Laboratory Exercise 4: Implementing a Shift Add Multiplier module.

Zi Xuan Li

The Grove School of Engineering, The City College of New York

CSC 34300 5DE[43223]: Computer Systems Design Laboratory

Professor Izidor Gertner, TA: Albi Arapi

April 18th, 2024

Table of Contents

Objective:

The objective of this exercise is to understand, design, and verify the functionality of an 8-by-8 shift/add multiplier from its initial conception to the simulation using Quartus and ModelSim.

Functionality and Specifications:

```
--
-- Library Name : DSD
-- Unit Name : Controller
--
-- Date : Mon Oct 27 12:36:47 2003
--
-- Author :
--
-- Description : Controller is a finite state machine
-- that performs the following in each
-- state:
-- IDLE > samples the START signal
-- INIT > commands the registers to be
-- loaded
-- TEST > samples the LSB
-- ADD > indicates the Add result to be stored
-- SHIFT > commands the register to be shifted
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Controller is
    port (
        reset   : in std_logic ;
        clk     : in std_logic ;
        START   : in std_logic ;
        LSB     : in std_logic ;
        ADD_cmd : out std_logic ;
        SHIFT_cmd   : out std_logic ;
        LOAD_cmd    : out std_logic ;
        STOP    : out std_logic
    );
end entity Controller;

architecture rtl of Controller is
    signal temp_count : std_logic_vector(2 downto 0);

    -- declare states
    type state_typ is (IDLE, INIT, TEST, ADD, SHIFT);
    signal state : state_typ;

begin
    process (clk, reset)
    begin
        if reset = '0' then
            state <= IDLE;
            temp_count <= "000";
        elsif rising_edge(clk) then
            case state is
                when IDLE =>
                    if START = '1' then
                        state <= INIT;
                    else
                        state <= IDLE;
                    end if;
                when INIT =>
                    state <= TEST;
                when TEST =>
                    if LSB = '0' then
                        state <= SHIFT;
                    else
                        state <= ADD;
                    end if;
                when ADD =>
                    state <= SHIFT;
                when SHIFT =>
                    if temp_count = "111" then -- verify if finished
                        temp_count <= "000"; -- re-initialize counter
                        state <= IDLE; -- ready for next multiply
                    else
                        temp_count <= temp_count + 1; -- increment counter
                        state <= TEST;
                    end if;
            end case;
        end if;
    end process;

    STOP <= '1' when state = IDLE else '0';
    ADD_cmd <= '1' when state = ADD else '0';
    SHIFT_cmd <= '1' when state = SHIFT else '0';
    LOAD_cmd <= '1' when state = INIT else '0';

end architecture rtl;
```

Figure 1: VHDL code of the Controller component.

```
--
-- Library Name : DSD
-- Unit Name : DFF
--
-- Date : Mon Oct 27 13:32:59 2003
--
-- Author :
--
-- Description : DFF is an active high D flip flop
-- with asynchronous clear.
--

library ieee;
use ieee.std_logic_1164.all;

entity Dflipflop is
    port (
        reset   : in std_logic ;
        clk     : in std_logic ;
        D       : in std_logic ;
        Q       : out std_logic
    );
end entity Dflipflop;

architecture behav of Dflipflop is
begin
    process (clk, reset)
    begin
        if reset = '0' then
            Q <= '0'; -- clear register
        elsif (clk'event and clk = '1') then
            Q <= D; -- load register
        end if;
    end process;
end architecture behav;
```

Figure 2: VHDL code of the DFF component.

```
--
-- Library Name : DSD
-- Unit Name : Multiplicand
--
-- Date : Mon Oct 27 13:32:59 2003
--
-- Author :
--
-- Description : Multiplicand is an 8-bit register
-- that is loaded when the LOAD_cmd is
-- received and cleared with reset.
--

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 entity Multiplicand;

architecture struc of Multiplicand is
    component Dflipflop
        port (
            reset    : in std_logic;
            clk      : in std_logic;
            D        : in std_logic;
            Q        : out std_logic
        );
    end component;

begin
    Dflipflops: for i in 7 downto 0 generate
        DflipflopReg: Dflipflop port map (reset, LOAD_cmd, A_in(i), RA(i));
    end generate;

end architecture struc;
```

Figure 3: VHDL code of the Multiplicand component.

```vhdl
--
-- Library Name : DSD
-- Unit Name : Multiplier_Result
--
-- Date : Mon Oct 27 14:13:51 2003
--
-- Author :
--
-- Description : Multiplier_Result performs the
-- following:
-- > loads B_in into register upon
-- receiving LOAD_cmd
-- > loads Adder output into register
-- upon receiving ADD_cmd
-- > shifts register right upon
-- receiving SHIFT_cmd
--

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 entity Multiplier_Result;

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 rising_edge(clk) 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);
                else
                    -- no add - simply shift right 1 bit
                    temp_register <= '0' & temp_register(16 downto 1);
                end if;
            end if;
        end if;
    end process;

    RB <= temp_register(15 downto 8);
    LSB <= temp_register(0);
    RC <= temp_register(15 downto 0);
end rtl;
```

Figure 4: VHDL code of the Multiplier Result component.

```
--
-- Library Name : DSD
-- Unit Name : Full_Adder
--
-- Date : Wed Sep 24 12:50:50 2003
--
-- Author :
--
-- Description : Basic Full Adder Block
--

library ieee;
use ieee.std_logic_1164.all;

entity Full_Adder is
    port (
        X       : in std_logic;
        Y       : in std_logic;
        C_in    : in std_logic;
        Sum     : out std_logic ;
        C_out   : out std_logic
    );
end entity Full_Adder;

architecture struc of Full_Adder is
begin
    Sum <= X xor Y xor C_in;
    C_out <= (X and Y) or (X and C_in) or (Y and C_in);
end architecture struc;
```

Figure 5: VHDL code of the Full Adder component.

```
--
-- Library Name : DSD
-- Unit Name : RCA
--
-- Date : Wed Sep 24 12:50:50 2003
--
-- Author :
--
-- Description : RCA is an 8-bit ripple carry
-- adder composed of 8 basic full
-- adder blocks.
--

library ieee;
use ieee.std_logic_1164.all;

entity RCA is
    port (
        RA      : in std_logic_vector (7 downto 0);
        RB      : in std_logic_vector (7 downto 0);
        C_out   : out std_logic ;
        Add_out : out std_logic_vector (7 downto 0)
    );
end entity RCA;

architecture struc of RCA is
    signal c_temp : std_logic_vector(7 downto 0);

    component Full_Adder
        port (
            X       : in std_logic;
            Y       : in std_logic;
            C_in    : in std_logic;
            Sum     : out std_logic;
            C_out   : out std_logic
        );
    end component;

begin
    c_temp(0) <= '0'; -- carry in of RCA is 0

    Adders: for i in 7 downto 0 generate
        -- assemble first 7 adders from 0 to 6
        Low: if i /= 7 generate
            FA: Full_Adder port map (RA(i), RB(i), c_temp(i), Add_out(i), c_temp(i+1));
        end generate;

        -- assemble last adder
        High: if i = 7 generate
            FA: Full_Adder port map (RA(7), RB(7), c_temp(i), Add_out(7), C_out);
        end generate;
    end generate;
end architecture struc;
```

Figure 6: VHDL code of the RCA component.

```
--
-- Library Name : DSD
-- Unit Name : RCA4
--
-- Date : Wed Sep 24 12:50:50 2003
--
-- Author :
--
-- Description : RCA is an 4-bit ripple carry
-- adder composed of 8 basic full
-- adder blocks.
--

library ieee;
use ieee.std_logic_1164.all;

entity RCA4 is
    port (
        C_in    : in std_logic;
        RA      : in std_logic_vector (3 downto 0);
        RB      : in std_logic_vector (3 downto 0);
        C_out   : out std_logic ;
        Add_out : out std_logic_vector (3 downto 0)
    );
end entity RCA4;

architecture rtl of RCA4 is
    signal c_temp : std_logic_vector(3 downto 1);

    component Full_Adder
        port (
            X       : in std_logic;
            Y       : in std_logic;
            C_in    : in std_logic;
            Sum     : out std_logic;
            C_out   : out std_logic
        );
    end component;

begin
    Adders: for i in 3 downto 0 generate
        Low: if i = 0 generate
            FA: Full_Adder port map (RA(0), RB(0), C_in, Add_out(0), c_temp(i+1));
        end generate;

        Mid: if (i > 0 and i < 3) generate
            FA: Full_Adder port map (RA(i), RB(i), c_temp(i), Add_out(i), c_temp(i+1));
        end generate;

        High: if i = 3 generate
            FA: Full_Adder port map (RA(3), RB(3), c_temp(i), Add_out(3), C_out);
        end generate;
    end generate;
end architecture rtl;
```

Figure 7: VHDL code of the RCA4 component.

```
--
-- Library Name : DSD
-- Unit Name : CSA8
--
-- Date : Wed Sep 24 12:50:50 2003
--
-- Author :
--
-- Description : CSA8 is an 8 bit carry select adder
--

library ieee;
use ieee.std_logic_1164.all;

entity CSA8 is
    port (
        RA      : in std_logic_vector (7 downto 0);
        RB      : in std_logic_vector (7 downto 0);
        C_out   : out std_logic ;
        Add_out : out std_logic_vector (7 downto 0)
    );
end entity CSA8;

architecture rtl of CSA8 is
    signal c_temp : std_logic_vector(5 downto 0);
    signal add_temp0 : std_logic_vector(3 downto 0);
    signal add_temp1 : std_logic_vector(3 downto 0);

    component RCA4
        port (
            C_in    : in std_logic;
            RA      : in std_logic_vector(3 downto 0);
            RB      : in std_logic_vector(3 downto 0);
            Add_out : out std_logic_vector(3 downto 0);
            C_out   : out std_logic
        );
    end component;

begin
    c_temp(0) <= '0';
    c_temp(2) <= '0';
    c_temp(3) <= '1';

    inst_RCA1: RCA4
    port map (
        C_in    => c_temp(0),
        RA      => RA(3 downto 0),
        RB      => RB(3 downto 0),
        Add_out => Add_out(3 downto 0),
        C_out   => c_temp(1)
    );

    inst_RCA2: RCA4
    port map (
        C_in    => c_temp(2),
        RA      => RA(7 downto 4),
        RB      => RB(7 downto 4),
        Add_out => add_temp0,
        C_out   => c_temp(4)
    );

    inst_RCA3: RCA4
    port map (
        C_in    => c_temp(3),
        RA      => RA(7 downto 4),
        RB      => RB(7 downto 4),
        Add_out => add_temp1,
        C_out   => c_temp(5)
    );

    Add_out(7 downto 4) <= add_temp0 when c_temp(1) = '0' else
                           add_temp1 when c_temp(1) = '1' else
                           "zzzz";

    C_out <= (c_temp(1) and c_temp(5)) or c_temp(4);
end rtl;
```

Figure 8: VHDL code of the CSA8 component.

```
--
-- Library Name : DSD
-- Unit Name : Multiplier
--
-- Description : Complete multiplier
--

library ieee;
use ieee.std_logic_1164.all;

entity Multiplier is
    port (
        A_in    : in std_logic_vector(7 downto 0 );
        B_in    : in std_logic_vector(7 downto 0 );
        clk     : in std_logic;
        reset   : in std_logic;
        START   : in std_logic;
        RC      : out std_logic_vector(15 downto 0 );
        STOP    : out std_logic
    );
end Multiplier;

use work.all;

architecture rtl of Multiplier is
    signal ADD_cmd    : std_logic;
    signal Add_out    : std_logic_vector(7 downto 0 );
    signal C_out      : std_logic;
    signal LOAD_cmd   : std_logic;
    signal LSB        : std_logic;
    signal RA         : std_logic_vector(7 downto 0 );
    signal RB         : std_logic_vector(7 downto 0 );
    signal SHIFT_cmd  : std_logic;

    component RCA
        port (
            RA      : in std_logic_vector(7 downto 0 );
            RB      : in std_logic_vector(7 downto 0 );
            C_out   : out std_logic;
            Add_out : out std_logic_vector(7 downto 0 )
        );
    end component;

    component Controller
        port (
            reset   : in std_logic;
            clk     : in std_logic;
            START   : in std_logic;
            LSB     : in std_logic;
            ADD_cmd : out std_logic;
            SHIFT_cmd : out std_logic;
            LOAD_cmd : out std_logic;
            STOP    : out std_logic
        );
    end component;

    component Multiplicand
        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 component;
```

```
    component Multiplier_Result
        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 component;

begin
    inst_RCA: RCA
        port map (
            RA => RA(7 downto 0),
            RB => RB(7 downto 0),
            C_out => C_out,
            Add_out => Add_out(7 downto 0)
        );

    inst_Controller: Controller
        port map (
            reset => reset,
            clk => clk,
            START => START,
            LSB => LSB,
            ADD_cmd => ADD_cmd,
            SHIFT_cmd => SHIFT_cmd,
            LOAD_cmd => LOAD_cmd,
            STOP => STOP
        );

    inst_Multiplicand: Multiplicand
        port map (
            reset => reset,
            A_in => A_in(7 downto 0),
            LOAD_cmd => LOAD_cmd,
            RA => RA(7 downto 0)
        );

    inst_Multiplier_Result: Multiplier_Result
        port map (
            reset => reset,
            clk => clk,
            B_in => B_in(7 downto 0),
            LOAD_cmd => LOAD_cmd,
            SHIFT_cmd => SHIFT_cmd,
            ADD_cmd => ADD_cmd,
            Add_out => Add_out(7 downto 0),
            C_out => C_out,
            RC => RC(15 downto 0),
            LSB => LSB,
            RB => RB(7 downto 0)
        );
end rtl;
```

Figure 9: VHDL code of the Multiplier entity.

```
--
-- Library Name : DSD
-- Unit Name : Multiplier
--
-- Description : Complete multiplier
--

library ieee;
use ieee.std_logic_1164.all;

entity Multiplier is
    port (
        A_in    : in std_logic_vector(7 downto 0 );
        B_in    : in std_logic_vector(7 downto 0 );
        clk     : in std_logic;
        reset   : in std_logic;
        START   : in std_logic;
        RC      : out std_logic_vector(15 downto 0 );
        STOP    : out std_logic
    );
end Multiplier;

use work.all;

architecture rtl of Multiplier is
    signal ADD_cmd    : std_logic;
    signal Add_out    : std_logic_vector(7 downto 0 );
    signal C_out      : std_logic;
    signal LOAD_cmd   : std_logic;
    signal LSB        : std_logic;
    signal RA         : std_logic_vector(7 downto 0 );
    signal RB         : std_logic_vector(7 downto 0 );
    signal SHIFT_cmd  : std_logic;

    component RCA
        port (
            RA      : in std_logic_vector(7 downto 0 );
            RB      : in std_logic_vector(7 downto 0 );
            C_out   : out std_logic;
            Add_out : out std_logic_vector(7 downto 0 )
        );
    end component;

    component Controller
        port (
            reset   : in std_logic;
            clk     : in std_logic;
            START   : in std_logic;
            LSB     : in std_logic;
            ADD_cmd : out std_logic;
            SHIFT_cmd : out std_logic;
            LOAD_cmd : out std_logic;
            STOP    : out std_logic
        );
    end component;

    component Multiplicand
        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 component;
```

```
    component Multiplier_Result
        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 component;

begin
    inst_RCA: RCA
        port map (
            RA => RA(7 downto 0),
            RB => RB(7 downto 0),
            C_out => C_out,
            Add_out => Add_out(7 downto 0)
        );

    inst_Controller: Controller
        port map (
            reset => reset,
            clk => clk,
            START => START,
            LSB => LSB,
            ADD_cmd => ADD_cmd,
            SHIFT_cmd => SHIFT_cmd,
            LOAD_cmd => LOAD_cmd,
            STOP => STOP
        );

    inst_Multiplicand: Multiplicand
        port map (
            reset => reset,
            A_in => A_in(7 downto 0),
            LOAD_cmd => LOAD_cmd,
            RA => RA(7 downto 0)
        );

    inst_Multiplier_Result: Multiplier_Result
        port map (
            reset => reset,
            clk => clk,
            B_in => B_in(7 downto 0),
            LOAD_cmd => LOAD_cmd,
            SHIFT_cmd => SHIFT_cmd,
            ADD_cmd => ADD_cmd,
            Add_out => Add_out(7 downto 0),
            C_out => C_out,
            RC => RC(15 downto 0),
            LSB => LSB,
            RB => RB(7 downto 0)
        );
end rtl;
```

Figure 10: VHDL code of the Multiplier entity testbench.

Simulation:

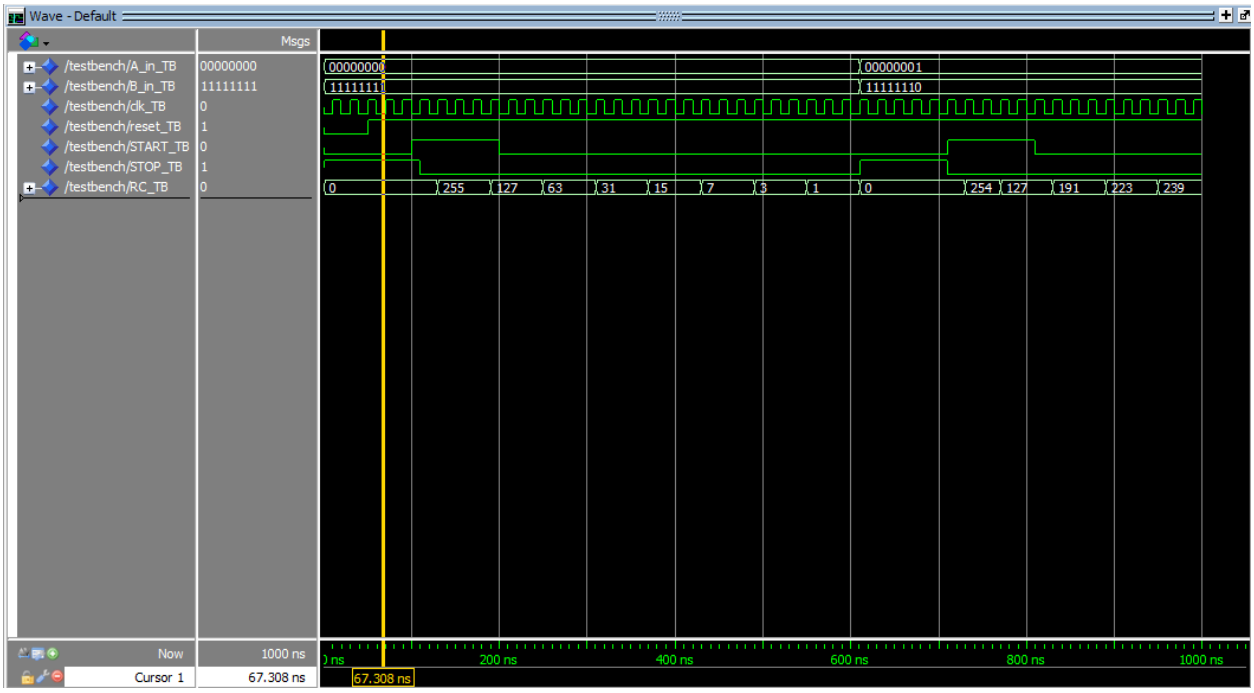

Figure 11: Simulation of the 8-by-8 bit multiplier.

The 8-by-8 bit shift/add multiplier implemented in the provided VHDL code performs multiplication utilizing a shift-and-add method. When initiated by the START signal, the multiplier module begins the multiplication process by loading the multiplicand (A_in) and multiplier (B_in). Through successive clock cycles, the multiplier shifts the multiplicand leftward and accumulates the result based on the current bit of the multiplier. This process continues until the multiplication is complete, indicated by the assertion of the STOP signal.

Conclusion:

This exercise offered a deeper understanding of designing and testing a shift/add multiplier. I successfully implemented an 8-by-8 bit multiplier using the add and shift method coded in VHDL. Using simulations, I was able to verify the correctness of the module and I

learned how the parametrization of shift/add multiplier works alongside the functionalities of its input ports, control signals, output port.