EEE4120F High Performance Embedded Systems

Prep Test Opportunity 1

Marking Memo

Date: [Insert Date]
Total Marks: 100

Section 1: Multiple Choice Questions (40 Marks)

Allocate 8 marks for each correct answer.

Question 1.1 Correct Answer: B) To verify the functional correctness of a Verilog module by providing inputs and checking outputs. *(Explanation: A testbench is specifically designed for simulation and verification, not for synthesis, timing analysis, or physical constraints.)*

Question 1.2 Correct Answer: B) Blocking assignments update variables immediately, while non-blocking assignments schedule updates to happen at the end of the current time step. *(Explanation: This is the fundamental difference. Blocking assignments execute sequentially within the same always block, potentially affecting subsequent assignments in the same delta cycle. Non-blocking assignments are essential for sequential logic as they prevent race conditions by scheduling updates after all right-hand sides have been evaluated for the current time step.)*

Question 1.3 Correct Answer: B) The order in which bytes (or larger data units) are stored in memory or transmitted over a bus. *(Explanation: Endianness deals with the byte order of multi-byte data types, such as integers, in memory or during data transfer.)*

Question 1.4 Correct Answer: C) The next-state logic. *(Explanation: In an FSM, the next-state logic determines the next state based on the current state and inputs, effectively describing the transitions.)*

Question 1.5 Correct Answer: C) Look-Up Table (LUT) *(Explanation: LUTs, along with flip-flops, are the primary configurable logic blocks within an FPGA, implementing combinational logic functions.)*

Section 2: Long Answer Questions (60 Marks)

Question 2.1: BoLaSpl Verilog Module Implementation (45 Marks)

Approach, Calculations, and Pseudocode (Max 10 Marks out of 45):

- **Approach (4 Marks):**
  - Student should propose a state machine approach (e.g., IDLE/RESET, WAIT_BDS1, APPROACHING, CALCULATE_DISPLAY).
  - Mention using fclk for precise time measurement and clk for the 10-second display timer.
  - Mention edge detection for START, bds_1, bds_2.
- **Calculations (3 Marks):**
  - Distance between bds_1 and bds_2 is 3 feet.
  - Speed (fps) = Distance (ft) / Time (s).
  - Time (s) = fclk_timer_value / 1,000,000 (since fclk is 1MHz).
  - Therefore, Speed (fps) = 3 / (fclk_timer_value / 1,000,000) = (3 * 1,000,000) / fclk_timer_value.
  - Inputs for udiv32: a = 3_000_000, b = fclk_timer_value. (Note: 3_000_000 fits in 22 bits, so 32-bit inputs for udiv32 are sufficient).
- **Pseudocode (3 Marks):** A clear, concise pseudocode snippet outlining the main states, transitions, and key actions within each state. For example:

```
STATE S_IDLE:
    Set START_LED=1, others=0, display_on=0.
    If posedge START, go to S_WAIT_BDS1.

STATE S_WAIT_BDS1:
    START_LED=1.
    If posedge bds_1, go to S_APPROACHING.

STATE S_APPROACHING:
    APPROACH_LED=1.
    Start fclk_timer.
    If posedge bds_2:
        Stop fclk_timer.
        Go to S_CALCULATE_DISPLAY.

STATE S_CALCULATE_DISPLAY:
    CALC_LED=1, display_on=1, speed_out = (3_000_000 /
fclk_timer_value).
    Start clk_10s_timer.
    If clk_10s_timer reaches 10s:
        Go to S_IDLE.
```

Verilog Code (Max 35 Marks out of 45):

```
Verilog
module BoLaSpI (
    input wire clk,     // 1KHz clock (1ms period)
    input wire fclk,    // 1MHz clock signal (1us period)
    input wire START,   // System activation button (posedge)
    input wire bds_1,   // First ball detection sensor
(posedge if ball detected)
    input wire bds_2,   // Second ball detection sensor
(posedge if ball detected)
    output reg START_LED, // system ready for a new ball
    output reg APPROACH_LED, // ball passed bds_1 and
                             // is approaching bds_2
    output reg CALC_LED,  // calculation progress/completed
    output wire display_on, // Controls speed display module
                            // (1=turn on, 0=turn off)
    output wire [32:0] speed_out // Output calculated speed
);

// --- Parameters ---
parameter S_IDLE = 3'b000; // Initial state: waiting for START
parameter S_WAIT_BDS1 = 3'b001; // START press, wait for bds_1
parameter S_APPROACHING = 3'b010; // bds_1 detect, timing ball
parameter S_CALCULATE_DISPLAY = 3'b011;
              // bds_2 detected, calculating/displaying

parameter FCLK_PER_SEC = 1_000_000;  // 1MHz
parameter CLK_PER_10S = 10_000;      // 10 seconds * 1KHz

// --- Internal Registers & Wires ---
reg [2:0] current_state, next_state;

// Edge detection registers
reg START_d0, START_d1;
reg bds_1_d0, bds_1_d1;
reg bds_2_d0, bds_2_d1;

wire posedge_START;  // for connecting up i/o
wire posedge_bds_1;
wire posedge_bds_2;

// High-res timer for ball speed (fclk)
reg [24:0] fclk_timer_count; // Max time for 3ft (e.g., 5mph =
0.4s, 400_000 cycles, needs 19 bits)
           // Using 25 bits to be safe, allows up to 33s.

// 10-second display timer (clk)
reg [13:0] clk_10s_timer_count;
     // Max count 10,000 for 10s at 1KHz
// Internal signals for display
reg display_on_reg;
reg [32:0] calculated_speed;

// Assign outputs from internal registers
assign display_on = display_on_reg;
assign speed_out = calculated_speed;

// --- Instantiate Helper Modules ---
wire [32:0] udiv_result;
udiv32 i_udiv32 (
    .result(udiv_result),
    .a      (33'd3_000_000), // Numerator: 3 feet * 1,000,000
(fclk cycles per sec)
    .b      (fclk_timer_count) // Denominator: Time in fclk
cycles
);

display_speed i_display_speed (
    .on(display_on),
    .speed(speed_out)
);

// --- Edge Detection Logic ---
always @(posedge clk) begin // Or posedge fclk for faster
 // sampling, but clk is usually fine for buttons/sensors
    START_d0 <= START;
    START_d1 <= START_d0;
    bds_1_d0 <= bds_1;
    bds_1_d1 <= bds_1_d0;
    bds_2_d0 <= bds_2;
    bds_2_d1 <= bds_2_d0;
end

assign posedge_START = START_d1 == 1'b0 && START == 1'b1;
assign posedge_bds_1 = bds_1_d1 == 1'b0 && bds_1 == 1'b1;
assign posedge_bds_2 = bds_2_d1 == 1'b0 && bds_2 == 1'b1;

// --- FSM State Register ---  could be done more elegantly
always @(posedge clk) begin
    current_state <= next_state;
end
```

// NB: this could is more lengthy than necessary, could be simpler; but this gives a thorough response to the coding, however this level of completeness and amount of coding would not be expected in an exam, indeed this development activity ended up being rather lengthier than anticipated for a test like this than what was anticipated at the time the question was written.

// --- Next State Logic (Combinational) ---

```
always @(*) begin
  next_state = current_state; // Default to self-loop

  // Default output values for safety or for transitions
  START_LED = 0;
  APPROACH_LED = 0;
  CALC_LED = 0;
  display_on_reg = 0; // Default off, turned on in specific state
  // calculated_speed remains at previous value or 0 if not calculated

  case (current_state)
    S_IDLE: begin
      START_LED = 1; // Only START_LED on
      if (posedge_START) begin
        next_state = S_WAIT_BDS1;
      end
    end

    S_WAIT_BDS1: begin
      START_LED = 1; // START_LED remains on until bds_1
      if (posedge_bds_1) begin
        next_state = S_APPROACHING;
      end
    end

    S_APPROACHING: begin
      APPROACH_LED = 1; // APPROACH_LED on
      if (posedge_bds_2) begin
        next_state = S_CALCULATE_DISPLAY;
      end
    end

    S_CALCULATE_DISPLAY: begin
      CALC_LED = 1;
      display_on_reg = 1; // Turn on display
      // Speed calculation result comes from udiv32 instance
      calculated_speed = udiv_result;

      if (clk_10s_timer_count == CLK_PER_10S - 1) begin
        next_state = S_IDLE; // After 10s, return to IDLE
      end
    end

    default: begin
      next_state = S_IDLE; // Safety default
    end
  endcase
```

```
end

// --- Counter and Timer Logic (Synchronous) ---
always @(posedge fclk) begin // fclk for precise timer
  if (current_state == S_APPROACHING) begin
    fclk_timer_count <= fclk_timer_count + 1;
  end else if (current_state == S_WAIT_BDS1) begin // Reset timer if START is pressed
again before bds_1
    fclk_timer_count <= 0;
  end else if (current_state == S_IDLE) begin
    fclk_timer_count <= 0; // Ensure reset in idle
  end
  // fclk_timer_count holds its value when not in S_APPROACHING or S_IDLE
end

always @(posedge clk) begin // clk for 10-second display timer
  if (current_state == S_CALCULATE_DISPLAY) begin
    if (clk_10s_timer_count < CLK_PER_10S - 1) begin
      clk_10s_timer_count <= clk_10s_timer_count + 1;
    end else begin
      clk_10s_timer_count <= 0; // Reset after 10s duration
    end
  end else begin
    clk_10s_timer_count <= 0; // Reset counter when not displaying
  end
end

endmodule
```

Marking Allocation (35 Marks for Code):

- **FSM Structure & State Transitions (8 marks):**
    - Defining clear states (e.g., IDLE, WAIT_BDS1, APPROACHING, CALCULATE_DISPLAY).
    - Correctly implementing state transitions based on inputs (START, bds_1, bds_2) and timers.
    - Synchronous state register update (always @(posedge clk) current_state <= next_state;).
- **LED Control (6 marks):**
    - Correctly turning on/off START_LED, APPROACH_LED, CALC_LED in appropriate states/transitions.
- **Precise Time Measurement (fclk_timer) (8 marks):**
    - Correctly starting the fclk_timer_count on bds_1 detection.
    - Stopping the fclk_timer_count on bds_2 detection.
    - Using fclk for this timer (always @(posedge fclk)).
    - Appropriate bit-width for fclk_timer_count.
- **Speed Calculation (7 marks):**

    - Correctly instantiating udiv32 module.
    - Correctly providing a = 3_000_000 (or equivalent scaled value for 3ft and 1MHz clock).
    - Correctly providing b = fclk_timer_count.
    - Connecting udiv_result to speed_out.
- **10-Second Display Timer (clk_10s_timer) (4 marks):**
    - Correctly starting the clk_10s_timer_count when speed calculation is done.
    - Stopping/resetting the timer after 10 seconds (10,000 clock cycles).
    - Using clk for this timer (always @(posedge clk)).
- **Display Control (2 marks):**
    - Asserting display_on when speed is being calculated/displayed.
    - De-asserting display_on afterwards.
    - Connecting speed_out to display_speed module.
- **Code Quality & Comments (Bonus/Deduction):** Reward clear, commented code; deduct for very messy/unreadable code.

Marker's Comment:

When marking Question 2.1, please remember that the provided sample solution is just one possible approach. Students may use different state machine structures (e.g., explicit state register with separate next-state/output logic, or a single always block combining these), different counter implementations, or slightly varied timing mechanisms (e.g., specific reset conditions for timers). The focus of the marking should be on assessing the student's logical process in translating the requirements into Verilog, the clarity and readability of their code (including comments), and their understanding of fundamental Verilog concepts for synchronous design, FSMs, and timing. Award marks based on the correct implementation of the required behaviors (LED sequences, accurate time measurement, correct speed calculation, display control, 10s timer, state transitions) rather than strict adherence to the sample solution's exact code structure or variable names. A well-commented, logically sound, even if slightly different, solution should be rewarded.

Question 2.2: BoLaSpl Enhancement (15 Marks)

Conceptual Explanation (15 Marks):

Students should explain their approach clearly without providing full Verilog code. Key points to cover:

1. **New State/Error Detection Logic (5 marks):**
   - Introduction of a new state, e.g., S_ERROR_XXX in the FSM.
   - Modify the S_WAIT_BDS1 or S_APPROACHING state logic to detect the invalid condition: if bds_2 posedge occurs *before* bds_1 posedge (or before S_APPROACHING state is entered). This often involves checking bds_2's edge in S_WAIT_BDS1 or S_IDLE.
   - Transition to the new S_ERROR_XXX state upon detection of this condition.
2. **Error Display and Timer (5 marks):**
   - In the S_ERROR_XXX state:
     - Activate the display_txt module (hypothetical, but student should mention its use) to show "XXX". This would imply a new output signal for display_txt (e.g., display_txt_on) and a data input (e.g., display_text_value).
     - Initiate a 1-second timer (using clk if display_txt is driven by 1KHz). This would be a separate counter or re-use clk_10s_timer_count with a different target.
3. **Return to Initial State (5 marks):**
   - After the 1-second S_ERROR_XXX timer expires, the system should transition back to the S_IDLE state (or equivalent initial state).
   - Ensure all LEDs are reset to the initial configuration (START_LED on, others off) and display_on is off (and display_txt_on for "XXX" is off).

Example Conceptual Snippet (optional for student, but helpful for marker):

```
// In FSM next_state logic
case (current_state)
    S_IDLE: begin
        // ... (existing logic) ...
        if (posedge_bds_2) begin // If bds_2 pressed before
START or bds_1
            next_state = S_ERROR_XXX;
        end
    end
    S_WAIT_BDS1: begin
        // ... (existing logic) ...
        if (posedge_bds_2) begin // If bds_2 pressed before
bds_1
            next_state = S_ERROR_XXX;
        end
    end
    S_ERROR_XXX: begin
        // Activate 'display_txt_on'
        // Count 1 second using clk_timer
        if (error_timer_expired) begin
            next_state = S_IDLE;
        end
    end
    // ... (other states) ...
endcase

// And output logic for display_txt_on based on S_ERROR_XXX
state.
```

Question 1 MCQ Answers

1.1	**B**
1.2	**B**
1.3	**B**
1.4	**C**
1.5	**C**

End of Memo