TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

THIRD YEAR EMBEDDED SYSTEM PROJECT REPORT

ON

CYCLIC REDUNDANCY CHECK (CRC) IN VHDL

SUBMITTED TO:

Er. SHIVA RAJ LUITEL

SUBMITTED BY:

SHIVA SHRESTHA(073/BCT/41)

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

JULY,2019

TITLE: CYCLIC REDUNDANCY CHECK (CRC) IN VHDL

OBJECTIVES:

- To generate the circuit diagram (RTL) for any CRC polynomial using VHDL.
- To simulate error detecting code using VHDL.

THEORY:

Cyclic Redundancy Check is a method adopted in the field of communication to detect errors during transmission through the communication channel. The data transmitted can be of any size depending on the type of data being transmitted. In this paper, we have designed a VHDL code which demonstrates how the CRC process works on a code word whose length can be changed by the user based on his requirements and the necessary simulations can be carried out to verify the results.

Cyclic Redundancy Check (CRC) is an error detecting code in which a transmitted message is appended with a few redundant bits from the transmitter and then the code word is checked at the receiver using modulo-2 arithmetic for errors. The message is then transmitted from the encoder and is received by the receiver where a CRC check is carried out. This process helps to determine any errors in transmission through the transmission channel. This entire process is demonstrated using Very high speed Integrated Circuit Hardware Description Language (VHDL). VHDL is a hardware description language used in electronic design automation to implement designs in systems such as field-programmable gate arrays. All the statements are executed concurrently in VHDL.

PROCESS OF CRC IMPLEMENTATION

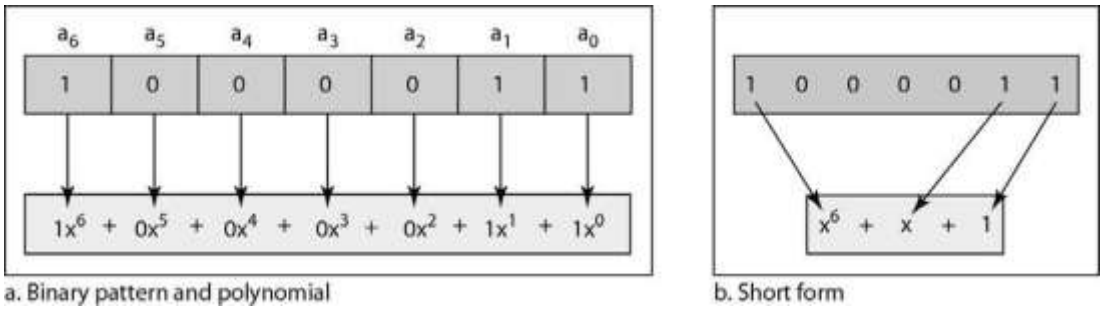


Figure 1: Method of polynomial detection

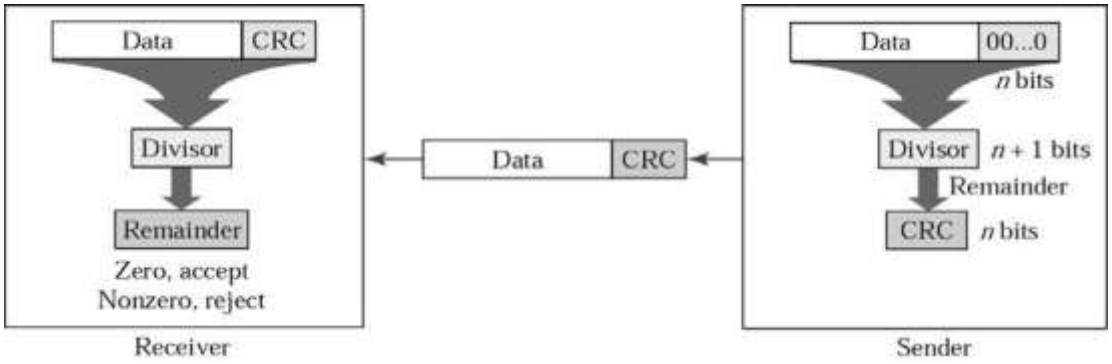


Figure 2: Block diagram of Receiver and Sender

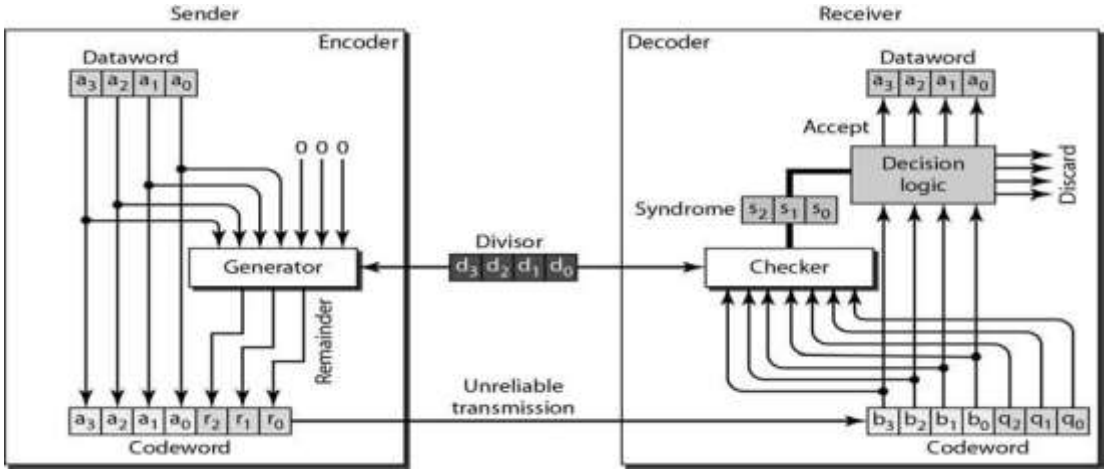


Figure 3: Bitwise Representation of the Encoder and Decoder

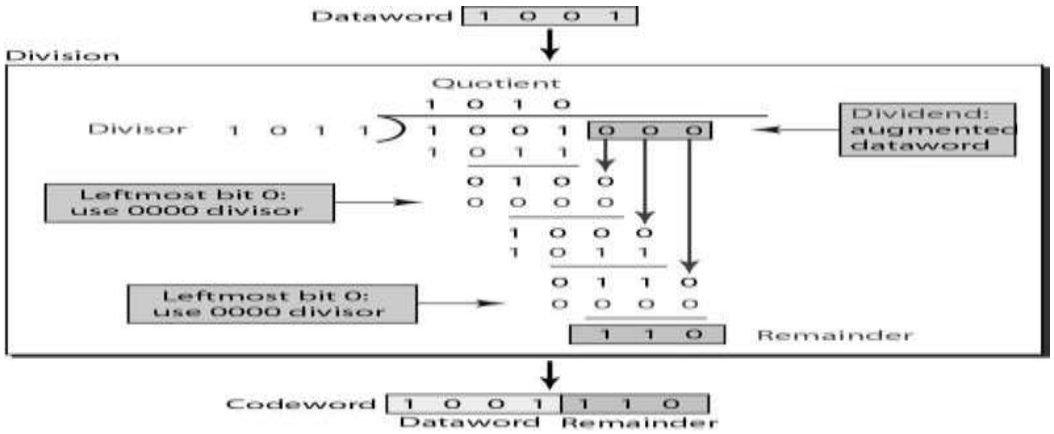


Figure 4: Division in CRC Generator

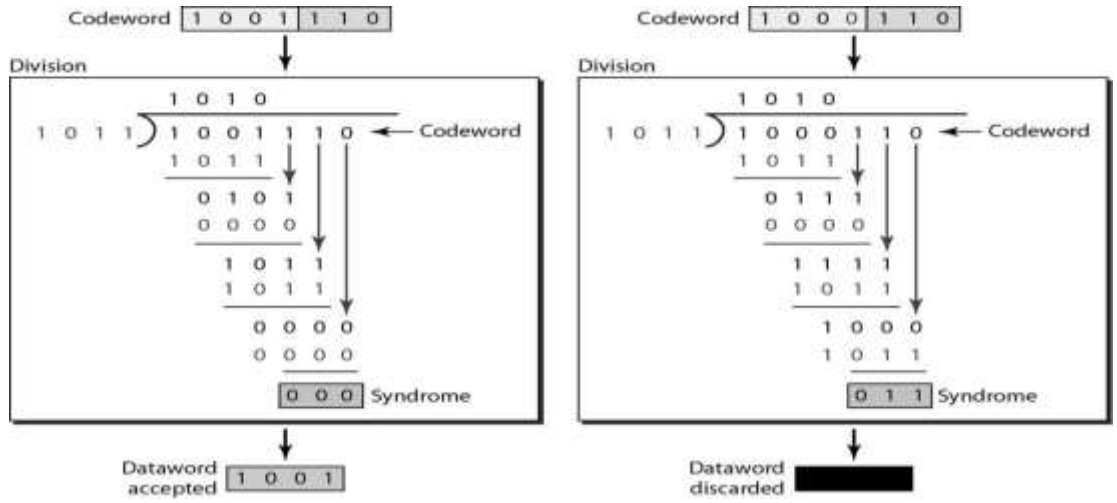


Figure 5: Division in CRC Checker with correct and erroneous codeword

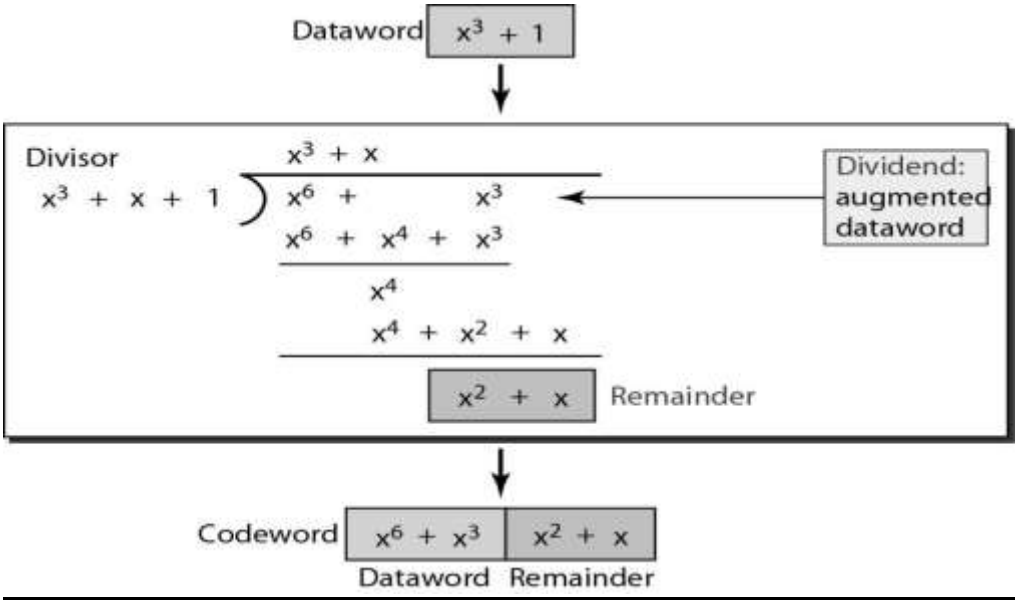


Figure 6: Example of a CRC Division using Polynomial

CRC VHDL CODE

```
library IEEE; use
IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL; use
ieee.std_logic_unsigned.all;

entity crc8bit is
   Port ( reset : in  STD_LOGIC;           clk
: in  STD_LOGIC;           size_data : in
unsigned(15 downto 0);           Data_in : in
STD_LOGIC;           crc_out : out
unsigned (7 downto 0);           crc_ready :
out  STD_LOGIC); end crc8bit;

architecture Behavioral of crc8bit is

signal count : unsigned(15 downto 0) := (others => '0'); signal
crc_temp : unsigned(7 downto 0) := (others => '0');

begin process(clk,reset)
begin
   if(reset = '1') then
crc_temp <= (others => '0');
count <= (others => '0');
crc_ready <= '0';
elsif(rising_edge(Clk)) then
   --crc calculation in the next four lines.
       crc_temp(0) <= Data_in xor crc_temp(7);
crc_temp(1) <= crc_temp(0) xor crc_temp(7);
```

```
crc_temp(2) <= crc_temp(1) xor crc_temp(7);
crc_temp(7 downto 3) <= crc_temp(6 downto 2);

      count <= count + 1; --keeps track of the number of rounds
if(count = size_data + 7) then --check when to finish the calculations
count <= (others => '0');            crc_ready <= '1';        end if;      end if;
end process;

crc_out <= crc_temp;

end Behavioral;
```

RTL SCHEMATICS

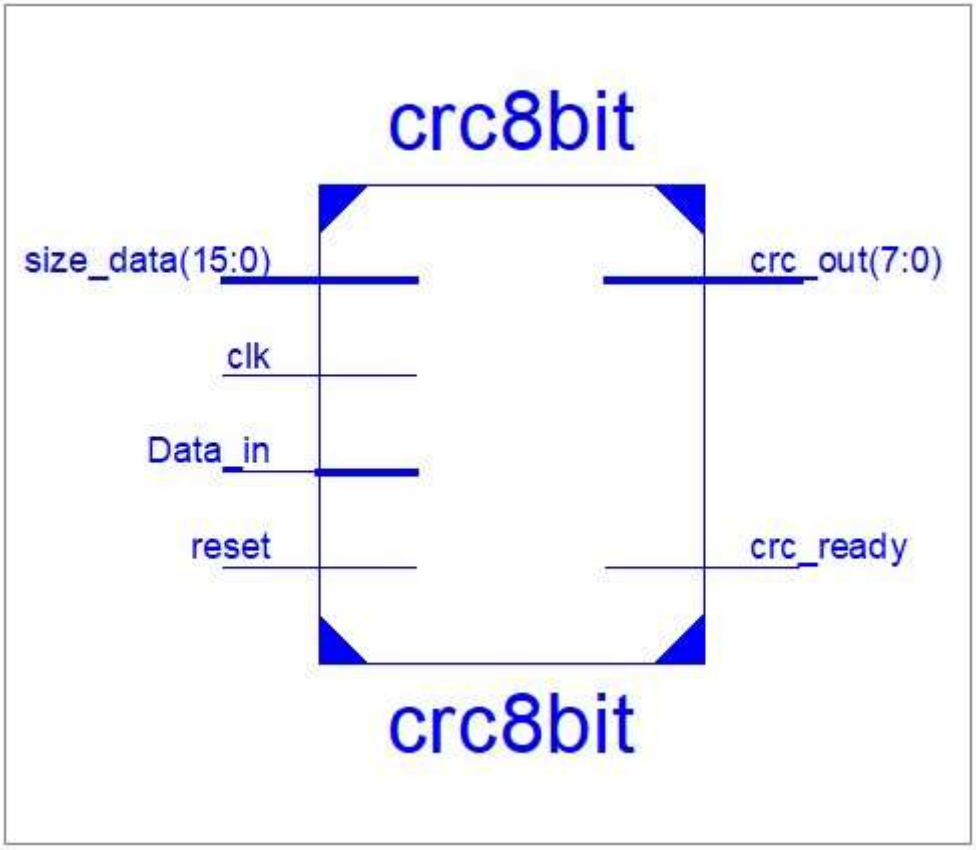


Figure (a): Black Box view of CRC

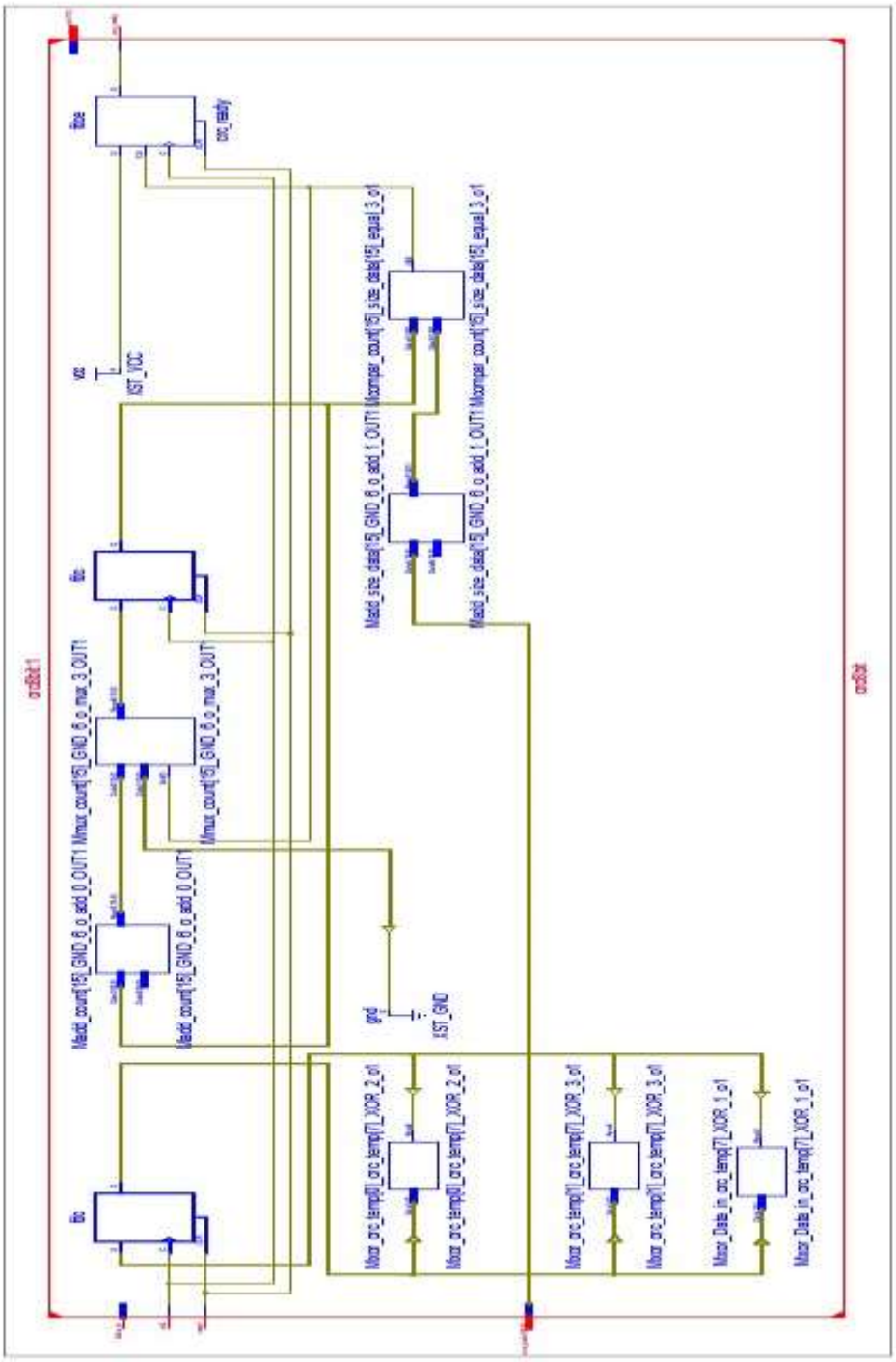


Figure (b): Internal connections of CRC

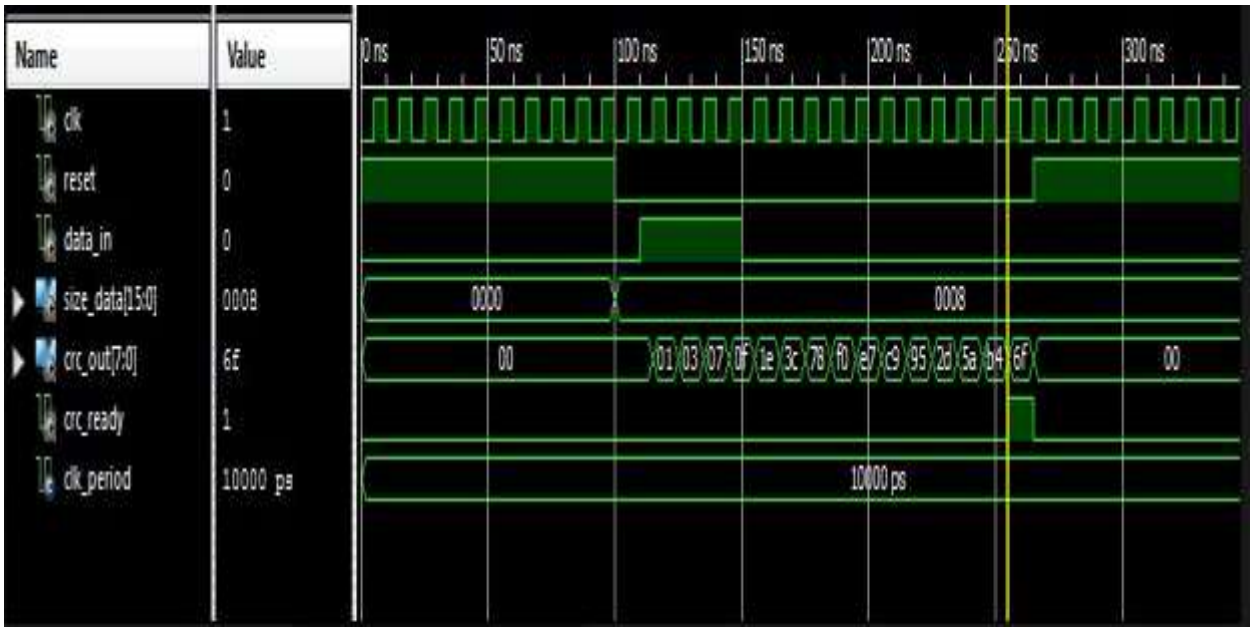
CRC VHDL TEST BENCH

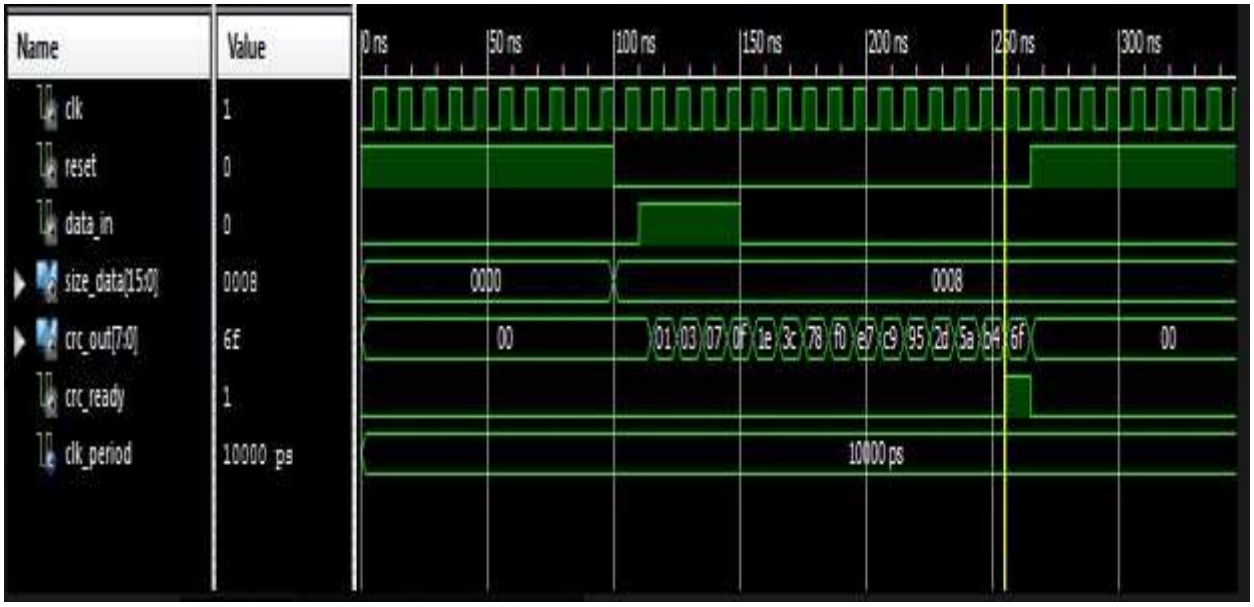
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb IS
END tb;

ARCHITECTURE behavior OF tb IS

   -- CRC Component Declaration for testing.
   COMPONENT CRC8
   port( Clk: in std_logic;          reset : in
std_logic;         size_data : in unsigned(15
downto 0);         Data_in : in std_logic;
crc_out : out unsigned(7 downto 0);
crc_ready : out std_logic
       );
   END COMPONENT;

  --Inputs    signal Clk : std_logic := '0';    signal reset :
std_logic := '0';    signal Data_in : std_logic := '0';     signal
size_data : unsigned(15 downto 0) := (others => '0');
   --Outputs    signal crc_out :
unsigned(7 downto 0);    signal
crc_ready : std_logic;    -- Clock period
definitions    constant Clk_period : time
:= 10 ns;
```

```
BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: CRC8 PORT MAP (
        Clk => Clk,          reset
=> reset,          size_data =>
size_data,          Data_in =>
Data_in,          crc_out =>
crc_out,          crc_ready =>
crc_ready
       );

  -- 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
reset <= '1';
wait for 100 ns;
--Data stream(in
hex) : 78
size_data <=
to_unsigned(8,16);
--data stream is 8
bits in size         wait
until
```

```
falling_edge(Clk);
reset <= '0';
        Data_in <= '0'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
Data_in <= '0'; wait for Clk_period;        wait until
crc_ready = '1'; wait for Clk_period;        reset <=
'1';

        wait for 100 ns;
        --Data stream(in hex) : B800        size_data <=
to_unsigned(16,16); --data stream is 16 bits in size        wait until
falling_edge(Clk);        reset <= '0';
        Data_in <= '1'; wait for Clk_period;
        Data_in <= '0'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
        Data_in <= '1'; wait for Clk_period;
Data_in <= '0'; wait for Clk_period;        wait until
crc_ready = '1'; wait for Clk_period;        reset <=
'1';      wait;    end process;
END;
```

SIMULATION





Figures: Simulation result of CRC's waveform

DISCUSSTION AND CONCLUSION

In this Embedded System project, we were familiarized with running XILINX ISE Design Suite. We have worked in Cyclic Redundancy Check(CRC) project and designed circuit diagram and simulation waveform. We knew that CRC is an errordetecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Since, CRC is only an error detecting method. It does not correct the errors. Our main perspective of this project was to look how actually CRC works in digital systems. For implementing it, we have to create VHDL TEST MODULE for generating RTL of CRC and VHDL TEST BENCH for creating simulation of CRC waveform.

Hence, in this way we were able to observed the RTL and simulation of CRC using VHDL.