**Lab 2 - The only time you Have to do math**

Embedded Systems

Lab Report 2

Anthony Lau

March 7, 2019

**Purpose:**

In this lab we are going to look at a 4-bit ALU. The first function that we will look at and understand is the ripple carry adder, when can be made from multiple single bit full adders in a structural model in VHDL. Next we will create a 16 function 4-bit ALU behaviorally.

**Pre-Lab:**

**A close up of text on a white background

Description automatically generated**

1. **Back to Digital Logic Design**

**Theory:**

In this part of the lab we are making a 4-bit ripple carry adder from a 4 single bit full adders. We can construct the truth table for a single bit full adder to write the logic eequations for the sum and the carry out. Once we know the equations we can write the VHDL to model the full adder and instantiate it multiple times to create a ripple adder. The ripple adders carry in will be the carry out of the previous full adder.

**Truth Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **S** | **Cout** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **1** | **0** |
| **1** | **0** | **0** | **1** | **0** |
| **1** | **1** | **0** | **0** | **1** |
| **0** | **0** | **1** | **1** | **0** |
| **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** |

**Schematic Diagram:**

**Full Adder**

**A close up of a map

Description automatically generated**

**Ripple Adder**

**A picture containing screenshot

Description automatically generated**

**Design:**

**Full Adder**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **use** **IEEE.NUMERIC\_STD.ALL**;  **entity** **full\_adder** **is**  **Port** (a, b, c\_in: **in** std\_logic;  sum, cout: **out** std\_logic);  **end** **full\_adder**;  **architecture** **Behavioral** **of** **full\_adder** **is**  **begin**  sum <= a **xor** b **xor** c\_in;  cout <= ((a **xor** b) **and** c\_in) **or** (a **and** b);  **end** **Behavioral**; |

**Ripple Adder**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **use** **IEEE.NUMERIC\_STD.ALL**;  **entity** **ripple\_adder** **is**  **port**(a, b: **in** std\_logic\_vector(3 **downto** 0);  c: **in** std\_logic;  s: **out** std\_logic\_vector(3 **downto** 0);  cout\_final: **out** std\_logic);  **end** **ripple\_adder**;  **architecture** **four\_bit** **of** **ripple\_adder** **is**  **component** **full\_adder** **is**  **Port** (a, b, c\_in: **in** std\_logic;  sum, cout: **out** std\_logic);  **end** **component**;  **signal** c1 : std\_logic;  **signal** c2 : std\_logic;  **signal** c3 : std\_logic;  **begin**  fa0: full\_adder  **port** **map**(a => a(0),  b => b(0),  c\_in => c,  sum => s(0),  cout => c1);  fa1: full\_adder  **port** **map**(a => a(1),  b => b(1),  c\_in => c1,  sum => s(1),  cout => c2);  fa2: full\_adder  **port** **map**(a => a(2),  b => b(2),  c\_in => c2,  sum => s(2),  cout => c3);  fa3: full\_adder  **port** **map**(a => a(3),  b => b(3),  c\_in => c3,  sum => s(3),  cout => cout\_final);  **end** **four\_bit**; |

**Test:**

**Testbench**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **use** **IEEE.NUMERIC\_STD.ALL**;  **entity** **ripple\_adder\_tb** **is**  **end** **ripple\_adder\_tb**;  **architecture** **testbench** **of** **ripple\_adder\_tb** **is**  **signal** tb\_a, tb\_b: std\_logic\_vector(3 **downto** 0);  **signal** tb\_sum: std\_logic\_vector(3 **downto** 0);  **signal** tb\_cout, tb\_c: std\_logic;  **component** **ripple\_adder** **is**  **port**(a, b: **in** std\_logic\_vector(3 **downto** 0);  s: **out** std\_logic\_vector(3 **downto** 0);  c: **in** std\_logic;  cout\_final: **out** std\_logic);  **end** **component**;    **begin**  generate\_inputs: **process**  **begin**  tb\_c <= '0';  tb\_a <= "0000";  tb\_b <= "0001";  **wait** **for** 100 ns;    tb\_c <= '0';  tb\_a <= "0101";  tb\_b <= "0011";  **wait** **for** 100 ns;    tb\_c <= '0';  tb\_a <= "0010";  tb\_b <= "1000";  **wait** **for** 100 ns;  tb\_c <= '0';  tb\_a <= "1100";  tb\_b <= "0001";  **wait** **for** 100 ns;      tb\_c <= '0';  tb\_a <= "0110";  tb\_b <= "0101";  **wait** **for** 100 ns;    tb\_c <= '0';  tb\_a <= "0111";  tb\_b <= "1010";  **wait** **for** 100 ns;  **end** **process** **generate\_inputs**;    dut: ripple\_adder  **port** **map**(a => tb\_a,  c => tb\_c,  b => tb\_b,  s => tb\_sum,  cout\_final => tb\_cout);    **end** **testbench**; |

**Simulation**

**A close up of a green screen

Description automatically generated**

1. **Somebody did the work already**

**Theory:**

In VHDL, many basic arithmetic and logical functions are already defined and synthesizable without us having to make our own hardware implementation. For this part we are going to take advantage of that t0 create a 4-bit 16 function ALU. The ALU will take 3 4-bit inputs A, B and opcode from the switches on the board and perform the operation based on the opcode. The result should be displayed on the leds.

**Schematic Diagram:**

**A close up of a map

Description automatically generated**

**Design:**

**Debounce**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **entity** **debounce** **is**  **Port** (clk, btn: **in** std\_logic;  dbnc: **out** std\_logic);  **end** **debounce**;  **architecture** **Behavioral** **of** **debounce** **is**  **signal** counter: std\_logic\_vector(21 **downto** 0);  **signal** count\_set: std\_logic;  **signal** shift\_register: std\_logic\_vector(1 **downto** 0);  **begin**  **process**(clk)  **begin**  count\_set <= shift\_register(1) **xor** shift\_register(0);  **if**(rising\_edge(clk)) **then**  shift\_register(0) <= btn;  shift\_register(1) <= shift\_register(0);  **if**(count\_set = '1') **then**  counter <= (**others** => '0');  **elsif**(counter(21) = '0') **then**  counter <= std\_logic\_vector(unsigned(counter) + 1);  **else**  dbnc <= shift\_register(1);  **end** **if**;  **end** **if**;  **end** **process**;  **end** **Behavioral**; |

**My\_ALU**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **use** **IEEE.STD\_LOGIC\_unsigned.ALL**;  **use** **IEEE.NUMERIC\_STD.ALL**;  **entity** **my\_alu** **is**  **Port**(clk, ld\_A, ld\_B, ld\_op, reset: **in** std\_logic;  A, B, opcode: **in** std\_logic\_vector(3 **downto** 0);  alu\_out: **out** std\_logic\_vector(3 **downto** 0));  **end** **my\_alu**;  **architecture** **Behavioral** **of** **my\_alu** **is**  **begin**  **process**(A, B, opcode, reset)  **begin**    **case**(opcode) **is**  **when** "0000" => alu\_out <= A + B;  **when** "0001" => alu\_out <= A - B;  **when** "0010" => alu\_out <= A + 1;  **when** "0011" => alu\_out <= A - 1;  **when** "0100" => alu\_out <= 0 - A;  **when** "0101" =>  **if**(A > B) **then**  alu\_out <= "0001";  **else**  alu\_out <= "0000";  **end** **if**;  **when** "0110" => alu\_out <= A(2 **downto** 0) & '0';  **when** "0111" => alu\_out <= '0' & A(3 **downto** 1);  **when** "1000" => alu\_out <= A(3) & A(3 **downto** 1);  **when** "1001" => alu\_out <= **not** A;  **when** "1010" => alu\_out <= A **and** B;  **when** "1011" => alu\_out <= A **or** B;  **when** "1100" => alu\_out <= A **xor** B;  **when** "1101" => alu\_out <= A **xnor** B;  **when** "1110" => alu\_out <= A **nand** B;  **when** "1111" => alu\_out <= A **nor** B;  **end** **case**;  **end** **process**;  **end** **Behavioral**; |

**ALU\_Tester**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | **library** **IEEE**;  **use** **IEEE.STD\_LOGIC\_1164.ALL**;  **entity** **alu\_tester** **is**  **Port** (btn, sw: **in** std\_logic\_vector(3 **downto** 0);  clk: std\_logic;  led: **out** std\_logic\_vector(3 **downto** 0));  **end** **alu\_tester**;  **architecture** **Behavioral** **of** **alu\_tester** **is**  *----------------------Intermediate Signals------------------------------*  **signal** debounced: std\_logic\_vector(3 **downto** 0);  **signal** temp\_A, temp\_b, temp\_op, temp\_reset: std\_logic\_vector(3 **downto** 0);  *--------------------ALU Component-------------------------------*  **component** **my\_alu** **is**  **Port** (clk, ld\_A, ld\_B, ld\_op, reset: **in** std\_logic;  A, B, opcode: **in** std\_logic\_vector(3 **downto** 0);  alu\_out: **out** std\_logic\_vector(3 **downto** 0));  **end** **component**;  *---------------------Debounce Component---------------------------*  **component** **debounce** **is**  **Port** (clk, btn: **in** std\_logic;  dbnc: **out** std\_logic);  **end** **component**;  **begin**  **Process**(clk)  **begin**  **if**(rising\_edge(clk)) **then**  **if**(debounced(3) = '1') **then**  temp\_a <= "0000";  temp\_b <= "0000";  temp\_op <= "0000";  **elsif**(debounced(0) = '1') **then**  temp\_b <= sw(3 **downto** 0);  **elsif**(debounced(1) = '1') **then**  temp\_a <= sw(3 **downto** 0);  **elsif**(debounced(2) = '1') **then**  temp\_op <= sw(3 **downto** 0);  **end** **if**;  **end** **if**;  **end** **process**;    Button\_Debounce0: debounce  **port** **map**(clk => clk,  btn => btn(0),  dbnc => debounced(0));  Button\_Debounce1: debounce  **port** **map**(clk => clk,  btn => btn(1),  dbnc => debounced(1));    Button\_Debounce2: debounce  **port** **map**(clk => clk,  btn => btn(2),  dbnc => debounced(2));  Button\_Debounce3: debounce  **port** **map**(clk => clk,  btn => btn(3),  dbnc => debounced(3));  ALU: my\_alu  **port** **map**(clk => clk,  ld\_A => debounced(0),  ld\_B => debounced(1),  ld\_op => debounced(2),  reset => debounced(3),  A(3 **downto** 0) => temp\_A(3 **downto** 0),  B(3 **downto** 0) => Temp\_B(3 **downto** 0),  opcode(3 **downto** 0) => Temp\_op(3 **downto** 0),  alu\_out(3 **downto** 0) => led(3 **downto** 0));  **end** **Behavioral**; |

**Implementation**

**Elaboration Schematic**

**A screenshot of a cell phone

Description automatically generated**

**Synthesis Schematic**

**A close up of a map

Description automatically generated**

**Project Summary**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**XDC File**

## This file is a general .xdc for the ZYBO Rev B board

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used signals according to the project

##Clock signal

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L11P\_T1\_SRCC\_35 Sch=sysclk

create\_clock -add -name sys\_clk\_pin -period 8.00 -waveform {0 4} [get\_ports { clk }];

##Switches

set\_property -dict { PACKAGE\_PIN G15 IOSTANDARD LVCMOS33 } [get\_ports { sw[0] }]; #IO\_L19N\_T3\_VREF\_35 Sch=SW0

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { sw[1] }]; #IO\_L24P\_T3\_34 Sch=SW1

set\_property -dict { PACKAGE\_PIN W13 IOSTANDARD LVCMOS33 } [get\_ports { sw[2] }]; #IO\_L4N\_T0\_34 Sch=SW2

set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { sw[3] }]; #IO\_L9P\_T1\_DQS\_34 Sch=SW3

##Buttons

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { btn[0] }]; #IO\_L20N\_T3\_34 Sch=BTN0

set\_property -dict { PACKAGE\_PIN P16 IOSTANDARD LVCMOS33 } [get\_ports { btn[1] }]; #IO\_L24N\_T3\_34 Sch=BTN1

set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { btn[2] }]; #IO\_L18P\_T2\_34 Sch=BTN2

set\_property -dict { PACKAGE\_PIN Y16 IOSTANDARD LVCMOS33 } [get\_ports { btn[3] }]; #IO\_L7P\_T1\_34 Sch=BTN3

##LEDs

set\_property -dict { PACKAGE\_PIN M14 IOSTANDARD LVCMOS33 } [get\_ports { led[0] }]; #IO\_L23P\_T3\_35 Sch=LED0

set\_property -dict { PACKAGE\_PIN M15 IOSTANDARD LVCMOS33 } [get\_ports { led[1] }]; #IO\_L23N\_T3\_35 Sch=LED1

set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { led[2] }]; #IO\_0\_35=Sch=LED2

set\_property -dict { PACKAGE\_PIN D18 IOSTANDARD LVCMOS33 } [get\_ports { led[3] }]; #IO\_L3N\_T0\_DQS\_AD1N\_35 Sch=LED3

In my XDC file I uncommented all the buttons, switches, and leds. I had to rename the port names with the outputs that I have in my VHDL code.

**Discussion:**

Overall, this lab was not too difficult to figure out. What I learned was that the std\_logic\_unsigned library made it easier to deal with any logic type conversions and allowed me to perform operations directly on the inputs. When there is a negative input, the result shows the two’s compliment of the negative value. The concepts that I completely understand is how a ripple adder can be made of multiple full adders, and how to implement a 16 function ALU.