Device Utilization Summary

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	2,010	11,440	17%	
Number used as Flip Flops	2,010			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	1,666	5,720	29%	
Number used as logic	1,390	5,720	24%	
Number using O6 output only	664			
Number using O5 output only	40			
Number using O5 and O6	686			
Number used as ROM	0			
Number used as Memory	8	1,440	1%	
Number used as Dual Port RAM	0			
Number used as Single Port RAM	0			
Number used as Shift Register	8			
Number using O6 output only	3			
Number using O5 output only	0			
Number using O5 and O6	5			
Number used exclusively as route-thrus	268			
Number with same-slice register load	261			
Number with same-slice carry load	7			
Number with other load	0			
Number of occupied Slices	778	1,430	54%	
Number of MUXCYs used	1,000	2,860	34%	
Number of LUT Flip Flop pairs used	2,813			
Number with an unused Flip Flop	1,085	2,813	38%	
Number with an unused LUT	1,147	2,813	40%	
Number of fully used LUT-FF pairs	581	2,813	20%	
Number of unique control sets	109			
Number of slice register sites lost to control set restrictions	297	11,440	2%	
Number of bonded IOBs	35	102	34%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	0	64	0%	

Number of BUFIO2/BUFIO2_2CLKs	1	32	3%	
Number used as BUFIO2s	1			
Number used as BUFIO2_2CLKs	0			
Number of BUFIO2FB/BUFIO2FB_2CLKs	1	32	3%	
Number used as BUFIO2FBs	1			
Number used as BUFIO2FB_2CLKs	0			
Number of BUFG/BUFGMUXs	2	16	12%	
Number used as BUFGs	2			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	1	4	25%	
Number used as DCMs	1			
Number used as DCM_CLKGENs	0			
Number of ILOGIC2/ISERDES2s	0	200	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	200	0%	
Number of OLOGIC2/OSERDES2s	0	200	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	128	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	16	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	2	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	3.70			

List of warning messages

```
WARNING:HDLCompiler:92 - "D:\Skripsi\backup
program\136079_Hard_PWM_UART_Communication (28-5-
2018)\Hard_PWM_UART_Communication\data_read.vhd" Line 85: state_reg
should be on the sensitivity list of the process

WARNING:HDLCompiler:92 - "D:\Skripsi\backup
program\136079_Hard_PWM_UART_Communication (28-5-
2018)\Hard_PWM_UART_Communication\data_read.vhd" Line 92: state_reg
should be on the sensitivity list of the process

WARNING:Xst:1710 - FF/Latch <out_servo_data_reg_8> (without init
value) has a constant value of 0 in block <Inst_data_read>. This
FF/Latch will be trimmed during the optimization process.

WARNING:Xst:2404 -  FFs/Latches <out_servo_data_reg<19:8>> (without
init value) have a constant value of 0 in block <data_read>.

WARNING:Xst:1710 - FF/Latch <s_reg_3> (without init value) has a
constant value of 0 in block <UART_Receiver>. This FF/Latch will be
trimmed during the optimization process.

WARNING:Xst:1710 - FF/Latch <Inst_UART_System/UART_Receiver/s_reg_3>
(without init value) has a constant value of 0 in block
<Hard_PWM_UART_Comm>. This FF/Latch will be trimmed during the
optimization process.
```

Listing VHDL program of FIFO buffer

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
--use work.Main_Controller_Param.all;

entity FIFO_Buffer is
        generic (
                B : natural :=11; -- number of bits
                W : natural :=3 -- number of address bits
        );
        port (
                clk : in std_logic;
                rst : in std_logic;
                rd, wr : in std_logic;
                w_data : in std_logic_vector (B-1 downto 0);
                empty, full : out std_logic;
                r_data : out std_logic_vector (B-1 downto 0)
        );
end FIFO_Buffer;

architecture arch of FIFO_Buffer is
        type reg_file_type is array (2**W-1 downto 0) of std_logic_vector (B-1 downto 0);
        signal array_reg : reg_file_type;
        signal w_ptr_reg, w_ptr_next, w_ptr_succ : std_logic_vector (W-1 downto 0);
        signal r_ptr_reg, r_ptr_next, r_ptr_succ : std_logic_vector (W-1 downto 0);
        signal full_reg, empty_reg, full_next, empty_next : std_logic;
        signal wr_op : std_logic_vector (1 downto 0);
        signal wr_en : std_logic;

begin
-- ================================================================
-- register file
-- ================================================================
process(clk, rst)
begin
        if (rst='1') then
                array_reg <= (others =>(others => '0'));
        elsif falling_edge(clk) then
                if wr_en='1' then
                array_reg (to_integer(unsigned(w_ptr_reg))) <= w_data;
                end if;
        end if;
end process;
-- read port
r_data <= array_reg(to_integer(unsigned(r_ptr_reg)));
-- write enabled only when FIFO is not full;
wr_en <= wr and (not full_reg);

-- ================================================================
-- fifo control logic
-- ================================================================
-- register for read and write pointers
process (clk, rst)
begin
        if rst ='1' then
                w_ptr_reg <= (others => '0');
                r_ptr_reg <= (others => '0');
                full_reg <= '0';
                empty_reg <= '1';
        elsif falling_edge(clk) then
                w_ptr_reg <= w_ptr_next;
                r_ptr_reg <= r_ptr_next;
                full_reg <= full_next;
                empty_reg <= empty_next;
        end if;
end process;

-- successive pointer values
w_ptr_succ <= std_logic_vector(unsigned(w_ptr_reg)+1);
r_ptr_succ <= std_logic_vector(unsigned(r_ptr_reg)+1);

-- next-state logic for read and write pointers
wr_op <= wr & rd;
process (w_ptr_reg, w_ptr_succ, r_ptr_reg, r_ptr_succ, wr_op, empty_reg, full_reg)
begin
        w_ptr_next <= w_ptr_reg;
        r_ptr_next <= r_ptr_reg;
        full_next <= full_reg;
        empty_next <= empty_reg;
        case wr_op is
                when "00" => -- no operation
                when "01" => -- read
                        if (empty_reg /= '1') then -- not empty
                                r_ptr_next <= r_ptr_succ;
                                full_next <= '0';
                                if r_ptr_succ = w_ptr_reg then
                                        empty_next <= '1';
                                end if;
                        end if;
                when "10" => -- write
                        if (full_reg /= '1') then -- not full
                                w_ptr_next <= w_ptr_succ;
                                empty_next <= '0';
                                if w_ptr_succ = r_ptr_reg then
                                        full_next <= '1';
                                end if;
                        end if;
                when others => -- write/read;
                        w_ptr_next <= w_ptr_succ;
                        r_ptr_next <= r_ptr_succ;
        end case;
end process;

-- output
full <= full_reg;
empty <= empty_reg;
end arch;
```

Listing VHDL program of baud generator

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Baud_Generator is
	generic (
		N : integer :=8; -- number of bits
		M : integer := 163	-- mod-M
	);
	port (
		clk, rst : in std_logic;
		max_tick : out std_logic;
		q : out std_logic_vector (N-1 downto 0)
	);
end Baud_Generator;

architecture arch of Baud_Generator is
	signal r_reg, r_next : unsigned (N-1 downto 0);

begin
-- ==================================================================
-- memory circuit
-- ==================================================================
process (clk, rst)
begin
	if rst='1' then
		r_reg <= (others => '0');
	elsif falling_edge(clk) then
		r_reg <= r_next;
	end if;
end process;

-- ==================================================================
-- next state circuit
-- ==================================================================
r_next <= (others => '0') when r_reg=(M-1) else r_reg + 1;

-- ==================================================================
-- output circuit
-- ==================================================================
q <= std_logic_vector(r_reg);
max_tick <= '1' when r_reg=(M-1) else '0';

end arch;
```

Listing VHDL program of Serial Receiver module (receiver part)

```
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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity UART_Receiver is
        generic (
                DBIT : integer:=8;                -- data bits
                SB_TICK : integer :=16    -- ticks for changing to the next bits
        );
        port (
                clk, rst : in std_logic;
                rx : in std_logic;
                s_tick : in std_logic; -- to activate the receiver module
                rx_done_tick : out std_logic;
                dout : out std_logic_vector (7 downto 0)
        );
end UART_Receiver;

architecture arch of UART_Receiver is
        type state_type is (idle, start, data, stop);
        signal state_reg, state_next : state_type;
        signal s_reg, s_next : unsigned (3 downto 0);
        signal n_reg, n_next : unsigned (2 downto 0);
        signal b_reg, b_next : std_logic_vector (7 downto 0);
begin
-- ===================================================================
-- memory circuit
-- ===================================================================
-- FSMD state & data registers
process (clk, rst)
begin
        if rst ='1' then
                state_reg <= idle;
                s_reg <= (others =>'0');
                n_reg <= (others =>'0');
                b_reg <= (others =>'0');
        elsif falling_edge(clk) then
                state_reg <= state_next;
                s_reg <= s_next;
                n_reg <= n_next;
                b_reg <= b_next;
        end if;
end process;

-- ===================================================================
-- next state circuit
-- ===================================================================
-- next state logic & data path functional units / routing
process (state_reg, s_reg, n_reg, b_reg, s_tick, rx)
begin
        state_next <= state_reg;
        s_next <= s_reg;
        n_next <= n_reg;
        b_next <= b_reg;
        rx_done_tick <='0';
        case state_reg is
                when idle =>
                        if rx='0' then
                                state_next <= start;
                                s_next <= (others => '0');
                        end if;
                when start =>
                        if s_tick = '1' then
                                if s_reg= (DBIT -1) then
                                        state_next <= data;
                                        s_next <= (others => '0');
                                        n_next <= (others => '0');
                                else
                                        s_next <= s_reg + 1;
                                end if;
                        end if;
                when data =>
                        if s_tick = '1' then
                                if s_reg = (SB_TICK -1) then
                                        s_next <= (others => '0');
                                        b_next <= rx & b_reg(7 downto 1);
                                        if n_reg = (DBIT-1) then
```

```vhdl
                        state_next <= stop;
                    else
                        n_next <= n_reg + 1;
                    end if;
                else
                    s_next <= s_reg + 1;
                end if;
            end if;
        when stop =>
            if s_tick = '1' then
                if s_reg=(SB_TICK-1) then
                    state_next <= idle;
                    rx_done_tick <= '1';
                else
                    s_next <= s_reg + 1;
                end if;
            end if;
    end case;
end process;

-- =================================================================
-- output circuit
-- =================================================================
dout <= b_reg;

end arch;
```

Listing VHDL program of Serial Receiver

```
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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity UART_System is
        generic(
                -- default setting :
                -- 74880 baud, 8 data bits, 1 stop bit, 2^2 FIFO
                DBIT : integer :=8;                -- data bits
                SB_TICK : integer :=8;    -- ticks for stop bits
                DVSR : integer := 434;    -- baud rate divisor
                                                                                        -- DVSR =
clock/(16*baud rate)
                DVSR_BIT : integer :=10;  -- bits of DVSR
                FIFO_W : integer :=4               -- address bits of FIFO
                                                                                        -- words in FIFO =
2^FIFO_W
        );
        port(
                clk, rst : in std_logic;
                sys_rd_uart : in std_logic;
                sys_r_data : out std_logic_vector (DBIT-1 downto 0);
                sys_rx_empty : out std_logic;
                rs_232_rx : in std_logic
        );
end UART_System;

architecture arch of UART_System is
        signal tick : std_logic;
        signal rx_done_tick : std_logic;
        signal rx_data_out : std_logic_vector (7 downto 0);

begin

        Baud_Generator : entity work.Baud_generator(arch)
                generic map
                ( M=>DVSR, N=>DVSR_BIT)
                port map
                ( clk=>clk, rst=>rst,
                  q=>open, max_tick => tick );
        UART_Receiver : entity work.UART_Receiver(arch)
                generic map ( DBIT => DBIT,
                                                    SB_TICK => SB_TICK)
                port map (clk=>clk, rst=>rst, rx=>rs_232_rx,
                                          s_tick=>tick, rx_done_tick=>rx_done_tick,
                                          dout=>rx_data_out);
        FIFO_Buffer_RX : entity work.FIFO_Buffer (arch)
                generic map (B=>DBIT, W=>FIFO_W)
                port map (clk=>clk, rst=>rst, rd=>sys_rd_uart,
                                          wr=>rx_done_tick, w_data=>rx_data_out,
                                          empty=>sys_rx_empty, full=>open, r_data=>sys_r_data);

end arch;
```

Listing VHDL program of Data Holder module

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity data_holder is
port (
        clk, rst : in std_logic;
        input : in std_logic_vector (7 downto 0);
        empty : in std_logic;
        rd : out std_logic;
        full : in std_logic;
        wr : out std_logic;
        dout : out std_logic_vector (15 downto 0)
        );
end data_holder;

architecture arch of data_holder is
        type state_type is (idle, one_data, wait_data, two_data);
        signal state_reg, state_next : state_type;
        signal rd_next, rd_reg : std_logic;
        signal dout_next, dout_reg : std_logic_vector (15 downto 0);
        signal wr_next, wr_reg : std_logic;
begin

-- ================================================================
-- memory circuit
-- ================================================================
-- FSMD state & data registers
process (clk, rst)
begin
        if rst ='1' then
                state_reg <= idle;
                rd_reg <= '0';
                wr_reg <= '0';
                dout_reg <= (others =>'0');

        elsif falling_edge(clk) then
                state_reg <= state_next;
                rd_reg <= rd_next;
                wr_reg <= wr_next;
                dout_reg <= dout_next;
        end if;
end process;


-- ================================================================
-- next state circuit
-- ================================================================
-- next state logic & data path functional units / routing
process (state_reg, dout_reg, rd_reg, wr_reg, input, full, empty)
begin

state_next <= state_reg;
dout_next <= dout_reg;
rd_next <= rd_reg;
wr_next <= wr_reg;

case state_reg is

when idle =>
rd_next <= '0';
wr_next <= '0';
if empty = '0' then
rd_next <= '1';
state_next <= one_data;
end if;

when one_data =>
rd_next <= '0';
wr_next <= '0';
if rd_reg = '1' then
dout_next (7 downto 0) <= input;
state_next <= wait_data;
end if;

when wait_data =>
rd_next <= '0';
```

```
wr_next <= '0';
if empty = '0' then
rd_next <= '1';
state_next <= two_data;
end if;

when two_data =>
rd_next <= '0';
wr_next <= '0';
if rd_reg = '1' then
dout_next (15 downto 8) <= input;
end if;
if full = '0' then
wr_next <= '1';
state_next <= idle;
end if;

end case;
end process;


-- =================================================================
-- output circuit
-- =================================================================
dout <= dout_reg;
wr <= wr_reg;
rd <= rd_reg;
end arch;
```

Listing VHDL program of Data Synchronizer (Data Read part)

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity data_read is
port (
                 in_data : in std_logic_vector (15 downto 0);
                 empty : in std_logic;
                 clk : in std_logic;
                 rst : in std_logic;
                 rd : out std_logic;
                 wr : out std_logic;
                 out_servo_data : out std_logic_vector (19 downto 0);
                 out_servo_id : out std_logic_vector (7 downto 0);
                 out_data : out std_logic_vector (15 downto 0)
                 );
end data_read;

architecture Behavioral of data_read is
        type state_type is (idle, read_data, write_data);
        signal state_reg, state_next : state_type;
        signal rd_reg, rd_next : std_logic;
        signal wr_reg, wr_next : std_logic;
        signal out_servo_data_reg, out_servo_data_next : std_logic_vector (19 downto 0);
        signal out_servo_id_reg, out_servo_id_next : std_logic_vector (7 downto 0);
        signal out_data_reg, out_data_next : std_logic_vector (15 downto 0);
begin
-- ====================================================================
-- memory circuit
-- ====================================================================
-- FSMD state & data registers
process (clk, rst)
begin
        if rst ='1' then
                state_reg <= idle;
                rd_reg <= '0';
                wr_reg <= '0';
                out_servo_data_reg <= (others =>'0');
                out_servo_id_reg <= (others =>'0');
                out_data_reg <= (others =>'0');

        elsif falling_edge(clk) then
                state_reg <= state_next;
                rd_reg <= rd_next;
                wr_reg <= wr_next;
                out_servo_data_reg <= out_servo_data_next;
                out_servo_id_reg <= out_servo_id_next;
                out_data_reg <= out_data_next;
        end if;
end process;

-- ====================================================================
-- next state circuit
-- ====================================================================
-- next state logic & data path functional units / routing
process (in_data, empty, rd_reg, wr_reg, out_servo_data_reg, out_servo_id_reg, out_data_reg)
begin
state_next <= state_reg;
rd_next <= rd_reg;
wr_next <= wr_reg;
out_servo_data_next <= out_servo_data_reg;
out_servo_id_next <= out_servo_id_reg;
out_data_next <= out_data_reg;

case state_reg is

when idle =>
        wr_next <= '0';
        rd_next <= '0';
        if empty = '0' then
                rd_next <= '1';
                state_next <= read_data;
        end if;

when read_data =>
        wr_next <= '0';
        rd_next <= '0';
        if rd_reg = '1' then
                wr_next <= '1';
                out_servo_data_next <= "000000000000" & in_data (7 downto 0) ;
                out_servo_id_next <= in_data (15 downto 8);
                out_data_next <= in_data;
                state_next <= write_data;
        end if;

when write_data =>
        wr_next <= '0';
        rd_next <= '0';
        if wr_reg = '1' then
                state_next <= idle;
        end if;
end case;
end process;

-- ================================================================
-- output circuit
-- ================================================================
wr <= wr_reg;
rd <= rd_reg;
out_servo_data <= out_servo_data_reg;
out_servo_id <= out_servo_id_reg;
out_data <= out_data_reg;

end Behavioral;
```

Listing VHDL program of Data Synchronizer module (Comparator part)

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity comparator is
generic
        (
        CONSTANT_ENABLER : std_logic_vector (15 downto 0) := "1111111111111111"
        );
port (
                clk : in std_logic;
                rst : in std_logic;
                in_comp : in std_logic_vector (15 downto 0);
                enabler_out : out std_logic
                );
end comparator;

architecture Behavioral of comparator is
        type state_type is (idle, enabler_active);
        signal state_reg, state_next : state_type;
        signal enabler_out_reg, enabler_out_next : std_logic;
        signal in_comp_reg, in_comp_next : std_logic_vector (15 downto 0);
begin
-- =================================================================
-- memory circuit
-- =================================================================
-- FSMD state & data registers
process (clk, rst)
begin
        if rst ='1' then
                state_reg <= idle;
                enabler_out_reg <= '0';
                in_comp_reg <= (others => '0');
        elsif falling_edge(clk) then
                state_reg <= state_next;
                enabler_out_reg <= enabler_out_next;
                in_comp_reg <= in_comp_next;
        end if;
end process;
-- =================================================================
-- next state circuit
-- =================================================================
-- next state logic & data path functional units / routing
process (in_comp, state_reg, enabler_out_reg, in_comp_reg)
begin
        state_next <= state_reg;
        enabler_out_next <= enabler_out_reg;
        in_comp_next <= in_comp;

        case state_reg is
        when idle =>
        enabler_out_next <= '0';
        if in_comp = "1111111111111111" and in_comp_reg /= in_comp then
                state_next <= enabler_active;
        end if;
        when enabler_active =>
        enabler_out_next <= '1';
        state_next <= idle;
        end case;
end process;
-- =================================================================
-- output circuit
-- =================================================================
enabler_out <= enabler_out_reg;

end Behavioral;
```

Listing VHDL program of Data Synchronizer (Latency part)

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity latency_module is
generic (
        IO_BIT : natural :=8
        );
port (
        clk, rst : in std_logic;
        trigger : in std_logic;
        input : in std_logic_vector (IO_BIT-1 downto 0);
        output : out std_logic_vector (IO_BIT-1 downto 0)
        );
end latency_module;

architecture Behavioral of latency_module is
type state_type is (idle, one_count, two_count);
        signal state_reg, state_next : state_type;
        signal output_next, output_reg : std_logic_vector (IO_BIT-1 downto 0);
        signal input_next, input_reg : std_logic_vector (IO_BIT-1 downto 0);

begin
-- ===================================================================
-- memory circuit
-- ===================================================================
-- FSMD state & data registers
process (clk, rst)
begin
        if rst ='1' then
                state_reg <= idle;
                output_reg <= (others => '0');
                input_reg <= (others => '0');
        elsif falling_edge(clk) then
                state_reg <= state_next;
                output_reg <= output_next;
                input_reg <= input_next;
        end if;
end process;

-- ===================================================================
-- next state circuit
-- ===================================================================
-- next state logic & data path functional units / routing
process (state_reg, output_reg, input_reg, input, trigger)
begin

state_next <= state_reg;
output_next <= output_reg;
input_next <= input_reg;

case state_reg is

when idle =>
if trigger = '1' then
input_next <= input;
state_next <= one_count;
end if;

when one_count =>
state_next <= two_count;

when two_count =>
output_next <= input_reg;
state_next <= idle;

end case;
end process;
-- ===================================================================
-- output circuit
-- ===================================================================
output <= output_reg;


end Behavioral;
```

Listing VHDL program of Data Synchronizer module (Demultiplexer part)

```
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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity demux is
port (
                 enabler_in : in std_logic;
                 pin_in : in std_logic;
                 select_signal : in std_logic_vector (7 downto 0);
                 update_all_signal : out std_logic;
                 pin_0 : out std_logic;
                 pin_1 : out std_logic;
                 pin_2 : out std_logic;
                 pin_3 : out std_logic;
                 pin_4 : out std_logic;
                 pin_5 : out std_logic;
                 pin_6 : out std_logic;
                 pin_7 : out std_logic;
                 pin_8 : out std_logic;
                 pin_9 : out std_logic;
                 pin_10 : out std_logic;
                 pin_11 : out std_logic;
                 pin_12 : out std_logic;
                 pin_13 : out std_logic;
                 pin_14 : out std_logic;
                 pin_15 : out std_logic;
                 pin_16 : out std_logic;
                 pin_17 : out std_logic;
                 pin_18 : out std_logic;
                 pin_19 : out std_logic;
                 pin_20 : out std_logic;
                 pin_21 : out std_logic;
                 pin_22 : out std_logic;
                 pin_23 : out std_logic;
                 pin_24 : out std_logic;
                 pin_25 : out std_logic;
                 pin_26 : out std_logic;
                 pin_27 : out std_logic;
                 pin_28 : out std_logic;
                 pin_29 : out std_logic;
                 pin_30 : out std_logic;
                 pin_31 : out std_logic
                 );

end demux;

architecture Behavioral of demux is

begin
process (enabler_in, select_signal, pin_in)
begin
update_all_signal <= '0';
pin_0 <= '0';
pin_1 <= '0';
pin_2 <= '0';
pin_3 <= '0';
pin_4 <= '0';
pin_5 <= '0';
pin_6 <= '0';
pin_7 <= '0';
pin_8 <= '0';
pin_9 <= '0';
pin_10 <= '0';
pin_11 <= '0';
pin_12 <= '0';
pin_13 <= '0';
pin_14 <= '0';
pin_15 <= '0';
pin_16 <= '0';
pin_17 <= '0';
pin_18 <= '0';
pin_19 <= '0';
pin_20 <= '0';
pin_21 <= '0';
pin_22 <= '0';
pin_23 <= '0';
pin_24 <= '0';
```

```
pin_25 <= '0';
pin_26 <= '0';
pin_27 <= '0';
pin_28 <= '0';
pin_29 <= '0';
pin_30 <= '0';
pin_31 <= '0';
case enabler_in is
        when '0' =>
                case select_signal is
                        when "00000000" =>
                        pin_0 <= pin_in;
                        when "00000001" =>
                        pin_1 <= pin_in;
                        when "00000010" =>
                        pin_2 <= pin_in;
                        when "00000011" =>
                        pin_3 <= pin_in;
                        when "00000100" =>
                        pin_4 <= pin_in;
                        when "00000101" =>
                        pin_5 <= pin_in;
                        when "00000110" =>
                        pin_6 <= pin_in;
                        when "00000111" =>
                        pin_7 <= pin_in;
                        when "00001000" =>
                        pin_8 <= pin_in;
                        when "00001001" =>
                        pin_9 <= pin_in;
                        when "00001010" =>
                        pin_10 <= pin_in;
                        when "00001011" =>
                        pin_11 <= pin_in;
                        when "00001100" =>
                        pin_12 <= pin_in;
                        when "00001101" =>
                        pin_13 <= pin_in;
                        when "00001110" =>
                        pin_14 <= pin_in;
                        when "00001111" =>
                        pin_15 <= pin_in;
                        when "00010000" =>
                        pin_16 <= pin_in;
                        when "00010001" =>
                        pin_17 <= pin_in;
                        when "00010010" =>
                        pin_18 <= pin_in;
                        when "00010011" =>
                        pin_19 <= pin_in;
                        when "00010100" =>
                        pin_20 <= pin_in;
                        when "00010101" =>
                        pin_21 <= pin_in;
                        when "00010110" =>
                        pin_22 <= pin_in;
                        when "00010111" =>
                        pin_23 <= pin_in;
                        when "00011000" =>
                        pin_24 <= pin_in;
                        when "00011001" =>
                        pin_25 <= pin_in;
                        when "00011010" =>
                        pin_26 <= pin_in;
                        when "00011011" =>
                        pin_27 <= pin_in;
                        when "00011100" =>
                        pin_28 <= pin_in;
                        when "00011101" =>
                        pin_29 <= pin_in;
                        when "00011110" =>
                        pin_30 <= pin_in;
                        when "00011111" =>
                        pin_31 <= pin_in;
                        when others =>
                end case;
        when  '1' =>
                update_all_signal <= '1';
        when others =>
end case;
end process;
end Behavioral;
```

Listing VHDL program of Data Synchronizer module

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity data_synchronizer is
        port (
                in_data_sync : in std_logic_vector (15 downto 0);
                empty : in std_logic;
                clk : in std_logic;
                rst : in std_logic;
                rd : out std_logic;
                update_all_servo : out std_logic;
      out_data : OUT  std_logic_vector(19 downto 0);
      out_wr_0 : OUT  std_logic;
      out_wr_1 : OUT  std_logic;
      out_wr_2 : OUT  std_logic;
      out_wr_3 : OUT  std_logic;
      out_wr_4 : OUT  std_logic;
      out_wr_5 : OUT  std_logic;
      out_wr_6 : OUT  std_logic;
      out_wr_7 : OUT  std_logic;
      out_wr_8 : OUT  std_logic;
      out_wr_9 : OUT  std_logic;
      out_wr_10 : OUT  std_logic;
      out_wr_11 : OUT  std_logic;
      out_wr_12 : OUT  std_logic;
      out_wr_13 : OUT  std_logic;
      out_wr_14 : OUT  std_logic;
      out_wr_15 : OUT  std_logic;
       out_wr_16 : OUT  std_logic;
      out_wr_17 : OUT  std_logic;
      out_wr_18 : OUT  std_logic;
      out_wr_19 : OUT  std_logic;
      out_wr_20 : OUT  std_logic;
      out_wr_21 : OUT  std_logic;
      out_wr_22 : OUT  std_logic;
      out_wr_23 : OUT  std_logic;
      out_wr_24 : OUT  std_logic;
      out_wr_25 : OUT  std_logic;
      out_wr_26 : OUT  std_logic;
      out_wr_27 : OUT  std_logic;
      out_wr_28 : OUT  std_logic;
      out_wr_29 : OUT  std_logic;
      out_wr_30 : OUT  std_logic;
      out_wr_31 : OUT  std_logic
                );

end data_synchronizer;

architecture Behavioral of data_synchronizer is

        COMPONENT multiplier
        PORT(
                a : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
                p : OUT STD_LOGIC_VECTOR(19 DOWNTO 0)
                );
        END COMPONENT;

        COMPONENT adder
        PORT(
                a : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
                s : OUT STD_LOGIC_VECTOR(19 DOWNTO 0);
                clk : IN STD_LOGIC
                );
        END COMPONENT;

        COMPONENT comparator
        PORT(
                clk : in std_logic;
                rst : in std_logic;
                in_comp : in std_logic_vector (15 downto 0);
                enabler_out : out std_logic
                );
        END COMPONENT;

        COMPONENT data_read
        PORT(
			clk : in std_logic;
			rst : in std_logic;
			in_data : in std_logic_vector (15 downto 0);
			empty : in std_logic;
			rd : out std_logic;
			wr : out std_logic;
			out_servo_data : out std_logic_vector (19 downto 0);
			out_servo_id : out std_logic_vector (7 downto 0);
			out_data : out std_logic_vector (15 downto 0)
			);
	END COMPONENT;

	COMPONENT demux
	PORT(
			enabler_in : in std_logic;
			pin_in : in std_logic;
			select_signal : in std_logic_vector (7 downto 0);
			update_all_signal : out std_logic;
			pin_0 : out std_logic;
			pin_1 : out std_logic;
			pin_2 : out std_logic;
			pin_3 : out std_logic;
			pin_4 : out std_logic;
			pin_5 : out std_logic;
			pin_6 : out std_logic;
			pin_7 : out std_logic;
			pin_8 : out std_logic;
			pin_9 : out std_logic;
			pin_10 : out std_logic;
			pin_11 : out std_logic;
			pin_12 : out std_logic;
			pin_13 : out std_logic;
			pin_14 : out std_logic;
			pin_15 : out std_logic;
			pin_16 : out std_logic;
			pin_17 : out std_logic;
			pin_18 : out std_logic;
			pin_19 : out std_logic;
			pin_20 : out std_logic;
			pin_21 : out std_logic;
			pin_22 : out std_logic;
			pin_23 : out std_logic;
			pin_24 : out std_logic;
			pin_25 : out std_logic;
			pin_26 : out std_logic;
			pin_27 : out std_logic;
			pin_28 : out std_logic;
			pin_29 : out std_logic;
			pin_30 : out std_logic;
			pin_31 : out std_logic
			);
	END COMPONENT;

signal enabler_wire : std_logic;
signal out_pin_0, out_pin_1, out_pin_2, out_pin_3, out_pin_4, out_pin_5, out_pin_6, out_pin_7, out_pin_8 :
std_logic;
signal out_pin_9, out_pin_10, out_pin_11, out_pin_12, out_pin_13, out_pin_14, out_pin_15, out_pin_16 :
std_logic;
signal out_pin_17, out_pin_18, out_pin_19, out_pin_20, out_pin_21, out_pin_22, out_pin_23, out_pin_24 :
std_logic;
signal out_pin_25, out_pin_26, out_pin_27, out_pin_28, out_pin_29, out_pin_30, out_pin_31 : std_logic;
signal in_comp_wire, out_data_wire, in_wire : std_logic_vector (15 downto 0);
signal out_multiplier, in_multiplier, out_adder : std_logic_vector (19 downto 0);
signal servo_id_wire, servo_data_wire, servo_id_lat_wire : std_logic_vector (7 downto 0);
signal fifo_wr, out_wr : std_logic;
begin

	in_wire <=  in_data_sync ;
	out_data <= out_adder ;
	out_wr_0 <= out_pin_0 ;
	out_wr_1 <= out_pin_1 ;
	out_wr_2 <= out_pin_2 ;
	out_wr_3 <= out_pin_3 ;
	out_wr_4 <= out_pin_4 ;
	out_wr_5 <= out_pin_5 ;
	out_wr_6 <= out_pin_6 ;
	out_wr_7 <= out_pin_7 ;
	out_wr_8 <= out_pin_8 ;
	out_wr_9 <= out_pin_9 ;
	out_wr_10 <= out_pin_10 ;
	out_wr_11 <= out_pin_11 ;
	out_wr_12 <= out_pin_12 ;
	out_wr_13 <= out_pin_13 ;
	out_wr_14 <= out_pin_14 ;
	out_wr_15 <= out_pin_15 ;
	out_wr_16 <= out_pin_16 ;
	out_wr_17 <= out_pin_17 ;
	out_wr_18 <= out_pin_18 ;
```

```vhdl
out_wr_19 <= out_pin_19 ;
out_wr_20 <= out_pin_20 ;
out_wr_21 <= out_pin_21 ;
out_wr_22 <= out_pin_22 ;
out_wr_23 <= out_pin_23 ;
out_wr_24 <= out_pin_24 ;
out_wr_25 <= out_pin_25 ;
out_wr_26 <= out_pin_26 ;
out_wr_27 <= out_pin_27 ;
out_wr_28 <= out_pin_28 ;
out_wr_29 <= out_pin_29 ;
out_wr_30 <= out_pin_30 ;
out_wr_31 <= out_pin_31 ;

Inst_data_read: data_read PORT MAP(
clk => clk,
rst => rst,
in_data => in_wire,
empty => empty,
out_data => in_comp_wire,
out_servo_data => in_multiplier,
out_servo_id => servo_id_wire,
rd => rd,
wr => fifo_wr
);

latency_module_wr: entity work.latency_module_one_bit(Behavioral)
        port map (
                clk => clk,
                rst => rst,
                trigger => fifo_wr,
                input => fifo_wr,
                output => out_wr);

latency_module_data_ID: entity work.latency_module(Behavioral)
        generic map (
                IO_BIT => 8)
        port map (
                clk => clk,
                rst => rst,
                trigger => fifo_wr,
                input => servo_id_wire,
                output => servo_id_lat_wire);

Inst_comparator: comparator PORT MAP(
clk => clk,
rst => rst,
in_comp => in_comp_wire,
enabler_out => enabler_wire
);

inst_multiplier: multiplier PORT MAP(
a => in_multiplier,
p => out_multiplier
);

inst_adder: adderPORT MAP(
a => out_multiplier,
s => out_adder,
clk => clk
);

Inst_demux: demux PORT MAP(
pin_in => out_wr,
select_signal => servo_id_lat_wire,
enabler_in => enabler_wire,
update_all_signal => update_all_servo,
pin_0 => out_pin_0,
pin_1 => out_pin_1,
pin_2 => out_pin_2,
pin_3 => out_pin_3,
pin_4 => out_pin_4,
pin_5 => out_pin_5,
pin_6 => out_pin_6,
pin_7 => out_pin_7,
pin_8 => out_pin_8,
pin_9 => out_pin_9,
pin_10 => out_pin_10,
pin_11 => out_pin_11,
pin_12 => out_pin_12,
pin_13 => out_pin_13,
pin_14 => out_pin_14,
pin_15 => out_pin_15,
pin_16 => out_pin_16,
pin_17 => out_pin_17,
pin_18 => out_pin_18,
pin_19 => out_pin_19,
pin_20 => out_pin_20,
        pin_21 => out_pin_21,
        pin_22 => out_pin_22,
        pin_23 => out_pin_23,
        pin_24 => out_pin_24,
        pin_25 => out_pin_25,
        pin_26 => out_pin_26,
        pin_27 => out_pin_27,
        pin_28 => out_pin_28,
        pin_29 => out_pin_29,
        pin_30 => out_pin_30,
        pin_31 => out_pin_31
        );

end Behavioral;
```

Listing VHDL program of PWM Generator module

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PWM_gen is

    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           rise_time : in  STD_LOGIC_VECTOR (19 downto 0);
                            period : std_logic_vector (21 downto 0);
           out_rise : out  STD_LOGIC);
end PWM_gen;

architecture Behavioral of PWM_gen is
signal cnt_p , cnt_n : unsigned(21 downto 0);
signal out_rise_p, out_rise_n : std_logic;

begin

-- ==================================================================
-- memory circuit
-- ==================================================================
process(clk, rst) is
begin
        if rst='1' then
                cnt_p <= (others => '0');
                out_rise_p <= '0';
        elsif falling_edge(clk) then -- update data
                cnt_p <= cnt_n;
                out_rise_p <= out_rise_n;
        end if;
end process;

-- ==================================================================
-- next state circuit
-- ==================================================================
NEXT_STATE_CIRCUIT: -- berhubungan dengan input dengan present state input
process(out_rise_p, cnt_p, rise_time, period) is
begin
        -- default :
        cnt_n <= cnt_p + 1; -- mencacah
        out_rise_n <= out_rise_p; -- menyimpan
        -- main process :
        if cnt_p = unsigned(rise_time) - 1 then
                out_rise_n <= '0';
        end if;
        if cnt_p = unsigned(period) - 1 then
                cnt_n <= (others=>'0');
                out_rise_n <= '1';
        end if;
end process;

-- ==================================================================
-- output circuit
-- ==================================================================
process(out_rise_p) is
begin
        out_rise <= out_rise_p; -- output modul dihubungkan dengan buffer
end process;

end Behavioral;
```

Listing VHDL program of PWM Module

```
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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PWM_Module is

        generic (
                FIFO_w : integer :=1;
                DBIT : integer :=20;
                period_constant : std_logic_vector (21 downto 0) := "1111010000100100000000"
                );

        port    (
                clk : in std_logic;
                rst : in std_logic;
                rd, wr : in std_logic;
                data_sudut : in std_logic_vector (19 downto 0);
                PWM_signal : out  STD_LOGIC
                );
end PWM_Module;

architecture Behavioral of PWM_Module is

        COMPONENT PWM_gen
        PORT(
                        clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        rise_time : in  STD_LOGIC_VECTOR (19 downto 0);
                        period : std_logic_vector (21 downto 0);
        out_rise : out  STD_LOGIC
                );
        END COMPONENT;

signal data_wire : std_logic_vector (19 downto 0);
begin

        FIFO_Buffer_20b : entity work.FIFO_Buffer (arch)
        generic map (B=>DBIT, W=>FIFO_W)
        port map (      clk=>clk, rst=>rst, rd=>rd,
                                        wr=>wr, w_data=>data_sudut,
                                        empty=>open, full=>open, r_data=>data_wire);

        Inst_PWM_gen: PWM_gen PORT MAP(
        clk => clk,
        rst => rst,
        period => period_constant,
        rise_time => data_wire,
        out_rise => PWM_signal
        );

end Behavioral;
```

Listing VHDL program of PWM *Hardware system*

```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 instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Hard_PWM_UART_Comm is
port (
                 clk : in std_logic;
                 rst : in std_logic;
                 rs_232_rx : in std_logic;
                 out_pwm_0 : OUT std_logic;
                 out_pwm_1 : OUT std_logic;
                 out_pwm_2 : OUT std_logic;
                 out_pwm_3 : OUT std_logic;
                 out_pwm_4 : OUT std_logic;
                 out_pwm_5 : OUT std_logic;
                 out_pwm_6 : OUT std_logic;
                 out_pwm_7 : OUT std_logic;
                 out_pwm_8 : OUT std_logic;
                 out_pwm_9 : OUT std_logic;
                 out_pwm_10 : OUT std_logic;
                 out_pwm_11 : OUT std_logic;
                 out_pwm_12 : OUT std_logic;
                 out_pwm_13 : OUT std_logic;
                 out_pwm_14 : OUT std_logic;
                 out_pwm_15 : OUT std_logic;
                 out_pwm_16 : OUT std_logic;
                 out_pwm_17 : OUT std_logic;
                 out_pwm_18 : OUT std_logic;
                 out_pwm_19 : OUT std_logic;
                 out_pwm_20 : OUT std_logic;
                 out_pwm_21 : OUT std_logic;
                 out_pwm_22 : OUT std_logic;
                 out_pwm_23 : OUT std_logic;
                 out_pwm_24 : OUT std_logic;
                 out_pwm_25 : OUT std_logic;
                 out_pwm_26 : OUT std_logic;
                 out_pwm_27 : OUT std_logic;
                 out_pwm_28 : OUT std_logic;
                 out_pwm_29 : OUT std_logic;
                 out_pwm_30 : OUT std_logic;
                 out_pwm_31 : OUT std_logic
                 );
end Hard_PWM_UART_Comm;

architecture Behavioral of Hard_PWM_UART_Comm is

        component clk_200
        port
                 (-- Clock in ports
                 CLK_IN1           : in     std_logic;
                 -- Clock out ports
                 CLK_OUT1          : out    std_logic;
                 -- Status and control signals
                 RESET             : in     std_logic
                 );
        end component;

        COMPONENT PWM_Module
        PORT(
                 clk : IN std_logic;
                 rst : IN std_logic;
                 rd : IN std_logic;
                 wr : IN std_logic;
                 data_sudut : IN std_logic_vector(19 downto 0);
                 PWM_signal : OUT std_logic
                 );
        END COMPONENT;

        COMPONENT UART_System
        PORT(
                 clk : IN std_logic;
                 rst : IN std_logic;
                 sys_rd_uart : IN std_logic;
                 rs_232_rx : IN std_logic;
                 sys_r_data : OUT std_logic_vector(7 downto 0);
                 sys_rx_empty : OUT std_logic
                 );
```

```vhdl
    END COMPONENT;

    COMPONENT data_holder
    PORT(
            clk : IN std_logic;
            rst : IN std_logic;
            input : IN std_logic_vector(7 downto 0);
            empty : IN std_logic;
            full : IN std_logic;
            rd : OUT std_logic;
            wr : OUT std_logic;
            dout : OUT std_logic_vector(15 downto 0)
            );
    END COMPONENT;

    COMPONENT data_synchronizer
    PORT(
            in_data_sync : in std_logic_vector (15 downto 0);
            empty : in std_logic;
            clk : in std_logic;
            rst : in std_logic;
            rd : out std_logic;
            update_all_servo : out std_logic;
out_data : OUT  std_logic_vector(19 downto 0);
out_wr_0 : OUT  std_logic;
out_wr_1 : OUT  std_logic;
out_wr_2 : OUT  std_logic;
out_wr_3 : OUT  std_logic;
out_wr_4 : OUT  std_logic;
out_wr_5 : OUT  std_logic;
out_wr_6 : OUT  std_logic;
out_wr_7 : OUT  std_logic;
out_wr_8 : OUT  std_logic;
out_wr_9 : OUT  std_logic;
out_wr_10 : OUT  std_logic;
out_wr_11 : OUT  std_logic;
out_wr_12 : OUT  std_logic;
out_wr_13 : OUT  std_logic;
out_wr_14 : OUT  std_logic;
out_wr_15 : OUT  std_logic;
 out_wr_16 : OUT  std_logic;
out_wr_17 : OUT  std_logic;
out_wr_18 : OUT  std_logic;
out_wr_19 : OUT  std_logic;
out_wr_20 : OUT  std_logic;
out_wr_21 : OUT  std_logic;
out_wr_22 : OUT  std_logic;
out_wr_23 : OUT  std_logic;
out_wr_24 : OUT  std_logic;
out_wr_25 : OUT  std_logic;
out_wr_26 : OUT  std_logic;
out_wr_27 : OUT  std_logic;
out_wr_28 : OUT  std_logic;
out_wr_29 : OUT  std_logic;
out_wr_30 : OUT  std_logic;
out_wr_31 : OUT  std_logic
            );
    END COMPONENT;

    signal clk_200MHz : std_logic;
    signal update_all_servo_wire : std_logic;
    signal out_wr_0_wire : std_logic;
    signal out_wr_1_wire : std_logic;
    signal out_wr_2_wire : std_logic;
    signal out_wr_3_wire : std_logic;
    signal out_wr_4_wire : std_logic;
    signal out_wr_5_wire : std_logic;
    signal out_wr_6_wire : std_logic;
    signal out_wr_7_wire : std_logic;
    signal out_wr_8_wire : std_logic;
    signal out_wr_9_wire : std_logic;
    signal out_wr_10_wire : std_logic;
    signal out_wr_11_wire : std_logic;
    signal out_wr_12_wire : std_logic;
    signal out_wr_13_wire : std_logic;
    signal out_wr_14_wire : std_logic;
    signal out_wr_15_wire : std_logic;
    signal out_wr_16_wire : std_logic;
    signal out_wr_17_wire : std_logic;
    signal out_wr_18_wire : std_logic;
    signal out_wr_19_wire : std_logic;
    signal out_wr_20_wire : std_logic;
    signal out_wr_21_wire : std_logic;
    signal out_wr_22_wire : std_logic;
    signal out_wr_23_wire : std_logic;
    signal out_wr_24_wire : std_logic;
    signal out_wr_25_wire : std_logic;
    signal out_wr_26_wire : std_logic;
```

```vhdl
        signal out_wr_27_wire : std_logic;
        signal out_wr_28_wire : std_logic;
        signal out_wr_29_wire : std_logic;
        signal out_wr_30_wire : std_logic;
        signal out_wr_31_wire : std_logic;
        signal out_data_wire : std_logic_vector (19 downto 0);
        signal out_pwm_0_wire : std_logic;
        signal out_pwm_1_wire : std_logic;
        signal out_pwm_2_wire : std_logic;
        signal out_pwm_3_wire : std_logic;
        signal out_pwm_4_wire : std_logic;
        signal out_pwm_5_wire : std_logic;
        signal out_pwm_6_wire : std_logic;
        signal out_pwm_7_wire : std_logic;
        signal out_pwm_8_wire : std_logic;
        signal out_pwm_9_wire : std_logic;
        signal out_pwm_10_wire : std_logic;
        signal out_pwm_11_wire : std_logic;
        signal out_pwm_12_wire : std_logic;
        signal out_pwm_13_wire : std_logic;
        signal out_pwm_14_wire : std_logic;
        signal out_pwm_15_wire : std_logic;
        signal out_pwm_16_wire : std_logic;
        signal out_pwm_17_wire : std_logic;
        signal out_pwm_18_wire : std_logic;
        signal out_pwm_19_wire : std_logic;
        signal out_pwm_20_wire : std_logic;
        signal out_pwm_21_wire : std_logic;
        signal out_pwm_22_wire : std_logic;
        signal out_pwm_23_wire : std_logic;
        signal out_pwm_24_wire : std_logic;
        signal out_pwm_25_wire : std_logic;
        signal out_pwm_26_wire : std_logic;
        signal out_pwm_27_wire : std_logic;
        signal out_pwm_28_wire : std_logic;
        signal out_pwm_29_wire : std_logic;
        signal out_pwm_30_wire : std_logic;
        signal out_pwm_31_wire : std_logic;
        signal sys_rd_uart_rd_holder_wire : std_logic;
        signal sys_r_data_input_holder_wire : std_logic_vector(7 downto 0);
        signal sys_rx_empty_empty_holder_wire : std_logic;
        signal wr_holder_wr_fifo_wire : std_logic;
        signal full_holder_full_fifo_wire : std_logic;
        signal dout_holder_w_data_fifo_wire : std_logic_vector(15 downto 0);
        signal empty_fifo_empty_sync_wire : std_logic;
        signal r_data_fifo_in_data_sync_wire : std_logic_vector (15 downto 0);
        signal rd_sync_rd_fifo_wire : std_logic;

begin
        out_pwm_0 <= out_pwm_0_wire;
        out_pwm_1 <= out_pwm_1_wire;
        out_pwm_2 <= out_pwm_2_wire;
        out_pwm_3 <= out_pwm_3_wire;
        out_pwm_4 <= out_pwm_4_wire;
        out_pwm_5 <= out_pwm_5_wire;
        out_pwm_6 <= out_pwm_6_wire;
        out_pwm_7 <= out_pwm_7_wire;
        out_pwm_8 <= out_pwm_8_wire;
        out_pwm_9 <= out_pwm_9_wire;
        out_pwm_10 <= out_pwm_10_wire;
        out_pwm_11 <= out_pwm_11_wire;
        out_pwm_12 <= out_pwm_12_wire;
        out_pwm_13 <= out_pwm_13_wire;
        out_pwm_14 <= out_pwm_14_wire;
        out_pwm_15 <= out_pwm_15_wire;
        out_pwm_16 <= out_pwm_16_wire;
        out_pwm_17 <= out_pwm_17_wire;
        out_pwm_18 <= out_pwm_18_wire;
        out_pwm_19 <= out_pwm_19_wire;
        out_pwm_20 <= out_pwm_20_wire;
        out_pwm_21 <= out_pwm_21_wire;
        out_pwm_22 <= out_pwm_22_wire;
        out_pwm_23 <= out_pwm_23_wire;
        out_pwm_24 <= out_pwm_24_wire;
        out_pwm_25 <= out_pwm_25_wire;
        out_pwm_26 <= out_pwm_26_wire;
        out_pwm_27 <= out_pwm_27_wire;
        out_pwm_28 <= out_pwm_28_wire;
        out_pwm_29 <= out_pwm_29_wire;
        out_pwm_30 <= out_pwm_30_wire;
        out_pwm_31 <= out_pwm_31_wire;

Main_Clock : clk_200
   port map
   (-- Clock in ports
    CLK_IN1 => clk,
    -- Clock out ports
    CLK_OUT1 => clk_200MHz,
-- Status and control signals
RESET   => rst);

     Inst_PWM_Module_0: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_0_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_0_wire
     );

     Inst_PWM_Module_1: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_1_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_1_wire
     );

     Inst_PWM_Module_2: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_2_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_2_wire
     );

     Inst_PWM_Module_3: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_3_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_3_wire
     );

     Inst_PWM_Module_4: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_4_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_4_wire
     );

     Inst_PWM_Module_5: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_5_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_5_wire
     );

     Inst_PWM_Module_6: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_6_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_6_wire
     );

     Inst_PWM_Module_7: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_7_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_7_wire
     );

     Inst_PWM_Module_8: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
          wr => out_wr_8_wire,
          data_sudut => out_data_wire,
          PWM_signal => out_pwm_8_wire
     );

     Inst_PWM_Module_9: PWM_Module PORT MAP(
          clk => clk_200MHz,
          rst => rst,
          rd => update_all_servo_wire,
        wr => out_wr_9_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_9_wire
);

Inst_PWM_Module_10: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_10_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_10_wire
);

Inst_PWM_Module_11: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_11_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_11_wire
);

Inst_PWM_Module_12: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_12_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_12_wire
);

Inst_PWM_Module_13: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_13_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_13_wire
);

Inst_PWM_Module_14: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_14_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_14_wire
);

Inst_PWM_Module_15: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_15_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_15_wire
);

Inst_PWM_Module_16: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_16_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_16_wire
);

Inst_PWM_Module_17: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_17_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_17_wire
);

Inst_PWM_Module_18: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_18_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_18_wire
);

Inst_PWM_Module_19: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_19_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_19_wire
);

Inst_PWM_Module_20: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_20_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_20_wire
);

Inst_PWM_Module_21: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_21_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_21_wire
);

Inst_PWM_Module_22: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_22_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_22_wire
);

Inst_PWM_Module_23: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_23_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_23_wire
);

Inst_PWM_Module_24: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_24_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_24_wire
);

Inst_PWM_Module_25: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_25_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_25_wire
);

Inst_PWM_Module_26: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_26_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_26_wire
);

Inst_PWM_Module_27: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_27_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_27_wire
);

Inst_PWM_Module_28: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_28_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_28_wire
);
```

```vhdl
Inst_PWM_Module_29: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_29_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_29_wire
);

Inst_PWM_Module_30: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_30_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_30_wire
);

Inst_PWM_Module_31: PWM_Module PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        rd => update_all_servo_wire,
        wr => out_wr_31_wire,
        data_sudut => out_data_wire,
        PWM_signal => out_pwm_31_wire
);

Inst_UART_System: UART_System PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        sys_rd_uart => sys_rd_uart_rd_holder_wire,
        sys_r_data => sys_r_data_input_holder_wire,
        sys_rx_empty => sys_rx_empty_empty_holder_wire,
        rs_232_rx => rs_232_rx
);

Inst_data_holder: data_holder PORT MAP(
        clk => clk_200MHz,
        rst => rst,
        input => sys_r_data_input_holder_wire,
        empty => sys_rx_empty_empty_holder_wire,
        rd => sys_rd_uart_rd_holder_wire,
        full => full_holder_full_fifo_wire,
        wr => wr_holder_wr_fifo_wire,
        dout => dout_holder_w_data_fifo_wire
);

FIFO_Buffer_16b : entity work.FIFO_Buffer (arch)
generic map (B=>16, W=>4)
port map (
        clk=>clk_200MHz,
        rst=>rst,
        rd=>rd_sync_rd_fifo_wire,
        wr=>wr_holder_wr_fifo_wire,
        w_data=>dout_holder_w_data_fifo_wire,
        empty=>empty_fifo_empty_sync_wire,
        full=>full_holder_full_fifo_wire,
        r_data=>r_data_fifo_in_data_sync_wire
);

Inst_data_synchronizer: data_synchronizer PORT MAP(
        in_data_sync => r_data_fifo_in_data_sync_wire,
        empty => empty_fifo_empty_sync_wire,
        clk => clk_200MHz,
        rst => rst,
        rd => rd_sync_rd_fifo_wire,
        update_all_servo => update_all_servo_wire,
        out_data => out_data_wire,
        out_wr_0 => out_wr_0_wire,
        out_wr_1 => out_wr_1_wire,
        out_wr_2 => out_wr_2_wire,
        out_wr_3 => out_wr_3_wire,
        out_wr_4 => out_wr_4_wire,
        out_wr_5 => out_wr_5_wire,
        out_wr_6 => out_wr_6_wire,
        out_wr_7 => out_wr_7_wire,
        out_wr_8 => out_wr_8_wire,
        out_wr_9 => out_wr_9_wire,
        out_wr_10 => out_wr_10_wire,
        out_wr_11 => out_wr_11_wire,
        out_wr_12 => out_wr_12_wire,
        out_wr_13 => out_wr_13_wire,
        out_wr_14 => out_wr_14_wire,
        out_wr_15 => out_wr_15_wire,
        out_wr_16 => out_wr_16_wire,
        out_wr_17 => out_wr_17_wire,
        out_wr_18 => out_wr_18_wire,
        out_wr_19 => out_wr_19_wire,
                        out_wr_20 => out_wr_20_wire,
                        out_wr_21 => out_wr_21_wire,
                        out_wr_22 => out_wr_22_wire,
                        out_wr_23 => out_wr_23_wire,
                        out_wr_24 => out_wr_24_wire,
                        out_wr_25 => out_wr_25_wire,
                        out_wr_26 => out_wr_26_wire,
                        out_wr_27 => out_wr_27_wire,
                        out_wr_28 => out_wr_28_wire,
                        out_wr_29 => out_wr_29_wire,
                        out_wr_30 => out_wr_30_wire,
                        out_wr_31 => out_wr_31_wire
            );

end Behavioral;
```

Listing VHDL program for simulation of Serial Receiver module

```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;

ENTITY Test_UART_System IS
END Test_UART_System;

ARCHITECTURE behavior OF Test_UART_System IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT UART_System
    PORT(
         clk : IN  std_logic;
         rst : IN  std_logic;
         sys_rd_uart : IN  std_logic;
         sys_r_data : OUT  std_logic_vector(7 downto 0);
         sys_rx_empty : OUT  std_logic;
         rs_232_rx : IN  std_logic
        );
    END COMPONENT;

   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';
   signal sys_rd_uart : std_logic := '0';
   signal rs_232_rx : std_logic := '1';

 --Outputs
   signal sys_r_data : std_logic_vector(7 downto 0);
   signal sys_rx_empty : std_logic;

   -- Clock period definitions
   constant clk_period : time := 5 ns;
   constant main_clk : time := 5 ns;
        constant DVSR : integer := 651;

BEGIN
        -- Instantiate the Unit Under Test (UUT)
   uut: UART_System PORT MAP (
          clk => clk,
          rst => rst,
          sys_rd_uart => sys_rd_uart,
          sys_r_data => sys_r_data,
          sys_rx_empty => sys_rx_empty,
          rs_232_rx => rs_232_rx
        );

   -- Clock process definitions
   clk_process :process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      rst <= '1';
      wait for 100 ns;
                rst <= '0';
      wait for clk_period*10;

                wait for 50 us;

                -- send first data
                -- start bit
--              marker <= '1';
                rs_232_rx <= '0';
                wait for DVSR*8*main_clk;
                -- bit 0
                rs_232_rx <= '0';
                wait for DVSR*8*main_clk;
                -- bit 1
                rs_232_rx <= '0';
                wait for DVSR*8*main_clk;
                -- bit 2
                rs_232_rx <= '0';
                wait for DVSR*8*main_clk;
```

```
                 -- bit 3
                 rs_232_rx <= '1';
                 wait for DVSR*8*main_clk;
                 -- bit 4
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 5
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 6
                 rs_232_rx <= '1';
                 wait for DVSR*8*main_clk;
                 -- bit 7
                 rs_232_rx <= '1';
                 wait for DVSR*8*main_clk;
                 -- stop bit
                 rs_232_rx <= '1';
                 wait for DVSR*8*main_clk;
--               marker <= '0';

                 -- send second data
                 -- start bit
--               marker <= '1';
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 0
                 rs_232_rx <= '1';
                 wait for DVSR*8*main_clk;
                 -- bit 1
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 2
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 3
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 4
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 5
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 6
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- bit 7
                 rs_232_rx <= '0';
                 wait for DVSR*8*main_clk;
                 -- stop bit
                 rs_232_rx <= '1';
                 wait for DVSR*8*main_clk;
--               marker <= '0';

                 wait for 50 us;
                 sys_rd_uart <= '1';
                 wait for clk_period;
                 sys_rd_uart <= '0';

      wait;
   end process;

END;
```

Listing VHDL program for simulation of Data Holder module

```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;
ENTITY test_data_holder IS
END test_data_holder;

ARCHITECTURE behavior OF test_data_holder IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT data_holder
    PORT(
         clk : IN  std_logic;
         rst : IN  std_logic;
         input : IN  std_logic_vector(7 downto 0);
         empty : IN  std_logic;
         rd : OUT  std_logic;
         full : IN  std_logic;
         wr : OUT  std_logic;
         dout : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';
   signal input : std_logic_vector(7 downto 0) := (others => '0');
   signal empty : std_logic := '1';
   signal full : std_logic := '0';
        --Outputs
   signal rd : std_logic;
   signal wr : std_logic;
   signal dout : std_logic_vector(15 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN
        -- Instantiate the Unit Under Test (UUT)
   uut: data_holder PORT MAP (
          clk => clk,
          rst => rst,
          input => input,
          empty => empty,
          rd => rd,
          full => full,
          wr => wr,
          dout => dout
        );

   -- Clock process definitions
   clk_process :process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
                rst <= '1';
      wait for 100 ns;
                rst <= '0';
      wait for clk_period*10;
      -- insert stimulus here
                input <= "11001000";
                empty <= '0';
                wait for clk_period*2;
                input <= "00000000";
                empty <= '1';
                wait for clk_period*2;
                input <= "00000001";
                empty <= '0';
                wait for clk_period*2;
                input <= "00000000";
                empty <= '1';
                wait;
   end process;
END;
```

Listing VHDL program for simulation of Data Synchronizer module

```
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;

ENTITY test_data_synchronizer IS
END test_data_synchronizer;

ARCHITECTURE behavior OF test_data_synchronizer IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT data_synchronizer
    PORT(
         in_data_sync : IN  std_logic_vector(15 downto 0);
         empty : IN  std_logic;
         clk : IN  std_logic;
                          rst : IN  std_logic;
         rd : OUT  std_logic;
         update_all_servo : OUT  std_logic;
         out_data : OUT  std_logic_vector(19 downto 0);
         out_wr_0 : OUT  std_logic;
         out_wr_1 : OUT  std_logic;
         out_wr_2 : OUT  std_logic;
         out_wr_3 : OUT  std_logic;
         out_wr_4 : OUT  std_logic;
         out_wr_5 : OUT  std_logic;
         out_wr_6 : OUT  std_logic;
         out_wr_7 : OUT  std_logic;
         out_wr_8 : OUT  std_logic;
         out_wr_9 : OUT  std_logic;
         out_wr_10 : OUT  std_logic;
         out_wr_11 : OUT  std_logic;
         out_wr_12 : OUT  std_logic;
         out_wr_13 : OUT  std_logic;
         out_wr_14 : OUT  std_logic;
         out_wr_15 : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal in_data_sync : std_logic_vector(15 downto 0) := (others => '0');
   signal empty : std_logic := '1';
   signal clk : std_logic := '0';
        signal rst : std_logic := '0';

        --Outputs
   signal rd : std_logic;
   signal update_all_servo : std_logic;
   signal out_data : std_logic_vector(19 downto 0);
   signal out_wr_0 : std_logic;
   signal out_wr_1 : std_logic;
   signal out_wr_2 : std_logic;
   signal out_wr_3 : std_logic;
   signal out_wr_4 : std_logic;
   signal out_wr_5 : std_logic;
   signal out_wr_6 : std_logic;
   signal out_wr_7 : std_logic;
   signal out_wr_8 : std_logic;
   signal out_wr_9 : std_logic;
   signal out_wr_10 : std_logic;
   signal out_wr_11 : std_logic;
   signal out_wr_12 : std_logic;
   signal out_wr_13 : std_logic;
   signal out_wr_14 : std_logic;
   signal out_wr_15 : std_logic;

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: data_synchronizer PORT MAP (
          in_data_sync => in_data_sync,
          empty => empty,
          clk => clk,
                          rst => rst,
          rd => rd,
          update_all_servo => update_all_servo,
          out_data => out_data,
          out_wr_0 => out_wr_0,
          out_wr_1 => out_wr_1,
          out_wr_2 => out_wr_2,
          out_wr_3 => out_wr_3,
          out_wr_4 => out_wr_4,
          out_wr_5 => out_wr_5,
          out_wr_6 => out_wr_6,
          out_wr_7 => out_wr_7,
          out_wr_8 => out_wr_8,
          out_wr_9 => out_wr_9,
          out_wr_10 => out_wr_10,
          out_wr_11 => out_wr_11,
          out_wr_12 => out_wr_12,
          out_wr_13 => out_wr_13,
          out_wr_14 => out_wr_14,
          out_wr_15 => out_wr_15
        );

   -- Clock process definitions
   clk_process :process
   begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
              rst <= '1';
      wait for 100 ns;
              rst <= '0';
      wait for clk_period*10;

              in_data_sync <= "0000000111001000";
              empty <= '0';
              wait for clk_period*2;
              in_data_sync <= "0000000000000000";
              empty <= '1';

              wait for clk_period*2;
              in_data_sync <= "0000001011111010";
              empty <= '0';
              wait for clk_period*2;
              in_data_sync <= "0000000000000000";
              empty <= '1';

              wait for clk_period*2;
              in_data_sync <= "0000010001111101";
              empty <= '0';
              wait for clk_period*2;
              in_data_sync <= "0000000000000000";
              empty <= '1';

      -- insert stimulus here

      wait;
   end process;

END;
```

Listing VHDL program for simulation of PWM Generator

```
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;

ENTITY Test_PWM_Module IS
END Test_PWM_Module;

ARCHITECTURE behavior OF Test_PWM_Module IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT PWM_Module
    PORT(
         clk : IN  std_logic;
         rst : IN  std_logic;
         rd : IN  std_logic;
         wr : IN  std_logic;
         data_sudut : IN  std_logic_vector(19 downto 0);
         PWM_signal : OUT  std_logic;
                         debug : OUT std_logic_vector(21 downto 0)
        );
    END COMPONENT;

   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';
   signal rd : std_logic := '0';
   signal wr : std_logic := '0';
   signal data_sudut : std_logic_vector(19 downto 0) := (others => '0');
        signal debug : std_logic_vector(21 downto 0)  := (others => '0');
        --Outputs
   signal PWM_signal : std_logic;

   -- Clock period definitions
   constant clk_period : time := 5 ns;

BEGIN
        -- Instantiate the Unit Under Test (UUT)
   uut: PWM_Module PORT MAP (
          clk => clk,
          rst => rst,
          rd => rd,
          wr => wr,
          data_sudut => data_sudut,
          PWM_signal => PWM_signal,
                          debug => debug
        );
   -- Clock process definitions
   clk_process :process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;
   -- Stimulus process
   stim_proc: process
   begin
    -- hold reset state for 100 ns.
      rst <= '1';
      wait for 100 ns;
                rst <= '0';
      wait for clk_period*10;
                 -- insert stimulus here
                wr <= '1';
                rd <= '1';
                data_sudut <= "00110000110101000000";
      wait;
   end process;
END;
```