SCHULICH
School of Engineering
UNIVERSITY OF CALGARY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

ENEL 353 -- *Digital Circuits*

Laboratory #2

Design and CPLD Implementation of Digital Arithmetic Circuits

1 Introduction

Thanks to the steady evolution of software and programmable hardware technologies in past years, Programmable Logic Devices (PLDs) are now indispensable tools for digital circuit designers. There are many clear advantages to using PLDs over the direct physical interconnection of discrete 7400-series (or other technology) logic gates, including the considerable comparative ease in which relatively complex circuits can be designed, implemented, tested, and debugged. Most modern devices can also be reprogrammed without fuss when design adjustments are necessary. In this and the remaining lab sessions, we now focus on PLD implementations of digital circuits. Here, you will implement some digital arithmetic circuits, including a 4-bit adder/subtracter, and a 3×3 multiplier.

The purpose of this lab experiment is as follows:

1. to design and build two important combinational circuits on a PLD: a digital adder/subtracter circuit, and a digital multiplier;

2. to become acquainted with a state-of-the-art computer-aided design environment called *Quartus* from Intel Corporation, a leading manufacturer of Complex PLDs (CPLDs) and Field-Programmable Gate Arrays (FPGAs). Using *Quartus*, you will enter your circuits via the *Quartus* schematic editing tool, then

download and test the arithmetic circuit designs onto a custom-built CPLD evaluation system, developed specially for this course by our technical staff in the Department of Electrical and Computer Engineering.

2 Digital Arithmetic Circuits

Digital circuits that perform basic binary arithmetic, such as addition, subtraction, and multiplication, are vital components in every modern digital electronic system. Such circuits are integral parts of general-purpose PC processors, embedded systems, hand-held scientific calculators, high-performance graphics engines, hand-held or mobile GPS systems, smartphones, and high-speed special-purpose digital signal processors used in the telecommunications and consumer electronics industries. When digital circuit technology was in its infancy, multipliers in particular were complex, power-hungry, and very expensive. Digital circuit designers often invented clever ways to *avoid* using them. Digital arithmetic, even multiplication, can now be performed with relative ease and very low cost.

2.1 Addition and Subtraction

2.1.1 Addition

The input-output diagram for a four-bit adder is shown in Fig. 1. The system has two sets of four inputs corresponding to the four-bit augend $E = e_3e_2e_1e_0$ and the four-bit addend $F = f_3f_2f_1f_0$, where e_3 and f_3 are the most-significant bits (MSBs) of E and F, respectively. There is also a one-bit carry-in signal C_{in}. The output is the four-bit *unsigned* sum $Y = y_3y_2y_1y_0$ given by

$$Y = E + F + C_{in}. \tag{1}$$

The output C_{out} is the carry-out from the MSB. As discussed in lectures, the same method of binary addition is used to add to two *unsigned* numbers as as well as two *signed* numbers in two's-complement format.

Recall the one-bit full adder tested in Lab #1. A straightforward method to design a four-bit adder is by using four of these full adders in a circuit called a *ripple-carry adder*, as shown in Fig. 2.

The operation of the ripple-carry adder closely mimics the operations performed in calculating the sum by hand. The least-significant bit (LSB) of the sum y_0 and the associated carry bit c_1 become valid first. The next full adder, however, must wait for this value of c_1 to become valid before y_1 and c_2 can be determined. Similarly, the next full adder must wait for c_2 to become valid before y_2 and c_3 can be determined. Thus, the outputs of each full adder in Fig. 2 wait on all full adders to their right. This is the "ripple-carry" effect.

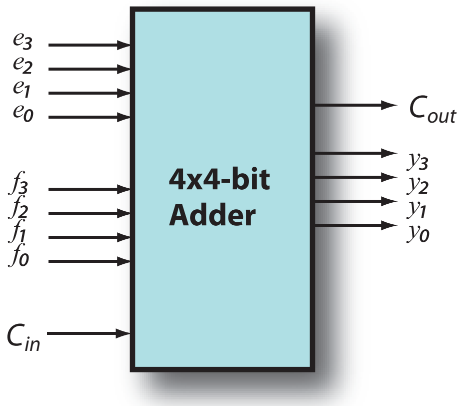


Fig. 1. A 4-bit binary adder

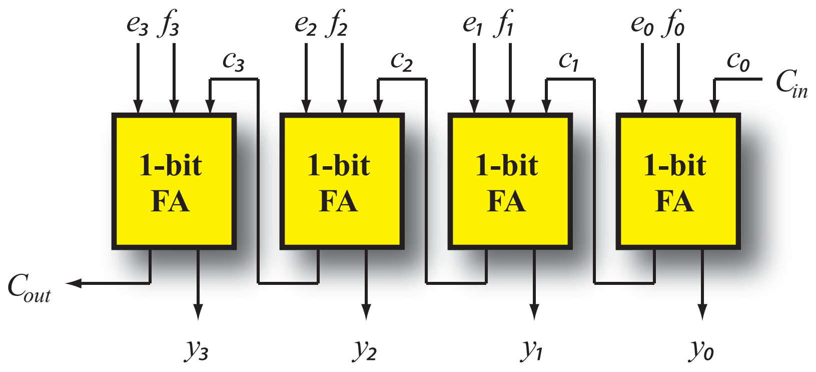


Fig. 2. A four-bit ripple-carry adder using full adders (FAs)

Such a four-bit adder can then be used as a building block in place of the one-bit adders to build larger adders in the same ripple-carry configuration as in Fig. 2. In this experiment, however, we will not be needing anything larger than a four-bit adder. For addition, we simply set $C_{in} = 0$.

2.1.2 Subtraction

The same ripple-carry adder circuit can be used to perform a four-bit binary subtraction with only a slight modification. The calculation of $Y = E - F$ is performed by the equivalent calculation

$$Y = E + (-F); \tag{2}$$

that is, we use the *two's-complement negative* of F as input to the four-bit adder in place of F. Recall that $(-F)$ is determined by the two-step process of first complementing all the bits in F, and then adding 1 to result. The first step is straightforward: simply insert a NOT gate on each of f_i before it enters the adder. The second step is to add one to the calculation, which is conveniently provided via the C_{in} input by simply setting $C_{in} = 1$.

2.2 The Multiplier

The multiplier is somewhat more challenging than the adder circuit above, but we may similarly mimic the steps performed in multiplying numbers by hand. The input-output diagram is shown in Fig. 3. The system has two sets of three inputs corresponding to the 3-bit *unsigned* multiplicand and the 3-bit *unsigned* multiplier. The output of the multiplier is the 6-bit *unsigned* product given by

$$Y = E \times F, \tag{3}$$

where $Y = y_5 y_5 y_3 ... y_0$ (y_5 is the MSB).

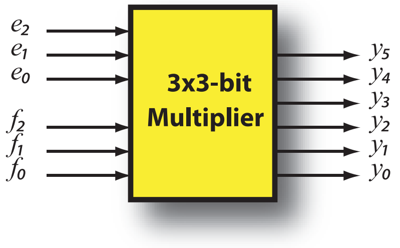


Fig. 3. A 3x3-bit digital multiplier

A straightforward approach to multiplying two binary numbers is to employ the same method that we learned for decimal numbers when first taught multiplication in grade school. For example, let $E = 7_{10}$ and $F = 5_{10}$, shown in Fig. 4 as binary numbers, and multiply them as described below.

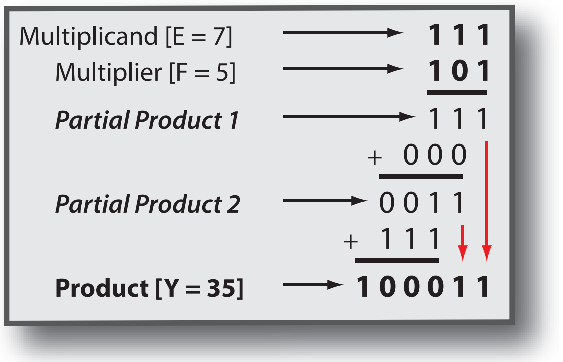


Fig. 4. Partial-Product method for multiplying binary numbers

The product Y can be determined using a sequence of *partial product* calculations. Starting with bit f_0 in the multiplier F, the first partial product is simply the product of f_0 with the multiplicand E. Thus, if $f_0 = 1$, the first partial product is simply E, otherwise it is 0. The second partial product is then formed by multiplying E by f_1, left-shifting the result by one bit, and adding it to the first partial product. Finally the complete product Y is formed by multiplying E by f_2, left-shifting the result by two bits, and adding it to the second partial product.

2.3 Comments on VHDL Coding and Complex Circuits

In this lab, all circuits will be entered using *Quartus*' schematic editor tool. The circuits are all reasonably simple, requiring at most nine two-input AND gates, two 4-bit full-adder modules, plus all required input and output pins. However, for larger circuits, such as a 4x4-bit multiplier, or *much worse*, say a 16x16-bit multiplier, circuit complexity quickly becomes completely unmanageable when entering a schematic this way.

One very important way of programming larger digital circuits onto PLDs is by using a *hardware-description language* such as the industry-standard VHDL language. VHDL is an acronym for VHSIC Hardware Description Language, where VHSIC is itself an acronym for Very High-Speed Integrated Circuit. The language was originally developed for the United States Department of Defence in 1985. Another similar and popular hardware-description language is *System Verilog*, and both these languages enjoy widespread use.

Although VHDL has been employed to a very limited extent in previous ENEL 353 labs, it is no longer used in the course. Students in our Electrical Engineering Program will explore VHDL in detail in their third year with the course ENEL 453 (Digital System Design).

3 Pre-Lab Preparation

3.1 Plan the 4-Bit Adder/Subtracter

In this experiment, you will use a pre-built 4-bit full adder module as supplied in the *Quartus* device library. This module differs structurally from the ripple-carry adder in Fig. 2, but provides exactly the same functionality. Once retrieved from the device library and placed in your schematic, the device will be appear as shown in Fig. 5.

As shown, there are 4 inputs labeled `A4`, `A3`, `A2`, and `A1`, and 4 more inputs labeled `B4`, `B3`, `B2`, and `B1`. These correspond to the e_3, e_2, e_1, e_0 inputs and f_3, f_2, f_1, f_0, respectively, in Fig. 2. Similarly, `C0` and `C4` are equivalent to C_{in} and C_{out}, respectively, and the output sum bits `S4`, `S3`, `S2`, `S1` match up with y_3, y_2, y_1, y_0.

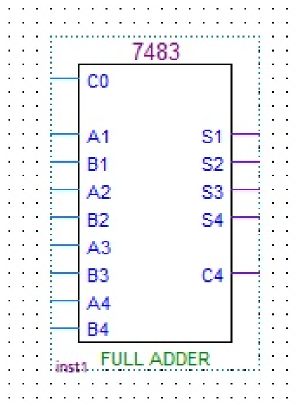


Fig. 5. The Quartus-supplied "7483" 4-bit full-adder module

Using this 4-bit full-adder module, sketch a 4-bit adder/subtracter that is switchable between the calculation of either *Y=E+F* or *Y=E+(-F)*. This will involve providing either F or its two's-complement negative $(-F)$ to the adder module.

Recall in Section 2.1.2 that subtraction is performed by using NOT gates on each of the adder inputs f_i, and by setting $C_{in} = 1$. Here, replace the four NOT gates with four two-input XOR gates. Each XOR gate serves to deliver an input bit in either its true form or its complementary form, as shown in Fig. 6. The C_{in} input is conveniently used to control this on each bit, while simultaneously providing the required addition of 1 to complete the two's complement negation. When $C_{in} = 1$, the calculation is *Y=E+(-F)* , and when $C_{in} = 0$, the calculation is *Y=E+F* .

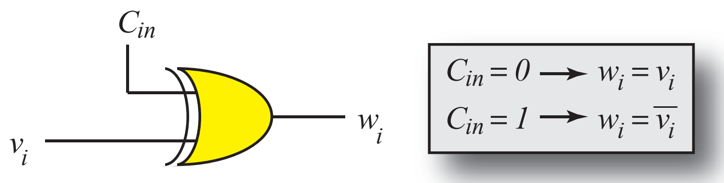

C_{in}
v_i
w_i
$C_{in} = 0 \longrightarrow w_i = v_i$
$C_{in} = 1 \longrightarrow w_i = \overline{v_i}$

Fig. 6. Using an XOR *gate to deliver a bit either in its true form or complementary form*

In the experimental work ahead, we will interpret the results in terms of the addition and subtraction of *unisgned* numbers.

3.2 Plan the Multiplier

Carefully study Fig. 4 and thereby try to make yourself comfortable with the above method of partial products. Try a couple of different numbers by hand and see if you can produce the right answers. You may also find helpful the brief description of multipliers on pp. 252-253 of the course textbook.

It is sensible to build the 3x3 multiplier in two stages. The first stage involves designing a 3x2 multiplier, which requires one 4-bit full-adder module and six AND gates. This is to produce the first partial product in Fig. 4.

1. Using three AND gates, produce the first partial product by multiplying $E = e_2e_1e_0$ by f_0. Assume that $e_3 = 0$, and make the resulting product one of the 4-bit inputs to the adder.

2. Similarly, form the second 4-bit input to the adder by multiplying $E = e_2e_1e_0$ by f_1, but *left-shifted by one bit* at the adder input bits. Use a zero for the LSB.

In the second stage, you will complete the 3x3 multiplier by building onto your 3x2 multiplier. For this, you will need three more AND gates and one more 4-bit full-adder module. Follow the process above for the 3x2 multiplier and Fig. 4 to produce the final 6-bit product Y.

3.3 Please Read These Instructions!

Much of the lab time will be devoted to becoming familiar with the Intel *Quartus* hardware and software environment. There are detailed step-by-step instructions ahead on setting up and using the system, and if you read through *the whole thing* ahead of time, it can greatly speed things up in the lab periods. This may seem quite a lot of reading, but it should be pretty easy going.

NOTE: *It is especially important to have your design work fully prepared in advance! The lab period is only three hours.*

4 Preparing the Hardware and Software

The *Quartus Prime Lite Edition* software (version 18.0.0) is installed on all of Windows-based computers in the department undergraduate labs. The same version can be downloaded for free from Intel's website at *fpgasoftware.intel.com* if you wish to use it on your own PC (Windows or Linux). The download is approximately 3.2 GB (Windows) with the Max-II device support. (Information and links for downloading the software are available on the ENEL 353 course website. A license is not required to operate this software.)

Each lab station will be supplied with the Intel/Altera MAX II "MicroKit System" (or simply "MKS") shown in Fig. 7. The system is constructed around a "MAX II Micro Kit" module that is commercially available from Intel and elsewhere for about $70.00. This module is "piggy-backed" onto a custom interface card developed by our department technical staff. A standard USB extension cable is also part of the system for connecting the MKS's USB connector to a USB port on the computer at your lab station.

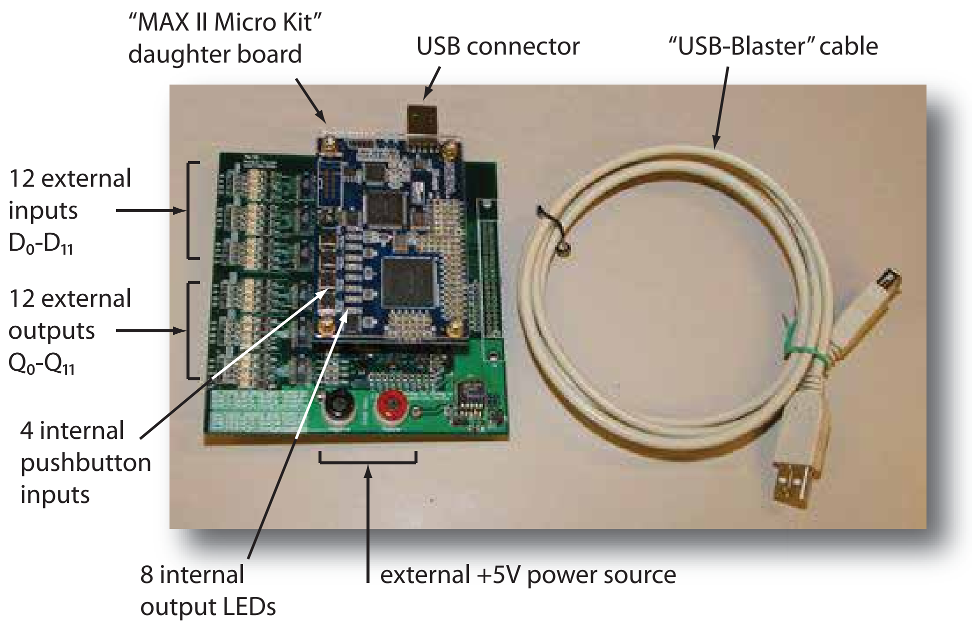

"MAX II Micro Kit" daughter board
USB connector
"USB-Blaster" cable
12 external inputs D_0-D_{11}
12 external outputs Q_0-Q_{11}
4 internal pushbutton inputs
8 internal output LEDs
external +5V power source

Fig. 7. The MAX II MicroKit System (MKS)

As indicated, the system has a number of conveniences built in. The Microkit module itself has four "internal" pushbuttons for use as inputs, and eight bright LEDs for outputs. The custom-built interface card that hosts the Microkit module provides 12 additional general-purpose inputs and 12 additional general-purpose outputs that you will use to connect to external devices. LEDs are installed adjacent to each pin on the interface card to indicate the state of the corresponding external inputs and outputs.

4.1 Wiring the MKS

The first step is to make all necessary electrical connections to the MKS, including the input and output signals, the power supply, and the "USB Blaster" programming cable. A typical installation suitable to carry out experiments in the course is shown in Fig. 8.

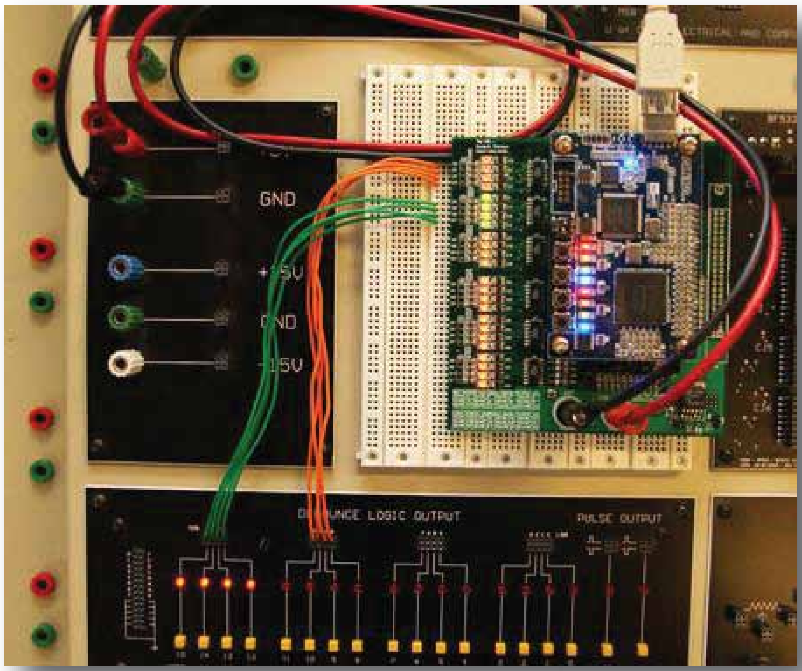


Fig. 8. Typical installation of the MKS for ENEL 353 experiments

On the laboratory breadboard at your station, insert the MKS onto the prototyping area as follows. Handle the unit *very carefully* so that you don't bend or damage any pins.

1. On the bottom of the MKS is a single column of pins that must be inserted into holes in the prototyping area. Pick a Type 1 socket strip on the prototyping area (i.e., a socket strip with *horizontal connectivity*). Note that using a Type 2 strip with vertical connectivity will simply short-circuit all of the pins! Carefully insert the pins of the MKS into the right-most of the five horizontally-connected holes on either half of the socket strip, such as where indicated in Fig. 9 and demonstrated in Fig. 8. This provides four holes in each row for wiring to each input/output pin. Press down gently but firmly so that the pins seat snugly in the holes on the socket strip.

2. Connect one end of the "USB Blaster" cable to the USB connector on the MKS and the other to a USB port on the front of the PC at your lab station. Many, if not all, of the LEDs on the MKS should illuminate.

Now make the following input and power-supply connections to the MKS.

3. Locate the pins on the MKS labeled D0, D1, D2, D3. Connect four wires from the holes in the prototyping area adjacent to these pins to switches on the

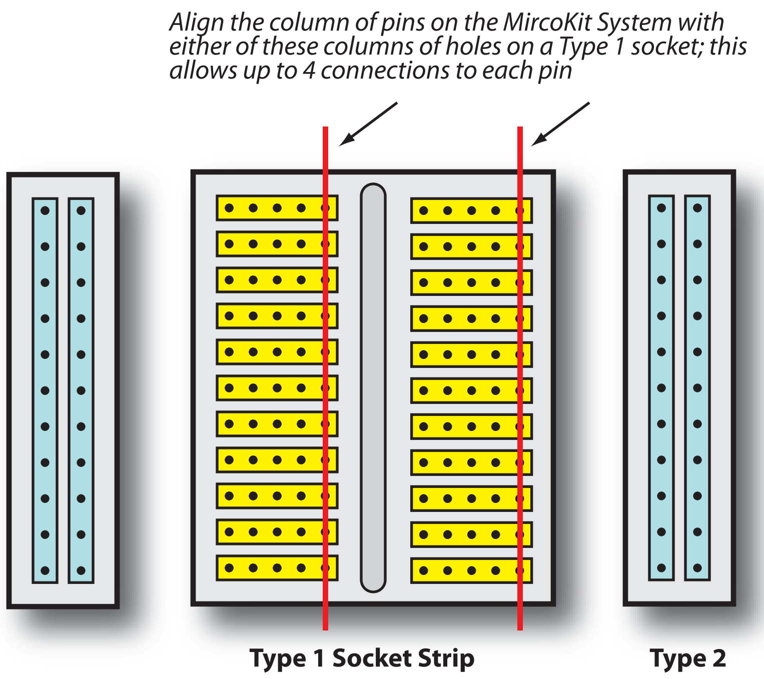

Align the column of pins on the MircoKit System with either of these columns of holes on a Type 1 socket; this allows up to 4 connections to each pin
Type 1 Socket Strip
Type 2

Fig. 9. Suitable locations for installing the MKS

breadboard. For Part 1 of the experiment, this is the 4-bit input E, with e_0 (the LSB) on pin D0. (These are shown as the orange wires in Fig. 8.)

4. Similarly, locate pins D4, D5, D6, D7 on the MKS, and connect four wires from holes adjacent to these pins to four other switches on the breadboard. This is the 4-bit input F, with f_0 (the LSB) on pin D4 (the green wires in Fig. 8).

5. Finally, connect pin D8 on the MKS to a ninth switch (not shown in Fig. 8).

TIP: *A sensible system of colour-coding the wires connecting the 8 inputs above could help in debugging a circuit. Although orange wires are used for all E inputs and green for the F inputs in Fig. 8, it might instead help to choose, say, orange for both e_0 and f_0, and then, say, green for both e_1 and f_1, etc.*

6. It is not necessary to connect anything to the output pins, labeled Q0-Q11 on the MKS. Instead, the state of the output bits can be observed by the colour of the built-in LEDs attached to each output pin; green indicates logic-1 and red indicates logic-0.

7. Connect a red cable from the red "+5V" power supply socket on the laboratory breadboard to the red socket on the MKS; similarly, connect a black cable from the "GND" socket on the breadboard to the black socket on the MKS. *Double-check these connections!*

4.2 A Note on Quartus Project-File Storage

You will need a place to store your *Quartus* project files. We recommend that you bring a USB "thumb drive" so that you can take your work with you when you leave the lab. Alternatively, you can create a folder locally on the **C:** drive of your lab PC under "This PC," but you can later only access those files from the same PC. If available on your lab PC, the network **H:** drive is an another alternative, providing access to your files from any lab PC where the **H:** drive is available.

4.3 Starting Quartus

Start *Quartus Prime Lite Edition* from the Windows Start button by selecting:

```
Intel FPGA 18.0.0.695 Lite Edition ⟶ Quartus (Quartus Prime 18.0)
```

If starting *Quartus* for the very first time with your U of C account, it may ask you what you wish to do with *Quartus*. Answer by selecting "Run the Quartus Prime software." *Quartus* will then exit. Restart it to begin using it.

5 Build and Test the 4-Bit Adder/Subtracter

The experiment involves using *Quartus* to enter the schematic diagram for the whole circuit, and then to compile, download, and test it on the MKS. *Carefully follow* the detailed step-by-step procedure given below.

5.1 Creating the Quartus Project

A *Quartus* welcome screen is displayed to allow you to create a new project or open an existing one. New projects are created very easily through the *Quartus* New Project Wizard. Click on "Create a New Project." (Alternatively, from the menu bar, select `File` ⟶ `New Project Wizard`). There are seven really quick set-up panels.

1. *Panel 1 – Introduction.* Press Next.
2. *Panel 2 – Directory, Name, Top-Level Entity.* Navigate to your newly created folder. In the box specifying the project's name, type the name "addsub." Press Next.

3. *Panel 3 – Project Type.* Select "Empty project," and hit Next.
4. *Panel 4 – Add Files.* There are no files to add, so press Next.
5. *Panel 5 –Family & Device & Board Settings.* In the box labeled "Device Family", select the MAXII series; then, in the box labeled "Devices" beneath, select "All". Near the end of the list (fourth-last) of available devices, locate and select the `EPM2210F324C3` device, and press Next.
6. *Panel 6 – EDA Tool Settings.* Press Next.
7. *Panel 7 – Summary.* if your changes look OK, hit Finish.

To begin, create a blank schematic diagram window by selecting the menu item `File` ⟶ `New`. In the list of categories that come up, choose "Block Diagram/Schematic File" under the "Design Files" category, and press OK. A blank schematic diagram editor window will appear with the title "Block1.bdf." Give this a name by selecting `File` ⟶ `Save As`. The default name that *Quartus* will offer is "addsub.bdf," which is automatically formed by appending the bdf (block diagram file) suffix to the project name. Make sure that the checkbox at the very bottom of the panel is checked to add this file to the project. Accept this default name and press SAVE.

5.2 Entering the Adder/Subtracter Circuit

5.2.1 Quick Overview

Your Block Editor window should now have the title "addsub.bdf," in which you may now draw your schematic diagram. Fig. 10 is a screenshot of what a finished circuit might look like (note that this is a 4-bit adder circuit, *not* the adder/subtracter circuit that you must enter).

Quartus has a considerable library of devices and objects that you may place in your circuit. It also provides a familiar and intuitive interface for cutting and pasting devices, or whole portions of your circuit. Once you've completed entering your schematic, you will "compile" your circuit, and then assign your inputs and outputs to physical pin numbers on the CPLD device using *Quartus'* "Pin Planner." Finally, you will use the *Quartus* Programmer to program the CPLD for testing.

5.2.2 Procedure in Detail

The toolbar at the top of the Block Editor window in Fig. 10 contains all the tools you will need to draw your circuits. The most-commonly used tools are shown in Fig. 11 with a brief description of each. After placing a couple of devices and making several connections, you should get the hang of it pretty quickly!

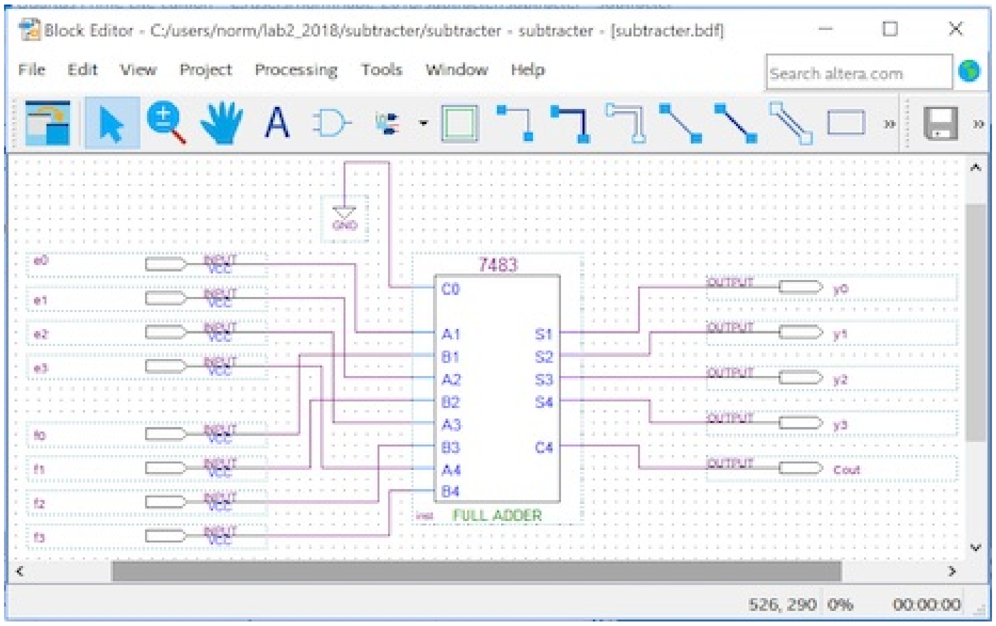


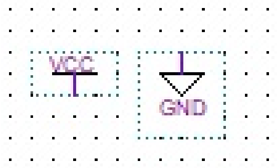
Fig. 10. A sample screenshot of the Block Editor window, showing a complete 4-bit binary adder circuit

Placing devices in your circuit

First select a device that you wish to place in your circuit, perhaps starting with the 4-bit adder module. Click the **Selection Tool**, and expand the device folder in the Symbol panel that comes up. Further expand the subfolder named "others," and then "Maxplus2". Scroll down the large list of devices available, and click on the "7483" device. The symbol shown in Fig. 5 will be displayed. Press OK, and then use the mouse to first drag the device to wherever your wish in the drawing area, and then to drop it by pressing the left mouse button. When you're finished placing devices of this type, hit the ESC key.

There is a nearly limitless variety of standard logic devices you may use as well. In the Symbol panel, expand the "primitives" subfolder in the device library, and then "logic." You will find, for example, such devices as `not`, `and2` (a two-input AND gate), `nand4` (a four-input NAND gate), and any others you may need. Select and place in the same way as above.

Other important objects that you may need in your circuit are direct connections to ground and Vcc to provide logical constants of 0 and 1, respectively. To place either of these objects, return to the "primitives" subfolder in the device library, and select "other." The ground symbol is named `gnd` and the Vcc symbol is simply named `vcc`. Their symbols are as shown at right.



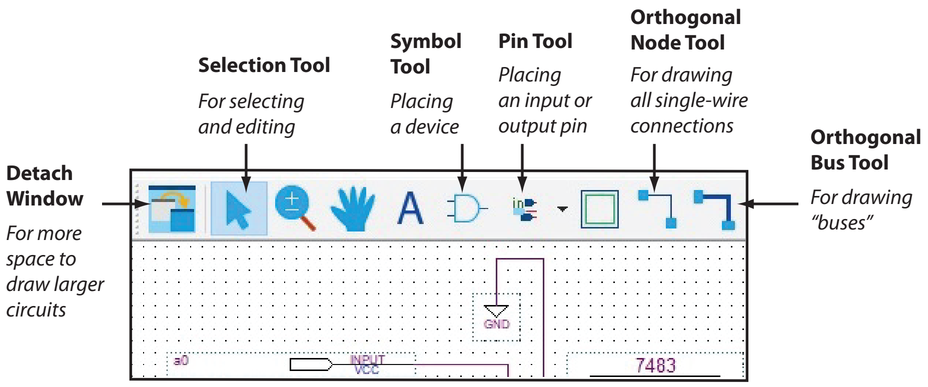


Fig. 11. A portion of the toolbar in the Schematic Editor window showing all the most commonly used tools

Making connections between devices

Connections between devices are easily made using the **Orthogonal Node Tool**. Select this tool, then move the mouse to one of the inputs or outputs of a device (including `VCC` or `GND`). Press and hold the left mouse button, and then move the mouse to any location and release the button. You can connect directly to any other device or another connection, as long as there is *at most one right-angle*. For more complex connections, you can attach any number of further right-angle turns. Intersecting line segments are allowed. If they are electrically connected, then a connection dot will be automatically shown at their intersection.

Important: Place your wires carefully so that they *do not touch other devices in any way (except at valid connection points)*. Good and bad ways to connect devices are shown in Fig. 12.

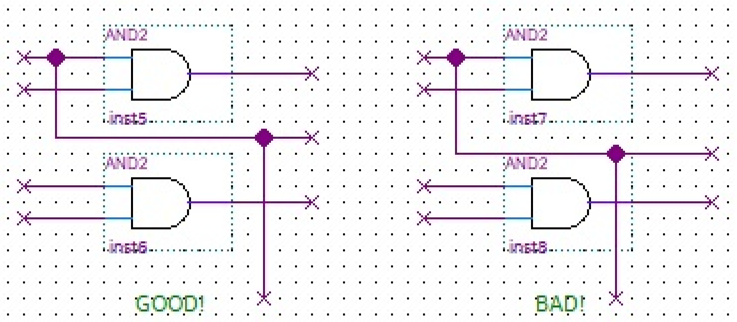


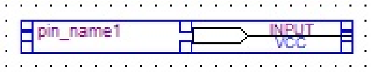
Fig. 12. Don't let wires touch the boxes enclosing devices, nor pass wires through boxes. Don't even let the boxes touch each other!

Bad connections will quietly go unnoticed by the *Quartus* compiler, but the circuit will almost always malfunction.

Inputs and output pins

Finally, you must provide physical inputs and outputs to your adder/subtracter, for which the Pin Tool in Fig. 11 is provided. The pins shown in Fig. 10 have all been created using this tool.

For example, the input pin `e0` in Fig. 10 is created by clicking the drop-down menu on the **Pin Tool**, and selecting "Input". This is placed in your circuit like any other device, and will appear as shown at right.



Give appropriate names to your inputs and outputs as you place them. Move the mouse over the *Quartus*-supplied name (such as "pin_name1" in the above example), double-click on it and enter your own name for this pin. We recommend you use names `e0`, `e1`, `e2`, `e3`, `f0`, `f1`, `f2`, `f3`, and `y0`, `y1`, `y2`, `y3`, `Cout`, corresponding to the names given in the block diagram in Fig. 1 and the sample screenshot in Fig. 10.

Save everything and save often!

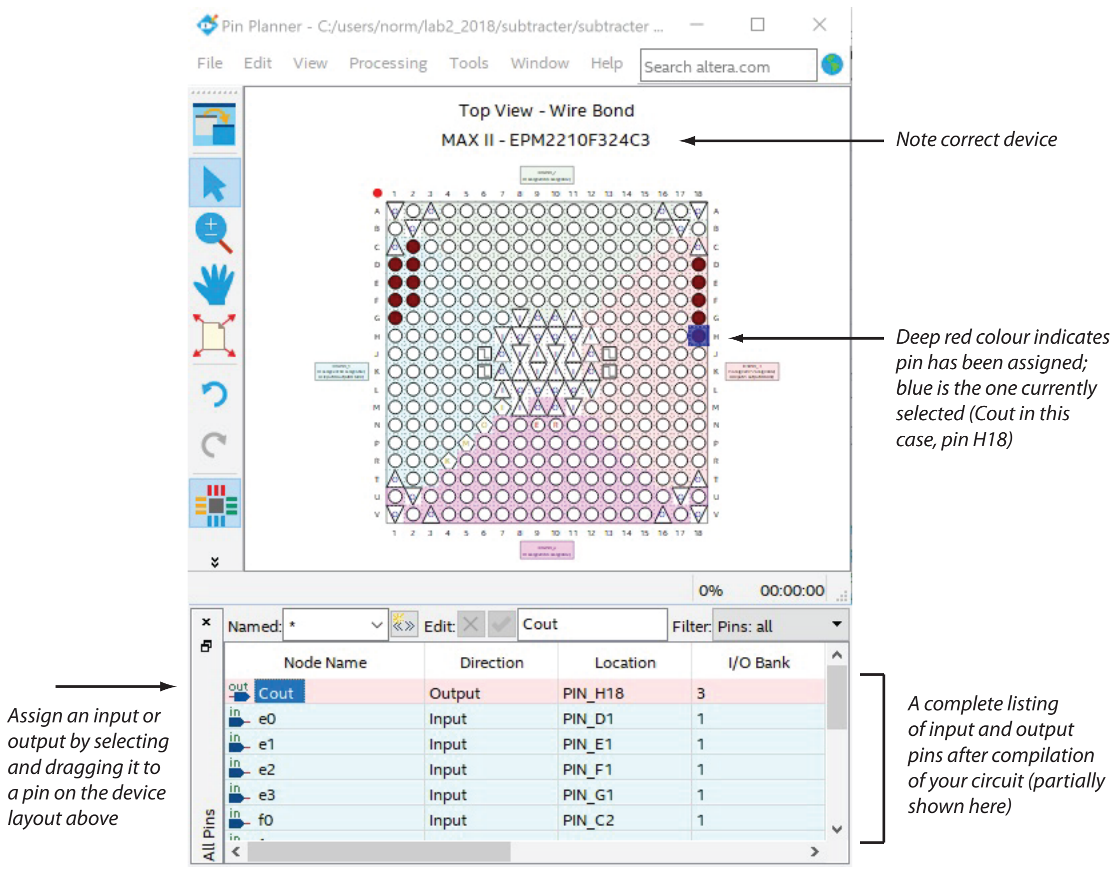
From the menu bar, select `File` ⟶ `Save Project`. This saves the complete project description in a file called "addsub.qpf" (Quartus Project File) in the working directory. If you need to restart *Quartus*, specify this file when you reopen the project.

5.3 Compiling and Making Pin Assignments

The next step is to assign the system inputs and outputs to pins on the CPLD.

Start the *Quartus* compiler by selecting `Processing` ⟶ `Start Compilation` from the menu. Compilation then proceeds through various phases and should finish with several harmless warnings (ignore these). You may follow the compiler's progress in the "Tasks" pane on the left side of the window (takes about a minute or two). Press OK. This compilation step is necessary so that *Quartus* knows what inputs and outputs you have specified in you schematic. Pin assignments can now be established easily using the "Pin Planner," for which a sample screenshot is shown in Fig. 13, and described below.

1. From the menu bar, select `Assignments` ⟶ `Pin Planner`. Enlarge the Pin Planner window that comes up so that you can easily read the information there. Pins on the device are organized as a matrix, with rows designated by letters of the alphabet, and columns designated by numbers.



Node Name	Direction	Location	I/O Bank
Cout	Output	PIN_H18	3
e0	Input	PIN_D1	1
e1	Input	PIN_E1	1
e2	Input	PIN_F1	1
e3	Input	PIN_G1	1
f0	Input	PIN_C2	1

Note correct device

Deep red colour indicates pin has been assigned; blue is the one currently selected (Cout in this case, pin H18)

Assign an input or output by selecting and dragging it to a pin on the device layout above

A complete listing of input and output pins after compilation of your circuit (partially shown here)

Fig. 13. Typical Pin Planner view. Pins on the CPLD are arranged as a matrix; assigned pins are shown highlighted.

2. Take a close look at your MKS to notice the text in the lower-left corner of the interface card. This gives the important mapping of the pins on the interface card (to which your inputs are connected) to physical pins on the CPLD.

3. Assign pins to the E bits, one bit at a time as follows.

   In the bottom pane of the Pin Planner window, locate the node name `e0`. Now simply drag the name `e0` to pin D1 on the device. Pin D1 should now be shown highlighted on the device. Similarly, drag the other three input bits `e1`, `e2`, `e3` to the pins E1, F1, and G1, respectively.

4. Repeat this procedure now for the F bits, starting with `f0` and dragging it to pin C2. Use pins D2, E2, and F2 for the other three bits `f1`, `f2`, `f3`, respectively.

5. The final input bit `Cin` will be used to select either addition or subtraction. Drag it to pin A11 (`Cin` is not shown in the sample screenshot of Fig. 13).

6. Finally, repeat this procedure for the Y outputs and C_{out}. Drag `y0` ... `y3` and `Cout` in sequence to pins D18, E18, F18, G18, and H18.

7. At any time, you may double-check your pin assignments. Simply move the mouse over a highlighted pin in the device pin diagram, and *Quartus* will flash the assignment.

Close the Pin Planner window. Your assignments are automatically saved.

5.4 Preparing for Initial Testing

Switch on the laboratory breadboard.

1. Select `Processing` ⟶ `Start Compilation` from the menu to compile your circuit a second time.

2. Start the *Quartus* programmer by selecting `Tools` ⟶ `Programmer`. The programmer window should appear. The default hardware configuration for programming the device is at the very top of this panel, as shown in Fig. 14. Make sure "USB-Blaster [USB0]" is shown with mode "JTAG." These settings are sensed automatically as long as the MKS is connected to the PC.

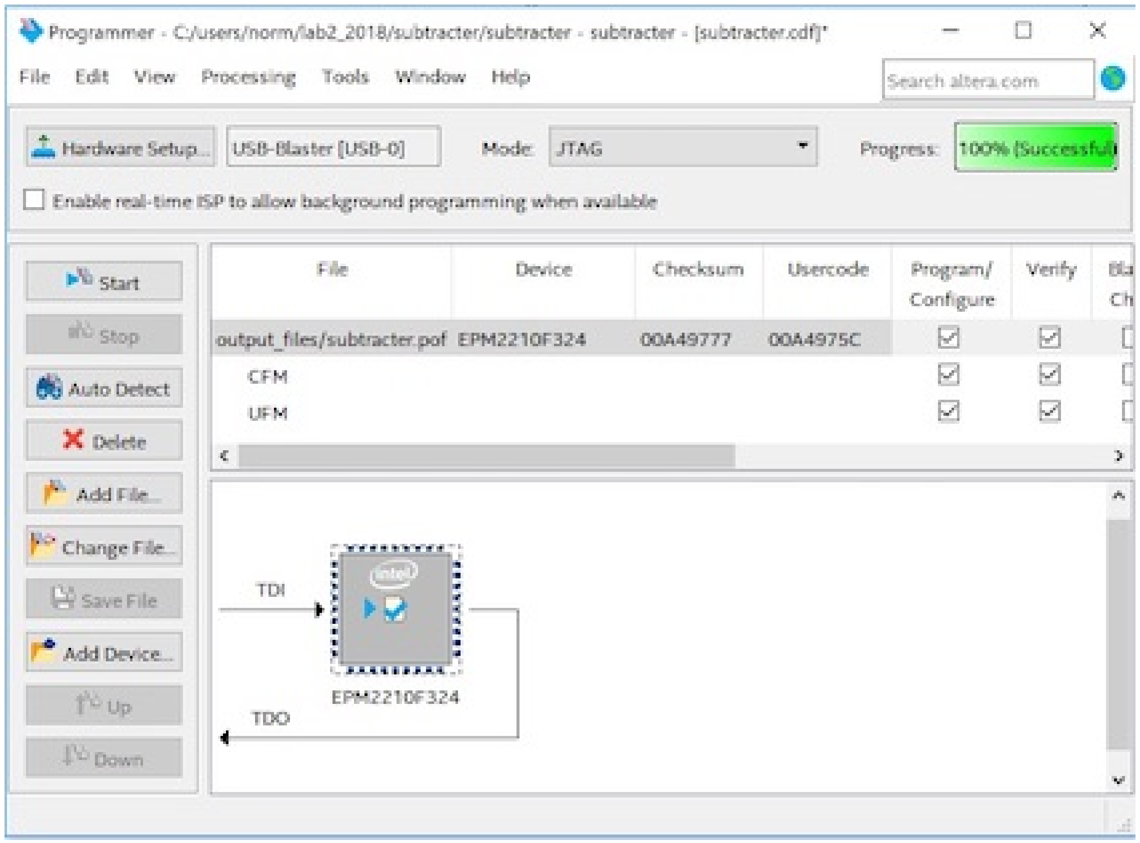
NOTE: *If the Quartus software is being run for the first time on your computer, the USB Blaster device driver might not be listed. Click "Hardware Setup" at the top of the panel, and USB Blaster should appear in a new panel. Double-click on it, and close the panel.*

3. If your "addsub.pof" file is not shown in the programmer window in a similar way to project name shown in Fig. 14, press the "Add File" button, and select the folder "output_files". Select "addsub.pof" and press OK. Click the boxes labeled "Program/Configure" and "Verify" beside the filename so that checkmarks are indicated. Then, hit the Start button, and device programming should commence. Watch the progress bar in the upper-right corner.

5.5 Experimental Results – Test the 4-bit Adder/Subtracter

What to Do

Partially test your design by completing the truth table below.



Programmer - C:/users/norm/lab2_2018/subtracter/subtracter - subtracter - [subtracter.cdf]*

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Bla Ch
output_files/subtracter.pof	EPM2210F324	00A49777	00A4975C	✓	✓	☐
CFM				✓	✓	☐
UFM				✓	✓	☐

Fig. 14. A sample programmer window; make sure output file is correctly indicated, boxes are checked, and "USB Blaster [USB-0]" shows correctly

		E-F (calc)		*E-F (meas)*		*E+F (calc)*		*E+F (meas)*	
E	*F*	C_{out}	Y	C_{out}	Y	C_{out}	Y	C_{out}	Y
1001	0111	1	0010						
0101	1010								
1100	0011								
0101	0110								
0110	1101								

Record the four-bit sum and difference Y as well as the C_{out} bit for each test.

What to Write About

For each addition and subtraction in the above table, calculate Y and C_{out} by hand, and compare with your measured values. Then, answer the following question.

- When we perform the *addition* of 4-bit unsigned numbers, the C_{out} bit indicates unsigned overflow if $C_{out} = 1$. For *subtraction* of unsigned numbers, does C_{out} indicate an overflow in the same way? Briefly explain.

Be sure to include your design work, and a *Quartus* printout of your final circuit.

6 Build and Test the 3x3-Bit Multiplier

In *Quartus*, select `File` $\longrightarrow$ `Close Project` to close the adder/subtracter project. Now make a new folder in your file space for the multiplier project. This **must** be a new folder.

6.1 Creating the Quartus Project

Following the steps in Section 5, launch the New Project Wizard to create a project with the name "multiplier", and then proceed to enter and compile your circuit. When drawing, be sure to leave adequate space between adder modules to make room for your AND gates and wires!

When it comes to assigning pins with Pin Planner, there will be only three bits each for the E and F inputs, and six bits for Y, which you may assign as follows.

- Assign inputs `e0`, `e1`, `e2` to pins D1, E1, and F1.
- Similarly, assign inputs `f0`, `f1`, `f2` to pins C2, D2, and E2.
- Assign, in sequence, the outputs `y0` - `y5` to pins D18 $-$ J18.

Again follow the procedure in Section 5 to download your circuit to the MKS, and you're ready to test it.

6.2 Experimental Results – Test the Multiplier

What to Do

Partially test your design by completing the truth table below.

		Y (calculated)	*Y (measured)*
011	011	001001	
101	010		
110	011		
110	101		
111	110		

What to Write About

For each multiplication in the above table, calculate Y by hand, and compare with your measured values.

Be sure to include your design work, and a *Quartus* printout of your final circuit.