International Journal of Students Research in Technology & Management
Vol 1(2), April 2013, pg192-202

A GENERALIZED CODE FOR COMPUTING CYCLIC REDUNDANCY CHECK

Debopam Ghosh, Arijit Mitra, Arijit Mukhopadhyay, Aniket Dawn, Devopam Ghosh
Electronics and Communication Engineering, Heritage Institute of Technology, Kolkata, India
debopamghosh2010@gmail.com, arijitmitra.mitra@gmail.com, arijit2758@gmail.com, aniketdawn@hotmail.com , dave1904@gmail.com

Abstract

This paper focuses on developing a generalized CRC code where the user can vary the size of the generator polynomial [1] such as 9 bits (CRC-8), 17 bits (CRC-16), 33 bits (CRC-32), 65 bits (CRC-64). The working of the code has been shown taking an example and the resulting simulations obtained are shown.

I. INTRODUCTION:

Cyclic Redundancy Check [2] 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 codeword 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 codeword 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 **V**ery high speed Integrated Circuit **H**ardware **D**escription **L**anguage (**VHDL**)[3]. 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.

II. PROCESS OF CRC IMPLEMENTATION:

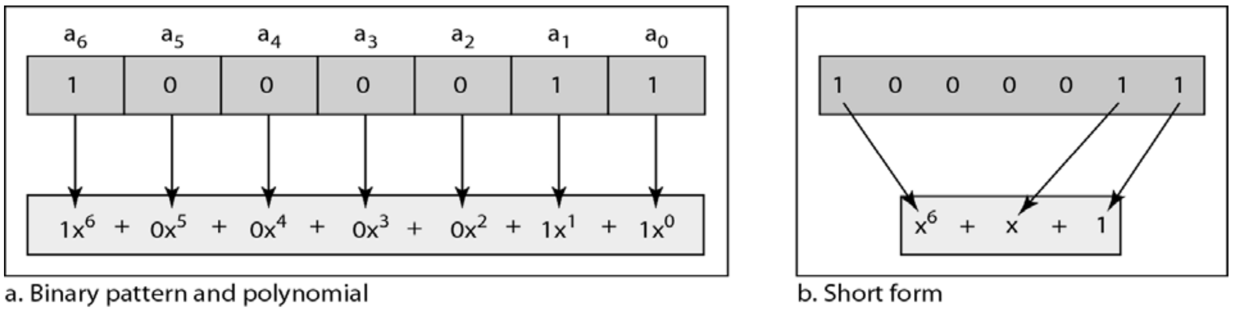
a_6 a_5 a_4 a_3 a_2 a_1 a_0
1 0 0 0 0 1 1
$1x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$
a. Binary pattern and polynomial
1 0 0 0 0 1 1
$x^6 + x + 1$
b. Short form

Figure 1: Method of polynomial detection

The method of determining the polynomial is as follows:

Each value is considered as the coeffiecient of a particular term which is an exponent of x. The rightmost bit is considered as the 0th, the next is the 1st, then 2nd and so on.

For example, 1011 would mean a polynomial of $[(1*x^0)+(1*x^1)+(0*x^2)+(1*x^3)]=x^3+x+1$ (starting from rightmost).

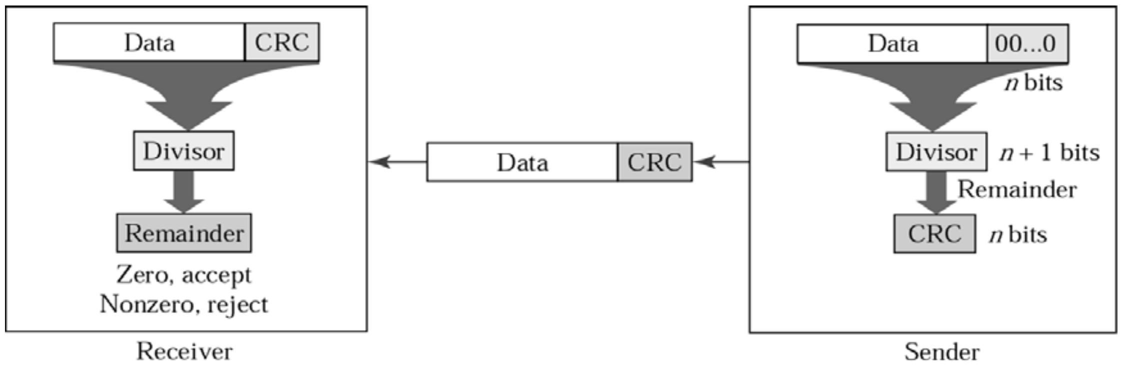

Figure 2: Block Diagram of Receiver and Sender

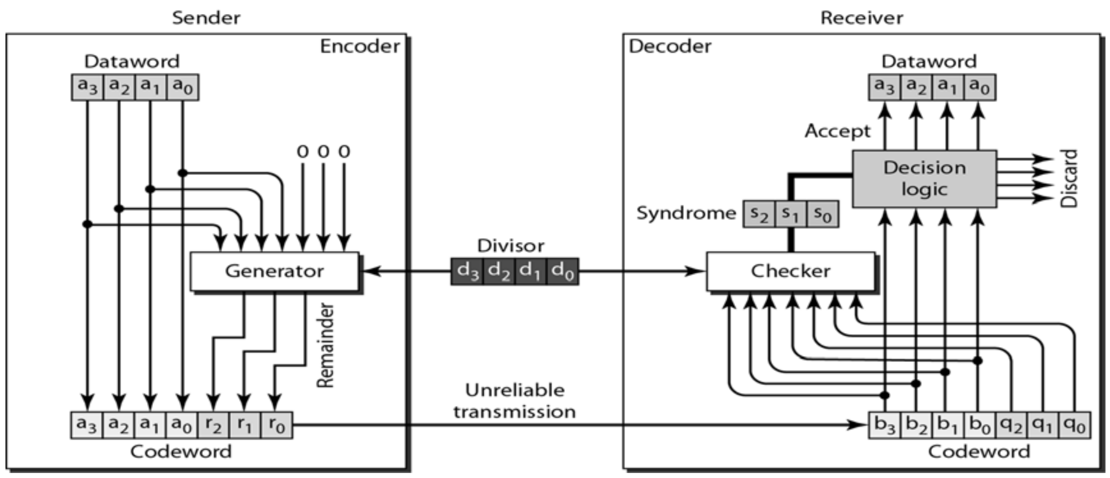

Figure 3: Bitwise Representation of the Encoder and Decoder

Considering a n-bit message is being transmitted and k is the number of data bits. According to the CRC process, a particular polynomial has to be chosen and this polynomial is known as the divisor polynomial. The message is treated as the dividend and the divisor polynomial is used to divide the message polynomial to generate a remainder. The method used for this purpose is known as modulo-2 division[4]. In modulo – 2 division, carry bit in addition and borrow bit in subtraction generated from one particular bit is not carried forward to the next bit. In other words, for subtraction process simple XOR can perform the necessary operations. The message is augmented with (n-k) number of 0's. Then the modulo-2 division is carried out and the remainder of (n-k) bits is generated. This remainder then replaces the (n-k) 0's at the end of the message sequence and it is then transmitted.

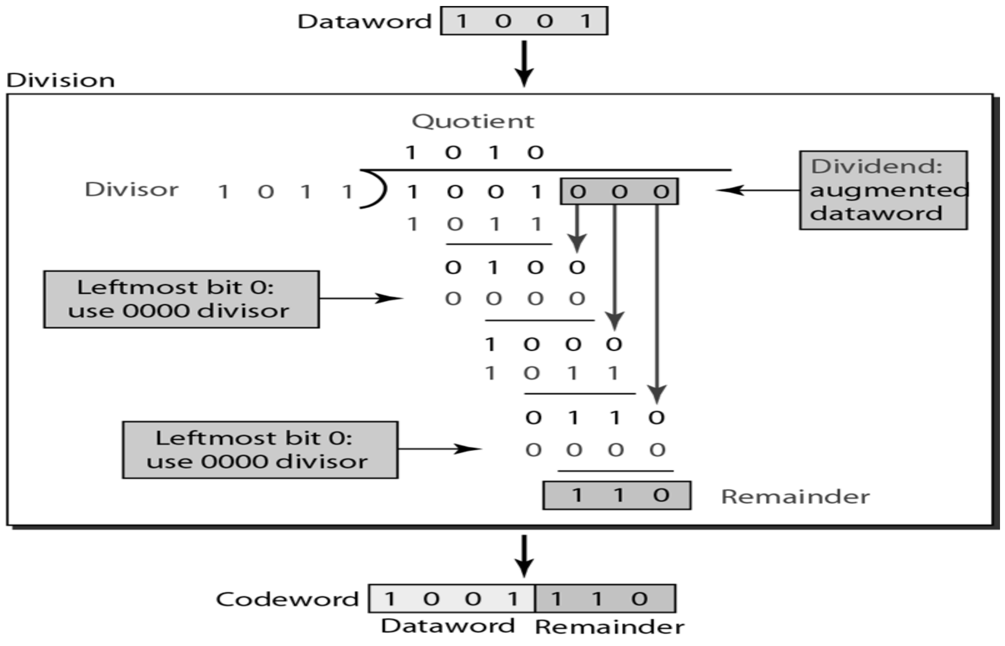

Fig 4: Division in CRC Generator

At the receiver, the data bits appended with the remainder is received as the dataword. The dataword is divided by the same generator polynomial to generate another remainder polynomial. If the polynomial generated is 0, then it is considered as error free. Otherwise the received message contains errors. The entire process is divided into two broad parts; ENCODER and DECODER. All the operations related to transmission of the dataword are carried out in the encoder while the checking operations are carried out in the decoder. For this reason, the encoder is known as CRC Generator and the decoder is known as CRC Checker.

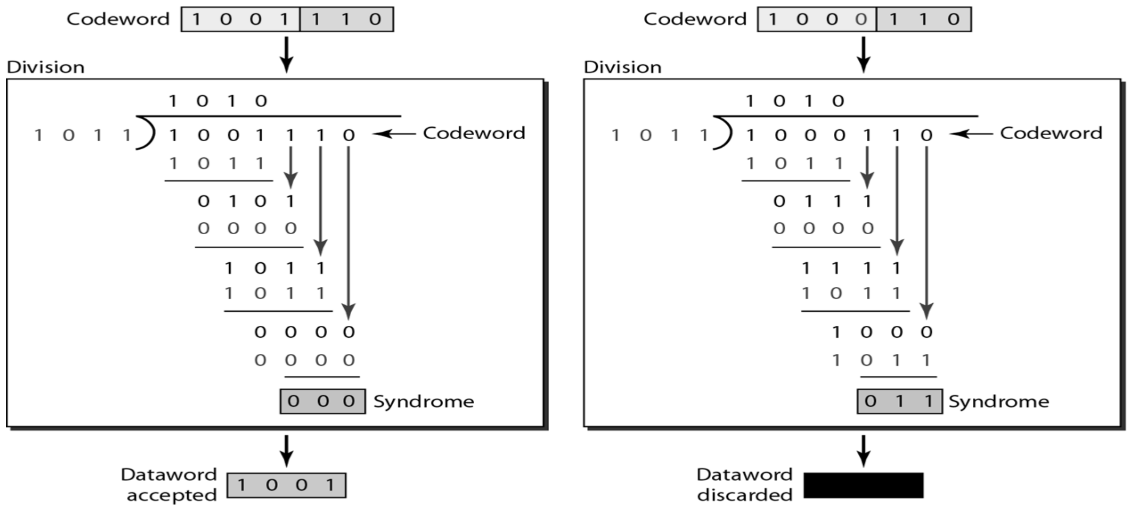

Fig 5: Division in CRC Checker with correct and erroneous codeword

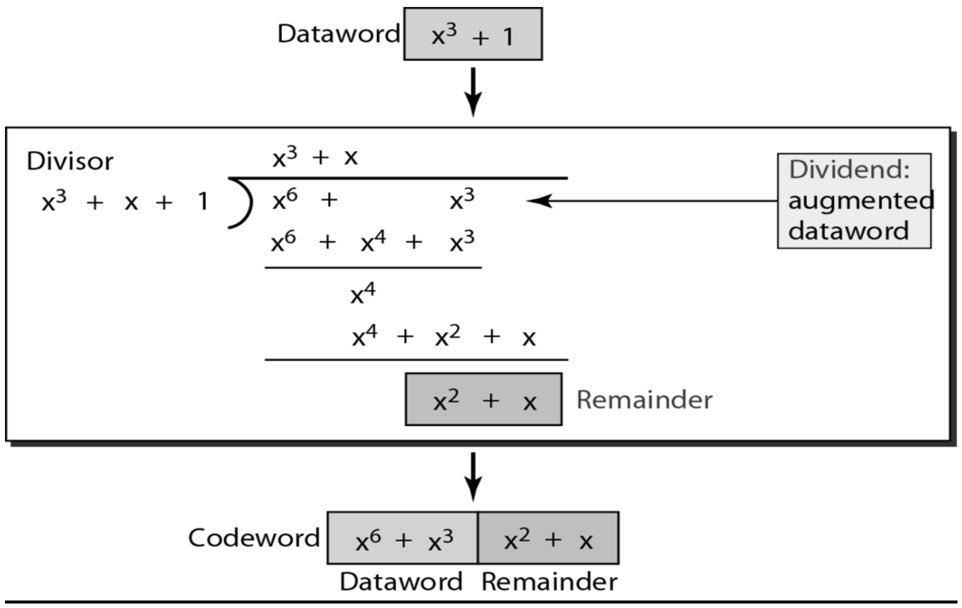

Fig 6: Example of a CRC Division using Polynomial

III. ALGORITHM:

CRC GENRATION:

Step1: Input the message to be sent through port **a**.

Step2: Input the CRC polynomial/divisor through port **b.**

Step3: Define the output port **x**,**t** (**x**-> stores the redundant bits, **t**-> stores the message + redundant bits).

Step4: Begin process and declare the required variables- **u**,**v**,**w**,**y**,**i**,**j**.

Step5: In the variable v store the message bits followed by *(n-1)* 0's.

Step6: **w**= first n bits of **v** and **u**=CRC polynomial/divisor.

Step7: If the MSB of w is 1 then **w** = **w** xor **u** (divisor) else **w** remains unchanged.

Step8: Left shift **w** and discard the MSB.

Step9: The next bit of **v** (in case of 1st iteration $(n+1)^{th}$ bit from the beginning,2nd iteration $(n+2)^{th}$ from beginning etc…) becomes the LSB of **w**.

Step10: Repeat steps 7-8-9 till the end of **v** is reached (there will be a single iteration after LSB of **v** is be added to **w**).

Step11: Port **x**(redundant bits/remainder) = first (*n-1)* bits of **w.**

Step12: Port t = message bits + redundant bits.

CRC CHECK:

Step1: Input the received bits and CRC polynomial through port **a** and **b** respectively.

Step2: As before follow the steps to generate the remainder and store in port **x**.

Step3: If the remainder is all 0's then the received message is error free and **t**=received bits- redundant bits.

Step4: If remainder is not all 0's then there is an error and the message is discarded so **t**= all 0's.

CRC GENERATOR VHDL CODE

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

package my_package is
        constant m:integer:=8;
        constant n:integer:=4;
end my_package;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.my_package.all;

entity crc_new is
  Port ( a : in  STD_LOGIC_VECTOR (m-1 downto 0);   ---message bits
       b : in  STD_LOGIC_VECTOR (n-1 downto 0);     ---crc polynomial
       clk : in  STD_LOGIC;
       x : out  STD_LOGIC_VECTOR (n-2 downto 0);   ---redundant bits
       t : out  STD_LOGIC_VECTOR (m+n-2 downto 0));        ---message with redundant bits
end crc_new;

architecture Behavioral of crc_new is
begin
        process(clk)
        variable v:std_logic_vector(m+n-2 downto 0);
        variable u:std_logic_vector(n-1 downto 0);
        variable w:std_logic_vector(n-1 downto 0);
        variable y:std_logic_vector(n-1 downto 0);
        variable i,j:integer:=0;
        begin
                v(m+n-2 downto n-1):=a(m-1 downto 0);

                for j in n-2 downto 0 loop
                        v(j):='0';
                        end loop;
                u:=b;
                w:=v(m+n-2 downto m-1);
          for i in m-1 downto 0 loop
                        if(w(n-1)='1') then
                                w:=w xor u;
                        else
                                null;
                        end if;
                        y:=w;
                        w(n-1 downto 1):=y(n-2 downto 0);
                        if(i=0) then
                                w(0):='0';
                                else
                                w(0):=v(i-1);
                                end if;
                        end loop;
                x<=w(n-1 downto 1); ---- redundant bits
                t(m+n-2 downto n-1)<=a;
                t(n-2 downto 0)<=w(n-1 downto 1);
                end process;
end Behavioral;
```

RTL SCHEMATICS

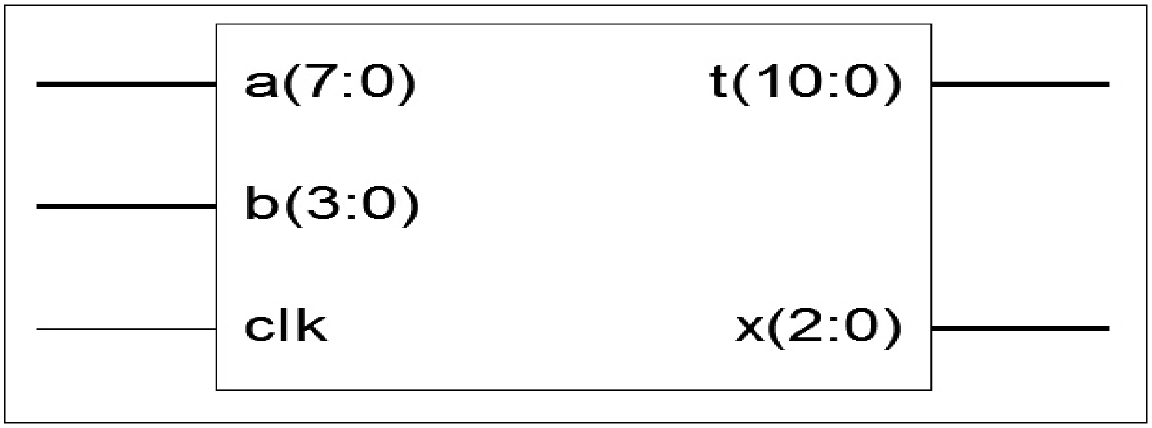

Fig 7: Black Box view of C RC Generator

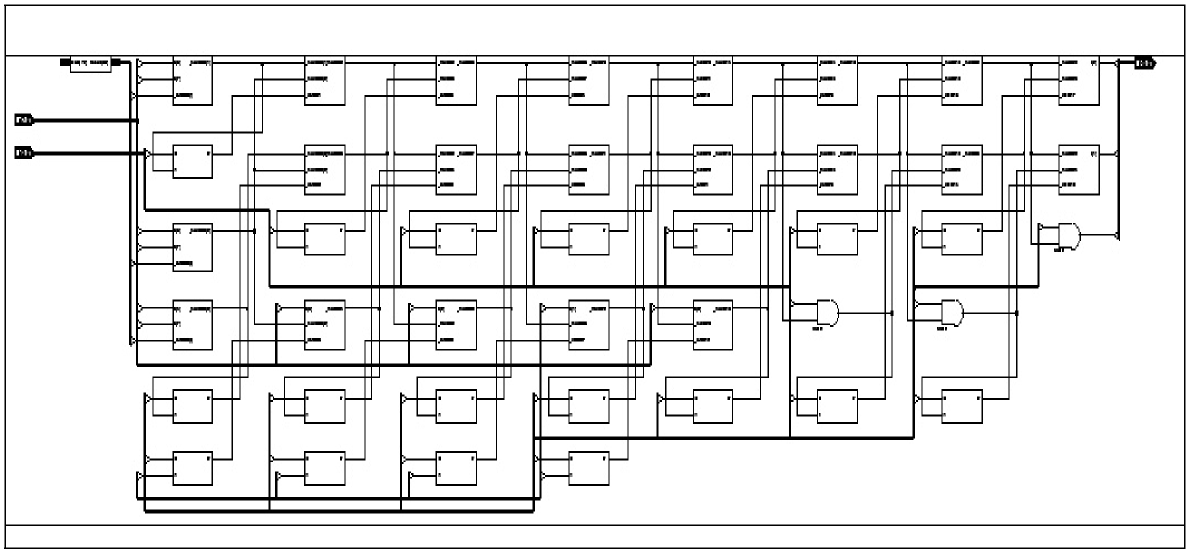

Fig 8: Internal connections of CRC Generator

CRC CHECKER VHDL CODE

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

package my_package is
        constant m:integer:=8;
        constant n:integer:=4;
end my_package;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.my_package.all;

entity crc_new is
   Port ( a : in  STD_LOGIC_VECTOR (m-1 downto 0);   ---message bits
          b : in  STD_LOGIC_VECTOR (n-1 downto 0);    ---crc polynomial
          clk : in  STD_LOGIC;
          x : out  STD_LOGIC_VECTOR (n-2 downto 0);  ---redundant bits
          t : out  STD_LOGIC_VECTOR (m+n-2 downto 0));        ---message with redundant bits
end crc_new;

architecture Behavioral of crc_new is

begin
            process(clk)
            variable v:std_logic_vector(m+n-2 downto 0);
            variable u:std_logic_vector(n-1 downto 0);
            variable w:std_logic_vector(n-1 downto 0);
            variable y:std_logic_vector(n-1 downto 0);
            variable i,j:integer:=0;
            begin
                        v(m+n-2 downto n-1):=a(m-1 downto 0);

                        for j in n-2 downto 0 loop
                                    v(j):='0';
                                    end loop;

                        u:=b;
                        w:=v(m+n-2 downto m-1);
                        for i in m-1 downto 0 loop
                                    if(w(n-1)='1') then
                                            w:=w xor u;
                                    else
                                            null;
                                    end if;
                                    y:=w;
                                    w(n-1 downto 1):=y(n-2 downto 0);
                                    if(i=0) then
                                            w(0):='0';
                                            else
                                            w(0):=v(i-1);
                                            end if;
                                    end loop;
                        x<=w(n-1 downto 1); ---- redundant bits
                        t(m+n-2 downto n-1)<=a;
                        t(n-2 downto 0)<=w(n-1 downto 1);   --- total message
            end process;
end Behavioral;
```

RTL SCHEMATICS

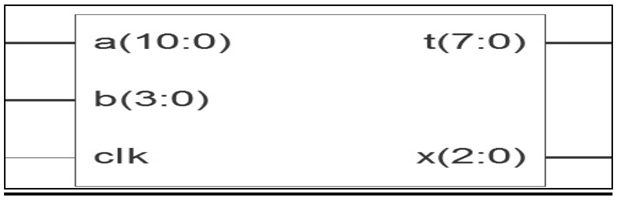

Fig 9: Black Box of CRC Checker

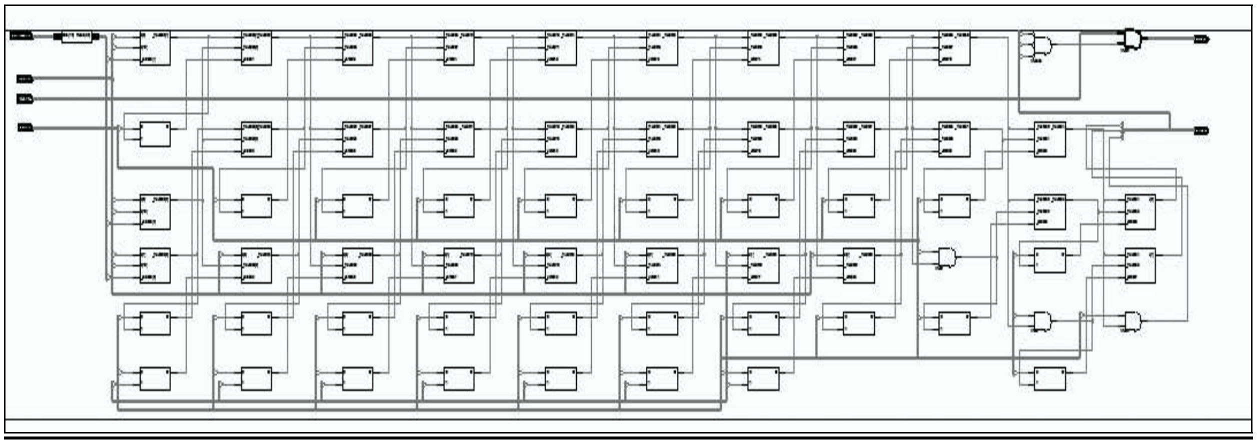

Fig 10: Internal connections of CRC checker

IV. SIMULATION:

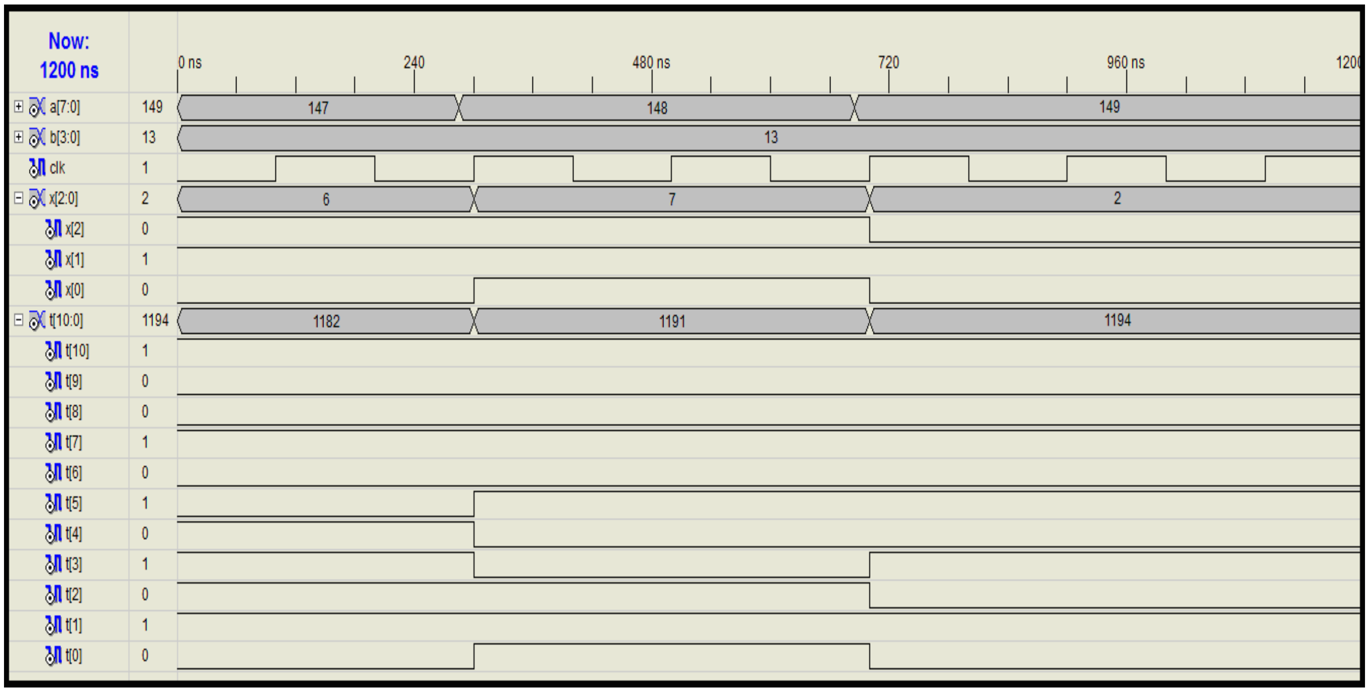

Fig 11: Simulation result of CRC generator waveform

Different input datawords have been sent at different instants of time. In the first instant, “10010011110”, at the next instant ‘10010100111” and at the next instant, ‘10010101010” is transmitted from the encoder. The different values like 1182, 1191, 1194 etc represent the input dataword in decimal.

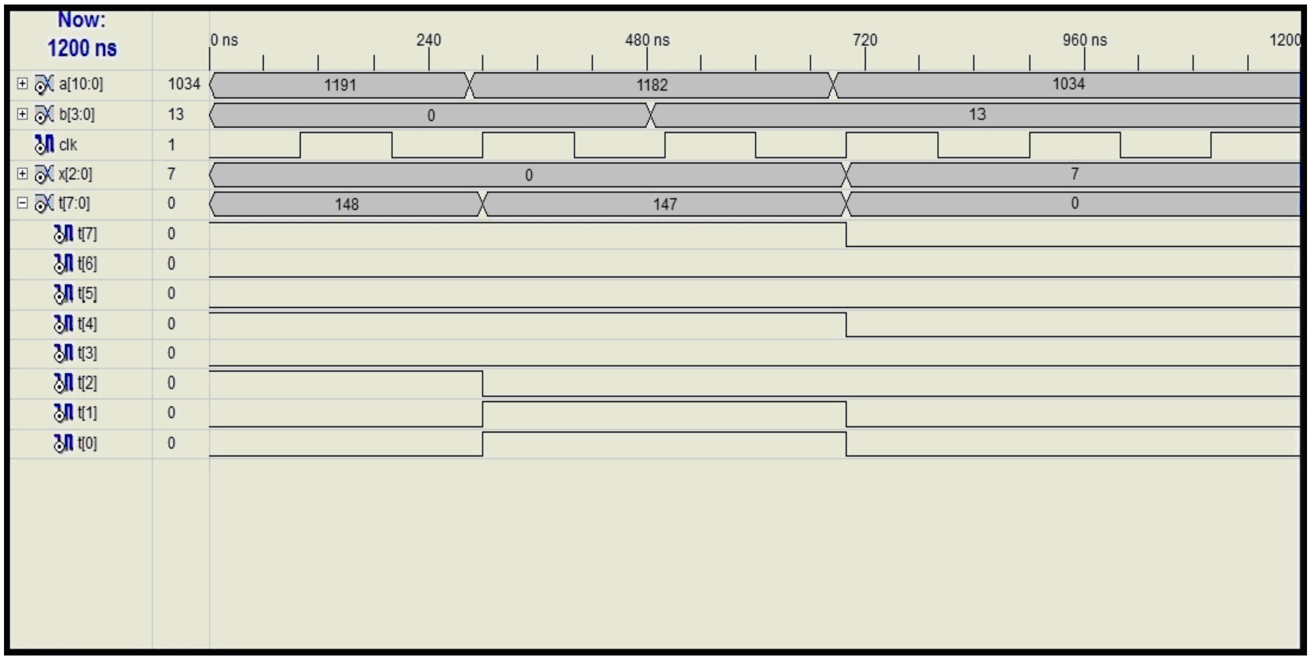

Fig 12: Simulation result of CRC check waveform

At the decoder, we see that the remainder (t) is equal to a string of 0's, showing that if the received codeword (a) is same as that of the transmitted codeword (t) in the encoder, there is no error. Here, the received codeword has been intentionally made to be equal to the transmitted codeword to show a successful transmission.

V. CONCLUSION:

In this paper, the process of CRC generation and checking has been discussed in detail. The methods applied to detect an error during transmission has been shown using simulation in VHDL. However CRC has some limitations:

- CRC is only an error detecting method. It does not correct the errors.
- The divisor polynomial should be chosen carefully. The divisor polynomial has to be a multiple of $(x+1)$. If any random polynomial is chosen then it may result into wrong calculation of the remainder (CRC).

REFERENCES:

1. Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks By Philip Koopman and Tridib Chakravarty
2. CRC Cyclic Redundancy Check Analysing and Correcting Errors By Prof. Dr. W. Kowalk
3. VHDL basics By Raunak Ranjan
4. http://en.wikipedia.org/wiki/Cyclic_redundancy_check