Robin Andersson 961102-5439 roband-5
Linus Arvidsson 950427-4490 linarv-4

Lab 3a

D0011E

1. One-bit full adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity FULL_ADDER is
5      Port ( A, B, Cin : in  STD_LOGIC;
6             Cout, R : out  STD_LOGIC);
7  end entity;
8
9  architecture Behavioral of FULL_ADDER is
10
11 signal xor_1, and_1, and_2: std_logic;
12
13 begin
14
15    xor_1 <= A xor B;
16    and_1 <= xor_1 and Cin;
17    and_2 <= A and B;
18    R <= xor_1 xor Cin;
19    Cout <= and_1 or and_2;
20
21 end Behavioral;
```

2. 4-bit adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ADDER is
5      Port ( A, B : in  STD_LOGIC_VECTOR(3 downto 0);
6             Cin : in  STD_LOGIC;
7             R : out  STD_LOGIC_VECTOR(3 downto 0);
8             C, V : out  STD_LOGIC);
9  end entity;
10
11 architecture Behavioral of ADDER is
12 component FULL_ADDER is
13     Port ( A, B, Cin : in  STD_LOGIC;
14            Cout, R : out  STD_LOGIC);
15 end component;
16  signal cin1, cin2, cin3, cout4: std_logic;
17 begin
18
19  FULL_ADDER_0: FULL_ADDER Port Map(A => A(0), B => B(0), Cin => Cin, R => R(0), Cout => cin1);
20  FULL_ADDER_1: FULL_ADDER Port Map(A => A(1), B => B(1), Cin => cin1, R => R(1), Cout => cin2);
21  FULL_ADDER_2: FULL_ADDER Port Map(A => A(2), B => B(2), Cin => cin2, R => R(2), Cout => cin3);
22  FULL_ADDER_3: FULL_ADDER Port Map(A => A(3), B => B(3), Cin => cin3, R => R(3), Cout => cout4);
23  V <= cin3 xor cout4;
24  C <= cout4;
25
26 end Behavioral;
```

3. 4-bit add/sub unit

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ARITH is
5      Port ( A, B : in  STD_LOGIC_VECTOR(3 downto 0);
6             Sub : in  STD_LOGIC;
7             R : out  STD_LOGIC_VECTOR(3 downto 0);
8             V, C : out  STD_LOGIC);
9  end ARITH;
10
11 architecture Behavioral of ARITH is
12
13 component ADDER is
14    Port ( A, B : in  STD_LOGIC_VECTOR(3 downto 0);
15           Cin : in  STD_LOGIC;
16           R : out  STD_LOGIC_VECTOR(3 downto 0);
17           C, V : out  STD_LOGIC);
18 end component;
19
20 signal xor_0: std_logic_vector(3 downto 0);
21
22 begin
23
24   xor_0(0) <= B(0) xor Sub;
25   xor_0(1) <= B(1) xor Sub;
26   xor_0(2) <= B(2) xor Sub;
27   xor_0(3) <= B(3) xor Sub;
28
29   ADDER_1: ADDER Port Map(A => A, B => xor_0, Cin => Sub, R => R, C => C, V => V);
30
31 end Behavioral;
```

4. The tables of the 4-bit add/sub unit

UNSIGNED A	UNSIGNED B	UNSIGNED R	OVERFLOW V	CARRY C
0100 (4)	0011 (3)	0111 (7)	0	0
0111 (7)	0010 (2)	1001 (9)	1	0
1000 (8)	1000 (8)	0000 (0)	1	1
1100 (12)	1100 (12)	1000 (8)	0	1

4 + 3 = 7 which is correct and therefore we should not get an overflow or a carry, which we did not.

7 + 2 = 9 which is correct, but we got an overflow which we should not get since the answer is correct.

8 + 8 = 0 which is incorrect because 8 + 8 = 16, therefore we should get an overflow and a carry since 16 is 10000_2 therefore the overflow and carry is correct.

12 + 12 = 8 which is incorrect because 12 + 12 = 24, therefore we should get an overflow and a carry since 24 is 11000_2 therefore the carry is correct but we did not get an overflow so the overflow is incorrect.

Since the overflow is not always one when the addition is incorrect we can not trust it, but the carry is one when the addition is correct, therefore the carry can be used to detect errors.

SIGNED A	SIGNED B	SIGNED R	OVERFLOW V	CARRY C
0100 (4)	0011 (3)	0111 (7)	0	0
0110 (7)	0010 (2)	1001 (-7)	1	0
0111 (7)	0010 (2)	0101 (5)	0	1
0000 (0)	1000 (-8)	1000 (-8)	1	0

4 + 3 = 7 which is correct and therefore we should not get an overflow or a carry, which we did not.

7 + 2 = -7 which is incorrect because 7 + 2 = 9, therefore we should get an overflow but not a carry since 9 is usually written as 1001_2 but in two's complement that equals to -7. Therefore, the overflow and carry is correct.

7-2 = 5 which is correct and since -2 is 1110_2 then the third and fourth columns have a carry, which means that the overflow is zero and the carry is one, therefore the overflow and carry is correct.

0-(-8) = -8 which is incorrect since 0-(-8) = 8. Since -8 = 1000_2 then we get 0111_2 when we invert it, then when we add 1 to the inverted number we get 1000_2 which means that the third carry gets a one and the fourth carry gets a zero, therefore we get an overflow.

The overflow is one when the addition is incorrect therefore it can be used to detect errors and the carry is only one when the fourth carry is one which does not mean that the answer is incorrect.

5. The test bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ARITH_tb IS
END ARITH_tb;

ARCHITECTURE behavior OF ARITH_tb IS

    COMPONENT ARITH
    PORT(
         A : IN  std_logic_vector(3 downto 0);
         B : IN  std_logic_vector(3 downto 0);
         Sub : IN  std_logic;
         R : OUT  std_logic_vector(3 downto 0);
         V : OUT  std_logic;
         C : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(3 downto 0) := (others => '0');
   signal B : std_logic_vector(3 downto 0) := (others => '0');
   signal Sub : std_logic := '0';

   --Outputs
   signal R : std_logic_vector(3 downto 0);
   signal V : std_logic;
   signal C : std_logic;

```

```
begin
 | -- Create an instance of the component under test
  ARITH_instance: ARITH port map( A => A, B => B, Sub=>Sub, R=>R, V=>V, C => C);

  -- Now define a process to apply some stimulus over time...
  process
    constant PERIOD: time := 40 ns;
  begin
    A <= "0100"; B <= "0011"; Sub <= '0';
    wait for PERIOD;
    A <= "0111"; B <= "0010"; Sub <= '0';
    wait for PERIOD;
    A <= "1000"; B <= "1000"; Sub <= '0';
    wait for PERIOD;
    A <= "1100"; B <= "1100"; Sub <= '0';
    wait for PERIOD;

    A <= "0100"; B <= "0011"; Sub <= '0';
    wait for PERIOD;
    A <= "0111"; B <= "0010"; Sub <= '0';
    wait for PERIOD;
    A <= "0111"; B <= "0010"; Sub <= '1';
    wait for PERIOD;
    A <= "0000"; B <= "1000"; Sub <= '1';
    wait for PERIOD;

    -- put breakpoint to line below
    wait for PERIOD;
  end process;
end behavior;
```

6. 4-bit logic unit

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity LOGIC is
5      Port ( A, B : in  STD_LOGIC_VECTOR(3 downto 0);
6             Op : in  STD_LOGIC_VECTOR(1 downto 0);
7             R : out  STD_LOGIC_VECTOR(3 downto 0));
8  end LOGIC;
9
10 architecture Behavioral of LOGIC is
11
12 component MUX4_1 is
13   Port ( A, B, C, D : in  STD_LOGIC_VECTOR(3 downto 0);
14             Op : in  STD_LOGIC_VECTOR(1 downto 0);
15             S : out  STD_LOGIC_VECTOR(3 downto 0));
16 end component;
17
18 signal and_1, or_1, xor_1, nor_1: std_logic_vector(3 downto 0);
19
20 begin
21  and_1 <= A and B;
22  or_1 <= A or B;
23  xor_1 <= A xor B;
24  nor_1 <= A nor B;
25
26  MUX: MUX4_1 Port Map(A => and_1, B => or_1, C => xor_1, D => nor_1, Op => Op, S => R);
27
28 end Behavioral;
```

The multiplexer:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MUX4_1 is
5      Port ( A, B, C, D : in  STD_LOGIC_VECTOR(3 downto 0);
6             Op : in  STD_LOGIC_VECTOR(1 downto 0);
7             S : out  STD_LOGIC_VECTOR(3 downto 0));
8  end MUX4_1;
9
10 architecture Behavioral of MUX4_1 is
11
12 begin
13
14  S <= A when (Op = "00") else
15       B when (Op = "01") else
16       C when (Op = "10") else
17       D;
18
19 end Behavioral;
```