SY303 Lab 2 **Boolean Arithmetic**

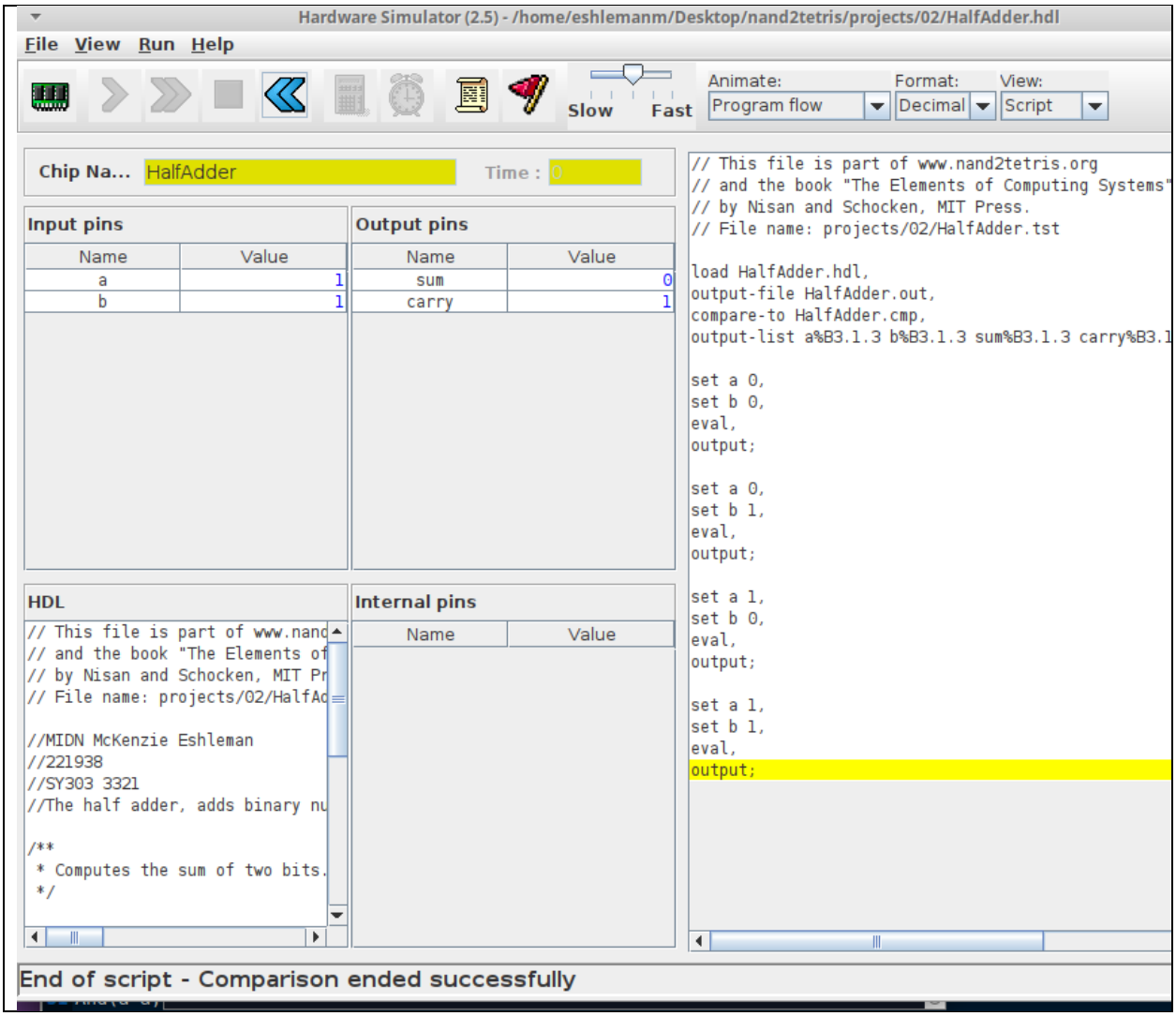
Name: McKenzie Eshleman

Partner(s): Cyrus Malek-Madani

Section: 3321

Objective: In this lab, you will build on experience gained with the last project and build components capable of performing addition. These parts will be built using HDL, documentation using Gate Diagrams and Truth Tables will be required.

1) **Pre-Lab: Half Adder**
    a. Screenshot of your Half-Adder Successfully Completed test in the Hardware Simulator. This chip has already been demonstrated for you. (10 pts)


Hardware Simulator (2.5) - /home/eshlemanm/Desktop/nand2tetris/projects/02/HalfAdder.hdl
File View Run Help
Slow Fast
Animate: Program flow
Format: Decimal
View: Script
Chip Na... HalfAdder
Time : 0
Input pins
Name Value
a 1
b 1
Output pins
Name Value
sum 0
carry 1
HDL
// This file is part of www.nand
// and the book "The Elements of
// by Nisan and Schocken, MIT Pr
// File name: projects/02/HalfAd
//MIDN McKenzie Eshleman
//221938
//SY303 3321
//The half adder, adds binary nu
/**
* Computes the sum of two bits.
*/
Internal pins
Name Value
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/HalfAdder.tst
load HalfAdder.hdl,
output-file HalfAdder.out,
compare-to HalfAdder.cmp,
output-list a%B3.1.3 b%B3.1.3 sum%B3.1.3 carry%B3.1
set a 0,
set b 0,
eval,
output;
set a 0,
set b 1,
eval,
output;
set a 1,
set b 0,
eval,
output;
set a 1,
set b 1,
eval,
output;
End of script - Comparison ended successfully

2) Part: Full-adder

Purpose: The Full-adder adds three total bits. The two that are inputs for the current place value, and a third input which is the carry from the previous place value.

Truth Table: Fill in the outputs that describe the **sum** and **carry** for a 3 input 1-bit addition problem**.** (10 pts)

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

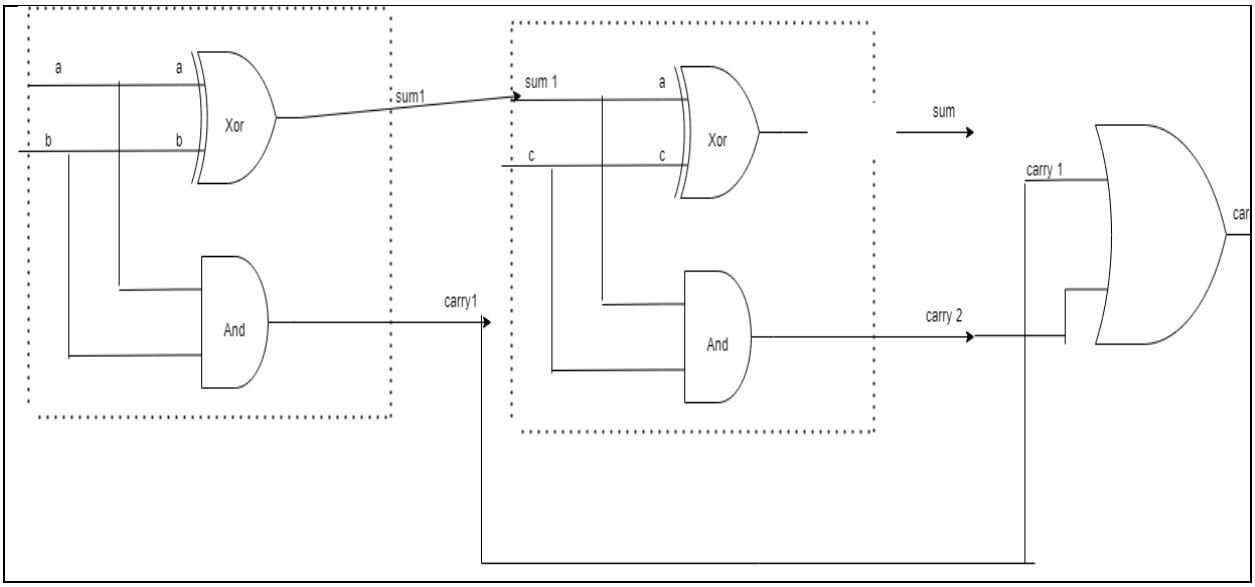
Canonical Form: (10 pts)

`sum = (A!*B!)*c + (A!*C!)*B + (B!*C!)*A +(A*B)*C`

`carry = (A*B)*!c + (A*C)* !B + (B*C)*!A +(A*B)*C`

Gate Diagram (Insert Drawing or Use Word Art): (10 pts)

Recall that combining parts you have already created will benefit you greatly. You can implement the full Canonical Expressions above, OR you can cleverly use the already created **`Half-Adder`**.



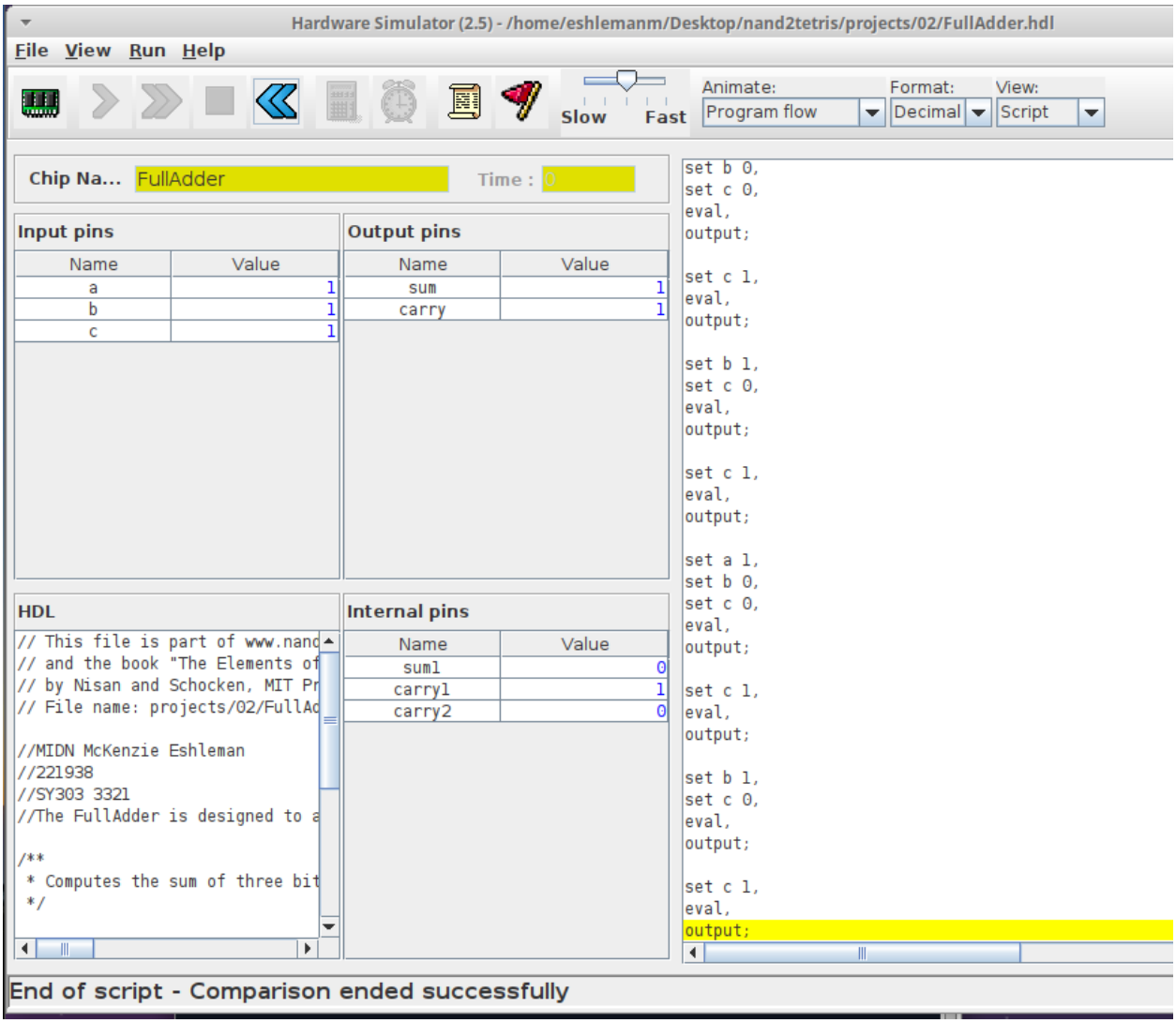
HDL (Screenshot of VM or Raw Text):

Enter your HDL below that describes your gate diagram above. Remember, there will be ONE line of HDL code for each gate in your diagram! (10 pts)

```
// Name: MIDN McKenzie Eshleman
//  Alpha: 221938
// Section: 3321
// Description :  The FullAdder is designed to add  3 bits together, should produce two outputs.

PARTS:
        // YOUR CODE BELOW
// a + b + c == (a + b) + c
HalfAdder(a=a, b=b, sum=sum1, carry=carry1);
HalfAdder(a=sum1, b=c, sum=sum, carry=carry2);
//If carry is true, it gets passed on
Or(a=carry1, b=carry2, out= carry)
```

Test in HW Simulator (Screenshot of VM): (10 pts)

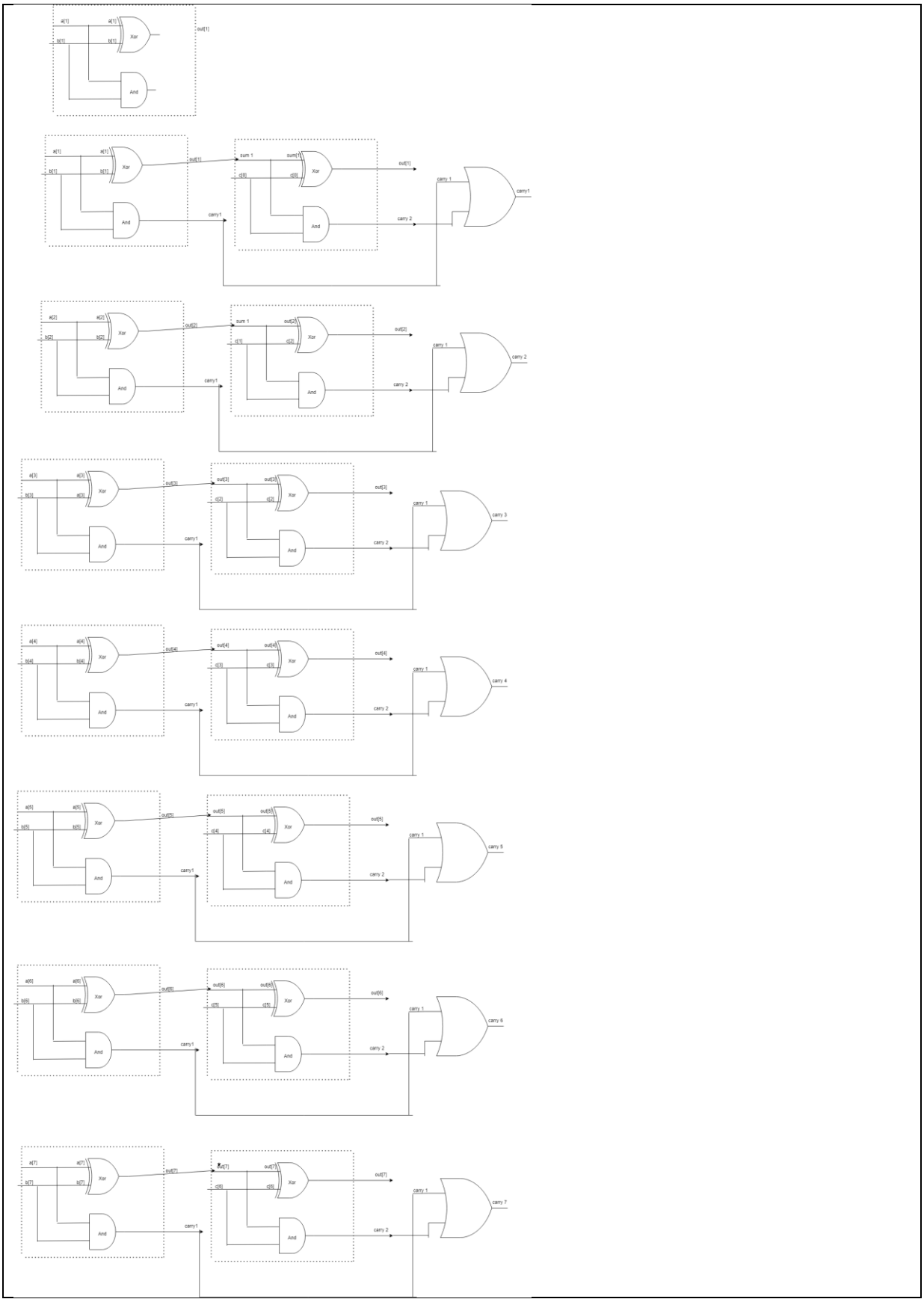


Additional Screenshots (as needed):

3) **Part: 16-bit Adder**

Purpose: < The Add16 Chip represents integer numbers by 16-bit patterns, it adds to a 16-bit adder. This allows you to add up to 16 bit numbers.> (5 pts)

Gate Diagram (Insert Drawing or Use Word Art): You should go straight into conceptualizing your gate diagram now. How can the parts you've already implemented help you to do the same thing with more bits? **HINT**: Try starting with a 2-bit Adder, then 3-bit, and try to recognize the pattern. (10 pts)

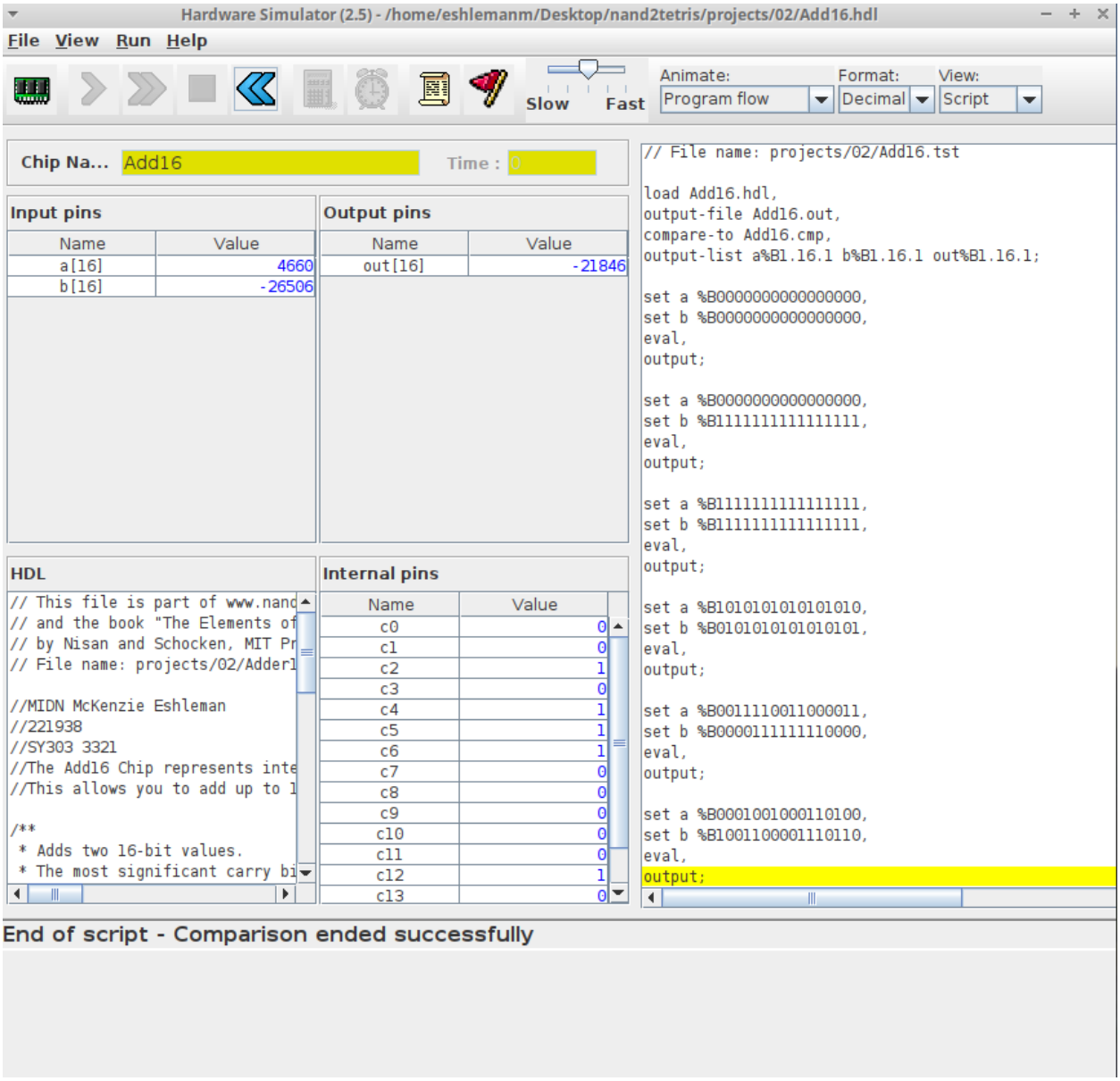

a[1]
b[1]
Xor
And
out[1]
sum 1
c[0]
carry 1
carry 2
carry1
a[2]
b[2]
c[1]
a[3]
b[3]
c[2]
a[4]
b[4]
c[3]
a[5]
b[5]
c[4]
a[6]
b[6]
c[5]
a[7]
b[7]
c[6]

HDL (Screenshot of VM or Raw Text): Enter your HDL below that describes your gate diagram above. (10 pts)

```
// Name: MIDN McKenzie Eshleman
//  Alpha: 221938
// Section: 3321
// Description :  The Add16 Chip represents integer numbers by 16-bit patterns, it adds to a 16-bit adder. This allows you to add up to 16 bit numbers

PARTS:
        // YOUR CODE BELOW
//starting with 0 as the inputs with a half adder
  HalfAdder(a=a[0], b=b[0], sum=out[0], carry=c0);
//Carrying out the rest of the way to 14 bits with the full adder chip
  FullAdder(a=a[1],  b=b[1],  c=c0,  sum=out[1],  carry=c1);
  FullAdder(a=a[2],  b=b[2],  c=c1,  sum=out[2],  carry=c2);
  FullAdder(a=a[3],  b=b[3],  c=c2,  sum=out[3],  carry=c3);
  FullAdder(a=a[4],  b=b[4],  c=c3,  sum=out[4],  carry=c4);
  FullAdder(a=a[5],  b=b[5],  c=c4,  sum=out[5],  carry=c5);
  FullAdder(a=a[6],  b=b[6],  c=c5,  sum=out[6],  carry=c6);
  FullAdder(a=a[7],  b=b[7],  c=c6,  sum=out[7],  carry=c7);
  FullAdder(a=a[8],  b=b[8],  c=c7,  sum=out[8],  carry=c8);
  FullAdder(a=a[9],  b=b[9],  c=c8,  sum=out[9],  carry=c9);
  FullAdder(a=a[10], b=b[10], c=c9,  sum=out[10], carry=c10);
  FullAdder(a=a[11], b=b[11], c=c10, sum=out[11], carry=c11);
  FullAdder(a=a[12], b=b[12], c=c11, sum=out[12], carry=c12);
  FullAdder(a=a[13], b=b[13], c=c12, sum=out[13], carry=c13);
  FullAdder(a=a[14], b=b[14], c=c13, sum=out[14], carry=c14);
//with the last bit we drop the last carry
  FullAdder(a=a[15], b=b[15], c=c14, sum=out[15], carry=c15);
```

Test in HW Simulator (Screenshot of VM): (5 pts)



Additional Screenshots (as needed):

4) **Discuss the ability of computers to represent data using only binary. How many classes of data can you think of which can be represented by an 8-bit value?** (4 pts)

Computers use binary to represent their coding system, the computer switches to represent data and switches to have only to states of Off and On. 256 classes of data are needed to represent an 8-bit value.

5) **Discuss the efficiency of the Adder you implemented. Would we want to use your implementation in a real-world high performance machine? You may use the Internet to research the carry look-ahead adder, compare and contrast it to what you created for this course. Be sure to cite your resource!** (6 pts)
The look-ahead-adder reduces the delay by allowing more complex hardware to take place. The ripple carry design is transformed such that the logic over fixed groups of bits reduces the adder to two level logic.