Christopher Badolato

Flip Flop Fundamentals

EEE 3342 LAB 0014

11/25/2018

**Objective:**

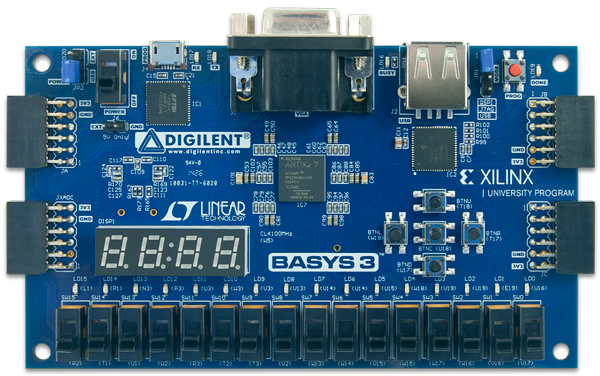
* To build and investigate the operation of an asynchronous SR flip-flop, a clocked SR flip-flop, and a clocked D flip-flop.

Apparatus List:

Xilinix’s FPGA VIVADO HLX edition

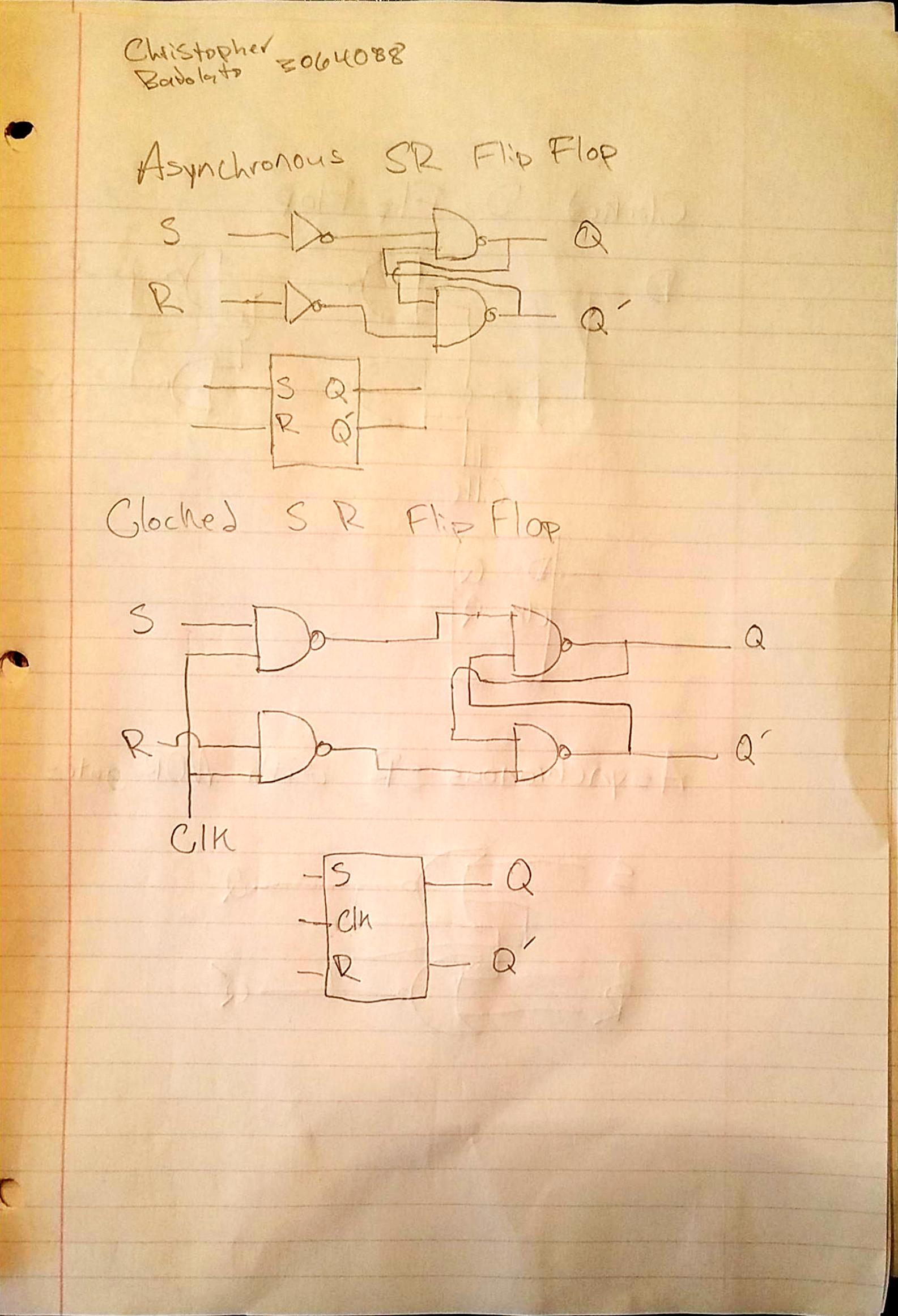
VIVDAO 2017.4

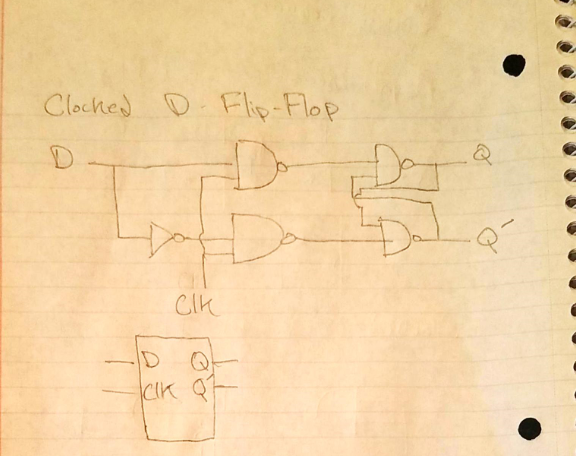
Basby Atrix-7



Procedure and/or Design Methodology:

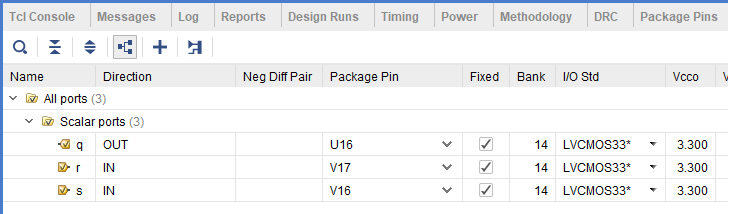
Block Diagram:

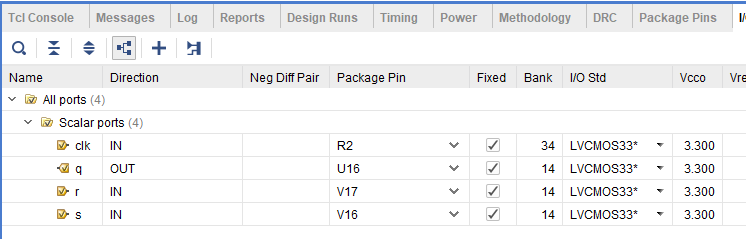




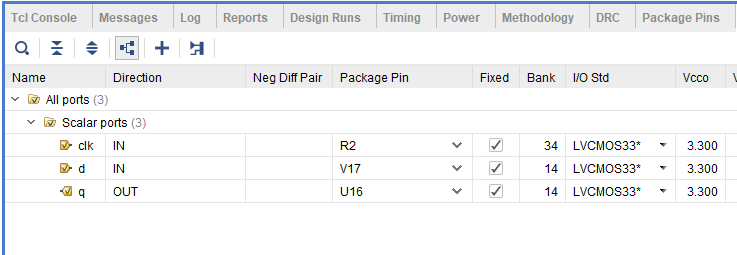
Design Specification Plan:

**Asynchronous flip flop**

****

**Clocked SR flip flop**

**Clocked D flip flop**

****

|  |  |
| --- | --- |
| **Asynchronous flip flop**  asy\_srff.v | srff\_sim.v |
| `timescale 1ns / 1ps  module asy\_srff (  input s,  input r,  output reg q);  always @ (s,r)  begin  if (s==1'b0 && r==1'b0)  q <= q;  else if (s==1'b0 && r==1'b1)  q <= 0;  else if (s==1'b1 && r==1'b0)  q <= 1;  else if (s==1'b1 && r==1'b1)  q <= 1'bx;  end  endmodule | `timescale 1ns / 1ps  module srff\_sim();  reg s\_sim;  reg r\_sim;  wire q\_sim;  // Declare inputs as reg and outputs as wire  asy\_srff UUT (  // Pair inputs/regs and outputs/wires  .s(s\_sim),  .r(r\_sim),  .q(q\_sim)  );  initial begin  s\_sim = 1'b0;  r\_sim = 1'b0;  // Initialize inputs to 0  end  always #10 s\_sim = ~s\_sim;  //invert the set input using this period  always #15 r\_sim = ~r\_sim;  //invert the reset input using this period  endmodule |
|  |  |
| **Clocked SR flip flop**  clk\_srff.v | ssrff\_sim.v |
| `timescale 1ns / 1ps  module clk\_srff(input s,  input r,  input clk,  output reg q  );  always @ (s,r)  begin  if (clk==1'b1)  begin  if (s==1'b0 && r==1'b0)  q <= q;  else if (s==1'b0 && r==1'b1)  q <= 0;  else if (s==1'b1 && r==1'b0)  q <= 1;  else if (s==1'b1 && r==1'b1)  q <= 1'bx;  end  end  endmodule | `timescale 1ns / 1ps  module ssrff\_sim(  );  reg s\_sim;  reg r\_sim;  reg clk\_sim;  wire q\_sim;  // Declare inputs as reg and outputs as wire  clk\_srff UUT (  // Pair inputs/regs and outputs/wires  .s(s\_sim),  .r(r\_sim),  .clk(clk\_sim),  .q(q\_sim)  );  initial begin  s\_sim = 1'b0;  r\_sim = 1'b0;  clk\_sim = 1'b0;  // Initialize inputs to 0  end  always #30 clk\_sim = ~clk\_sim;  always #20 s\_sim = ~s\_sim;  always #40 r\_sim = ~r\_sim;    endmodule |
| **Clocked D flip flop**  clk\_diff.v | dlatch\_sim.v |
| `timescale 1ns / 1ps  module clk\_dff(  input d,  input clk,  output reg q  );  always @ (posedge clk)  if(clk)  begin  q <= d;  end  endmodule | `timescale 1ns / 1ps  module clk\_dff(  input d,  input clk,  output reg q  );  always @ (posedge clk)  if(clk)  begin  q <= d;  end  endmodule |

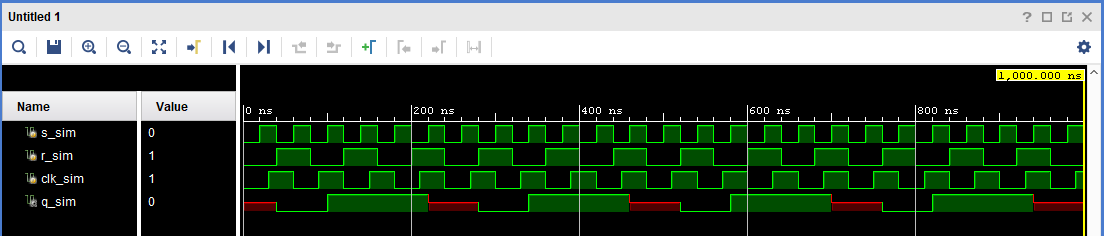
Resulting Wave Forms:

**Asynchronous flip flop**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **R** | **S** | **Qv** | **Qv+1** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 (Not Allowed) | 1 (Not Allowed) | 0 | 1 |
| 1 (Not Allowed) | 1 (Not Allowed) | 1 | 1 |

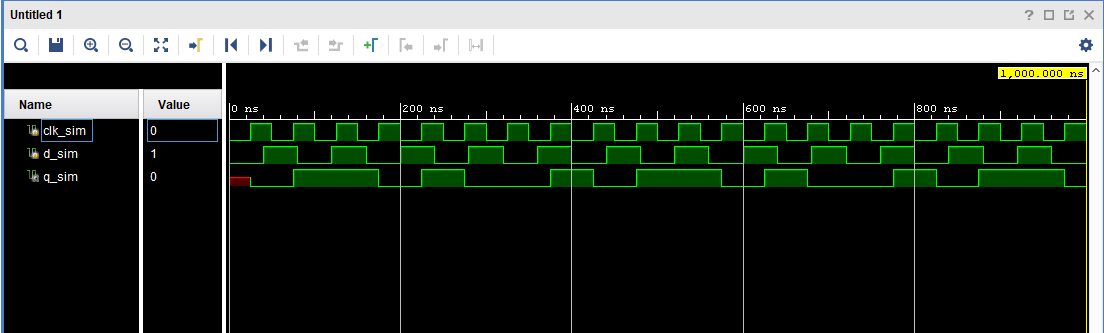
**Clocked SR flip flop**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **R** | **S** | **Qv** | **Qv+1** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 (Not Allowed) | 1 (Not Allowed) | 0 | 1 |
| 1 (Not Allowed) | 1 (Not Allowed) | 1 | 1 |

**Clocked D flip flop**

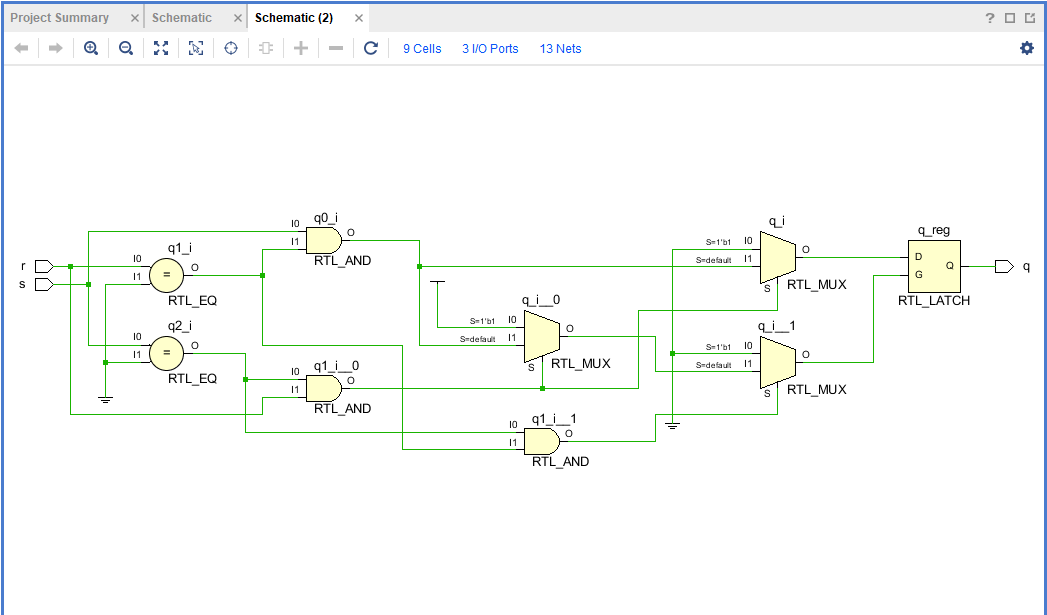
|  |  |  |
| --- | --- | --- |
| **D (Data)** | **Qv** | **Qv+1** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

****

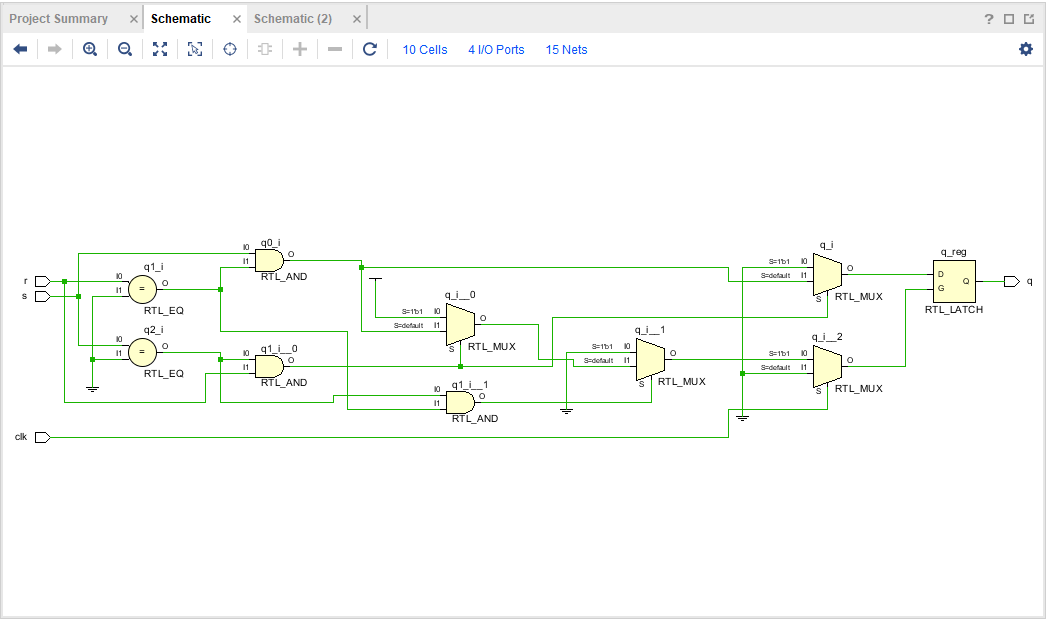
Detailed Schematic Diagram:

wasn’t sure if we needed these schematics or not, added them just in case

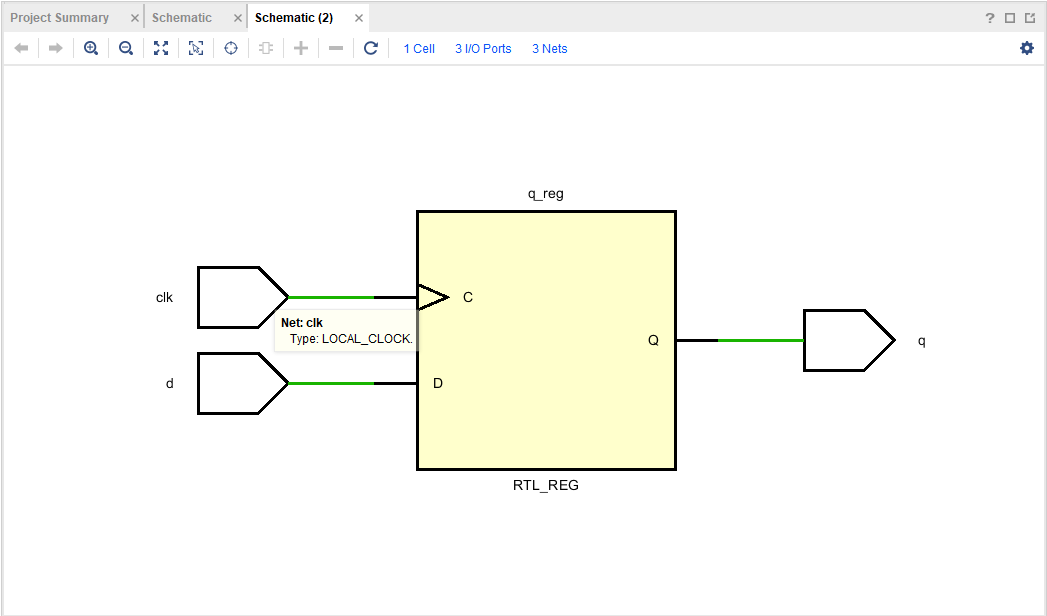
**Asynchronous flip flop**

****

**Clocked SR flip flop**

****

**Clocked D flip flop**

****

Test Plan:

Asynchronous SR flip-flop:

Set pins a code according to above. In Verilog. Since there is no clock, we can set or reset our value but it will not be stored. If we follow the truth table, if S(V16)= 0 R(V17) = 0 there is no change, S = 1 R = 0, set value to 1. S = 0 R = 1, reset q back to 0.

Clocked SR:

Same as above except will only store the value if our clock is post edge. Or clk(R2) = 1.

Otherwise the value of Q will not change.

Clocked D FF:

If our clock is equal to 1, clk(R2) = 1. Our value of q(LED U16) will equal the value of d(v17).

Results:

Results of this experiment are the corresponding Flip Flop which each store a value.

Conclusion

We have created each of the flip-flops that will store the value corresponding to set, reset or d.

Questions:

(To be incorporated within the Conclusion section of your lab report.)

**1. What are the advantages and disadvantages of each flip-flop?**

SR:

Set-Reset is really simple to understand and to implement but, if s=1, r =1 there is no valid choice. As well as no clock

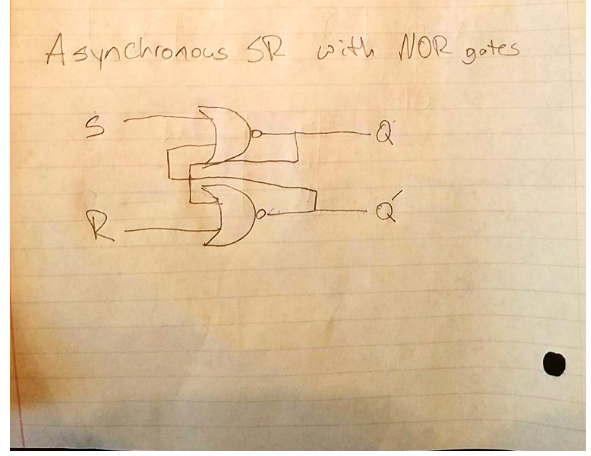
Clocked SR:

Simple, same a set reset but includes a clock and therefore is not “random” though s=1 and r=1 still is not a valid choice.

Clocked D FF

When our enable clock input is 0, the D input has no effect on the output of Q. On post edge the output will equal D.

**2. Draw an asynchronous SR flip-flop schematic using NOR gates.**



**3. Using the lectures textbook, look up level flip-flops versus edge flips-flops and discuss**

**the difference.**

**Level triggering:**

The circuit will become active when the clock pulse in on a particular level decided by the designer. We can have low or, high level triggering (low is negative edge, high is positive edge)

Edge Triggering:

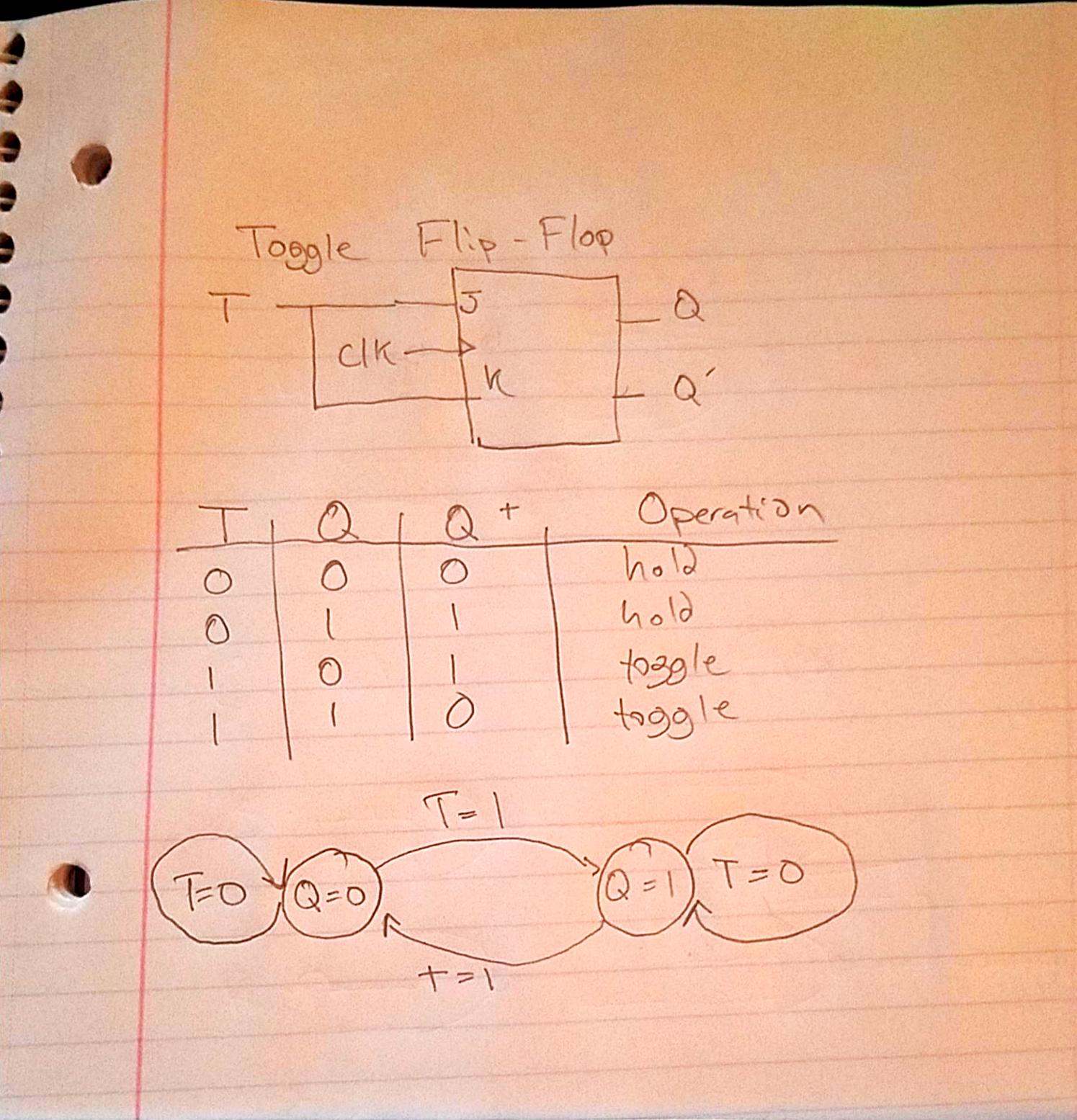
With edge triggering, the circuit becomes active at the positive or negative edge of the clock signal also decided by the designer. It will trigger the second the signal goes from high to low, or vice versa.

**4. What are some applications that might require the use of flip-flops?**

Counting, store bits using shift registers, as well as a frequency measurement.

**5. Another type of edge flip-flop is the Toggle (T) flip-flop. Discuss how this flip-flop**

**works and give its state transition table.**



**6. How can the VERILOG program that implemented the D flip-flop in this experiment**

**be converted to a positive edge or a negative edge D flip-flop?**

Instead of posedge we change it to trigger on the negative edge, or we trigger the signal on both edges of the clock, so we know that when it is 1 or 0.

We can then only run code if the opposite is true.