2015 Paper E2.1: Digital Electronics II- Solutions

Q1 (a) This question tests students basic understanding of FPGA architecture and how to deal with simple timing and Verilog codes.

(i) 16-bit adder requires 16 LEs to implement; each already has an in-built D-FF. Therefore total required LEs = 16.

[2]

(ii) Total adder delay = 16 x 250ps. Worst-case delay is only dictated by the FF propagation delay and the setup time here. The hold time has no relevance. Therefore td(max) = 4ns + 80ps + 100ps. Maximum operating frequency of the 16-bit counter is: 1/(4.18ns) = 239MHz.

[3]

(iii)

```
module counter_16 (CLK, EN, CTR);

   input    CLK, EN;    // system clock and enable signal
   output   CTR;        // counter output

   reg   [15:0]   CTR;     // internal states

   initial  CTR = 16'b0;

   always @ (posedge CLK)
      CTR <= CTR + EN;
   end               // always

endmodule
```

[3]

Q1 (b) This question tests students understanding of R-2R DAC converters.

(i) This part of the question is purely bookwork.

Resolution – the voltage step equivalent to one least significant bit change of the digital number. Assuming that the digital number is N-bits, then this is the same as Full-scale volrage / (2^N-1).

Linearity error – this is the maximum deviation of the output vs input characteristic from the linear line joining the maximum point with the minimum point.

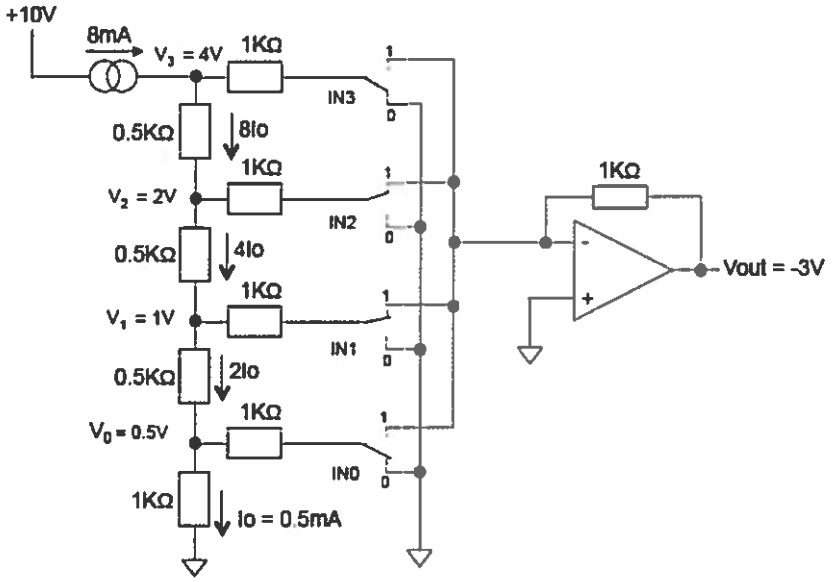
Monotonic DAC – One that always goes up as the input number increases.

Settling time – Time taken to reach final value as input changes.

Monotonic behaviour is most important in a control loop implementation using DAC. If the characteristic is not monotonic, the gain of the system may change sign, and could then cause the system to become unstable.

[2]

(ii) The currents and voltages are shown below.



[4]

Q1 c) (i) In one-hot encoding, each state is identified by one bit in the state code.

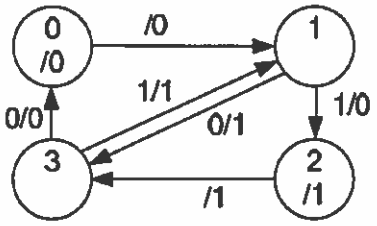
Therefore a N-bit state machine would require N flip flops. In binary state encoding, the binary number is used to represent the state code. Therefore a N-bit state machine would only need $\lceil log_2 N \rceil$, flip flops.

One-hot encoding is particularly suited to FPGA architecture because:

1) Generally it has less combinational logic because state decoding is not required. FPGA LUTs can generally only implement simple Boolean equations with few variables. This fine-grain architecture matches the simpler logic well.
2) FPGAs LEs contains D-FF which makes the architecture FF rich. Hence having to use more FFs in one-hot encoded state machine generally does not consume more hardware resources since the FF comes free.

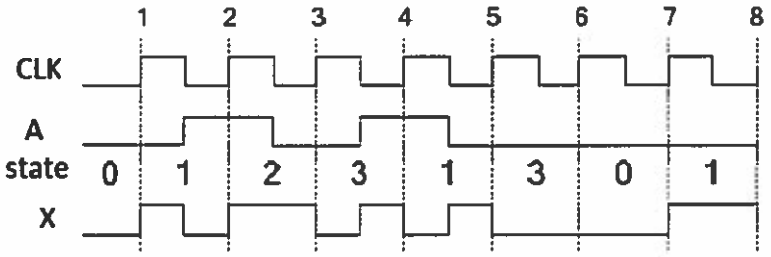
[3]

(ii) The state diagram is:



[3]

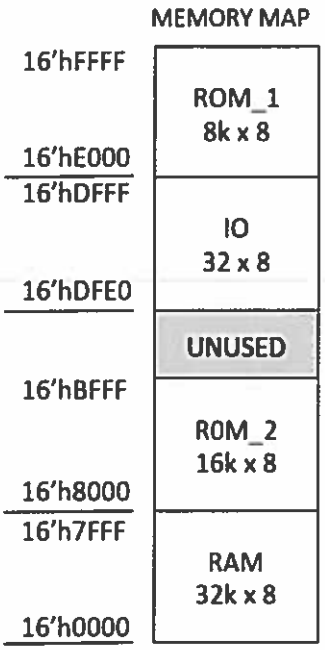
The timing diagram is:



[2]

Q1 (d) This question tests students understanding memory addresses and address decoding.

(i)



[4]

(iii) Enable signals are:

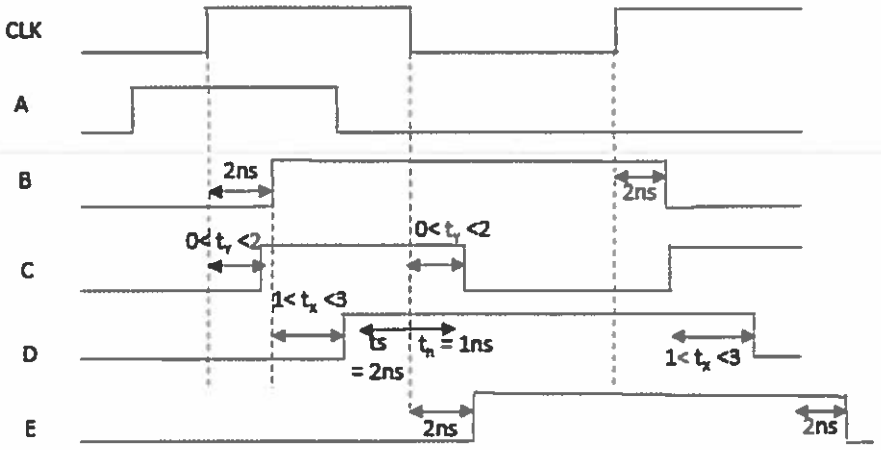
EN_ROM_1 = A15&A14&A13
EN_ROM_2 = A15&/A14
EN_RAM = /A15
EN_IO = A15&A14&/A13&A12&A11&A10&A9&A8&A7&A6&A5

[4]

Q1 (e) This question tests student's ability to work out setup and hold timing constraints for generic circuit.

Note that the right FF is clock on the FALLING edge of the clock – hence clock period dictating timing is ½ T.

(i)



[3]

(ii)

Setup inequality:

$t_p + t_{X\_max} + t_s < t_{Y\_min} + ½\ T$

⇨ $2 + 3 + 2 < 0 + ½\ T$

⇨ $T > 14ns$

[3]

(iii)

Hence T > 14ns, fmax < 71.4MHz. [2]

Q2 This question tests student's understanding of the successive approximation algorithm, and how it could be applied to problems outside ADC.

(a) Bookwork – students are expect to provide succinct description of the successive approximation algorithm compatible with the allocated marks (equivalent to 5 to 6 minutes).

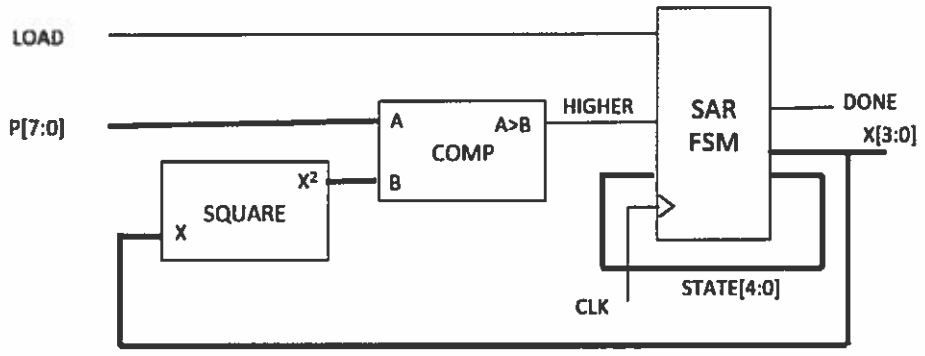
[5]

(b) This absolutely straightforward:

```
module square (x, p);
    input [3:0]     x;
    output [7:0]    p;
    multiplier MULT (x, p);
endmodule;
```

[2]

(c) This is similar to the design for a SAR ADC, except that the DAC and the analogue comparator are replaced by the squaring circuit and the digital comparator respectively as shown below. The SAR FSM design is exactly the same as the for the ADC design.



[15]

(d) The progression for evaluation square root of P = 4'h65 cycle by cycle is:

1. X[3:0] = 4'h8, X^2 = 8'h40, HIGHER = true. Therefore keep X[3] = 1.
2. X[3:0] = 4'hC, X^2 = 8'h90, HIGHER = false. Therefore reset X[2] to 0.
3. X[3:0] = 4'hA, X^2 = 8'h64, HIGHER = true. Therefore keep X[1] = 1.
4. X[3:0] = 4'hB, X^2 = 8'h79, HIGHER = false. Therefore reset X[0] to 0.
5. Final square root value is 4'hA.

[8]

Q3 This question tests student's understanding of the serial peripheral interface, which is covered both through the lectures and the laboratory experiment.

(a) This is relatively straightforward:

```
// --- Submodule: Generate internal clock at 1 MHz -----
reg         clk_1MHz;   // 1Mhz clock derived from 50MHz
reg [4:0]   ctr;        // internal counter
parameter   TIME_CONSTANT = 5'd24;  // change this for di
initial begin
   clk_1MHz = 0;        // don't need to reset - don't ca
   ctr = 5'b0;          //  ... Initialise when FPGA is c
end

always @ (posedge sysclk)    //
  if (ctr==0) begin
     ctr <= TIME_CONSTANT;
     clk_1MHz <= ~clk_1MHz; // toggle the output clock fo
   end
  else
     ctr <= ctr - 1'b1;
// ---- end internal clock generator ----------
```

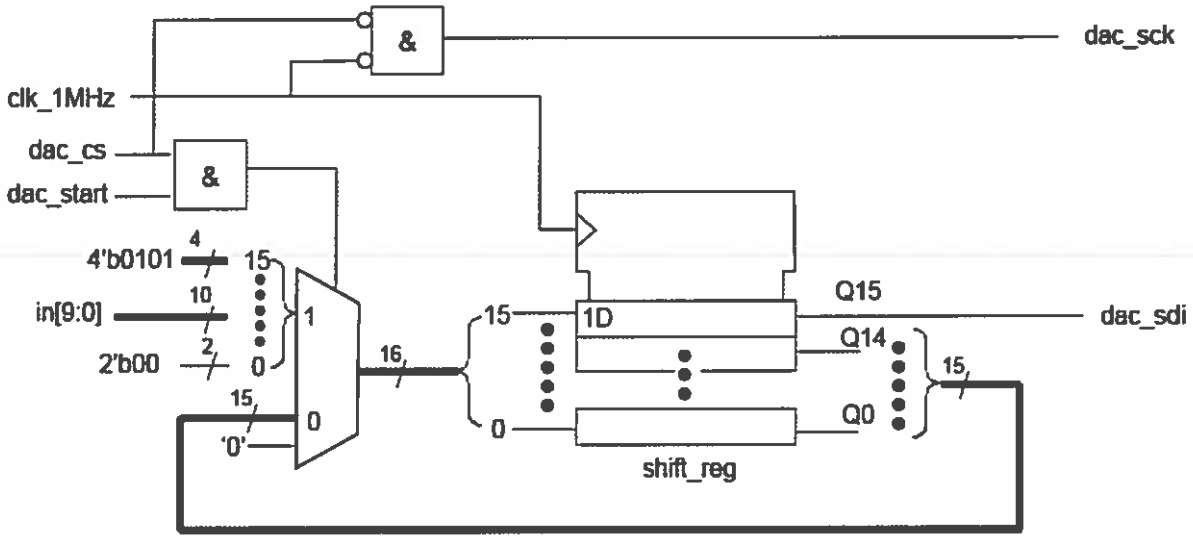
[5]

(b)

```
// ---- Detect posedge of load with a small state machine
// .... FF set on posedge of load
// .... reset when dac_cs goes high at the end of DAC output cycle
reg [1:0]   sr_state;
parameter   IDLE  = 2'b00,WAIT_CSB_FALL = 2'b01, WAIT_CSB_HIGH = 2'b10;
reg         dac_start;      // set if a DAC write is detected

initial begin
   sr_state = IDLE;
   dac_start = 1'b0; // set while sending data to DAC
   end

always @ (posedge sysclk)
   case (sr_state)
      IDLE: if (load==1'b0) sr_state <= IDLE;
            else  begin
               sr_state <= WAIT_CSB_FALL;
               dac_start <= 1'b1;
               end
      WAIT_CSB_FALL: if (dac_cs==1'b1) sr_state <= WAIT_CSB_FALL;
            else sr_state <= WAIT_CSB_HIGH;

      WAIT_CSB_HIGH: if (dac_cs==1'b0) sr_state <= WAIT_CSB_HIGH;
            else begin
               sr_state <= IDLE;
               dac_start <= 1'b0;
               end
      default: sr_state <= IDLE;
   endcase
//------- End circuit to detect start and end of conversion
```

[10]

(c) This part of the question tests student's understanding of how Verilog HDL is mapped to digital circuits.



[15]