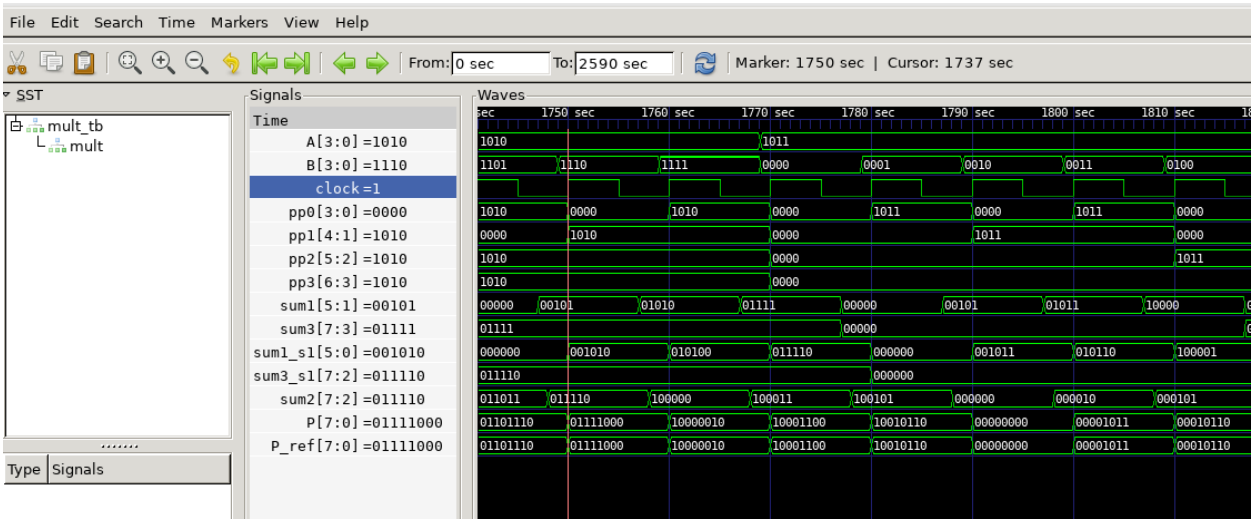
4. (a)

```
// pipelined fast multiplier
module mult_fast(
    output reg[7:0] P,  // product
    input[3:0] A, B,    // multiplicand and multiplier
    input clk           // clock (posedge)
    );
    // stage 0 (input)
    reg[3:0] a_s0, b_s0;
    always @(posedge clk) begin
        a_s0 <= A;
        b_s0 <= B;
    end
    // stage 1
    wire[3:0] pp0 = a_s0 & {4{b_s0[0]}}; // ignore the delays of AND gates
    wire[4:1] pp1 = a_s0 & {4{b_s0[1]}}; // ignore the delays of AND gates
    wire[5:2] pp2 = a_s0 & {4{b_s0[2]}}; // ignore the delays of AND gates
    wire[6:3] pp3 = a_s0 & {4{b_s0[3]}}; // ignore the delays of AND gates
    reg[5:1] sum1;
    always @(pp0, pp1)
        sum1[5:1] <= #7 pp0[3:1] + pp1[4:1]; // delay of the 4-bit adder
    reg[7:3] sum3;
    always @(pp2, pp3)
        sum3[7:3] <= #7 pp2[5:3] + pp3[6:3]; // delay of the 4-bit adder
    reg[5:0] sum1_s1;
    reg[7:2] sum3_s1;
    always @(posedge clk) begin
        sum1_s1 <= {sum1, pp0[0]}; // concat pp0[0] to sum1, and assign back to
sum1
        sum3_s1 <= {sum3, pp2[2]}; // concat pp2[2] to sum3, and assign back to
sum3
    end
    // stage 2 (outout)
    reg[7:2] sum2;
    always @(sum1_s1, sum3_s1)
        sum2[7:2] <= #8 sum1_s1[5:2] + sum3_s1[7:2]; // delay of the 6-bit adder
    always @(posedge clk) begin
        P <= {sum2, sum1_s1[1:0]}; //concat sum1[1:0] to sum2, and assign to P
    end
endmodule
```

(b)



(c). 16 ticks, because the worst case is that the previous input is still at stage 2, so 8 + 8.

5.

(a) min clock cycle = 8, because I set the delay of the clock to be 4. Hence, the clock will toggle its value every 4 time units and thus the clock cyle is 8.

(b)