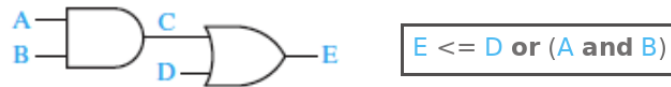


# Digital Logic Design with VHDL



## Introduction

A hardware description language is a computer language used in order to model the behavior and structure of electronic circuits, with the design of digital logic circuits being one of their most common applications. They often share much of their syntax and function with general-purpose programming languages like C or Java, though they are not programming languages in themselves.

As modern electronic circuits become more and more complex, it is becoming increasingly impractical and expensive for designers to fabricate hardware prototypes of their systems. Hardware description languages enable cheap, rapid design and prototyping of electronic systems as they allow circuit simulation on a variety of different systems without the need for a physical prototype.

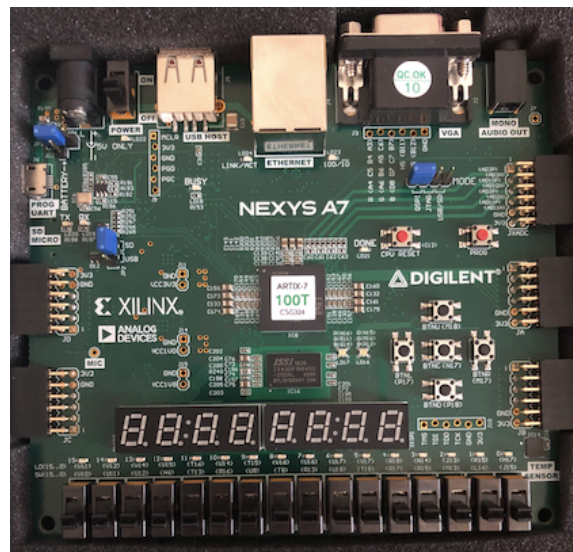


Figure 1: An FPGA, a special type of hardware that can be programmed with an HDL

In short, a hardware description language allows you to trade in those cumbersome 74-series chips for lines of code. For example, if you want to model the logic function  $D = C$  and  $(A \text{ or } B)$ , it can be implemented in VHDL with this line:

```
D <= C and (A or B);
```

The hardware description language that will be used for this lab assignment is VHDL (an acronym for VHSIC (very high speed integrated circuit) Hardware Description Language)

one of the most widely-used HDLs in use today. A program you are very familiar with is LogicWorks; it is capable of VHDL simulation and will be used for this exercise.

### Prerequisites

You need to understand the basics of designing digital logic circuits using boolean algebra and techniques such as Karnaugh maps. You must be familiar with constructing combinational logic circuitry with LogicWorks. You need to have a basic understanding of the structure, function, and syntax of VHDL code.

### Objectives

The objective of this lab assignment is to introduce how to use VHDL within the LogicWorks environment. You will learn how to design and create VHDL-based modules. In this assignment you will be shown how to create a 1-bit full adder using VHDL. Then, you will follow the same procedure to design a new VHDL module on your own. You will also learn how to combine different modules to build a more complicated digital system.

### Theory

The truth table for a 1-bit full adder is shown below. It is all that is needed to create an n-bit full adder. From this truth table, it is necessary to create equations for both the Sum and Carry-out outputs.

| A | B | Cin | Sum | Cout |
|---|---|-----|-----|------|
| 0 | 0 | 0   | 0   | 0    |
| 0 | 0 | 1   | 1   | 0    |
| 0 | 1 | 0   | 1   | 0    |
| 0 | 1 | 1   | 0   | 1    |
| 1 | 0 | 0   | 1   | 0    |
| 1 | 0 | 1   | 0   | 1    |
| 1 | 1 | 0   | 0   | 1    |
| 1 | 1 | 1   | 1   | 1    |

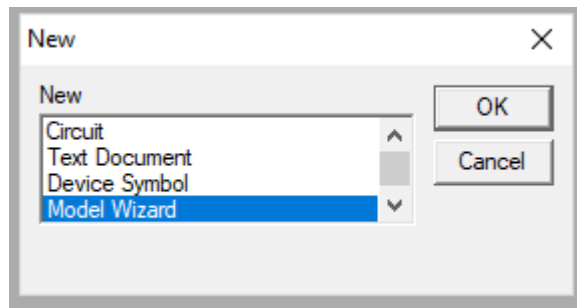
Using these equations, a VHDL module can be developed for the 1-bit full adder. How does one create a 4-bit adder from 1-bit adders? Think about how an adder works with carry bits. You may find it useful to review the theory of the overflow circuit as well. Remember that you have built it in the previous lab assignment about adders.

## Procedure

In this assignment we are going to design a 4-bit adder circuit in **LogicWorks 5**. The circuit is divided into two types of modules, namely: 1-bit full adder and an overflow detector. Please make sure to save all files to your **P: drive** as this will help you during the submission process and for future reference.

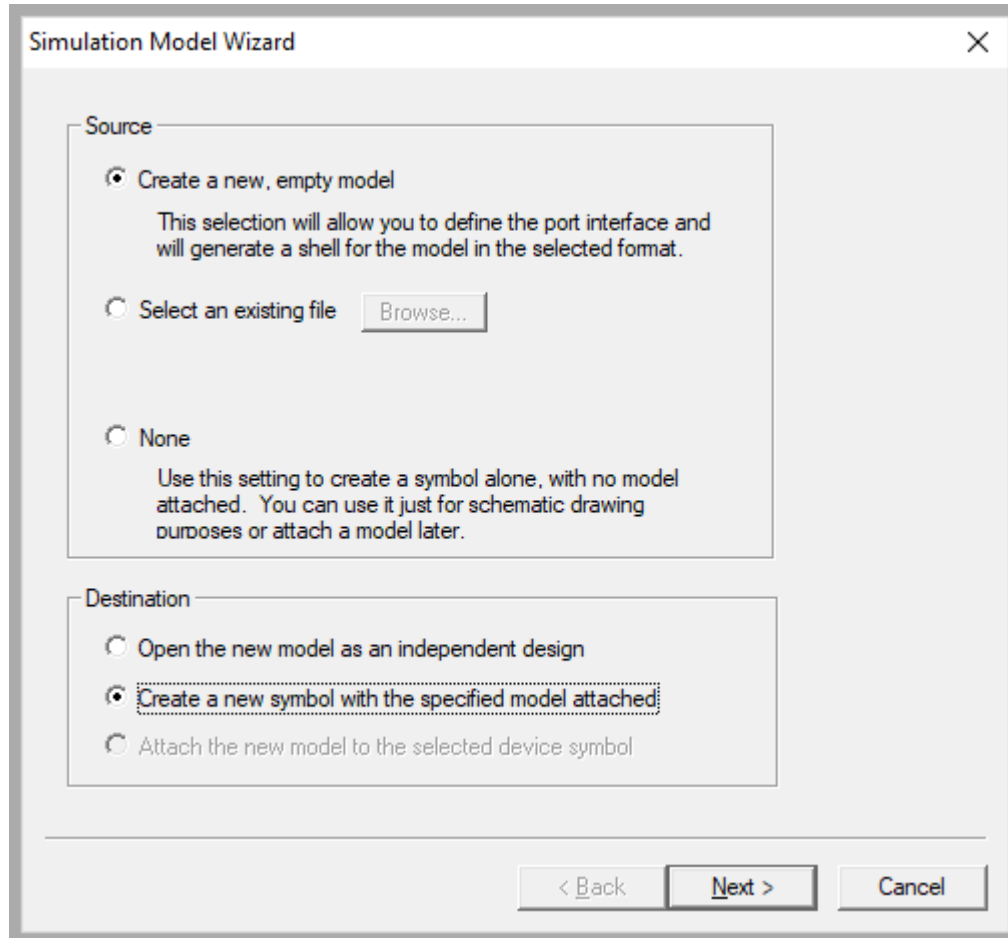
Please follow the instructions below to build the 1-bit full adder module:

- Open **LogicWorks 5**
- From the Main menu select *File -> New -> Model Wizard*



- Click on the *OK* button.

- The *Simulation Model Wizard* window will appear. In the *Source* group select “*Create a new, empty model*” also make sure that the option “*Create a new symbol with the specified model attached*” in the *Destination* group is chosen as shown below then click on the *Next* button.



- The *Model Info* window will appear. Please select the desired model type as *VHDL* and enter *FullAdder* (or any name of your choice) as a name for the new model as shown below (Please note that spaces are not allowed in model names. Also, avoid VHDL keywords such as *and*, *or*, *xor*, etc...).

Model Info

Select the desired model type

Structural Circuit  
VHDL

Create a VHDL language file which can be used to describe the function of this device.

Enter a name for the new model FullAdder

< Back Next > Cancel

- Click *Next*

- Now that we have told LogicWorks about our model type and its name, it is time to describe its interface through the *Model Port Interface* window. First define the inputs for the full adder as follows: Set the function to *Input*, enter '*a*' in the Name box, then click on the *Add Single Bit* button. Do the same for inputs '*b*' and '*c*'. Now for defining the outputs, select *Output* from the *Function* group, enter '*sum*' in the *Name* box and click on the *Add Single Bit* button. Do the same thing for the '*carry*' output. You should end up with a window like this:

Model Port Interface

Use the controls at right to add pins to the interface list. NOTE: If you are attaching this model to an existing device symbol, the interface list must exactly match the pins on the symbol.

| Name  | Func | Left | Right |
|-------|------|------|-------|
| a     | In   |      |       |
| b     | In   |      |       |
| c     | In   |      |       |
| sum   | Out  |      |       |
| carry | Out  |      |       |

Drag and drop to re-order items in the list

Function

☐ Input

☒ Output

☐ Bidirectional

Name

<< Add Single Bit

Vector

Left Bit Number

Right Bit Number

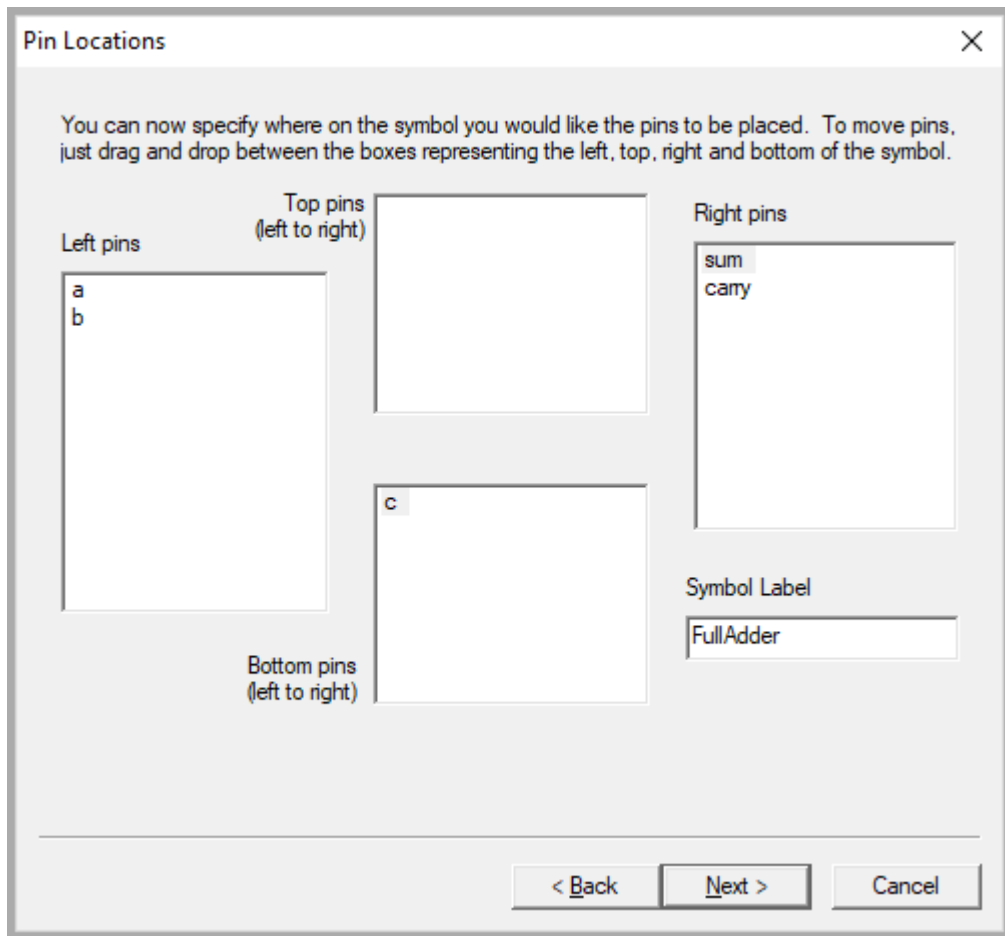
<< Add Vector

>> Remove

< Back Next > Cancel

- Click *Next*

- Now it is time to decide on pin locations for our new symbol. It is always considered a good practice to keep the input pins on the left hand side and the output pins on the right hand side. Please drag and drop signal names to the appropriate boxes to configure the pin locations according to the suggested configuration shown below. Also note that you can drag and drop signals to reorder the corresponding pins within any of the four boxes.



The image shows a 'Pin Locations' dialog box with a close button (X) in the top right corner. Inside the dialog, there is instructional text: 'You can now specify where on the symbol you would like the pins to be placed. To move pins, just drag and drop between the boxes representing the left, top, right and bottom of the symbol.'

The dialog contains four boxes for pin placement:

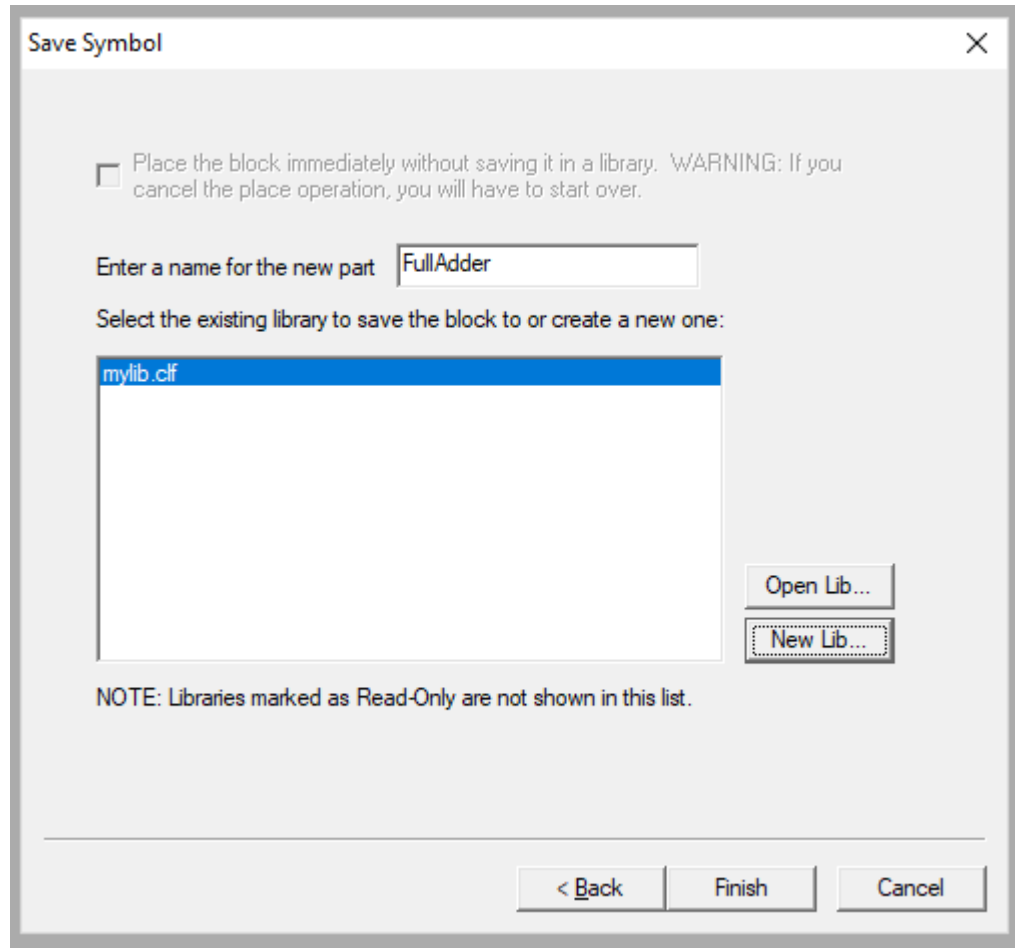
- Left pins**: A box containing the text 'a' and 'b'.
- Top pins (left to right)**: An empty box.
- Bottom pins (left to right)**: A box containing the text 'c'.
- Right pins**: A box containing the text 'sum' and 'carry'.

Below these boxes is a 'Symbol Label' text field containing the text 'FullAdder'.

At the bottom of the dialog are three buttons: '< Back', 'Next >', and 'Cancel'.

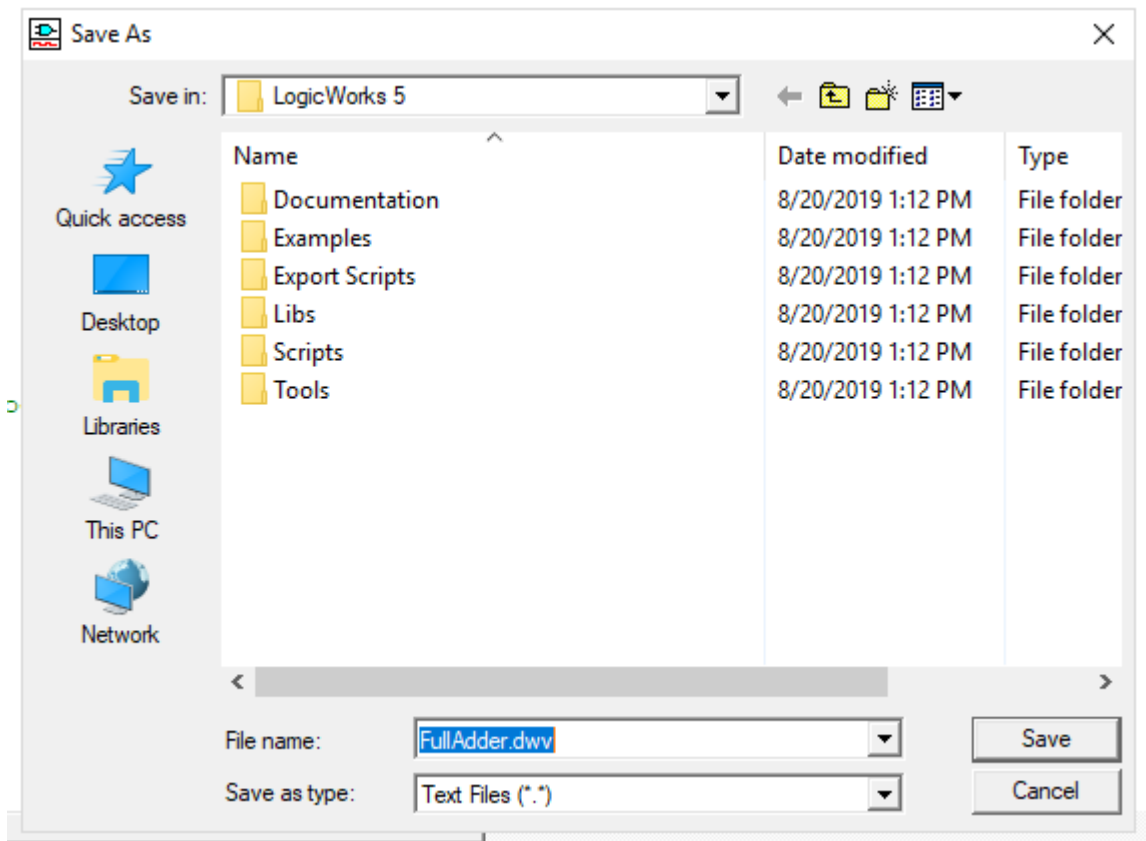
- Make sure your window looks exactly as shown above then click on *Next*

- The *Save Symbol* window will appear as shown below. Click on the *New Lib ...* button to save our new symbol to a new library file (Although you can save the symbol to any of the existing libraries, such as those provided by *LogicWorks*, It is highly recommended to save it to your own new library and remember, save it to your P: drive). We will call this new library *myLib*. The default extension is *.clf*. You can leave the name of the new part as it is and click on *Finish*.





- *LogicWorks* will generously create a VHDL template for our model but we have to save it first, Note that *LogicWorks* is also suggesting the filename for us as shown below (note that the default file extension is *.dwv*)



- Navigate to a suitable folder in your P: drive and click on *Save*
- In case the file is closed, you can browse and reopen the *dwv* file through the *File - > Open menu* command (Shortcut Ctrl+O).

- The **dww** file initially contains the following lines:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FullAdder is

    port(
        b : in    std_logic;
        a : in    std_logic;
        carry : out std_logic;
        sum : out  std_logic;
        c : in    std_logic
    );

end FullAdder;

architecture arch1 of FullAdder is

begin

    -- Your VHDL code defining the model goes here

end arch1;
```

- In place of the green line start defining the behavior of the full adder circuit by adding its two assignment statements. Your final VHDL file should look like this:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FullAdder is

    port(
        b : in    std_logic;
        a : in    std_logic;
        carry : out std_logic;
        sum : out  std_logic;
        c : in    std_logic
    );

end FullAdder;

architecture arch1 of FullAdder is

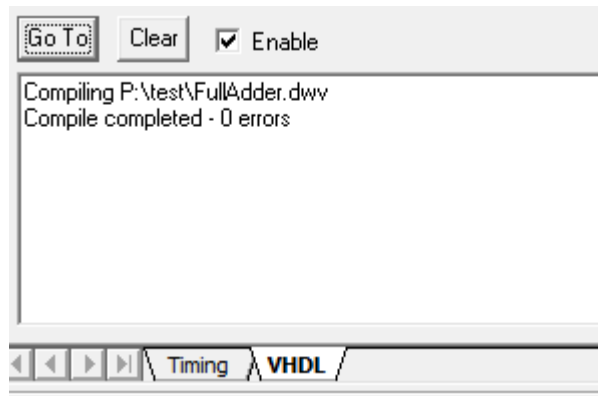
begin

    sum <= (a xor b) xor c;
    carry <= (a and b) or (c and (a xor b));

end arch1;
```

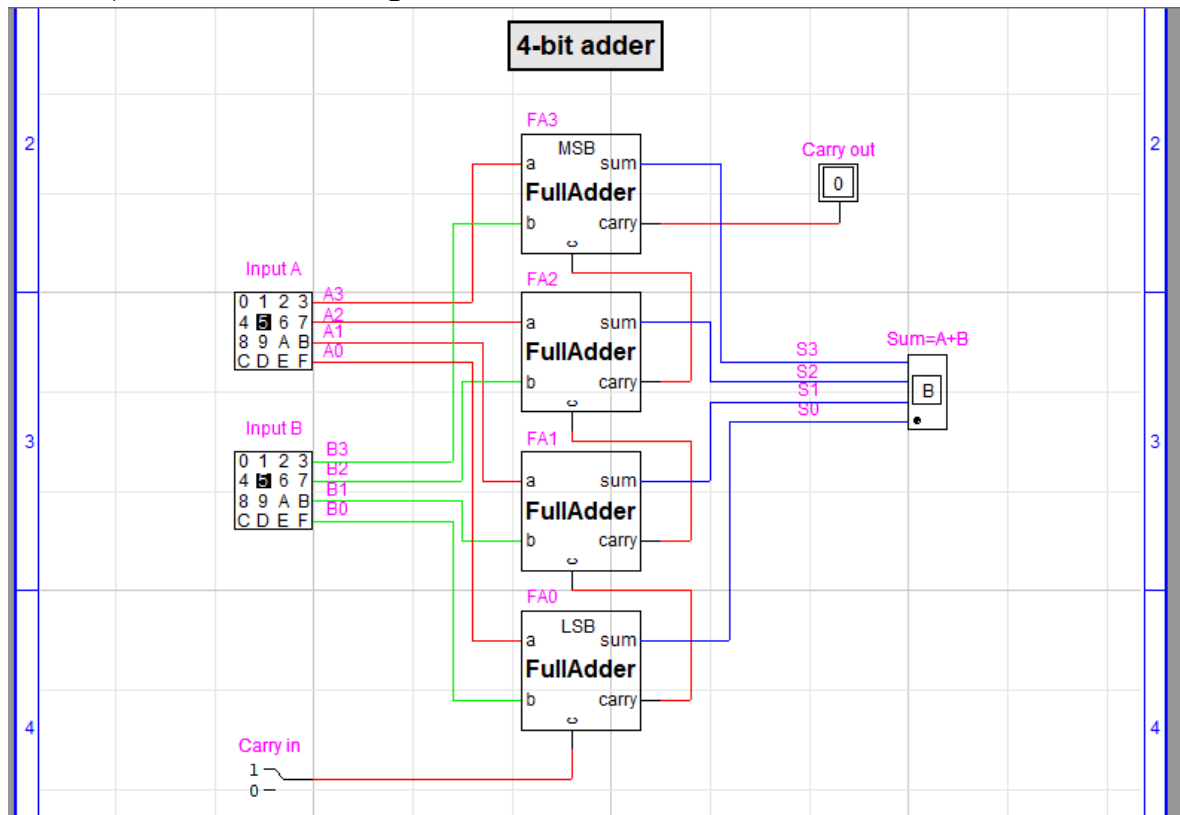
- Now save the file again.
- From the main menu select the command **VHDL -> Compile** (shortcut **Ctrl+Q**).

- Look for the output of the compile command in the **VHDL tab** (probably this shows next to the **Timing tab** in the bottom split window as shown below). It should display ***"Compile completed - 0 errors"***. If there is (are) any error(s), please edit your file to make sure you have typed the VHDL statements correctly. Don't forget the semi-colons!!



- Once the VHDL is compiled with no errors, the full adder model becomes ready!!

- Now, use the newly created symbol, *FullAdder*, as a component in order to design a 4-bit adder circuit. Please use the following schematic diagram as a guide to do this step. At this stage you should be familiar with **LogicWorks** and know how to complete this circuit on your own.



- Note that we are using two **Hex Keyboards** and one **Binary Switch** to stimulate the 4-bit adder with inputs. Also, we use a **Hex Display** and a **Binary Probe** to display the output.
- Now test the above circuit assuming unsigned 4-bit numbers. Make sure to introduce at least one case where the carry out is 1.
- Now that you've learned how to use VHDL to describe and build a module in **LogicWorks 5**, it is time to use these skills. Design and build an **overflow detection** module using VHDL (**Hint**: It might be helpful to review the overflow circuit you have designed before in the Adders lab).
- Add the **overflow detection** module to the 4-bit adder circuit (note that it is not included in the schematic shown above as you should already know where to connect it). Please use a *binary probe* and label it as '**overflow**'.
- Using the built-in 4-bit adders in LogicWorks, turn the above circuit into an adder/subtractor circuit. (**Hint**: use **XOR gates** and one built-in 4-bit adder component to toggle input **B** into  $-B$  (the 2's complement of  $B$ )).
- Don't forget to save the circuit (.cct) file to a convenient location in your **P: drive**.
- Now apply some **test cases** and record them along with the associated outputs in a carefully designed table.

- The test cases should cover the four different combinations of positive and negative inputs in the 2's complement system.
- Also, please make sure to include at least one case where an **overflow** occurs on adding two positive numbers and another one where an **overflow** occurs while adding two negative numbers.

### Deliverables

You must demonstrate your VHDL-based overflow detector to your TA. Submit the lab report as it is required for this assignment. Include the .dwv file for your overflow detection module in your submission.

### References

IEEE Standard VHDL Language Reference Manual

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4772740>

LogicWorks 5: Interactive Circuit Design Software

### Questions

Since you have built a similar circuit in the Adders assignment, which design method did you prefer for constructing the 4-bit adder with LogicWorks: writing VHDL modules or connecting discrete gates? Why?