EECS 151/251A: Homework № 3

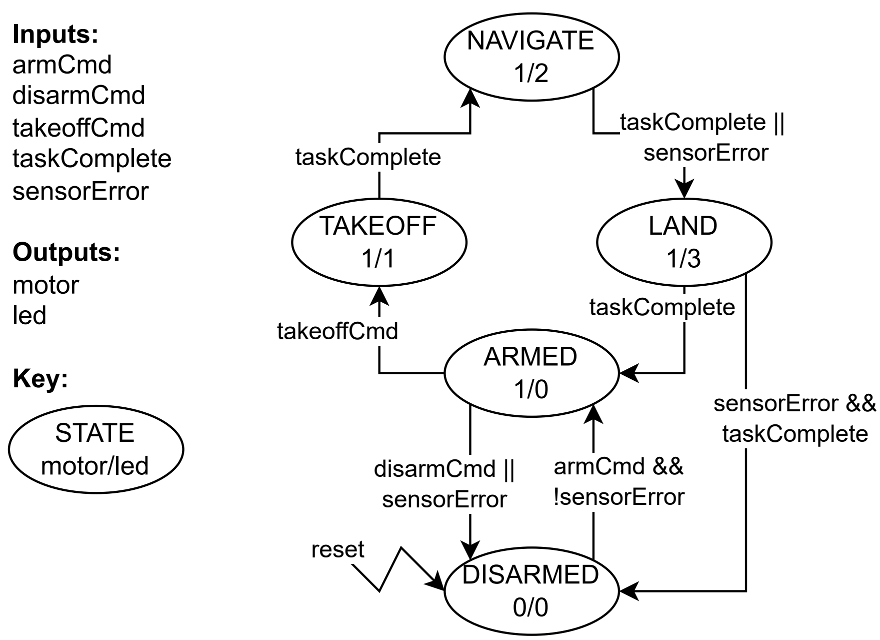
Due Friday, February 18th

Problem 1: FSM

You have been tasked with designing a custom hardware FSM for managing the state of an autonomous drone. The desired state transition diagram depicted below.

The system inputs are `armCmd`, `disarmCmd`, and `takeoffCmd`, which are commands provided by the autonomous controller,`taskComplete`, which signals when the current maneuver completes, and `sensorError`, which indicates that sensor fault has occurred and the FSM must override the controller's commands and return the drone to a safe state.

The system outputs are `motor`, routing power to the drone's motors for flight, and `led`, a status LED onboard for debugging.



Part a) Is this FSM a Mealy or Moore Machine?

Part b) Complete the Verilog module below to implement the specified FSM. In addition to the system inputs, the module also takes in a system clock and reset signal.

```
`define DISARMED    3'd0
`define ARMED       3'd1
`define TAKEOFF     3'd2
`define NAVIGATE    3'd3
`define LAND        3'd4

module droneFSM(
    input reset, clk,
    // TODO
    );

    // Internal state variables
    reg [2:0] state;
    reg [2:0] nextState;

    // Combinational assignments for output logic
    assign motor = // TODO
    assign led   = // TODO

    // Combinational block for next-state logic
    always @(*) begin
        case (state)
            `DISARMED: // TODO
            `ARMED:    // TODO
            `TAKEOFF:  // TODO
            `NAVIGATE: // TODO
            `LAND:     // TODO
            default:   nextState = state;
        endcase
    end

    // Sequential block for state transitions
    always @(posedge clk) begin
        if (reset) begin
            // TODO
        end else begin
            // TODO
        end
    end
endmodule
```

Part c) A system designer wants to immediately set the LED to 5 whenever a `sensorError` occurs, without waiting for a clock edge. Is this possible without changing if this is a Mealy/Moore Machine? If not, what type of FSM would the system become?

Part d) Another system designer wants to add some safety features to the drone. If any `sensorError` occurs, even if it is de-asserted later, the drone must land and be unable to be armed until the system is reset.

The designer suggests adding `LAND_ERR` and `DISARMED_ERR` states to track if a sensor error occurs at any point. Draw an updated FSM that implements these changes.

Problem 2: RISC-V Instructions

Consider the following potential new 32-bit RISC-V instructions. Consider whether or not they are feasible to implement, and if so, which of the 32-bit instruction formats could be used?

If one instruction is not feasible, is it possible to implement the instruction as a sequence of existing instructions? If so, list the sequence.

Note: We recommend referring to the RISC-V specification found here, as well as the RISC-V green card.

Part a) An integer power function, `pow rd, rs1, rs2`, defined as `rd = rs1 ** rs2`

Part b) An integer square root, `isqrt rd, rs2`, defined as `rd = sqrt(rs1)`

Part c) A three integer addition, `add3, rd, rs1, rs2, rs3`, defined as `rd = rs1 + rs2 + rs3`

Part d) A three integer accumulating addition, `add3, rd, rs1, rs2`, defined as `rd = rd + rs1 + rs2`

Part e) Add a 20-bit immediate, `addi20, rd, imm20`, defined as `rd = rd + imm20`

part f) Branch if integers are within a threshold, `brth, rs1, rs2, rs3, imm`, defined as `pc = abs(rs1 - rs2) < rs3 ? pc + imm : pc + 4`

Problem 3: Hand Assembly

Manually construct the binary instruction for the following assembly instructions. Submit all of the following for each instruction:

- The 32-bit binary number for the instruction
- The core instruction format it belongs to
- Delineate the 32 bits into the subfields of the instruction format and label each field with the opcode/registers/immediate/offset etc. specified by the instruction.

Note: we highly encourage you to do this by hand from the ISA spec, but it is possible to assemble them using RISC-V GCC or venus.

Part a) `sub x4, x2, x1`

Part b) `xori x2, x3, 15`

Part c) `lb x3, 8(x5)`

Part d) `jalr x10, x11, 2048`

Problem 4: Assembly Execution

Write down the values of the specified registers after the following programs have run. Show your work by annotating the what happens/changes after each instruction. Note that some instructions are pseudo-instructions, such as `li` for load immediate. Refer to Table 25.2 in the RISC-V spec for a list of pseudo-instructions and their base implementations.

Part a)

```
li x0, 30
li x1, 10
addi x2, x0, 20
sub x2, x2, x1
```

```
x0 = ________
x1 = ________
x2 = ________
```

Part b)

```
li x1, 0xbeef
li x2, 0x64
sb x1, 0(x2)
sra x1, x1, x2
sb x1, 1(x2)
sb x1, 2(x2)
sra x1, x1, x2
sb x1, 3(x2)
lw x3, 0(x2)
```

```
x1 = ________
x2 = ________
```

Part c)

```
    li x1, 1
    slli x1, x1, 31
    li x2 1
    li x3 -1
    li x4 0
f1: sra x1 x1 x2
    addi x4 x4 1
    blt x1 x3 f1
```

```
x1 = ________
x2 = ________
x3 = ________
x4 = ________
```