Digital System Design
Design Experiences & Case Study

Lecturer: Min-An Chao
Advisor: Prof. An-Yeu Wu
Date: 2009.5.13 Wednesday
Outline

- Digital Circuit Design Guideline
  - Understanding Datasheet & Timing Diagram
  - Control Unit & Finite State Machine
  - Data Path
- Common Questions about Verilog Syntax
  - Signed signal
  - Vector array
- Debugging
  - RTL Debugging & Easy-Debugging Coding Style
  - Gate-Level Debugging
- Case Study
  - Common DSP Engines
  - Computation Engines
  - Pipelined MIPS
Digital Circuit Design Guideline

- Digital circuits usually consist of
  - Combinational function unit
  - Clock triggered control logic
  - Clock triggered flip-flops as registers

- Digital circuit design methodology
  - First, checking functional & timing specification
  - Then, control logic modeling
  - Last, functionality & data path modeling
  - It is very hard to debug with wrong control signals!
Understanding Datasheet & Timing Diagram (1/3)

- At the beginning of digital circuit design
  - Check the I/O specification, including bit lengths, active-high or active-low, in or out, etc.
  - Check the timing specification, especially the input delay

- Draw the timing diagram of your control signals
Understanding Datasheet & Timing Diagram (2/3)

- Store the impulse input signal if necessary
Buffer the input if necessary
Control Unit & Finite State Machine (1/2)

- Correct control signals, then correct data flow!
- Sequential modeling skills
  - Finite state machine (FSM)
  - Counter
- All control signals are based on state/counter
- State transition event modeling skills
  - Conditional flag
  - Counter count-down
Control Unit & Finite State Machine (2/2)

- Relationship between timing diagram & FSM

- Don’t try to reduce number of states
- Keep state behavior simple
Data Path (1/3)

- Data flow through combinational blocks
Data Path (2/3)

- **Pipeline**
  - Gain throughput & clock speed
  - Feedback loops can not be pipelined

![Diagram showing pipeline and feedback loop with timing mismatch]
Data Path (3/3)

- Function block reusing
  - Reduce idle combinational area

- Reduce area when input comes sequentially
Outline

- Digital Circuit Design Guideline
  - Understanding Datasheet & Timing Diagram
  - Control Unit & Finite State Machine
  - Data Path
- Common Questions about Verilog Syntax
  - Signed signal
  - Vector array
- Debugging
  - RTL Debugging & Easy-Debugging Coding Style
  - Gate-Level Debugging
- Case Study
  - Common DSP Engines
  - Computation Engines
  - Pipelined MIPS
Signed Signal (1/3)

“Regard as” basis of signed signal
- MSB is regarded as sign bit and nothing changes, that’s all!
- NO automatic sign extension will be done by compiler

Usage of signed signal
- Addition/Subtraction
  - equivalent to unsigned signals
- Multiplication/Division
  - all inputs & outputs should be defined as signed signals
- Comparison
  - No signed comparison is supported!
  - Signed to unsigned translation should be done manually
Signed Signal (2/3)

- Signed addition bit length
  - A(8 bits) + B(8 bits) → C(8+1 bits)
    ```
    wire signed [7:0] A, B;
    wire signed [8:0] C;
    assign C = {A[7], A} + {B[7], B};
    ```

- Signed multiplication bit length
  - A(3 bits) x B(5 bits) → C((3-1)+(5-1)+1 bits)
    ```
    wire signed [2:0];
    wire signed [4:0];
    Wire signed [6:0];
    assign C = A * B;
    ```
Signed Signal (3/3)

- Signed comparison
  - If \(-3 < A(4\text{ bits}) < 4\), raise flag, otherwise not
  - Signed representation from 2’b0000 to 2’b1111
    - 0 1 2 3 4 5 6 7 -8 -7 -6 -5 -4 -3 -2 -1
  - Unsigned representation from 2’b0000 to 2’b1111
    - 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

- Correct Verilog behavior modeling
  ```verilog
  wire signed [3:0] A;
  wire                      flag;
  assign flag = (A<4'd4 || A>4'd13)? 1'b1: 1'b0;
  ```
Vector Array (1/2)

- Vector array
  - Declaration and usage of vector array (4 vectors of 8 bits)

```verilog
reg[7:0] C[0:3];
assign data_out = C[index_o];
always @(posedge clock) begin
  C[index_i] <= data_in;
end
```

- Hardware translation
Vector Array (2/2)

- **Encoder/decoder issue**
  - If the array size is large or the array is accessed by multiple signals, encoder & decoder may be very large!
  - When input is coming in order, the encoder can be reduced by using shift registers
  - Decoder MUX can be customized if some are not used

- **Debugging issue**

```verilog
reg[7:0] C[0:3];
wire [7:0] dbg_C0 = C[0]; // for debugging
```
Outline

- Digital Circuit Design Guideline
  - Understanding Datasheet & Timing Diagram
  - Control Unit & Finite State Machine
  - Data Path
- Common Questions about Verilog Syntax
  - Signed signal
  - Vector array
- Debugging
  - RTL Debugging & Easy-Debugging Coding Style
  - Gate-Level Debugging
- Case Study
  - Common DSP Engines
  - Computation Engines
  - Pipelined MIPS
RTL Debugging

- Backward trace
  - When S is incorrect, see what drives S first
  - Backward trace until the source of error is found
- Check control signals then data flow
  - Again, *wrong control signals won’t let data flow correct!*
- Partial vector signal in nWave
Easy-Debugging Coding Style (1/2)

- Signal naming with prefix & suffix
  - Utilize the prefix & suffix to show the attributes of signals
  - signal_n: low-active
  - signal_w: wire
  - signal_r: register/flip-flop
  - next_signal: next-state signal of FSM
  - cur_signal: current-state signal of FSM

- Alphabetically naming for waveform debugging
● Pure sequential block

```verilog
always@(posedge clk) begin
    if(state==2’d0) begin
        state <= 2’d1;
    end
    else if(state==2’d1) begin
        state <= 2’d2;
    end
    else if(state==2’d2) begin
        if(flag) state <= 2’d3;
        else state <= state;
    end
    else begin
        state <= 2’d0;
    end
end
```

```verilog
always@(posedge clk) begin
    if(state==2’d0) begin
        next_state = 2’d1;
    end
    else if(state==2’d1) begin
        next_state = 2’d2;
    end
    else if(state==2’d2) begin
        if(flag) next_state = 2’d3;
        else next_state = state;
    end
    else begin
        next_state = 2’d0;
    end
end
```

```verilog
always@(*) begin
    if(state==2’d0) begin
        state <= 2’d1;
    end
    else if(state==2’d1) begin
        state <= 2’d2;
    end
    else if(state==2’d2) begin
        if(flag) state <= 2’d3;
        else state <= state;
    end
    else begin
        state <= 2’d0;
    end
end
```
Gate-Level Debugging (1/2)

- SDF annotation
  - Make sure SDF file exists
  - Make sure SDF file is parsed correctly when running simulation

- Input delay and output delay
  - Synthesis tool does not know when your inputs come and when the outputs are captured
  - No input delay & output delay is default setting

Error occurs! Also setup-time violation!
Gate-Level Debugging (2/2)

- **Setup-time violation**
  - Input of flip-flop not stable when clock triggers
  - Check if there is …
    - Unexpected long combinational delay
    - Combinational loop
    - Improper input delay

- **Hold-time violation**
  - Output of flip-flop is asked stable too early
  - If caused by shift registers, it will be fixed at place & route stage (backend stage)
  - Otherwise it should not happen at frontend stage
Outline

- Digital Circuit Design Guideline
  - Understanding Datasheet & Timing Diagram
  - Control Unit & Finite State Machine
  - Data Path
- Common Questions about Verilog Syntax
  - Signed signal
  - Vector array
- Debugging
  - RTL Debugging & Easy-Debugging Coding Style
  - Gate-Level Debugging
- Case Study
  - Common DSP Engines
  - Computation Engines
  - Pipelined MIPS
Digital System Category

- Common DSP engines
  - Pure data signal flow
  - Simple control unit or no control unit

- Common computation engines
  - Both data & control signals
  - Typical case of digital system

- Device controllers
  - Handling control signals between host & device
  - A robust FSM is very important

- Processors
  - MIPS
Case Study I: Common DSP Engines

- I/O property: pure data signal flow
- Can be directly modeled as circuit
- Very simple control unit
- FIR filter
  - Directly modeled from transfer function
  - Delay unit modeled by flip-flops
- Fast Fourier transform (FFT)
  - Modeled as butterfly structure by mathematical derivation
  - Parallel multipliers and adders
Case Study II: Computation Engines (1/2)

- I/O property: both data & control signals
- Typical case of digital circuits
- Control signals & Combinational

![Computation Engines Diagram]
Case Study II: Computation Engines (2/2)

- Usually only functionality is specified when designing a computation engine
- Thus, we need to separate the total execution into different stages and then design the FSM
- Control unit triggers the computational block using start/finish flags
- Try pipelining & hardware reusing if possible
Case Study III: Controllers

- I/O property: control signals between host & device, sometimes with data buffer
- It may consists tasks of …
  - Translating control signals
  - Pass data with data buffer
  - Handshaking procedures
- FSM plays an extremely essential role in this case
Case Study IV: Pipelined MIPS

- Further features in the final project …
  - Branch prediction
    - Destination address carried out at ID stage
    - Modifications needed at IF stage
  - Immediately jump
    - Jump immediately at IF stage
    - No flushing
    - Increase the complexity of control unit significantly
  - Customized caches
    - Read-only instruction cache
    - Different number of blocks for two caches