ELECTRICAL AND ELECTRONICS ENGINEERING
&
COMPUTER ENGINEERING

E E E 2 4 8 | C N G 2 3 2
L O G I C D E S I G N

2 1 | S p r i n g | 2 2

LABORATORIES

D r . G ü r t a ç Y e m i ş ç i o ğ l u
M b o n e a G o d w i n M j e m a
A h m e d M a m d o u h

2 TABLE OF CONTENTS

REGULATIONS

- Students are not permitted to perform an experiment without doing **the preliminary work** before coming to the laboratory. It is **not allowed** to do the preliminary work at the laboratory during the experiment. **Students, who do not turn in the complete preliminary work printout at the beginning of the laboratory session, cannot attend the lab. No "make-up" is given in that case.**

- No food or drink in the lab.

- There may be a quiz before each laboratory session, which will start promptly at the beginning of the lab. **Students who miss the quiz cannot attend the lab. No "make-up" is given in that case.** There won't be any extensions in the quiz time for latecomers. Therefore, students have to be at the lab on time for the quiz.

- Only the following excuses are valid for taking lab make-up:
  1. **Health Make-up:** Having a health report from METU Medical Center.
  2. **Exam Make-up:** Having an exam coinciding with the time of the laboratory session. The student needs to notify the instructor in advance if this is the case.

- Experiments will be done **individually**. The lab instructor will inform you of any part that you can do in groups.

- **Cheating or plagiarism is not tolerated. Plagiarism is a form of cheating as is using someone else's written word with minor changes and no acknowledgement. If you are caught cheating or plagiarising, you will at the very least receive a zero for the whole experiment. Disciplinary action may be taken.**

- Students who miss the lab **two times** without a valid excuse get zero as the laboratory portion of the course grade.

- *Those who fail to get a satisfactory score from the laboratory portion may fail the class. This score is expected to be 70% but may be adjusted up or down with the initiative of the course instructor.*

EXPERIMENT #2 & #3

HIERARCHICAL DESCROPTION OF COMBINATIONAL LOGIC WITH MULTIBIT SIGNAL

2.1 OBJECTIVE

This laboratory will build on the design entry knowledge acquired in the first LAB to design fundamental combinational circuits to be used by Arithmetic Logic Unit (ALU). ALU is important digital logic circuit which can be found on most of the microcontroller, microprocessors and DSPs to perform arithmetical, logical and compare operations between the inputs. In LAB 2 and LAB 3 the combinational circuits such as adder, multiplexers, comparator and decoder circuits which are used to build ALU will be designed.

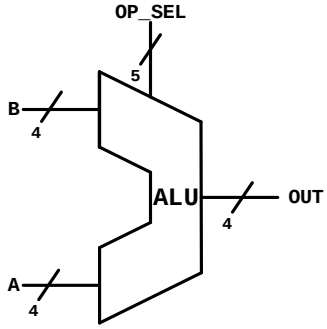


Figure 1: Arithmetic Logic Unit (ALU)

2.1.1 LAB 2

In the first part of the experiment, 1-bit Full Adder will be designed using behavioural (continuous) description. This will then be extended to a 4-bit Adder/Subtractor through hierarchical design. In order to design a 1-bit Full Adder, 2-to-1 multiplexer is required to select either original input or complemented input.

In the second part, 4-bit comparator will be designed to compare the two 4-bit numbers using behavioural (procedural) description. These two logic blocks will then be used for arithmetic unit.

In the third part of the experiment a decoder will be designed to display some numbers at the 7-segment LEDs on DE0 Demo Board.

2.1.2 LAB 3

In the fourth part of the experiment basic logic gates designed in LAB 1 will be combined with multiplexers to be used as a logic unit of ALU.

In the fifth part of the experiment, multiplexers will be designed to choose the operation either from arithmetic unit or from logic unit (ADD/SUB, AND, OR, NAND, NOR, XOR, XNOR, NOTA, NOTB).

Finally, all of the Verilog HDL design modules will be integrated in the top-level design to implement complete ALU design as depicted in Figure 2.

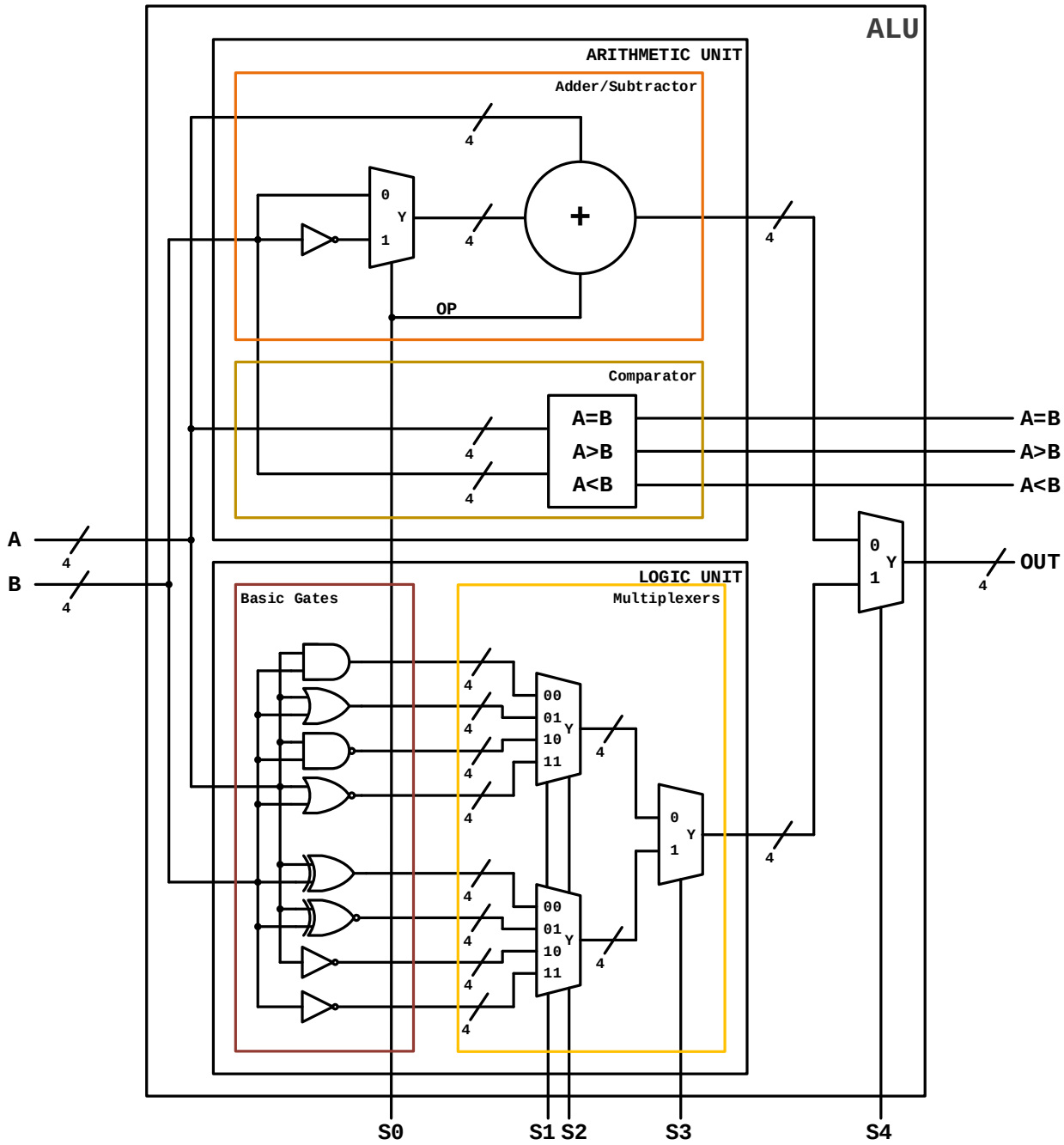


Figure 2: Complete ALU Block Diagram

2.2 PRELIMINARY WORK

2.2.1 VERILOG HDL

Read through Section 2.3 - 2.3.4 to learn more about some of the Verilog HDL structural and behavioural modelling constructs, and hierarchical schematics with multi-bit signals.

2.2.2 2-TO-1 MULTIPLEXER (Pre-work)

1. Design a 2-to-1 multiplexer such that a 4-bit signal and its 1's complement will be its inputs. Introduce a *Select (S)* input so that one of the inputs will be the output of the multiplexer. It is sufficient to show the multiplexer symbol (and the symbol of any other gates that you use for the corresponding design) in the answer, and label all the inputs and outputs.

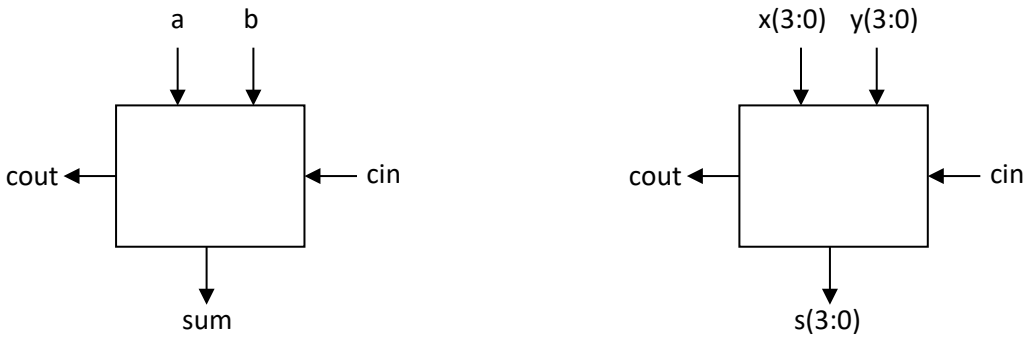
Table 1: 2-to-1 Multiplexer Operation

Select	Multiplexer Output
0	Input [3..0]
1	~Input [3..0]

2. Write a Verilog code module to implement the multiplexer designed in 2.2.2.1. Take the 1's complement of the input signal inside your Verilog code, i.e. there is no inverter gate available. Please follow the procedural approach of behavioural modelling and use parametric design principles.

2.2.3 4-BIT ADDER/SUBTRACTOR (Pre-work)

1. A full Adder has three inputs, A, B, CIN (Carry-IN), and two outputs, SUM and COUT (Carry-OUT). The output represents the result from computing binary addition of the three inputs. Derive a 1-bit Full adder truth-table and write down the logic expressions for SUM and COUT.
2. Write a behavioral Verilog code to implement the full adder designed in 2.2.3.1.
3. Use the symbol in Figure 3(a) for 1-bit Full-Adder and indicate how would you interconnect four such adders in order to build a 4-bit Ripple-Carry-Adder for which a symbol is depicted in figure Figure 3(b).



(a) (b)

Figure 3: (a) 1-bit Full Adder Symbol, (b) 4-bit Ripple Carry Adder Symbol

4. Write a parametrized Verilog code using 2.2.3.2 to implement 4-bit ripple carry adder.

5. Use structural (hierarchical) design principles and modify the Verilog code in 2.2.3.4 in order to create a 4-bit adder/subtractor circuit. Please note that instead of "CIN" signal (Carry-IN) used in 2.2.3.4, use "OP" (operation) signal as depicted in Figure 4. This will make it easier to get the 2's complement of a number when it is necessary. Note that when the "OP" signal changes, the operation performed also changes. Also, please consider that "OP" signal also affects one of the input signals coming to the addition/subtraction unit. See table
6. Table 2. This will make it easier to code your unit since you are just using 1-bit adder unit to design adder/subtractor.

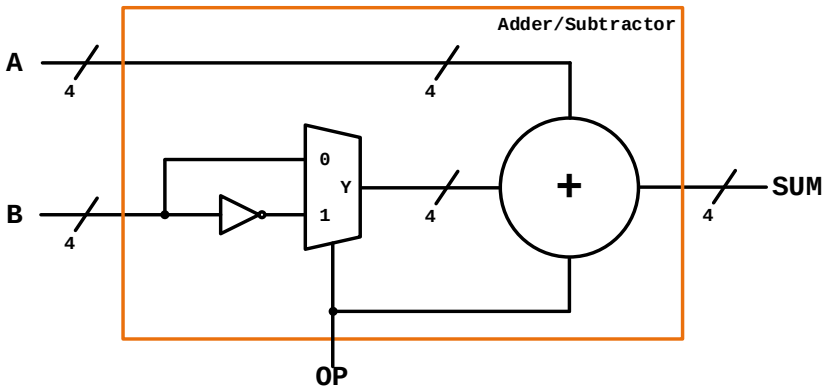


Figure 4: Adder / Subtractor

Table 2: OP Signal

OP Signal	Operation
0	A+B
1	A+(B'+1)

7. Refer to Modelsim® tutorial and write a testbench for your designs using Modelsim® and simulate your Verilog codes in 2.2.3.2 - 2.2.3.5. By showing all inputs and outputs waveform.

2.2.4 4-BIT COMPARATOR (Pre-work)

1. A 2-bit comparator has two inputs A1, A0 and B1, B0 and three outputs, A<B, A>B, A = B. Derive a 2-bit comparator truth-table and using K-MAPs write down the simplified logic expressions for A<B, A>B, A = B.
2. Write a parametrized Verilog code to implement 4-bit comparator module.
3. Write a testbench for your design using Modelsim® and simulate your comparator Verilog code.

2.2.5 ARITHMETIC UNIT (Experimental)

1. After a 4-bit Adder/Subtractor and a 4-bit Comparator module have been implemented and tested in 2.2.3 and 2.2.4, combine these two modules as shown in Figure 5 in one structural Verilog HDL design named as **`ARITHMETIC_UNIT`**.

2. Write a testbench for your design using Modelsim® and simulate your arithmetic unit.

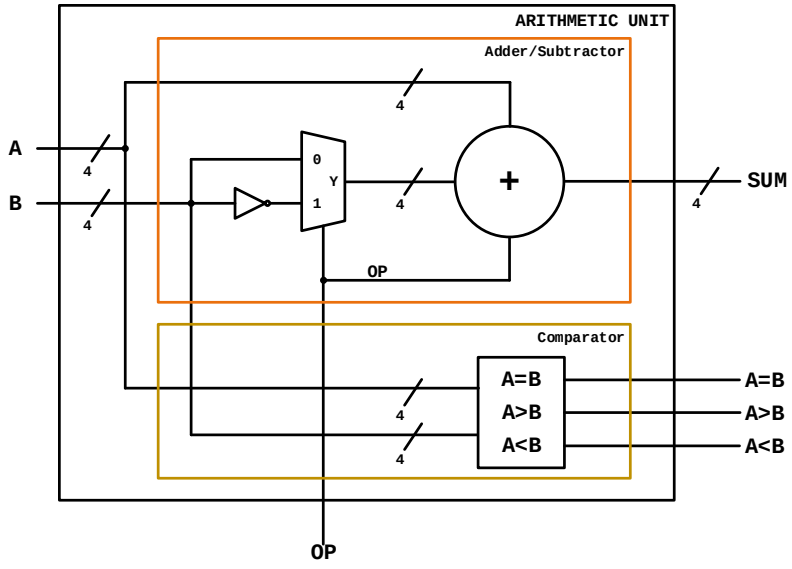


Figure 5: Arithmetic Unit

2.2.6 DECODER (Pre-work)

1. DE0 board has 4 adjacent 7-Segment Display. By considering this, use combinational logic design principles to design a 7-Segment Display decoding logic such that when a 4-bit signed binary number is entered, the number will show up on the rightmost 7-Segment Display. Also, if the number is negative, the sign will be displayed on the 7-Segment Display next to the one that shows the number (Rightmost two 7-Segment Display will be employed). For example, when the user enters 0001, two of the vertically aligned LEDs of the rightmost 7-Segment Display should light up. Similarly, when the user enters 1110 (-2) in binary, five of the LEDs on the rightmost 7-Segment Display should light up, and the LED located in the middle of the adjacent 7-Segment Display should light up to indicate the number is negative. Please *optimize* your logic as much as possible to yield the lowest cost i.e. the lowest number of input switches, gates, literals, and gate inputs. Input-to-Output delay is *not* a concern in this application.

Hint: Remember each 7-Segment Display on DE0 board is characterized by 8 independent ***active low*** outputs (including the dot in the lower right) that need to be all defined to determine the displayed character, as depicted in Table 4.

Table 3: Mapping of DE0 7-Segment Display to Decoder

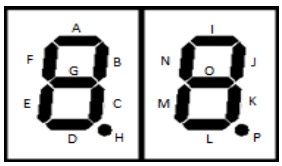


Table 4: Decoder Outputs

Binary Value	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
-8	off	off	off	off	off	off	on	off	on	on	on	on	on	on	on	off
-7	off	off	off	off	off	off	on	off	on	on	on	off	off	off	off	off
-6	…															
…	…															
+6	off	off	off	off	off	off	off	off	on	off	on	on	on	on	on	off
+7	off	off	off	off	off	off	off	off	on	on	on	off	off	off	off	off

2. Write a Verilog code to implement the decoder module designed in Section 2.2.6.1. Please follow the procedural approach of behavioural modelling. For more information about procedural approach, you can refer to your first lab manual or textbook.

2.2.7 LOGIC UNIT WITH MULTIPLEXERS (Experimental)

1. Using the basic gates in LAB 1, design the logic unit by combining with 4-to-1 and 2-to-1 Multiplexers as to fulfil the correct operation as described in Table 4.
2. Write a parametrized and structural verilog HDL code named as **`LOGIC_UNIT`** using the Verilog code that have been implemented in LAB 1 for basic gates and implement the logic unit designed in 2.2.7.1.

Table 5: Logic Unit Operations

Select			Multiplexer Output
L2	L1	L0	F
0	0	0	A.B
0	0	1	A⊕B
0	1	0	A+B
0	1	1	A⊙B
1	0	0	(A.B)’
1	0	1	A’
1	1	0	(A+B)’
1	1	1	B’

3. Write a testbench for your design using Modelsim® and simulate your Logic Unit.

2.2.8 ARITHMETIC LOGIC UNIT (ALU) TOP LEVEL DESIGN (Experimental)

1. After all components required for ALU is designed, implemented and tested. Create a top-level structural (hierarchical) Verilog HDL file called **`ALU`** and instantiate and connect all the components as depicted in Figure 2.
2. Write a testbench for your design using Modelsim® and simulate your ALU.

2.3 STRUCTURAL AND BEHAVIOURAL VERILOG HDL CODE

2.3.1 MULTIBIT SIGNALS

In the first experiment, you learned how to define input, output and intermediate signals (wire). Many practical codes deal with buses instead of wires which are literally multibit ports or wires. In Verilog, to define the bus we use arrays. To define arrays, we simply add [most significant bit: least significant] before the name of the input, output or wires. The following are two examples:

```
Output [0: 3]  D;
   wire[7: 0]  SUM;
```

The first statement declares an output vector D with four bits, 0 through 3. The second declares a wire vector SUM with eight bits numbered 7 through 0. (Note: The first (leftmost) number (array index) listed is always the most significant bit of the vector.) The individual bits are specified within square brackets, so D [2] specifies bit 2 of D. It is also possible to address parts (contiguous bits) of vectors. For example, SUM [2: 0] specifies the three least significant bits of vector SUM. Example below defines how multi-bit decoder can be implemented.

```
module decoder (data_in, data_out);

input [1:0] data_in;
output [3:0] data_out;
reg [3:0] data_out;

always @(data_in)

      case (data_in)
            2'b00 : data_out = 4'b0001;
            2'b01 : data_out = 4'b0010;
            2'b10 : data_out = 4'b0100;
            default: data_out = 4'b0000;
      endcase

endmodule
```

2.3.2 BEHAVIOURAL MODELLING

In the previous lab, you learned how to write behavioural Verilog code with two different approaches. These two approaches were:

Continuous: which is based on defining outputs using Boolean algebra with keyword assign

Procedural: defining the input-output behaviour only (like defining the truth table) using keyword always and statements like if … else, case, while, etc.

One of the main reasons Verilog or similar languages are popular for hardware description is the capability of behavioural modelling. Behavioural modelling allows a designer to describe hardware using

high-level abstractions by focusing on the desired behaviour instead of details of the logic to get to that behaviour. It is important to recognize that a good digital designer still needs to be very familiar with the logic derivation in order to check if the final hardware solution delivered is optimal.

2.3.3 STRUCTURAL (MODULAR) CODING

As described in the first experiment, structural coding uses gates. These gates can either be predefined Verilog gates, such as and, or, etc. or can be user-defined gates. Example below shows such hierarchical design in Verilog HDL where user-defined Simple_Circuit is used to build a more complex circuit.

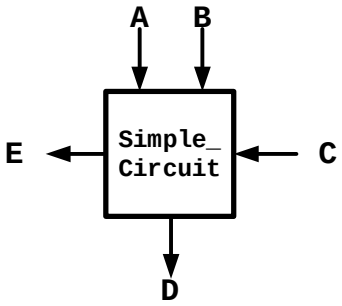


Figure 6: Simple Circuit Block Diagram

```
module    Simple_Circuit (A,B,C,D,E);
output    D,E;
input     A,B,C;
wire      w1;
and       G1(w1,A,B);
not       G2(E,C);
or        G3(D,w1,E);
endmodule
```

Note that in the structural method if you are using user-defined blocks, the Verilog file that you want to instantiate must be added to your project. To do this, open assignment menu and from there follow these steps: settings → files → find your file using the browse icon → add to your project. For more details, you can refer to the tutorial documents. “Simple_Circuit” statement allows the “Simple_Circuit” entity to be used as a component (subcircuit) in the architecture body. The design is described using four instantiations of the Simple_Circuit. Each instantiation statement starts with the instance name “Simple_Circuit” in this example, and then a name, which can be any legal Verilog name followed by the output, input port. The names must be unique.

2.3.4 BEHAVIOURAL CONSTRUCTS EXAMPLES

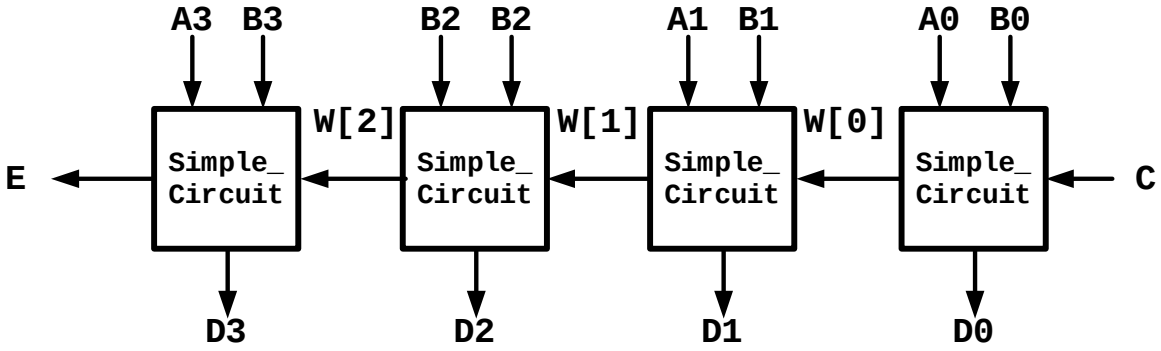


Figure 7: 4-Bit Simple Circuit Block Diagram

```
module top (A, B, C, D, E);

parameter size = 4;

input [size-1 : 0] A,B; //A[3:0], B[3:0]
input C;
output [size-1 : 0] D;  //D[3:0]
output E;
wire [size-2 : 0] W;    //W[2:0]

genvar i;
generate
for (i = 0; i < size; i= i+1) begin: top


      if(i==0)
            Simple_Circuit U1 (A[i], B[i], C, D[i], W[i]);

      else if (i == size-1)
            Simple_Circuit U1 (A[i], B[i], W[i-1], D[i], E);

      else
            Simple_Circuit U1 (A[i], B[i], W[i-1], D[i], W[i]);

end

endgenerate

endmodule
```

2.4 HIERARCHICAL SCHEMATIC ENTRY WITH MULTIBIT SIGNALS

Modular design entry in Verilog HDL has been described in Section 2.3.4. Similarly, multiple schematic or Verilog design modules can be pulled together in the schematic capture tool using their symbols.

2.4.1 SYMBOL CREATION

Once a design has been fully completed, choose **File** from Top Menu, and roll the mouse over to **Create/Update** selection. A side menu appears, from this menu select **Create Symbol Files for Current File** in order to create a symbol (.bsf file) for your design. The procedure is the same for both Schematic and Verilog design modules.

2.4.2 MODULE INSTANTIATION IN SCHEMATIC CAPTURE TOOL

If the design module of interest has been created as part of another project, pull it into the current project using **File → Open …,** browse and choose the relevant schematic or Verilog design file, making sure that **Add file to current project** checkbox is selected. This can be repeated for all the modules that need to be pulled into the current project. The symbols created by the user can be used for the current project in same fashion as other symbols such as AND gate, OR gate, etc in schematic design.

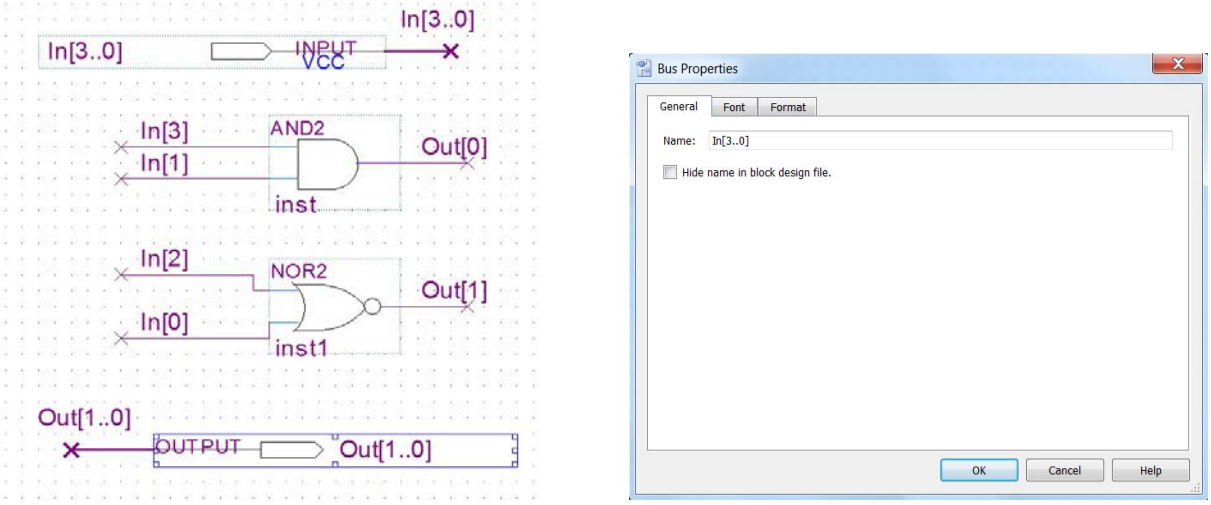


Figure 8: Multi-bit signal connectivity and naming example.

2.4.3 MULTI-BIT SIGNAL ENTRY IN SCHEMATIC TOOL

Multi-bit signals are entered using **Orthogonal Bus Tool**. A signal is interpreted as multi-bit only after it is labelled. Some examples of valid multi-bit signals names are:

`X [3..0]` : A 4-bit signal named using vector notation
`A, B, C, D` : Four 1-bit signals grouped together to form a 4-bit signal
`Y [4..2]` : A 3-bit signal; the name implies it may be a sub-branch of the Y signal with more bits

1-bit or multi-bit signals (or I/O ports) are sometimes combined into a multi-bit signal. Quartus II schematic tool requires that a multi-bit signal and each net connected to it through a bus are all clearly labelled. One clean way to do this is to right-click on the segment you need to name and select **Properties**. Below is an example of multi-bit signal connectivity. As observed in the figure, different bits making up the signal **`Out [1..0]`** are sourced from different single-bit signals.
The Figure 8 above is used as an example to explain multi-bit entry. Note that you don't need to necessarily connect the wires to the buses which is much easier than connecting them.

2.5 EXPERIMENTAL WORK

2.5.1 EXPERIMENTAL SETUP

- Verify to make sure your workbench has all of the following items:
- A Personal Computer (PC) with Altera Quartus II ISE 13.0 Service pack 1 Project Navigator
- DEO Demo Board with Cyclone III EP3C16F484C6 FPGA installed on a card with 10 input toggle switches, three pushbuttons, and four 7-Segment LED displays, among other components.
- A cable connected between the demo board and the PC using USB interface in order to transfer design information from the PC to the Demo Board.

2.5.2 4-BIT ARITHMETIC UNIT WITH DECODER

2.5.2.1 ADDER/SUBTRACTOR

1. Refer to your Modelsim® tutorial and create a project in ModelSim®**,** simulate your Verilog design and the testbench and check your waveform to verify the functionality of your design.
    a. When defining the test bench waveform for functional simulation, make sure the test is long enough. Carefully define your vectors for your inputs and run exhaustive tests to test each possible combination.
2. **Demonstrate the functionality of your design in ModelSim® to the lab instructor before proceeding.**
3. Launch Quartus II Project Navigator
4. Create a project named **<your_name>_addsub**, Copy-Paste your Verilog codes from ModelSim®.
5. Compile your code.
6. Assign package pins and implement your design. When doing this, assign the clock input to a push button.
7. Generate the programming file, upload your design to the onboard FPGA, and test using the toggle switches to control all inputs except the clock, which is controlled by a push-button.
8. **Demonstrate the hardware-tested functionality of your design to the lab instructor.**

2.5.2.2 DECODER

1. Refer to your Modelsim® tutorial and create a project in ModelSim®**,** simulate your Verilog design and the testbench and check your waveform to verify the functionality of your design.
    a. When defining the test bench waveform for functional simulation, make sure the test is long enough. Carefully define your vectors for your inputs and run exhaustive tests to test each possible combination.
2. **Demonstrate the functionality of your design in ModelSim® to the lab instructor before proceeding.**
3. Launch Quartus II Project Navigator.
4. Create a project named **<your_name>_decoder**, Copy-Paste your Verilog codes from ModelSim®.
5. Compile your code.
6. Assign package pins and implement your design. When doing this, assign the clock input to a push button.
7. Generate the programming file, upload your design to the onboard FPGA, and test using the toggle switches to control all inputs except the clock, which is controlled by a push-button.
8. **Demonstrate the hardware-tested functionality of your design to the lab instructor.**

2.5.2.3 ARITHMETIC UNIT

1. Refer to your Modelsim® tutorial and create a project in ModelSim®**,** simulate your Verilog design and the testbench and check your waveform to verify the functionality of your design.
    a. When defining the test bench waveform for functional simulation, make sure the test is long enough. Carefully define your vectors for your inputs and run exhaustive tests to test each possible combination.
2. **Demonstrate the functionality of your design in ModelSim® to the lab instructor before proceeding.**
3. Launch Quartus II Project Navigator.
4. Create a project named **<your_name>_arithmetic_unit**, enter the complete combinational design using **Schematic Capture, simulate**, download program to FPGA and test.
    a. Refer to section 2.4.2 to pull in all the previous design modules into the current project and create a symbol for each module (add/sub, comparator and decoder).
    b. Instantiate the symbols for the design modules, interconnect and assign I/O ports. Pay attention to multi-bit vs. single-bit wires and naming conventions as described in 2.4.3
    c. Ensure in functional and timing simulations that your design works correctly before programming to the FPGA. Then program, and test the same vectors as in (iii) above to make sure you get the same results as your functional simulations. If you do not have a sufficient number of switches to implement your inputs, remember you can also take advantage of push buttons on DE0 board as inputs.
    d. **Demonstrate** the functional simulation and hardware test results from above to your lab instructor, **after completing all of the steps up to** Error! Reference source not found.

2.5.3 4-BIT LOGIC UNIT WITH MUXES

1. Refer to your Modelsim® tutorial and create a project in ModelSim®**,** simulate your Verilog design and the testbench and check your waveform to verify the functionality of your design.
    a. When defining the test bench waveform for functional simulation, make sure the test is long enough. Carefully define your vectors for your inputs and run exhaustive tests to test each possible combination.
2. **Demonstrate the functionality of your design in ModelSim® to the lab instructor before proceeding.**
3. Launch Quartus II Project Navigator.
4. Create a project named **<your_name>_logic_unit**, enter the complete combinational design using **Schematic Capture, simulate**, download program to FPGA and test.
    a. Refer to section 2.4.2 to pull in all the previous design modules into the current project and create a symbol for each module.
    b. Instantiate the symbols for the design modules, interconnect and assign I/O ports. Pay attention to multi-bit vs. single-bit wires and naming conventions as described in 2.4.3
    c. Ensure in functional and timing simulations that your design works correctly before programming to the FPGA. Then program and test the same vectors as in (iii) above to make sure you get the same results as your functional simulations. If you do not have a sufficient number of switches to implement your inputs, remember you can also take advantage of push buttons on DE0 board as inputs.
    d. **Demonstrate** the functional simulation and hardware test results from above to your lab instructor, **after completing all of the steps up to** Error! Reference source not found.

2.5.4 4-BIT ALU TOP-LEVEL

1. Refer to your Modelsim® tutorial and create a project in ModelSim®**,** simulate your Verilog design and the testbench and check your waveform to verify the functionality of your design.
    a. When defining the test bench waveform for functional simulation, make sure the test is long enough. Carefully define your vectors for your inputs and run exhaustive tests to test each possible combination.
2. **Demonstrate the functionality of your design in ModelSim® to the lab instructor before proceeding.**
3. Launch Quartus II Project Navigator.
4. Create a project named **<your_name>_ALU**, enter the complete combinational design using **Schematic Capture, simulate**, download program to FPGA and test.
    a. Refer to section 2.4.2 to pull in all the previous design modules into the current project and create a symbol for each module.
    b. Instantiate the symbols for the design modules, interconnect and assign I/O ports. Pay attention to multi-bit vs. single-bit wires and naming conventions as described in 2.4.3
    c. Ensure in functional and timing simulations that your design works correctly before programming to the FPGA. Then program and test the same vectors as in (iii) above to make sure you get the same results as your functional simulations. If you do not have a sufficient number of switches to implement your inputs, remember you can also take advantage of push buttons on DE0 board as inputs.
    d. **Demonstrate** the functional simulation and hardware test results from above to your lab instructor, **after completing all of the steps up to** Error! Reference source not found.

2.6 REFERENCES

- DE0 Board User Manual, Terasic Technologies Inc.
- Digital Design with An Introduction to the Verilog HDL, 5th Edition, M. Morris Mano & Michael D. Ciletti, Pearson
- Quartus II Introduction Using Schematic Design, ALTERA.