ECEN302 Lab 3 - Functions, Procedures, and Test Benches

Joshua Benfell - 300433229

August 30, 2020

1 Objectives

The primary objectives of this lab was to learn to develop reusable VHDL code with the inbuilt procedures and functions. We are to learn the differences between both constructs in VHDL and this will help us develop our thinking and understanding about sequential and procedural programming. We will be able to verify this learning when we have developed models for combinatorial logic using both functions and procedures and when we develop test benches to verify their functionality.

2 Methodology

2.1 Introduction

In this lab we will be implementing models for various combinatorial logic circuits in VHDL. This will be done using the inbuilt procedures and functions. The functionality of each of the implemented models will be verified through the use of test benches. Finally a test bench will be used to generate a waveform. This will provide a deeper understanding of how each of the constructs work and the features that are provided with them.

2.2 Procedures

The first procedural model that was created was one that would add two 4 bit values into a sum nybble and a carry bit. To do this a procedure was created that took the two numbers as input and outputted a 5 bit number. Inside this procedure is a single line that adds the numbers together. From there, the carry bit was assigned to bit 5 of the summed 5 bit number and the sum was set to the first 4 bits, this was then outputted from the module. To call the procedure, it was done in a process, this makes the call sequentially, but allows us to run concurrent code in a sequential statement. Using the provided test bench shows that this code works successfully in Figure 1. An improvement that could be made to this program is to call the procedure outside of a process as this is unnecessary.

A further exercise done with procedures was a even parity bit generator. The hardest part of this section of the lab was figuring out how to generate a parity bit. After writing it down on paper, it was found that the logical XOR operation would do the trick. By XORing the parity bit's current value with the current bit as we loop through the bits. By defining the parity bit as being the bit that ensures there's an even number of 1's in a series of bits, we can initialise the parity bit as 0, this was if the first bit is a 1, then the parity bit will

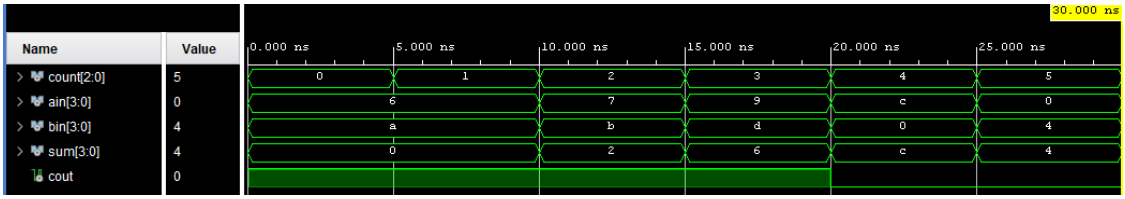

Figure 1: Testbench results for add two values

also be a 1 to ensure there are 2 1's. We can see this behaviour in the testbench results, Figure 2.

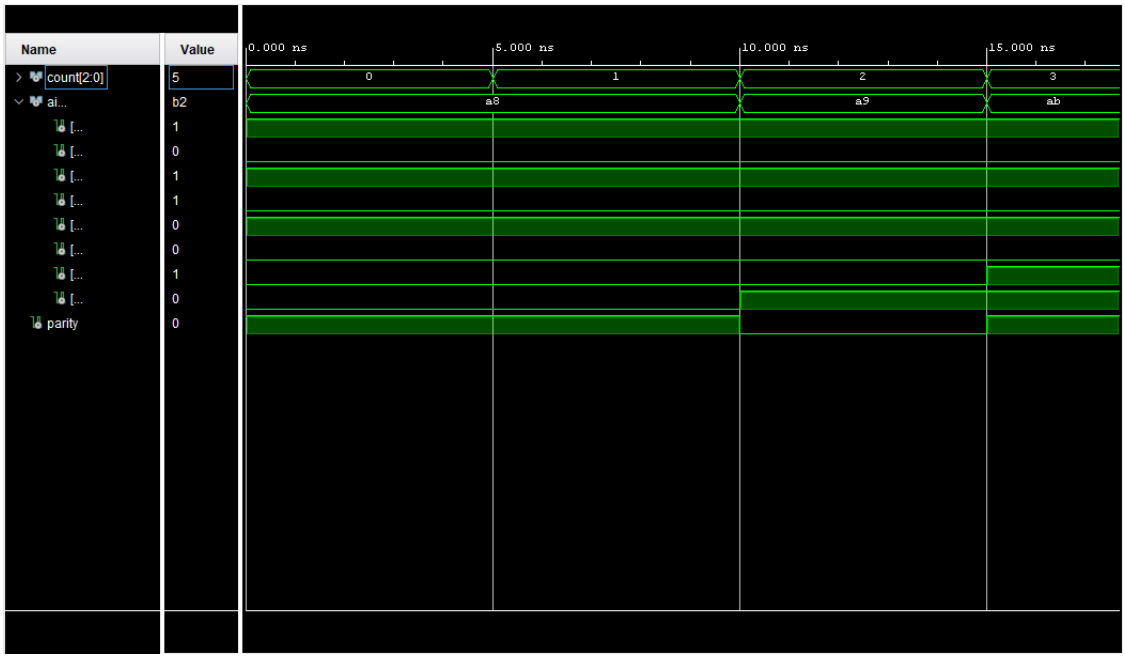

Figure 2: Testbench results for calc parity bit

2.3 Functions

The next VHDL construct to be used is the function. The module being implemented first is another one that adds two numbers together. The overall functionality of the function is very similar to the procedure version where it adds the numbers into a 5 bit number, however, there was no requirement for a carry bit here, so the number was not required to be split up. The function does have the noticeable difference of the return keyword, this is because it's not possible to have the output be a signal, it has to be assigned to a variable and then that variable must be returned. This in turn is sequential and not concurrently happening. The functionality of the code can be seen in the testbench results obtained from using the provided test bench, in Figure 3.

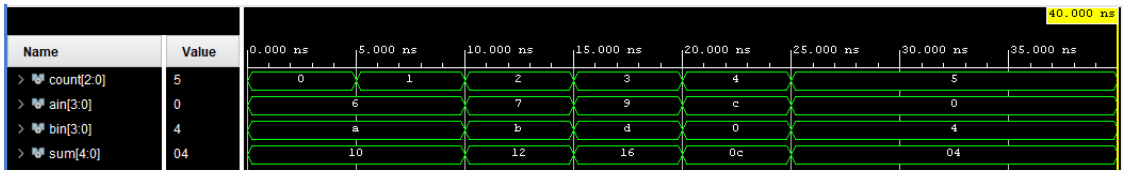

Figure 3: Testbench results for add two values

The next module to be implemented will count the number of 1's in a binary sequence. This had the same issue with the parity bit code of finding a clever way to do this. I couldn't think of anything so I went with looping through it and adding 1 every time a 1 was found in the sequence. The results of this implementation can be seen in the results from the provided testbench, Figure 4.

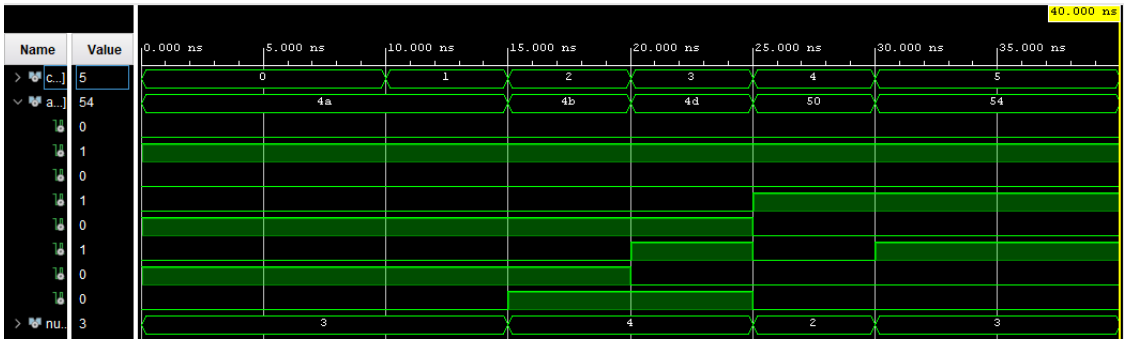

Figure 4: Testbench results for calc ones

2.4 Testbench

The final tool explored in this lab was the testbench. Our first task was to fix up the testbench for the RCA dataflow. This was done by first un-commenting all the commented out write lines. This didn't make the testbench work, and it turns out the strings needed to be explicitly cast to strings by surrounding it with string'(). This was odd, but made it work, although it never seemed necessary as the strings were surrounded by double quotes, which typically denotes a string in languages like C++, and VHDL uses them to denote sequences of bits. The results and tcl console can be seen in the testbench results, Figure 5

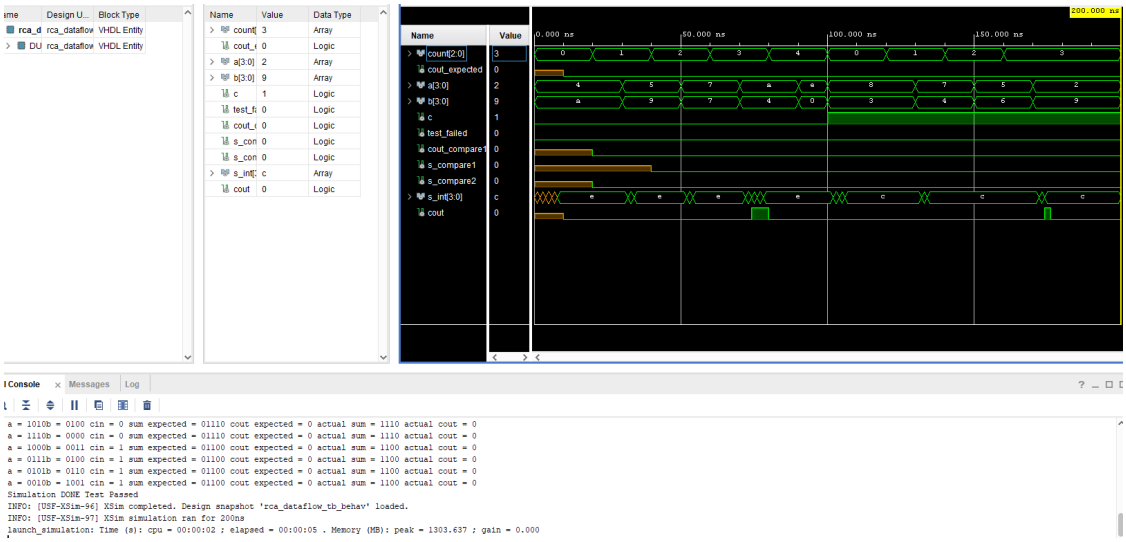

Figure 5: Testbench results for RCA_dataflow_tb

The final task to do was generate a waveform using a testbench, while initially confusing it was determined that the creation of a module was not necessary, just a testbench. To do this, the relevant variables were created and with the correct time intervals, set to their required states to produce the waveform. This was easiest done as a process as the wait for instruction would allow for accurate time intervals. The resulting waveform can be seen in

Figure 4. This was only done for 160 ns as that was what the example showed in the lab script, even though it asked for a 200 ns snapshot. So this better mirrors the script.

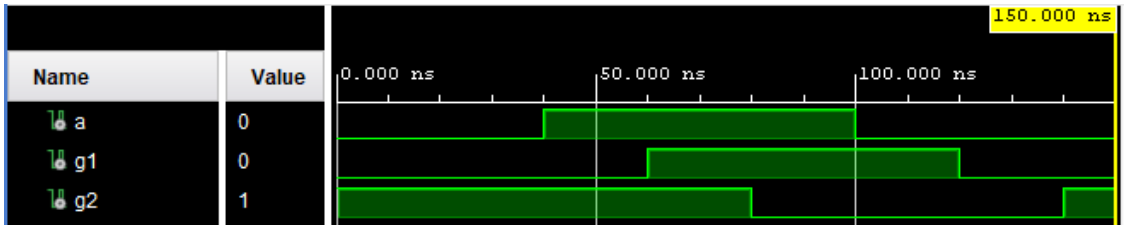

Figure 6: Testbench Waveform

2.5 Common Challenges

A common issue that was regularly encountered throughout this lab was syntax errors. I don't remember exactly what issues came up, but I do know that all of them were solved by reviewing the previous labs, looking at the lab scripts example code and googling examples of the various constructs. Through the use of these tools, I was able to solve all of the syntax and compilation issues.

2.6 Code

2.6.1 1-1 Procedures Add two Values

```
----------------------------------------------------------------------------------
-- Module Name: add_two_values_procedure
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

Entity add_two_values_procedure Is Port (
    Signal ain : in STD_LOGIC_VECTOR (3 downto 0);
    Signal bin : in STD_LOGIC_VECTOR (3 downto 0);
    Signal sum : out STD_LOGIC_VECTOR (3 downto 0);
    Signal cout : out STD_LOGIC
);
end add_two_values_procedure;

Architecture behavior of add_two_values_procedure Is

    Signal total_out : STD_LOGIC_VECTOR (4 downto 0);

    procedure add_two_values (
        Signal ain_int : in STD_LOGIC_VECTOR (3 downto 0);
        Signal bin_int : in STD_LOGIC_VECTOR (3 downto 0);
        Signal total_out_int : out STD_LOGIC_VECTOR (4 downto 0)
    ) is

```

```
    begin
        total_out_int(4 downto 0) <= ('0' & ain_int) + ('0' & bin_int);
    end add_two_values;

begin
    cout <= total_out(4);
    sum <= total_out(3 downto 0);

    process (ain, bin)
    begin
          add_two_values (ain, bin, total_out);
    end process;
end behavior;
```

code/lab311.vhd

2.6.2 1-2 Calc Even Parity

```
----------------------------------------------------
-- Module Name: calc_even_parity_procedure
----------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

Entity calc_even_parity_procedure  Is Port (
    Signal ain : in STD_LOGIC_VECTOR (7 downto 0);
    Signal parity : out STD_LOGIC
);
end calc_even_parity_procedure ;



Architecture behavior of calc_even_parity_procedure  Is

    procedure calc_even_parity (
        signal input : in STD_LOGIC_VECTOR (7 downto 0);
        signal output : out std_logic
    ) is
    variable par_bit : STD_LOGIC := '0';
    begin
        for k in 0 to input'length-1 loop
            par_bit := par_bit xor ain(k);
        end loop;
        output <= par_bit;
    end calc_even_parity;

begin

    calc_even_parity(ain, parity);

end behavior;
```

code/lab312.vhd

2.6.3 2-1 Functions Add Two Values

```
----------------------------------------------------------------------------------
-- Module Name: add_two_values_function
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

Entity add_two_values_function  Is Port (
    Signal ain : in STD_LOGIC_VECTOR (3 downto 0);
    Signal bin : in STD_LOGIC_VECTOR (3 downto 0);
    Signal sum : out STD_LOGIC_VECTOR (4 downto 0)
);
end add_two_values_function ;

Architecture behavior of add_two_values_function  Is

    signal ain_int : STD_LOGIC_VECTOR(4 downto 0);
    signal bin_int : STD_LOGIC_VECTOR(4 downto 0);

    function add_two_values  (signal ain, bin : in STD_LOGIC_VECTOR)
    return std_logic_vector is
        variable result : STD_LOGIC_VECTOR(4 downto 0);
    begin
        result := ain + bin;
        return result;
    end add_two_values;

begin
    ain_int <= "0" & ain;
    bin_int <= "0" & bin;

    process (ain_int, bin_int) begin
        sum <= add_two_values(ain_int, bin_int);
    end process;
end behavior;
```

code/lab321.vhd

2.6.4 2-2 Calc Ones

```
----------------------------------------------------------------------------------
-- Module Name: calc_ones_function
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

Entity calc_ones_function  Is Port (
```

```
    Signal ain : in STD_LOGIC_VECTOR (7 downto 0);
    Signal number_of_ones : out STD_LOGIC_VECTOR (2 downto 0)
);
end calc_ones_function ;

Architecture behavior of calc_ones_function  Is

    function count_ones(
        signal input : std_logic_vector (7 downto 0)
    )
    return std_logic_vector is
       variable count : std_logic_vector (2 downto 0) := "000";
    begin
       for k in 0 to input'length-1 loop
           if (input(k) = '1') then
              count := count + 1;
           end if;
       end loop;
    return count;
    end count_ones;


begin
    number_of_ones <= count_ones(ain);


end behavior;
```

code/lab322.vhd

2.6.5 3-1 RCA Test Bench

```
----------------------------------------------------------------------------------
-- Module Name: rca_dataflow_tb
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use STD.textio.all;
use IEEE.std_logic_textio.all;

library UNISIM;
use UNISIM.VComponents.all;

Entity rca_dataflow_tb Is
end rca_dataflow_tb;

Architecture behavior of rca_dataflow_tb Is
    Component rca_dataflow
    port(
        a    : in STD_LOGIC_VECTOR (3 downto 0);
        b    : in STD_LOGIC_VECTOR (3 downto 0);
        cin  : in STD_LOGIC;
        s    : out STD_LOGIC_VECTOR (3 downto 0);
        cout : out STD_LOGIC
    );
    End Component;
```

```
29
30      Signal count : STD_LOGIC_VECTOR (2 downto 0) := "000";
31      Signal cout_expected: STD_LOGIC;
32
33      Signal a : STD_LOGIC_VECTOR (3 downto 0);
34      Signal b : STD_LOGIC_VECTOR (3 downto 0);
35      Signal c : STD_LOGIC;
36      Signal test_failed : STD_LOGIC;
37      Signal cout_compare1 : STD_LOGIC;
38      Signal s_compare1 : STD_LOGIC;
39      Signal s_compare2 : STD_LOGIC;
40      Signal s_int : STD_LOGIC_VECTOR (3 downto 0);
41      Signal cout : STD_LOGIC;
42
43      procedure add_two_values (
44          a_in : in std_logic_vector(3 downto 0);
45          b_in : in std_logic_vector(3 downto 0);
46          c_in : in std_logic;
47
48          sum : out std_logic_vector(4 downto 0)
49      ) is
50
51      begin
52          sum := ("0" & a_in) + ("0" & b_in) + ("0" & c_in);
53      end add_two_values;
54
55  begin
56      DUT:  rca_dataflow PORT MAP (
57              a => a,
58              b => b,
59              cin => c,
60              s => s_int,
61              cout => cout
62           );
63
64      process
65          variable k : integer := 0;
66          variable s : line;
67          variable sum_out : STD_LOGIC_VECTOR (4 downto 0);
68
69      begin
70
71          a <= "0100"; b <= "1010"; c <= '0'; count <= "000"; test_failed <=
                '0';
72          wait for 10 ns;
73
74          add_two_values(a, b, c, sum_out);
75          cout_expected <= sum_out(4);
76
77          wait for 10 ns;
78          write (s, string'("a = "));
79          write (s, a);
80          write (s,  string'("b = "));
81          write (s, b);
82          write (s, string'(" cin = "));
83          write (s, c);
84          write (s, string'(" sum expected = "));
85          write (s, sum_out);
86          write (s, string'(" cout expected = "));
```

```
87          write (s, cout_expected);
88          write (s, string'(" actual sum = "));
89          write (s, s_int);
90          write (s, string'(" actual cout = "));
91          write (s, cout);
92          writeline (output, s);
93
94          cout_compare1 <= cout_expected XOR cout;
95          s_compare2 <= (sum_out(3) XOR s_int(3)) OR (sum_out(2) XOR s_int(2))
               OR (sum_out(1) XOR s_int(1)) OR (sum_out(0) XOR s_int(0)) ;
96
97          if (cout_compare1 = '1') or (s_compare2 = '1') then
98              test_failed <= '1';
99          end if;
100
101         for k in 1 to 4 loop
102           count <= count + "1";
103
104           wait for 10 ns;
105           a <= a + count; b <=  b - count;
106           add_two_values(a, b, c, sum_out);
107           cout_expected <= sum_out(4);
108
109           wait for 10 ns;
110             write (s, string'("a = "));
111             write (s, a);
112             write (s,  string'("b = "));
113             write (s, b);
114             write (s, string'(" cin = "));
115             write (s, c);
116             write (s, string'(" sum expected = "));
117             write (s, sum_out);
118             write (s, string'(" cout expected = "));
119             write (s, cout_expected);
120             write (s, string'(" actual sum = "));
121             write (s, s_int);
122             write (s, string'(" actual cout = "));
123             write (s, cout);
124             writeline (output, s);
125
126           cout_compare1 <= cout_expected XOR cout;
127           s_compare1 <= (sum_out(3) XOR s_int(3)) OR (sum_out(2) XOR
                 s_int(2)) OR (sum_out(1) XOR s_int(1)) OR (sum_out(0) XOR
                 s_int(0));
128
129           if (cout_compare1 = '1') or (s_compare2 = '1') then
130             test_failed <= '1';
131           end if;
132         end loop;
133
134
135         a <= "1000"; b <= "0011"; c <= '1'; count <= "000";
136         wait for 10 ns;
137
138         add_two_values(a, b, c, sum_out);
139         cout_expected <= sum_out(4);
140
141         wait for 10 ns;
142         write (s, string'("a = "));
```

```
143         write (s, a);
144         write (s,  string'("b = "));
145         write (s, b);
146         write (s, string'(" cin = "));
147         write (s, c);
148         write (s, string'(" sum expected = "));
149         write (s, sum_out);
150         write (s, string'(" cout expected = "));
151         write (s, cout_expected);
152         write (s, string'(" actual sum = "));
153         write (s, s_int);
154         write (s, string'(" actual cout = "));
155         write (s, cout);
156         writeline (output, s);
157
158         cout_compare1 <= cout_expected XOR cout;
159         s_compare2 <= (sum_out(3) XOR s_int(3)) OR (sum_out(2) XOR s_int(2))
               OR (sum_out(1) XOR s_int(1)) OR (sum_out(0) XOR s_int(0)) ;
160
161         if (cout_compare1 = '1') or (s_compare2 = '1') then
162             test_failed <= '1';
163         end if;
164
165         for k in 2 to 4 loop
166           count <= count + '1';
167
168           wait for 10 ns;
169           a <= a - count; b <=  b + count;
170           add_two_values(a, b, c, sum_out);
171           cout_expected <= sum_out(4);
172
173           wait for 10 ns;
174             write (s, string'("a = "));
175             write (s, a);
176             write (s,  string'("b = "));
177             write (s, b);
178             write (s, string'(" cin = "));
179             write (s, c);
180             write (s, string'(" sum expected = "));
181             write (s, sum_out);
182             write (s, string'(" cout expected = "));
183             write (s, cout_expected);
184             write (s, string'(" actual sum = "));
185             write (s, s_int);
186             write (s, string'(" actual cout = "));
187             write (s, cout);
188             writeline (output, s);
189
190           cout_compare1 <= cout_expected XOR cout;
191           s_compare1 <= (sum_out(3) XOR s_int(3)) OR (sum_out(2) XOR
                 s_int(2)) OR (sum_out(1) XOR s_int(1)) OR (sum_out(0) XOR
                 s_int(0));
192
193           if (cout_compare1 = '1') or (s_compare1 = '1') then
194             test_failed <= '1';
195           end if;
196         end loop;
197
198         wait for 10 ns;
```

```
199
200         if (test_failed = '1') then
201             write (s, string'("Simulation DONE"));
202             write (s, string'(" Test Failed"));
203             writeline (output, s);
204         else
205             write (s, string'("Simulation DONE"));
206             write (s, string'(" Test Passed"));
207             writeline (output, s);
208         end if;
209
210         wait;
211     end process;
212 end behavior;
```

code/lab331_tb.vhd

2.6.6 3-2 Waveform Generator Testbench

```
1  ----------------------------------------------------------------------------------
2  -- Module Name: waveform_generation_tb
3  ----------------------------------------------------------------------------------
4
5  library IEEE;
6  use IEEE.STD_LOGIC_1164.ALL;
7
8  library UNISIM;
9  use UNISIM.VComponents.all;
10
11 Entity waveform_generation_tb Is
12 end waveform_generation_tb;
13
14 Architecture behavior of waveform_generation_tb Is
15     Signal a : STD_LOGIC := '0';
16     Signal g1 : STD_LOGIC := '0';
17     Signal g2 : STD_LOGIC := '1';
18
19 begin
20
21     process
22     begin
23         wait for 40ns;
24         a <= '1';
25         wait for 20ns;
26         g1 <= '1';
27         wait for 20ns;
28         g2 <= '0';
29         wait for 20ns;
30         a <= '0';
31         wait for 20ns;
32         g1 <= '0';
33         wait for 20ns;
34         g2 <= '1';
35     end process;
36
37
38 end behavior;
```

code/lab332_tb.vhd

3 Questions