The City College of New York

Computer Organization Lab

CSC 34300

(Spring 2024)

LAB 4

8 Bit Shift_ADD_Multiplier

Mahir Asef

Instructor: Albi Arapi

Date: 04.18.2024

Table of Contents

Objective:

The purpose of this lab experiment is to introduce ourselves to working with add shift multiplier in Quartus. By the end of this experiment, we will be able to understand how the add shift multiplier operates on multiplier and multiplicands as well as how it stores the product at each iteration in the product register. We will learn how to create the components necessary to create an 8 bit add shift multiplier by using D flip flop, full adder, ripple carry adder etc. Throughout the experiment, we will be replicating VHDL code for each component part and then connect them together to build the entire design of the add shift multiplier. We will also do a design verification using modelSim at the end to demonstrate the functionality of our circuit for which we will also replicate testbench code.

Functionality and Specifications:

Controller:

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.std_logic_arith.all;
 4  use ieee.std_logic_unsigned.all;
 5 entity Controller is
 6 port (reset : in std_logic ;
 7  clk : in std_logic ;
 8  START : in std_logic ;
 9  LSB : in std_logic ;
10  ADD_cmd : out std_logic ;
11  SHIFT_cmd : out std_logic ;
12  LOAD_cmd : out std_logic ;
13  STOP : out std_logic);
14  end;
15 architecture rtl of Controller is
16  signal temp_count : std_logic_vector(2 downto 0);
17  -- declare states
18  type state_typ is (IDLE, INIT, TEST, ADD, SHIFT);
19  signal state : state_typ;
20 begin
21 process (clk, reset)
22  begin
23 if reset='0' then
24  state <= IDLE;
25  temp_count <= "000";
26 elsif (clk'event and clk='1') then
27 case state is
28  when IDLE =>
29 if START = '1' then
30  state <= INIT;
31 else
32  state <= IDLE;
33  end if;
34  when INIT =>
35  state <= TEST;
36  when TEST =>
37 if LSB = '0' then
38  state <= SHIFT;
39 else
40  state <= ADD;
41  end if;
42  when ADD =>
43  state <= SHIFT;
44  when SHIFT =>
45 if temp_count = "111" then -- verify if finished
```

```
46   temp_count <= "000"; -- re-initialize counter
47   state <= IDLE; -- ready for next multiply
48  else
49   temp_count <= temp_count + 1; -- increment counter
50   state <= TEST;
51  end if;
52  end case;
53  end if;
54  end process;
55   STOP <= '1' when state = IDLE else '0';
56   ADD_cmd <= '1' when state = ADD else '0';
57   SHIFT_cmd <= '1' when state = SHIFT else '0';
58   LOAD_cmd <= '1' when state = INIT else '0';
59   end rtl;
```

D Flip Flop:

```
 1   library ieee;
 2   use ieee.std_logic_1164.all;
 3  entity DFF is
 4  port (reset : in std_logic ;
 5   clk : in std_logic ;
 6   D : in std_logic ;
 7   Q : out std_logic);
 8   end;
 9   architecture behav of DFF is
10  begin
11  process (clk, reset)
12   begin
13  if reset='0' then
14   Q <= '0'; -- clear register
15  elsif (clk'event and clk='1') then
16   Q <= D; -- load register
17   end if;
18   end process;
19   end behav;
20
```

Multiplicand:

```
library ieee;
use ieee.std_logic_1164.all;
entity Multiplicand is
port (reset : in std_logic ;
A_in : in std_logic_vector (7 downto 0);
LOAD_cmd : in std_logic ;
RA : out std_logic_vector (7 downto 0));
end;
architecture struc of Multiplicand is
component DFF
port (
reset : in std_logic;
clk : in std_logic;
D : in std_logic;
Q : out std_logic
);
end component;
begin
DFFs: for i in 7 downto 0 generate
DFFReg:DFF port map (reset, LOAD_cmd, A_in(i), RA(i));
end generate;
end struc;
```

Multiplier:

```
library ieee;
use ieee.std_logic_1164.all;
entity Multiplier_Result is
port (reset : in std_logic ;
clk : in std_logic ;
B_in : in std_logic_vector (7 downto 0);
LOAD_cmd : in std_logic ;
SHIFT_cmd : in std_logic ;
ADD_cmd : in std_logic ;
Add_out : in std_logic_vector (7 downto 0);
C_out : in std_logic ;
RC : out std_logic_vector (15 downto 0);
LSB : out std_logic ;
RB : out std_logic_vector (7 downto 0));
end;
architecture rtl of Multiplier_Result is
signal temp_register : std_logic_vector(16 downto 0);
signal temp_Add : std_logic;
begin
process (clk, reset)
begin
if reset='0' then
temp_register <= (others =>'0'); -- initialize temporary register
temp_Add <= '0';
elsif (clk'event and clk='1') then
if LOAD_cmd = '1' then
temp_register (16 downto 8) <= (others => '0');
temp_register(7 downto 0) <= B_in; -- load B_in into register
end if;
if ADD_cmd = '1' then
temp_Add <= '1';
end if;
if SHIFT_cmd = '1' then
if temp_Add = '1' then
-- store adder output while shifting register right 1 bit
temp_Add <= '0';
temp_register <= '0' & C_out & Add_out & temp_register (7 downto 1);
```

```
38  else
39   -- no add - simply shift right 1 bit
40   temp_register <= '0' & temp_register (16 downto 1);
41  end if;
42  end if;
43  end if;
44  end process;
45   RB <= temp_register(15 downto 8);
46   LSB <= temp_register(0);
47  RC <= temp_register(15 downto 0);
48   end rtl;
```

Full Adder:

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3  entity Full_Adder is
4  port (X : in std_logic;
5   Y : in std_logic;
6   C_in : in std_logic;
7   Sum : out std_logic ;
8   C_out : out std_logic);
9   end;
10  architecture struc of Full_Adder is
11 begin
12  Sum <= X xor Y xor C_in;
13  C_out <= (X and Y) or (X and C_in) or (Y and C_in);
14  end struc;
15
```

Ripple Carry Adder:

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3  entity RCA is
4  port (RA : in std_logic_vector (7 downto 0);
5   RB : in std_logic_vector (7 downto 0);
6   C_out : out std_logic ;
7   Add_out : out std_logic_vector (7 downto 0));
8   end;
9  architecture struc of RCA is
10  signal c_temp : std_logic_vector(7 downto 0);
11 component Full_Adder
12 port (
13  X : in std_logic;
14  Y : in std_logic;
15  C_in : in std_logic;
16  Sum : out std_logic;
17  C_out : out std_logic
18  );
19  end component;
20  begin
21  c_temp(0) <= '0'; -- carry in of RCA is 0
22 Adders: for i in 7 downto 0 generate
23  -- assemble first 7 adders from 0 to 6
24 Low: if i/=7 generate
25  FA:Full_Adder port map (RA(i), RB(i), c_temp(i), Add_out(i), c_temp(i+1));
26  end generate;
27  -- assemble last adder
28 High: if i=7 generate
29  FA:Full_Adder port map (RA(7), RB(7), c_temp(i), Add_out(7), C_out);
30  end generate;
31  end generate;
32  end struc;
33
```

8 Bit Add Shift Multiplier:

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  --library synplify; -- required for synthesis
 4  --use synplify.attributes.all; -- required for synthesis
 5  entity Multiplier is
 6  port (
 7  A_in : in std_logic_vector(7 downto 0 );
 8  B_in : in std_logic_vector(7 downto 0 );
 9  clk : in std_logic;
10  reset : in std_logic;
11  START : in std_logic;
12  RC : out std_logic_vector(15 downto 0 );
13  STOP : out std_logic);
14  end Multiplier;
15  use work.all;
16  architecture rtl of Multiplier is
17  signal ADD_cmd : std_logic;
18  signal Add_out : std_logic_vector(7 downto 0 );
19  signal C_out : std_logic;
20  signal LOAD_cmd : std_logic;
21  signal LSB : std_logic;
22  signal RA : std_logic_vector(7 downto 0 );
23  signal RB : std_logic_vector(7 downto 0 );
24  signal SHIFT_cmd : std_logic;
25  component RCA
26  port (
27  RA : in std_logic_vector(7 downto 0 );
28  RB : in std_logic_vector(7 downto 0 );
29  C_out : out std_logic;
30  Add_out : out std_logic_vector(7 downto 0 )
31  );
32  end component;
33  component Controller
34  port (
35  reset : in std_logic;
36  clk : in std_logic;
37  START : in std_logic;
38  LSB : in std_logic;
39  ADD_cmd : out std_logic;
40  SHIFT_cmd : out std_logic;
41  LOAD_cmd : out std_logic;
42  STOP : out std_logic
43  );
44  end component;
45  component Multiplicand
```

```
46  port (
47  reset : in std_logic;
48  A_in : in std_logic_vector(7 downto 0 );
49  LOAD_cmd : in std_logic;
50  RA : out std_logic_vector(7 downto 0 )
51  );
52  end component;
53  component Multiplier_Result
54  port (
55  reset : in std_logic;
56  clk : in std_logic;
57  B_in : in std_logic_vector(7 downto 0 );
58  LOAD_cmd : in std_logic;
59  SHIFT_cmd : in std_logic;
60  ADD_cmd : in std_logic;
61  Add_out : in std_logic_vector(7 downto 0 );
62  C_out : in std_logic;
63  RC : out std_logic_vector(15 downto 0 );
64  LSB : out std_logic;
65  RB : out std_logic_vector(7 downto 0 )
66  );
67  end component;
68  begin
69  inst_RCA: RCA
70  port map (
71  RA => RA(7 downto 0),
72  RB => RB(7 downto 0),
73  C_out => C_out,
74  Add_out => Add_out(7 downto 0)
75  );
76  inst_Controller: Controller
77  port map (
78  reset => reset,
79  clk => clk,
80  START => START,
81  LSB => LSB,
82  ADD_cmd => ADD_cmd,
83  SHIFT_cmd => SHIFT_cmd,
84  LOAD_cmd => LOAD_cmd,
85  STOP => STOP
86  );
87  inst_Multiplicand: Multiplicand
88  port map (
89  reset => reset,
90  A_in => A_in(7 downto 0),
```

```
 91   LOAD_cmd => LOAD_cmd,
 92   RA => RA(7 downto 0)
 93   );
 94   inst_Multiplier_Result: Multiplier_Result
 95  port map (
 96   reset => reset,
 97   clk => clk,
 98   B_in => B_in(7 downto 0),
 99   LOAD_cmd => LOAD_cmd,
100   SHIFT_cmd => SHIFT_cmd,
101   ADD_cmd => ADD_cmd,
102   Add_out => Add_out(7 downto 0),
103   C_out => C_out,
104   RC => RC(15 downto 0),
105   LSB => LSB,
106   RB => RB(7 downto 0)
107   );
108   end rtl;
109
```

This is our full VHDL code for the 8 bits add shift multiplier. It's built by using controller component as well as a multiplicand component (A_in) for the 8-bit multiplicand input and a multiplier component (B_in) for the multiplier input which is also 8 bits. Multiplicand component is built from DFF. In addition, the design also a Full adder as well as a ripple carry adder. The add shift multiplier stores the product result in a 16-bit register named RC.

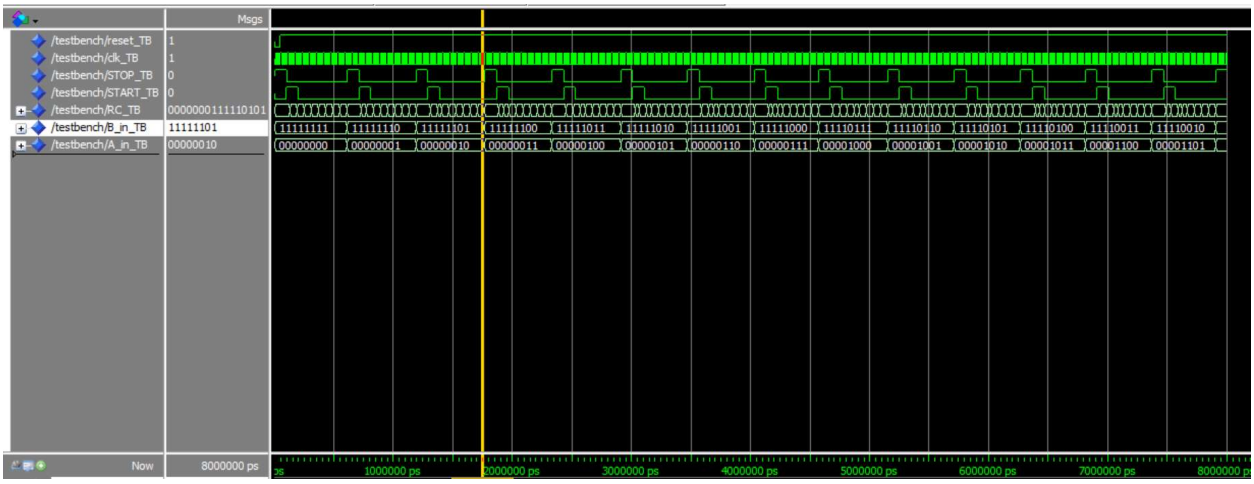
Testbench Code:

```
 1   library ieee;
 2   use ieee.std_logic_1164.all;
 3   use ieee.numeric_std.all;
 4   use std.textio.all; --required for file I/O
 5   use ieee.std_logic_textio.all; --required for file I/O
 6  entity TESTBENCH is
 7   end TESTBENCH;
 8  architecture BEHAVIORAL of TESTBENCH is
 9  component Multiplier
10  port (
11   A_in : in std_logic_vector(7 downto 0 );
12   B_in : in std_logic_vector(7 downto 0 );
13   clk : in std_logic;
14   RC : out std_logic_vector(15 downto 0 );
15   reset : in std_logic;
16   START : in std_logic;
17   STOP : out std_logic
18   );
19  end component;
20   signal A_in_TB, B_in_TB : std_logic_vector(7 downto 0 );
21   signal clk_TB, reset_TB, START_TB : std_logic;
22   signal STOP_TB : std_logic;
23   signal RC_TB: std_logic_vector(15 downto 0);
24   begin
25   -- instantiate the Device Under Test
26   inst_DUT : Multiplier
27  port map (
28   A_in => A_in_TB,
29   B_in => B_in_TB,
30   clk => clk_TB,
31   reset => reset_TB,
32   RC => RC_TB,
33   START => START_TB,
34   STOP => STOP_TB);
35   -- Generate clock stimulus
36  STIMULUS_CLK : process
37   begin
38   clk_TB <= '0';
39   wait for 10 ns;
40   clk_TB <= '1';
41   wait for 10 ns;
42   end process STIMULUS_CLK;
43   -- Generate reset stimulus
44  STIMULUS_RST : process
45   begin
```

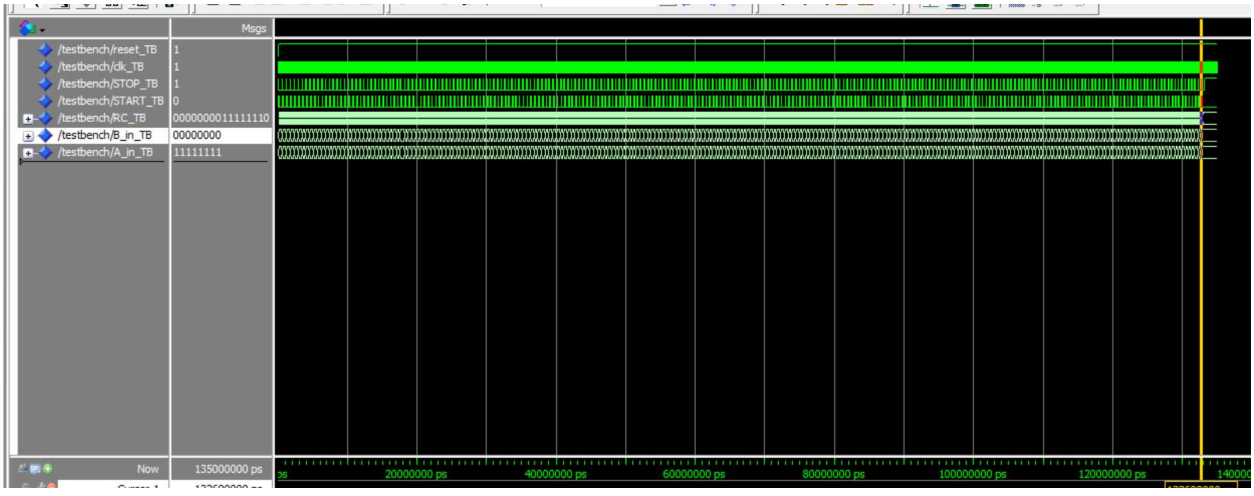
```
46  reset_TB <= '0';
47  wait for 50 ns;
48  reset_TB <= '1';
49  wait;
50  end process STIMULUS_RST;
51  -- Generate multiplication requests
52  STIMULUS_START : process
53  file logFile : text is out "bus_log.txt"; -- set output file name
54  variable L: line;
55  variable A_temp, B_temp, i : integer;
56  begin
57  write(L, string'("A B Result")); -- include heading in file
58  writeline(logFile,L);
59  A_temp := 0; -- start A at 0
60  B_temp := 255; -- start B at 255
61  i := 1;
62  for i in 1 to 256 loop
63  A_in_TB <= STD_LOGIC_VECTOR(to_unsigned(A_temp,8));
64  B_in_TB <= STD_LOGIC_VECTOR(to_unsigned(B_temp,8));
65  START_TB <= '0';
66  wait for 100 ns;
67  START_TB <= '1'; -- request the multiplier to start
68  wait for 100 ns;
69  START_TB <= '0';
70  wait until STOP_TB = '1'; -- wait for the multiplier to finish
71  hwrite(L, A_in_TB); -- insert hex value of A in file
72  write(L, string'(" "));
73  hwrite(L, B_in_TB); -- insert hex value of B in file
74  write(L, string'(" "));
75  hwrite(L, RC_TB); -- insert hex value of result in file
76  writeline(logFile,L);
77  A_temp := A_temp + 1; -- increment value of A (Multiplicand)
78  B_temp := B_temp - 1; -- decrement value of B (Multiplier)
79  end loop;
80  wait;
81  end process STIMULUS_START;
82  end BEHAVIORAL;
83
```

This is the testbench code that tests the add shift multiplier in modelsim in a work library. After simulation has been done the output regster that stores product and the inputs multiplier and multiplicand will be saved in a text file called “bus_log.txt”.

Simulation:



This is the modelsim simulation of our circuit design. Here we have ran our simulation for 8000ns for a more zoomed in view to see how the multiplier value and multiplicand values changes. As we roll fro left to right, we can see that when multiplier bit is 1, the multiplicand value get added to the product register value. In the meantime multipicand register shift 1 bit to the left while the multiplier register shifts 1 bit to the right and this process starts over and over again until the multiplier value is all 0 at each bit (in this case 00000000).



This simulation demonstartes the full process of 8 bit shift add multiplier which starts with the muliplier being set to 1111111 and multiplicand being set to 00000000. We run this simulation for 133000 ns and as we can see once it ends, all the multiplier values shift to the right making it 00000000 and all the multiplicand values shift to the left making it 11111111. This whole process lasts for 256 cycles since we are working with a 8 bit add shift multiplier.

Conclusion:

We have reached the end of this laboratory experiment. Throughout this experiment, we were able to correctly understand and explain what a add shift multiplier is and define the functionality of add shift multiplier as well as understanding how they store data or operate on data. Then we have started building our circuit that consists of connecting different components together. As shown above, we were able successfully create a controller, a DFF, a multiplier and a multiplicand a full adder and a ripple carry adder component and we have connected all of this together to build our add shift multiplier circuit. We were also able to successfully replicate a testbench code for simulation verfication on modelsim. We have ran two different modelsim simulations to show the fucntionality and process by rolling through our simulation and explaining how the values changes and get stored in the output as demonstarted above. Overall, this lab has been very informative on learning about the design implemenation of a 8 bit add shift multiplier which will allow us to build even more enhanced multiplier in future as well as expanding our knowledge on the realm of data storage and data process in hadware memory.