ENGIN 341 – Advanced Digital Design

Engineering Department

University of Massachusetts Boston

Semester – Spring 2021

Instructor – Dr. Michael Rahaim

Lab Report for

Lab #1: Modular Design and Test Bench Simulation

By

Name: Tyler McKean

Student ID: 01098154

Date: 2/20/2021

I pledge to uphold the governing principles of the Code of Student Conduct of the University of Massachusetts Boston. I will refrain from any form of academic dishonesty or deception, cheating, and plagiarism. I pledge that all the work submitted here is my own, and that I have clearly acknowledged and referenced other people's work. I am aware that it is my responsibility to turn in other students who have committed an act of academic dishonesty; and if I do not, then I am in violation of the Code. I will report to formal proceedings if summoned.

Signed:

CONTENTS

NOTE: Right click and select “Update Field” above to automatically update page numbers.

NOTE: Remove all highlighted text prior to submission

OVERVIEW

The goal for Lab 1 was to introduce the concept of modular design techniques, testbench file simulations, and the I/O planning tool for pin constraints in the Vivado software application. With these three primary objectives, I was able to design modular blocks of a half adder and full adder circuit and implement them into the design of a 2-bit full adder. Each modular design block was tested by creating several testbench simulations and analyzing their expected outputs to confirm their output behaviors prior to synthesizing the logic onto the Zybo board. After confirming that the modular blocks functioned correctly, the 2-bit full adder design was programmed into the Zybo board hardware switches, pushbuttons, and LEDs. The following sections of the report will discuss my process and results from my 2-bit full adder design.

DESIGN DESCRIPTION

The design of a 2-bit full adder consisted of multiple modular design blocks from both a half adder and full adder logic design. The design of a half adder consists of two logic gates with two inputs (a,b) and two outputs (sum, carry). The two inputs are tied to an XOR gate for the sum output and the carry output is derived from the inputs running through an AND gate. Figure 1 shows a depiction of the modular design below.

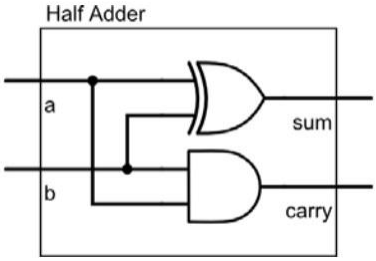

Figure 1- Half Adder logic design, consisting of an XOR gate and an AND gate.

This logic design became the architecture of the half adder modular block designed in Vivado. With the half adder block built, I was then able to use this design to create a modular design for a full adder, respectively. The design of a full adder contains two half adders with two standard inputs, a carry input, a sum output, and a carry out. Figure 2 depicts the input and output relationship of the full adder module.

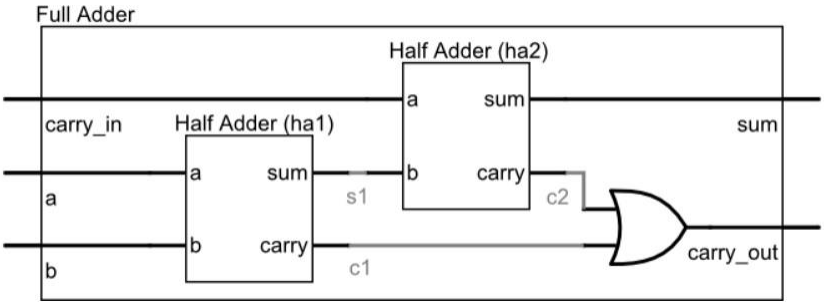

Figure 2 - Full Adder logic design, which consists of two half adders and an OR gate.

Inputs a and b are tied to one of the half adders (ha1), while the carry in and sum output from the first half adder are the inputs of the second half adder (ha2). The sum from ha2 is a result from an XOR operation with s1 and carry_in, while both carry outputs (c1, c2) from both half adders are logically tied to an OR gate for the carry_out result. With the completion of a full adder module, I could then use the full adder design to create the architecture of a 2-bit full adder.

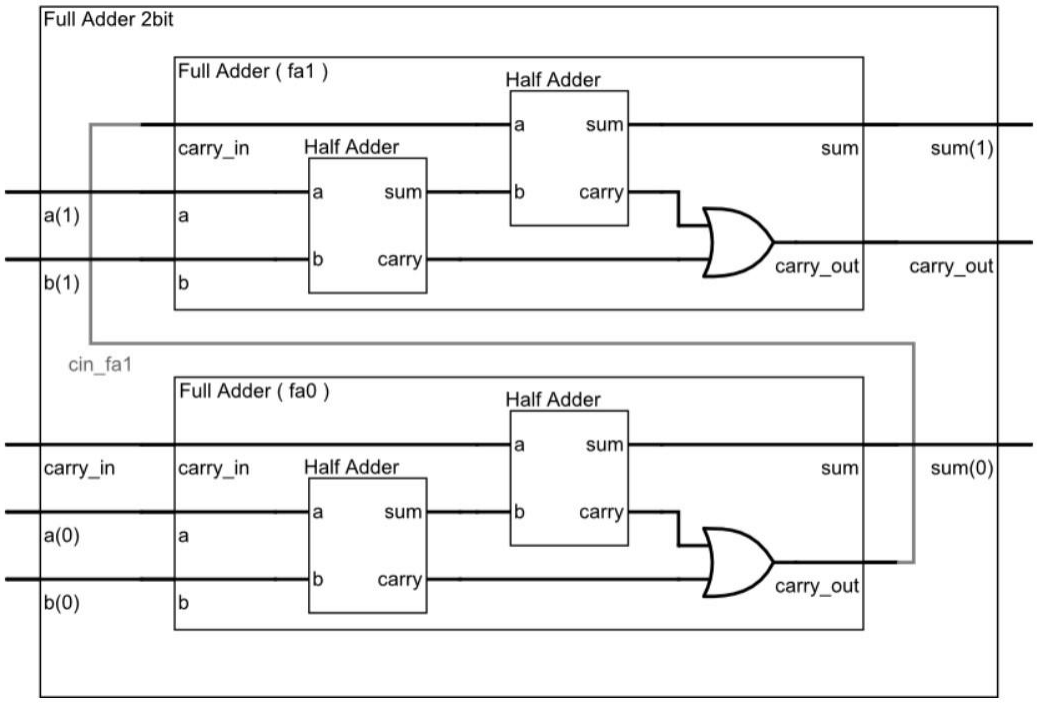

Figure 3 - 2-bit Full Adder design consists of two full adder modular design blocks.

Figure 3 above shows the design layout for the 2-bit full adder. The design requires five inputs and results in three outputs. Having to create this final design from the top down would have been a more difficult approach to the design process. Being able to start with a half adder modular block and implementing the blocks into a full adder, and eventually the 2-bit full adder increased the efficiency in the design process. This lab demonstrates the effectiveness of modular design and how these techniques can be utilized to create more complexed digital systems.

The next steps were taken by creating these modules as design sources in Vivado, as well as creating simulation sources as a means to test the functionality of each individual block.

DESIGN ENTRY

The beginning of the design started by creating the half adder design source in Vivado. This was done by defining the port names, I/O, entity name, and architecture name. The VHDL code can be seen from [1] in the Appendix. Each design source in this lab started by using the IEEE library and using the standard logic from IEEE.STD_LOGIC_1164.ALL file. As was mentioned in the previous section, the half adder contains two inputs (a, b) and two outputs (sum, carry). These port names were defined in the entity definition with the port's correct input/output directions. The architecture was created using a dataflow design since this block was intended to be used as a modular block in the entire digital design. The architecture described

the logic expressions for both the sum and carry outputs. The sum port was the result of input a XOR b, and carry was the result of a AND b. Just these two concurrent statements defined the logic behind the half adder module.

The full adder design can be seen from [2] in the Appendix. Similarly, the full adder design source followed the same setup process in Vivado as the half adder. However, the full adder needed another input port for the carry_in bit in the design description. The entity was defined as a mixed architecture with ports a, b, carry_in as inputs, and sum, carry_out as output bits. In the architecture of the full adder, the half adder was described as a component with its appropriate ports. Listing the half adder as a component allows the half adder entity to be instantiated into the full adder modular design. Interface signals s1, c1, and c2 were then added to the VHDL definition to serve as the signals that interconnect the two half adders. The architecture of the full adder details two half adders: ha1 and ha2. These two half adders followed the same connections as in Figure 2 with the addition of a or_gate definition for the OR operation of both half adder's carry outs. The mixed architecture ends having been defined using two half adder modular blocks.

The last design source to be defined was the 2-bit full adder and its code can be seen as [3] in the Appendix. The entity followed a structural architecture with five inputs and three outputs. The port definitions contain bit vectors to simplify the description of the sets of signals. The structural architecture includes the full adder entity listed as a component and is defined such that we can use multiple modules of the full adder to build our 2-bit full adder. The addition of an interface signal called cin_fa1 was defined to connect the carry out of the first full adder (fa0) with the carry_in input of the second full adder (fa1). Both fa0 and fa1 were then defined from the full adder module with their ports assigned to the correct architecture, as in Figure 3 from before. The full adder fa0 ports were defined by pointing towards the bit vectors a(0), b(0), sum(0), and its carry_out port towards the interface signal cin_fa1. The full adder fa1 then pointed towards a(1), b(1), sum(1), cin_fa1, and carry_out in its respective port definitions. The structural architecture ends having utilized the full adder modular block to define two separate full adders that, when connected, make up the architecture for a 2-bit full adder.

RESULTS AND OBSERVATIONS

Simulation Results

Before moving on to synthesizing and implementing the 2-bit full adder design, I created testbench simulations for each submodule to confirm their functionality. For this project, our professor provided us with the half adder testbench VHDL file and were asked to create the full adder and 2-bit full adder testbenchs ourselves. The half adder testbench provided to us can be seen as [4] in the Appendix. In every testbench environment there is no port definitions for the entity; there is only the module of interest listed as a component definition inside the architecture. Every test bench contains "stimulus" signals that we should define that will drive and observe the I/O of the unit under test. All these stimulus signals are initialized to zero. For the ha_tb_stimulus, the stimulus signals a_s, b_s, sum_s, and carry_out_s were defined. Within the architecture, a Unit Under Test (UUT) for the half adder module is instantiated and connected to the internal stimulus signals. Some signal delay concurrent statements are included for a_s to turn on/off every 10ns and b_s to turn on/off every 20ns. The testbench was then ready to be simulated. In order to check if the module is operating correctly, I needed to create a simulation that would run through every possible outcome of the half adder. Since there are only two inputs that means there are 2^2 or 4

possible outcomes, so I created a simulation that lasted for 40ns in Vivado. The timetable for the ha_tb results can be seen in Figure 4 below.

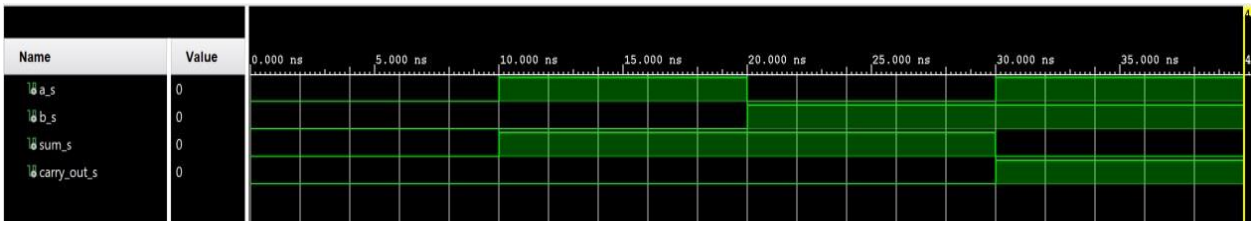

Figure 4 - testbench simulation results for half adder modular design.

The figure shows that the internal stimulus signals are oscillating at the correct time delays mentioned before. The half adder's outputs are also functioning correctly with the sum output being high when only a_s is high or only b_s is high. The carry_out_s was also only high when both a_s and b_s are high, which occurs at the 30ns mark. With the results of the half adder testbench looking good, I moved onto the full adder testbench.

Once again, creating a testbench requires there to be no ports defined for the entity. I created a VHDL file called fa_tb to serve as my testbench simulation and can be referenced from as [5] in the Appendix. I named the architecture fa_tb_stimulus and included the full adder as the module of interest as a component inside the testbench. With the component previously defined, I defined all the necessary stimulus signals to test the full adder module. I initialized stimulus signals a_s, b_s, cin_s, sum_s, and cout_s to zero and instantiated the full adder as the UUT in the architecture. With the ports connected to the appropriate corresponding stimulus signals, I initialized the three input signals with time delays. The a_s signal served as my MSB with a time delay of 40ns, b_s with a time delay of 20ns, and cin_s as my LSB with a 10ns delay. To simulate all possible combinations of the full adder, I had to run through 2^3 or 8 combinations in the timetable. Thus, I setup a stimulation that was 80ns in length and the results can be seen in Figure 5 below.

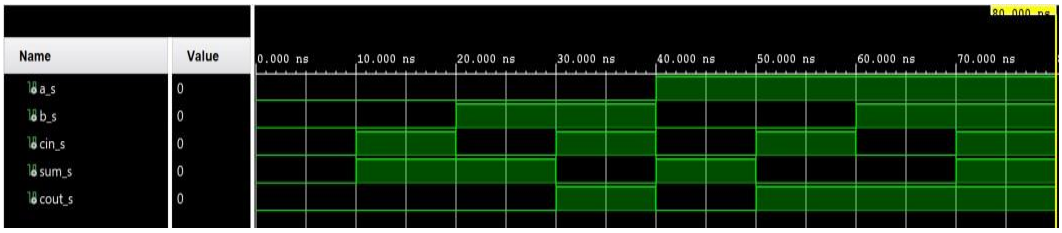

Figure 5 - testbench simulation results for full adder modular design.

Analyzing the results, the sum_s output is only high when cin_s, a_s, or b_s are each individually high when the others are low. The cout_s output is only high when at least two of the input are high. Both sum_s and cout_s are high if all three inputs are high. With these results, I was able to conclude that every possible combination of outcomes is correct for the full adder testbench.

The third and final testbench to setup was the 2-bit full adder and to see if the whole design functions as it should. I created a simulation source called fa2b_tb and the code for the VHDL can be seen in [6] in the Appendix. In a similar format as the testbenches before it, I named this architecture fa2b_tb_stimulus and included the 2-bit full adder module as a component. I then created stimulus signals for the appropriate

number of inputs and outputs for the module. The stimulus signals a0_s, a1_s, b0_s, b1_s, cin_s, sum0_s, sum1_s, and cout_s were all defined and initialized to zero. The full_adder_2bit module was then instantiated as the UUT and I proceeded to connect all the corresponding stimulus signals to their respected inputs and outputs. For this simulation, I instantiated a 160ns delay for a0_s, an 80ns delay for a1_s, a 40ns delay for b0_s, a 20ns delay for b1_s, and a 10ns delay for cin_s. I then created a simulation that could run through every possible combination for the 2-bit full adder, which was 320ns in length.

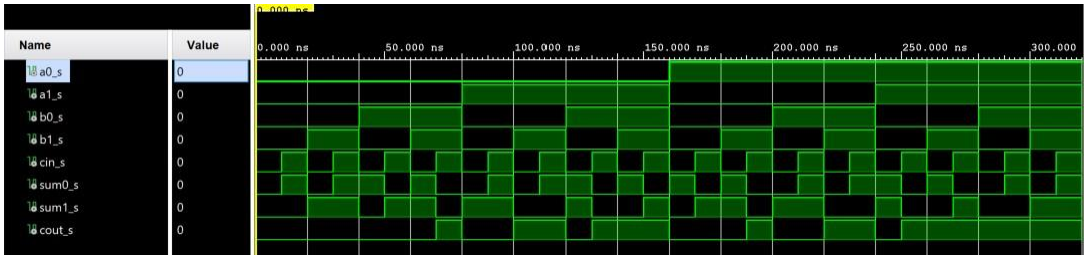

Figure 6 - testbench simulation results for 2-bit full adder modular design

Looking at the simulation results, the 2-bit full adder appears to be operating as intended. The output sum0_s is only high when one of the inputs from the first full adder, a0_s, b0_s, or cin_s, are high. The sum1_s output will be high when either (a1_s OR b1_s) are high or if (a0_s AND b0_s) are high or (cin_s AND a0_s) are high. The sum1_s output will be high when (a1_s AND b1_s) are high or if either a1_s or b1_s are high in addition to the carry_out from the first full adder being high as well. Looking over all 32 possible combinations for the 2-bit full adder, the digital design passed the simulation phase and received the approval to implement the design onto the Zybo board hardware.

Synthesized Results

In order to program the 2-bit adder into the Zybo board hardware, I needed to manually create a constraint file that would map the ports of my 2-bit full adder to the switches, pushbuttons, and LEDs on the Zybo board. This required me to utilize the I/O Planning tab in Vivado and from there I was able to map specific ports of the 2-bit adder to certain switches, pushbuttons, and LEDs. Within the constraint file, I first mapped the switches of the Zybo board. I arranged it such that port A_1 was tied to pin T16, B_1 to W13, A_0 to P15, and B_0 to G15. This makes it so the switches on the Zybo board are descending from MSB to LSB with A_1 and B_1 are directly next to A_0 and B_0. The C_{in} bit was programmed to pin R18, which is a pushbutton and then the LEDs to show the two sum outputs and carry output were assigned to pins M14, M15, and D18.

To test out the 2-bit full adder with the Zybo board, I tried several input combinations to check if the correct outputs were displayed from the LEDs.

The first combination I tried was $A_1B_1 = 11$ and $A_0B_0 = 00$. The figure below shows the Zybo board results.

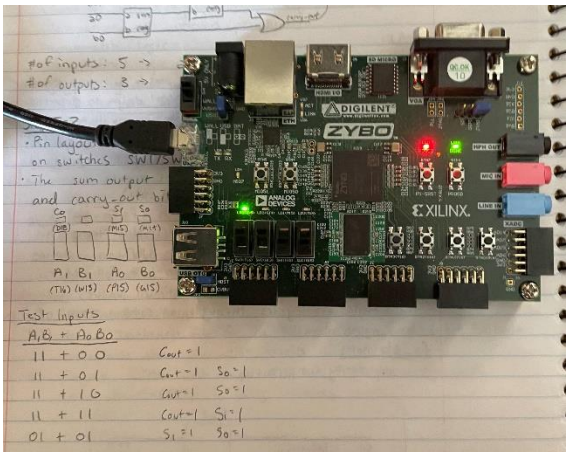

Figure 7 - Switches A_1B_1 = 11 and A_0B_0 = 00, LED for C_{out} illuminated as a result.

With A_1B_1 set to 11, the result should illuminate the LED assigned to C_{out}, which can be seen illuminating in Figure 7. This combination is functioning as expected.

The next combination I tested was A_1B_1 = 00 and A_0B_0 = 11. The results from the Zybo board are displayed in Figure 8 below.

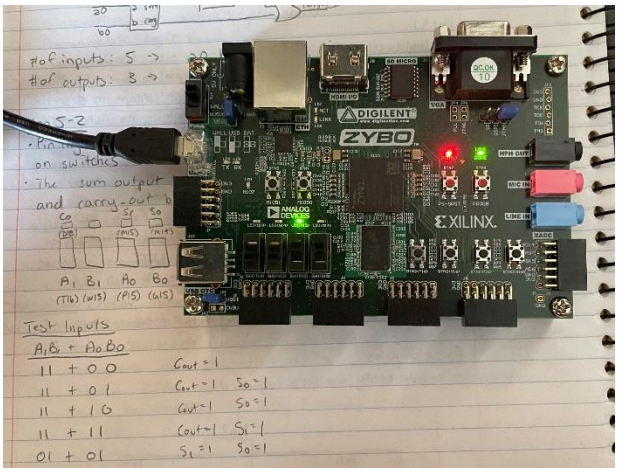

Figure 8 – Switches A_1B_1 = 00 and A_0B_0 = 11, LED for S_1 illuminates as a result.

With only A_0B_0 set to 11, the only LED to display would be the S_1 LED. The result from Figure 8 shows that this input combination was successful as well.

For the next input combination, I set all the switches to high, so A_1B_1 = 11 and A_0B_0 = 11. This would result in the LEDs for S_1 and C_{out} to illuminate. The results of the test can be seen in Figure 9 below.

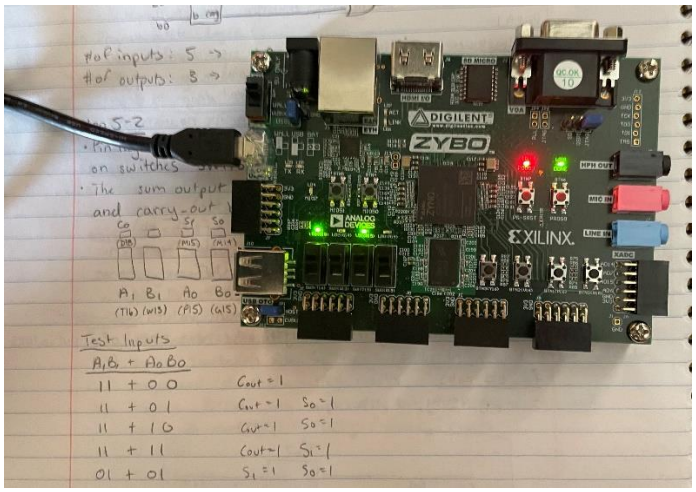

Figure 9 - Switches A_1B_1 = 11 and A_0B_0 = 11, LEDs for S_1 and C_{out} are illuminating as a result.

With every switch set to high, the result did in fact show the S_1 and C_{out} LEDs turned out, which indicates this test was successful as well.

For the final check, I wanted to keep all the switches set to high while also setting the C_{in} pushbutton to high. This input combination would illuminate all three outputs on the Zybo board. The results for this input combination can be seen in Figure 10 below.

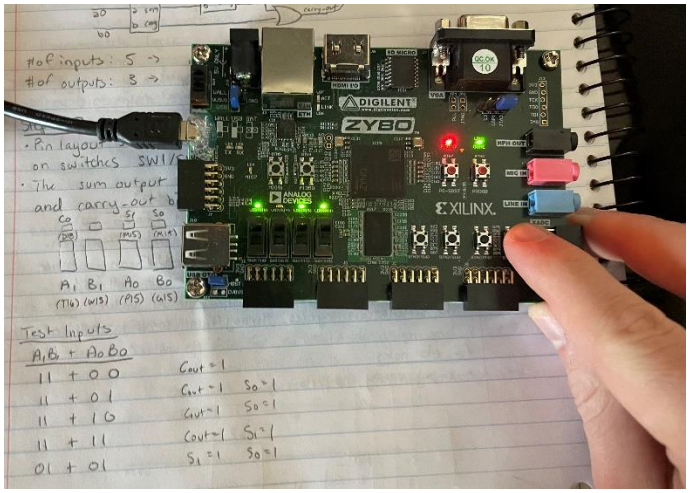

Figure 10 - Switches A_1B_1 = 11, A_0B_0 = 11, and C_{in} = 1, LEDs for S_0, S_1, and C_{out} are illuminated as a result.

This test resulted in every LED lighting up when every single input combination of the 2-bit full adder was set to high. This final test concludes that the logic design implementation of the 2-bit full adder was successful, and functions as intended.

SUMMARY

The main objectives of this lab project were to demonstrate the effectiveness of Modular Design, Testbench Simulations, and the flexible I/O Planning of the Zybo board hardware. By simply starting with the logic design of a half adder circuit, I was able to build and instantiate a 2-bit full adder by using this building block technique with modules. Once I was able to construct the digital design for the 2-bit full adder, I was then able to setup testbench simulations for each subcomponent and the overall digital design to validate their functionalities. After analyzing the results of the testbench simulations, I was able to confirm that the modular design architectures were operating correctly and could thus be synthesized and implemented onto the Zybo board hardware. With the design of the 2-bit full adder implemented onto the hardware switches, pushbuttons, and LEDs of the Zybo board, I conducted a series of tests of different input combinations and to check their corresponding outputs. After several different tests, I was able to confirm that the 2-bit full adder design was successfully implemented onto the Zybo board and it functioned as I expected.

APPENDIX

[1]
```
entity half_adder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sum : out STD_LOGIC;
           carry : out STD_LOGIC);
end half_adder;
architecture dataflow of half_adder is
begin
    sum <= a XOR b;
    carry <= a AND b;
end dataflow;
```

[2]
```
entity full_adder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           carry_in : in STD_LOGIC;
           sum : out STD_LOGIC;
           carry_out : out STD_LOGIC);
end full_adder;
```

```
architecture mixed of full_adder is
component half_adder is
    port(
        a : in STD_LOGIC;
        b : in STD_LOGIC;
        sum : out STD_LOGIC;
        carry : out STD_LOGIC );
end component;
signal s1, c1, c2 : STD_LOGIC;
begin
    ha1: half_adder port map (
        a => a,
        b => b,
        sum => s1,
        carry => c1 );
    ha2: half_adder port map (
        a => carry_in,
        b => s1,
        sum => sum,
        carry => c2 );

    or_gate: carry_out <= c1 OR c2;
end mixed;
```

[3]
```
entity full_adder_2bit is
    Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           carry_in : in STD_LOGIC;
           sum : out STD_LOGIC_VECTOR (1 downto 0);
```

```
        carry_out : out STD_LOGIC);
end full_adder_2bit;
architecture structural of full_adder_2bit is
component full_adder is
    port(
        a : in STD_LOGIC;
        b : in STD_LOGIC;
        carry_in : in STD_LOGIC;
        sum : out STD_LOGIC;
        carry_out : out STD_LOGIC
        );
end component;
signal cin_fa1 : STD_LOGIC;
begin
fa0: full_adder port map (
    a => a(0),
    b => b(0),
    carry_in => carry_in,
    sum => sum(0),
    carry_out => cin_fa1 );
fa1: full_adder port map (
    a => a(1),
    b => b(1),
    carry_in => cin_fa1,
    sum => sum(1),
    carry_out => carry_out );
end structural;
```

[4] entity ha_tb is

```
end ha_tb;
architecture ha_tb_stimulus of ha_tb is
component half_adder is
    port(
        a : in STD_LOGIC;
        b : in STD_LOGIC;
        sum : out STD_LOGIC;
        carry : out STD_LOGIC
        );
end component;
-- Stimulus signals initialized to '0'
signal a_s, b_s, sum_s, carry_out_s : STD_LOGIC := '0';
begin
    -- Instantiate a Unit Under Test (UUT) and connect to internal stimuli
    UUT: half_adder port map (
        a => a_s,
        b => b_s,
        sum => sum_s,
        carry => carry_out_s);

    a_s <= not a_s after 10ns;
    b_s <= not b_s after 20ns;
end ha_tb_stimulus;
```

[5]
```
entity fa_tb is
end fa_tb;
architecture fa_tb_stimulus of fa_tb is
component full_adder is
    port(
```

```
        a : in STD_LOGIC;
        b : in STD_LOGIC;
        carry_in : in STD_LOGIC;
        sum : out STD_LOGIC;
        carry_out : out STD_LOGIC
        );
end component;
-- Stimulus signals initialized to '0'
signal a_s, b_s, cin_s, sum_s, cout_s : STD_LOGIC := '0';
begin
-- Instantiate a Unit Under Test (UUT) and connect to internal stimuli
    UUT: full_adder port map (
        a => a_s,
        b => b_s,
        carry_in => cin_s,
        sum => sum_s,
        carry_out => cout_s );
    a_s <= not a_s after 40ns;
    b_s <= not b_s after 20ns;
    cin_s <= not cin_s after 10ns;
end fa_tb_stimulus;

[6] entity fa2b_tb is
end fa2b_tb;
architecture fa2b_tb_stimulus of fa2b_tb is
component full_adder_2bit is
    port(
        a : in STD_LOGIC_VECTOR (1 downto 0);
        b : in STD_LOGIC_VECTOR (1 downto 0);
```

```
        carry_in : in STD_LOGIC;
        sum : out STD_LOGIC_VECTOR (1 downto 0);
        carry_out : out STD_LOGIC  );
end component;
-- Stimulus signals initialized to '0'
signal a0_s, a1_s, b0_s, b1_s, cin_s, sum0_s, sum1_s, cout_s : STD_LOGIC := '0';
begin
-- Instantiate a Unit Under Test (UUT) and connect to internal stimuli
    UUT: full_adder_2bit port map (
        a(0) => a0_s,
        a(1) => a1_s,
        b(0) => b0_s,
        b(1) => b1_s,
        carry_in => cin_s,
        sum(0) => sum0_s,
        sum(1) => sum1_s,
        carry_out => cout_s );
    a0_s <= not a0_s after 160ns;
    a1_s <= not a1_s after 80ns;
    b0_s <= not b0_s after 40ns;
    b1_s <= not b1_s after 20ns;
    cin_s <= not cin_s after 10 ns;
end fa2b_tb_stimulus;
```