Lab 1: Introduction to Cadence Schematic Capture & Simulation

Alex Anderson

UIN: 728001757

ECEN 714-603

To begin, the full adder was designed using NAND and XOR gates. The schematic and symbol created are below.

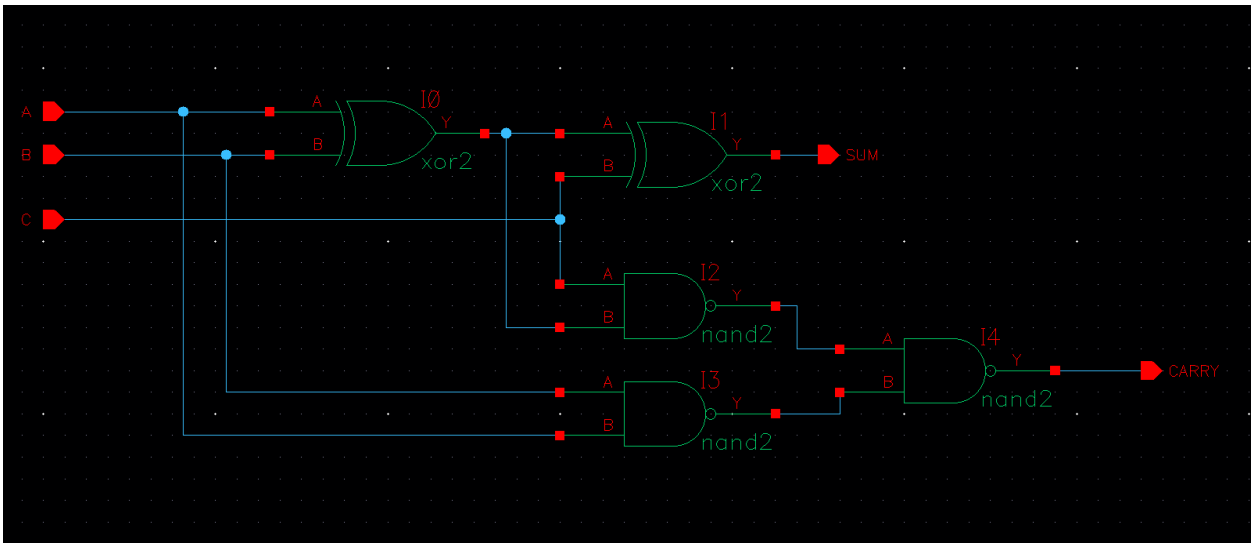

Figure 1: Full Adder Schematic

Figure 2: Full Adder Symbol

Next, the design was checked using a Verilog test bench. The results were verified as correct for all possible input combinations.

```
// Verilog stimulus file.
// Please do not create a module in this file.


// Default verilog stimulus.

initial
begin
        A = 1'b0; B = 1'b0; C = 1'b0; // ABC = 000
        #50 C = 1'b1; // ABC = 001
        #50 B = 1'b1; C = 1'b0; // ABC = 010
        #50 C = 1'b1; // ABC = 011
        #50 A = 1'b1; B = 1'b0; C = 1'b0; // ABC = 100
        #50 C = 1'b1; // ABC = 101
        #50 C = 1'b0; B = 1'b1; // ABC = 110
        #50 C = 1'b1; // ABC = 111
end

initial
$monitor ($time," A=%b, B=%b, C=%b, SUM=%b, CARRY=%b", A, B, C, SUM, CARRY);
```

Figure 3: Full Adder Test Bench

```
-------------------------------------
Relinquished control to SimVision ...
ncsim>
ncsim> source /opt/coe/cadence/INCISIVE152/tools/inca/files/ncs
ncsim> database -open shmWave -shm -default -into shm.db
Created default SHM database shmWave
ncsim> probe -create -shm test  -all   -depth 1
Created probe 1
ncsim> run
                   0 A=0, B=0, C=0, SUM=0, CARRY=0
                  50 A=0, B=0, C=1, SUM=1, CARRY=0
                 100 A=0, B=1, C=0, SUM=1, CARRY=0
                 150 A=0, B=1, C=1, SUM=0, CARRY=1
                 200 A=1, B=0, C=0, SUM=1, CARRY=0
                 250 A=1, B=0, C=1, SUM=0, CARRY=1
                 300 A=1, B=1, C=0, SUM=0, CARRY=1
                 350 A=1, B=1, C=1, SUM=1, CARRY=1
ncsim>
[alexd43anderson]@n01-zeus ~/ecen714/fulladder_run1> (12:19:55
```

Figure 4: Full Adder Verilog Output

Next, the 4-bit adder was designed by cascading 4 of the full adders. The schematic and symbol are included below.

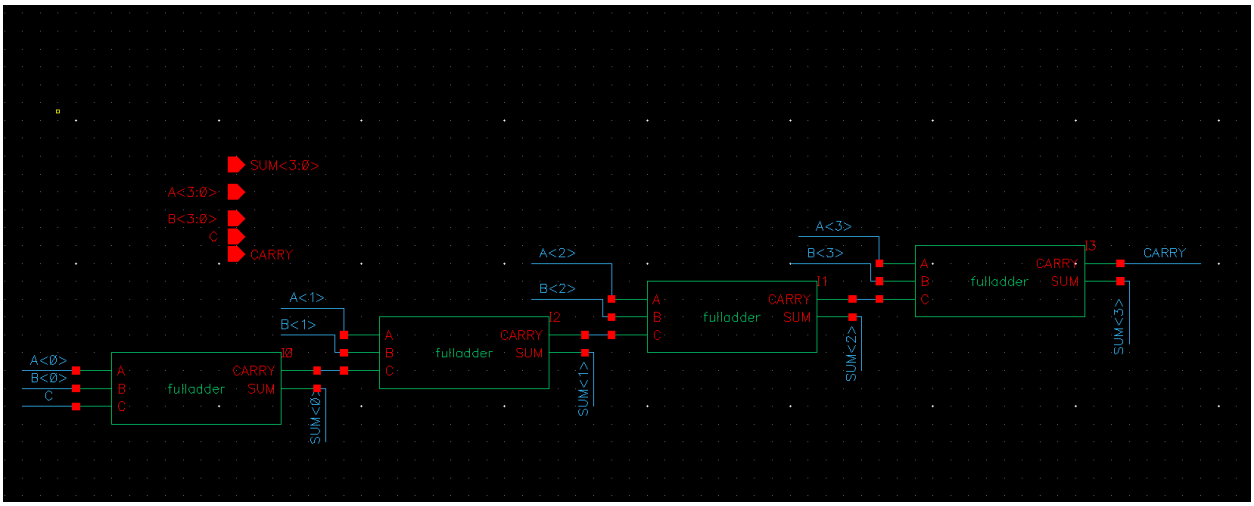

Figure 5: 4-bit Adder Schematic

Figure 6: 4-bit Adder Symbol

Again, the results were verified using a Verilog test bench for the required input cases.

```
// Verilog stimulus file.
// Please do not create a module in this file.


// Default verilog stimulus.

initial
begin

        #50 A[3:0] = 4'b0000; B[3:0] = 4'b0000; C = 1'b0;
        #50 A[3:0] = 4'b1111; B[3:0] = 4'b1111; C = 1'b0;
        #50 A[3:0] = 4'b1010; B[3:0] = 4'b1010; C = 1'b1;
        #50 A[3:0] = 4'b0101; B[3:0] = 4'b0101; C = 1'b1;
end

initial
$monitor ($time," A=%b, B=%b, C=%b, SUM=%b, CARRY=%b", A, B, C, SUM, CARRY);
```

Figure 7: 4-Bit Adder Test Bench

```
ncsim> cat simout.tmp
ncsim> run
                   0 A=xxxx, B=xxxx, C=x, SUM=xxxx, CARRY=x
                  50 A=0000, B=0000, C=0, SUM=0000, CARRY=0
                 100 A=1111, B=1111, C=0, SUM=1110, CARRY=1
                 150 A=1010, B=1010, C=1, SUM=0101, CARRY=1
                 200 A=0101, B=0101, C=1, SUM=1011, CARRY=0
ncsim>
[alexd43anderson]@n01-zeus ~/ecen714/4bitadder_run1> (12:11:0
```

Figure 8: 4-bit Adder Verilog Output

Finally, in order to create the 8-bit pipelined adder, a 4-bit register was created using D-flip-flops and verified.

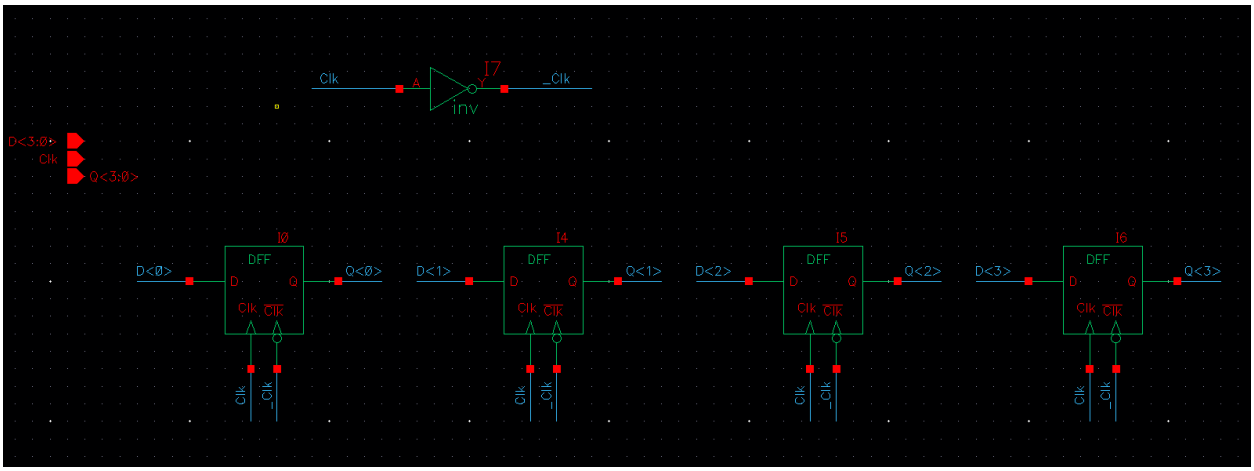

Figure 9: 4-bit Register Schematic

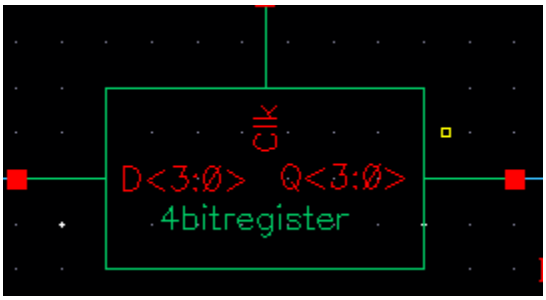

Figure 10: 4-bit Register Symbol

With all the sub-blocks complete, the final 8-bit pipelined adder was created. The design was verified using a Verilog test bench for the desired input cases.

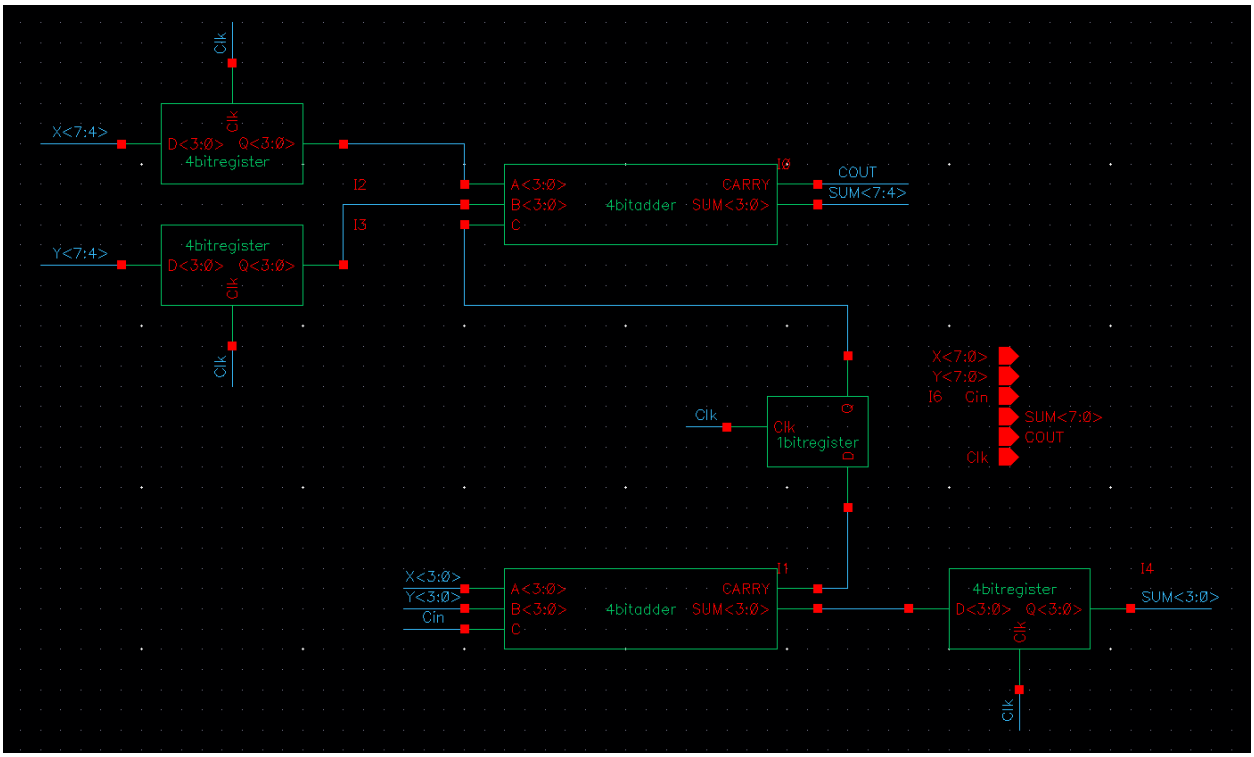

Figure 11: 8-bit Adder Schematic

```
// Verilog stimulus file.
// Please do not create a module in this file.


// Default verilog stimulus.

initial
begin
        Cin = 1'b0; X[7:0] = 8'b00000000; Y[7:0] = 8'b00000000;
        #40 Cin = 1'b0; X[7:0] = 8'b01111110; Y[7:0] = 8'b11100111;
        #40 Cin = 1'b1; X[7:0] = 8'b11111111; Y[7:0] = 8'b00000000;
        #40 Cin = 1'b0; X[7:0] = 8'b10101010; Y[7:0] = 8'b01010101;
        #40 Cin = 1'b1; X[7:0] = 8'b10101010; Y[7:0] = 8'b01010101;
        #40 Cin = 1'b0; X[7:0] = 8'b11001100; Y[7:0] = 8'b00110011;
        #40 Cin = 1'b1; X[7:0] = 8'b11001100; Y[7:0] = 8'b00110011;
end

initial
begin
        Clk = 1'b1;
        repeat (40)
                #10 Clk = ~Clk;
end


initial
$monitor ($time," X=%b, Y=%b, Cin=%b, SUM=%b, COUT=%b", X, Y, Cin, SUM, COUT);
```

Figure 12: 8-bit Adder Test Bench

In the test bench, some logic was added to control the clock signal. Since the minimum number of clock cycles to generate an output is 2, some margin was given to the delays. Specifically, a clock period of 20ns (10ns high / 10ns low) was used. Delays of 40ns were used in order to allow for 2 clock cycles so the output has correctly propagated through the registers.

```
Created probe 1
ncsim> run
                   0 X=00000000, Y=00000000, Cin=0, SUM=xxxxxxxx, COUT=x
                  20 X=00000000, Y=00000000, Cin=0, SUM=00000000, COUT=0
                  40 X=01111110, Y=11100111, Cin=0, SUM=01010000, COUT=1
                  60 X=01111110, Y=11100111, Cin=0, SUM=01100101, COUT=1
                  80 X=11111111, Y=00000000, Cin=1, SUM=00000101, COUT=1
                 100 X=11111111, Y=00000000, Cin=1, SUM=00000000, COUT=1
                 120 X=10101010, Y=01010101, Cin=0, SUM=00000000, COUT=1
                 140 X=10101010, Y=01010101, Cin=0, SUM=11111111, COUT=0
                 160 X=10101010, Y=01010101, Cin=1, SUM=11111111, COUT=0
                 180 X=10101010, Y=01010101, Cin=1, SUM=00000000, COUT=1
                 200 X=11001100, Y=00110011, Cin=0, SUM=00000000, COUT=1
                 220 X=11001100, Y=00110011, Cin=0, SUM=11111111, COUT=0
                 240 X=11001100, Y=00110011, Cin=1, SUM=11111111, COUT=0
                 260 X=11001100, Y=00110011, Cin=1, SUM=00000000, COUT=1
ncsim> ^C
```

Figure 13: 8-bit Adder Verilog Output

In the results above, it is important to consider only the second entry for each set of inputs. The first entry represents only 1 clock cycle, meaning the carry from the 4 LSB has not propagated to the top adder. Considering these second entries, the results are correct for each case.