ECEN302 : Integrated Digital Electronics
Lab 3 Submission

Daniel Eisen : 300447549

September 3, 2020

1 Objectives

This lab focuses on code reusability and packaging. It's purpose is to show and familiarise ourselves with wrapping up sections of code for multiple reuses throughout the project. These are done with Procedures and Functions; each with different specification and use cases, as well as unique restrictions.
Lastly there is an exercise in testbench development, an integral part of HDL design as it allows for the quickly modelling an implementation and extracting more internal information about the operation that can even be seen in hardware.

2 Methodology

For procedures and functions the flow was to read and run a simpler provided vhdl program to confirm it functionality then to write a more complex/larger design and confirm its functionality with a provided testbench.
In the testbench section, we are provided with an example in which be enable/explore tcl-console output and then we write our own to provide a specified output.

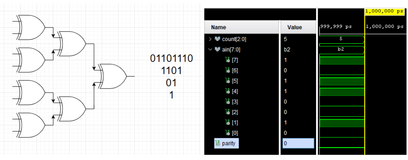
2.1 Procedures

Procedures are a more general purpose sub-programming construct, and allow for wrapping up practically any logical block of code for reuse. It can be sequential or combinatorial, contain delays, take any port types as arguments etc. It is simply wrapped and called later.

```
-- 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 is_par : out std_logic) is
begin
  is_par <= (((input(0)) XNOR (input(1))) XNOR ((input(2)) XNOR (input(3))))
      XOR (((input(4)) XNOR (input(5))) XNOR ((input(6)) XNOR (input(7))));
end calc_even_parity;

begin

calc_even_parity(ain, parity);

end behavior;
```

The procedure we designed and wrote was an 8-bit parity checker that took a 8-bit input number and determined if it had even parity i.e. equal 0 and 1. Parity being represented by a 0 and non-parity a 1.
To achieve this simply, cascading XOR operations on the neighbouring bit pairs, as shown below and implemented above.

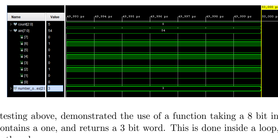


2.2 Functions

Functions are a more specialised method of code wrapping. Differing primary in that they can return a value directly for use, not just modifying a signal/port. More specifically they have much stricter requirements: The logic must be purely combinatorial, a return type must be specified, can only accept input arguments, and contain non-blocking operations.

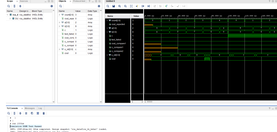
```
-- Module Name: calc_ones_function
```

```
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 add_two_values (
                        signal input : std_logic_vector (7 downto 0))
        return std_logic_vector is
            variable count : std_logic_vector(2 downto 0)  := "000";
        begin
            for i in 0 to 7 loop
                if input(i)='1' then
                    count := count+1;
                end if;
            end loop;
        return count;
        end add_two_values;

 begin

 number_of_ones <= add_two_values(ain);

end behavior;
```



The code and testing above, demonstrated the use of a function taking a 8 bit input and counting each bit that contains a one, and returns a 3 bit word. This is done inside a loop, and testing with the provided testbench.

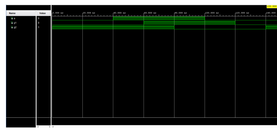
2.3 Testbench



Above shows the tcl console output writing achieved by the testbench, this required some uncommenting write calls and casting string literals as `string'`.

```
-- Module Name: waveform_generation_tb
```

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

library UNISIM;
use UNISIM.VComponents.all;

Entity waveform_generation_tb Is
end waveform_generation_tb;

Architecture behavior of waveform_generation_tb Is
        Signal a : STD_LOGIC := '0';
        Signal g1 : STD_LOGIC := '0';
        Signal g2 : STD_LOGIC := '1';

begin
        process
          begin

        wait for 40ns; a <= not a;
        wait for 20ns; g1 <= not g1;
        wait for 20ns; g2 <= not g2;
        wait for 20ns; a <= not a;
        wait for 20ns; g1 <= not g1;
        wait for 20ns; g2 <= not g2;
    end process;
end behavior;
```



This is testbench written to achieve the specified output. It uses wait statements for timing and simply toggles thee signal using not.