EEE102-02 Lab 4 Report:

Arithmetic Logic Unit

Kaan Ermertcan - 22202823

23.10.2023

Purpose:

In this lab, we will design an arithmetic logic unit with VHDL and implement it on a Basys 3 board. We will use switches as inputs and LEDs as outputs. The ALU has at least 8 functions, including addition and subtraction.

Design Specifications:

I wanted to design an 8-bit ALU with 16 functions. (Which is more than necessary for this lab.) All 16 switches on Basys3 are used as the two 8-bit inputs, and 5 push-buttons are used for operation selection. 8 LEDs are used as the output and 1 LED is used as an overflow indicator for addition/subtraction module. Overflow indicator lights up when overflow has happened. All switches and LEDs are interpreted as: Switch - Down:0/Up:1, LED – Illuminated:1/OFF:0

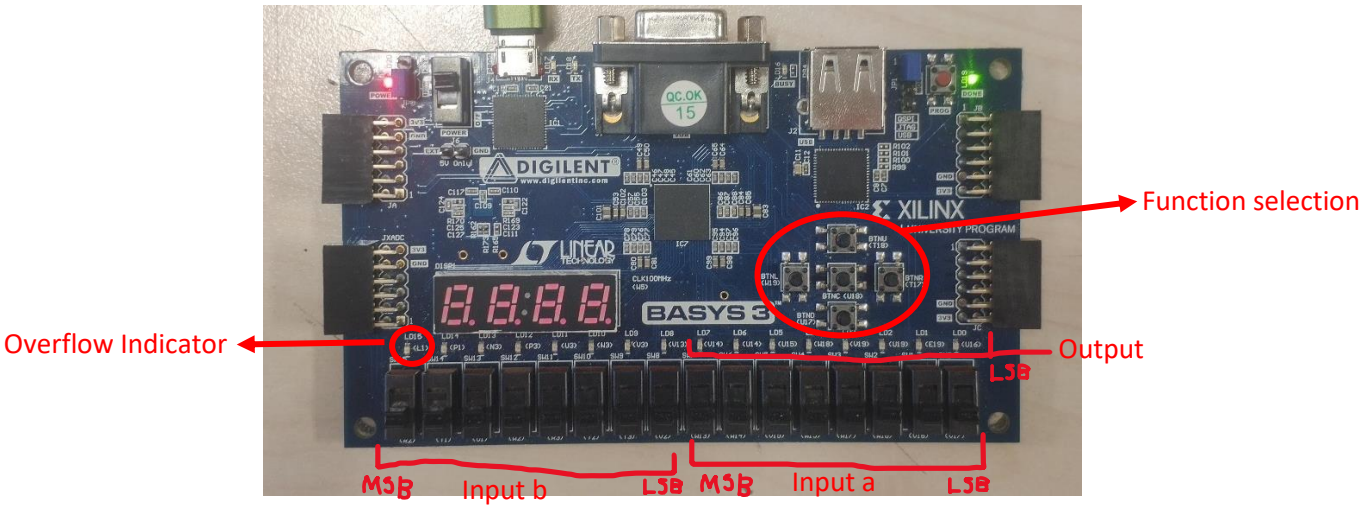

Figure 1.1: Inputs and outputs.

For operations requiring a single input, only the Input a is taken as the operand. To select the operation All 0, 1 and 2 combinations of the buttons are mapped to an operation. To activate the operation, all buttons in the combination must be held in pressed position. The following button mapping is used for

operation selection. Remaining button combinations results in output 00000000. All implemented operations are also listed below.

Button combination for activation	Buttons vector (BTNL, BTNC, BTNR, BTNU, BTND)	Operation
BTNC	01000	Signed Addition
BTNR	00100	Signed Subtraction
BTNU	00010	Signed Increment
BTND	00001	Signed Decrement
BTNL	10000	Negate
None	00000	Passthrough
BTNL and BTNU	10010	Logical Left Bit Shift
BTNR and BTNU	00110	Logical Right Bit Shift
BTNL and BTND	10001	Arithmetic Left Bit Shift
BTNR and BTND	00101	Arithmetic Right Bit Shift
BTNL and BTNC	11000	Rotate Left Bit Shift
BTNC and BTNR	01100	Rotate Right Bit Shift
BTNL and BTNR	10100	One's Complement
BTNU and BTND	00011	Bitwise AND
BTNC and BTNU	01010	Bitwise OR
BTNC and BTND	01001	Bitwise XOR

Table 1.1: Operation list and selection.

Methodology:

The design is implemented in Vivado in a modular structure using components. The module structure tree of the design design can be written as:

Top module (main.vhd)

- Arithmetic Operations (adder_8bit.vhd)
    - One's Complement (ones_complement.vhd) (for subtraction)
    - Full Adder (full_adder.vhd)
        - Half Adder (half_adder.vhd)
- Logical Left Shift (logical_shift_left.vhd)
- Logical Right Shift (logical_shift_right.vhd)
- Arithmetic Right Shift (arithmetic_shift_right.vhd)
- Rotate Left Shift (rotate_left.vhd)
- Rotate Right Shift (rotate_right.vhd)
- One's Complement (ones_complement.vhd)
- Bitwise AND (bitwise_and.vhd)
- Bitwise OR (bitwise_or.vhd)
- Bitwise XOR (bitwise_xor.vhd)

For all arithmetic operations (Add, subtract, increment, decrement, negate), the same module (adder_8bit.vhd) is used together with different inputs switched by a demultiplexer and an extra input. (i.e., add_sub signal determines addition/subtraction to be applied, decrement is equivalent to subtraction of 1 from input a, etc.) Demux is created with a case statement inside a process. All arithmetic inputs are assumed to be 8-bit signed binary numbers. As arithmetic left shift is the same as logical left shift, the same module is used for both operations. Outputs of all modules (result vectors) are switched with a multiplexer controlled by buttons vector. Multiplexer is created with a select statement. Generate statements and loops are also researched and used to improve the VHDL code (i.e., when chaining 8 full adders to create 8-bit adder, generating bitwise operations). In operation modules, primary input is denoted by the logic vector a, and secondary input is denoted by the vector b. Output of the module is denoted by result vector (denoted sum in adder_8bit.vhd).

After writing the modules, a testbench code (named xxx_test.vhd where xxx is the name of the tested module) is written for each module to test it. Two different test strategies are used depending on the module. If the number of possible different inputs were too many (as there are 256 different inputs for an 8-bit input), then a pseudo-random number generator is used inside a for loop to generate 10 different inputs. Otherwise, all possible inputs are iterated one by one using for loops. For the main module, predetermined values are used for input a and b as only switching is done in main module. It is tested only for all different possible button combinations (or all selections of operations).

Results:

All 16 functions and 13 modules are successfully implemented on Basys 3 and tested using testbenches. RTL schematics of the main module and arithmetic module are in the figures below.

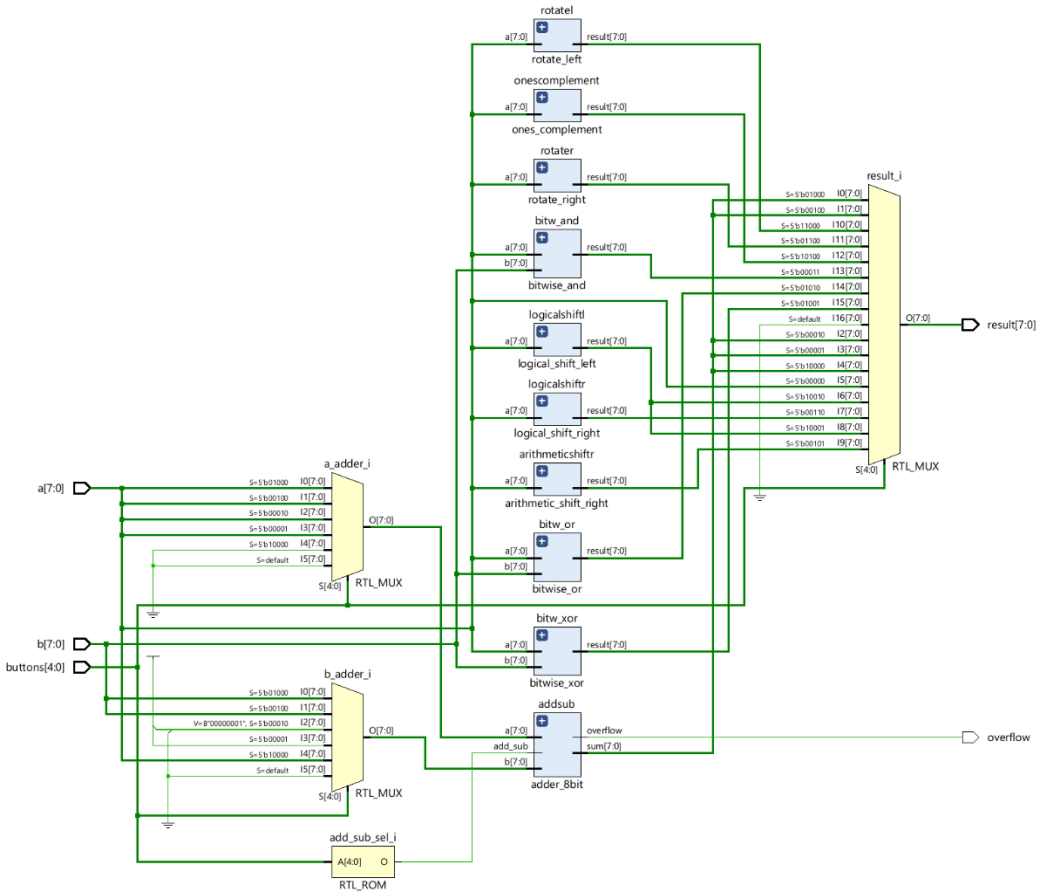

Figure 2.1: RTL schematic of the main module.

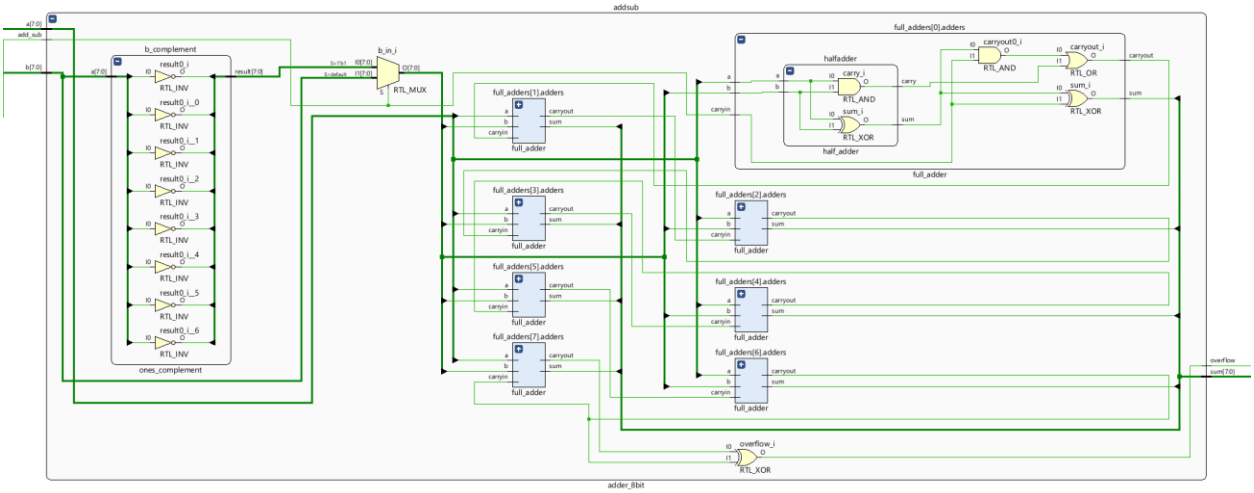

Figure 2.2: RTL schematic of the arithmetic module.

As can be seen from the schematics, the design consists of multiple multiplexers for switching inputs and outputs and a lot of logic gates. Bit shifters' and bitwise operations' schematics were not included as they are trivial.

Simulation results from the main module with inputs a=10101010, b=00110100 can be found below. Other modules' simulation figures are in appendices.

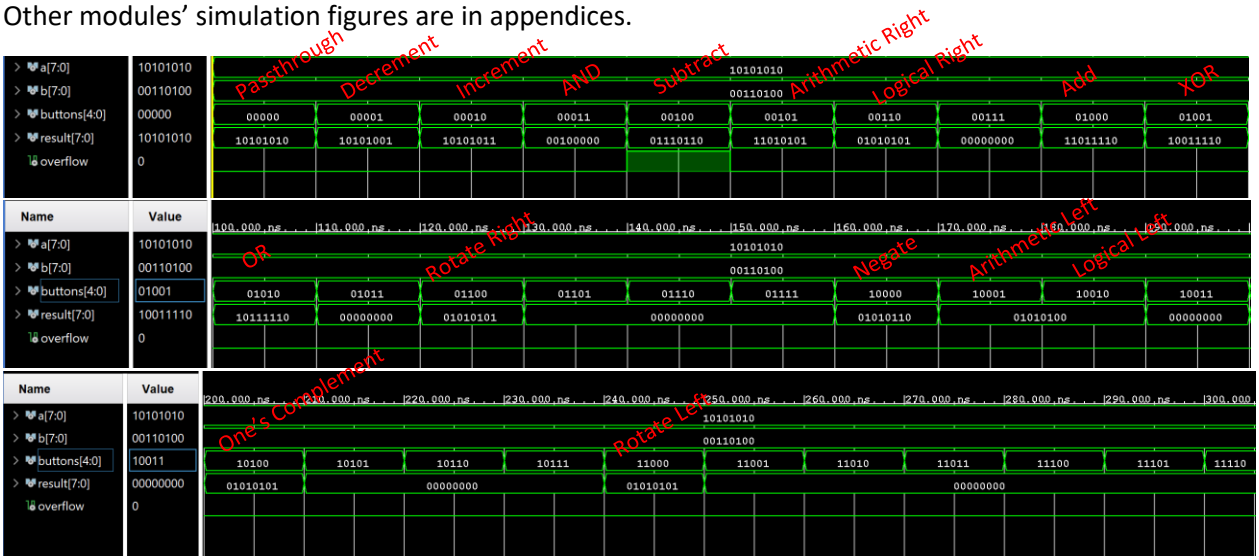

Figure 2.3: Simulation waveform of the main module.

Same inputs (a=10101010, b=00110100) are tested on the Basys 3 to compare the simulation with physical results. The sample figures can be found below:

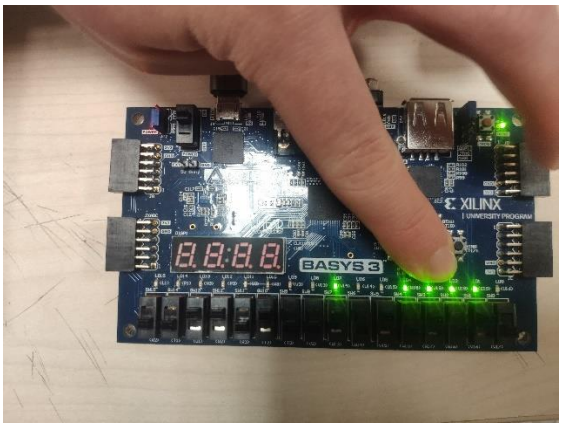

Figure 3.1: Bitwise XOR.

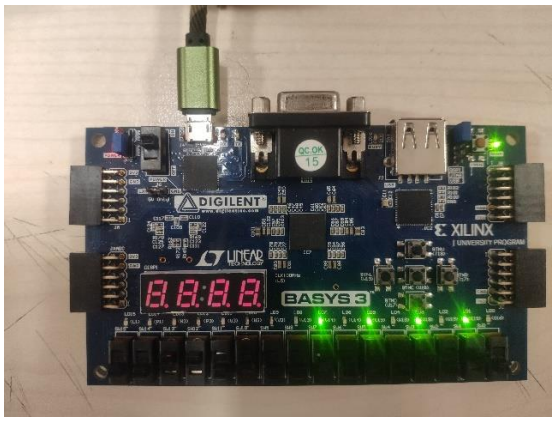

Figure 3.2: Passthrough.

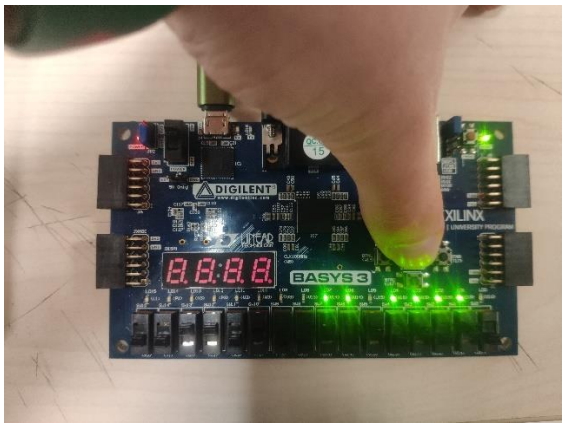

Figure 3.3: Addition.

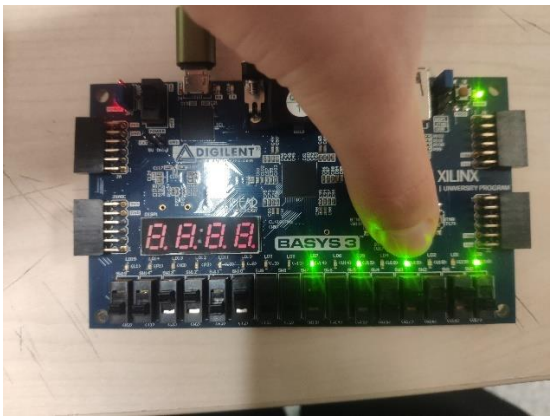

Figure 3.4: Decrement.

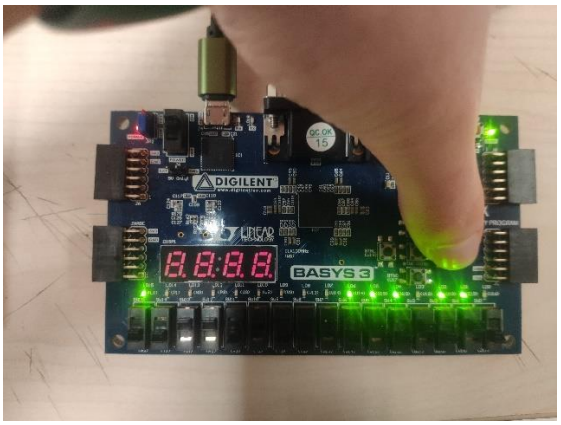

Figure 3.5: Subtraction.

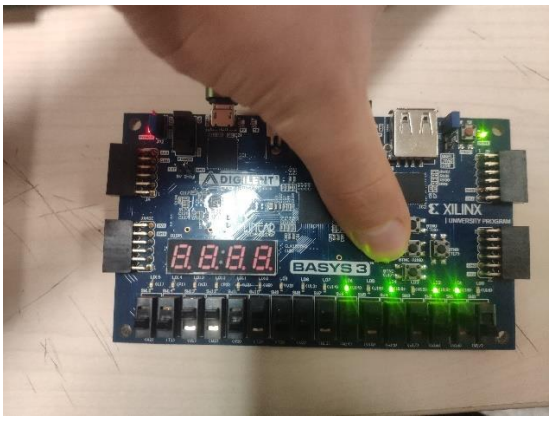

Figure 3.6: Negate.

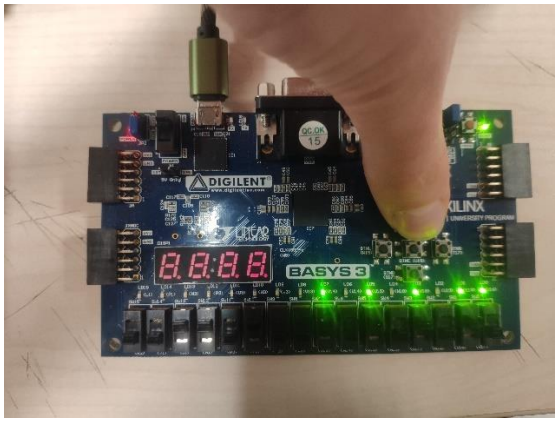

Figure 3.7: Increment.

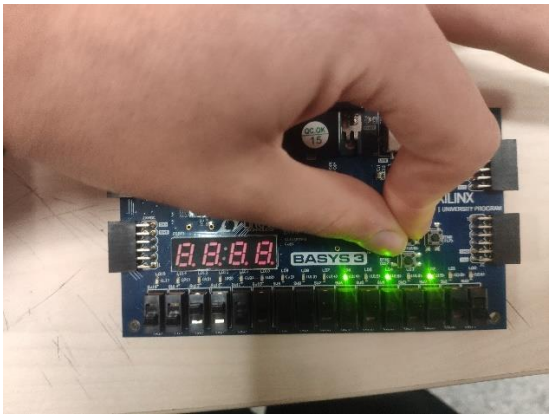

Figure 3.8: Logical Left Shift.

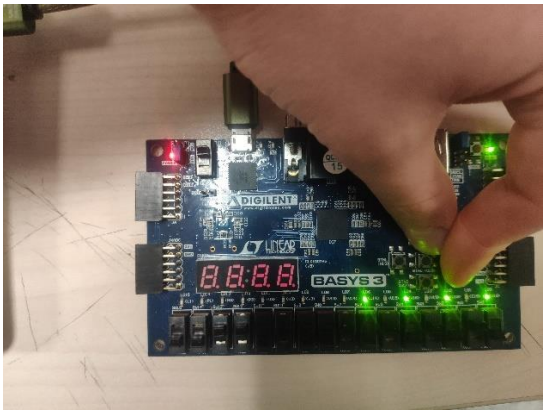

Figure 3.9: Logical Right Shift.

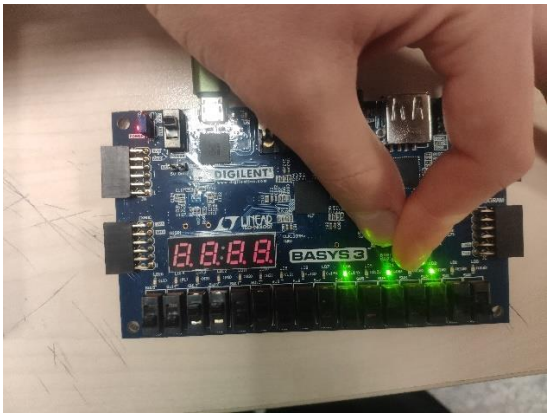

Figure 3.1: Arithmetic Left Shift.

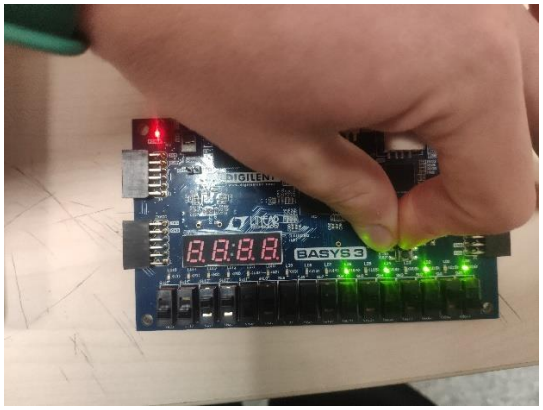

Figure 3.1: Rotate Left.

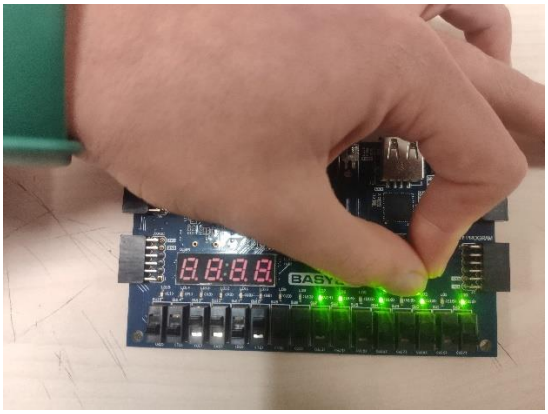

Figure 3.1: Arithmetic Right Shift.

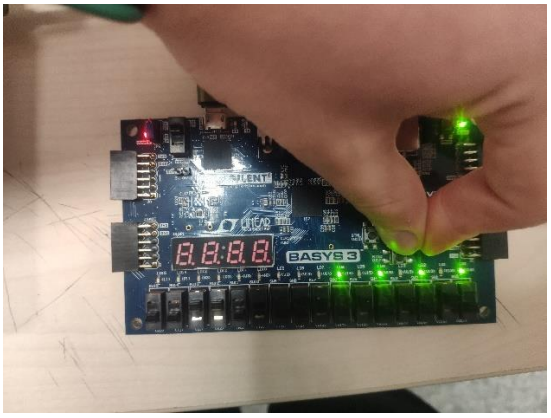

Figure 3.1: Rotate Right.

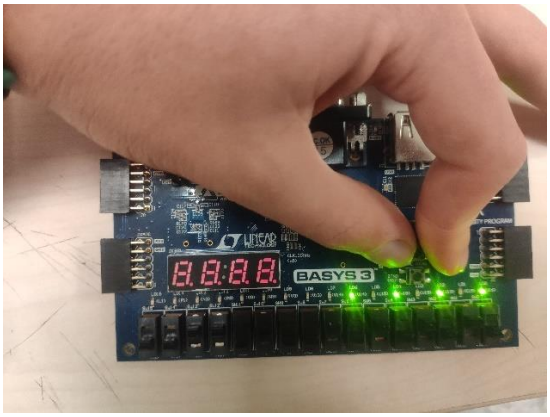

Figure 3.1: One's Complement.

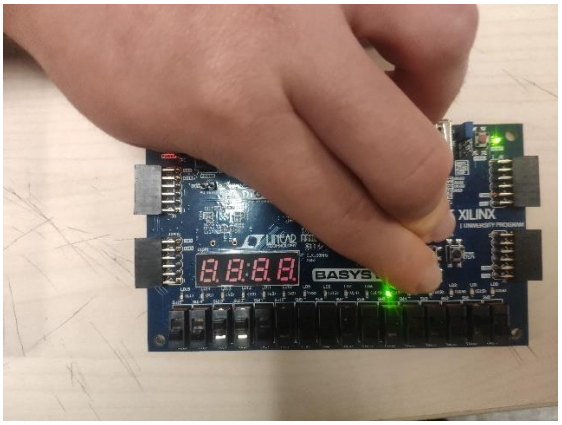

Figure 3.1: Bitwise AND.

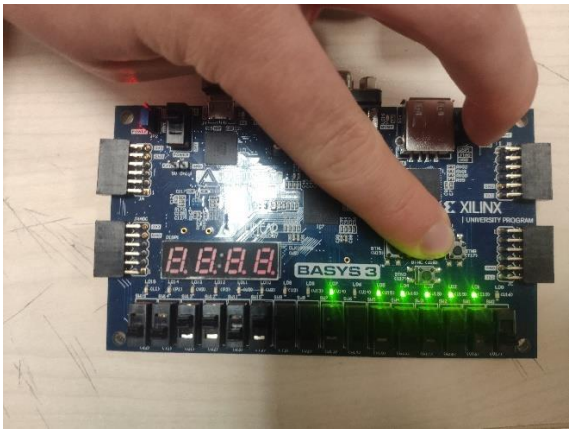

Figure 3.1: Bitwise OR.

As can be seen from the figures, my design works as intended.

Conclusion:

In this lab, I successfully implemented an ALU with a wide range of operations. I got more comfortable with using VHDL and implementing circuits on Basys 3. At the beginning of the lab, I struggled a bit as I was not familiar with VHDL syntax but after some trials and errors, and some research I was able to overcome problems in my code one by one. Implementing my design required me to use what we learned in the lectures about adders/subtractors and signed binary numbers.

Appendices:

Simulation Figures:

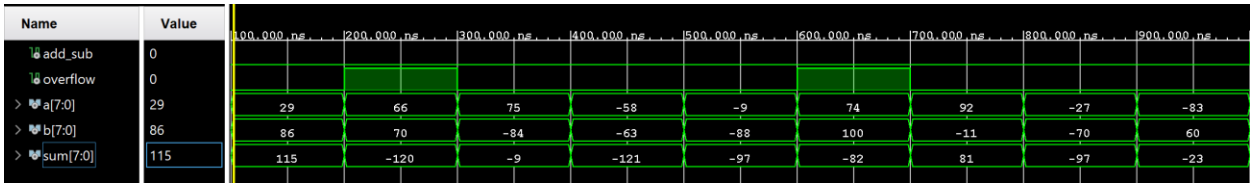

Figure 2.4: adder_8bit_test - Addition.

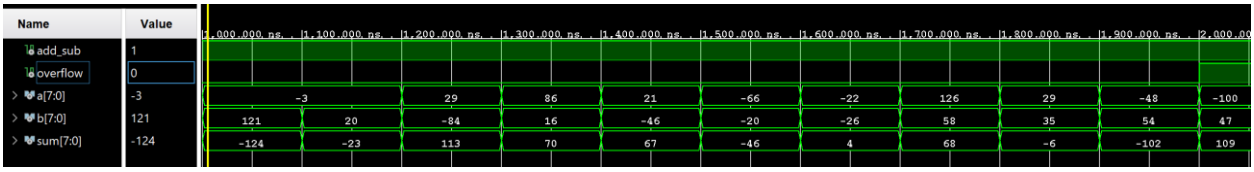

Figure 2.5: adder_8bit_test - Subtraction

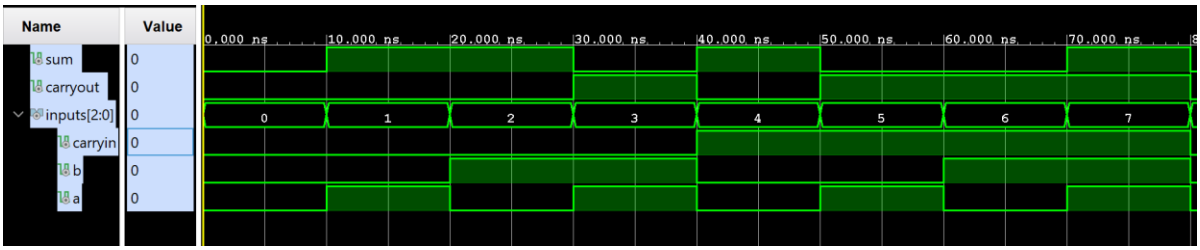

Figure 2.6: full_adder_test.

Figure 2.7: half_adder_test.

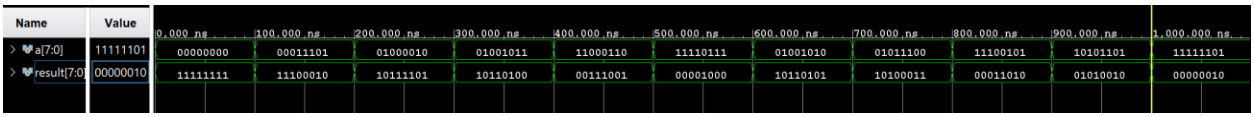

Figure 2.8: ones_complement_test.

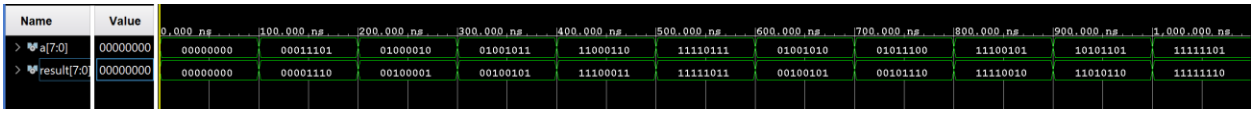

Figure 2.9: arithmetic_right_shift_test.

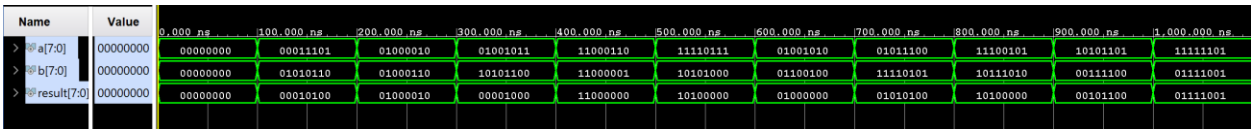

Figure 2.10: bitwise_and_test.

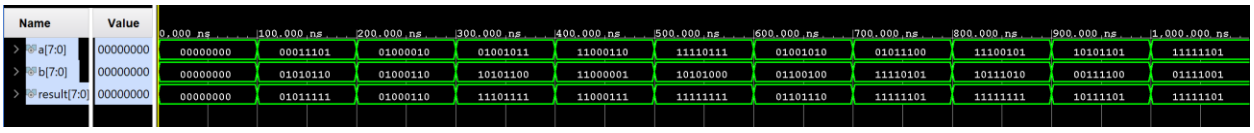

Figure 2.11: bitwise_or_test.

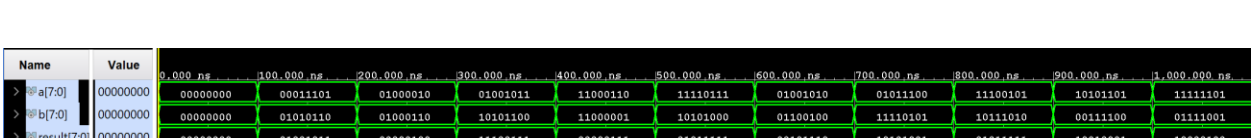

Figure 2.12: bitwise_xor_test.

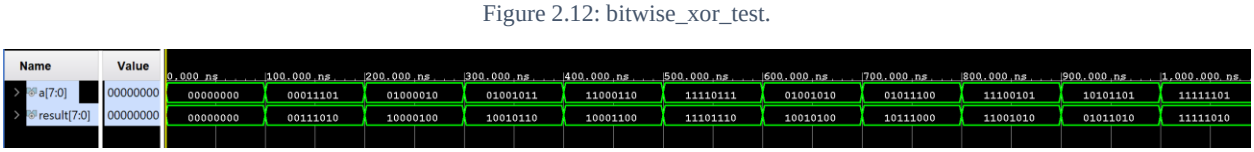

Figure 2.13: logical_left_shift_test.

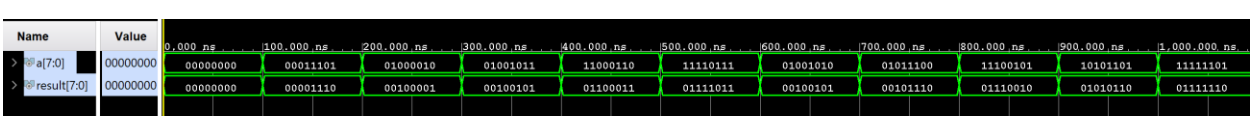

Figure 2.14: logical_right_shift_test.

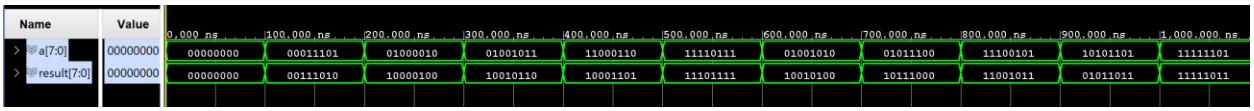

Figure 2.15: rotate_left_test.

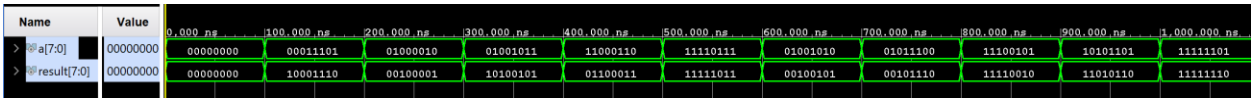

Figure 2.16: rotate_right_test.

VHDL Codes:

Code 1: main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          buttons : in STD_LOGIC_VECTOR (4 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0);
          overflow : out STD_LOGIC);
end main;

architecture Behavioral of main is
signal a_adder : STD_LOGIC_VECTOR (7 downto 0);
signal b_adder : STD_LOGIC_VECTOR (7 downto 0);
signal result_adder : STD_LOGIC_VECTOR (7 downto 0);
signal result_arithmetic_right : STD_LOGIC_VECTOR (7 downto 0);
signal result_logical_right : STD_LOGIC_VECTOR (7 downto 0);
signal result_logical_left : STD_LOGIC_VECTOR (7 downto 0);
signal result_rotate_right : STD_LOGIC_VECTOR (7 downto 0);
signal result_rotate_left : STD_LOGIC_VECTOR (7 downto 0);
signal result_onescomplement : STD_LOGIC_VECTOR (7 downto 0);
signal result_and : STD_LOGIC_VECTOR (7 downto 0);
signal result_or : STD_LOGIC_VECTOR (7 downto 0);
signal result_xor : STD_LOGIC_VECTOR (7 downto 0);
signal add_sub_sel : STD_LOGIC;
component adder_8bit is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          add_sub : in STD_LOGIC;
```

```
        sum : out STD_LOGIC_VECTOR;
        overflow : out STD_LOGIC);
end component;
component ones_complement is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component logical_shift_left is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component logical_shift_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component arithmetic_shift_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component rotate_left is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component rotate_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component bitwise_and is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        b : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component bitwise_or is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        b : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component bitwise_xor is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        b : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
onescomplement: ones_complement port map (a => a,  result => result_onescomplement);
```

```
arithmeticshiftr: arithmetic_shift_right port map (a => a,  result => result_arithmetic_right);
logicalshiftr: logical_shift_right port map (a => a,  result => result_logical_right);
logicalshiftl: logical_shift_left port map (a => a,  result => result_logical_left);
rotater: rotate_right port map (a => a,  result => result_rotate_right);
rotatel: rotate_left port map (a => a,  result => result_rotate_left);
bitw_and: bitwise_and port map (a => a, b => b,  result => result_and);
bitw_or: bitwise_or port map (a => a, b => b,  result => result_or);
bitw_xor: bitwise_xor port map (a => a, b => b,  result => result_xor);
addsub: adder_8bit port map (a => a_adder, b => b_adder,  sum => result_adder, add_sub =>
add_sub_sel, overflow => overflow);

process(buttons) begin
   case buttons is       -- demux for arithmetic inputs
                  when "01000" => a_adder <= a; b_adder <= b; add_sub_sel <= '0';              -- Add
                  when "00100" => a_adder <= a; b_adder <= b; add_sub_sel <= '1';              -- Subtract
                  when "00010" => a_adder <= a; b_adder <= "00000001"; add_sub_sel <= '0';--Increment
                  when "00001" => a_adder <= a; b_adder <= "11111111"; add_sub_sel <= '0';--Decremen
                  when "10000" => a_adder <= "00000000"; b_adder <= a; add_sub_sel <= '1';-- Negate
                  when others  => a_adder <= "00000000"; b_adder <= "00000000"; add_sub_sel <= '0';
          end case;
end process;

with buttons select                                    -- mux for outputs
   result <= result_adder          when "01000",       --Add
          result_adder             when "00100",       --Subtract
          result_adder             when "00010",       --Increment
          result_adder             when "00001",       --Decrement
          result_adder             when "10000",       --Negate
          a                      when "00000",         --Pass Through
          result_logical_left      when "10010",       --Logical Left Shift
          result_logical_right     when "00110",       --Logical Right Shift
          result_logical_left      when "10001",       --Arithmetic Left Shift
          result_arithmetic_right  when "00101",       --Arithmetic Right Shift
          result_rotate_left       when "11000",       --Rotate Left Shift
          result_rotate_right      when "01100",       --Rotate Right Shift
          result_onescomplement    when "10100",       --One's Complement
          result_and               when "00011",       --Bitwise AND
          result_or                when "01010",       --Bitwise OR
          result_xor               when "01001",       --Bitwise XOR
          "00000000"               when others;
end Behavioral;
```

Code 2: adder_8bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity adder_8bit is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          sum : out STD_LOGIC_VECTOR (7 downto 0);
          add_sub : in STD_LOGIC;
          overflow : out STD_LOGIC);
end adder_8bit;

architecture Behavioral of adder_8bit is
component full_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          carryin : in STD_LOGIC;
          sum : out STD_LOGIC;
          carryout : out STD_LOGIC);
end component;

component ones_complement is
   Port ( a : in std_logic_vector (7 downto 0);
          result : out std_logic_vector (7 downto 0));
end component;
signal b_in : std_logic_vector (7 downto 0);
signal b_comp : std_logic_vector (7 downto 0);
signal carry : std_logic_vector (8 downto 0);
begin
with add_sub select
   b_in <= b_comp when '1',
           b when others;
carry(0) <= add_sub;
full_adders: for i in 0 to 7 generate
adders: full_adder port map (a => a(i), b => b_in(i), carryin => carry(i), sum => sum(i), carryout =>
carry(i+1));
end generate;
b_complement: ones_complement port map (a => b,  result => b_comp);
overflow <= carry(8) xor carry(7);
end Behavioral;
```

Code 3: full_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          carryin : in STD_LOGIC;
          sum : out STD_LOGIC;
          carryout : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is
signal half_sum : std_logic;
signal half_carry : std_logic;
component half_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          carry : out STD_LOGIC;
          sum : out STD_LOGIC);
end component;
begin
halfadder: half_adder port map (a => a, b => b, carry => half_carry, sum => half_sum);
sum <= half_sum xor carryin;
carryout <= (half_sum and carryin) or half_carry;
end Behavioral;
```

Code 4: half_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity half_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          carry : out STD_LOGIC;
          sum : out STD_LOGIC);
end half_adder;
```

```
architecture Behavioral of half_adder is
begin
sum <= a xor b;
carry <= a and b;
end Behavioral;
```

Code 5: ones_complement.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity ones_complement is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end ones_complement;

architecture Behavioral of ones_complement is

begin
inverters: for i in 0 to 7 generate
result(i) <= not a(i);
end generate;
end Behavioral;
```

Code 6: arithmetic_shift_right.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity arithmetic_shift_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end arithmetic_shift_right;

architecture Behavioral of arithmetic_shift_right is
begin
shift: for i in 0 to 6 generate
result(i) <= a(i+1);
end generate;
```

```
result(7) <= a(7);
end Behavioral;
```

Code 7: logical_shift_right.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity logical_shift_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end logical_shift_right;

architecture Behavioral of logical_shift_right is
begin
shift: for i in 0 to 6 generate
result(i) <= a(i+1);
end generate;
result(7) <= '0';
end Behavioral;
```

Code 8: logical_shift_left.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity logical_shift_left is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end logical_shift_left;

architecture Behavioral of logical_shift_left is
begin
shift: for i in 0 to 6 generate
result(i+1) <= a(i);
end generate;
result(0) <= '0';
end Behavioral;
```

Code 9: rotate_right.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rotate_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end rotate_right;

architecture Behavioral of rotate_right is
begin
shift: for i in 0 to 6 generate
result(i) <= a(i+1);
end generate;
result(7) <= a(0);
end Behavioral;
```

Code 10: rotate_left.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rotate_left is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end rotate_left;

architecture Behavioral of rotate_left is
begin
shift: for i in 0 to 6 generate
result(i+1) <= a(i);
end generate;
result(0) <= a(7);
end Behavioral;
```

Code 11: bitwise_and.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity bitwise_and is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end bitwise_and;

architecture Behavioral of bitwise_and is
begin
shift: for i in 0 to 7 generate
result(i) <= a(i) and b(i);
end generate;

end Behavioral;
```

Code 12: bitwise_or.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity bitwise_or is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end bitwise_or;

architecture Behavioral of bitwise_or is
begin
shift: for i in 0 to 7 generate
result(i) <= a(i) or b(i);
end generate;

end Behavioral;
```

Code 13: bitwise_xor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity bitwise_xor is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end bitwise_xor;

architecture Behavioral of bitwise_xor is
begin
shift: for i in 0 to 7 generate
result(i) <= a(i) xor b(i);
end generate;

end Behavioral;
```

Code 14: main_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity main_test is
end main_test;

architecture Behavioral of main_test is
signal a : STD_LOGIC_VECTOR (7 downto 0) := "10101010";
signal b : STD_LOGIC_VECTOR (7 downto 0) := "00110100";
signal buttons : STD_LOGIC_VECTOR (4 downto 0) := "00000";
signal result : STD_LOGIC_VECTOR (7 downto 0);
signal overflow : STD_LOGIC;

component main is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        buttons : in STD_LOGIC_VECTOR (4 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0);
        overflow : out STD_LOGIC);
end component;
begin
uut: main
port map (a=>a, b=>b, buttons=>buttons, result=>result, overflow=>overflow);
process begin
   for i in 0 to 31 loop
      wait for 10 ns;
      buttons <= std_logic_vector(unsigned(buttons) + 1);
   end loop;
   wait;
end process;
end Behavioral;
```

Code 15: adder_8bit_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity adder_8bit_test is
end adder_8bit_test;

architecture Behavioral of adder_8bit_test is
signal add_sub : STD_LOGIC := '0';
signal overflow : STD_LOGIC;
signal a : std_logic_vector(7 downto 0):= "00000000";
signal b : std_logic_vector(7 downto 0):= "00000000";
signal sum : std_logic_vector(7 downto 0):= "00000000";

component adder_8bit is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        b : in STD_LOGIC_VECTOR (7 downto 0);
        sum : out STD_LOGIC_VECTOR (7 downto 0);
        add_sub : in STD_LOGIC;
        overflow : out STD_LOGIC);
end component;
begin
```

```
uut: adder_8bit
port map (a=>a, b=>b, add_sub=>add_sub, sum=>sum, overflow=>overflow);
process

variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable seed3: positive :=86898;
variable seed4: positive :=67633;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable rand2: real;
variable int_rand1: integer;      -- random integers in range 0..255
variable int_rand2: integer;
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
variable stim2: std_logic_vector(7 downto 0);
begin
for i in 0 to 1 loop
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
            uniform(seed3, seed4, rand2);
   int_rand1 := integer(trunc(rand1*256.0));            -- rescale to integer between 0-256
   int_rand2 := integer(trunc(rand2*256.0));
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
   stim2 := std_logic_vector(to_unsigned(int_rand2, stim2'length));
            a <= stim1;
            b <= stim2;
         end loop;
         add_sub <= '1';
         end loop;

wait;
end process;
end Behavioral;
```

Code 16: full_adder_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity full_adder_test is
```

```
end full_adder_test;

architecture Behavioral of full_adder_test is
signal sum : STD_LOGIC;
signal carryout : STD_LOGIC;
signal inputs : std_logic_vector(2 downto 0):= "000";
component full_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          carryin : in STD_LOGIC;
          sum : out STD_LOGIC;
          carryout : out STD_LOGIC);
end component;
begin
uut: full_adder
port map (a=>inputs(0), b=>inputs(1), carryin=>inputs(2), carryout=>carryout, sum=>sum);
process begin
   for i in 0 to 7 loop
      wait for 10 ns;
      inputs <= std_logic_vector(unsigned(inputs) + 1);
   end loop;
   wait;
end process;
end Behavioral;
```

Code 17: half_adder_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity half_adder_test is
end half_adder_test;

architecture Behavioral of half_adder_test is
signal sum : STD_LOGIC;
signal carry : STD_LOGIC;
signal inputs : std_logic_vector(1 downto 0):= "00";
component half_adder is
   Port ( a : in STD_LOGIC;
          b : in STD_LOGIC;
          carry : out STD_LOGIC;
          sum : out STD_LOGIC);
```

```
end component;
begin
uut: half_adder
port map (a=>inputs(0), b=>inputs(1), carry=>carry, sum=>sum);
process begin
   for i in 0 to 3 loop
      wait for 10 ns;
      inputs <= std_logic_vector(unsigned(inputs) + 1);
   end loop;
   wait;
end process;
end Behavioral;
```

Code 18: ones_complement_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity ones_complement_test is
end ones_complement_test;

architecture Behavioral of ones_complement_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component ones_complement is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: ones_complement
port map (a=>a, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable rand1: real;              -- random real-numbers in range 0 to 1
variable int_rand1: integer;       -- random integers in range 0..255
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
begin
```

```
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
   int_rand1 := integer(trunc(rand1*256.0));           -- rescale to integer between 0-256
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
            a <= stim1;
         end loop;
wait;
end process;
end Behavioral;
```

Code 19: arithmetic_shift_right_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity arithmetic_shift_right_test is
end arithmetic_shift_right_test;

architecture Behavioral of arithmetic_shift_right_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component arithmetic_shift_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: arithmetic_shift_right
port map (a=>a, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable int_rand1: integer;      -- random integers in range 0..255
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
```

```
   int_rand1 := integer(trunc(rand1*256.0));            -- rescale to integer between 0-256
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
            a <= stim1;
         end loop;
wait;
end process;
end Behavioral;
```

Code 20: logical_shift_right_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity logical_shift_right_test is
end logical_shift_right_test;

architecture Behavioral of logical_shift_right_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component logical_shift_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: logical_shift_right
port map (a=>a, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable int_rand1: integer;      -- random integers in range 0..255
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
   int_rand1 := integer(trunc(rand1*256.0));            -- rescale to integer between 0-256
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
```

```
            a <= stim1;
        end loop;
wait;
end process;
end Behavioral;
```

Code 21: logical_shift_left_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity logical_shift_left_test is
end logical_shift_left_test;

architecture Behavioral of logical_shift_left_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component logical_shift_left is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: logical_shift_left
port map (a=>a, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable int_rand1: integer;      -- random integers in range 0..255
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
   int_rand1 := integer(trunc(rand1*256.0));           -- rescale to integer between 0-256
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
            a <= stim1;
        end loop;
wait;
```

```
end process;
end Behavioral;
```

Code 22: rotate_right_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity rotate_right_test is
end rotate_right_test;

architecture Behavioral of rotate_right_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component rotate_right is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: rotate_right
port map (a=>a, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable int_rand1: integer;      -- random integers in range 0..255
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
   int_rand1 := integer(trunc(rand1*256.0));           -- rescale to integer between 0-256
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
            a <= stim1;
         end loop;
wait;
end process;
end Behavioral;
```

Code 23: rotate_left_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity rotate_left_test is
end rotate_left_test;

architecture Behavioral of rotate_left_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component rotate_left is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: rotate_left
port map (a=>a, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable int_rand1: integer;      -- random integers in range 0..255
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
   int_rand1 := integer(trunc(rand1*256.0));           -- rescale to integer between 0-256
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
            a <= stim1;
         end loop;
wait;
end process;
end Behavioral;
```

Code 24: bitwise_and_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity bitwise_and_test is
end bitwise_and_test;

architecture Behavioral of bitwise_and_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal b : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component bitwise_and is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: bitwise_and
port map (a=>a, b=>b, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable seed3: positive :=86898;
variable seed4: positive :=67633;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable rand2: real;
variable int_rand1: integer;      -- random integers in range 0..255
variable int_rand2: integer;
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
variable stim2: std_logic_vector(7 downto 0);
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);            -- generate random number
            uniform(seed3, seed4, rand2);
   int_rand1 := integer(trunc(rand1*256.0));           -- rescale to integer between 0-256
   int_rand2 := integer(trunc(rand2*256.0));
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
   stim2 := std_logic_vector(to_unsigned(int_rand2, stim2'length));
            a <= stim1;
            b <= stim2;
```

```
        end loop;
wait;
end process;
end Behavioral;
```

Code 25: bitwise_or_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity bitwise_or_test is
end bitwise_or_test;

architecture Behavioral of bitwise_or_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal b : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component bitwise_or is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        b : in STD_LOGIC_VECTOR (7 downto 0);
        result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: bitwise_or
port map (a=>a, b=>b, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable seed3: positive :=86898;
variable seed4: positive :=67633;
variable rand1: real;            -- random real-numbers in range 0 to 1
variable rand2: real;
variable int_rand1: integer;      -- random integers in range 0..255
variable int_rand2: integer;
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
variable stim2: std_logic_vector(7 downto 0);
begin
for j in 0 to 9 loop
            wait for 100 ns;
```

```
            uniform(seed1, seed2, rand1);            -- generate random number
            uniform(seed3, seed4, rand2);
   int_rand1 := integer(trunc(rand1*256.0));            -- rescale to integer between 0-256
   int_rand2 := integer(trunc(rand2*256.0));
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
   stim2 := std_logic_vector(to_unsigned(int_rand2, stim2'length));
            a <= stim1;
            b <= stim2;
         end loop;
wait;
end process;
end Behavioral;
```

Code 26: bitwise_xor_test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
use ieee.numeric_std.all;


entity bitwise_xor_test is
end bitwise_xor_test;

architecture Behavioral of bitwise_xor_test is
signal a : std_logic_vector(7 downto 0):= "00000000";
signal b : std_logic_vector(7 downto 0):= "00000000";
signal result : std_logic_vector(7 downto 0):= "00000000";
component bitwise_xor is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          result : out STD_LOGIC_VECTOR (7 downto 0));
end component;
begin
uut: bitwise_xor
port map (a=>a, b=>b, result=>result);
process
variable seed1: positive :=28938;  -- seed values for random generator
variable seed2: positive :=22380;
variable seed3: positive :=86898;
variable seed4: positive :=67633;
```

```
variable rand1: real;           -- random real-numbers in range 0 to 1
variable rand2: real;
variable int_rand1: integer;      -- random integers in range 0..255
variable int_rand2: integer;
variable stim1: std_logic_vector(7 downto 0);  -- random 8-bit stimuli
variable stim2: std_logic_vector(7 downto 0);
begin
for j in 0 to 9 loop
            wait for 100 ns;
            uniform(seed1, seed2, rand1);           -- generate random number
            uniform(seed3, seed4, rand2);
   int_rand1 := integer(trunc(rand1*256.0));           -- rescale to integer between 0-256
   int_rand2 := integer(trunc(rand2*256.0));
   stim1 := std_logic_vector(to_unsigned(int_rand1, stim1'length)); -- convert to std_logic_vector
   stim2 := std_logic_vector(to_unsigned(int_rand2, stim2'length));
            a <= stim1;
            b <= stim2;
         end loop;
wait;
end process;
end Behavioral;
```

References:

Bhatt, A. (n.d.). *VHDL tutorial 14: Design 1×8 demultiplexer and 8×1 multiplexer using VHDL*. Engineers Garage. https://www.engineersgarage.com/vhdl-tutorial-14-design-1x8-demultiplexer-and-8x1-multiplexer-using-vhdl/

For Loops. VHDL-Online. (n.d.). https://www.vhdl-online.de/courses/system_design/vhdl_language_and_syntax/sequential_statements/for_loops

Generate Statement. HDL Works. (n.d.). https://www.hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/GenerateStatement.htm

Harvey, M. (2015, October 9). *Generating random values in VHDL testbenches*. Mark Harvey. http://www.markharvey.info/rtl/rnd_09.10.2015/rnd_09.10.2015.html