ELEC2221
Digital Systems and Signal Processing

Writing synthesisable SystemVerilog code
- Timing and reliability
Timing

• For all synchronous systems (with a single clock),
  – flip-flop outputs change just after the clock edge
  – flip-flop inputs must be stable just before the clock edge (setup time)

• Failure to observe the 2nd condition (slow combinational logic) will cause unpredictable behaviour

• In that sense, partitioned systems are exactly the same as simple state machines
Asynchronous Inputs

• The "real world" is asynchronous
• Inputs from the real world can violate setup and hold times
• If we are simply waiting for an input change, a delay of one clock cycle is not important (exact time of change is unpredictable)
• Danger of metastability
  – Flip-flops have three states!
  – Intermediate, metastable state with both outputs at "1/2"
  – Setup time violation can put a flip-flop into metastable state - could send state machine to wrong state
  – Flip-flop will switch from metastable state at an unpredictable time
Synchronising inputs

- If 1st flip-flop goes metastable, 2nd won't, so normal system inputs are maintained.
- 2 clock cycle delay is not important
- Code this in standard SystemVerilog
Reset Release Problem

- Asynchronous reset (or set) has priority over clock, so applying the reset is not a problem

- If reset is released at the same time as the rising edge of the clock and D=1, there is the possibility of metastability:
  - If reset -> 1, then clock -> 1, D is stored, Q=1
  - If clock -> 1, then reset -> 1, D is not stored, Q=0
  - Often overlooked, but a potential problem
Reset Release Solution

- 1\textsuperscript{st} flip-flop can go metastable, 2\textsuperscript{nd} won't
- Good practice for all chips. Standard SystemVerilog
Clock Domains

- Multiple clocks can be derived from one master clock.
  - Can divide down the frequency
  - Each domain is in sync with the master clock (and hence with each other), but may be out of phase (i.e. rising edges do not coincide).
  - Makes clock distribution easier.
  - Need to be careful when transferring data from one domain to another. Timing analysis gets difficult. May need to buffer data (FIFO – see later).
Independent Clocks

• Clocks derived from one source are a special case of synchronous design.
• Independent clocks are asynchronous... and therefore difficult.
• Always the danger of setup time violations, and hence lost data and metastability.
• Clock jitter – *average* clock frequency/period might be the same for two clocks, but edges cannot be assumed to coincide cycle-by-cycle.
• Queueing theory says that if arrival rate and departure rate are the same (i.e. two clocks at an interface have the same frequency), queue will grow to infinity.
FIFO

- Common clock and reset
- Circular addressing.
- Can't write when full, can't read when empty
- Needs careful design
Asynchronous FIFO

Design for Reliability

```vhdl
always_comb
begin: COM
...
unique case (present_state)
  A : begin
      ready = '1;
      if (twenty)
          next_state = D;
      else if (ten)
          next_state = C;
      else
          next_state = A;
      end
...
  I : begin
      ret = '1;
      next_state = A;
      end
  default: next_state = A;
endcase
end
```

- unique case should make the default choice redundant.

- Including the default clause won't change the simulation (because we can't ever go there).

- By itself, it won't change the synthesis results – it's necessary, but we also need to tell the synthesis tool what to do.
Design for Reliability

- Synplify.
- Changes logic to implement default assignment, i.e. need both
- More complex logic might mean more resources.

Use "set_global_assignment -name SAFE_STATE_MACHINE OFF" directive in Quartus .qsf file.
Design for Reliability

• Vending machine example. Xilinx Spartan 3.

• Without reliability option: With:

  • 10 LUTs
  • 22 LUTs

• Altera Cyclone V uses 3 registers and 4 logic modules for both.
Initial Values (reprise)

• The initial value of a logic variable is 'X. Shows in simulation as a red line. Useful for debugging.
• Initial value of an enumerated type is the 'left-hand' value. i.e. A in the example. Less useful for debugging.

```c
typedef enum logic[2:0]{A, B, C, D, F, I} state;
(* syn_encoding="sequential" *)
state present_state, next_state;
```
• Initial values (in simulation) for state variables are now 3'bXXX
  – Can easily see if the state machine is not reset.
  – But this is not consistent with reliable design
always_comb
begin: COM
...
unique case (present_state)
  A : begin
    ready = '1;
    if (twenty)
      next_state = D;
    else if (ten)
      next_state = C;
    else
      next_state = A;
  end
...
'ifdef SYNTHESIS
  // for simulation, stay in 'x state
  default: next_state <= state'('x);
'else
  // for synthesis go to safe state
  default: next_state <= A;
'endif
endcase
end
Design for Security

• "Safe State Machine" design is intended to protect against accidental errors.

• With the Internet of Things etc. attackers are turning to deliberate "fault attacks". These might not be a complete cyber attack, but could be used to create vulnerabilities.

• Safe State Machine design can help to protect against cyber attacks
  – But good coding practices are also essential