Project 2: Adder/Subtractor Testbench and SLT

Computer Organization CSC 34300 - EF

By: Sebastian Grygorczuk

Tabel of Content:

Objective:

The objective of this lab is to create a test bench program which will check the addition and subtraction of an n bit adder/subtractor for all the 2^{2n} cases. A secondary task of this lab is to create a SLT (Set Less Than) instruction which will result in 1 if A < B and 0 otherwise.

Part 1: Bench Test

Adder Code

Before looking at the test bench let's look at the files used to build the adder/subtractor, to do so we started off with Half Adder, which then built a Full Adder which then those were used to build the N-Bit Adder. Below you can see the builds of the Half Adder, Full Adder, N Bit Adder in code figures 1,2 and 3 respectively.

```
library ieee;
use ieee.std_logic_1164.all;

entity HALF_ADDER is
	port(
		grygorczuk_input_1, grygorczuk_input_2 : in std_logic;
		grygorczuk_output, grygorczuk_carry_out : out std_logic
	);
end HALF_ADDER;

architecture HALF_ADDER_LOGIC of HALF_ADDER is
begin
	grygorczuk_output <= grygorczuk_input_1 xor grygorczuk_input_2;
	grygorczuk_carry_out <= grygorczuk_input_1 and grygorczuk_input_2;
end HALF_ADDER_LOGIC;
```

Code 1. Half Adder VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity FULL_ADDER is
	port(
		grygorczuk_input_3, grygorczuk_input_4, grygorczuk_carry_in: in std_logic;
		grygorczuk_output, grygorczuk_carry_out_2 : out std_logic
	);
end FULL_ADDER;

architecture FULL_ADDER_LOGIC of FULL_ADDER is
signal HALF_ADDER_OUTPUT_1,HALF_ADDER_OUTPUT_2, HALF_ADDER_CARRY_OUT_1,
HALF_ADDER_CARRY_OUT_2 : std_logic;
component HALF_ADDER
port
	(
                                grygorczuk_input_1, grygorczuk_input_2 : in std_logic;
                                grygorczuk_output, grygorczuk_carry_out : out std_logic
                );
end component;

begin
HALF_ADDER_1: HALF_ADDER port map(grygorczuk_input_3, grygorczuk_input_4, HALF_ADDER_OUTPUT_1,
HALF_ADDER_CARRY_OUT_1);
HALF_ADDER_2: HALF_ADDER port map(grygorczuk_carry_in, HALF_ADDER_OUTPUT_1, HALF_ADDER_OUTPUT_2,
HALF_ADDER_CARRY_OUT_2);
grygorczuk_output <= HALF_ADDER_OUTPUT_2;
grygorczuk_carry_out_2 <= HALF_ADDER_CARRY_OUT_1 or HAlF_ADDER_CARRY_OUT_2;
end FULL_ADDER_LOGIC;
```

Code 2. Full Adder VHDL

```
 library ieee;
use ieee.std_logic_1164.all;

entity N_BIT_ADDER is
  generic(
    n: integer := 4
  );
                port(
                                grygorczuk_carry_in_2, grygorczuk_signed : in std_logic;
                                grygorczuk_carry_x, grygorczuk_carry_y: in std_logic_vector(n-1 DOWNTO 0);
                                grygorczuk_overflow_flag, grygorczuk_negative_flag, grygorczuk_zero_flag, grygorczuk_carry_out_flag: out
std_logic;
                                grygorczuk_o: out std_logic_vector(n-1 DOWNTO 0)
                );
end N_BIT_ADDER;

architecture N_BIT_ADDER_LOGIC of N_BIT_ADDER is
signal OUTPUT, Y_TRANSFORMED : std_logic_vector(n-1 DOWNTO 0);
signal CARRY_OUT, OUTPUT_ZERO_CHECK: std_logic_vector(n DOWNTO 0);
signal ZERO: std_logic;

component FULL_ADDER
port
                (
                                grygorczuk_input_3, grygorczuk_input_4, grygorczuk_carry_in: in std_logic;
                                grygorczuk_output, grygorczuk_carry_out_2 : out std_logic
                );
end component;

begin

CARRY_OUT(0) <= grygorczuk_carry_in_2; -- Mistake '1'

NEGATE_LOOP: for i in 0 to n-1 generate
  Y_TRANSFORMED(i) <= grygorczuk_carry_y(i) xor grygorczuk_carry_in_2;
end generate;

FULL_ADDER_LOOP: for i in 0 to n-1 generate
 FULL_ADDER_PART: FULL_ADDER port map(grygorczuk_carry_x(i), Y_TRANSFORMED(i), CARRY_OUT(i),
OUTPUT(i),CARRY_OUT(i+1));
end generate;

OUTPUT_ASSING_LOOP: for i in 0 to n-1 generate
  grygorczuk_o(i) <= OUTPUT(i);
end generate;

grygorczuk_carry_out_flag <= CARRY_OUT(n);
```

```
grygorczuk_overflow_flag <= CARRY_OUT(n) xor CARRY_OUT(n-1);
grygorczuk_negative_flag <= grygorczuk_signed and OUTPUT(n-1);

OUTPUT_ZERO_CHECK(0) <= '0';
ZERO_LOOP: for i in 0 to n-1 generate
  OUTPUT_ZERO_CHECK(i+1) <= OUTPUT_ZERO_CHECK(i) or OUTPUT(i);
end generate;
grygorczuk_zero_flag <=  not OUTPUT_ZERO_CHECK(n);

end N_BIT_ADDER_LOGIC;
```

Code 3. N Bit Adder

Bench Test Code

The test bench code below shows the set up for testing a n bit adder, in this case its set n =4

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity TEST_BENCH_N_BIT_ADDER is
end TEST_BENCH_N_BIT_ADDER;

architecture TEST_BENCH_N_BIT_ADDER_LOGIC of TEST_BENCH_N_BIT_ADDER is

constant n: integer := 4;

component N_BIT_ADDER is
  generic
  (
    n: integer := 4
  );
        port(
                grygorczuk_carry_in_2, grygorczuk_signed : in std_logic;
                grygorczuk_carry_x, grygorczuk_carry_y: in std_logic_vector(n-1 DOWNTO 0);
                grygorczuk_overflow_flag, grygorczuk_negative_flag, grygorczuk_zero_flag, grygorczuk_carry_out_flag: out
std_logic;
                grygorczuk_o: out std_logic_vector(n-1 DOWNTO 0)
        );
end component;

signal grygorczuk_A, grygorczuk_B, grygorczuk_S : std_logic_vector(n-1 DOWNTO 0):= (0 => '0', others => '0');
signal grygorczuk_add_one : std_logic_vector(n-1 DOWNTO 0):= (0 => '1', others => '0');
signal grygorczuk_CI, grygorczuk_CO, grygorczuk_Sign : std_logic;

begin
  ------Instantiate the Unit Under Test (UUT)
utt: N_BIT_ADDER port map(
  grygorczuk_carry_in_2 => grygorczuk_CI,
  grygorczuk_signed => grygorczuk_Sign,
  grygorczuk_carry_x => grygorczuk_A,
  grygorczuk_carry_y => grygorczuk_B,
  grygorczuk_o => grygorczuk_S,
  grygorczuk_carry_out_flag => grygorczuk_CO
);

--Test Bench
Test_Bench : process
begin
```

```
  grygorczuk_CI <= '0';
  grygorczuk_Sign <= '0';
  for i in 0 to 16 loop
    for j in 0 to 16 loop
    wait for 100 ps;
    assert (grygorczuk_S = grygorczuk_A+grygorczuk_B) report "The sum from four bit adder is S =" &
    integer'image(to_integer(unsigned((grygorczuk_S)))) & " while the expected A+B = " &
    integer'image(to_integer(unsigned((grygorczuk_A+grygorczuk_B)))) severity ERROR;
    grygorczuk_B <= grygorczuk_B + grygorczuk_add_one;
    end loop;
  grygorczuk_A <= grygorczuk_A + grygorczuk_add_one;
  end loop;
  report "Test Completed";

end process;



end Test_Bench_N_Bit_Adder_LOGIC;
```

Code 4. N Bit Adder Test Bench

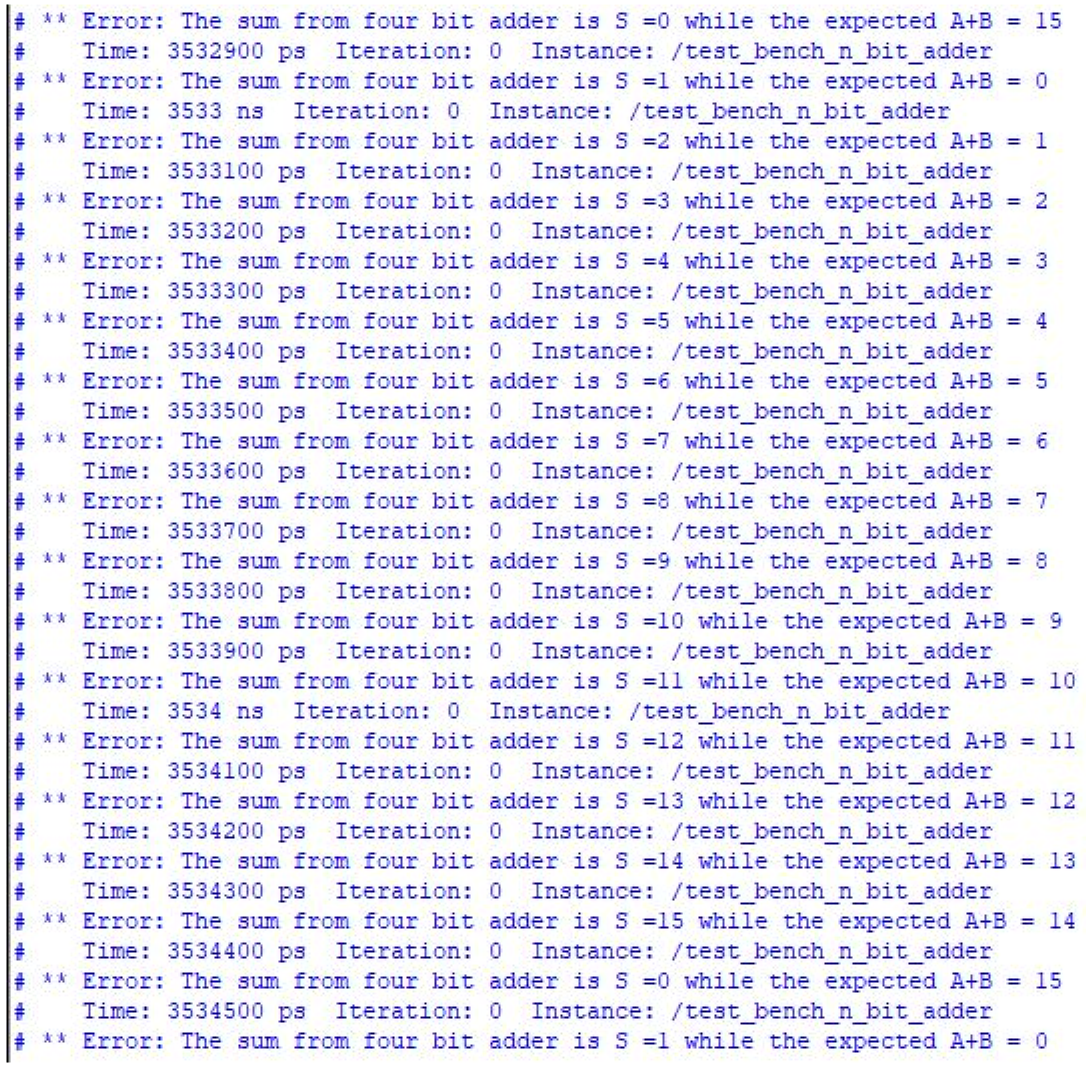
Error Test

Now this code runs perfectly without any errors, thus to prove that it works I place this line in the thirty two bit adder/subtractor as an error

```
CARRY_OUT(0) <= grygorczuk_carry_in_2; -- Mistake '1'
```

Code 6. Intentional Error

The error is changing the initial carry in to ‘1’ which will increase the value of each signal by one, below is the show of this error.



```
# ** Error: The sum from four bit adder is S =0 while the expected A+B = 15
#    Time: 3532900 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =1 while the expected A+B = 0
#    Time: 3533 ns  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =2 while the expected A+B = 1
#    Time: 3533100 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =3 while the expected A+B = 2
#    Time: 3533200 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =4 while the expected A+B = 3
#    Time: 3533300 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =5 while the expected A+B = 4
#    Time: 3533400 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =6 while the expected A+B = 5
#    Time: 3533500 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =7 while the expected A+B = 6
#    Time: 3533600 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =8 while the expected A+B = 7
#    Time: 3533700 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =9 while the expected A+B = 8
#    Time: 3533800 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =10 while the expected A+B = 9
#    Time: 3533900 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =11 while the expected A+B = 10
#    Time: 3534 ns  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =12 while the expected A+B = 11
#    Time: 3534100 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =13 while the expected A+B = 12
#    Time: 3534200 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =14 while the expected A+B = 13
#    Time: 3534300 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =15 while the expected A+B = 14
#    Time: 3534400 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =0 while the expected A+B = 15
#    Time: 3534500 ps  Iteration: 0  Instance: /test_bench_n_bit_adder
# ** Error: The sum from four bit adder is S =1 while the expected A+B = 0
```

Figure 1.

Part 2: Set Less Than (SLT)

Set Less Than requires us to have a situation where A subtract B to affirm if A < B. For that we will use the Comparator which will subtract A-B setting the flags that we will use to set the result of SLT.

Comparator

Below is the code used for creation of the Comparator

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity COMPARATOR is
  generic(
    n: integer := 4
  );
        port(grygorczuk_sign : in std_logic;
                grygorczuk_X, grygorczuk_Y: in std_logic_vector(n-1 DOWNTO 0);
                grygorczuk_OVF, grygorczuk_NF, grygorczuk_ZF, grygorczuk_COF: out std_logic
        );
end COMPARATOR;

architecture COMPARATOR_LOGIC of COMPARATOR is
signal OUTPUT : std_logic_vector(n-1 DOWNTO 0);

component N_BIT_ADDER
  generic(
    n: integer := 4
  );
        port(
                grygorczuk_carry_in_2, grygorczuk_signed : in std_logic;
                grygorczuk_carry_x, grygorczuk_carry_y: in std_logic_vector(n-1 DOWNTO 0);
                grygorczuk_overflow_flag, grygorczuk_negative_flag, grygorczuk_zero_flag, grygorczuk_carry_out_flag: out
std_logic;
                grygorczuk_o: out std_logic_vector(n-1 DOWNTO 0)
        );
end component;

begin
N_BIT_ADDER_PART: N_BIT_ADDER port map('1', grygorczuk_sign, grygorczuk_X,grygorczuk_Y, grygorczuk_OVF,
grygorczuk_NF, grygorczuk_ZF,grygorczuk_COF, OUTPUT);

end COMPARATOR_LOGIC;
```

Code 7. Comparator

SLT

With these two I will be able to turn the B to a - B. After that we use the Thirty Two Bit Adder/Subtractor to perform the subtraction of A - B, and using the Overflow, Negative, and Zero Flags and the Signed/Unsigned Input we set the rules for the outcomes of SLT.

Overflow	Negative	Zero	Carry	Output
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0

0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Tabel 1. SLT Truth Table

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity SLT is
  generic(
    n: integer := 4
  );
  port(
  grygorczuk_x_1, grygorczuk_y_1 : in std_logic_vector(n-1 DOWNTO 0);
  grygorczuk_output : out std_logic
  );
end SLT;



architecture SLT_LOGIC of SLT is

component COMPARATOR is
  generic(
    n: integer := 4
  );
          port(grygorczuk_sign : in std_logic;
                            grygorczuk_X, grygorczuk_Y: in std_logic_vector(n-1 DOWNTO 0);
                            grygorczuk_OVF, grygorczuk_NF, grygorczuk_ZF, grygorczuk_COF: out std_logic
              );
end component;


signal grygorczuk_op :  std_logic_vector(n-1 DOWNTO 0);
signal SIGN, OVF_FLAG, N_FLAG, Z_FLAG, OC_FLAG : std_logic;

--y = B'C'D' + A'BC'
begin
SIGN <= '0';
COMPARATOR_PART : COMPARATOR port map(SIGN, grygorczuk_x_1, grygorczuk_y_1, OVF_FLAG, N_FLAG, Z_FLAG,
OC_FLAG);
grygorczuk_output <= (not N_FLAG and not Z_FLAG and not OC_FLAG) or (not OVF_FLAG and  N_FLAG and not Z_FLAG);

end SLT_LOGIC;
```

Code 9. SLT