ECEN302 Lab 2 - Various Hardware Designs of a Multiplexer

Joshua Benfell - 300433229

August 7, 2020

1 Objectives

The primary objectives of this lab were to learn and use the different modelling types offered by the VHDL language. There are 3 different ways of modelling available in the language, these are: Dataflow, Structural and Behavioural Modelling. To illustrate the concepts further, the circuit being modelled will be the same for all types, that being a multiplexer. To further communicate the uses of each type, small extensions to a base multiplexer design will be made, to change it from a one bit mux to a two bit mux. It is important the different models are learnt and understood as they are the main foundation of circuit design on a FPGA.

2 Methodology

The first type of modelling that was implemented was the dataflow modelling. This was done with signal assignment using the '¡=' operator in the parallel architecture block of VHDL. This functions by pushing the result of *(y and s) or (x and (not s))* into the mux output signal every clock cycle. It is done so in parallel with other operations, however, there are no more than this one at this point. This was then uploaded to the FPGA to test functionality. When testing this physically, one of the switches was constantly on, which was not the intended behaviour. To verify the circuit design was correct, a test bench was created. Once simulated, this returned the results that the circuit design was correct. So a different FPGA board was used to test that it was just the board not functioning correctly. This turned out to be the case and the design now functioned correctly.

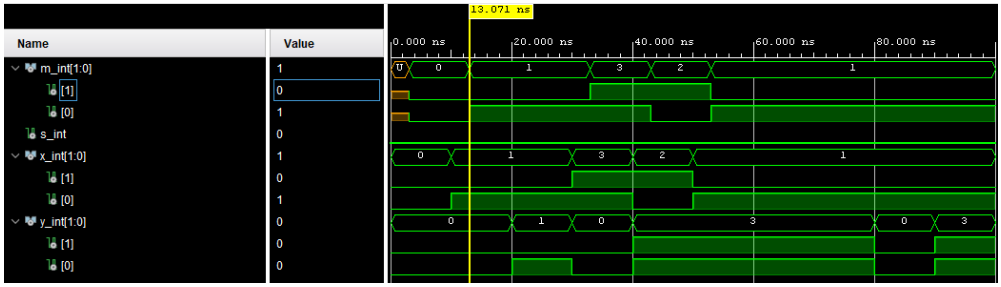

Figure 1: Timing Diagram of simulated delayed mux

To extend this functionality and demonstrate the parallelism of dataflow modelling, two multiplexers were implemented in the same design. This was done by adding a copy of

the line used for the single mux as well as changing the mux inputs and output to the *STD_LOGIC_VECTOR* data type. In this each bit of the inputs and output were selected to make understanding simpler, however, this could be done by using the whole vector in the operation. By separating them, we can treat them as separate operations and in fig. 1, it can be seen that the mux inputs are switching at the same time when told to. The next thing added to the mux was a transitional delay of 3ns, which can be seen in fig. 1 as the input changes at 10ns, but the output changes at 13ns.

Structural modelling differs from dataflow modelling in that the model is built up from other pre-existing blocks, whether made or primitive. For this exercise the primitive blocks *and2*, *or2* and *inv* are used as the and, or and not gates respectively. The main difficulty with this exercise was figuring out what these primitive gates were. Once that was done, they were registered in the model so that it had reference to what was to be used, then the ports were mapped to the respective signals. Because this was being built up from bare gates new variables/signals needed to be created for the intermediary stages so that the output of each gate could be stored. As this was a two bit mux, it could possibly be compressed and simplified by not doing each bit individually, but doing each input and output as it's vector. This form of modelling was much more tedious and code size intensive, which does not bode well to it offering benefits as an option of modelling. However, being able to abstract the design into design blocks will make the design simpler and easier to split up and implement as well as maintain.

The final type of modelling that will be implemented is behavioural modelling. This form of modelling can be both parallel and sequential, it makes the use of processes which allows code to be written in a similar manner to other languages while also running sequentially. This was done as both a single bit and two bit mux, both of these are functionally the same, the main difference is that one uses *STD_LOGIC* and the other uses *STD_LOGIC_VECTOR*. The logic for this was inverted to all the other muxes, this was found to be because the code that was in the lab script regarding this section had the select bits in reverse order to all the previous descriptions, so when this was changed, it matched up with what was expected.

Now that a 2 input mux module has been created, it can be used in other VHDL files to implement it's behaviour. To demonstrate this, a 3 input 2 bit mux was designed. I used one of the implementations of a 2 input 2 bit multiplexer done prior and followed structural modelling principles to implement this 3 bit mux. It is implemented such that the output of a first stage mux (so a intermediary variable is required) is set as the input to a second mux with a third input, the select bits can then be used to determine if the final output is the output of the first mux, or the third input. This was a tricky design to implement as finding a nice way to lay it out on the in line switches didn't allow good separation.

The final task implemented on the FPGA was a BCD to seven segment display. This was done using a lookup table and dataflow modelling, by utilising the *with x select* statement. This statement was populated with all possible inputs and what output this lined up with. As the seven segment display was used before in a previous lab exercise, setting up the displaying was trivial. However, this exercise asked for only one number of the 8 to display number. This was trickier as it was not clear from the get go how the anodes worked. After testing, it was found that setting the AN pins high would turn off a display with the LSB being the right most display. After uploading it to the FPGA, you were able to count with the switches and display the appropriate decimal number on a seven segment display. To improve this, multiple digits need to be implemented so that numbers like 15 can be displayed. This requires setting the displays separately as writing to the CA pins sets all displays, however, this was beyond the scope of this exercise.

3 Code

3.1 2.2.1 - Dataflow Modelling

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

entity MUX is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : in STD_LOGIC;
           m : out STD_LOGIC);
end MUX;

architecture Behavioral of MUX is

begin
    m <= (y and s) or (x and (not s));

end Behavioral;
```

code/dataflow_modelling.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MUX_tb is
--  Port ( );
end MUX_tb;

architecture Behavioral of MUX_tb is

Component MUX
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : in STD_LOGIC;
           m : out STD_LOGIC);
end Component;

signal x : std_logic := '1';
signal y : std_logic := '0';
signal s : std_logic := '0';
signal m : std_logic := '0';



begin

uut: MUX PORT MAP(
    x => x,
    y => y,
    s => s,
    m => m
);

process begin
    s <= '1';
    wait for 200ns;
    s <= '0';
    wait for 200ns;
```

```
end process;

end Behavioral;
```

code/dataflow_modelling_tb.vhd

3.2 2.2.2 - Two Bit Dataflow Modelling

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

entity twobitmux is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           y : in STD_LOGIC_VECTOR (1 downto 0);
           s : in STD_LOGIC;
           m : out STD_LOGIC_VECTOR (1 downto 0));  -- Was left as in, had
               to change to output.
end twobitmux;

architecture Behavioral of twobitmux is

begin
    m(1) <= (y(1) and s) or (x(1) and (not s));
    m(0) <= (y(0) and s) or (x(0) and (not s));

end Behavioral;
```

code/dataflow_modelling_two_bit.vhd

3.3 2.2.3 - Adding Net Delay

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

entity twobitmux is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           y : in STD_LOGIC_VECTOR (1 downto 0);
           s : in STD_LOGIC;
           m : out STD_LOGIC_VECTOR (1 downto 0));  -- Was left as in, had
               to change to output.
end twobitmux;

architecture Behavioral of twobitmux is

begin
    m(1) <= (y(1) and s) or (x(1) and (not s)) after 3ns;
    m(0) <= (y(0) and s) or (x(0) and (not s)) after 3ns;

end Behavioral;
```

code/dataflow_modelling_delay.vhd

3.4 2.3.1 - Structural Modelling

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

entity twobitmux is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           y : in STD_LOGIC_VECTOR (1 downto 0);
```

```
7             s : in STD_LOGIC;
8             m : out STD_LOGIC_VECTOR (1 downto 0));  -- Was left as in, had
                  to change to output.
9 end twobitmux;
10
11 architecture Behavioral of twobitmux is
12
13     -- Declare Signals to store intermediary Stages
14     Signal a_int0 : STD_LOGIC;
15     Signal a_int1 : STD_LOGIC;
16     Signal a_int2 : STD_LOGIC;
17     Signal a_int3 : STD_LOGIC;
18     signal not_s : STD_LOGIC;
19
20     -- Declare all primitive gates to use
21
22     Component and2
23         port (
24             i0, i1 : in STD_LOGIC;
25             O : out STD_LOGIC
26         );
27     end component;
28
29     component or2
30         port (
31             i0, i1 : in STD_LOGIC;
32             O : out STD_LOGIC
33         );
34     end component;
35
36     component inv
37         port (
38             i : in STD_LOGIC;
39             O : out STD_LOGIC
40         );
41     end component;
42
43     begin
44
45         NOT_COMP : inv port map (
46             i => s,
47             O => not_s
48         );
49
50         AND_COMP0 : and2 port map (
51             i0 => x(0),
52             i1 => not_s,
53             O  => a_int0
54         );
55         AND_COMP1 : and2 port map (
56             i0 => y(0),
57             i1 => s,
58             O  => a_int1
59         );
60         OR_COMP1 : or2 port map (
61             i0 => a_int0,
62             i1 => a_int1,
63             O  => m(0)
64         );
```

```
65
66
67          AND_COMP2 : and2 port map (
68              i0 => x(1),
69              i1 => not_s,
70              O  => a_int2
71          );
72          AND_COMP3 : and2 port map (
73              i0 => y(1),
74              i1 => s,
75              O  => a_int3
76          );
77          OR_COMP2 : or2 port map (
78              i0 => a_int2,
79              i1 => a_int3,
80              O  => m(1)
81          );
82
83  end Behavioral;
```

code/structural_modelling.vhd

3.5 2.4.1 - Behavioural Modelling

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MUX is
5      Port ( x : in STD_LOGIC;
6             y : in STD_LOGIC;
7             s : in STD_LOGIC;
8             m : out STD_LOGIC);
9  end MUX;
10
11 architecture Behavioral of MUX is
12 begin
13
14     process (x, y, s)
15         begin
16             if(s='0') then
17                 m <= y;
18             else
19                 m <= x;
20             end if;
21     end process;
22 end Behavioral;
```

code/behavioural_modelling.vhd

3.6 2.4.2 - Two Bit Behavioural Modelling

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MUX is
5      Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
6             y : in STD_LOGIC_VECTOR (1 downto 0);
7             s : in STD_LOGIC;
8             m : out STD_LOGIC_VECTOR (1 downto 0));
9  end MUX;
```

```
10
11 architecture Behavioral of MUX is
12 begin
13
14     process (x, y, s)
15         begin
16             if(s='0') then  -- This is backwards to the previous exercises
17                 m <= y;
18             else
19                 m <= x;
20             end if;
21     end process;
22 end Behavioral;
```

code/behavioural_modelling_two_bit.vhd

3.7 2.5.1 - 3 to 1 MUX

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity three_to_one_mux is
5      Port ( u : in STD_LOGIC_VECTOR (1 downto 0);
6             y : in STD_LOGIC_VECTOR (1 downto 0);
7             w : in STD_LOGIC_VECTOR (1 downto 0);
8             s : in STD_LOGIC_VECTOR (1 downto 0);
9             m : out STD_LOGIC_VECTOR (1 downto 0)
10        );
11 end three_to_one_mux;
12
13 architecture Behavioral of three_to_one_mux is
14
15     component twobitmux  -- Register a two bit mux module from one of the
           previous sections
16         port (
17             x : in STD_LOGIC_VECTOR (1 downto 0);
18             y : in STD_LOGIC_VECTOR (1 downto 0);
19             s : in STD_LOGIC;
20             m : out STD_LOGIC_VECTOR (1 downto 0)
21         );
22     end component;
23     signal mux_out : std_logic_vector (1 downto 0);
24
25     begin
26         MUX1 : twobitmux port map (
27             x => u,
28             y => y,
29             s => s(0),
30             m => mux_out
31         );
32
33         MUX2 : twobitmux port map (
34             x => w,
35             y => mux_out,
36             s => s(1),
37             m => m
38         );
39
```

```
40 end Behavioral;
```

code/three_to_one_mux.vhd

3.8 2.5.2 - BCD to Seven Segment Display

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity bcdto7segment_dataflow is
5      Port ( x : in STD_LOGIC_VECTOR (3 downto 0);
6             an : out STD_LOGIC_VECTOR (7 downto 0);
7             seg_out : out STD_LOGIC_VECTOR (6 downto 0));
8  end bcdto7segment_dataflow;
9
10 architecture Behavioral of bcdto7segment_dataflow is
11
12 begin
13     an <= "11111110";
14     with x select
15     seg_out <= "1000000" when "0000", -- 0
16                "1111001" when "0001", -- 1
17                "0100100" when "0010", -- 2
18                "0110000" when "0011", -- 3
19                "0011001" when "0100", -- 4
20                "0010010" when "0101", -- 5
21                "0000010" when "0110", -- 6
22                "1111000" when "0111", -- 7
23                "0000000" when "1000", -- 8
24                "0010000" when "1001", -- 9
25                "1111111" when others;
26 end Behavioral;
```

code/bcdto7segment_dataflow.vhd