Zack Fravel

010646947

L001 4:10 - 5:55 PM

Lab 1 - Adder, Mux, and Register

2/1/16

Introduction

The goal of this lab was to get started with the ISE Design Suite/Xilinx Webpack and learn how to create new projects, compile VHDL code, and run simulations. Once we had a general feel for the program, the assignment for the lab was to write the VHDL code and test benches for a 16 bit adder, a 4:1 mux with 16 bit inputs, and a 16 bit register.

Approach

The high level design description for this lab is relatively straight forward, since we're dealing with such building-block pieces. We want to design a 16 bit adder, which allows two inputs to be added together using unsigned addition and accounts for overflow. We also want a 4:1 mux using 16 bit inputs, allowing us to transmit different 16 bit chunks of data depending on a selector. Finally the 16 bit register allows us to pass through data depending on the clock cycle.

Experimentation

The experimentation process began with writing VHDL code for each of the three designs. Most of the code we needed to complete the lab was actually given to us in lecture, maybe barring a few lines. The VHDL for all three designs is below.

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SixteenBitAdder is
   port (cin: in std_logic;
         cout: out std_logic;
         x: in std_logic(15 downto 0);
         y: in std_logic(15 downto 0);
         s: out std_logic(15 downto 0));

end SixteenBitAdder;

architecture Behavioral of SixteenBitAdder is
   signal sum: std_logic_vector(16 downto 0);
begin
   sum <= ('0'&x)+y+cin;
   s <= sum(15 downto 0);
   cout <= sum(16);


end Behavioral;
```

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instant
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux4 is
   port (s: in std_logic_vector(1 downto 0);
         w0,w1,w2,w3: in std_logic_vector(15 downto 0);
         f: out std_logic_vector(15 downto 0));
end mux4;

architecture Behavioral of mux4 is

begin

   with s select
      f <= w0 when "00",
           w1 when "01",
           w2 when "10",
           w3 when "11";

end Behavioral;
```

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instar
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Register16 is
   port (resetn, clock: in std_logic;
         D: in std_logic_vector(15 downto 0);
         Q: out std_logic_vector(15 downto 0));
end Register16;

architecture Behavioral of Register16 is

begin
   process(clock, resetn)
   begin
      if resetn = '0' then
         Q <= (others=>'0');
      elsif rising_edge(clock) then
         Q <= D;
      end if;
   end process;
end Behavioral;
```

Most of the knowledge needed for writing these came from Digital Design, since these are pretty basic operations. With the adder, all that was needed was the use of another signal, sum, that is used to temporarily assign the sum so it can be split into an “s” output (sum) and the “cout” output for the carry out. For the mux, a “with __ select” was used to change the output depending on the value of the two-bit number “s.” Finally, with the register a process was needed. In the process, the compiler looks for changes in both the clock and the resent inputs. If resent = ‘0’ then the output will be redirected to ‘0.’ However, in all other cases and when the clock is on a rising edge, the output is redirected to whatever the value of D is.

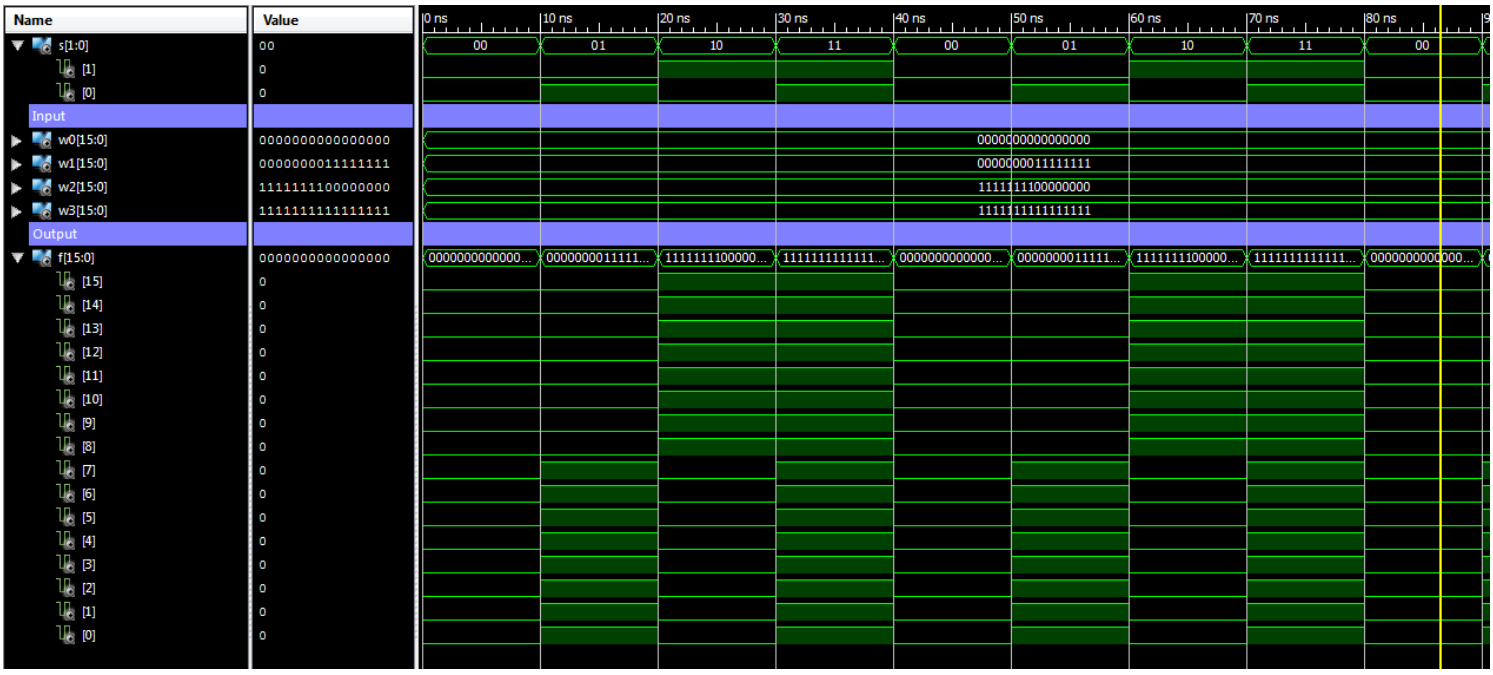
Once the VHDL was completed, all of them were able to compile and save with no errors, it was time to create the test bench files for each so the simulation could be run. I ran into an issue with my 16 bit adder, the code compiles with no errors however when I attempt to generate a test bench file, the program doesn’t do it properly and I am lost as to what to do from there. However, generating a test bench file for the mux and the 16 bit register was no problem at all. Below are the test benches for the 4:1 mux and the 16 bit register.

```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: mux4 PORT MAP (
          s => s,
          w0 => w0,
          w1 => w1,
          w2 => w2,
          w3 => w3,
          f => f
        );

   drive : process
   begin
         w0 <= "0000000000000000";
         w1 <= "0000000011111111";
         w2 <= "1111111100000000";
         w3 <= "1111111111111111";
         s <= "00";
      wait for tick;
         w0 <= "0000000000000000";
         w1 <= "0000000011111111";
         w2 <= "1111111100000000";
         w3 <= "1111111111111111";
         s <= "01";
      wait for tick;
         w0 <= "0000000000000000";
         w1 <= "0000000011111111";
         w2 <= "1111111100000000";
         w3 <= "1111111111111111";
         s <= "10";
      wait for tick;
         w0 <= "0000000000000000";
         w1 <= "0000000011111111";
         w2 <= "1111111100000000";
         w3 <= "1111111111111111";
         s <= "11";
      wait for tick;
         w0 <= "0000000000000000";
         w1 <= "0000000011111111";
         w2 <= "1111111100000000";
         w3 <= "1111111111111111";
         s <= "00";
   end process drive;

END;
```

Results

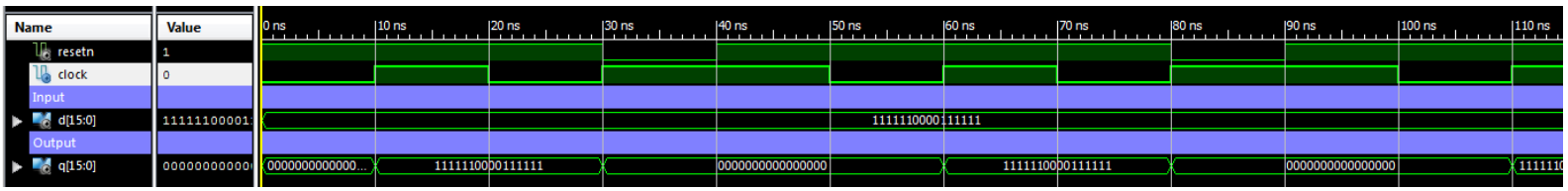
```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: Register16 PORT MAP (
          resetn => resetn,
          clock => clock,
          D => D,
          Q => Q
        );

   -- Stimulus process
   drive : process
   begin
         D <= "1111110000111111";
         clock <= '0';
         resetn <= '1';
      wait for tick;
         D <= "1111110000111111";
         clock <= '1';
         resetn <= '1';
      wait for tick;
         D <= "1111110000111111";
         clock <= '0';
         resetn <= '1';
      wait for tick;
         D <= "1111110000111111";
         clock <= '1';
         resetn <= '0';
      wait for tick;
         D <= "1111110000111111";
         clock <= '1';
         resetn <= '1';
      wait for tick;
         D <= "1111110000111111";
         clock <= '1';
         resetn <= '1';
   end process drive;

END;
```

Using the test benches above, I was able to obtain some pretty good waveforms that should demonstrate all the functionality of both the mux and the register. We'll start with the 4:1 16 bit mux.



It can be seen that every 10 ns, the selector, s, changes its value. It goes from "00" to "01" to "10" and finally to "11." If you follow the selector and look at the output below, it can be seen that the output also changes based on the value of the selector signal. Notice that the inputs and outputs are 16 bits. I made the 4 inputs mimic the selector value in 16 bits to more clearly show how the value is changing. This is the full functionality of the 4:1 mux. The test bench shows the full functionality because it goes through a full "cycle" of all possible selector values.

Once the mux was completed it was time to move on to the 16 bit register. A register works similarly to a D-Latch, with the value of a signal D being transmitted through to the output, Q, only on a rising clock edge. This circuit also contains an asynchronous reset input, which makes the output '0' anytime the reset input is also '0.' Below is the waveform digram for the 16 bit register.



It can be observed that whenever “resetn” is set to ‘0’, the value of Q is also set to ‘0.’ However, whenever resetn = ‘1,’ that allows the circuit to pass through the value of D to Q on the next rising clock edge. The value of Q doesn’t depend on the clock falling, only a rising edge. Since the reset is asynchronous, the reset does not depend on the rising clock edge.

Conclusion

All in all, all three of my designs that allowed testing were functional. The adder I was not able to ever get a test bench file started with because the program did not work properly, I asked for assistance and no one could figure out the problem. The mux and register however work exactly as advertised! This lab was very useful in learning the general workflow of using the ISE lab tools and what is required to fully test an implementation of a circuit design.