Allowed in SM Block

(a) incorrect

(b) correct
EQUIVALENT ASM BLOCKS

\[ X_1 = X_2 = 1, \quad X_3 = 0 \]

(a) Parallel form

(b) Serial form
CONVERSION FROM STATE GRAPH

(a) State Graph

(b) Equivalent SM chart
TIMING CHART

(b) Equivalent SM chart
ASM Block and Timing Relation

Control + Datapath

Control alone

State Diagram

Timing Relation
ASMD Chart

- No register operation in state box
- The edge are annotated with register operation
- Conditional box are annotated with register operation
Binary Multiplier

골 레지스터의 초기 내용

\[ \begin{align*}
\text{Load} & : 000001011 \quad M \quad (11) \\
\text{Sh} & : 1101 \quad \text{M}
\end{align*} \] （13）

\[ \begin{align*}
\text{Ad} & : 011011011 \\
\text{Cm} & : 001101101 \quad M
\end{align*} \]

\[ \begin{align*}
\text{M} & : 1101 \\
\text{M} & : 100111101 \\
\text{M} & : 01001110 \\
\text{M} & : 00100111 \\
\text{M} & : 01000111
\end{align*} \] （143）

\[ \text{골과 피승수 사이의 분할선} \]
SM Chart Implementation

\[ B^+ = A'B'X + A'BX + ABX \]

링크 1  릴크 2  릴크 3

\[ A^+ = A'BX + ABX \]
Load = A'B'St
Sh = A'BM'(K + K) + AB'(K' + K) = A'BM' + AB'
Ad = A'BM
Done = AB
Design a digital system with
  2 FF’s (E and F)
  1 4-bits binary counter A (A₃A₂A₁A₀)
  1 start input, Start

Operation
  “Start=1” initiate the system operation by clearing A and F, then
  A counts up until the operation stop
  A₂ and A₃ determine the operation sequence

  If A₂=0, E ← 0 and keeps counting
  else E ← 1 then
      if A₃=0, keeps counting
      else F ← 1 on the next CP and stops counting

  Then if Start=0, the system remains in the initial state, else the cycle repeats by clearing A and F

States
  initial state : S_idle
  counting : S_1
  F ← 1 and stop : S_2
Decision of data-path register and I/O

- Register: A3, A2, A1, A0 and E/F
- Input: Start, reset_b, CLK
- Output: E, F, A1, A0
- In and Out: A3, A2
Step2: Define Data-path Register Operation

- Decide Hardware Algorithm Defining Register Operation
- Using C or Matlab, verify the hardware algorithm
- No Clock, no datapath and controller separation yet
Step 3: Block Diagram of Controller and Datapath

- Separation of Controller and Datapath
- Define the interface between Controller and Datapath
- Define the states of the Controller
Step 4: ASMD Chart

- ASMD chart considering timing relation (Clock)
- Using C programming, verify the ASMD chart and obtain the golden answer
Step 5: State Diagram and Assignment

(a)

Step 5: State Diagram and Assignment

(a)

\[\begin{align*}
S_{idle} & \rightarrow S_1, \text{clr}_A:F: & A & \leftarrow 0, F \leftarrow 0 \\
S_1 & \rightarrow S_1, \text{incr}_A: & A & \leftarrow A + 1 \\
 & \quad \text{if } (A_2 = 1) \text{ then } \text{set}_E: & E & \leftarrow 1 \\
 & \quad \text{if } (A_2 = 0) \text{ then } \text{clr}_E: & E & \leftarrow 0 \\
S_2 & \rightarrow S_{idle}, \text{set}_F: & F & \leftarrow 1 \\
\end{align*}\]

(b)
STATE ASSIGNMENT

+ One-Hot State Encoding
  • One Flip-Flop per State
  • Simple Decoding Logic

+ Minimal State Encoding
  • Use \([\lceil \log_2(#\text{states}) \rceil]\) Flip-Flops
  • Assignment of State Codes
    • Impacts Amount of Combinational Logic
module Design_Example_STR
(output reg [3:0] A, output reg E, F, input Start, clock, reset_b);

Controller_STR M0 (clr_A_F, set_E, clr_E, set_F, incr_A, Start, A[2], A[3],
clock, reset_b);
Datapath_STR M1 (A, E, F, clr_A_F, set_E, clr_E, set_F, incr_A, clock);
endmodule

Module Controller_RTL(set_E, clr_E, set_F, clr_A_F, incr_A, A2, A3, Start,
clock, reset_b);
output reg set_E, clr_E, set_F, clr_A_F, incr_A;
input A2, A3, Start, clock, reset_b;
reg [1: 0] state, next_state;
parameter S_idle = 2'b00, S_1 = 2'b01, S_2 = 2'b11;

always @ (posedge clock, negedge reset_b)
if (reset_b == 0) state <= S_idle;
else state <= next_state;
always @ (state, Start, A2, A3) begin
next_state = S_idle; clr_A_F = 0; set_E = 0; set_F = 0; incr_A = 0;
case (state)
S_idle: if (Start) begin next_state = S_1; clr_A_F = 1; end
else next_state = S_idle;
S_1: begin incr_A = 1;
if (A2 & A3) next_state = S_2;
else next_state = S_1;
if (A2 == 0) clr_E = 1; else set_E = 1;
end
S_2: begin set_F = 1; next_state = S_idle; end
endcase
end
endmodule

Module Datapath_RTL(A, E, F, clr_A_F, set_E, clr_E, set_F, incr_A, clock);
output reg [3:0] A;
output reg E,F;
input set_E, clr_E, set_F, clr_A_F, incr_A, clock;

always @ (posedge clock) begin
if (clr_E) E <= 0;
if (set_E) E <= 1;
if (clr_A_F) begin A <= 0; F <= 0; end
if (set_F) F <= 1;
if (incr_A) A <= A + 1;
end
endmodule
Design Practice: Binary Multiplier

Decision of data-path register and I/O
- Register: B(8-bit), Q(8-bit), A(8-bit), C(1-bit), P(4-bit)
- Input: Start, reset, CLK
- Output: Product

10111 multiplicand
10011 multiplier
10111
10111
00000
00000
10111
110110101 product
Design Algorithm

Start

B = Multiplicand, Q = Multiplier, P = 8

P = P - 1

Q[0] >= A + B

Zero

{C, A, Q} <= {C, A, Q} >> 1

END
Block Diagram of Controller and Datapath
ASMD Chart of Binary Multiplier

(a) - S_idle
- Start
- S_add
- Q[0] \(1\)
- S_shift
- Zero

(b) - S_idle
- Ready
- Start
- Load_regs
- S_add
- Decr_P
- Q[0] \(1\)
- Add_regs
- S_shift
- Shift_regs
- Zero

Rules:
- \(A \leq 0\)
- \(C \leq 0\)
- \(B \leq \text{Multiplicand}\)
- \(Q \leq \text{Multiplier}\)
- \(P \leq m_{\text{size}}\)

- \(P \leq P - 1\) Decrement counter
- \([C, A] \leq A + B\) Add multiplicand to shifted sum
- \([C, A, Q] \leq [C, A, Q] >> 1\) 17-bit register shifts to the right by one bit
State Assignment

State Assignment for Control

<table>
<thead>
<tr>
<th>State</th>
<th>Binary</th>
<th>Gray Code</th>
<th>One-Hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{idle}$</td>
<td>00</td>
<td>00</td>
<td>001</td>
</tr>
<tr>
<td>$S_{add}$</td>
<td>01</td>
<td>01</td>
<td>010</td>
</tr>
<tr>
<td>$S_{shift}$</td>
<td>10</td>
<td>11</td>
<td>100</td>
</tr>
</tbody>
</table>

State Table for Control Circuit

<table>
<thead>
<tr>
<th>Present-State Symbol</th>
<th>$G_1$</th>
<th>$G_0$</th>
<th>Start</th>
<th>Q[0]/1</th>
<th>Zero</th>
<th>$G_1$</th>
<th>$G_0$</th>
<th>Ready</th>
<th>Load_regs</th>
<th>Decr_P</th>
<th>Inc_regs</th>
<th>Add_regs</th>
<th>Shift_regs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{idle}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_{idle}$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_{add}$</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$S_{add}$</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$S_{shift}$</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$S_{shift}$</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Behavioral HDL Model of Binary Multiplier

module Algorithmic_Binary_Multiplier #(parameter dp_width = 5) (  
    output [2*dp_width - 1: 0] Product, input [dp_width - 1: 0] Multiplicand, Multiplier);  
reg [dp_width - 1: 0] A, B, Q; // Sized for datapath  
reg C;  
integer k;  
assign Product = {C, A, Q};

always @ (Multiplier, Multiplicand) begin  
    Q = Multiplier;  
    B = Multiplicand;  
    C = 0;  
    A = 0;  
    for (k = 0; k <= dp_width -1; k = k+1) begin  
        if (Q[0]) {C, A} = A + B;  
        {C, A, Q} = {C, A, Q} >> 1;  
    end  
end  
endmodule

* Not synthesizable, more detailed hardware description
module Sequential_Binary_Multiplier (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

// Default configuration: 8-bit datapath
parameter dp_width = 8;

output [2*dp_width: 0] Product;
output Ready;

input [dp_width - 1: 0]  Multiplicand, Multiplier;
input Start, clock, reset_b;

parameter BC_size = 4; // Size of bit counter
parameter S_idle = 3'b001, // one-hot code
S_add = 3'b010,
S_shift = 3'b100;

reg [2: 0] state, next_state;
reg [dp_width - 1: 0] A, B, Q;
reg C;
reg [BC_size - 1: 0] P;
reg Load_regs, Decr_P, Add_regs, Shift_regs;

assign Product = {C,A,Q};

wire Zero = (P == 0); // counter is zero
wire Ready = (state == S_idle); // controller status

// control unit
always @ (posedge clock, negedge reset_b)
if (~reset_b) state <= S_idle; else state <= next_state;

always @ (state, Start, Q[0], Zero) begin
next_state = S_idle;
Load_regs = 0;
Decr_P = 0;
Add_regs = 0;
Shift_regs = 0;

endcase
end

// datapath unit
always @ (posedge clock) begin
if (Load_regs) begin
P <= dp_width;  A <= 0;  C <= 0;
B <= Multiplicand;
Q <= Multiplier;
end
if (Add_regs) {C, A} <= A + B;
if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
if (Decr_P) P <= P - 1;
end
endmodule