The Design and Implementation of a Programmable Cyclic Redundancy Check (CRC) Computation Circuit Architecture Using FPGA

Rameshwr T. Murade, MD. Manan Mujahid, M.A.M. Sabir

Abstract*: *Many communication systems use the cyclic redundancy code (CRC) technique for protecting key data fields from transmission errors by enabling both single-bit error correction and multi-bit error detection.[6] Cyclic redundancy check (CRC) coding is an error-control coding technique for detecting errors that occur when a message is transmitted. Data integrity is imperative for many network protocols, especially data-link layer protocols.[4] Techniques using parity codes and Hamming codes can be used for data verification, but CRC is the preferred and most efficient method used for detecting bit errors produced from medium related noise. For example, Ethernet uses a 32-bit CRC polynomial for error detection. Data storage is another area where CRC error detection is becoming increasingly important. iSCSI implementations that utilize the TCP/IP protocol to implement Storage Area Networks (SANs) require error detection to be deployed. These operate using multi-gigabit connection speeds and thus require CRC checks to be executed at high speed as well. [9]

Index Terms*— *CRC, Error Correction, implementation with CRC 32, FPGA CRC

I. INTRODUCTION

Error correction codes provides a mean to detect and correct errors introduced by the transmission channel. Two main categories of codes exist: block codes and convolution codes. They both introduce redundancy by adding parity symbols to the message data. Cyclic redundancy check (CRC) codes are the subset of the cyclic codes that are also a subset of linear block codes. Cyclic Redundancy Check (CRC) is widely used to detect errors in data communication and storage devices. CRC is a very powerful and easily implemented technique to obtain data reliability. The CRC technique is used to verify the integrity of blocks of data called Frames. In this technique, the transmitter appends an extra n bit sequence to every frame called Frame Check Sequence (FCS). FCS holds redundant information about the frame that helps the receiver detect errors in the frame. When the transmission is received or the stored data is retrieved, the CRC residue is regenerated and confirmed against the appended residue.[3] yclic Redundancy Check is an established technique for detecting errors on serial data communication links and in mass storage devices. A frame check sequence is appended to the message for transmission error detection in Many (high-speed) serial data communication protocols.

Manuscript Received November 2013.

Mr. Rameshwar T. Murade . MTech (ECE), SCET, Hyderabad,India.
Prof. MD. Manan Mujahid, Asst. Prof. & H.O.D, Dept. of ECE, SCET Hyderabad, India .
Prof. M.A.M. Sabir , Asst. Prof., Dept. of ECE, SCET, Hyderabad, India

The common hardware solution for CRC calculation is the linear feedback shift register (LFSR), consisting a few Flip Flops and Logic Gates. CRC is the preferred and most efficient method used for detecting bit errors produced from medium related noise. Data storage is another area where CRC error detection is becoming increasingly important. CRC checks to be executed at high speed as well as parallel processing. The ability of CRC implemented in hardware to be reconfigurable to handle new Generator polynomials will offer a key advantage in this fast developing area. The reconfigurable CRC circuit that has been implemented can quickly switch between any polynomial gives it a key advantage over the other circuits.[5] The design and implementation of a programmable cyclic redundancy check (CRC) computation circuit architecture, suitable for deployment in network related system-on-chips (SoCs) is presented. The architecture has been designed to be field reprogrammable so that it is fully flexible in terms of the polynomial deployed and the input port width. The circuit includes an embedded configuration controller that has a low reconfiguration time and hardware cost. The circuit has been synthesized and mapped to less than 130-nm UMC standard cell ASIC technology and is capable of supporting line speeds of more than 5Gb/s.[4] This paper begins with CRC and background related to CRC. The subsequent section describes CRC as error correcting code. Next section include software implementation with CRC32,,next is FPGA & finally future scope & conclusion.

II. CYCLIC REDUNDANCY CHECK

In digital communication systems, the error detection is performed by computing checksum on the message that needs to be transmitted. The computed checksum is then concatenated to the end of the message to generate the codeword or the check sequence number to be transmitted. At the receiving end, the received word is compared with the transmitted codeword. If both are equal, then the message received is treated as error free, otherwise there is an error detected in the received word.[2]

Cyclic Redundancy Codes (CRCs) are used in a wide variety of computer networks and data storage devices to provide inexpensive and effective error detection capabilities. As data transfer rates and the amount of data stored increase, the need for simple, cheap, and robust error detection codes increases as well. Thus it is important to be sure that the CRCs in use are as effective as possible.[8]

CRC is a polynomial-based block coding method for detecting errors in blocks or

frames of data. A set of check digits is computed for each frame scheduled for transmission over a medium that may introduce error and is appended to its end. The computed check digits are known as the frame check sequence (FCS). A CRC value is calculated as a remainder of the modulo-2 division of the original transmitted data with a specific CRC generator polynomial. For example, Ethernet uses the 32-bit polynomial value [1] A CRC is one of the an error-detecting code. Its computation resembles a polynomial long division operation in which the quotient is discarded and the remainder becomes the result, with the important distinction that the polynomial coefficients are calculated according to the carry-less arithmetic of a finite field. The length of the remainder is always less than the length of the divisor called the generator polynomial , which therefore determines how long the result can be. The definition of a particular CRC specifies the divisor to be used, among other things. An important reason for the popularity of CRCs for detecting the accidental alteration of data is their efficiency guarantee. Typically, an n-bit CRC, applied to a data block of arbitrary length, will detect any single error burst not longer than n bits and will detect a fraction 1−2−n of all longer error bursts. Errors in both data transmission channels and magnetic storage media tend to be distributed non-randomly i.e. are „bursty" , making CRCs' properties more useful than alternative schemes such as multiple parity checks.The simplest error-detection system, the parity bit, is in fact a trivial 1-bit CRC, it uses the generator polynomial x+1.[5] Cyclic redundancy check (CRC) is an error detecting code that is widely used to detect corruption in blocks of data that have been transmitted or stored. A standalone intellectual property (IP) core is ideal for accelerating CRC computation in many network and server applications. Hardware configurability that will allow unrestricted CRC sizes and polynomials to be deployed enables a wide range of network transmission, storage and security applications to be supported at a low cost. The cost of chip design continues to increase due to factors such as high mask and respin costs. Next generation system-on-chip (SoC) designs are highly expensive and therefore must be configurable to a range of applications and future proof where either product updates or protocol migration can occur. Adding flexibility through in-field hardware configurability is a key method that enables the cost of designs to be reduced.[9]

A. Background related to CRC

Cyclic redundancy codes (also known sometimes as cyclic redundancy checks) have a long history of use for error detection in computing. [Peterson72] and [Lin83] are among the commonly cited standard reference works for CRCs. A treatment more accessible to non-specialists can be found in [Wells99].[8].CRC functions have been widely implemented in software using methods such as lookup tables and shift and addition . Further research has investigated hardware architectures that can better exploit parallelism. The fundamental work on parallel CRC computation was introduced by Pei in 1992 [6]. Braun addressed the hardware mapping problem of the parallel CRC algorithm by introducing a slightly different matrix computation technique than Pei. Braun incorporated pre- and post- CRC computation circuits to achieve a 32-bit checksum word at 450 Mbps using FPGA technology in 1996. addresses a technique that allows pipelining to increase the circuit speed, independent of the underlying technology[1] A CRC can be thought of as a (non-secure) digest function for a data word that can be used to detect data corruption. Mathematically, a CRC can be described as treating a binary data word as a polynomial over GF(2) (i.e., with each polynomial coefficient being zero or one) and performing polynomial division by a generator polynomial G(x). The generator polynomial will be called a CRC polynomial for short. (CRC polynomials are also known as feedback polynomials, in reference to the feedback taps of hardware-based shift register implementations.) The remainder of that division operation provides an error detection value that is sent as a Frame Check Sequence (FCS) within a network message or stored as a data integrity check. Whether implemented in hardware or software, the CRC computation takes the form of a bitwise convolution of a data word against a binary version of the CRC polynomial.[8] Further research CRC addresses the problem of processing variable sized packets in parallel by simply duplicating circuits and multiplexing between multiple custom implementations as required, i.e., if processing 32 bits and the last cycle of data is only 8 bits wide then this implementation multiplexes the data from a 32-bit circuit to an 8-bit circuit. The research details the VLSI implementation of a CRC-32 circuit for Ethernet. A standard cell and full custom implementation are presented using 180- and 350-nm technology respectively, operating at 1.09 GHz and 625 MHz. The circuits presented are highly customized and targeted for the CRC-32 polynomial selected. Although they operate very fast, the designs are not flexible or adaptable as they are intended for a single polynomial [4] VHDL code with a generic construct that allows a designer to synthesise CRC circuits for any desired polynomial of length up to 32 bits. Word widths of 8, 12, 16, and 32 bits have been analyzed. The research concentrates on generating code in a generic style that includes parallelism in its structure, which is based on the linear feedback shift register (LFSR) presented by Pei. While this generic description is useful in terms of design reusability, it is only configurable presynthesis, after which the hardware is fixed and the CRC function is not configurable.[9] The overall implementation operates on a 1.7 GHz Pentium M processor, which supports 3.6 Gb/s. None of the aforementioned state-of-the-art options support full in-field configuration flexibility at high speed specifically on hardware. Some allow flexibility in the design phase and others offer very high line-speed performance, however none offer high line-speed with full flexibility, such as the support of different data-path widths and CRC generator polynomials. Although the software option is likely to be very flexible, it comes at the expense of a Pentium processor.[4] CRC is a commercially available core that operates on FPGA. Again, this uses a fixed CRC polynomial that cannot be reconfigured after deployment. The CRC-32 core is able to support 10/40 Gb/s line speeds by utilizing 64-/256-bit data buses, respectively. It is the wide data buses that allow this performance to be achieved. However, using wide input buses adds complexity to the CRC calculation where the end of a word does not fully fill the input bus. If the end of a word is 16-bits wide

then the CRC must be computed for 16 bits, this cannot be done using a 32-bit input configuration.[9]

III. CRC AS ERROR CORRECTING CODE

In information theory and coding theory with applications in computer science and telecommunication, error detection and correction or error control are techniques that enable reliable delivery of digital data over unreliable communication channels. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors, and error correction enables reconstruction of the original data. Error detection is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver. Error detection techniques are that enable reliable delivery of digital data over unreliable communication channel. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors.[5]

CRC functions have been widely implemented in software using methods such as lookup tables and addition. CRC is most efficient method used for detecting bit errors produced from medium related noise .CRCs are particularly easy to implement in hardware. Cyclic Redundancy Check (CRC) is an error detecting code that is widely used to detect corruption in blocks of data that have been transmitted or stored. Enables a wide range of network transmission, storage and security applications to be supported at a low cost. CRCs are specifically designed to protect against common types of errors on communication channels., it is a single or burst error detecting code designed to detect accidental changes to digital data in computer networks. It is characterized by specification called G(x), Generator Polynomial. Goal to maximize the probability of detecting an error.[5]

A. Computation of CRC using binary division

The most powerful of the redundancy checking techniques is the cyclic redundancy check (CRC). CRC is based on binary division. In CRC, instead of adding bits to achieve a desired parity, a sequence of redundant bits, called the CRC or the CRC remainder, is appended to the end of a data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number. At its destination, the incoming data unit is divided by the same number. If at this step there is no remainder the data unit is assumed to be intact and is therefore accepted. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

The redundancy bits used by CRC are derived by dividing the data unit by a predetermined divisor; the remainder is the CRC. To be valid, a CRC must have two qualities: It must have exactly one less bit than the divisor, and appending it to the end of the data string must make the resulting bit sequence exactly divisible by the divisor.[10]

Both the theory and the application of CRC error detection are straightforward. The only complexity is in deriving the CRC. To clarify this process, we will start with an overview and add complexity as we go. Figure 1 provides an outline of the basic steps.

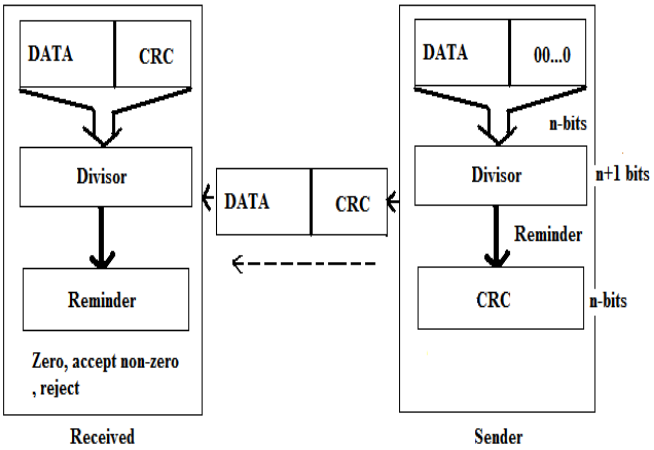

Fig. CRC sender & Receiver

First, a string of n 0's is appended to the data unit. The number n is 1 less if-number of bits in the predetermined divisor which is n + 1 bits.

Second, the newly elongated data unit is divided by the divisor, using a p called binary division. The remainder resulting from this division is the CRC.

Third, the CRC of n bits derived in step 2 replaces the appended 0's at the data unit. Note that the CRC may consist of all 0's.

The data unit arrives at the receiver data first, followed by the CRC. The receiver treats the whole siring as a unit and divides it by the same divisor that was used the CRC remainder. If the siring arrives without error, the CRC checker yields a remainder of zero, the data unit passes. If the string has been changed in transit, the division yields zero remainder and the data unit does not pass.

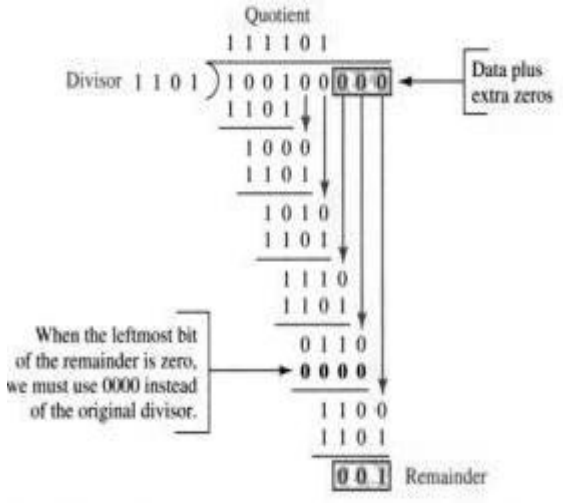

Fig. 2 generation of CRC

A CRC checker functions exactly as the generator does. After receiving the data appended with the CRC, it does the same modulo-2 division. If the remainder is all 0's, the CRC is dropped and the data are accepted: otherwise, the received stream of bits is discarded and data are resent. Figure 14 shows the same process of division in the receiver. We assume that there is no error

The remainder is therefore all 0's, and the data are accepted.[10]

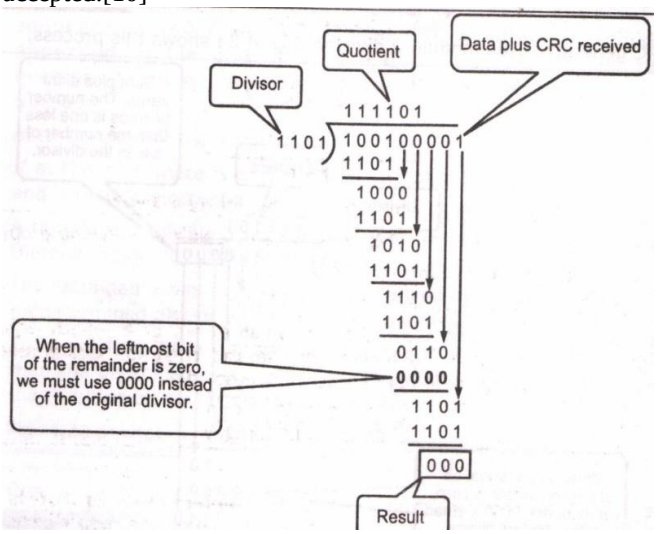

Fig. 3 CRC checker

B. Performance of CRC

CRC is a very effective error detection method. If the divisor is chosen according to the previously mentioned rules,

1. CRC can detect all burst errors that affect an odd number of bits.
2. CRC can detect all burst errors of length less than or equal to the degree of the polynomial
3. CRC can detect, with a very high probability, burst errors of length greater than the degree of the polynomial.[10]

IV. SOFTWARE IMPLEMENTATION OF CRC

CRC is an error checking mechanism for a block of data, such as a frame of network traffic. CRCs are popular because they are simple to implement in digital hardware, are easy to analyze mathematically, and are particularly good at detecting common errors caused by noise in transmission channels.[11]

A. Implementation of CRC-32

The single-bit data input (serial) calculation of CRC-32 is implemented with a linear feedback
shift register (LFSR). The CRC-32 LFSR is illustrated in fig.4 (register bits "3" through "25"
are left out of the figure to simplify the drawing).[12]
Presetting the flip-flops to 0xFFFFFFFF is equivalent to complementing the first 32-bits of the
data stream. For the first 32 cycles, the right-most XOR gate in the figure is an inverter. The
XOR of any data with a binary "1" results in the complement of the original data. [12]

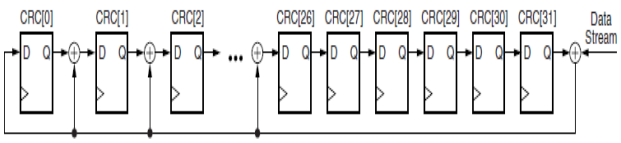

Fig.4 Linear Feedback Shift Register Implementation of CRC-32

The serial implementation is not optimal as most IEEE 802.3 transceivers communicate the data stream to the Media Access Controller (MAC) over a 4-bit bus; therefore, a parallelimplementation is necessary. . [12]

B. *A Complete Implementation of CRC-32*

A simplified block diagram of the complete implementation is illustrated in fig.5 The "32-bit CRC" register shown fig.5 has three modes of operation: initialize, shift, and calculate next CRC value. This register is enabled when either load_init or d_valid are asserted. . [12]

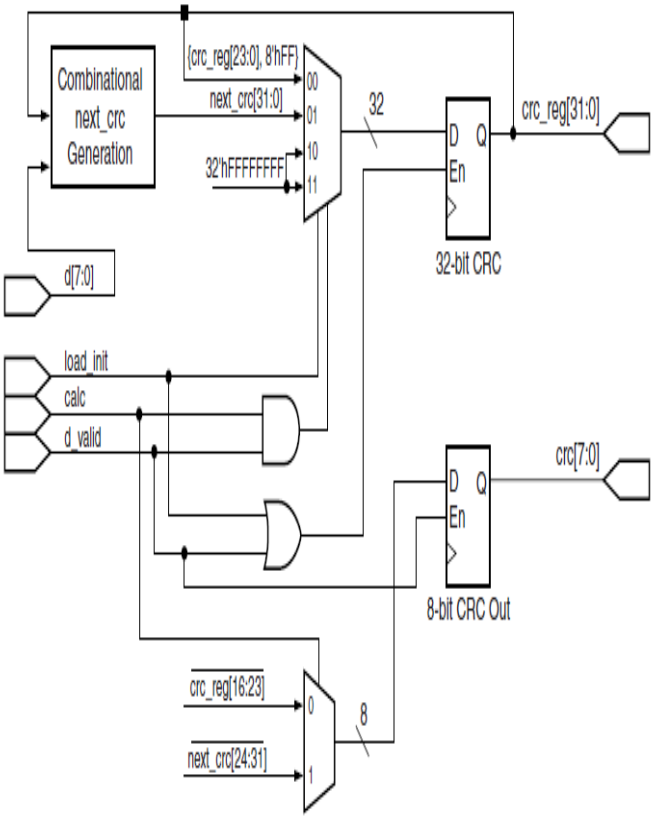

Fig.5 Block Diagram of CRC-32 Implementation

The "8-bit CRC Out" register duplicates 8-bits of crc_reg[31:0] in order to do the final bit
reversal and complement in parallel with the calculation of the CRC. When the calc input is
asserted, the "8-bit CRC Out" register gets the bit-reversed and complemented most significant
eight bits of next_crc. On the other hand, when the calc input is de-asserted, the "8-bit CRC
Out" register gets the bit-reversed and complemented second most significant eight bits of
crc_reg - the second most significant eight bits are used here to replicate the shift operation
of the "32-bit CRC" register. . [12]
The "8-bit CRC Out" register always contains the bit-reversed and complimented most
significant bits of the "32-bit CRC" register. The final IEEE 802.3 FCS can be read from the
"8-bit CRC Out" register by asserting d_valid four times after the de-assertion of calc. . [12]
An example waveform is shown in fig.6 and fig.7. [12]

I.: CRC-32 Implementation Truth Table. [12]

{load_init, calc, d_valid}	crc_reg[31:0]	crc[7:0]
0 0 0	no change	no change
0 0 1	{crc_reg[23:0], 8'hFF}	~crc_reg[16:23]
0 1 0	no change	no change
0 1 1	next_crc[31:0]	~next_crc[24:31]
1 0 0	32'hFFFFFFFF	no change
1 0 1	32'hFFFFFFFF	~crc_reg[16:23]
1 1 0	32'hFFFFFFFF	no change
1 1 1	32'hFFFFFFFF	~next_crc[24:31]

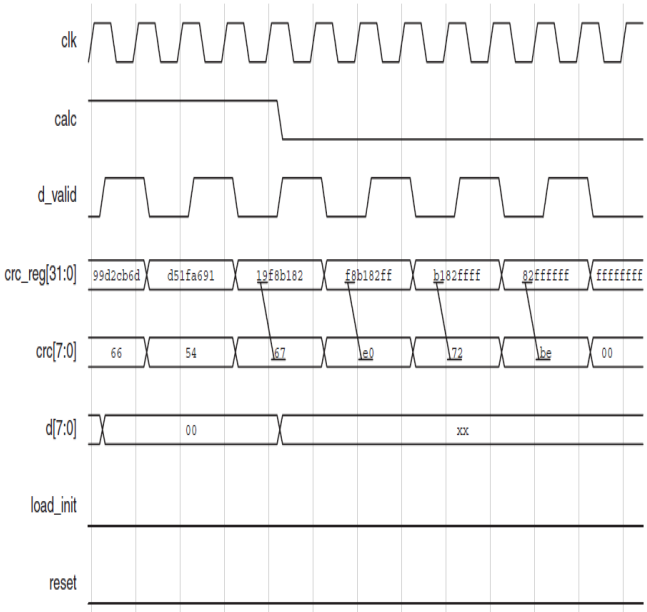

Fig. 6: Example Use of "CRC Out" Register During Transmit

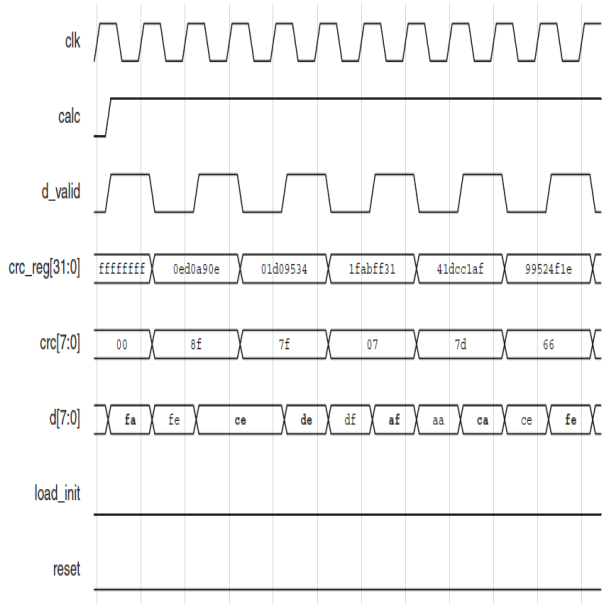

Fig. 7: Example CRC-32 Waveform From the Beginning of the Frame

A 32-bit compare block can be added for checking the CRC of received frames. First, the data stream and CRC of the received frame are sent through the circuit in fig5 .Then the value left in the CRC-32 registers can be compared with a constant, commonly referred to as the residue. In this implementation, the value of the residue is 0xC704DD7B when no CRC errors are detected. [12]

V. FPGA EDITOR ROUTED DESIGN

The FPGA editor is a graphical application for displaying and configuring Field Programmable Gate Arrays(FPGAs). The FPGA editor requires a Native circuit description(.ncd) file. This file contains the logic of your design mapped to components (such as CLBs and IOBs). In addition, the FPGA editor reads from and writes to a Physical constraints file (PCF).

The following is a list of a few functions we can perform on our designs in the FPGA editor.

a) Place and route critical components before running the automatic place and route tools.
b) Finish placement and routing if the routing program does not completely route our design.
c) Add probes to our design to examine the signal states of the targeted device. Probes are used to route the value of internal nets to an IOB for analysis during the debugging of a device.
d) Run the Bitgen program and download the resulting BIT file to the targeted device.
e) View and change the nets connected to the capture units of an ILA core in our design.
f) Use the ILA command to write .cdc file.
g) Create an entire design by hand(Advanced users).

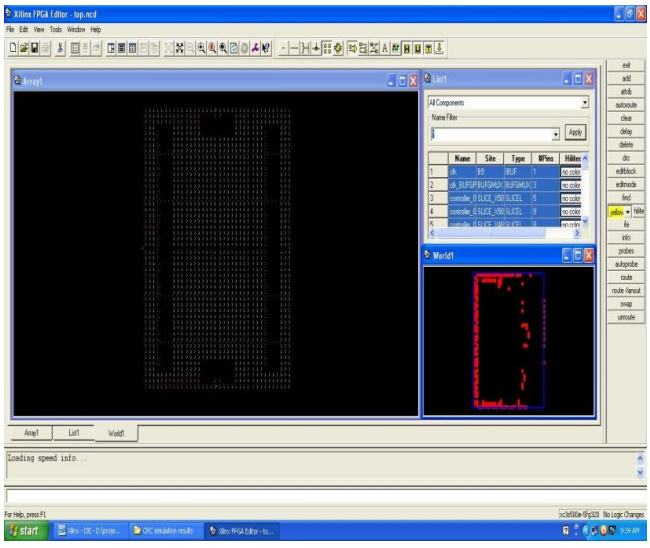

Fig. 8 Floor plan editor for routing of the CRC top module

The Fig. 8 gives the details of the amount of area occupied by the circuit on the device and the amount of area efficiently utilized by the circuit. In this the area is efficiently utilized by placing the pins nearer are moved apart and the pins farther are moved closer to them so that efficient area can be utilized

VI. FUTURE SCOPE

Cyclic Redundancy Check (CRC) is an error-checking code that is widely used in data communication systems and other serial data transmission systems. CRC is based on polynomial manipulations using modulo arithmetic. Some of the common Cyclic Redundancy Check standards are CRC-8, CRC-12, CRC-16, CRC-32, and CRC-CCIT. This application note discusses the implementation of an IEEE 802.3 CRC in a Virtex™ device. The reference design provided with this application note provides Verilog point solutions for CRC-8, CRC-12, CRC-16, and CRC-32. The Perl script (crcgen.pl) used to generate this code is also included. The script generates Verilog source for CRC circuitry of any width (8, 12, 16, 32), any polynomial, and any data input width.

VII. CONCLUSION

CRC gives tradeoff between flexibility, performance and cost has been taken further than those enabled by traditional heterogeneous architectures based on microprocessor, DSP . The application of this method in CRC fields in GFP is presented using FPGAs, which is efficient in both circuit area and speed. The memory address length has been minimized, while keeping a very simple circuit implementation. This enables different CRC polynomials and I/O port and processing data-path widths to be deployed. The architecture is also generic in its design and can be scaled to 64-, 128-, or 256-bits in the data path.

REFERENCES

1. Analysis of an error detecting code in block based transmissionB. RAMYA SREE1, B. MANJULA2, K. MURALI KRISHNA2, B. V. RAMA MOHANA RAO3
2. FPGA Implementation of CRC with Error CorrectionWael M El-Medany
3. VLSI Implementation of Parallel CRC Using Pipelining, Unfolding and Retiming Sangeeta Singh1, S. Sujana2, I. Babu3, K. Latha4
4. Design and Synthesis of a Field Programmable CRC Circuit Architecture K.V.GANESH*,D.SRI HARI**,M.HEMA***
5. CHIPSCOPE Implementation of CRC circuit architecture G.Shanthi1, Dr.L.Padmasree
6. CRC Look-up Table Optimization for Single-Bit Error CorrectionPAN Yun , GE Ning , DONG Zaiwang
7. IEEE 802.3 Cyclic Redundancy Check Author: Chris Borrelli
8. 32-Bit Cyclic Redundancy Codes for Internet Applications Philip Koopman ECE Department & ICES
9. Design and Implementation of a Field Programmable CRC Circuit Architecture Ciaran Toal, Kieran McLaughlin, Sakir Sezer, and Xin Yang
10. Information theory by Ass.Prof.Dr. Thamer
11. References for Xilinx,inc. at www.xilinx.com/support
12. SP006: LocalLink Interface Specification
13. UG189: Virtex-5 FPGA CRC Wizard v1.3 User Guide
14. UG196: Virtex-5 FPGA RocketIO GTP Transceiver User Guide
15. DS100: Virtex-5 Family Overview
16. IEEE 802.3 Cyclic Redundancy Check Author: Chris Borrelli