ELEC2221
Digital Systems and Signal Processing

Writing synthesisable SystemVerilog code
- Sequential logic
Sequential logic synthesis

• When writing code for sequential logic, use `always_ff` and `always_latch` blocks with incomplete conditional statements

• Do not mix complex combinational and sequential logic in one block; partition designs and use multiple blocks

• For state machines use edge-triggered flip-flops (`always_ff`)

• On new FPGAs, e.g. Altera Cyclone IV, V, RAM is synchronous and requires `always_ff`. Use the FPGA vendor’s code templates for RAM
Edge-triggered flip-flops

// rising edge triggered D-type flip-flop with no reset
module ff1 (input logic clk, d, output logic q);

always_ff @ (posedge clk)
    q <= d;

endmodule

// falling edge triggered D-type flip-flop with
// active-high asynchronous clear
module ff2 (input logic clk, D, Clear, output logic Q);

always_ff @ (negedge clk, posedge Clear)
    if (Clear)
        Q <= 1'b0;
    else
        Q <= D;
endmodule
// rising edge triggered D-type flip-flop with
// active-high synchronous preset
module ff3 (input logic clk, D, Preset, output logic Q);

always_ff @ (posedge clk)
  if (Preset)
    Q <= 1'b1;
  else
    Q <= D;
endmodule

// rising edge triggered D-type flip-flop with clock enable
module ff4 (input logic clk, D, CE, output logic Q);

always_ff @ (posedge clk)
  if (CE)
    Q <= D;
endmodule
Latches

// 8-bit latch with an inverted (active-low) enable and active-high preset
module latch2
    (input logic nEN, Preset; input logic [7:0] D; output logic [7:0] Q);

always_latch
    if (Preset)
        Q <= 8'b1111_1111;
    else if (~nEN)
        Q <= D;
endmodule

// latch with active-high enable and active-low clear
module latch1 (input logic EN, D, nClear; output logic Q);

always_latch
    if (~nClear)
        Q <= 1'b0;
    else if (EN)
        Q <= D;
endmodule
RAM synthesis

- RAMs used in older FPGAs are typically arrays of latches with a separate input and a separate output data bus, an address bus and a Write Enable signal; read is asynchronous.

- Modern FPGAs (e.g. Cyclone V) support only synchronous memory, where both read and write is synchronous.

- Note that there are many types of RAMs, eg.
  - RAM with synchronous read
  - RAM with one Enable controlling both ports
  - RAM with separate Enables controlling each port
  - Multiple-Port RAMs

- Synthesis tools are usually able to infer the correct type of RAM supported by the target FPGA, regardless of the SystemVerilog description, but it is useful to be familiar with the specific types of RAM supported by the target FPGA.
Synchronous RAM blocks in modern FPGAs

Altera Cyclone IV devices feature memory structures that consist of M9K memory blocks that can be configured to provide RAM, shift registers, ROM, and FIFO buffers.

An M9K block contains 8,192 memory bits and is **synchronous**, i.e. requires a clock.

Altera Cyclone V devices contain two types of memory blocks. **Both are synchronous.**

1. M10K blocks—10-kilobit (Kb) blocks for larger memory configurations.

2. Memory logic array blocks (MLABs)—640-bit memory blocks for small memories. Each MLAB can be configured as ten 32 x 2 blocks, giving one 32 x 20 simple dual-port SRAM.
Recommended coding for synchronous RAM in modern FPGAs

// synchronous RAM, 128x8
module ram128x8sync(
    output logic [7:0] dout,
    input logic [6:0] address,
    input logic [7:0] din,
    input logic we, clk);

logic [7:0] mem [127:0]; // unpacked array of
// packed vectors

always_ff @(posedge clk)
begin
    if (we)
        mem[address] <= din; // memory write

    dout <= mem[address]; // synchronous memory read
end
endmodule

Quartus synthesis summary

Family   Cyclone V
Logic utilization   N/A
Combinational ALUTs  0
Memory ALUTs        0
Dedicated logic registers  0
Total registers      0
Total pins           25
Total virtual pins   0
Total block memory bits  1,024
Shift register (Bad!)

// an 8-bit shift register with serial in and serial out, note there is no parallel out module shift (input logic clk,SI; output logic SO);

logic [7:0] tmp; // this logic vector defines the 8-bit memory for the shift register

always_ff @(posedge clk)
  begin
    tmp = tmp << 1; // shift the register by 1 bit
    tmp[0] = SI; // overwrite bit 0
  end

assign SO = tmp[7]; // serial out block

endmodule

Note the use of blocking assignments in the always_ff block (not recommended – why not?)
Would this code work with non-blocking assignments?
Shift register with serial in, parallel load and parallel out

// an 8-bit shit register with serial in, parallel load and parallel out
// shift and load are two control inputs, note no serial out port – use Q[7]

module shift2 (input logic clk, SI, shift, load; input logic [7:0] D; output logic [7:0] Q);

always_ff @(posedge clk)
  if(load) // execute parallel load
    Q <= D;
  else if (shift)
    // execute shift, SI goes on Q[0], note non-blocking assignment
    Q <= {Q[6:0], SI}; // shift Q left by 1 bit, shift in SI into Q[0]
endmodule
Counters

parameter N = 8;
logic [N-1:0] q;

always_comb // or always_ff
q++;
module vending(
    output logic ready, dispense, ret, coin
    input logic clock, n_reset, twenty, ten);

enum {A, B, C, D, F, I} present_state, next_state;

always_ff @(posedge clock, negedge n_reset) begin:
    if (~n_reset)
        present_state <= A;
    else
        present_state <= next_state;
end
State Encoding

- Synplify uses one-hot encoding by default

```c
enum {s0, s1, s2} state;
```

will map s0 to 001, s1 to 010 and s2 to 100.

- To force sequential encoding, change to:

```c
typedef enum {s0, s1, s2} state_type;
(* syn_encoding="sequential" *)
state_type state;
```

State encoding will now be s0=00, s1=01, s2=10.
State Encoding

- Verilog standard defines a different form:

  (* synthesis, fsm_state="sequential" *)

- Might be recognised by synthesis tools