

Lab 2: Verilog

CpE64 Section 1

Instructor: Telles, Eric

Friday: 4:30 pm – 6:50pm

Thao, Shammah

Part 1A: IDE Tutorial

Description: Using the tutorial of the FPGA pdf that was given, try to make the led blink on the breadboard using the FPGA

Procedures: Complete the tutorial (Altera-Quartus) for the Integrated Design Environment (IDE) you will be using. For demo follow the Tutorial: Altera Quartus and the Terasic DE0 Nano Board and make a led blink on the breadboard.

Engineering Data:

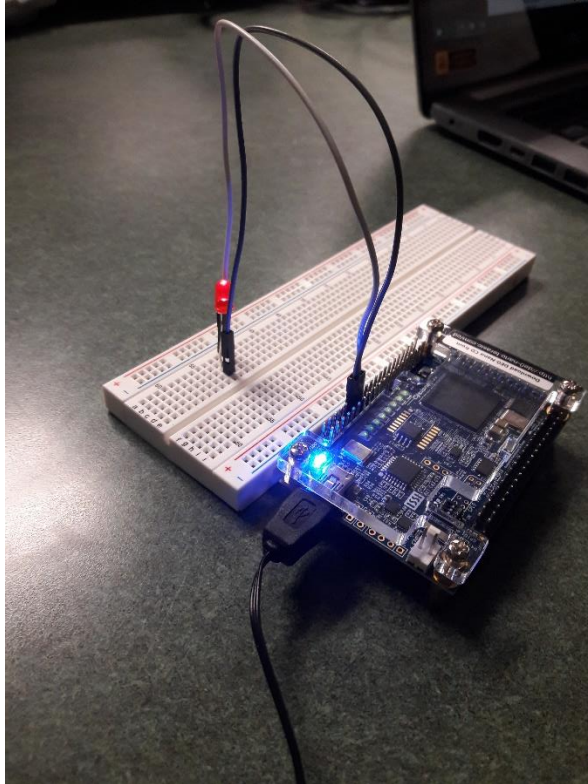


Figure 1

in figure one, we basically follow the instruction given in the tutorial until the in where the FPGA is blinking at a constant rate of one tick per second, as changing the pin arrangement, we were able to get it to light up.

Part 1B: Gates in Verilog

Description: We have to create another Verilog hardware that represent 7408, 74LS00, 74LS04, and 74LS32 gates

Procedures: Create a Verilog hardware description for the 7408 AND gate. When this is written, do a SYNTAX check by compiling this file. IF you get errors, go back and fix them. Usually it's best to fix the first error and re-compile. When you get a successful compilation, you then assign pins to the two inputs and one output. Inputs of the gates should be assigned a switch, and the output should be assigned an LED. Deciding which switch and LED is uHp to you! Once the pin assignment is done, you move on to

downloading your design to the FPGA. (Again, see the tutorial in this lab manual) If that is done successfully, you can now TEST your circuit against the truth table you have prepared for lab. Write down your OBSERVED results next to your PREDICTED results. Create 3 more project (one file for each remaining gates) using structural Verilog programming for the 74LS00, 74LS04, and 74LS32. After you have done this, compile, download, and test each of them as you did the 7408 AND gate. Report your results.

Engineering Data:

```

1  module lab2_1b(in1,in2,led1,led2,led3,led4);
2      input in1,in2;
3      output led1,led2,led3,led4;
4
5      wire in1,in2,led1,led2,led3,led4;
6
7      //NAND function simulating 7400
8      nand g1(led1,in1,in2);
9
10     //NOT function simulating 7404
11     not g2(led2,in1);
12
13     //AND function simulating 7408
14     and g3(led3,in1,in2);
15
16     //OR function simulating 7408
17     or g4(led4,in1,in2);
18 endmodule
19

```

Figure 2: allgate

In this part of the lab, basically just had change the input and output coding. Since there got to be 2 input and 4 output. We will have to make basically 2 addition output from the original code from the tutorial of the and gate. Instead for this one we are using all the gates together after testing each one in wave form which is feature in figure 3.

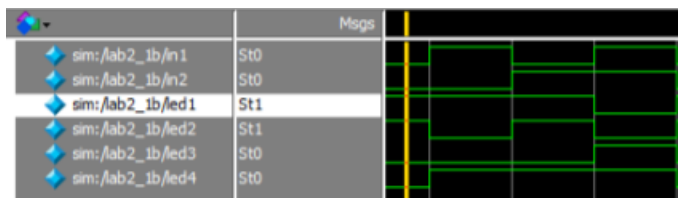


Figure 3: waveform

Part 2 Part 2: Exclusive OR gate

Description: We basically have to remake the exclusive or gate into multisim and also write it into Verilog and get a wave form.

Procedures: Each circuit below is a two input EXCLUSIVE-OR gate. Recall that an EXCLUSIVE-OR has a high output if one or the other inputs are high. Write a structural model to implement this function. You can write two description programs, one for each logic diagram, or combine them into one module.

Compile, download, and test each circuit. Show that each circuit satisfies the truth table for an EXCLUSIVE-OR function. Build the Exclusive OR gates with the 74XX gates and the CPLD/FPGA.

Using the given :

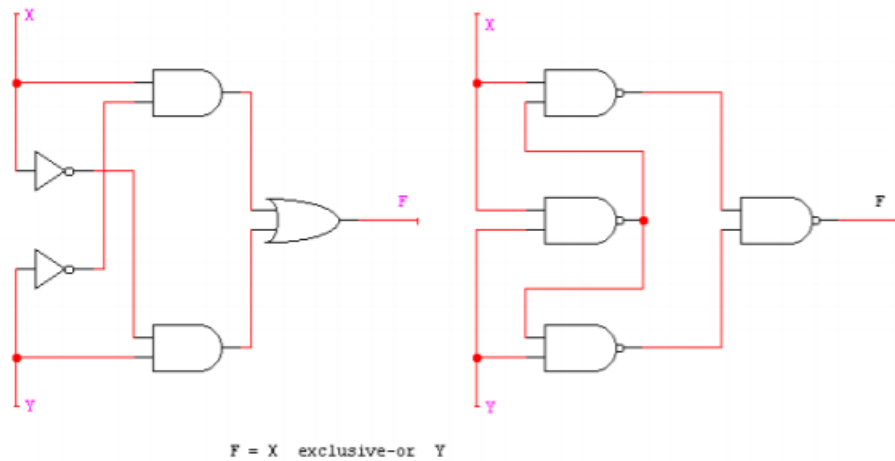


Figure 4: schematic

```
module xorgate(x,y,f);
    input x,y;
    output f;
    wire x,y,f;
    wire notxout,notyout,and1out,and2out;
    or or1(f,and1out,and2out);
    and and1(and1out,x,notyout),
    and2(and2out,y,notxout);
    not notx(notxout,x),
    noty(notyout,y);
endmodule
```

Engineering Data:

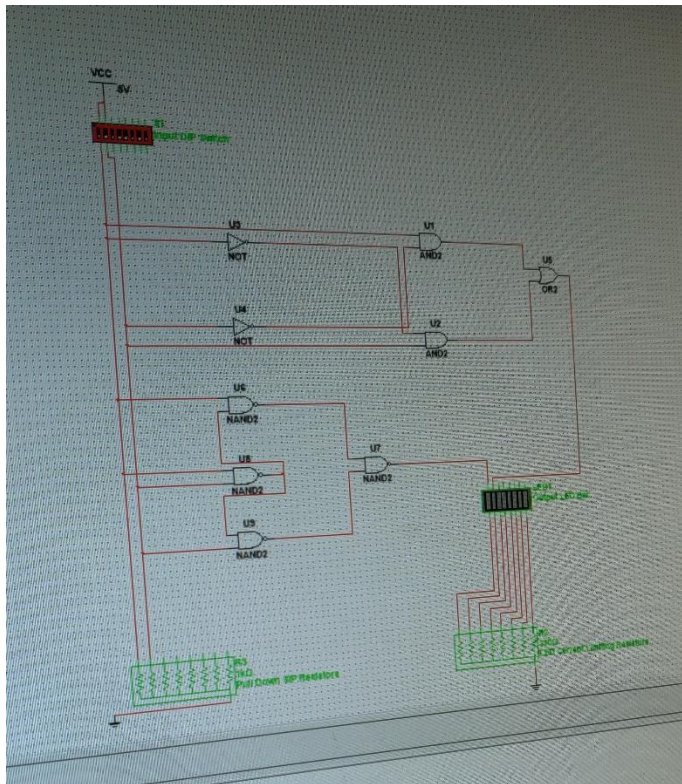


Figure 5: multisim

```

1 module Verilog1(A,B,C,D);
2   input A,B;
3   output C,D;
4
5   wire A,B,C,D;
6
7   wire Aout, Bout, C2, D2;
8
9   or OR1(C,Aout,Bout);
10  and A1(Aout,A,D2);
11  and A2(Bout,D,C2);
12  not not1(C2, A);
13  not not2(D2,B);
14
15  wire nandout, nandout2, nandout3;
16  nand nand1(D, nandout, nandout3);
17  nand nand2(nandout3, nandout2, A);
18  nand nand3(nandout2, A, B);
19  nand nand4(nandout, B, nandout2);
20
21 endmodule

```

Figure 6: Xor Verilog

For figure 6, we had to make an exclusive or Verilog using and or and not gate, for one and the other is made out of 4 nand gate. They are all based off figure 4 given in the procedure. Which was then used to be made into a multisim featured in figure 5

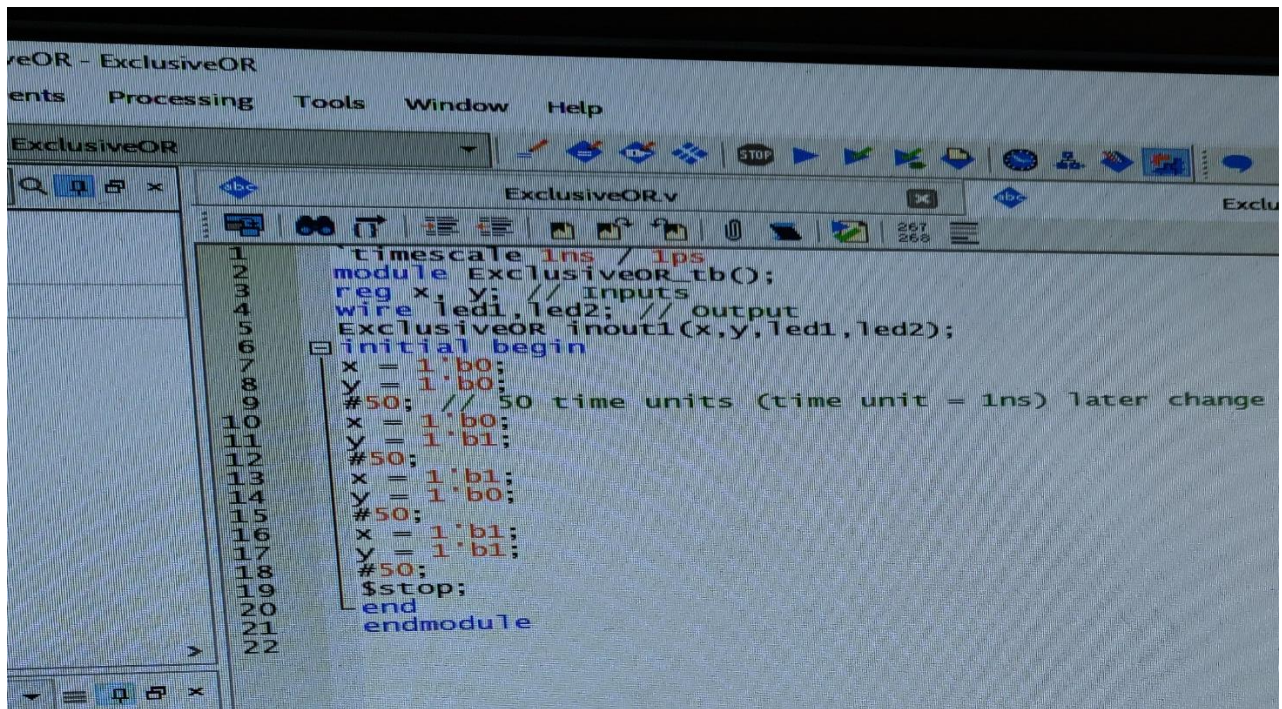


Figure 7: testbench

The testbench is then also used the same as the other Verilog, the only different is that the naming had to be changed . which is used to compute the waveform. In figure 7.

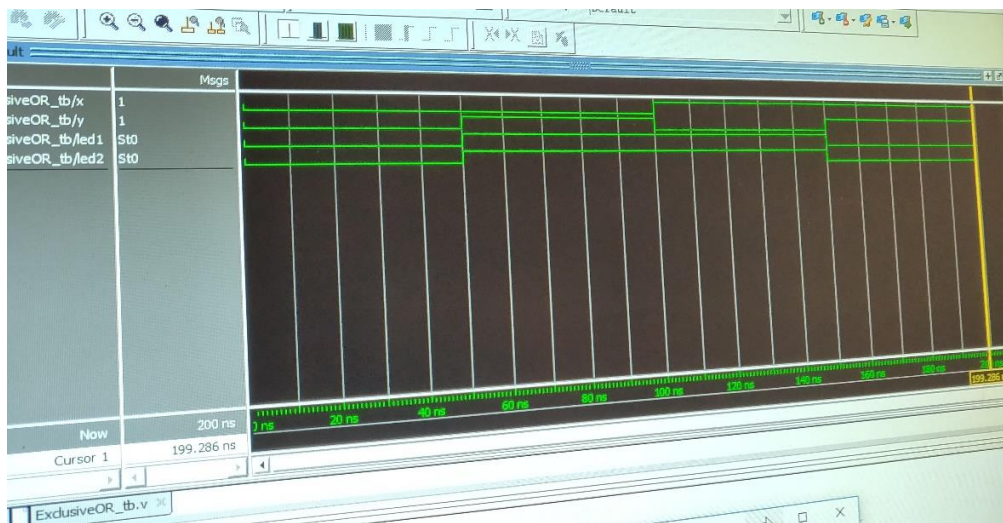


Figure 8: waveform

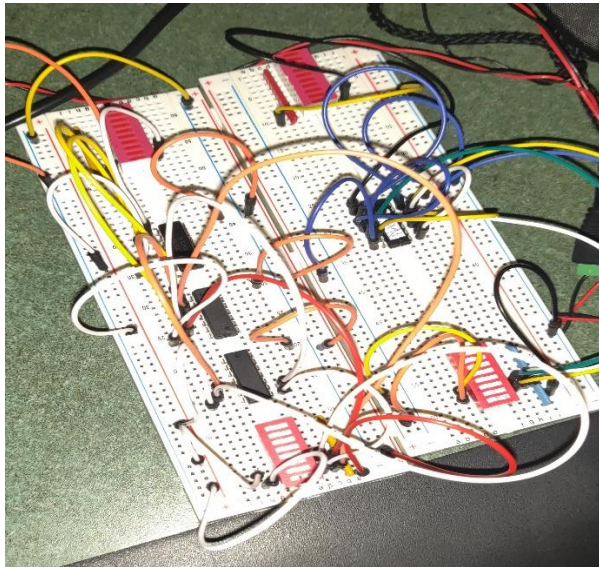


Figure 9: breadboard

As the exclusive or Verilog and waveform was done, we had to make a board that will output the result, so using to bread board and following the multisim, we were able to create figure 9. Both breadboard was working, only issue was some part of the gate chip was dead

Lab 2 Part 3

Description:

We must write a truth table and k-map given equation and then we must test it out on Quartus.

Procedures:

Consider a circuit with 4 inputs A, B, C, and D and one output F. Input A should be the most significant bit. Create a truth table to implement the equation below. Show all the work (Boolean Algebra) as part of the design to reduce the equation for F. Draw the logic diagram (with names of internal and external signals) for your final reduced equation. After completing your logic diagram, write the Verilog description using "Structural Modeling". Compile, download and test your design against the truth table you made from the original equation below.

Engineering Data:

Part 2

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

$$F = (\bar{C} + \bar{D})(\bar{A} + B + \bar{D})$$

$$F = (\bar{C} + \bar{D}) + (\bar{A} + B + \bar{D})$$

$$F = \bar{C}\bar{D} + \bar{A}\bar{B}\bar{D}$$

$$F = CD + A\bar{B}D$$

K-map

CD \ AB	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	1	1	0

Figure 10: simplification

For the simplification, all that was done was using the given equation and use demorgan on it to find its most reduce form. Featured in figure 10.

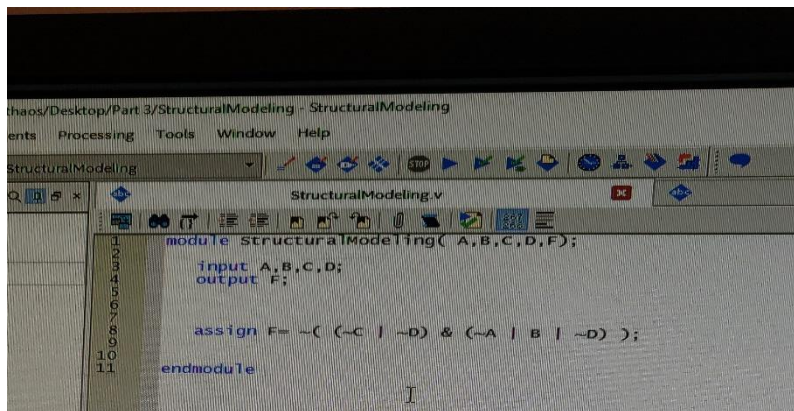


Figure 11: modeling

In figure 11, we basically assign F the given equation, with all the input and output. Using figure 12, we tested it out on and see if the equation works and It gave us a waveform which is in figure 13.

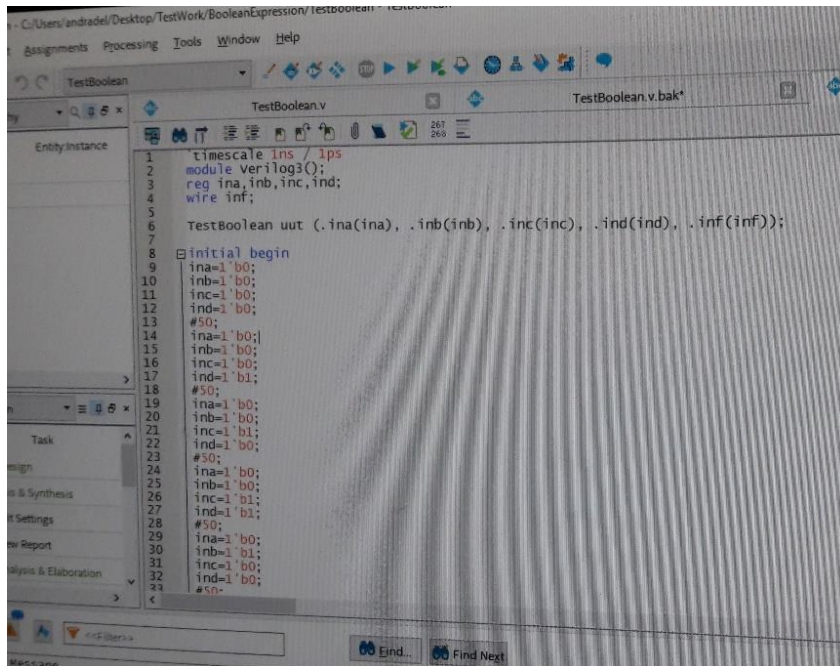


Figure 12: testbench



Figure 13: Waveform

Lab 2 Part 4

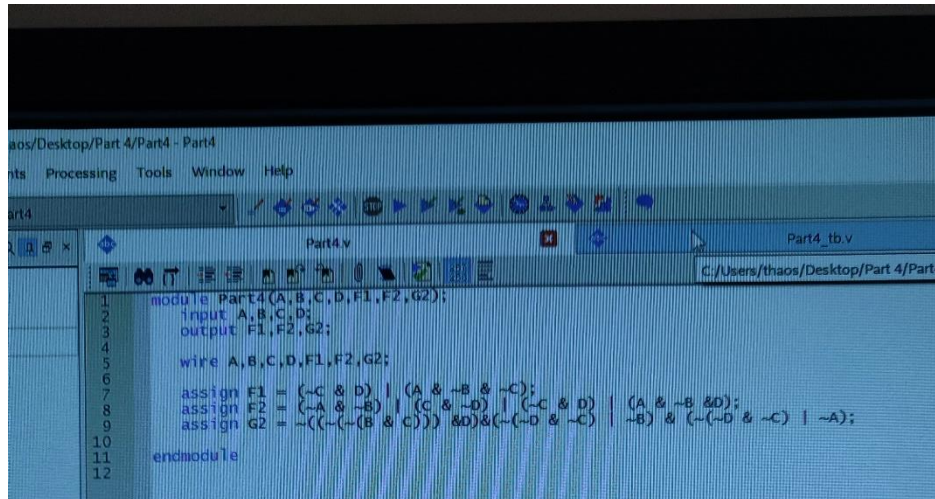
Description:

We must create a multisim and Verilog for the nand, not, 'and' and or gate. Then verify it with the truth table

Procedures:

This exercise is to improve your method of reduction techniques. Use only the four integrated circuits (ICs) packages: the 7400 (NAND), 7404 (NOT), 7408 (AND) and 7432 (OR). Wire your design in Multisim and Verilog and verify your designs with the Truth Table.

Engineering Data:



```
1 module Part4(A,B,C,D,F1,F2,G2);
2   input A,B,C,D;
3   output F1,F2,G2;
4
5   wire A,B,C,D,F1,F2,G2;
6
7   assign F1 = (~C & D) | (A & ~B & ~C);
8   assign F2 = (~A & ~B) | (C & ~D) | (~C & D) | (A & ~B & D);
9   assign G2 = ~((~((~(B & C))) & D) & (~(~D & ~C) | ~B) & (~(~D & ~C) | ~A));
10
11 endmodule
12
```

Figure 14: Verilog

Using the code given, in figure 14, we used the code to assign it to an individual character name, then assign it to an equation. We will have 4 input and 3 different outputs.

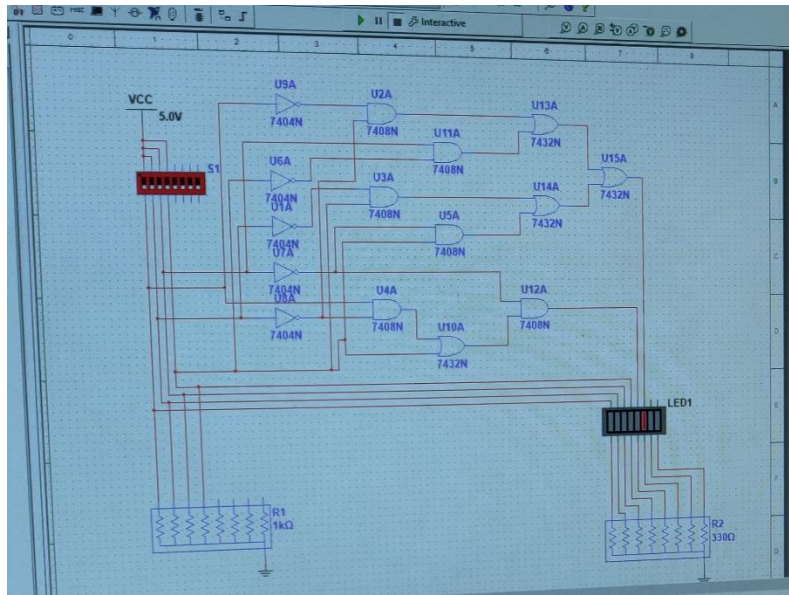


Figure 17: Multisim

For figure 17, a multisim of an nand, not, and and or gate was used. It was the result of the truth table being reduce to its simplest form.

Conclusion:

This Lab was very time consuming if you didn't have the correct equipment or correct setting then it wouldn't be do able. Also if one of your gate is busted, then basically you'll have to spend some more time trying to figure out what the issue was by testing each gates. The issue with this lab for me was that I wasn't able to do it at home since the FPGA wasn't register for my laptop no matter what I try to do. The Lab was very helpful in understanding reduction technique and also how the FGPA works. But I still feel it was lacking in teach me on learning how to operate the quartus and modelsim correctly, since during the whole process I have time where the program either didn't work or produces many error due to either the naming scheme or where the file was located on the computer.

Question:

#1: Is there an easier way to write an EXCLUSIVE-OR function still using primitives and structural modeling? Explain in the conclusion portion of your report.

Primitive form for Exclusive-or : " \wedge "

Structural modeling: `xor<gatename>(out,in,in);`

#2: If you really needed to use EXCLUSIVE-OR gates could you buy a 7400 Series TTL Integrated Circuit (IC)? If

so, what is the TTL part number and explain and draw the pin out of the device.

