Embedded Controller Stage C

Acknowledgements: Developed by Bassam Matar, Faculty at Chandler-Gilbert Community College, Chandler, Arizona.

Lab Summary: In this stage, we will build and design decoders, registers, and buffers to be added to our system we developed in the Programmable Logic Devices Module. When they are combined with an ALU, we can process data. However, we will need to manually set certain control switches as we still have no language in which we can program our system (i.e. brainless microprocessor).

Lab Goal: The goal of this lab is to understand how to create a 4-bits brainless microprocessor circuit and understand its functionality.

Learning Objectives

1. Compile and simulate a Decoder symbol in Quartus® II (web edition).
2. Create, compile, and simulate a one and four-bits register circuit/symbol in Quartus® II.
3. Create, compile, and simulate 4-bit buffer circuit/symbol in Quartus® II.
4. Create, compile, and simulate the Data Shuffling Circuit: The Brainless Microprocessor**.**
5. Program the design of the brainless microprocessor on a CPLD test board to determine its truth table.

Grading Criteria: Your grade will be determined by your instructor.

Time Required: 7 - 8 hours

Special Safety Requirements

Static electricity can damage the CPLD device used in this lab. Use appropriate ESD methods to protect the devices. Be sure to wear a grounded wrist-strap at all times while handling the electronic components in this circuit. The wrist strap need not be worn after the circuit construction is complete.

No serious hazards are involved in this laboratory experiment, but be careful to connect the components with the proper polarity to avoid damage.

Lab Preparation

- Read the WRE Micro& Embedded Controllers Part 2 Module Narrative.
- Read this document completely before you start on this experiment.

NOTE: This lab uses the information provided in Labs A and B that were completed in the WRE Programmable Logic Devices Module. You will be referred to these sections to review the needed information as required.

Equipment and Materials

Each team of students will need the test equipment, tools, and parts specified below. Students should work in teams of two.

Test Equipment and Power Supplies	Quantity
The following items from the UP2 Educational Kit: • Altera UP-2 circuit board with ByteBlaster Download Cable • Quartus® II Web Edition software • AC adapter, minimum output: 7 VDC, 250 mA DC	1
ESD Anti-static Wrist Strap	1
#22 Solid-core wire	As needed
Wire Strippers	1

Additional References:

1. *Digital Design and Implementation with Field Programmable Devices* textbook found with UP2 educational kit.
2. UP2 educational kit data sheet found on Altera web site: **www.altera.com**

Introduction

In the WRE Programmable Logic Controller Module, we designed an arithmetic logic unit (ALU) that can be used in a microprocessor. To use it in the microprocessor, the circuits must produce the input data and store the output data of the ALU. To do this, we need circuits that activate memory locations that are described by a binary address. Such circuits are one type of decoder.

In this stage of the lab, we will compile decoders, create the register, build a buffer, and build a data shuffling circuit.

Decoders

Decoders take a numerical input and have one output for every possible input value. The outputs are numbered. Usually the decoder will operate so that all outputs but one are 0 (active low decoder). Occasionally a decoder will be built so all outputs but one are 1 (active high decoder). The input number determines the long exceptional output. For example if the input number is two bits wide, there are four outputs, Y_0, Y_1, Y_2, and Y_3 (one for each possible value of the input number) or $2^3 = 8$, where 3 is the number of inputs. If the input number is 3 or (11 in binary), then output Y_3 will be 1 and all other outputs will be 0 if it is an active high decoder.

Often a decoder will be built with an “enable” input. This enable input will disable the decoder if it is a 0 so that all outputs of the decoder will be 0. This extra input allows decoders to be stacked so that bigger decoders can be made from smaller decoders. The decoder is used for selection like a multiplexer and demultiplexer. A decoder's outputs are usually connected to the enables of other circuits so that the decoder selects which devices are enabled and which devices are disabled. This is most often seen in memory address decoding in computers.

We could stack five (2-4) decoders in a pyramid to form a (4-16) decoder. However, in this lab, we will use the Altera library and test and compile an active low 4-16 decoder (the 74154 device from ***Others→Maxplus2*** library).

A low level on G_{1N} and G_{2N} is required in order to make the enable gate output high. Hence, G_{1N} and G_{2N} are active low. The enable gate output is connected to an input of each NAND gate in the decoder so it must be high for the NAND gates to be enabled. If the enable gate is not activated by a low on both inputs, then all sixteen decoder outputs will be high regardless of the states of the four input variables (DCBA) as shown below where D is the MSB and A is the LSB.

G1N	G2N	D	C	B	A	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
1	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	1	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	**0**	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	1	1	1	1	**0**	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	**0**	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	**0**	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	**0**	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	1	**0**	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	**0**	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	**0**	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	1	1	1	1
0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	**0**	1	1	1
0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	1	1
0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**

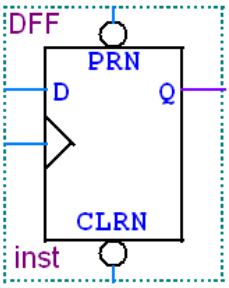
Sequential Circuits

Combinational logic is logic whose output is a function of only the present inputs: its outputs do not depend on past inputs. On the other hand, sequential logic has memory: its outputs depend on past inputs in addition to the present inputs. One of the most important aspects of digital hardware is its ability to store information and it is totally necessary in our microprocessor.

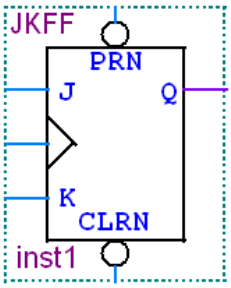
The most elementary storage device is the latch. A typical latch has one input for data and another input which tells the latch when to store new data and when to hold the data it has. The latch has a single output which always represents the current value stored in the latch.

Latches are useful and asynchronous devices. However, for our purposes, we need to deal with synchronous devices. For this reason, we will not talk any more about latches but instead we will talk about an extended latch called the flip-flop.

Flip-flops also hold data like a latch but they do not have a normal input as their control input. The control input of a flip-flop is edge-triggered which means that it is not activated by a 1 or by a 0 but rather by a transition from a 0 to 1. This means that the time when the flip-flop loads data can be precisely controlled to a single instant. Because the control input keeps the time of when data is to be loaded, the control input keeps the time of when data is to be loaded. The control input is called a **clock** and is often given a special symbol on the diagram of the device. In Altera software, there are several available flip-flops to we could use. The standard D and JK flip-flops look like this:



D Flip-Flop



JK Flip-Flop

In the D flip-flop, you can think of the D as standing for "data," because the D input of the D flip-flop gives the input data. The other input with arrow "→" is the clock which is the flip-flop's control input. Whenever the clock goes from 0 to 1, a snapshot of the value present on the D pin is loaded into the internal storage of the flip-flop. The PRN and CLRN (active low) pins are for setting and resetting the flip-flop. They are normally connected directly to V_{cc} and are used to reset logic that is only activated in extreme circumstances. If PRN is set to 0, the value inside the flip-flop is instantly set to 1. If CLRN is set to 0, the value inside the flip-flop is instantly set to 0. It is unclear what the internal value will be set to if both PRN and CLRN are set to zero at the same time. The value of Q is the value stored in the flip-flop.

JK flip-flops are similar to D flip-flops. The inputs are "→" CLOCK, PRN, CLRN, and Q pins are exactly the same as on the D flip-flop. The difference is in the data inputs. Where the D flip-flop had one data input D, the JK flip-flop has two data inputs, J and K. This gives more possibilities for what can happen when the clock changes and a snapshot of the inputs is taken. The next value of the data stored in the JK flip-flop is determined by the values of J and K at the time of the transition, as follows:

J	K	NEXT STORED VALUE
0	0	SAME AS LAST STORED VALUE
0	1	0 (RESET)
1	0	1 (SET)
1	1	INVERSE OF LAST STORED VALUE

Note: PRN and CLRN must be connected directly to V_{cc}

In contrast, the D flip-flop has the following table:

D	NEXT STORED VALUE
0	0 (SAME AS INPUT)
1	1 (SAME AS INPUT)

So with the JK flip-flop, you have some additional capabilities built in. These capabilities are not too big a difference; you can make a D flip-flop from a JK flip-flop and vice versa using only combinational logic.

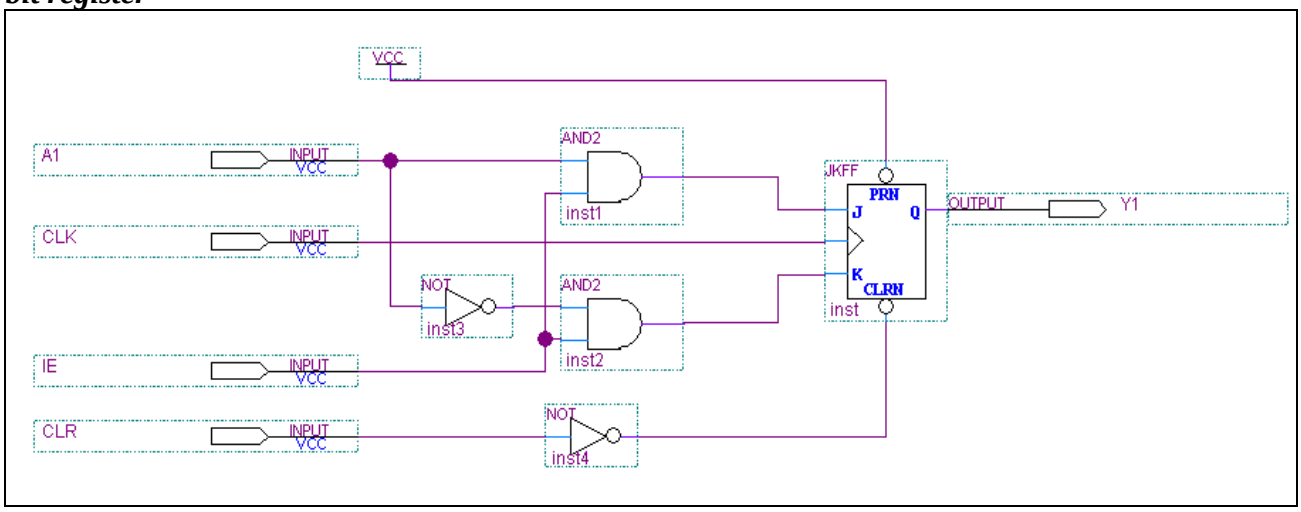
The Altera JK flip-flop we will be using is a positive edge triggered flip-flop which means that it stores data on the low to high (0 to 1) transition of the clock. Whether a flip-flop is negative edge triggered or positive edge triggered makes little difference since you can easily change from one to the other with a single inverter. The only considerations are which is most easily available and which you need in your design.

When you use a flip-flop, every input should be connected. Flip-flops have been known to behave strangely if inputs are left unconnected, since electrical noise can reset or set the flip-flop.

Registers

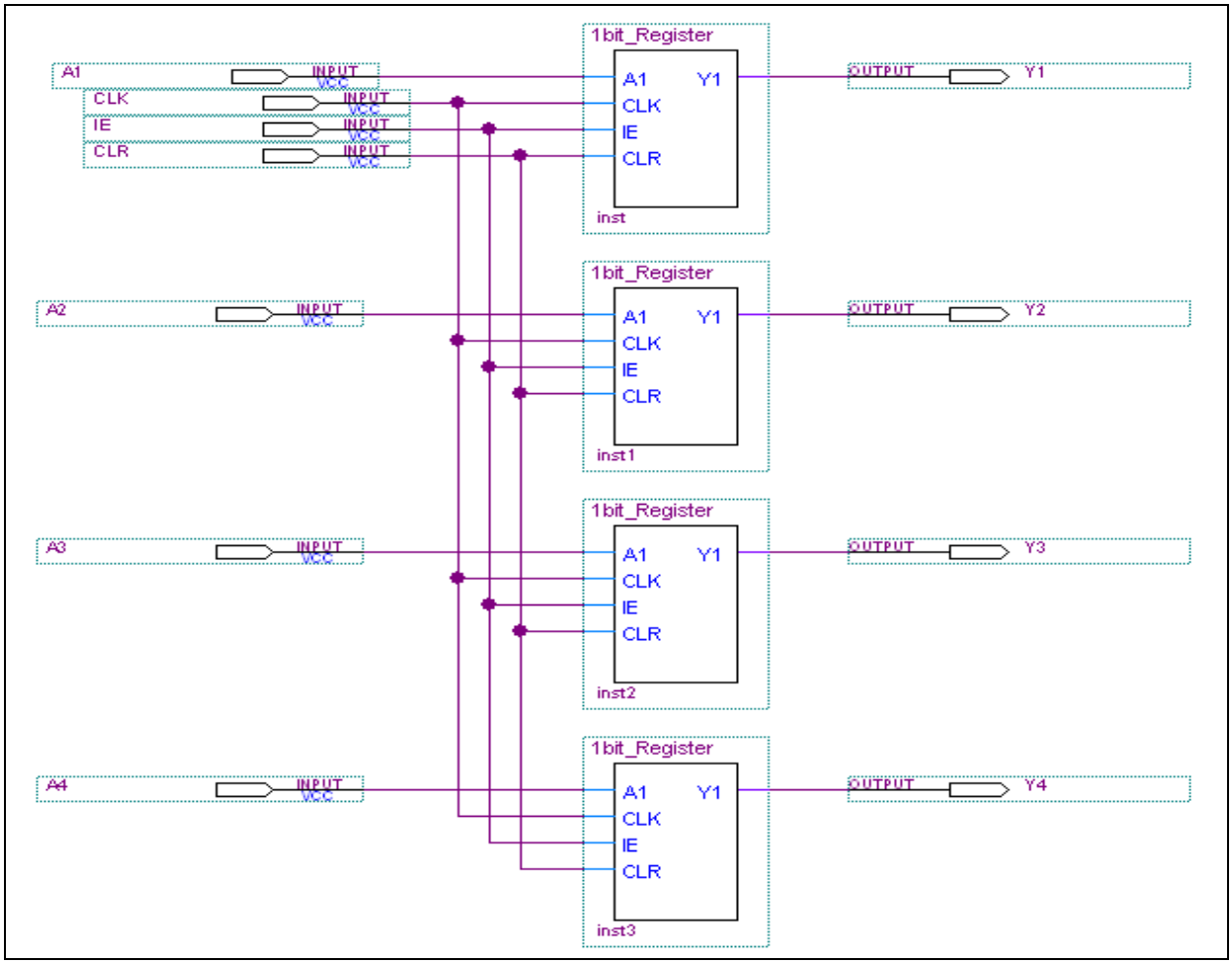
The circuit formed from flip-flops that we will use in our microprocessor design project is a register. The following is a circuit for a one-bit register (REGISTER–1). When four of these are combined, we get a four-bit register, which has the Altera picture shown below. Note that the only differences between a register and a D flip-flop are the absence of a PRN and the presence of an enable. It turns out that this enable (also called EN or IE, which stands for input enable) is much more useful than you would think.

1-bit register



The one-bit register holds a single bit of information. In order for the information in the register to change (i.e., for data to be loaded into it), the input enable must be a 1, and the clock must simultaneously have a transition from 0 to 1. When both of these occur, a snapshot of the data input(s) is loaded into the register. It is most convenient in practice to have the clock be a square wave at a fixed frequency. When the clock is a square wave, the input enable selects on which edges of the square wave the register is loaded. The input enable becomes a load control, which is set to a 1 for those clock periods during which the register is to load new data.

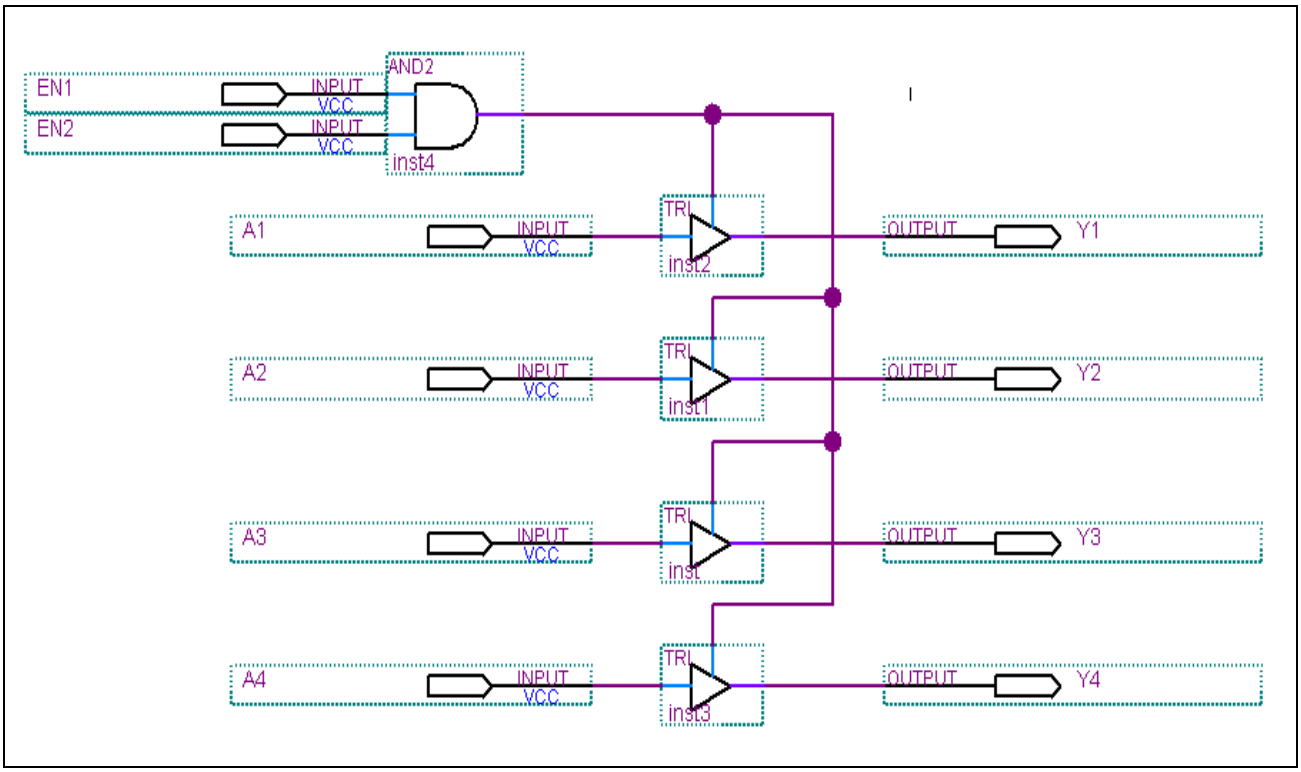
We can cascade four 1-bit registers to build a register-4 circuit as shown here



The REGISTER–4 has four data inputs, four data outputs, one clear input, one enable input, and one clock.

Buffer Circuit

For the buffer circuit we could use the tri-state buffer available in the Altera software library. However, we will need something just a little more elaborate, so we will build our own buffer circuit. The tri-state buffer in Altera has a single active high enable. For our buffer (called BUFFER4), we want two active high enables, which have to both be high to enable the four tri-state outputs. We will do this with four tri-state buffers and a AND gate.



EN1	EN2	A's Inputs	Y's Output
0	0	X	Z
0	1	X	Z
1	0	X	Z
1	1	4	4
1	1	A	A
1	1	C	C

As long as one or more of the EN is 0, it does not matter what the input is, the output is always high impedance (Z). When both ENs are high, the 4-bits buffer passes whatever you put on your inputs. INPUTS = OUTPUTS.

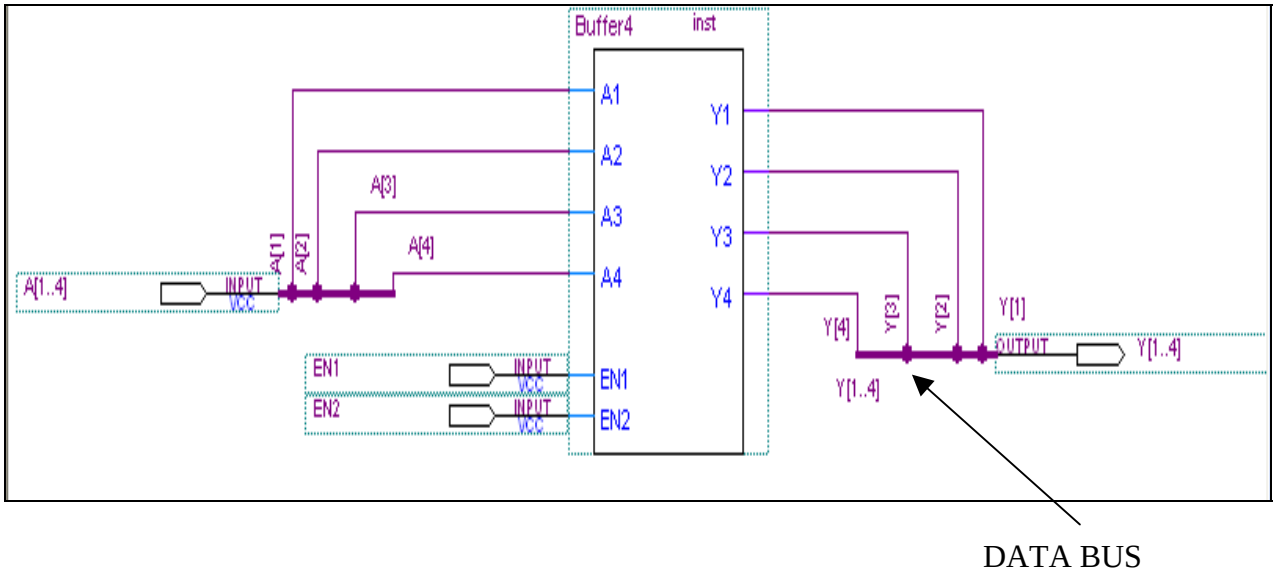
Data Shuffling Circuit

In the data shuffling circuit that we build in the lab, when the ALU is in "pass through" mode, the circuit of the brainless microprocessor is in a "data shuffling" circuit. We can select how data is moved from the register after the ALU back into "memory." If the ALU is not in "pass through" mode, the circuit still shuffles data, but now the data is manipulating the ALU control lines. You can also select what operations are performed on the data in between reading it in and writing it out.

The circuit is almost a microprocessor. It would be a microprocessor if it only had enough "brains," more frequently called control circuitry so that it could follow a set of instructions about what operations to perform. We will add the control circuitry in the next lab. For now, you will function as the "brains" of the microprocessor to manipulate the control lines to get it to process the data. This is used to familiarize you with the microprocessor layout before we add the control.

Most of the bottom right of the circuit in the brainless microprocessor is the microprocessor memory. There are three types of microprocessor memory here: Read Only Memory (ROM), Read/Write Memory (more often called Random Access Memory, or RAM), and Write Only Memory, which corresponds to an output device.

The circuit below is a single 4-bit ROM memory circuit:

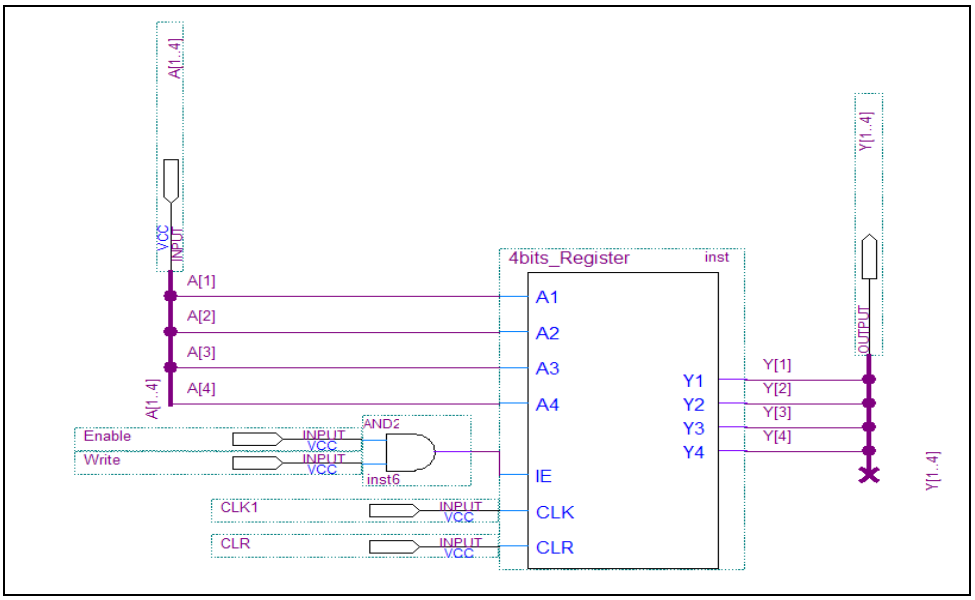


You select what data is "stored" in the memory by your selection on the 4 inputs. When both of the enables are high, the 4-bit ROM circuit puts whatever data is stored in it out onto the data bus.

In the brainless microprocessor circuit, one of the enables is always connected to the read line. The Read line is high whenever data is being read from memory. If data is being stored to memory, the read line will be a 0. Since we only want to activate the buffer after reading a value from memory, the READ is connected to one of the enables on the buffers.

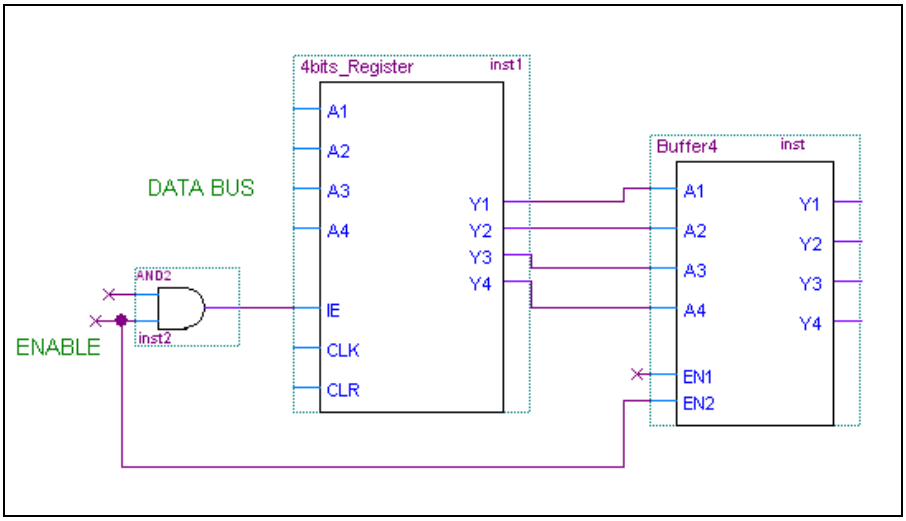
The other enable is connected to a decoder which decodes the address of the data we wish to access. Each ROM has a different address so by making the read line a 1 and making the address equal to the address of a particular ROM, the value in that ROM will be put onto the data bus. This is how memory is accessed for a typical read cycle.

In addition to memory which can be read, there is also memory, which can be written to. One example of this is the write-only output circuit shown below:



This output circuit also has two enables. In our brainless microprocessor, one enable goes to the decoder to select the memory address of this output, and the other enable goes to the write line. Read and write are never 1 at the same time even though they may both be 0. When write is a 1, data is being taken from the bus and put into memory. So when the address of an output unit is selected and the write line is high, the register in the output unit will store data off of the bus. Whatever is stored in the register, it should be seen on the register output (OUT1…4), which is an output from the circuit. Note that since the output circuit is only active when data is being written, and the ROM is only active when data is being read, an output circuit and a ROM can share the same address.

Another type of circuit, the RAM, can be both written to and read from. Its circuit has elements from both the output circuit and from the ROM:



Data can be written into a RAM from the data bus, and then that data can be put back onto the data bus later. Since a RAM can both read and write, it cannot share the same address with a ROM or with an output.

The register to the right of the ALU in the brainless microprocessor is special. It is not part of the normal memory. When you are doing a write to memory, it is this special register that will be putting data onto the bus, while one of the output's or RAM's reads the data off of the bus. When you read data from memory, the result that comes out of the ALU will go into this register. A register used like this is often called an accumulator.

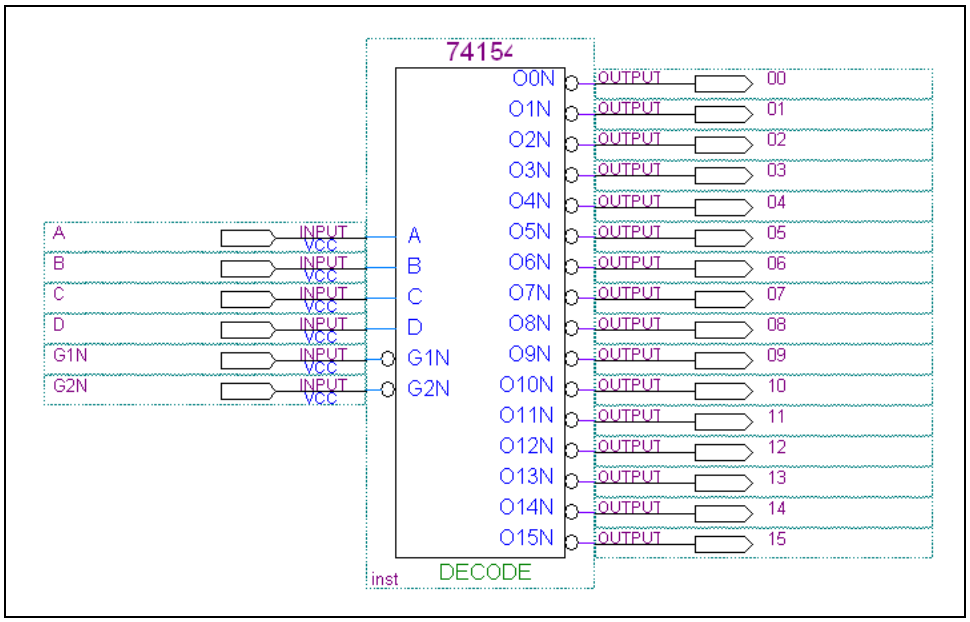
Understanding the Brainless Microprocessor

For those of you who have wondered how a computer works, in this task we come very close to asking you to explain how a computer works. Because this circuit contains an ALU, both arithmetic and logical operations can be performed. It is important that you stop and make sure that you understand how this circuit works.

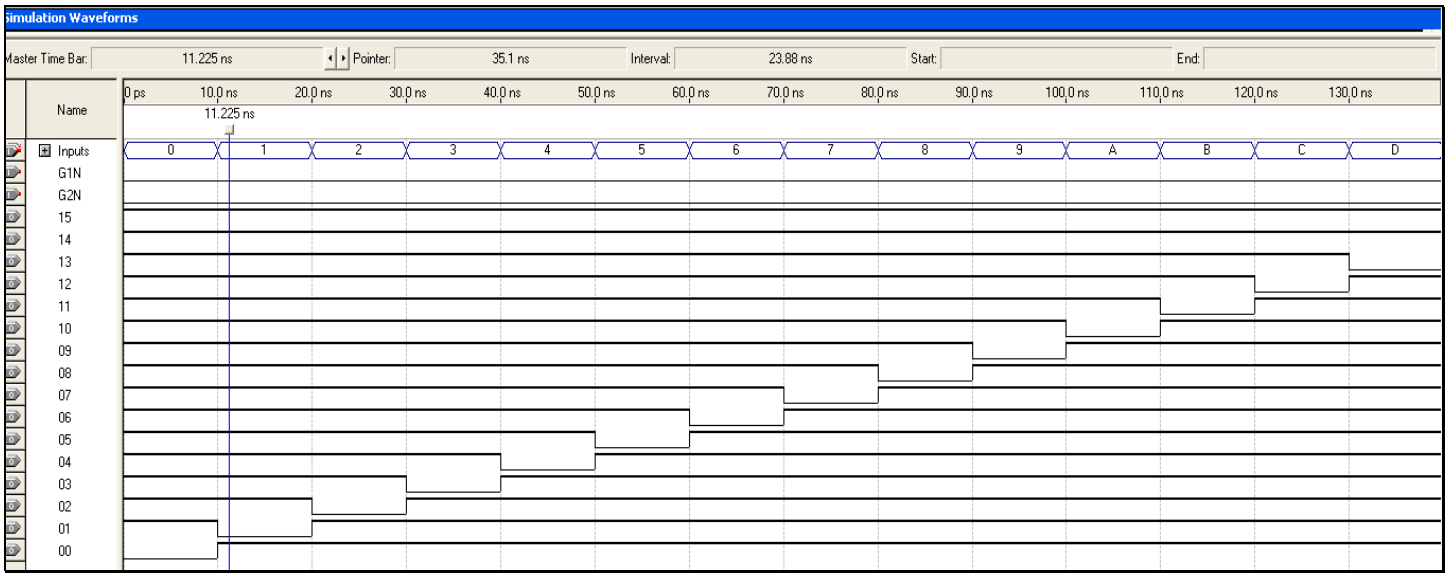
We will briefly explain how an addition is performed. By selecting the appropriate control lines and then giving the circuit a clock pulse, data can be moved from memory address 1 (ROM 1) to the accumulator. By selecting the appropriate control lines again, data can be taken from memory location 2, added to the value in the accumulator, and the result can be stored back into the accumulator. This is where the accumulator gets its name, because here it is "accumulating" a sum. Next, by selecting the appropriate control lines, data can be moved from the accumulator onto the bus, and stored back to output register. A complete addition has been performed.

Lab Procedure:

1. Compile and simulate a 4-16 Decoder
2. Design and Build a DECODER–4.
   a. Use the Altera library and test and compile the 74154 device from *Others →Maxplus2* library. Notice this is an active *LOW* 4-16 decoder.

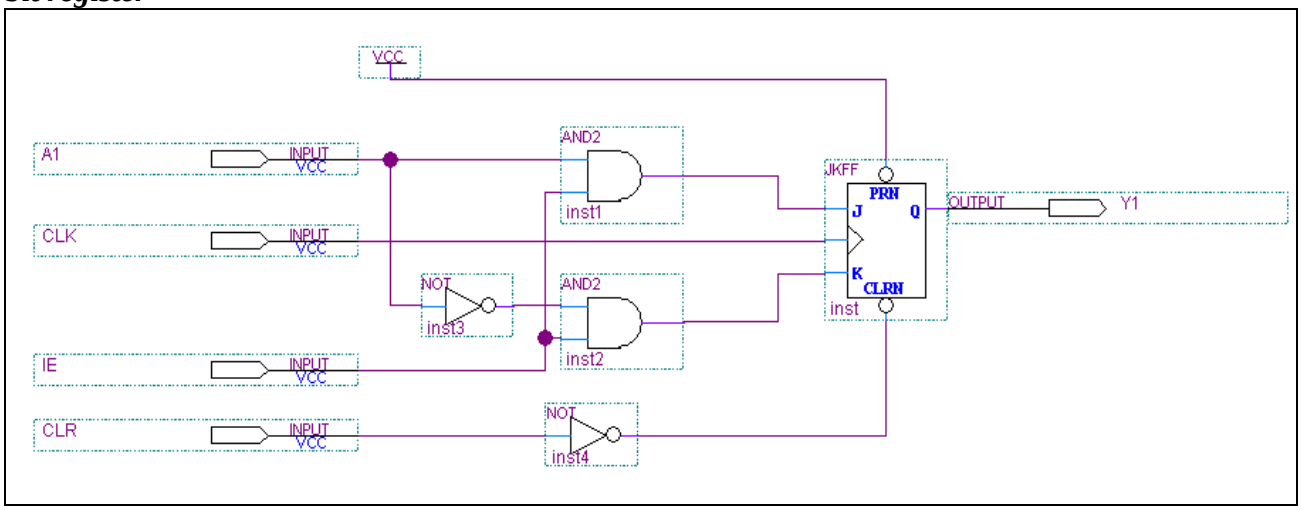


   b. Test the above device and make sure your results correspond to the truth table as shown in the introduction. Note: A is the LSB and D is MSB.
   c. Your timing diagram results should look like this

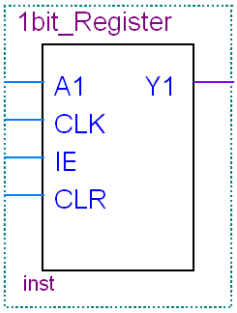


3. Create, compile, and simulate a one bit register.
Note: In Altera, these flip-flops can be found in the "***Primitives, Storage***" library

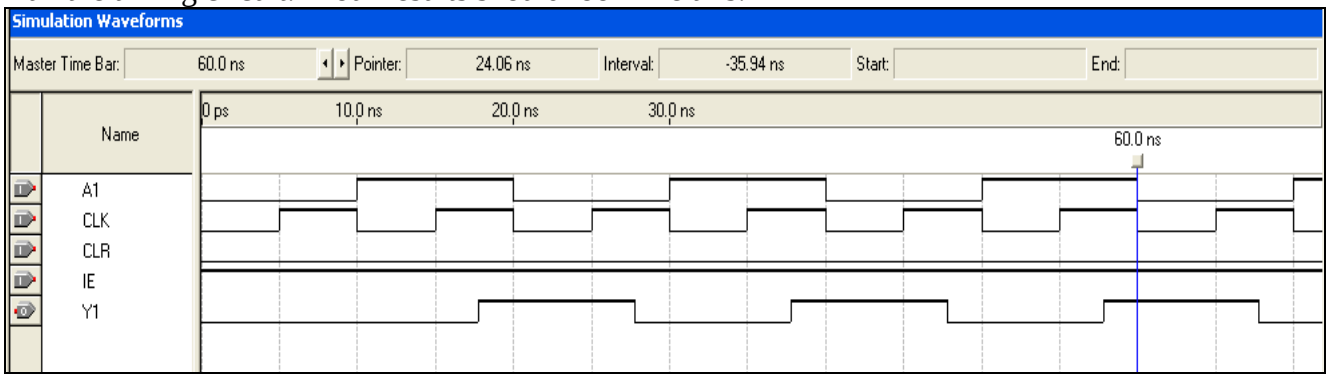
1-bit register



a. Use this symbol for register-1.

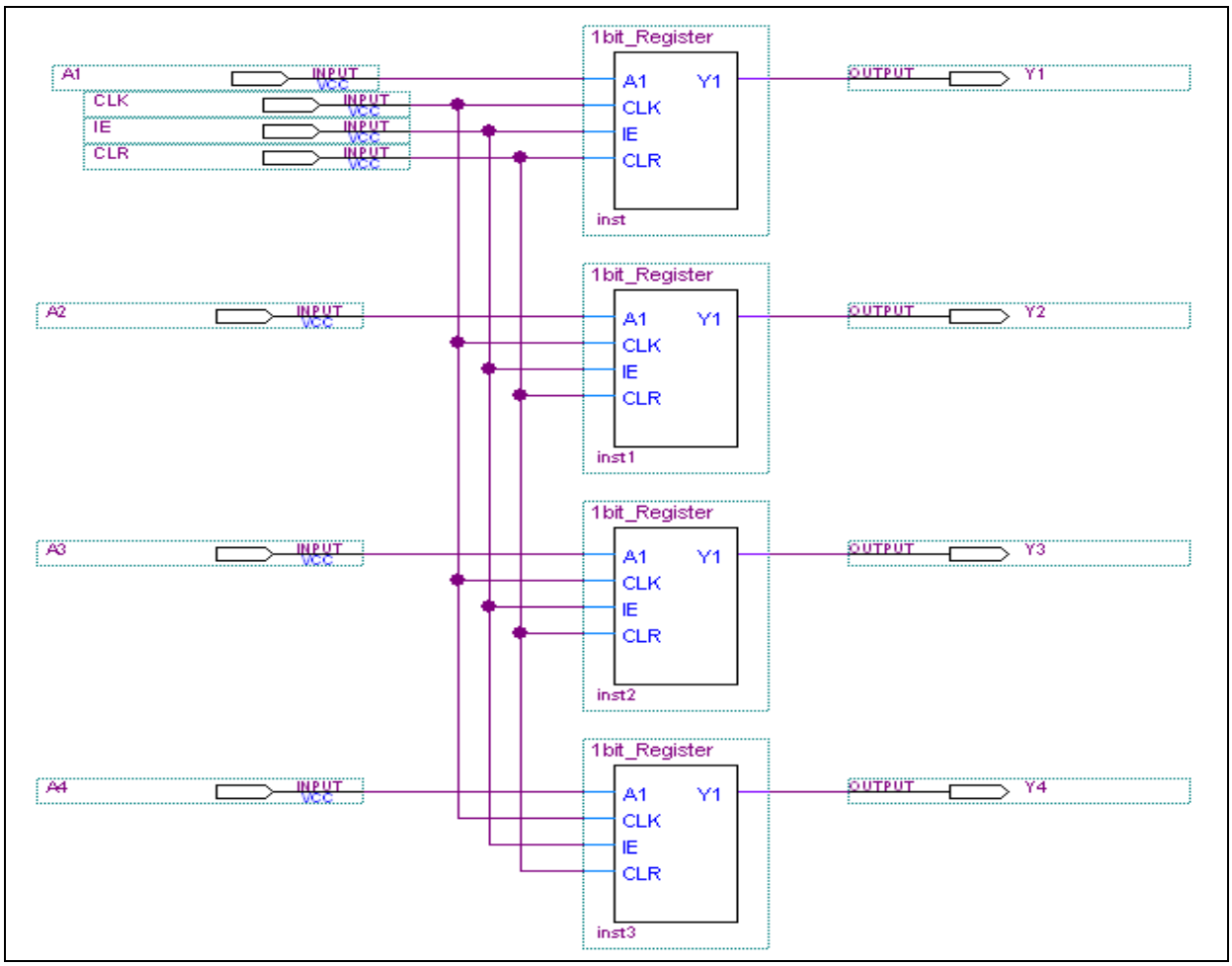


b. Run the timing circuit. Your results should look like this:

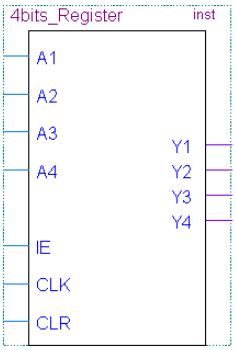


Note: The JK flip-flop is a positive edge triggered.
There is a about 3ns delay between input A_1 and output Y_1 according to our compilation.

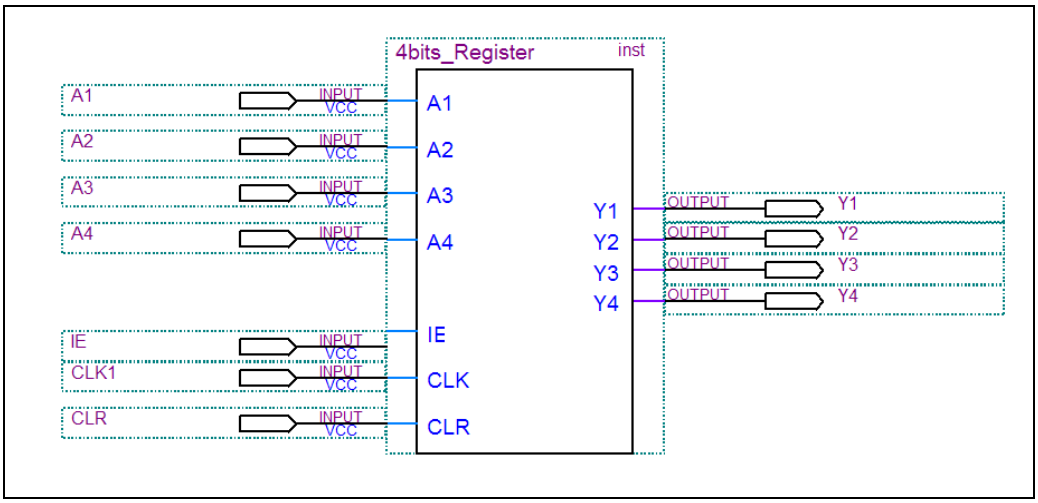
4. Use Altera Software to build the REGISTER–4 circuit by cascading 4 Register-1s as shown below. The REGISTER–4 should have four data inputs, four data outputs, one clear input, one enable input, and one clock.



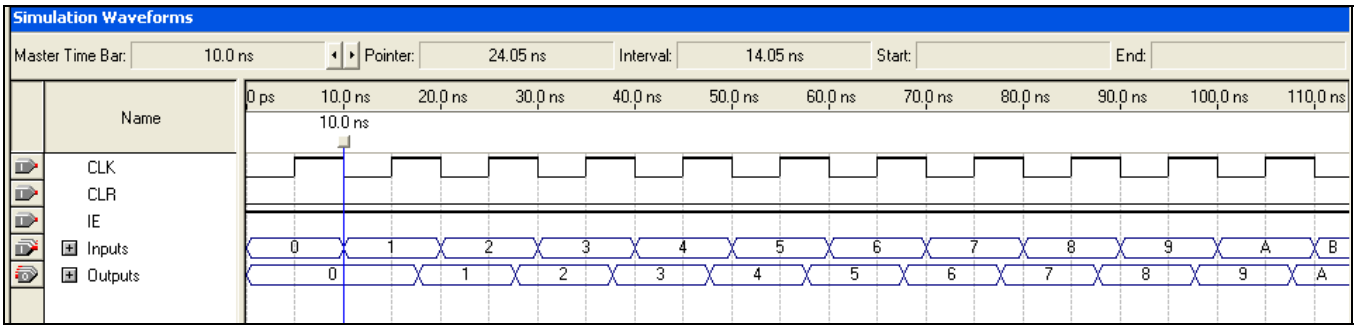
5. Design the REGISTER–4's picture and add it to your library.



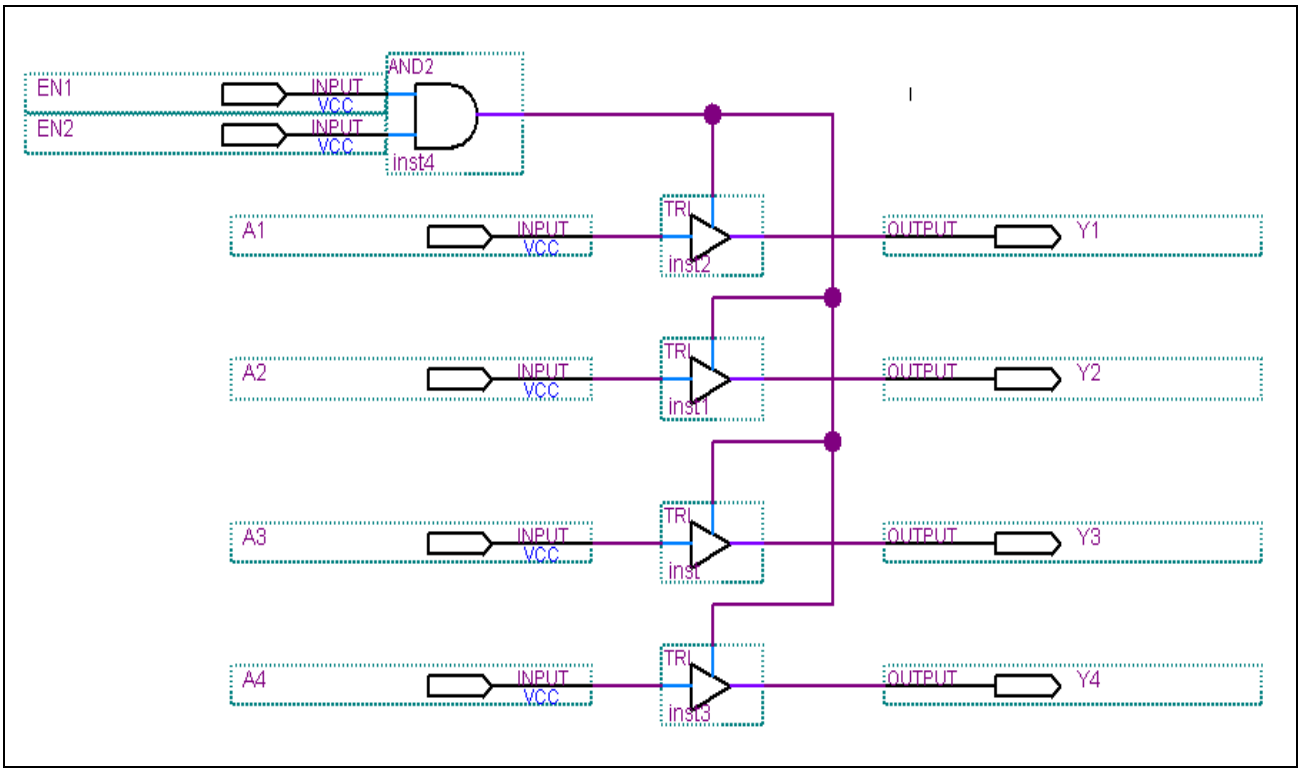
a. Test the Register_4 device. In your report, mention briefly how you tested them and comment on the reliability of your test. Remember to add the device actual circuit when you create the new circuit. Ex: Register4.bdf



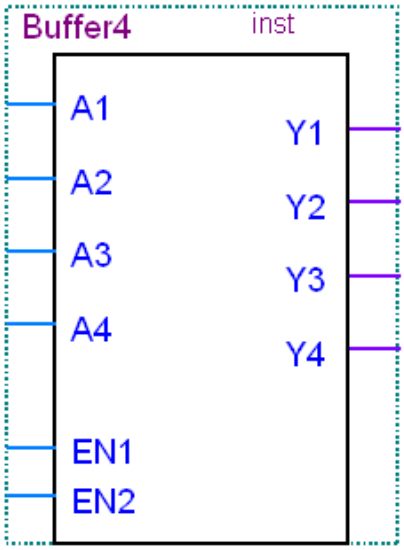
b. Run the timing circuit. Your results should look like this:



6. Build the 4-bit Buffer with two active high enables which have to BOTH be high to enable the four tri-state outputs.
   a. You can do this with four tri-state buffers and a AND gate.
   b. Design this circuit, and add it to your Library.



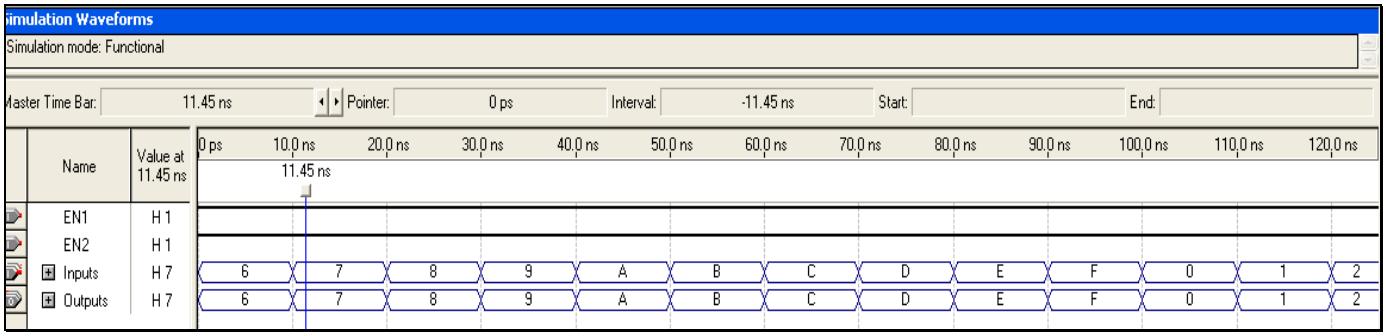
   c. Use the following symbol layout:



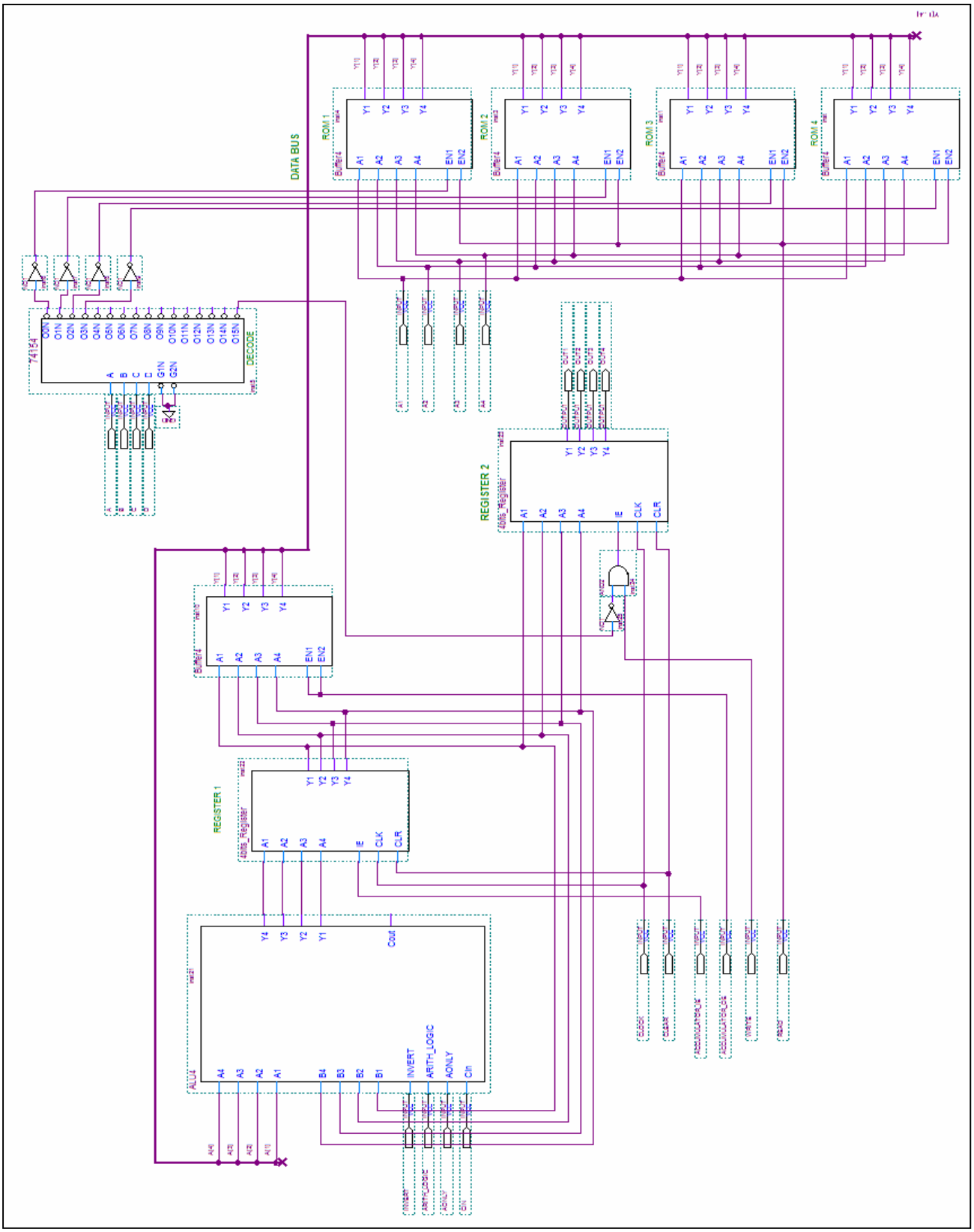
d. As long as one or more of the EN is 0, it does not matter what the input is, the output is always high impedance (Z). When both ENs are high, the 4-bits buffer passes whatever you put on your inputs. INPUTS = OUTPUTS.

EN1	EN2	A's Inputs	Y's Output
0	**0**	**X**	**Z**
0	**1**	**X**	**Z**
1	**0**	**X**	**Z**
1	**1**	**4**	**4**
1	**1**	**A**	**A**
1	**1**	**C**	**C**

e. Test your Buffer_4 device. Your results should look like this:



7. Build the Data Shuffling Circuit: The Brainless Microprocessor on the next page. Do not panic when you look at this circuit! You already have everything you need to wire up this circuit. It is really not too hard to build, but it has some very promising capabilities.

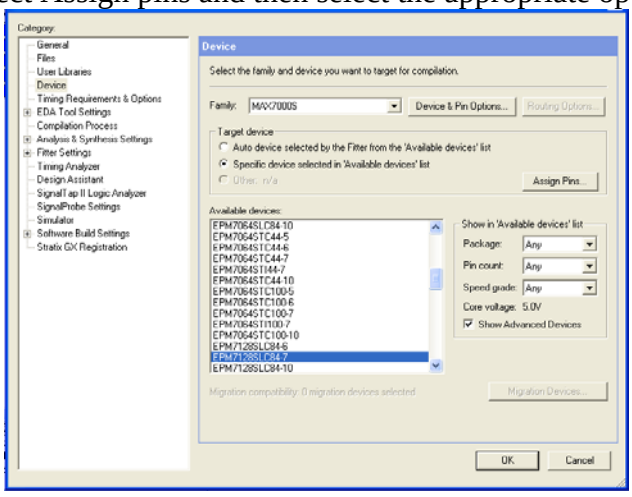

DATA BUS
ROM 1
ROM 2
ROM 3
ROM 4
Buffer4
A1
A2
A3
A4
EN1
EN2
Y1
Y2
Y3
Y4
74154
DECODE
REGISTER 2
REGISTER 1
ADSB_Register
IE
CLK
CLR
ALU4
Cout
INVERT
ARITH_LOGIC
AONLY
Cin

8. Write a report to show that you understand
   a. For your report:
      i) Detail what the control lines must be for each step of this addition process.
      ii) Repeat this to perform an OR operation on two numbers.
      iii) Repeat this again to subtract two numbers by taking a two's complement. Do not worry about overflow or carries.
   b. Test all of your control line configurations on your circuit in Altera.
   c. Record and report the results by making a truth table that includes all the control lines and the appropriate functionality of the brainless microprocessor as follow:

Addition Operation:

Clear	Clock	Decoder	ROM 1	ROM 2	A Only	Invert	Logic	Read	Write	ACC. IE	ACC. OE	RESULTS
0	0→1	0	3	5	0	0	0	1	0	0	0	Pass data 3 through the ALU
0	0→1	0	3	5	0	0	0	1	0	1	0	Data 3 is the input value of the ALU B inputs
0	0→1	1	3	5	1	0	0	1	0	0	0	Data 5 from ROM2 is on the A's inputs of the ALU. You should also see the addition of A=5 to B=3 of 8 on the output of the ALU
0	0→1	F	3	5	0	0	0	0	1	1	1	The sum 8 should show up on the output of register 2

6. Program the design of the Brainless Microprocessor on a CPLD test board to determine its truth table.
   a. Configure the device using the MAX7000s device family and the EPM7128SLC84-7 CPLD.
   b. From ***Assignments*** menu, select Assign pins and then select the appropriate options as shown below:

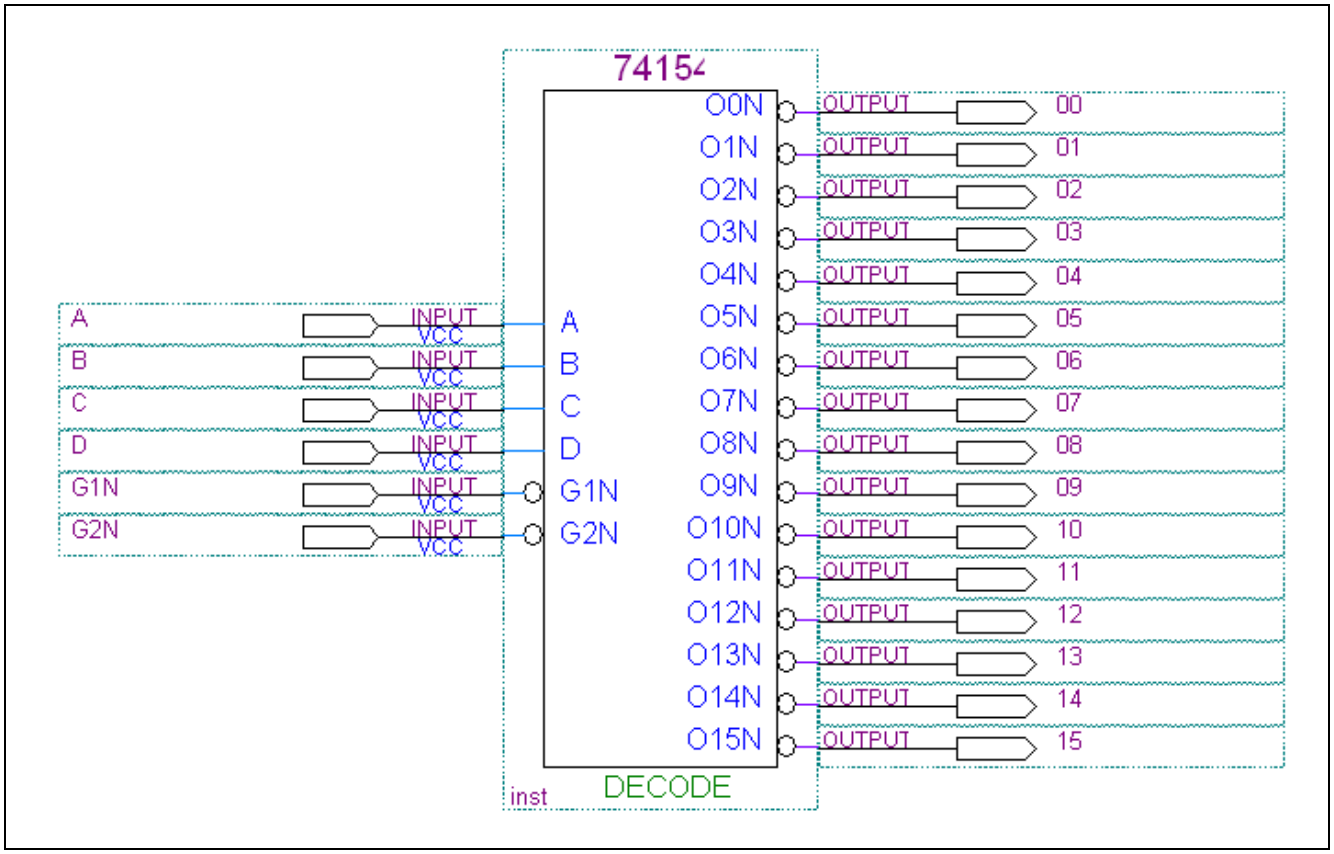


c. Compile your circuit one more time to make sure there are no errors.
d. Run functional simulation for sub circuits.
e. Run a timing simulation.

7. Follow the directions from the previous labs to program the brainless microprocessor into the UP2 board.

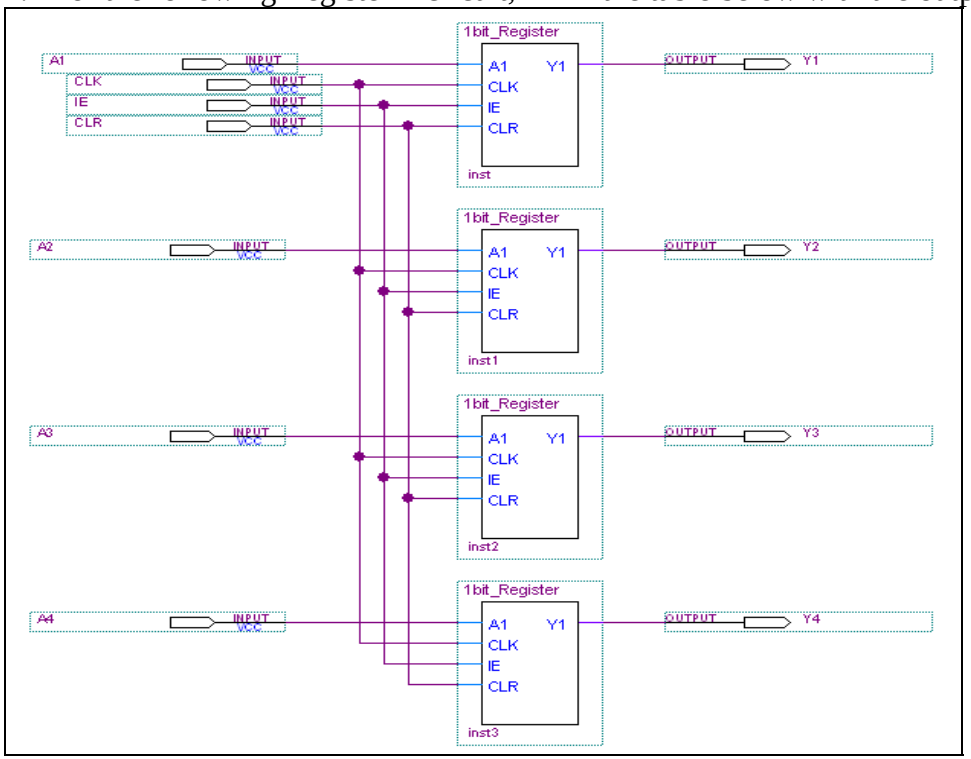
Lab Questions

1. For the following Decoder-4 device, which of the outputs (00…15) will be active based on the inputs of ABCD? *Remember, A is the LSB and D is the MSB.*



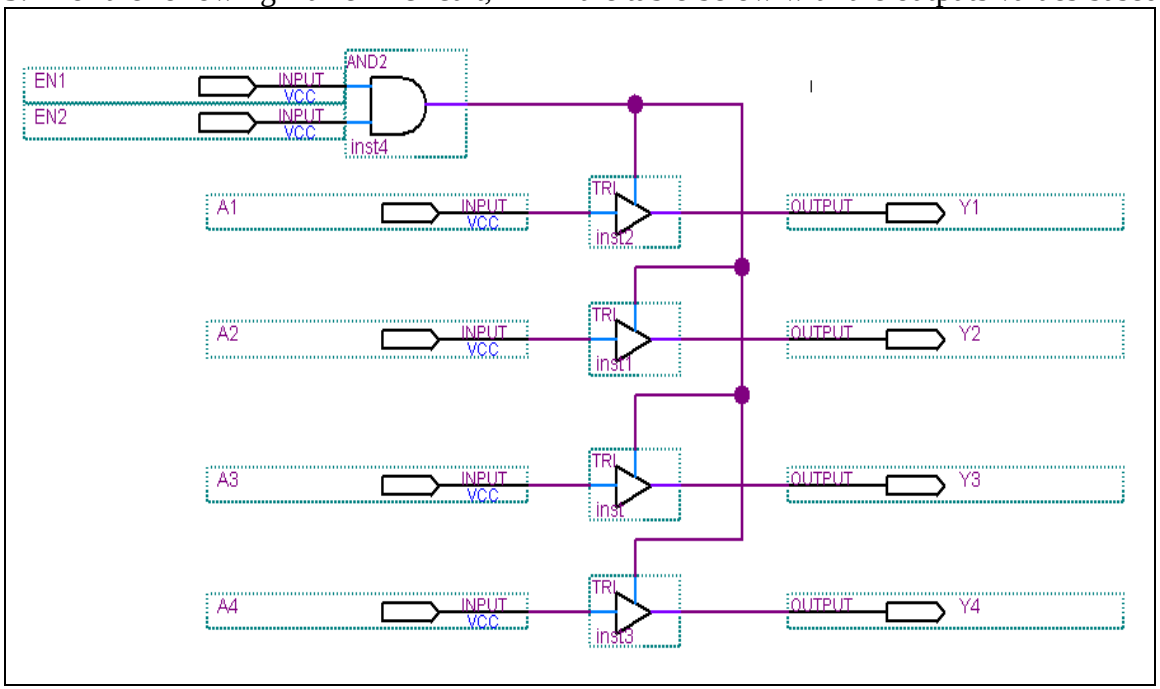
G1N	G2N	D	C	B	A	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	1	1	0	0													**0**			
0	0	0	1	0	1						**0**										
0	0	1	0	1	0											**0**					
0	0	0	0	1	1				**0**												
0	0	1	1	1	1																**0**
0	0	1	1	1	0															**0**	
0	0	0	0	0	0	**0**															

2. For the following Register-4 circuit, fill in the table below with the outputs values based on the inputs.



CLK	IE	CLR	A1	A2	A3	A4	Y1	Y2	Y3	Y4
0→1	1	0	1	1	0	0				
0→1	1	0	0	1	0	1				
0→1	1	0	1	0	1	0				
0→1	1	0	0	0	1	1				
0→1	1	0	1	1	1	1				
0→1	1	0	1	1	1	0				
0→1	1	0	0	0	1	1				
0→1	0	0	0	1	1	1				

3. For the following Buffer-4 circuit, fill in the table below with the outputs values based on the inputs.



EN1	EN2	A1	A2	A3	A4	Y1	Y2	Y3	Y4
0	0	1	1	0	0				
1	0	0	1	0	1				
0	1	1	0	1	0				
1	1	0	0	1	1				
1	1	1	1	1	1				
1	1	1	1	1	0				
1	1	0	0	1	1				
1	1	0	1	1	1				

4. For the **Brainless Microprocessor**, fill up the following table that with the appropriate control lines and the appropriate functionality.

a. **OR Operation**

Clear	Clock	Decoder	ROM 1	ROM 2	A Only	Invert	Logic	Read	Write	ACC. IE	ACC. OE	RESULTS
0			**3**	**5**								
0			**3**	**5**								
0			**3**	**5**	**1**	**0**	**1**					
0			**3**	**5**								

b. **Subtract Operation**

Clear	Clock	Decoder	ROM 1	ROM 2	A Only	Invert	Logic	Read	Write	ACC. IE	ACC. OE	RESULTS
0			**9**	**6**								
0			**9**	**6**								
0			**9**	**6**	**1**	**1**	**0**					
0			**9**	**6**								

c. **NOT Operation:**

Note: Before passing any data to the ALU, set up its control for the NOT operation (1's complement). Aonly = 0, Invert = 1 and Arith/Logic = 1.

Clear	Clock	Decoder	ROM 1	A Only	Invert	Logic	Read	Write	ACC. IE	ACC. OE	RESULTS
0			**7**	**0**	**1**	**1**					
0			**7**								
0			**7**								
0			**7**	**0**	**1**	**1**					

5. After you have programmed the brainless microprocessor into the UP2 board, show the floor plans of our brainless microprocessor circuit targeting the EPM7128SLC84-CPLD.