**ELEC50001 – Circuits and Systems**

**2022 - 2023**

Answer ALL questions.

There are THREE questions on the paper.

Question ONE counts for 50% of the marks, other questions 25% each

Time allowed: 2 hours

# SOLUTIONS

1. (a) This question tests student's understanding of using shift register to implement a pseudorandom binary sequence generator using primitive polynomial in SystemVerilog.

```
module pseudo8 (
    input logic         rst,    // reset to initial state
    input logic         clk,    // clock input
    output logic [7:0]  random  // 8-bit pseudo random number
);

    logic [8:1]     sreg;       // internal shift register

    always_ff @(posedge clk)
        if (rst)
            sreg <= 8'b1;
        else
            sreg <= {sreg[7:1], sreg[2]^sreg[3]^sreg[4]^sreg[8]};

    assign random = sreg;
endmodule
```

[6]

The sequence is: 8'h02, 8'h05, 8'h0B, 8'h16, 8'h2C, or in binary:

| Binary | Hex |
|---|---|
| 0000 0010 | 0x02 |
| 0000 0101 | 0x05 |
| 0000 1011 | 0x0B |
| 0001 0110 | 0x16 |
| 0010 1100 | 0x2C |

[4]

Feedback:

Most students who engaged with the lab experiments serious found this an easy question. The common mistake was the confusion of the indices. While the indices of the output "random" is [7:0], the example used in lecture and lab used the index of [n:1]. This is intentional – it matches the index with the power of the primitive polynomial.

(b) This question tests student's basic understanding of memory sizes, address ranges, address decoding and composition of memory devices to form larger bank of memory.

(i)

| CS signal | Range | size |
|---|---|---|
| CS_FLASH | 16'h0000 – 16'h1FFF | 8k |
| CS_RAM | 16'h4000 – 16'h7FFF | 16k |
| CS_IO | 16'h9800 – 16'h9BFF | 1k |

[6]

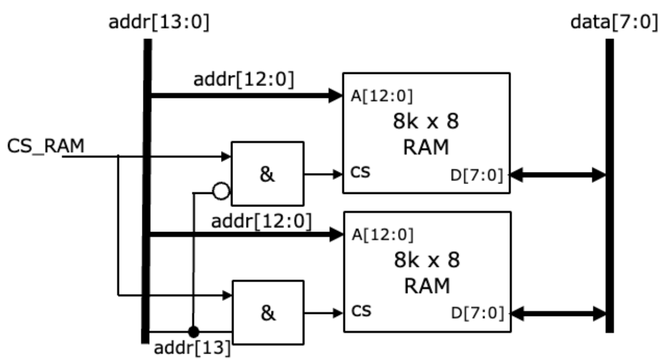
(ii)

```
module decoder (
    input  logic [15:0]  addr,      // address bus
    output logic         CS_FLASH,  // chip-select signal for flash memory
    output logic         CS_RAM,    // chip-select signal for RAM
    output logic         CS_IO      // chip-select signal for input-output
);

    assign CS_FLASH = ~addr[15] & ~addr[14] & ~addr[13];
    assign CS_RAM = ~addr[15] & addr[14];
    assign CS_IO = addr[15] & ~addr[14] & ~addr[13] & addr[12] & addr[11] & ~addr[10];
endmodule
```

[2]

(iii)




[4]

Feedback:

Those who revised the section of the course on address decoding for memory found this question quite easy, particularly (i) and (ii). Some who clearly had no clue even where to start, suggesting that they skipped this part of the course. Some used if-else statement to test for the range. This approach is fine but highly inefficient (and lost 1 mark as a result). Some used "==" operator such as:

assign CS_FLASH = (addr[15:13] == 3'b000);

is entirely correct and is even better than the solution above because it is even more readable and less typing.

Many students could not do (iii). Some did not use CS properly, nor label the address bits. Finally, many added unnecessary mux at the output. They did not lose any mark but is not required.

(c) This question examines student's understanding of digit circuit timing constraints in a pipelined system.

(i) For FF2 data input, the constraint is:

$Tcq\_1 + tp\_P(max) + tsu\_2 \leq Tclk/2$. Therefore. $1 + 5 + 2 \leq Tclk/2$, $Tclk \geq 16ns$.

For FF3 data input, the constraint is:

$Tcq\_2 + tp\_Q(max) + tsu\_3 \leq Tclk/2$.

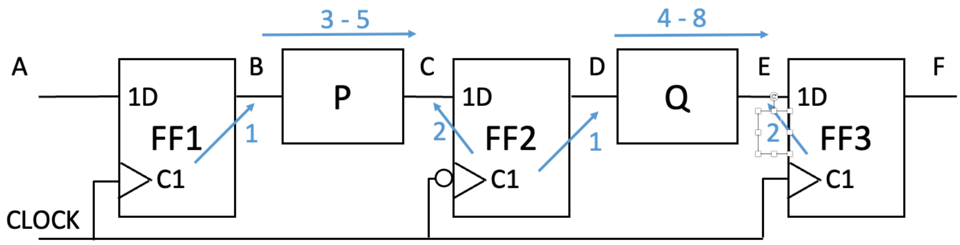
Therefore $1 + 8 + 2 \leq Tclk/2$, $Tclk \geq 22ns$.

FF3 data input dictates the max freq = 45.5MHz.

[5]

(ii) If the clock signal has a high width of 8ns and a low width of 11ns, all setup timing constraints are satisfied. Therefore, max freq = 52.6MHz. The mark-space ratio will be 8:11.

[3]



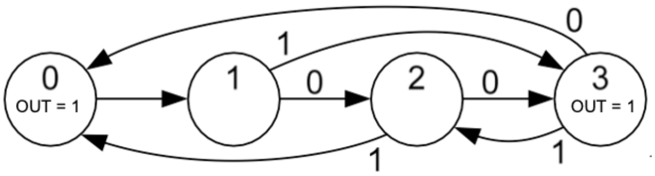

Feedback:

The tricky bit of this question is that FF1, FF3 are clocked on the rising edge, while FF2 is clocked on the falling edge of clock. That means timing constraints must be considered for high and low part of the clock separately. Once that's done for (i), (ii) becomes very easy.

(d) This question tests student's understanding FSM, how to specify this in SystemVerilog and how FPGA hardware (in MAX10) is related to the FSM specifications.

(i)




[4]

(ii) One could implement the FSM from the equations or from the state diagram. The solution here is designed for the latter. Note that the four states are labelled as X0 to X3, not be confused with the state variables S0 and S1.

```
module FSM (
    input  logic    clk,    // clock
    input  logic    rst,    // rest state machine
    input  logic    in,      // input signal
    output logic    out     // output signal
);

    // define states
    typedef enum {S0, S1, S2, S3} my_states;
    my_state current_state, next_state;

    // state registers
    always_ff @(posedge clk)
        if (rst)     current_state <= S0;
        else         current_state <= next_state;

    // next state logic
    always_comb
        case (current_state)
            S0: next_state = S1;
            S1: if (in) next_state = S3;
                else next_state = S2;
            S2: if (in) next_state = S0;
                else next_state = S3;
            S3: if (in) next_state = S2;
                else next_state = S0;
        default: next_state = current_state;
        endcase

    // output logic
    assing out = S0 + S1;
endmodule
```

[4]

(iii) Since there are three Boolean variables to be produce: S0, S1, , and each LUT can only provide ONE output, we need a minimum of three LEs. Further, each 4-LUT can implement any Boolean equation with 4 variables, all three equations only have three variables, we only need three LE for the entire FSM.

[2]

Feedback:

Most students got (i) and (ii), particularly those actually did FSM designs in the lab. I marked this most leniently, not caring that much about syntax, but deducted marks for incompleteness.

Many students got (iii) wrong – showing lack of understanding how internal hardware of FPGA links to implementations.

(e) This question tests student's basic understand of op-amp, gain-bandwidth product specification, cascaded amplifiers and AC coupling.

(i)

$$V_{out} = -\frac{R2}{R1}(V_{in} - 2.5) + 2.5, \quad \text{therefore the gain is } -\frac{R2}{R1}.$$

[2]

(ii)

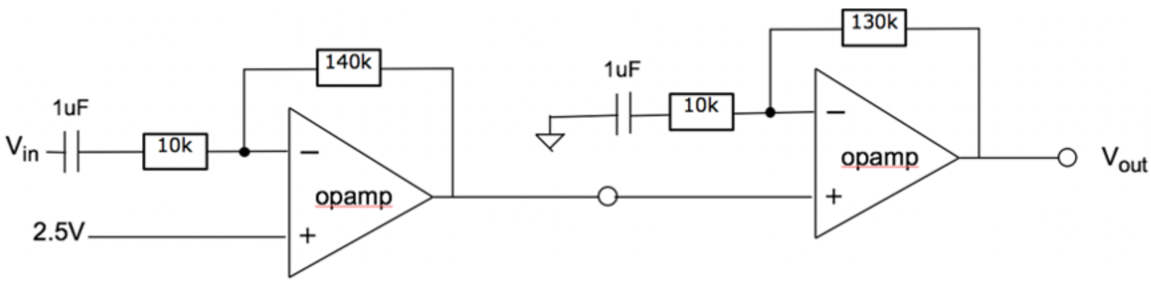
For a gain of -14 and given that R1 = 2.2k, R2 = 30.8k.

[2]

(iii) The gain-bandwidth product is 1MHz. Therefore, at 50kHz, the maximum gain is only x20. Furthermore, the amplifier is DC coupled. It needs to add a DC blocking capacitor at the input.

[2]

(iv) The 1kHz lower frequency requirement defines the capacitor coupling we need. With the circuit shown here, the lower corner frequency is compatible with the 1kHz lower limit:




Student's design may vary.

[4]

Feedback:
Most students got (i) correctly. Since the question only asked for the gain of the circuit, the full equation was not required. Most students did not consider how the DC offset propagates from stage 1 to stage 2. They lost 1 mark as a result. The exact gain of stage 1 and stage 2 does not matter as long as it is within the GBP specification of the opamp.

Overall feedback for Q1:

In general, students did well with Q1. The average was 32.8/50, or 65.6%. I normally aim to achieve a higher, around 70% – 75%. This suggests that I may have pitched Q1 questions slighted too hard this year.

2. This question tests student's ability to interpret datasheet of an analogue comparator and how it is used to perform analogue signal comparison. It also tests students understanding of hysteresis and the use of principle of superposition.

(a)

(i) **The input offset voltage $V_{OS}$** - determines how accurate is the comparator. From the datasheet, typical offset is 2mV with a worst-case condition of 10mV. That means this comparator would switch the output state within ±2mV difference in voltage between the differential inputs.

(ii) **Output sink current $I_O$** – The maximum current that the comparator output can absorb from its load. This has an "open collector" output state (2nd paragraph of the datasheet), therefore the output is a switching bipolar transistor that pulls the output pin low. There needs to be a pull-up resistor to the supply voltage. This resistor must be large enough so that it is significantly less than 6mA.

(i) **Voltage Gain $A_V$** – The comparator is no more than a special design op-amp. With large AV such as $120 \times 10^3$, the output of the comparator will switch quickly when the threshold voltage is crossed.

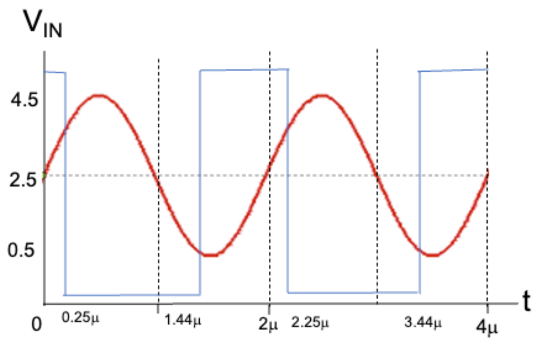
(ii) **Propagation Delay (High to Low) $t_{pHL}$** – There is a delay between the threshold voltage is crossed and the output switches states. That delay also depends on whether the output switches from H to L or from L to H. Here the delay is from H to L, and it is typically 250ns for an overdrive of 50mV – i.e. when the voltage difference between the two input terminals is 50mV or more.

[8]

Feedback:

Most students did well in (a). I was particularly lenient since the word "significance" in the question was interpreted differently by many in the cohort.

(b)




Not expecting additional assumptions, but students may state that the input current is negligible and the gain of the comparator is nearly infinite.

[7]

Feedback:

Many students found (b) difficult – not sure why. Many who did produce the output square signal had it inverted, not noticing that the input voltage $V_{IN}$ is applied to the negative input of the comparator.

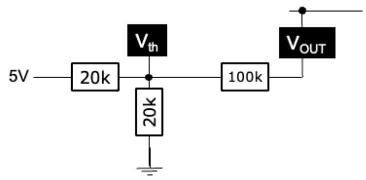
(c) Hysteresis is the change in threshold depending on direction of travel of the input signal. If the input signal is slow varying, the comparator output may exhibit oscillation during the time window when the two input signals are very close to each other. Such oscillation is avoided by design the comparator circuit with different threshold when input is increasing or decreasing relative to the reference (or threshold) voltage.

[4]

Feedback:

Those students who attempted (a) and (b) generally also did well with (c). Some drew a triangular signal in the explanation. This shows some undigested memory of the notes where hysteresis was exhibited in the function generator application of the op-amp where integrator was involved. There is integrator here, and therefore no triangular signal! Output switches between GND and V+ and immediately changes the threshold voltage of the comparator.

(d) The simplified circuit is:




First, note that 20k//100k = 16.7k.

When Vout = GND, Vth = (16.7/ 36.7) * 5V = 2.27V.

When Vout = 5V, Vth = (20/36.7) * 5V = 2.72V.

[6]

Feedback:
Most students with a clear head got this right. It is actually much easier than it may appear. Those who used KCL would have taken much longer and more prone to error. The right approach is to see the circuit shown above, and now the problem is a simple voltage divider with parallel resistors.

Overall feedback for Q2:

Question 2 has an expected average of 17/25, or 68% and a SD of 5.3%. This shows that most students attempted this successfully.

3. This question tests student's understanding of circuits that involve both analogue and digital components. They have not encountered dual-slop ADC before, but they know all the components that make up such a ADC.

(a) This is a well-known technique to perform ADC for very slow varying signal. We basically charge the capacitor C with a current that is proportional to the input voltage, then discharge the capacitor at a fixed rate (slope).

The input voltage is integrated for 20msec. The slope of X is $\frac{dX}{dt} = -\frac{V_{in}}{RC}$. Therefore, the minimum voltage $X_{min} = -0.02 \times \frac{V_{in}}{RC}$.

The integrated input is then switched to integrating a fixed voltage of -10V, which as a fixed gradient of $\frac{dX}{dt} = \frac{10}{RC}$. Therefore, the time it takes for X to return to zero from $X_{min}$ is:

$$k = -X_{min} \times \frac{RC}{10} = \frac{V_{in}}{10} \times 20ms = V_{in} \times 0.02 \text{ second.}$$

[6]

Feedback:
A significantly large fraction of the cohort did not attempt Q3 at all. Part (a) requires appreciation of the how the integrator charges and discharges due to the current. The idea, as stated in the question, is to charge the capacitor with a current proportional to the input voltage for a FIXED period of 20ms, then discharge the capacitor with a fixed current until the output returns to the same starting voltage (which is 0V). The rest is easy! Many students did not get the concept and therefore had very muddled answer.

(b) Since the reference voltage is -10V, the maximum Vin must be 10V or lower. Also, *Vin* cannot be negative, otherwise X will keep decreasing until it reaches -15V. Therefore, *Vin* has a range of 0 to 10V.

[2]

Feedback:
Most students did not get (b) at all. Many responded with ±15V, which is the range of the power supply. Some got the sign wrong, but no mark was deducted.

(c) At most k can be 20ms. Therefore, the worst-case conversion time is 40ms. Hence the maximum sampling rate is 1/40ms = 25 samples/second.

[2]

Feedback:
Many students attempt this and applied the Sampling (or Nyquist) Theorem. This demonstrates lack of understanding of both. The question has nothing to do with Sampling Theorem! Some gave the answer of $f_{max}$ = 1/(20ms + k), which is strictly not correct because k should be $k_{max}$. But no mark was deducted for this.

(d) Assuming that lowest $X_{min}$ is -10V, then RC = 0.02sec. If C = 1uF, then R = 20kΩ.

[2]

Feedback:
Many got this correct despite not able to do the other part of the question.

(e) We need to derive two timing signals from the 50MHz clock:

1. A 20ms tick signal *tick_20ms* for the VIN integration period.
2. A much faster tick signal *tick_5us* to count the duration of k that provides the 12-bit digital output *data[11:0]*.

Maximum k value is 20ms when data[11:0] = 4000. Therefore we need to generate a tick with a period of 20ms/4000 = 5uS.

To generate 5us tick, we need to divide 50MHz clock by 250 (or 200kHz tick). To generate 20ms tick, we further divide 200kHz tick by 4000.

Here is a model SystemVerilog design as a single module. Student may choose to divide this into separate modules.

[13]

```
module controller (
    input  logic    clk,    // clock at 1MHz
    input  logic    start,  // start conversion signal
    input  logic    Y,      // comparator signal 1 = end of
    output logic    done,   // high if new converted data is available
    output logic    C0,     // S0 control signal
    output logic    C1,     // S1 control signal
    output logic [11:0] data    // converted data 12-bits
);

    // CTR1 - 5us tick generator
    logic tick_5us;             // 5 us tick generator
    logic [7:0] counter1;        // interal counter to generate tick
    initial counter1 = 8'd249;
    always_ff @(posedge clk)
        if (counter1==8'b0) begin
            counter1 <= 8'b249;
            tick_5us <= 1'b1;
        end
        else  begin
            counter <= counter - 1;
            tick_5us <= 1'b0;
        end

    // CTR2 - 20ms tick generator
    logic tick_20ms;            // 20ms tick generator
    logic [11:0] counter2;        // interal counter to generate tick
    initial counter2 = 12'd3999;
    always_ff @(posedge clk)
        if (tick_5us)
            if (counter2==12'b0) begin
                counter2 <= 12'd3999;
                tick_20ms <= 1'b1;
            end
            else  begin
                counter2 <= counter2 - 1;
                tick_20ms <= 1'b0;
            end

    // CTR3 - counting 5us tick between t1 and t2 (see Figure 3.2)
    logic [11:0] data;                // result counter
    always_ff @(posedge tick_5us)   // deliberately use asynchronous clock to simplif
        if (start==1'b1)
            data <= 12'b0;
        else if (C1==1'b1)
            data <= data + 1'b1;

    // The FSM has three states - waiting to start, charge cap for 20ms
    // .... and dischargeeg cap at fixed rate while counting

    typedef enum {IDLE, CHARGING, DISCHARGING} my_states;
    my_state current_state, next_state;
    initial current_state = IDLE;

    // state transition
    always_ff @(posedge clk)
        current_state <= next_state;

    // next state logic
    always_comb
        case (current_state)
            IDLE:        if (start==1'b1)
                             next_state = CHARGING;
                         else next_state = IDLE;
            CHARGING:    if (tick_20ms==1'b1) next_state = DISCHARGING;
                         else next_state = CHARGING;
            DISCHARGING: if (Y=='1b0) next_state = IDLE;
                         else next_state = DISCHARGING;
            default: next_state = current_state;
        endcase

    // output logic
    always_comb
        case (current_state)
            IDLE:        assign {C1, C0, done} = 3'b001;
            CHARGING:    assign {C1, C0, done} = 3'b010;
            DISCHARGING: assign {C1, C0, done} = 3'b110;
        endcase
endmodule
```

Feedback:

Many student attempted this in spite of not answering other part of Q3 because the circuit's block diagram is given in the question. Of course, without understanding how the ADC works, they did not implement the FSM and therefore received only some marks for the attempt and implementation of the counters.

Overall feedback for Q3:

This is indeed a hard question and requires quite a lot of reading and understanding. A large proportion of students did not attempt Q3 or only answer small part of it. The average mark is low: 6.2/25 or 24.8%. The SD is 5.5%. I think the low number of attempts is due to the paper as a whole takes too long – many students simply ran out of time. Notwithstanding a few students answer this question nearly perfectly, showing that it is "do-able".

Overall comments on the entire paper:

The external examiner commented that perhaps this paper is "too long", and I think with hindsight, the external was correct. The overall average 57.2% for the entire class with a SD of 16.6%, which is also a high standard deviation. It shows that the paper probably has mostly difficult questions, resulting in large variations – some can do it, and others simply can't.

The key statistics for the examination paper (60% of module) are:

Average: 57.2%

SD: 16.6%

Exam grade distribution is:

| Exam Only | | |
|---|---|---|
| Grade | Count | % |
| A | 15 | 23 |
| B | 16 | 25 |
| C | 15 | 23 |
| D | 10 | 15 |
| E | 9 | 14 |

Fortunately, the examination only counts for 60% of the module and most students did well in the two Lab Orals because of the very high degree of engagement among the cohort.

The key statistics for the Coursework (40% of module) are:

Average: 67.5%

SD: 7.3%

The key statistics for the entire module are:

Average: 61.4%

SD: 12%

Module grade distribution is:

| Module | | |
|---|---|---|
| Grade | Count | % |
| A | 16 | 25 |
| B | 23 | 35 |
| C | 17 | 26 |
| D | 7 | 11 |
| E | 2 | 3 |

I also computed the correlation between Coursework, Examination and Module marks. It is:

| Type | Correlation |
|---|---|
| Exam vs Final | 0.98 |
| CW vs Final | 0.76 |
| CW vs Exam | 0.62 |

This shows the importance of engagement with the laboratory and coursework. The correlation coefficient of CW and final marks is very high: 0.76!