

Lab Report 1: Introduction to Cadence Schematic Design & Simulation

ECEN 454/714

1/29/2024

TA: Saichand Samudrala

Ryan Peng
631003156

Full adder:

To start, we first must design and simulate a full adder properly to use it moving forward in the lab. To do this, I followed lab manual zero to set up cadence properly then proceeded to start my design.

Following the appendix in lab manual one, I was able to create the following schematic for a full adder circuit using XOR2 and NAND2 gates:

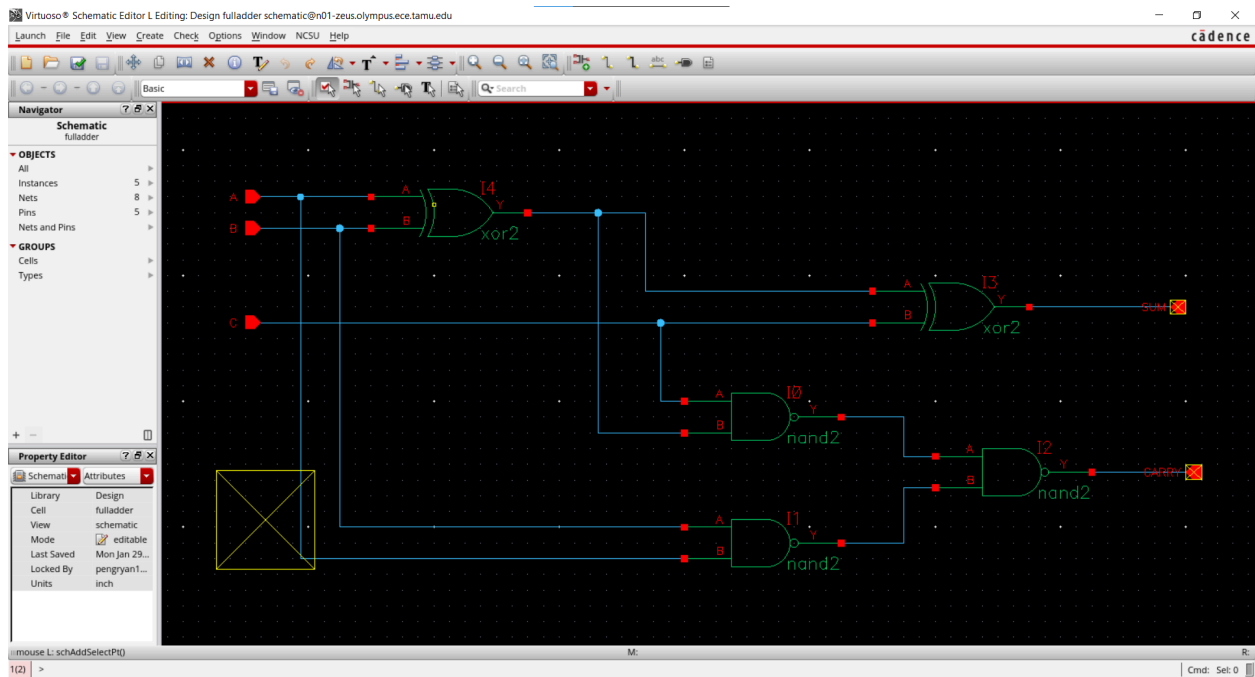


Figure 1: Schematic of a full adder using NAND and XOR gates.

In order to use this schematic in a convenient package, we created a symbol for the full adder as shown below:

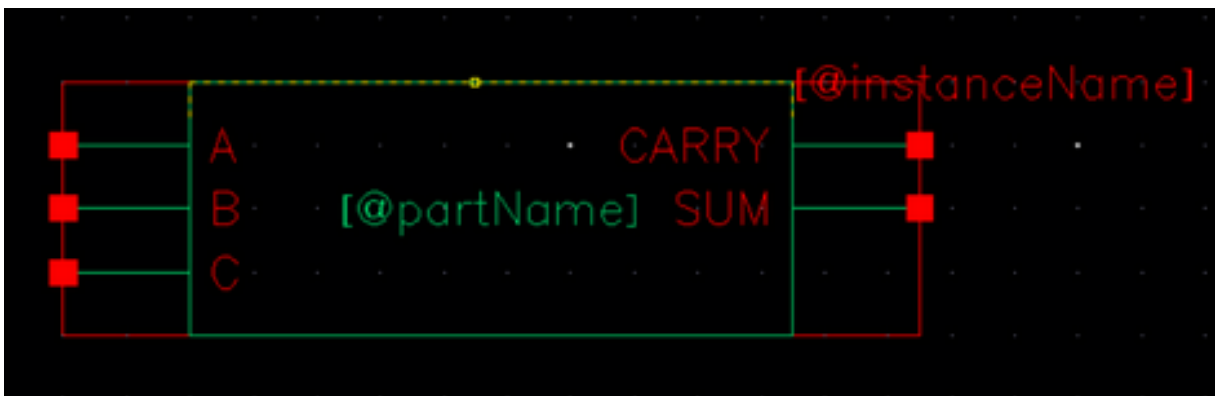
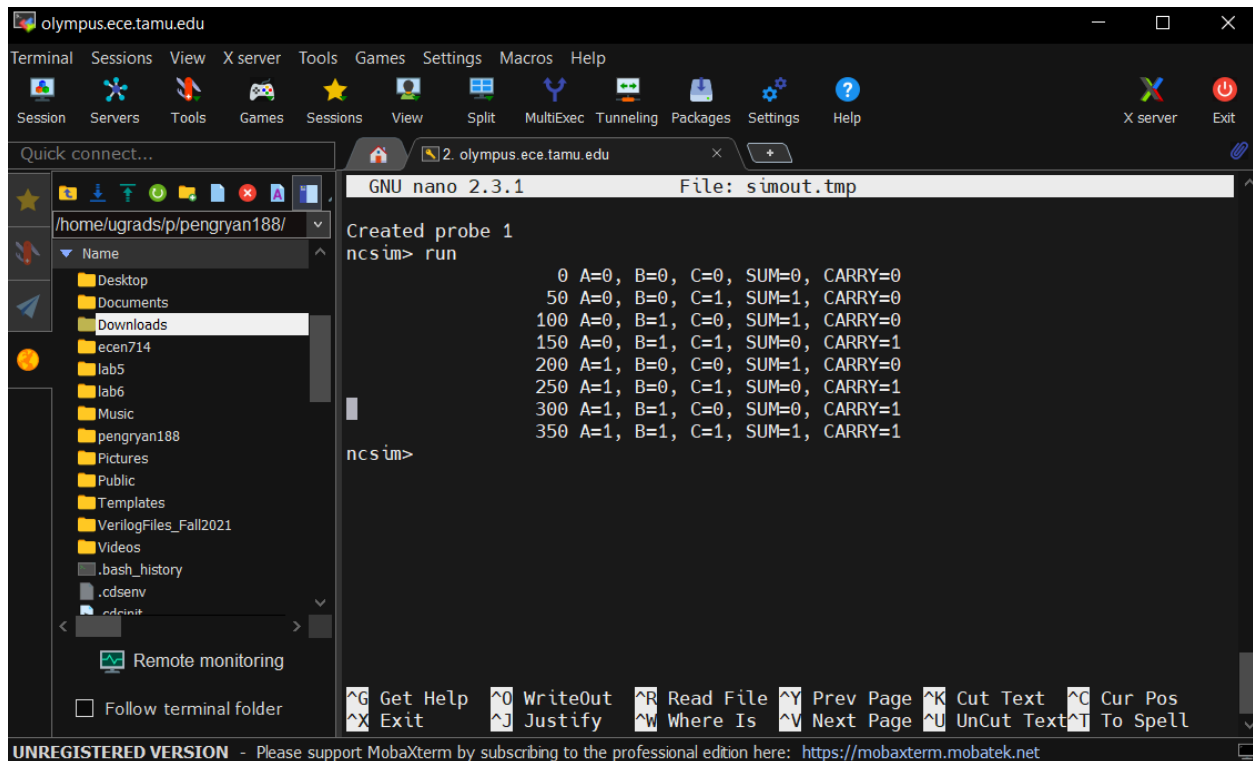


Figure 2: Symbol generated for the full adder circuit.

Then, we were to check if the results for the full adder circuit are assembled correctly and behave as expected. I followed the steps in the manual to edit the test fixture file in order to display the results after the simulation passed. Once that was completed, the simulation results are as shown:



```
olympus.ece.tamu.edu
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
/home/ugrads/p/pengryan188/
Name
Desktop
Documents
Downloads
ecen714
lab5
lab6
Music
pengryan188
Pictures
Public
Templates
VerilogFiles_Fall2021
Videos
.bash_history
.cdserve
.cdrinit
Remote monitoring
Follow terminal folder
GNU nano 2.3.1 File: simout.tmp
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>
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```

Figure 3: Simulation results for the full adder circuit

4-bit adder:

Then, we are to use the full adder symbol to create a 4-bit adder circuit. The way we are to assemble it was explained in the manual, we were to connect four full adder circuits and use a 4 bit input, A and B, and a 1 bit input Cin. The output would be a 4 bit SUM value, and a 1 bit output Cout. Then, the carry values from each adder up until the last one will be used as in input for C.

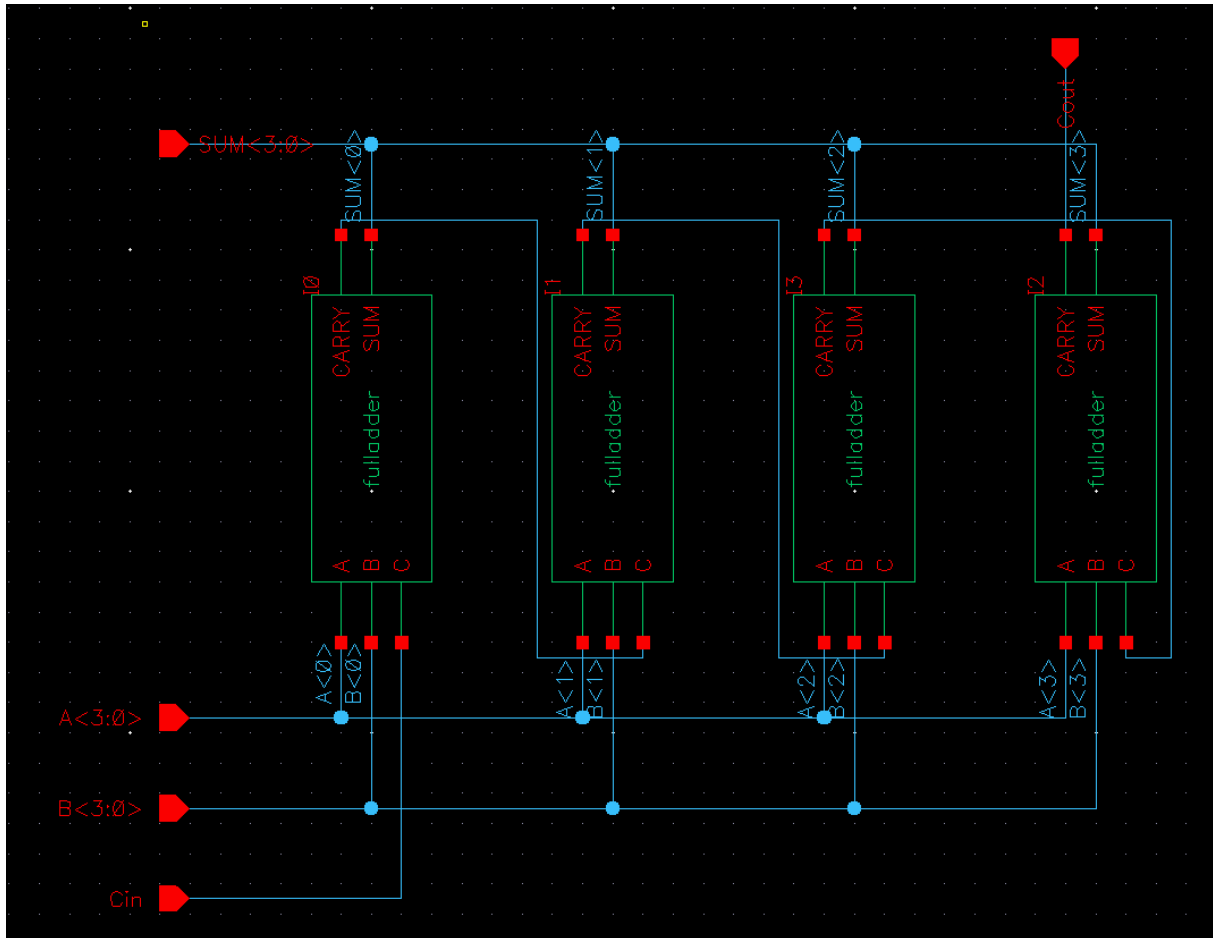


Figure 4: Schematic of a 4-bit adder using 4 full adder circuits.

Then, we are to export it as a symbol again to be used for reference:

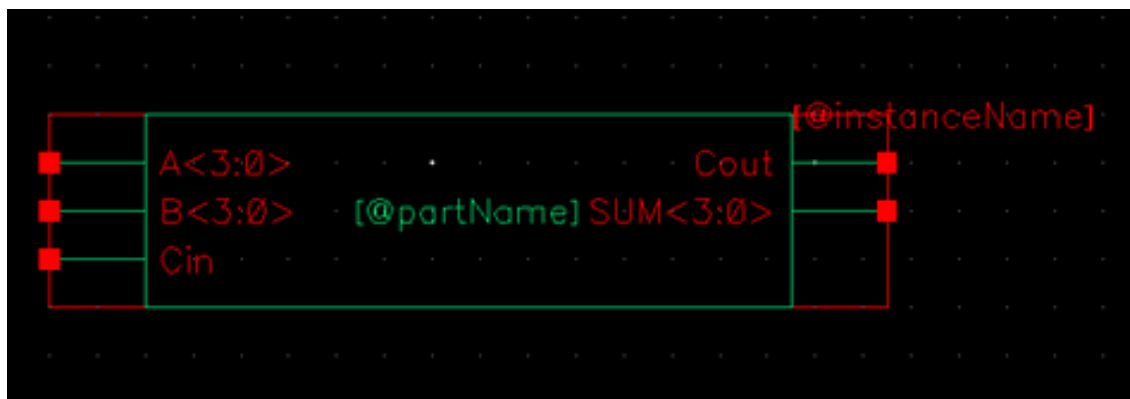
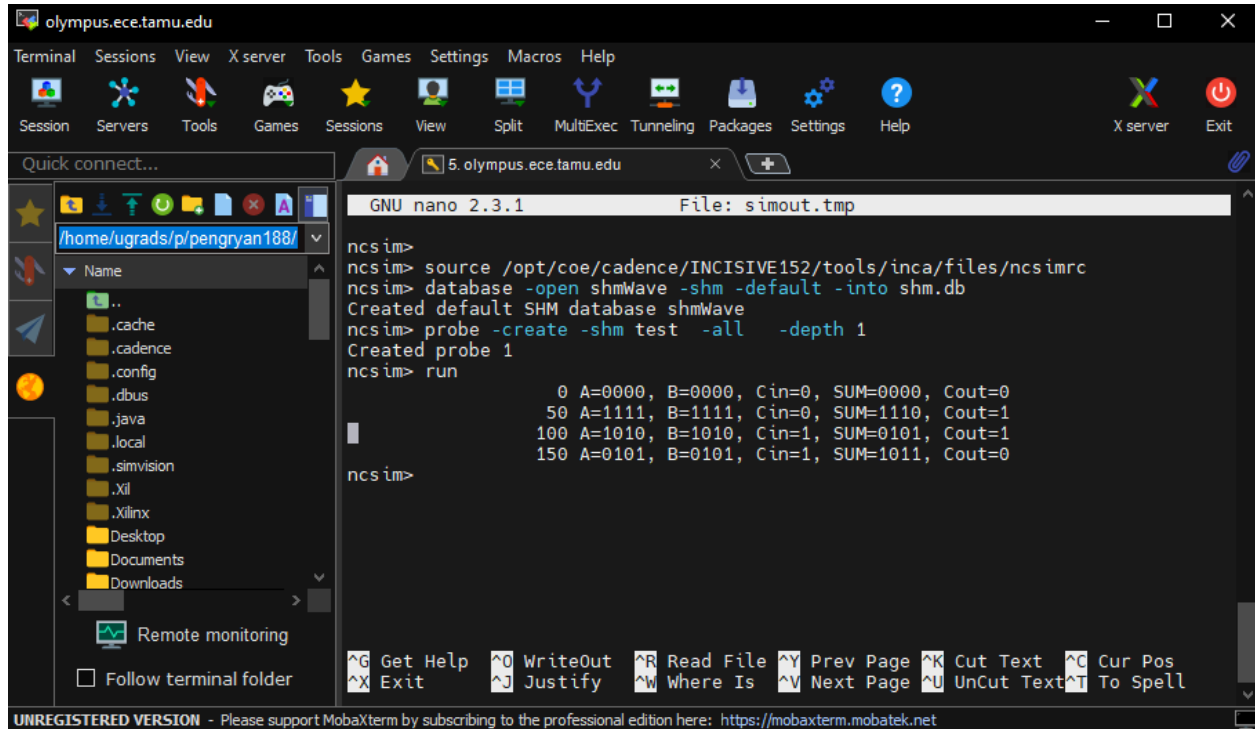


Figure 5: Symbol for a 4-bit adder

Then, to make sure we simulated and created the circuit correctly, we were asked to go into the test fixture and change the parameters to accommodate for the 4 bit inputs, and the test cases we were asked to check. Once those were changed, the simulation results were as shown:



```

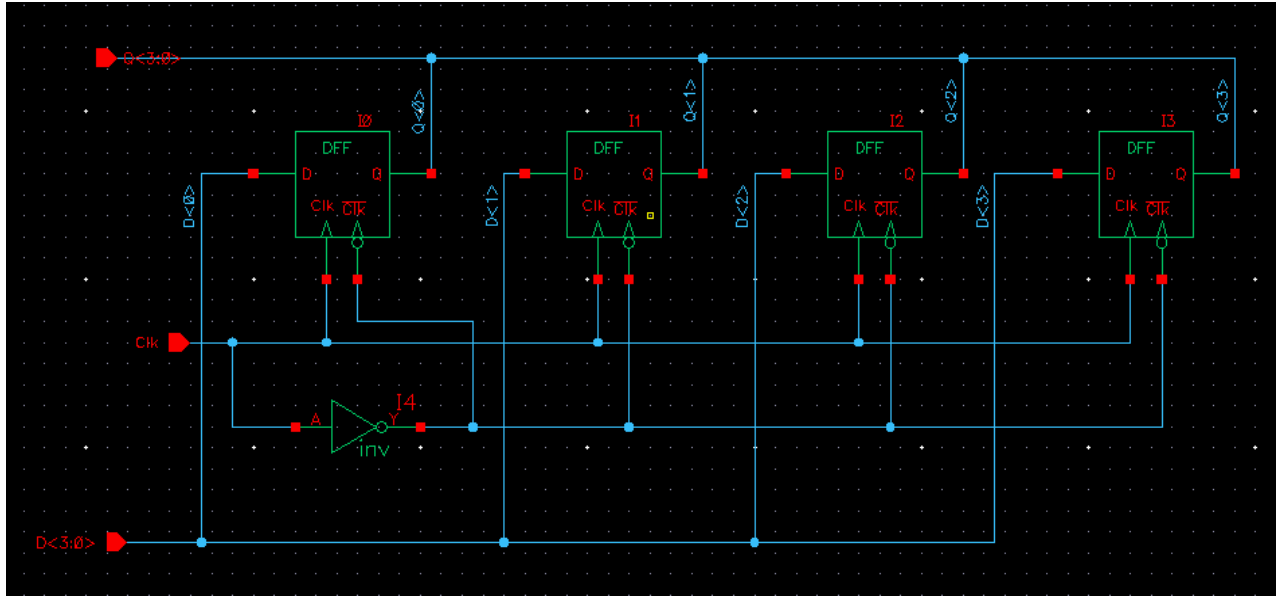
olympus.ece.tamu.edu
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect... 5. olympus.ece.tamu.edu
GNU nano 2.3.1 File: simout.tmp
ncsim>
ncsim> source /opt/coe/cadence/INCISIVE152/tools/inca/files/ncsimrc
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=0000, B=0000, Cin=0, SUM=0000, Cout=0
     50 A=1111, B=1111, Cin=0, SUM=1110, Cout=1
    100 A=1010, B=1010, Cin=1, SUM=0101, Cout=1
    150 A=0101, B=0101, Cin=1, SUM=1011, Cout=0
ncsim>
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

```

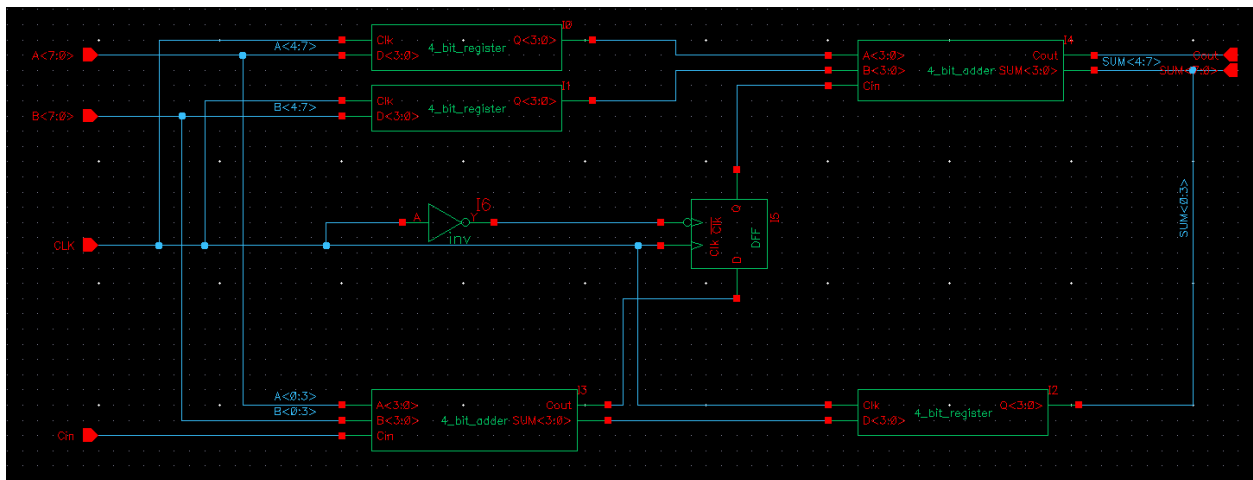
Figure 6: Simulation results for a sample of test cases

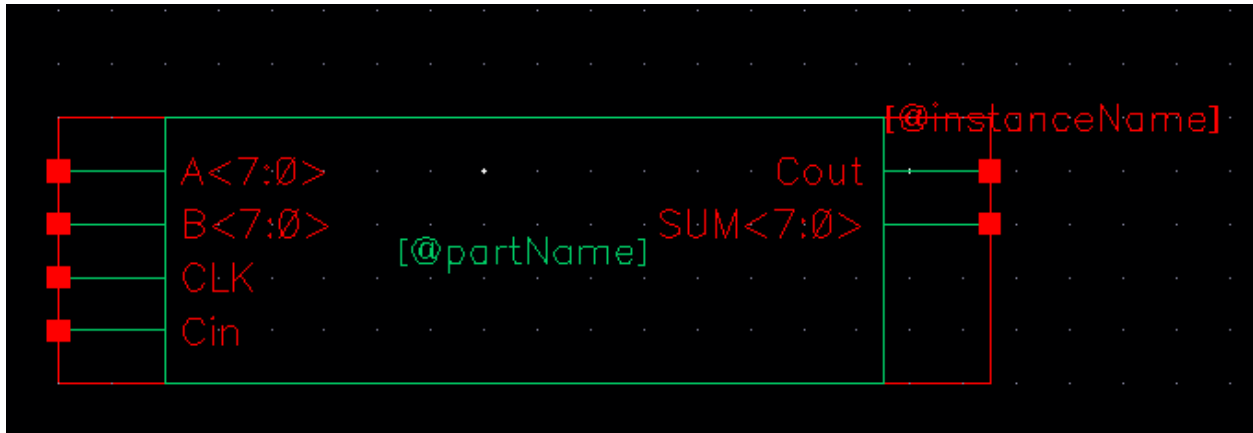
8-bit adder:

Prior to designing the 8-bit adder, we are to first create a 4-bit register using flip flops. I created this 4-bit register using 4 flip-flops, where we provide its inputs from 4 pins and tap its outputs through 4-other pins (such as D<3:0> and Q<3:0> respectively) while supplying each flip flop with a clock and inverse clock (clock). The schematic and symbol of the 4-bit register are shown below:



Using this flip-flop, we can finally follow lab manual 0 to assemble the 8-bit pipelined adder as shown below:





Then to ensure that our results are correct, I edited the testfixture.verilog and simulated the results as shown below:

```

ncsim> run
      0 A=00000000, B=00000000, Cin=0, SUM=xxxxxxx, Cout=x, CLK=0
     50 A=01111110, B=11100111, Cin=0, SUM=00000000, Cout=0, CLK=1
    100 A=11111111, B=00000000, Cin=1, SUM=00000000, Cout=0, CLK=0
    150 A=11111111, B=00000000, Cin=1, SUM=00000000, Cout=1, CLK=1
    200 A=10101010, B=01010101, Cin=0, SUM=00000000, Cout=1, CLK=0
    250 A=10101010, B=01010101, Cin=0, SUM=11111111, Cout=0, CLK=1
    300 A=10101010, B=01010101, Cin=1, SUM=11111111, Cout=0, CLK=0
    350 A=10101010, B=01010101, Cin=1, SUM=00000000, Cout=1, CLK=1
    400 A=11001100, B=00110011, Cin=0, SUM=00000000, Cout=1, CLK=0
    450 A=11001100, B=00110011, Cin=0, SUM=11111111, Cout=0, CLK=1
    500 A=11001100, B=00110011, Cin=1, SUM=11111111, Cout=0, CLK=0
    550 A=11001100, B=00110011, Cin=1, SUM=00000000, Cout=1, CLK=1
ncsim>

```

These results are accurate, because when we go from time 0 to time 50, CLK is at zero, so the sum would hold 00000000. Then from line 50 to 100, because the rising clock coincides with A = 11111111 and B = 00000000, while Cin = 1, the SUM would equal 00000000 with a Cout value of 1. From line 100 to 150 it's a similar idea, the CLK is zero so the sum doesn't change. Then at time 200, because the CLK is zero, the output holds 00000000. At 250, when A = 10101010 and B = 01010101 and Cin = 0, the sum would equal to 11111111 which checks out. Then at line 300, because the CLK = 0, the output holds 11111111. At line 350, because CLK is 1, it would hold the new sum, that being 00000000 because adding a Cin of 1 would make 11111111 equal to zero and have a Cout of 1. At time 400, it holds SUM as 00000000 and Cout as 1 because CLK is 0. Then at time 450, the SUM would equal to 11111111 because 11001100 plus 00110011 with a Cin of 0 equals 11111111. Then again, at time 500, normally the output would be 00000000 with a Cout of 1, but because CLK is zero, SUM stays as 11111111 with a Cout of 0. Then at line 550, the sum between 11001100 and 00110011 with a Cin of 1 would be 00000000 with a Cout of 1 which checks out with our results.