Mark Mitri and Ryan Forelli 9/13/2021
ECE 128 Lab 3

Note: Parts 3 & 4 only asked for the flip-flop symbols. Gate-level designs were included as well in case of typo.

Part 1 RCA and CLA Simulation:

RCA Testbench code:

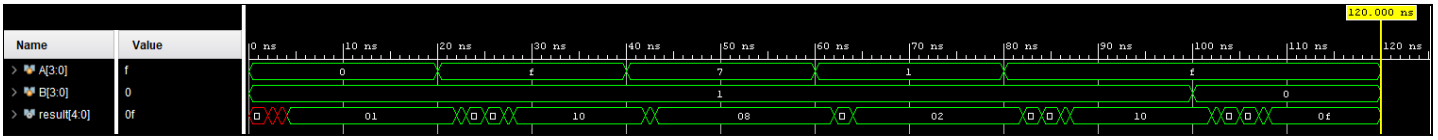
```
`timescale 1ns / 1ps

module adder_4bit_tb;

  reg[3:0] A, B;
  wire[4:0] result;

  ripple_adder_4bit adder1(A, B, result);

  initial begin

    A=4'b0000;B=4'b0001;#20 //0x0 and 0x1
    A=4'b1111;B=4'b0001;#20 //0xf and 0x1
    A=4'b0111;B=4'b0001;#20 //0x7 and 0x1
    A=4'b0001;B=4'b0001;#20 //0x1 and 0x1
    A=4'b1111;B=4'b0001;#20 //0xf and 0x1
    A=4'b1111;B=4'b0000;#20 //0xf and 0x0

    $finish; //end the simulation
  end

endmodule //circuit_tester
```

RCA Modified Verilog code:

```
`timescale 1ns / 1ps

module full_adder(input A, B, Cin, output SUM, Cout);
```

```
  wire wire1, wire2, wire3, wire4;

  xor #1 x1(wire1, A, B);
  and #1 a1(wire3, wire1, Cin);
  and #1 a2(wire4, A, B);
  or #1 o1(Cout, wire3, wire4);
  xor #1 x2(SUM, wire1, Cin);

endmodule

module ripple_adder_4bit(input[3:0] num1, num2, output[4:0] result);

  wire carry1, carry2, carry3;

  full_adder bit0(num1[0], num2[0], 1'b0, result[0], carry1);
  full_adder bit1(num1[1], num2[1], carry1, result[1], carry2);
  full_adder bit2(num1[2], num2[2], carry2, result[2], carry3);
  full_adder bit3(num1[3], num2[3], carry3, result[3], result[4]);

endmodule
```

RCA Simulation Waveform:



CLA Testbench Code:

```
`timescale 1ns / 1ps

module full_adder_carry_lookahead_tb;

  reg[3:0] A, B;
  wire[4:0] result;

  carry_lookahead_adder_4bit adder1(A, B, result);
```

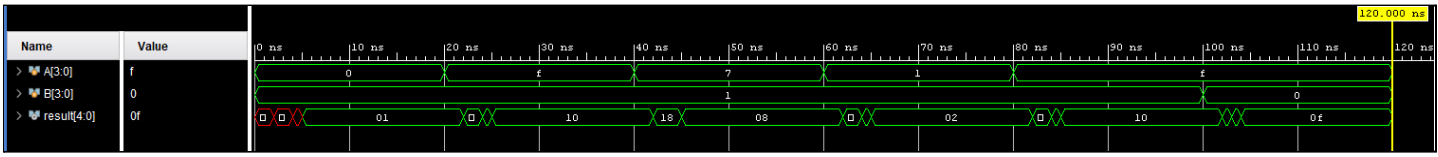
```
  initial begin

    A=4'b0000;B=4'b0001;#20 //0x0 and 0x1
    A=4'b1111;B=4'b0001;#20 //0xf and 0x1
    A=4'b0111;B=4'b0001;#20 //0x7 and 0x1
    A=4'b0001;B=4'b0001;#20 //0x1 and 0x1
    A=4'b1111;B=4'b0001;#20 //0xf and 0x1
    A=4'b1111;B=4'b0000;#20 //0xf and 0x0

    $finish; //end the simulation
  end
endmodule //circuit_tester
```

CLA Modified Verilog code:

```
`timescale 1ns / 1ps

module carry_lookahead_adder(input A, B, Cin, output SUM, Pi, Gi);

  wire wire1;

  xor #1 x1(wire1, A, B);
  xor #1 x2(SUM, wire1, Cin);
  or #1 o1(Pi, A, B);
  and #1 a1(Gi, A, B);

endmodule

module carry_lookahead_adder_4bit(input[3:0] num1, num2, output[4:0]
result);

  wire c1, c2, c3;
  wire p0, p1, p2, p3, g0, g1, g2, g3;
  carry_lookahead_adder bit0(num1[0], num2[0], 1'b0, result[0], p0,
g0);
  carry_lookahead_adder bit1(num1[1], num2[1], c1, result[1], p1,
g1);
  carry_lookahead_adder bit2(num1[2], num2[2], c2, result[2], p2,
```

```
g2);
  carry_lookahead_adder bit3(num1[3], num2[3], c3, result[3], p3,
g3); //Generate function gi is last carry

  wire wire1, wire2, wire3, wire4, wire5, wire6, wire7, wire8, wire9,
wire10, wire11, wire12;

  //Simplified version
  //C1=G0
  buf(c1, g0);

  //Simplified verion
  //C2=G1+P1G0
  and #1 a1(wire1, g0, p1);
  or #1 o1(c2, wire1, g1);

  //Simplified version
  //C3=G2+P2G1+P2P1G0
  and #1 a2(wire2, g1, p2);
  and #1 a3(wire3, p1, p2);
  and #1 a4(wire4, wire3, g0);
  or #1 o2(wire5, g2, wire2);
  or #1 o3(c3, wire4, wire5);

   //Simplified version, find c4 (MSB of answer)
  //C4=G3+P3G2+P3P2G1+P3P2P1G0
   and #1 a5(wire6, p3, p2);
   and #1 a6(wire7, p1, wire6);
   and #1 a7(wire8, g0, wire7);
   and #1 a8(wire9, g1, wire6);
   and #1 a9(wire10, g2, p3);
   or #1 o4(wire11, g3, wire10);
   or #1 o5(wire12, wire11, wire9);
   or #1 o6(result[4], wire12, wire8);
endmodule
```

CLA Simulation Waveform:



Simulation Result Table:

Time of input	input a	input b	RCA output	CLA output	RCA latency	CLA latency
0	0x0	0x1	0x01	0x01	4 ns	5 ns
20	0xf	0x1	0x10	0x10	8 ns	5 ns
40	0x7	0x1	0x08	0x08	3 ns	5 ns
60	0x1	0x1	0x02	0x02	4 ns	5 ns
80	0xf	0x1	0x10	0x10	7 ns	5 ns
100	0xf	0x0	0x0f	0x0f	8 ns	4 ns

Explanation:

We observe that the RCA latency is more sporadic than the CLA latency as each adder depends on the latency and output of the previous adder. In contrast, the CLA latency is more consistent. While the RCA latency is lower at times due to the greater amount of logic involved in the CLA, the consistency of the CLA latency is due to the carry-look-ahead logic as each carry is calculated separately and does not depend on the output of the previous adder.

Part 2 D Latch:

Testbench Code:

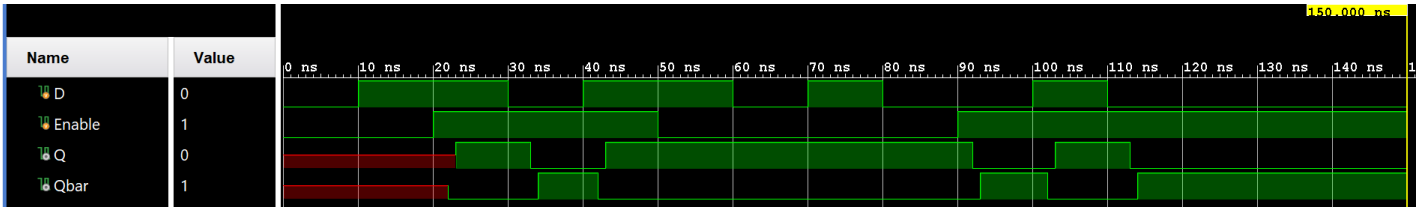
```
`timescale 1ns / 1ps

module d_latch_tb;

  reg D, Enable;
  wire Q, Qbar;

  d_latch latch1(D, Enable, Q, Qbar);
```

```
  initial begin
    D=1'b0;Enable=1'b0;#10
    D=1'b1;Enable=1'b0;#10
    D=1'b1;Enable=1'b1;#10
    D=1'b0;Enable=1'b1;#10
    D=1'b1;Enable=1'b1;#10
    D=1'b1;Enable=1'b0;#10
    D=1'b0;Enable=1'b0;#10
    D=1'b1;Enable=1'b0;#10
    D=1'b0;Enable=1'b0;#10
    D=1'b0;Enable=1'b1;#10
    D=1'b1;Enable=1'b1;#10
    D=1'b0;Enable=1'b1;#40

    $finish; //end the simulation
  end
endmodule
```

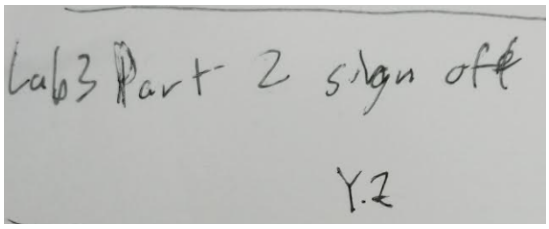
D Latch Design Code:

```
`timescale 1ns / 1ps

module d_latch(input D, Enable, output Q, Qbar);

  wire w1, w2, w3;

  not #1 n1(w1, D);
  and #1 a1(w2, w1, Enable);
  and #1 a2(w3, Enable, D);
  nor #1 no1(Q, w2, Qbar);
  nor #1 no2(Qbar, w3, Q);

endmodule
```

D Latch Simulation Waveform:

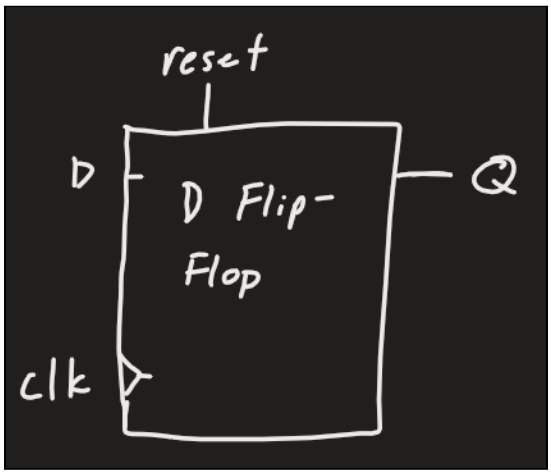


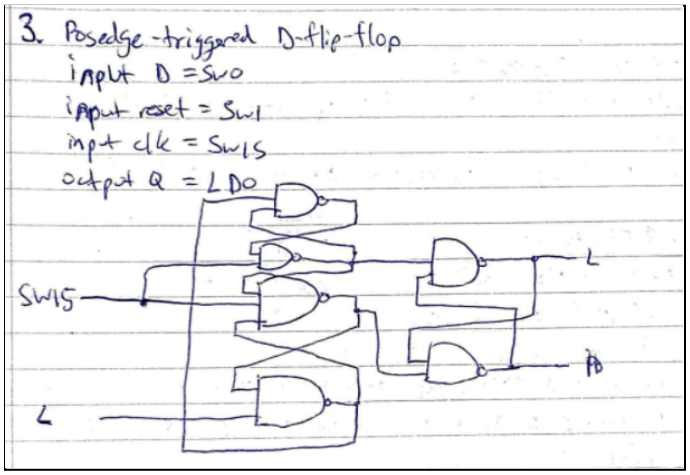
This waveform is expected as before enable is set high, the initial values of Q and Qbar are undetermined. When enable is set high, Q and Qbar are set according to D. When enable is set low at t=50ns, future changes in D do not affect the state of Q and Qbar. When enable is set high again at t=90ns, Q and Qbar are set again according to D until the end of the simulation.



Part 3 D Flip Flop:

Design Diagram:





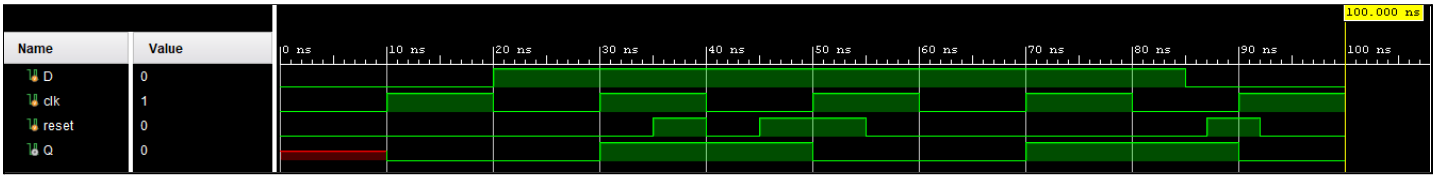
Testbench Code:

```
`timescale 1ns / 1ps

module d_flip_flop_tb;

  reg D, clk, reset;
  wire Q;

  d_flip_flop dflip1(D, clk, reset, Q);

  initial begin

    clk=1'b0;D=1'b0;reset=1'b0;#10
    clk=1'b1;D=1'b0;reset=1'b0;#10
    clk=1'b0;D=1'b1;reset=1'b0;#10
    clk=1'b1;D=1'b1;reset=1'b0;#5
    clk=1'b1;D=1'b1;reset=1'b1;#5
    clk=1'b0;D=1'b1;reset=1'b0;#5
    clk=1'b0;D=1'b1;reset=1'b1;#5
    clk=1'b1;D=1'b1;reset=1'b1;#5
    clk=1'b1;D=1'b1;reset=1'b0;#5
    clk=1'b0;D=1'b1;reset=1'b0;#10
    clk=1'b1;D=1'b1;reset=1'b0;#10
    clk=1'b0;D=1'b1;reset=1'b0;#5
    clk=1'b0;D=1'b0;reset=1'b0;#2
    clk=1'b0;D=1'b0;reset=1'b1;#3
    clk=1'b1;D=1'b0;reset=1'b1;#2
    clk=1'b1;D=1'b0;reset=1'b0;#8

    $finish; //end the simulation
  end
endmodule
```

Verilog Design Code:

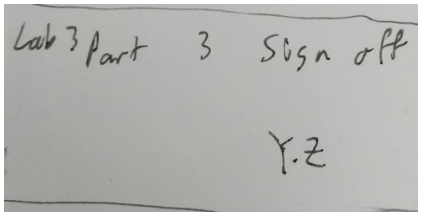
```
`timescale 1ns / 1ps
//D flip-flop using behavioral modeling

module d_flip_flop(
  input D, clk, reset,
  output reg Q
  );

  always @ (posedge clk) //Only run on positive clock edge
    begin
      if (reset) Q <= 1'b0;
      else
        if (D) Q <= 1'b1;
        else Q <= 1'b0;
    end
endmodule
```

Simulation Waveform:



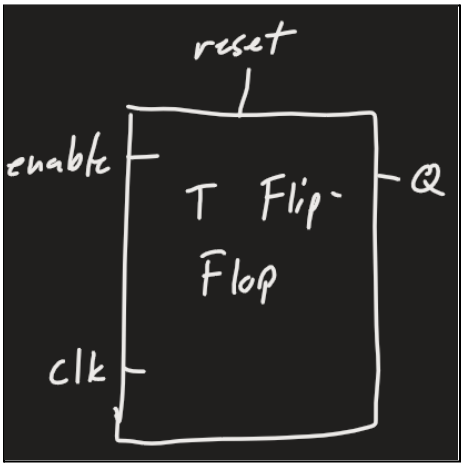
Explanation:

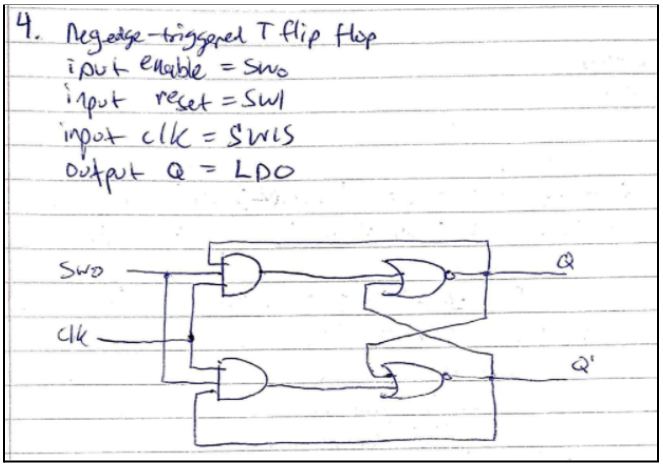
The waveform is as expected. The state of Q is initially unknown until the first positive clock edge where Q is set according to D which is low. D is set high on the negative clock edge at t=20ns which does not affect Q (as expected). On the next positive clock edge, Q is set high as D is still high. When reset is enabled at t=35ns, Q stays high as this flip flop has a **synchronous reset**. Later, when reset is enabled, Q is set low despite D being high at the positive clock edge at t=50ns. This behavior is expected from a synchronous posedge-triggered D flip flop.



Part 4 T Flip Flop:

Design Diagram:





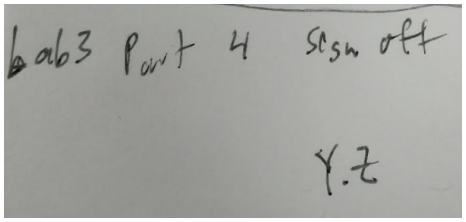
Testbench Code:

```
`timescale 1ns / 1ps

module t_flip_flop_tb;
  reg enable, clk, reset;
  wire Q;

  t_flip_flop tflip1(enable, clk, reset, Q);

  initial begin
    clk=1'b0;enable=1'b0;reset=1'b1;#10
    clk=1'b1;enable=1'b0;reset=1'b0;#10
    clk=1'b0;enable=1'b1;reset=1'b0;#10
    clk=1'b1;enable=1'b1;reset=1'b1;#5
    clk=1'b1;enable=1'b1;reset=1'b0;#5
```

```
    clk=1'b0;enable=1'b1;reset=1'b1;#5
    clk=1'b0;enable=0'b1;reset=1'b0;#5
    clk=1'b1;enable=0'b1;reset=1'b0;#5
    clk=1'b1;enable=1'b1;reset=1'b1;#5
    clk=1'b0;enable=1'b1;reset=1'b1;#10
    clk=1'b1;enable=1'b1;reset=1'b1;#10
    clk=1'b0;enable=1'b1;reset=1'b1;#5
    clk=1'b0;enable=1'b0;reset=1'b1;#2
    clk=1'b0;enable=1'b0;reset=1'b0;#3
    clk=1'b1;enable=1'b0;reset=1'b0;#2
    clk=1'b1;enable=1'b0;reset=0'b0;#8
    clk=1'b0;enable=1'b0;reset=0'b0;#10

    $finish; //end the simulation
  end
endmodule
```

Verilog Design Code:

```
`timescale 1ns / 1ps
//T flip-flop using behavioral modeling

module t_flip_flop(
  input enable, clk, reset,
  output reg Q
  );

  always @ (negedge clk) //Only run on negative clock edge
    begin
      if (!reset) Q <= 1'b0; //If reset is pulled low, Q is 0
      else
        if (enable) Q <= ~Q;
        else Q <= Q;
    end
endmodule
```


Lab3 Part 4 scsw off
Y.Z