Lab 2: Arithmetic and Logic Circuits

Summary:
Students will learn how to create and test the components of an ALU's arithmetic unit.

Learning Objectives:

- Learn basic logic structures used in arithmetic operations
- Experience designing the building blocks of an arithmetic-logic unit
- Gain experience running simulations with *Virtuoso*

Resources and Supplies:

- Engineering PC with PuTTY and Xming
- *Cadence Virtuoso Setup Guide**
- *Cadence Virtuoso Logic Gates Tutorial**
- *simulation_skeleton.txt**
- *Lab2A.txt & Lab2B.txt**

All documents* are available on the class website

Important Reminders:

- Eye protection is required for admittance to the lab. See Lab 1 for details.
- Pre-lab assignments must be completed before coming to the lab.

Pre-lab Assignment:
Each student must complete his/her own pre-lab before coming to the lab. Pre-lab check-off sheets must be turned in to the TA before starting the lab assignment.

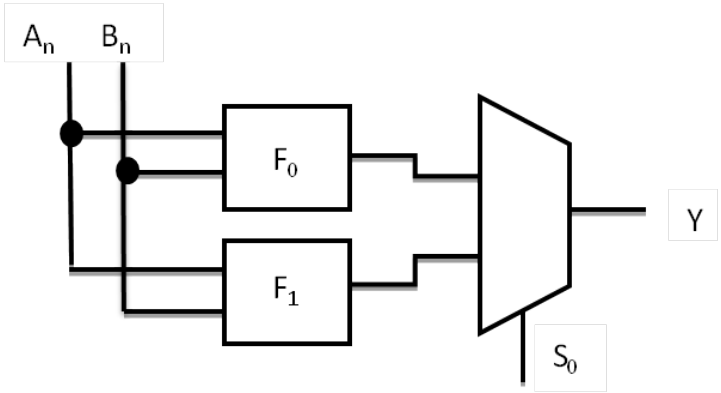
1. Read the **Important Reminders** and **Background** sections of this lab.
2. Print the Pre-lab 2 Check-off Sheet near the end of this document and perform the required tasks. Turn in the completed sheet to the TA at the beginning of your lab.
3. Read through the entire **Laboratory Assignment** section so you know what to expect in the lab.

Background:
Design of an Arithmetic-Logic Unit:

In Lab 1, the basic library of logic gates (ECE331 library) was expanded to provide a larger set of logic functions that can now be used to construct more complex digital circuit functions. In fact, with the logic gates you now have available in your libraries, you could implement any digital function. For example, over the next two labs, these logic gates will be applied to the design of a limited-function 8-bit arithmetic-logic unit (ALU). The 8-bit ALU will be constructed in single-bit modules called a *bit slice*. Each bit slice will contain a 1-bit arithmetic block and a 1-bit logic block, both composed of the basic logic gates you were given or built in Lab 1. Designing an ALU should give you very good insight into the architecture and operation of a basic ALU, which is a key component of any microcontroller/microprocessor.

The general structure of the ALU bit slice is shown in Figure 1. Two inputs, A and B, go into two functional blocks, F_0 and F_1. The outputs of each functional block are input to a 2:1

MUX where the control signal S_0 select which function is sent to the output Y. With this architecture, if F_0 were a 1-bit logic block and F_1 were a 1-bit arithmetic block, then S_0 would determine if the logic or arithmetic result was routed to Y. In this lab, you will be constructing a 1-bit logic block and a 1-bit arithmetic block. Each of these blocks will generate multiple functions that can be selected to a single output. In the next lab, you will put these blocks together to form the bit slice and then cascade 8 of the 1-bit slices to implement an 8-bit ALU.



S_0	Y
0	F_0
1	F_1

Figure 1: Bit slice architecture that routes the 1-bit results from two functional blocks into a single output.

Multifunction Logic Block:

In Lab 1, you worked with digital circuits that can implement the following 1-bit Boolean functions: INV, NAND, NOR, AND, OR, and XOR. Each of these logic functions were implemented within an individual circuit. For an ALU, it is desirable to bring several of these functions together so that you can select from multiple logic operations. However, the ALU has a single output, so the logic module of an ALU bit slice must perform multiple logic operations and then allow only one, selectable, result to pass to the output. This can be achieved using a MUX to select from multiple input sources similar to the way the MUX in Figure 1 selects one output between two possible inputs. The more logic functions the logic module generates, the larger the MUX would need to be and the more control bits would be needed. For example, four inputs can be down-selected by a 4:1 MUX that is controlled by two select bits. In general, a MUX with *n* select signals can select between up to 2^n inputs.

Multifunction Arithmetic Block:

Full Adder:

A full adder is a digital circuit that will add three binary input bits, A, B, and Cin. Because three single bits are added, the output requires two bits to express all possible values (0-3_{10} or 00_2-11_2), typically referred to as the sum, S, (LSB) and carry out, Cout, (MSB) bits. To implement a multi-bit addition, Cout of the least significant bit (LSB) can be cascaded to Cin of the next highest bit, and so forth, forming a standard ripple-carry adder.

The gate-level schematic for a typical full adder is shown in Figure 2. It is composed of two XOR gates, two AND gates, and a NOR gate. The longest delay (signal propagation delay) through this cell is from A or B to Cout. Thus, the speed of a ripple carry adder is limited by how many bits the Cout signal must cascade through.

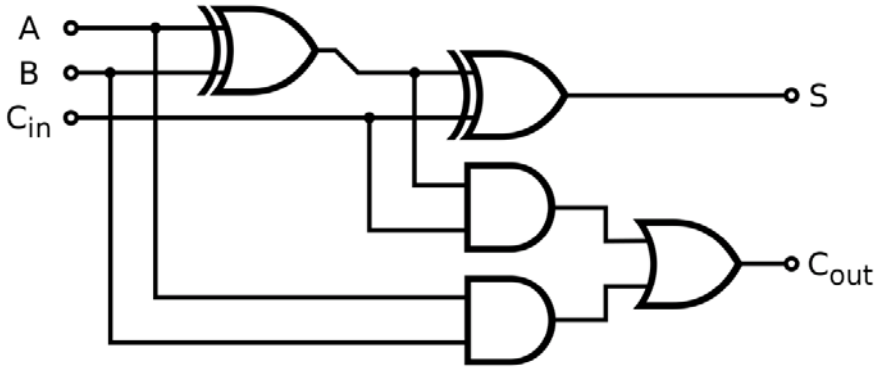
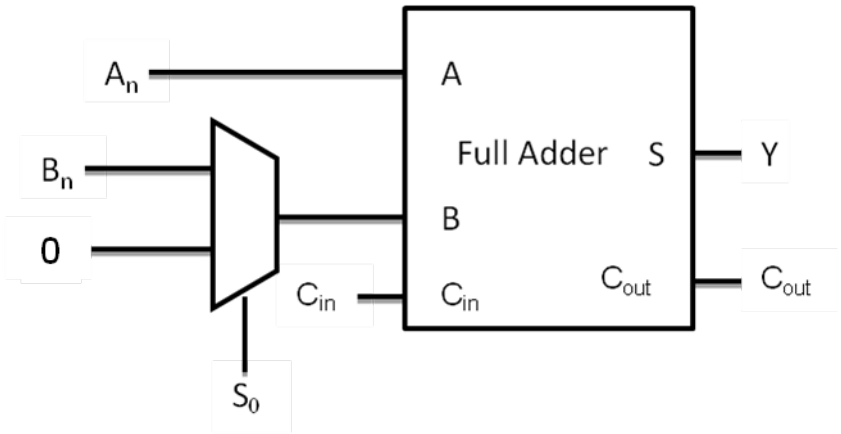


Figure 2: Typical schematic for a full adder.

Implementing Multiple Arithmetic Functions:

Although the full adder physically performs an addition operation, subtraction can also be implemented by using 2's complement notation. That is, A – B = A + [B]*, where [x]* means the 2's complement of x. Thus, **A – B = A + B' +1**, so by inverting B and setting Cin = 1, the full adder can also implement A-B (subtract). Furthermore, if the B input were set to 0, a *Pass-A* function could be implemented. Similarly, if the B input were set to 0, the full adder circuit could implement an *Increment-A* function when Cin = 1. Thus, a single full adder cell could be used to generate multiple ALU functions by controlling one of the inputs. Figure 3 shows schematically how a circuit could be built to implement two functions, A+B and Pass A. Notice that unlike the Logic Block which uses a MUX at the output to select from multiple logic functions, the Arithmetic Block would use a MUX on one of the inputs in order to realize multiple functions.



S_0	Y
0	A+B (+Cin)
1	A (+Cin)

Figure 3: Two-function arithmetic circuit implementing the selectable functions shown in the table.

Stimulus Files:

To run circuit simulations in *Virtuoso*, input signals (including DC voltage sources) are needed. Unlike SPICE, where you include the sources within the circuit netlist, *Virtuoso* separates the circuit from the sources. This allows you to construct circuits by instantiating other circuits that do not include unnecessary sources. To define the sources, *Virtuoso* uses a **stimulus file** to tell the simulator what all the input sources (constant, or variable) should be. For digital simulations, sources are typically either DC (like power supply or a

constant input) or signals that alternate from 1 to 0 over time. Although it is possible to generate a source with arbitrary digital values (e.g., 1 0 1 1 1 0 0 1), it is often more simple to represent variable inputs by a periodic digital waveform. The *Virtuoso* simulator (called *spectre*) uses a periodic "pulse" source defined by a width (length of time the signal is high, after which it falls to low) and period (length of time before it goes high again). For example, if the period is twice the width, a square wave with 50% duty cycle would be generated. Other parameters needed to define a pulse source are illustrated in Figure 4.

All sources defined within the stimulus file must follow a specific syntax. Refer back to the *Cadence Virtuoso Logic Gates Tutorial* for more information and an example stimulus file.

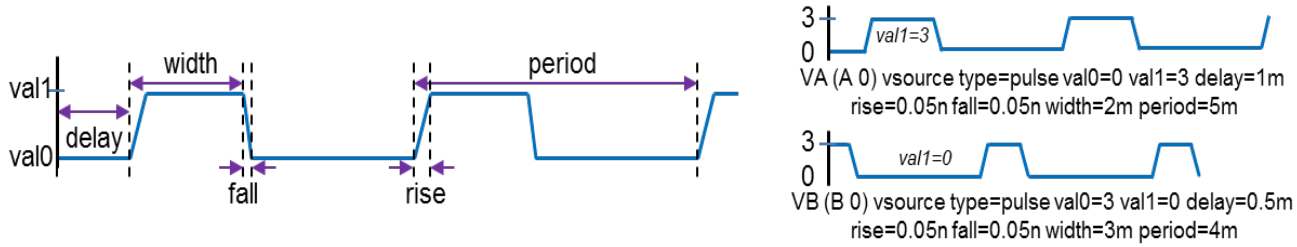


Figure 4. Illustration of pulse waveform parameters (left) and two example pulse definitions (right).

Laboratory Assignment:

This lab consists of two parts. A check-off sheet is included at the end of this lab document to record your design work and results. After you successfully finish each part of the lab, show the TA your results and ask him/her to sign the check-off sheet.

- Print the Lab 2 Check-off Sheet.

Part 1: Logic Block

In Part 1 you will create the Logic Block for a 1-bit ALU bit slice.

Design

1. Prepare a plan for a 2-input Logic Block circuit that meets the following requirements:
   - The Logic Block must implement four logic functions, AND, OR, XOR, and INV using only existing digital cells (in the ECE331 library and your project library).
   - The circuit must allow one of these functions to be selected (via selection input signals) as the output of the Logic Block.
   - The inputs should be named **A** and **B**, and the output should be **Y**. The select signals should be named **S0**, **S1**, etc. depending on how many you need, with S0 being the lowest (least significant) bit, i.e. Select = 1 in the table below means S_1S_0 = 0 1, Select = 2 means S_1S_0 = 1 0, etc.

   Sketch a diagram of your Logic Block in the designated area of the check-off sheet. Refer to the Background section for hints. In order to keep the function selection code the same for all students, please assign function selection according to the table below. (Note: this selection code will become part of the op-code for your 8-bit ALU).

Select	*Function*		*Select*	*Function*
0	INV A		2	OR
1	XOR		3	AND

If you have any questions about your Logic Block design, please consult the TA and ensure you have a correct design before continuing.

Implementation in Cadence Virtuoso

2. Start Cadence *Virtuoso* in your ECE331/virtuoso directory. Refer to the *Cadence Virtuoso Setup Guide* and Cadence Virtuoso Logic Gates Tutorial if you run into problems.
3. Select your project library and create a new schematic cellview for the Logic Block. Name the circuit **LogicBlk**.

"i" = add instance

4. Using your sketched Logic Block plan, implement the circuit in Virtuoso by instantiating gates from the ECE331 library and your project library (generated in Lab 1).

"w" = add wire

5. Wire the gates together using the wiring tool. Also add wires to the external input and output nodes so I/O pins can be attached.

"p" = add pin

6. Add I/O pins for all external inputs and outputs. Make sure the inputs are input pins and the output is an output pin. Use the signal names defined in step 1.
7. Check-and-Save the schematic cellview and fix any errors or warnings.
8. Create a symbol for the LogicBlk schematic (Create->from cellview…->symbol). A rectangular shape is fine. Generally, inputs are placed on the left and outputs on the right. You can move the select signals to the bottom or leave them on the left but separate them from the A & B inputs. Your future schematics will become messy if the pins are unorganized (e.g., if the default puts S0 between A and B).
9. Check and Save the symbol, then exit the symbol tool.

Demonstration

10. For any given set of inputs, your LogicBlk circuit should generate a known output value. Based on your design, fill in the expected output (**Y**) column of the partial function table in the check-off sheet. Although this does not show all input combinations, it provides a sample of outputs from each logic function.
11. In *Virtuoso*, run a simulation of your LogicBlk schematic using the Lab2A.txt stimulus file available from the class website. Note, you must assign logic functions to selection bits as defined in Step 1 or this stimulus file will not work correctly. You may need to adjust the Stop Time in the Analysis setup to ensure all desired input combinations are simulated.
12. Observe the simulation waveforms and identify the output value for each of the input combinations on the check-off sheet function table. For each output that matches your expected Y value, place a checkmark in the last column. If you find any errors, search for the cause and fix it.
13. Print the simulation output waveform to include in your lab report. Be sure all signals are clearly identifiable and the plot is clearly labeled and titled.

14. Show the TA your LogicBlk schematic, symbol, and simulation results along with the plan sketch and function table in the check-off sheet. Ask the TA to sign your check-off sheet.

Part 2: Arithmetic Block

In Part 2 you will create the Arithmetic Block for a 1-bit ALU bit slice.

Design

1. Prepare a plan for a 2-input Arithmetic Block circuit that meets the following requirements:

   - The Arithmetic Block must implement the following four functions using only existing digital cells (in the ECE331 library and your project library), including the full adder from Pre-lab 2.

     Add (Y=A+B), Subtract (Y=A-B), Increment A (Y=A+1) and Pass A (Y=A)
     **Increment must be implemented using Cin=1, not B=1, as in the table below.*

   - The circuit must allow any one of these functions to be selected (via selection input signals) as the output of the Arithmetic Block.

   - The inputs should be named **A**, **B** and **Cin**, and the outputs should be **S** and **Cout**. The select signals should be named **S0**, **S1**, etc. depending on how many you need, with S0 being the lowest (least significant) bit, i.e., Select = 1 in the table below means S_1S_0 = 0 1, Select = 2 means S_1S_0 = 1 0, etc.

   Sketch a diagram of your Arithmetic Block in the of the check-off sheet. Refer to the Background section for hints. In order to keep the function selection code the same for all students, please assign function selection according to the table below. (Note: this selection code will become part of the op-code for your 8-bit ALU).

Select	*Function*		*Select*	*Function*
0	ADD		2	SUBT
1	PASS A		3	INC A

   Every operation in the Arithmetic circuit should use the full adder so that only one output comes from this circuit. Thus, to implement multiple functions you will need to manipulate the full adder inputs (specifically the B input to the adder) with selection signals. This can be a bit tricky to do correctly, so to simplify this design step, refer to the table below that shows what each full adder input (here named fa_A, fa_B, fa_Cin) should be, in terms of Arithmetic Block inputs A and B, for each function. Note that fa_A is always just A and fa_Cin is a binary input that can be chosen as needed for each function. The adder input fa_B is where you must add circuitry that will override or manipulate the B signal before it is input to the full adder.

full adder inputs	***fa_A***	***fa_B***	***fa_Cin***
ADD A+B	A	B	0
PASS A	A	0	0
SUBT A-B	A	B' (inv B)	1
INC A	A	0	1

If you have any questions about your Arithmetic Block design, please consult the TA and ensure you have a correct design before continuing. However, the TA will not tell you the answer so you must have some design sketched before asking for help.

Implementation in Cadence Virtuoso

2. Repeat Part 1 steps 2-9 to implement a schematic and symbol for the Arithmetic Block. Name the schematic **ArithBlk**. If you design this circuit correctly, you will need to a ground symbol in your schematic. You can get a ground symbol from the library ***NCSU_Analog_Parts***; select the subfolder ***Supply_Nets*** then choose the **gnd** cell.

Demonstration

3. For any given set of inputs, your ArithBlk circuit should generate a known output value. Based on our design, fill in the expected output columns (**S**, **Cout**) of the partial function table in the check-off sheet. Although this table does not show all input combinations, it provides a sample output for each function.
4. In *Virtuoso*, run a simulation of your ArithBlk schematic using the Lab2B.txt stimulus file available from the class website. Note, you must assign functions to selection bits as defined in Step 1 or this stimulus file will not work correctly. You may need to adjust the Stop Time in the Analysis setup to ensure all desired input combinations are simulated.
5. Observe the simulation waveforms and identify the output values for each of the input combinations on the check-off sheet function table. For each output that matches your expected S and Cout values, place a checkmark in the last column. If you find any errors, search for the cause and fix it.
6. Print the simulation output waveform to include in your lab report. Be sure all signals are clearly identifiable and the plot is clearly labeled and titled.
7. Show the TA your ArithBlk schematic, symbol, and simulation results and the check-off sheet plan sketch and function table. Ask the TA to sign your check-off sheet.

Wrap Up

Clean up your lab bench before you exit the lab.

Remember, your Lab 2 report will be due in lab next week. Each student must submit his/her own lab report. Include your check-off sheet and waveform printouts with your report. You may want to review the Discussion Points below and talk to the TA about anything that is not clear to you.

Discussion Points

Address these discussion points in a designated section of your report.

1. Why are rise and a fall times included in the stimulus files? How does this relate to how an actual circuit would work?
2. Explain the difference between a half adder and a full adder. What are the advantages and disadvantages of using a half adder over a full adder?
3. How would you implement a decrement function for one of the Arithmetic Block functions? This may be difficult to envision for a single bit, so try thinking about an 8-bit cell. How could the adder inputs be manipulated to generate A – 1 (decrement)?

PRE-LAB 2 CHECK-OFF SHEET
Due: At the beginning of lab.

Student Name: _____ **Lab. Section (time):** _____

For 2 Pre-lab 2, you will create a full adder cell in Cadence Virtuoso and test it with a stimulus file to ensure correct operation. Refer back to the *Guide* or *Tutorial* if you run into trouble starting or using *Virtuoso*.

1. Start *Virtuoso* from your ECE331/virtuoso directory and create a new schematic cellview called FullAdder.

2. Within the FullAdder schematic, implement the design shown in Figure 2 of the Background section by instantiating gates from the basic ECE331 library and your project library. Define the inputs to be A, B and Cin and the outputs to be S and Cout. Make sure you assign the correct name to the correct signal!

3. Check and save the schematic. Then create a symbol and save it. A square or rectangle shape will be fine (there is no unique shape typical for a full adder).

4. Fill out the *Expected* columns of the function table below to show what each output of a full adder should be for each of the given input combinations. Leave the *Observed* column blank for now.

			Expected		***Observed***	
C_{in}	**A**	**B**	**S**	**Cout**	**S**	**C_{out}**
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

5. Use the grid below to draw out waveforms for each of the signals in the full adder function table above. Draw the signals so that each column in the grid represents one row of the table above. This should create waveforms for the inputs that transition from one input combination (e.g., 101) to the next combination (e.g., 110) every millisecond. For example, in the first column of the grid, A, B, and Cin will be 0. In the second column, B will transition to 1 while A and Cin remain 0. Be sure to include the expected outputs also.

B										
A										
C_{in}										
S										
C_{out}										
time	1m	2m	3m	4m	5m	6m	7m	8m	9m	10m

6. In the grid above, you should notice a periodic behavior for the input signals. Create a complete full adder stimulus file which includes pulse voltage sources that will produce the input signal waveforms matching your grid above. To help you get started, you can download a skeleton stimulus file (*stimulus_skeleton.txt*) from the class website or open a stimulus file from Lab 1 and edit it. After editing the file to produce the full adder input signals (be sure the source names match your schematic pin names!), save the file as ***FAtest.txt***.

   Notice, you can use the *delay* parameter to create an offset at the beginning of the periodic signal. Although you don't *need* a delay here, it can be very useful to generate a state (logic combination) not easily achieved with periodic waveforms.

7. Simulate the full adder schematic with the stimulus file *FAtest.txt* selected. Be sure to set the Stop Time in the Analysis setup to a value that allows all desired input combinations to be simulated (probably 10m but depends on how you define your pulse sources). If you have trouble with running the simulation, refer back to the *Cadence Virtuoso Logic Gates Tutorial* document. Try to arrange your plot waveforms to match the grid above for easy comparison.

8. Record the output values for each input combination in the *Observed* columns of the table above. If they don't match, double-check your *Expected* values and review your schematic to find the problem.

9. After confirming the simulation results, print the full adder simulation waveform and turn it in with Pre-lab 2.

At the beginning of lab, turn in this Pre-lab 2 Check-off Sheet to the TA along with your full adder simulation plot. You should not need these items during your lab. If you do, ask the TA to sign below indicating they have seen your completed pre-lab, then keep this sheet and include it within your lab report.

(if needed) TA initials _____

LAB 2 CHECK-OFF SHEET

Student Name: _____ **Lab. Section (time):** _____

Part 1: Logic Block

Step 1. Sketch of Logic Block design

Step 10. Partial Logic Block function table

A	B	S_1	S_0	Y	√
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		

Part 1:TA sign off Initial_____

LAB 2 CHECK-OFF SHEET

Student Name: _____ **Lab. Section (time):** _____

Part 2: Arithmetic Block

Step 1. Sketch of Arithmetic Block design

Step 3. Partial Arithmetic Block function table

	A	**B**	**S_1**	**S_0**	**C_{in}**	**S**	**C_{out}**	√
Add	1	0	0	0	0			
Pass A	1	0	0	1	0			
Subt A-B	1	0	1	0	1			
INC A	1	0	1	1	1			

Part 2: TA sign off Initial_____