Lab 7
Sequential Testbench & Two-bit Adder

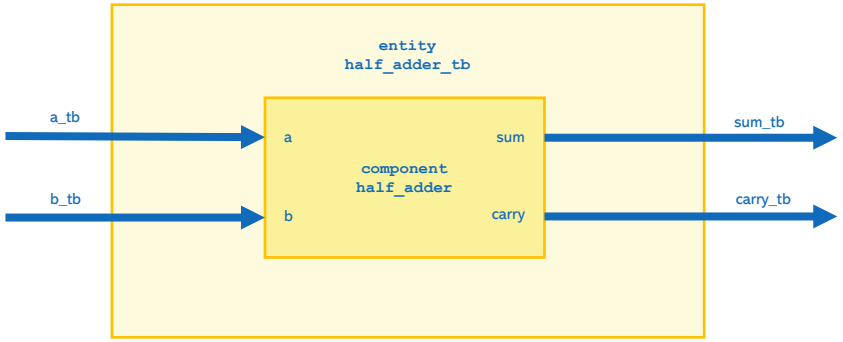
Digital System Fundamentals

Sorayut Glomglome

1. SequentialTestbench

Last Lab Testbench | Concurrent

```
21
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
33
34  entity ha_tb is
35  --  Port ( );
36  end ha_tb;
37
38  architecture Behavioral of ha_tb is
39
40  component ha is port(
41      a : in STD_LOGIC;
42      b : in STD_LOGIC;
43      sum : out STD_LOGIC;
44      carry : out STD_LOGIC);
45  end component;
46
47  signal a_tb, b_tb, sum_tb, carry_tb : std_logic;
48
49  begin
50
51      uut : ha port map (a => a_tb,
52                         b => b_tb,
53                         sum => sum_tb,
54                         carry => carry_tb);
55
56      a_tb <= '0', '1' after 40 ns;
57      b_tb <= '0', '1' after 20 ns, '0' after 40 ns, '1' after 60 ns;
58
59  end Behavioral;
60
```



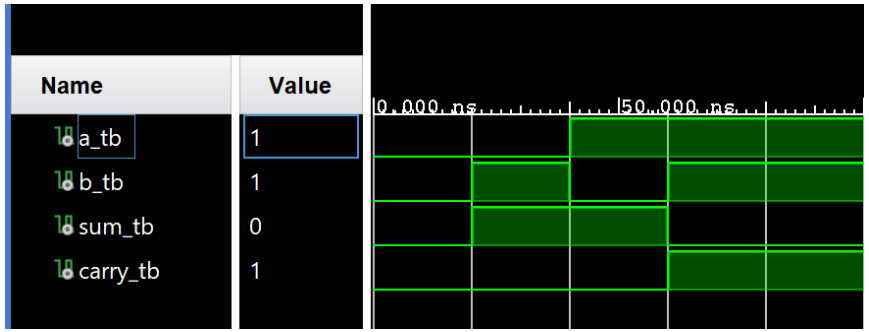
- Concurrent => Difficult
- Manual output checking

Sequential Testbench

1. Simple testbench with concurrent statements.

2. Use **process, assertion, and report** statements.

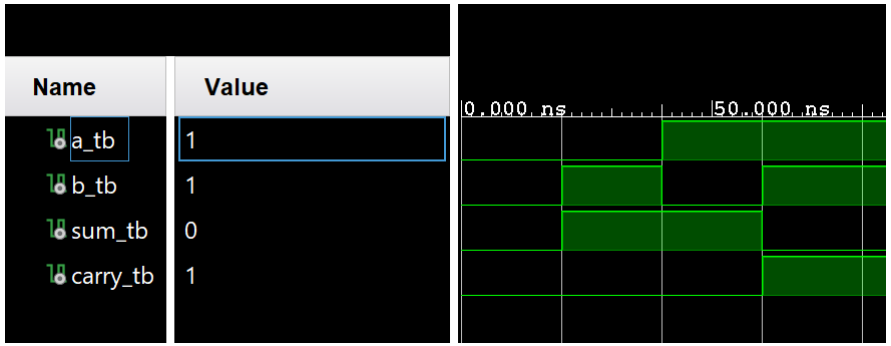
   - Verify output automatically.
   - Print error message.



1.1 Process, Assertion, and Report Statements

Process & Wait Statement

- Sequential execution
- Similar to other high level languages.
- wait for / wait




```
48
49  begin
50
51      uut : half_adder port map (a => a_tb,
52                                 b => b_tb,
53                                 sum => sum_tb,
54                                 carry => carry_tb);
55
56    tb : process
57
58          constant period: time := 20 ns;
59
60          begin
61
62              -- case 0
63              -- input combination a, b ==> 0, 0
64              -- expected output carry, sum ==> 0, 0
65              a_tb <= '0';
66              b_tb <= '0';
67              wait for period;
71
72              -- case 1
73              -- input combination a, b ==> 0, 1
74              -- expected output carry, sum ==> 0, 1
75              a_tb <= '0';
76              b_tb <= '1';
77              wait for period;
81
82              -- case 2
83              -- input combination a, b  ==> 1, 0
84              -- expected output carry, sum ==> 0, 1
85              a_tb <= '1';
86              b_tb <= '0';
87              wait for period;
91
92              -- case 3
93              -- input combination a, b  ==> 1, 1
94              -- expected output carry, sum ==> 1, 0
95              a_tb <= '1';
96              b_tb <= '1';
97              wait for period;
101
102             wait; -- wait indefinitely aka suspend process
103         end process;
104
105 end Behavioral;
```

Assertion & Report Statements

- Check output automatically.
- Print error message onto Tcl console.

```
-- case 0
-- input combination a, b ==> 0, 0
-- expected output carry, sum ==> 0, 0
a_tb <= '0';
b_tb <= '0';
wait for period;
assert (( carry_tb = '0') and (sum_tb = '0'))
report "===> Test failed for input combination 00 <===="
severity error;
```

```
Tcl Console   Messages   Log
# }
# run 1000ns
Error: ===> Test failed for input combination 00 <===
Time: 20 ns  Iteration: 0  Process: /half_adder_tb2/tb  File: G:/Xi
INFO: [USF-XSim-96] XSim completed. Design snapshot 'half_adder_tb2
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:08 .
```

```
-- case 0
-- input combination a, b ==> 0, 0
-- expected output carry, sum ==> 0, 0
a_tb <= '0';
b_tb <= '0';
wait for period;
assert (( carry_tb = '0') and (sum_tb = '1'))
report "===> Test failed for input combination 00 <===="
severity error;
```

Complete Code

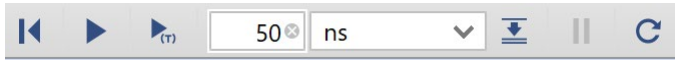
- Complete code for process, assertion and report statements.

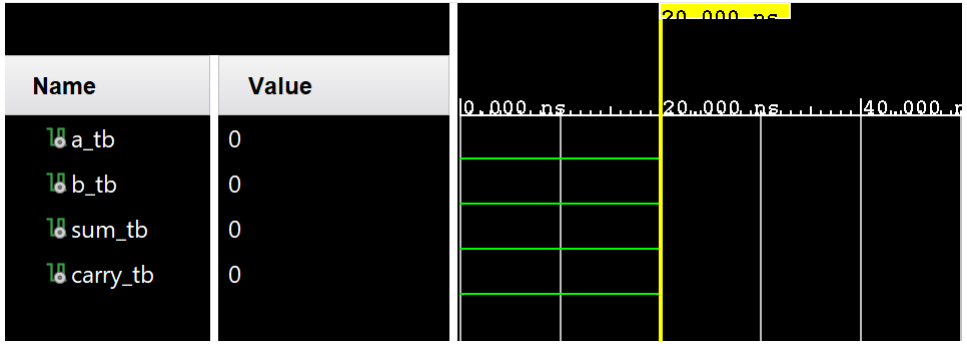
```
tb : process

    constant period: time := 20 ns;

    begin

        -- case 0
        -- input combination a, b ==> 0, 0
        -- expected output carry, sum ==> 0, 0
        a_tb <= '0';
        b_tb <= '0';
        wait for period;
        assert (( carry_tb = '0') and (sum_tb = '0'))
        report "===> Test failed for input combination 00 <===="
        severity error;

        -- case 1
        -- input combination a, b ==> 0, 1
        -- expected output carry, sum ==> 0, 1
        a_tb <= '0';
        b_tb <= '1';
        wait for period;
        assert (( carry_tb = '0') and (sum_tb = '1'))
        report "===> Test failed for input combination 01 <===="
        severity error;

        -- case 2
        -- input combination a, b  ==> 1, 0
        -- expected output carry, sum ==> 0, 1
        a_tb <= '1';
        b_tb <= '0';
        wait for period;
        assert (( carry_tb = '0') and (sum_tb = '1'))
        report "===> Test failed for input combination 10 <===="
        severity error;

        -- case 3
        -- input combination a, b  ==> 1, 1
        -- expected output carry, sum ==> 1, 0
        a_tb <= '1';
        b_tb <= '1';
        wait for period;
        assert (( carry_tb = '1') and (sum_tb = '0'))
        report "===> Test failed for input combination 11 <===="
        severity error;

        wait; -- wait indefinitely aka suspend process
    end process;
```

Breakpoint

```
56  tb : process
57
58      constant period: time := 20 ns;
59
60      begin
61
62          -- case 0
63          -- input combination a, b ==> 0, 0
64          -- expected output carry, sum ==> 0, 0
65          a_tb <= '0';
66          b_tb <= '0';
67          wait for period;
68          assert (( carry_tb = '0') and (sum_tb = '0'))
69          report "===> Test failed for input combination 00 <===="
70          severity error;
71
72          -- case 1
73          -- input combination a, b ==> 0, 1
74          -- expected output carry, sum ==> 0, 1
75          a_tb <= '0';
76          b_tb <= '1';
77          wait for period;
78          assert (( carry_tb = '0') and (sum_tb = '1'))
79          report "===> Test failed for input combination 01 <===="
80          severity error;
81
```

- Restart
- Run All
- Run for 50 ns
- Step
- Break
- Relaunch Simulation





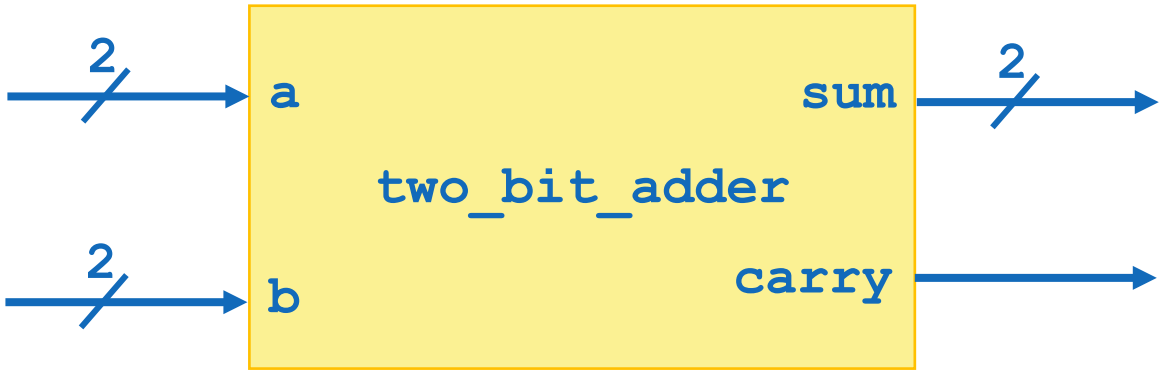
1.2 Checkpoint 1

Checkpoint 1

- Test half adder from Lab 6 using the sequential testbench and provide *one failed case*.

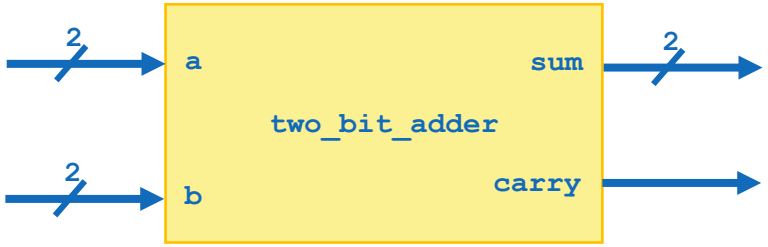
2. Two-bit Adder

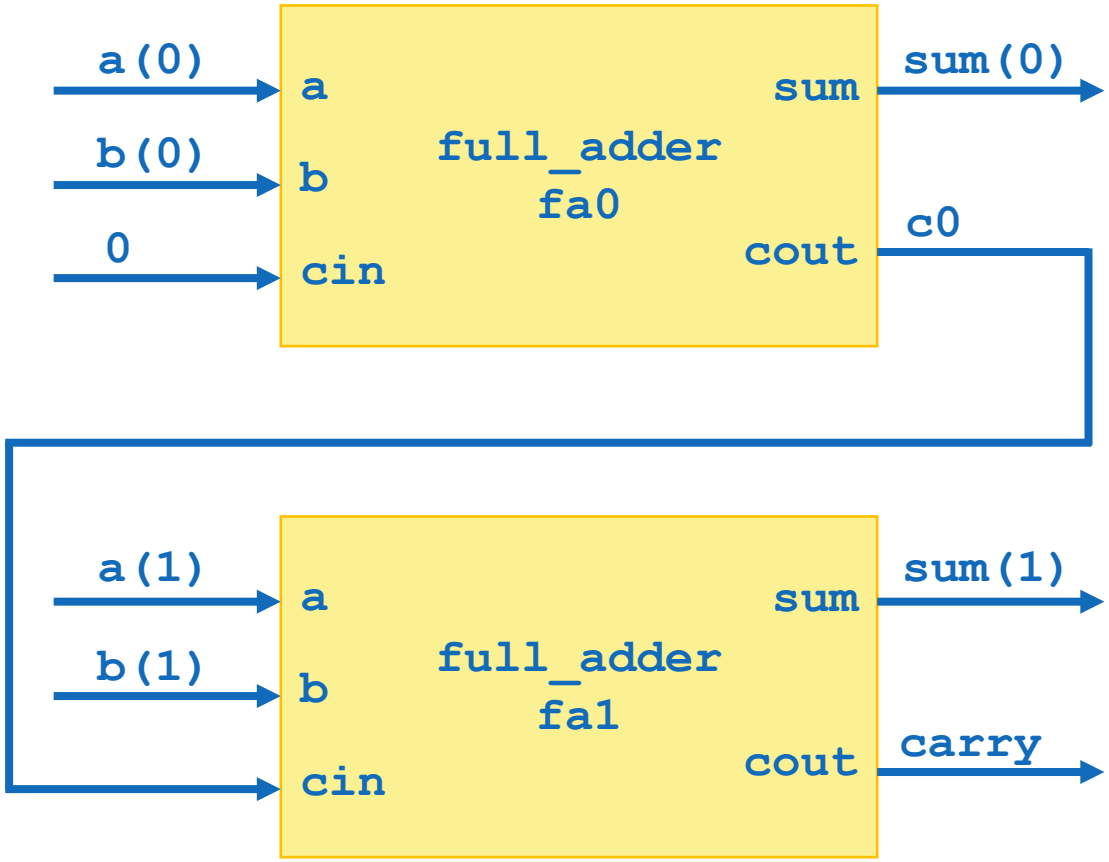
Entity | Two-Bit Adder




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

entity two_bit_adder is
    port ( a : in STD_LOGIC_VECTOR (1 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           sum : out STD_LOGIC_VECTOR (1 downto 0);
           carry : out STD_LOGIC);
end two_bit_adder;
```

Component | Full Adder






```
41 architecture dataflow of two_bit_adder is
42
43     component full_adder is
44         port ( a, b, cin : in STD_LOGIC;
45                sum, cout : out STD_LOGIC);
46     end component;
47
48     signal c0 : std_logic;
49
50 begin
51
52     fa0: full_adder port map (a=>a(0), b=>b(0), cin=>'0', sum=>sum(0), cout=>c0);
53     fa1: full_adder port map (a=>a(1), b=>b(1), cin=>c0, sum=>sum(1), cout=>carry);
54
55
56 end dataflow;
```

Step

1. Add the first design source file for full adder entity.
   a) Use VHDL code from Lab 6.
2. Add the second design source file for two-bit adder.
   a) Declare full adder component using the same port names from the first file.
   b) Port mapping two full adders.

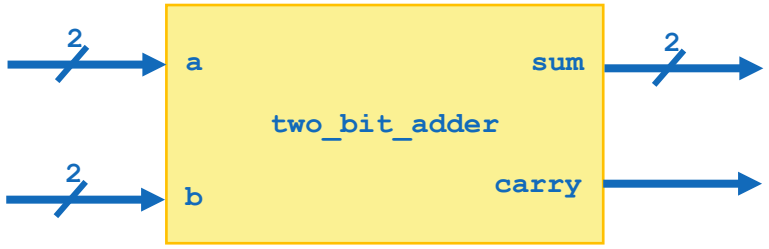
2.1 Checkpoint 2

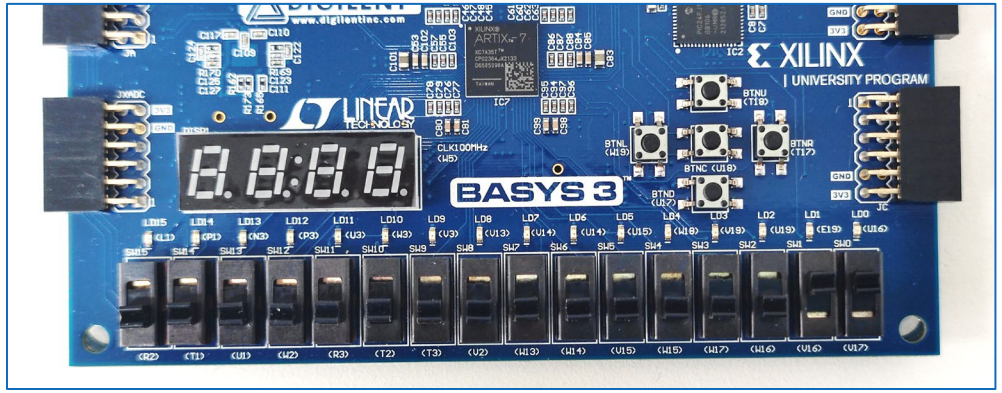
Checkpoint 2

1. Construct two-bit adder from full adders and do the sequential testbench with all possible input cases and provide *two failed cases*.

Checkpoint 2

2. Perform hardware test on FPGA using the following constraints.





Port	I/O Type	Label	Pin
a(1)	switch	SW3	W17
a(0)	switch	SW2	W16
b(1)	switch	SW1	V16
b(0)	switch	SW0	V17
sum(1)	LED	LD1	E19
sum(0)	LED	LD0	U16
carry	LED	LD2	U19

THANK YOU