P.S. this is a poorly designed question because if combo == 4'b1111, then auto unlock ... :((thanks Peyton/Andrew)

6.004 Tutorial Problems
L11B – Sequential Circuits in Minispec

Note: A subset of problems are marked with a red star (★). We especially encourage you to try these out before recitation. You may prefer to complete these problem in the 6.004 JupyterHub.

Problem 1. ★

Remember the implementation of the 4-bit Lock module from the L11A worksheet. Below is a variant, `Lock4`, that matches against an arbitrary pattern, given as a *module argument*. Use `Lock4` to implement `Lock8`, a lock module that unlocks with an 8-bit combination.

```
module Lock4(Bit#(4) combo);
    Reg#(Bit#(4)) lastFourBits(4'b1111);

    input Bit#(1) in;

    rule tick;
        lastFourBits <= {lastFourBits[2:0], in};
    endrule

    method Bool unlock = (lastFourBits == combo);
endmodule

module Lock8(Bit#(8) combo);

    Lock4 upper(combo[7:4]);

    Lock4 lower(combo[3:0]);

    Reg#(Bit#(4)) lastFourBits(-1); // Hint: You need some extra state
                                    // to make both locks operate in sync
    input Bit#(1) in;

    rule tick;
        lower.in = in;

        upper.in = lastFourBits[3];

        lastFourBits <= {lastFourBits[2:0], in};

        ;

    endrule

    method Bool unlock = ;
endmodule
```

Run both simultaneously

Variant 1: Run upper 4 cycles behind

Variant 2: Run upper & lower in lockstep, remember upper's decision
- if upper unlocked 4 cycles ago, then if lower is also unlock

combo[7:4]

combo[3:0]

Reg#(Bit#(4)) lastUpperUnlocks(0);

lower.in = in;

upper.in = in;

lastUpperUnlocks <= {lastUpperUnlocks[2:0], upper.unlock?1:0}

upper.unlock && lower.unlock

(lastUpperUnlocks[3] == 1) && lower.unlock

Problem 2. ★

read in every digit to store inside lastBits. Then check if lastBits unlocks the combo

(A) Composing two 4-bit Lock modules to make an 8-bit Lock module is kludgy. Instead, we can make a parametric module, `Lock#(n)`, that unlocks on an n-bit combination sequence (given as a module argument). Implement `Lock#(n)` by filling out the code skeleton below.

```
module Lock#(Integer n)(Bit#(n) combo                    );

    Reg#(Bit#(n)                    ) lastBits(-1                    );

    input Bit#(1) in;

    rule tick;

        lastBits <= {lastBits[n-2:0], in}                    ;
    endrule

    method Bool unlock = (lastBits == combo)                    ;
endmodule
```

(B) Test your `Lock#(n)` module by completing the testbench module below, called `LockTest`. Ideally, your testbench should test all possible 8-bit input sequences; at a minimum, it should check a few incorrect sequences as well as the correct sequence. Your testbench should print PASS if all tests are correct, and FAIL otherwise. You can add additional registers or submodules, though they aren't needed.

```
module LockTest;
    Bit#(8) combo = 8'b01100111;
    Lock#(8) lock(combo);
    Reg#(Bit#(16)) cycle(0);

    rule test;
        // Feed the lock all the possible input patterns
        // An easy way to do this is by giving it all possible
        // 8-bit sequences one after the other; this will have
        // many duplicate patterns, but covers the whole space
        Bit#(8) curPattern = cycle[10:3];
        Bit#(3) curIdx = cycle[2:0];
        lock.in = curPattern[7 - curIdx];


        cycle <= cycle + 1;
    endrule
endmodule
```

```
        // Check whether output matches what we expect.
        // We derive the last few bits from cycle,
        // but you could keep them in a register
        Bit#(16) inputWindow = {curPattern-1, curPattern};
        Bit#(4) winIdx = {0, curIdx};
        Bit#(8) lastBits = inputWindow[15-winIdx:8-winIdx];

        if (lastBits == combo && !lock.unlock) begin
            $display("FAIL: lock didn't unlock with correct combo");
            $finish;
        end
        if (lastBits != combo && lock.unlock) begin
            $display("FAIL: unlocked with wrong combo %b", lastBits);
            $finish;
        end

        // We pass once we've tested all the patterns
        if (cycle == 1<<12) begin
            $display("PASS after testing %d patterns", cycle-1);
            $finish;
        end

        cycle <= cycle + 1;
    endrule
endmodule


%%sim LockTest

PASS after testing  4095 patterns
```