THE DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A DYNAMICALLY RECONFIGURABLE AND AREA OPTIMIZED CYCLIC REDUNDANCY CHECK ARCHITECTURE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZCAN YURT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2013

Approval of the thesis:

THE DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A DYNAMICALLY RECONFIGURABLE AND AREA OPTIMIZED CYCLIC REDUNDANCY CHECK ARCHITECTURE

submitted by **ÖZCAN YURT** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** ______________

Assoc. Prof. Dr. Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering Dept., METU** ______________

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU ______________

Assoc. Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering Dept., METU ______________

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU ______________

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Dept., METU ______________

Dr. Nizam Ayyıldız
ASELSAN Inc. ______________

Date: 27/08/2013

Name, Last name	: Özcan YURT
Signature	:

ABSTRACT

THE DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A DYNAMICALLY RECONFIGURABLE AND AREA OPTIMIZED CYCLIC REDUNDANCY CHECK ARCHITECTURE

Yurt, Özcan
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Ece Güran Schmidt

August 2013, 62 pages

The Cyclic Redundancy Check (CRC) calculation for data communication protocols is implemented by hardware calculators in several systems due to increasing throughput requirements of data communication protocols. Furthermore CRC is employed in many small scale embedded systems with different types of data communication interfaces that are implemented on FPGA. Resource utilization of these systems is frequently a critical parameter with regards to cost. In many cases, limited logic units of an FPGA have to be used very carefully to fit the design into that platform. In this thesis, we present DAROC-Dynamically Reconfigurable and ARea Optimized CRC, which is a run-time reconfigurable and *area-minimized* CRC calculator. The ability of reconfiguration enables DAROC calculating different CRCs for several standards with a single instance of implementation. DAROC reaches the throughput of 705 Mbps that is sufficient for the target embedded systems with less resource consumption compared to the previous reconfigurable CRC implementations.

Keywords: CRC, Cyclic Redundancy Check, FPGA, dynamic reconfiguration, area optimization.

ÖZ

DİNAMİK OLARAK YENİDEN YAPILANDIRILABİLEN VE ALAN İYİLEŞTİRİLMİŞ BİR DÖNGÜSEL ARTIKLIK DENETİMİ MİMAMİRİSİ GELİŞTİRİLMESİ VE DONANIM GERÇEKLEMESİ

Yurt, Özcan
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Ece Güran Schmidt

Ağustos 2013, 62 sayfa

Veri iletişim protokollerinin hız gereksinimleri arttığı için, bu protokollerde yer alan döngüsel artıklık denetimi (CRC) hesaplaması birçok sistemde donanım tabanlı hesaplayıcılar ile gerçeklenmektedir. Ayrıca, birden fazla iletişim ara yüzüne sahip, FPGA üzerinde gerçeklenmiş, birçok küçük ölçekli gömülü sistemde CRC kullanılmaktadır. Bu sistemlerde donanımsal kaynak kullanımı, maliyet açısından, çoğu zaman kritik bir parametredir. Birçok durumda, tasarımı hedef platform olan FPGA içerisine sığdırabilmek için, o FPGA'e ait kısıtlı mantık birimlerini idareli bir şekilde kullanmak gerekir. Bu tezde, koşum zamanında yeniden yapılandırılabilen ve alan anlamında küçültülmüş bir CRC hesaplayıcı olan DAROC – Dinamik Olarak Yeniden Yapılandırılabilen ve Alan İyileştirilmiş Döngüsel Artıklık Denetimi sunulmaktadır. Yeniden yapılandırılabilme yeteneği, DAROC'un sadece bir örnek gerçeklenmesi ile birçok standart için farklı CRC hesaplama yapabilmesini sağlamaktadır. DAROC, daha önceki yeniden yapılandırılabilir CRC uygulamalarıyla karşılaştırıldığında daha az kaynak kullanımı ile hedef gömülü sistemler için yeterli olan 705 Mbps veri işleme hacmine ulaşmaktadır.

Anahtar Kelimeler: CRC, Döngüsel Artıklık Denetimi, FPGA, dinamik yeniden yapılandırma, alan en iyileştirme.

To My Family,

ACKNOWLEDGEMENTS

I would like to express my special thanks to my supervisor Assoc. Prof. Dr. Ece Güran Schmidt for her guidance, support, encouragement, trust, patience and valuable contributions throughout the preparation of my thesis.

I would like to acknowledge the support of ROKETSAN Inc. for the realization of this thesis.

The last but not the least, I express my sincerest thanks to Muhammet Hamdi Yavuz, Yılmaz Fırat Kaya, Metin Kazkayası, Fatih Çelik, Serkan Öztürk, Çiğdem Türkmendağ, İpek Yağcan, Enes Aykurt, Tamer Uz, Mazhar Gökhan Özkeser, Halil Ertuğrul, my wife Elif and my family who have given me encourage and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	: Application Specific Integrated Circuit
BRAM	: Block Random-Access-Memory
CRC	: Cyclic Redundancy Check
DECT	: Digital Enhanced Cordless Telecommunications
DUT	: Device Under Test
FPGA	: Field Programmable Gate Array
HDL	: Hardware Description Language
iSCSI	: Internet Small Computer System Interface
LFSR	: Linear Feedback Shift Register
LSB	: Least Significant Bit
LUT	: Look Up Table
MSB	: Most Significant Bit
NA	: Not Applicable
NoC	: Network on Chip
SoC	: System on Chip
SPI	: Serial Peripheral Interface
UART	: Universal Asynchronous Receiver/Transmitter
UMC	: United Microelectronics Corporation

CHAPTER 1

INTRODUCTION

Cyclic Redundancy Check (CRC) is an error-detecting code based upon polynomial division. It is widely-used in communication protocols because of the efficiency on detecting transmission errors [1]. It is also used in data storage systems.

Due to increasing throughput requirements of data communication protocols, software implementations of CRC calculation can be inadequate [2]. When speed requirements of the CRC calculation cannot be met with a software implementation, a hardware solution is employed.

Hardware implementation methods of CRC are generally categorized as serial, parallel and table-based [3]. The data is processed one bit per clock cycle in the serial implementation. Parallel implementation method is based on unrolling the serial circuit. So, *n*-bit data is processed per clock cycle where the *n* is parallelization level. Lastly, in table-based implementation, pre-calculated values are read from a table for given input values.

FPGA is a preferred hardware platform for CRC implementations because of their programmability. Resource utilization of these systems is frequently a critical parameter with regards to cost. In many cases, limited logic units of an FPGA required to be used very carefully with the aim of fitting the design into that platform. In such a case, the system may have multiple communication protocols with different CRC standards. If the area utilizations of the CRC calculators are minimized as far as possible in such a system, it can be helpful for that system in terms of fitting into platform. Furthermore, many FPGA-based systems have different types of data communication interfaces and protocols that possibly require different CRC calculations.

Most of the studies about hardware implementation of CRC are based on one-polynomial CRC calculation which calculates CRC only for a specific CRC standard. Although the polynomials of these CRC calculators can be changed easily, this operation has to be done before the run-time.

In this thesis, we focus on run-time reconfigurable CRC calculators where one calculator can be used for all CRC standards that required for the system. In other words, the calculator can be shared in time by communication protocols. The main advantage of this approach is that only one instance of a CRC calculator is in use at a time resulting in reduction of area utilization.

There are previous hardware CRC implementation studies such as [4], [5] and [6] which have the ability of run-time reconfiguration. They proposed a cell-array based parallel CRC calculator in [4]. Cell arrays consist of an XOR gate, two MUX and a register. Similarly, in [5], they proposed a design which consist of XOR and AND gate arrays. A LUT-based reconfigurable approach was proposed in [6]. Although these designs are run-time reconfigurable, they require relatively high area utilization with respect to logic for small-scale systems. These systems may require a reduced calculator in terms of area utilization mitigating the advantages of the reconfigurable CRC design.

In this thesis, we propose DAROC-Dynamically Reconfigurable and ARea Optimized CRC, which is a run-time reconfigurable and *area-minimized* CRC calculator. Due to the ability of reconfiguration, DAROC meets the need of the systems that have to calculate CRC for several standards. Although the throughput is doubled, DAROC requires the same number of logic blocks on FPGA with the serial implementation. Area minimization is achieved by using the minimum number of logic blocks, which are required for CRC implementation, in full capacity.

The proposed design is implemented on Xilinx XC6SLX45T platform which has dynamically reconfigurable blocks. Number of slice register utilization is 32 out of 54576. It utilizes 37 out of 27288 slice LUTs. On the other hand, maximum achieved throughput is 705 Mbps for processing 2-bits at a time with 16 bit polynomial.

The remainder of this thesis organized as follows. CHAPTER 2 introduces the literature overview on CRC and its hardware implementations. Performance metrics of CRC calculation such as resource utilization and throughput are defined. Then, relevant previous works on dynamically reconfigurable CRC calculation in hardware are discussed.

CHAPTER 3 describes the architecture of DAROC-Dynamically Reconfigurable and ARea Optimized CRC and the implementation of DAROC architecture that is constructed with the dynamically reconfigurable Look-Up-Table (LUT) resources on FPGA.

In CHAPTER 4, the simulation and hardware platforms for implementing DAROC are introduced. Simulation and implementation results are presented in terms of performance metrics that we define in CHAPTER 2. Evaluation results are discussed.

Finally, in CHAPTER 5, the conclusion is drawn and potential future directions are listed. The summary of studies and evaluations in this thesis is presented. The implementation results are summarized.

CHAPTER 2

LITERATURE OVERVIEW

2.1 Cyclic Redundancy Check

CRC is used to keep the integrity of data in communication and storage systems [7]. It is a commonly used polynomial division based error detection method.

The basic idea behind the CRC is calculating a check value over a data stream and attaching this check value to the data stream. So, the data integrity can be checked with the calculated value. This check value is called as CRC value. In communication systems, the transmitter calculates the CRC of the data stream which is going to be transmitted and concatenates the data stream and CRC. Then, the transmitter sends the concatenated stream to the receiver. In the receiver side, the CRC of received data stream is calculated. Afterwards, the receiver compares the calculated CRC and received CRC value. If there is a difference between these values, the receiver realizes that an error has been occurred during the transmission. After the error detection, the action to take is determined by the protocol of the communication system. Correspondingly to the communication systems, data validity is checked by CRC code in various data storage systems such as magnetic and optical storages.

The following subsection describes the CRC computation. Then the most common CRC standards are proposed in adjacent subsection.

2.1.1 Computation of CRC

Computation of a Cyclic Redundancy Check is based on polynomial division in finite binary field [7]. In the finite field arithmetic, a binary polynomial is a polynomial that has coefficients in binary field (0 or 1). Binary representations of some polynomials can be seen in Table 2-1. The leftmost bit of a binary sequence represents the highest degree term of the associated polynomial.

Table 2-1 Binary Representations of Polynomials

Polynomial	Polynomial (with coefficients)	Binary Representation
$x^7 + x^4 + x^3 + 1$	$1.x^7 + 0.x^6 + 0.x^5 + 1.x^4 + 1.x^3 + 0.x^2 + 0.x^1 + 1.x^0$	10011001
$x^3 + x^1$	$1.x^3 + 0.x^2 + 1.x^1\ 0.x^0$	1010
$x^4 + x^2 + 1$	$1.x^4 + 0.x^3 + 1.x^2 + 0.x^1 + 1.x^0$	10101
$x^6 + x^5 + x^1$	$1.x^6 + 1.x^5 + 0.x^4 + 0.x^3 + 0.x^2 + 1.x^1 + 0.x^0$	1100010

Computation procedure of CRC is described as follows. The data sequence that we want to calculate the CRC for is represented as a polynomial *P(x).* As an example, *P(x)* is a seventh degree polynomial in (1) that is derived from the binary data sequence "11001001".

$$P(x) = 1 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 \qquad (1)$$

$$P(x) = x^7 + x^6 + x^3 + 1 \qquad (2)$$

P(x) is multiplied by a polynomial x^p where *p* is the degree of a certain polynomial *G(x)* called the *generator polynomial*.

$$\begin{aligned} P(x) \cdot x^p &= (x^7 + x^6 + x^3 + 1) \cdot x^p \qquad (3) \\ &= x^{7+p} + x^{6+p} + x^{3+p} + x^{0+p} \end{aligned}$$

The generator polynomial is used to generate the CRC value of a given data sequence. An example *G(x)* is given in (4).

$$G(x) = x^3 + x^1 + 1 \tag{4}$$

The degree of the polynomial *G(x)* is 3 (*p* = 3). So, *P(x)* is multiplied by x^3.

$$\begin{aligned} P(x) \cdot x^p &= (x^7 + x^6 + x^3 + 1) \cdot x^3 \\ &= x^{10} + x^9 + x^6 + x^3 \end{aligned} \tag{5}$$

So, the new data sequence can be represented in binary format as in (6).

$$\begin{gathered} R(x) = 11001001000 \\ R(x)\ is\ P(x)\ shifted\ by\ p \end{gathered} \tag{6}$$

The derived polynomial is divided by the generator polynomial *G(x)*.

$$\begin{gathered} (x^{10} + x^9 + x^6 + x^3) \div (x^3 + x^1 + 1) = x^7 + x^6 + x^5 + x^3 + x^2 + x^1 + 1 \\ remainder = 1 \end{gathered} \tag{7}$$

The division is shown in Figure 2-1.

$$\begin{array}{l|l}
x^{10} + x^9 + x^6 + x^3 & x^3 + x^1 + 1 \\ \cline{2-2}
x^{10} + x^8 + x^7 & x^7 + x^6 + x^5 + x^3 + x^2 + x^1 + 1 \\ \cline{1-1}
\multicolumn{1}{l}{x^9 + x^8 + x^7 + x^6 + x^3} & \uparrow \\
\multicolumn{1}{l}{x^9 + x^7 + x^6} & \text{quotient} \\ \cline{1-1}
\multicolumn{1}{l}{x^8 + x^3} & \\
\multicolumn{1}{l}{x^8 + x^6 + x^5} & \\ \cline{1-1}
\multicolumn{1}{l}{x^6 + x^5 + x^3} & \\
\multicolumn{1}{l}{x^6 + x^4 + x^3} & \\ \cline{1-1}
\multicolumn{1}{l}{x^5 + x^4} & \\
\multicolumn{1}{l}{x^5 + x^3 + x^2} & \\ \cline{1-1}
\multicolumn{1}{l}{x^4 + x^3 + x^2} & \\
\multicolumn{1}{l}{x^4 + x^2 + x^1} & \\ \cline{1-1}
\multicolumn{1}{l}{x^3 + x^1} & \\
\multicolumn{1}{l}{x^3 + x^1 + 1} & \\ \cline{1-1}
\multicolumn{1}{l}{1} & \longleftarrow \text{remainder}
\end{array}$$

Figure 2-1 Polynomial Division

Finally, the remainder of the division is added to *R(x)* as seen in (8). The remainder "001" is the CRC value of the data sequence "110010011".

$$11001001000 + 001 = 11001001001 \tag{8}$$

So, the derived data sequence is a multiple of the generator polynomial *G(x).* In communication systems, the receiver checks the received data by dividing it to the *G(x).* If the remainder is different than zero, the receiver realizes that an error has occurred during the transmission. Otherwise, the data sequence is passed from CRC. Similarly, in data storage systems, the validity of data which is on a storage device is checked by memory controller using the division procedure of CRC.

There is a possibility that the remainder of division is zero in CRC computation even if an error has been occurred. Such a situation may arise when the data sequence is a multiple of *G(x)* after a corruption.

These undetected errors may occur in a probability that depends on various factors such as the generator polynomial, the degree of the generator polynomial and lengths of data sequences.

Although the possibility exists that the occurrence of undetected errors, CRC is a powerful and simple method for error detection. It can be controlled by suitable choice of the generator polynomial. In addition, there is a final XOR operation in some CRC standards for obfuscation.

2.1.2 CRC Standards

Some commonly used standards are listed in Table 2-2.

Table 2-2 Commonly Used CRCs

Name	Polynomial	Uses
CRC-4-ITU	$x^4 + x^1 + 1$	G.704
CRC-5-EPC	$x^5 + x^3 + 1$	Gen 2 RFID
CRC-5-USB	$x^5 + x^2 + 1$	USB
CRC-7	$x^7 + x^3 + 1$	MMC, SD
CRC-8-CCIT	$x^8 + x^2 + x^1 + 1$	ATM
CRC-8-WCDMA	$x^8 + x^7 + x^4 + x^3 + x^1 + 1$	WCDMA
CRC-11	$x^{11} + x^9 + x^8 + x^7 + x^2 + 1$	FlexRay
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	CAN
CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$	USB, MODBUS, ANSI X3.28
CRC-16-CCIT	$x^{16} + x^{12} + x^5 + 1$	HDLC, XMODEM, Bluetooth
CRC-16-DECT	$x^{16} + x^{10} + x^8 + x^7 + x^3 + 1$	DECT
CRC-16-T10 DIF	$x^{16} + x^{15} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^4 + x^2 + 1$	SCSI DIF
CRC-24	$x^{24} + x^{22} + x^{20} + x^{19} + x^{18} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^8 + x^7 + x^6 + x^3 + x^1 + 1$	FlexRay
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$	Ethernet, SATA, MPEG-2

2.2 Hardware Implementation Techniques for Cyclic Redundancy Check

Hardware implementation techniques for CRC are classified as serial, parallel and table-based. Serial implementation has the lowest cost. On the other hand, the throughputs of parallel and table-based implementations are generally higher than the serial one.

To implement the CRC in hardware, required components are determined depending on the operations in calculation procedure.

In binary field, operations are performed using modulo-2. Truth table of the addition operation which is defined in the binary field is given in Table 2-3. A and B are the binary digits that are added.

Table 2-3 Truth Table of Addition

Inputs		Output
A	B	Sum
0	0	0
0	1	1
1	0	1
1	1	0

Correspondingly, the subtraction operation is defined in Table 2-4. *A* is the minuend and *B* is subtrahend.

Table 2-4 Truth Table of Subtraction

Inputs		Output
A	B	Difference
0	0	0
0	1	1
1	0	1
1	1	0

As seen on truth tables in Table 2-3 and Table 2-4, subtraction in the binary field is the same as addition. So, an exclusive or (XOR) gate can be used for the addition and subtraction operation in the binary field.

In polynomial arithmetic, a division operation can be made as demonstrated in Figure 2-2.

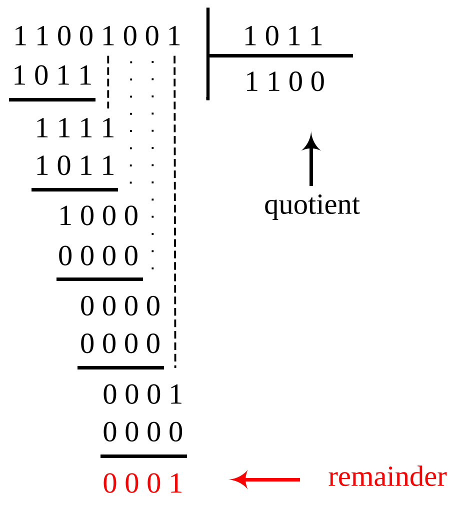


Figure 2-2 Polynomial Division in the Binary Field

The division operation that is shown in Figure 2-2 can be implemented by a shift register and XOR gates [14]. Shift register is used for shifting the dividend and XOR gates are used for subtraction operations.

In the following subsections, hardware implementation techniques are described briefly.

2.2.1 Serial Implementation

Serial implementation is the simplest approach in the hardware implementation methods of CRC calculation. Traditionally, a simple CRC architecture is based on a linear feedback shift register (LFSR) [8]. A typical LFSR based serial CRC architecture is shown in Figure 2-3.

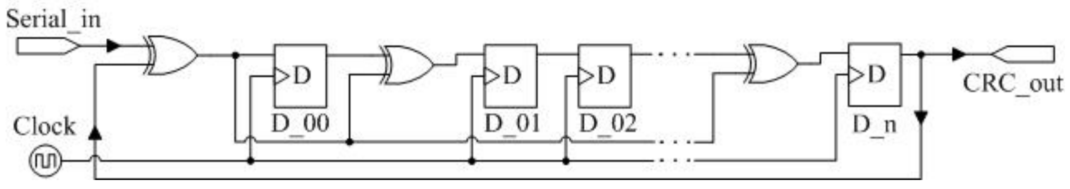


Figure 2-3 A LFSR-based Serial CRC Circuit

The data is processed one bit per clock cycle in the serial circuit depicted in Figure 2-3. The serial input of the circuit is marked as `Serial_in`. The data which we wanted to compute the CRC for are provided from `Serial_in`. The output of the circuit is the `CRC_out` which gives the CRC result of computation. At each clock pulse, some bits are shifted directly and the others are shifted after XOR operation. The bits to be XOR operated are determined by the used CRC polynomial. The computation takes as many cycles as the number of bits in the data + *n* bits shift where the *n* is the length of *the generator polynomial.*

2.2.2 Parallel Implementation

A serial LFSR circuit sometimes can be inadequate with regards to throughput. This limitation led to various other studies that focused on parallel implementations. Generally, parallelization is recommended in the cases where the data transfer is parallel or transfer rates are very high [9].

The parallelization method of CRC implementation is unrolling the serial circuit [10]. In terms of combinational logic, parallel implementation requires more elements and area than the serial one. In Figure 2-4, a 2-bit parallel CRC calculator is shown.

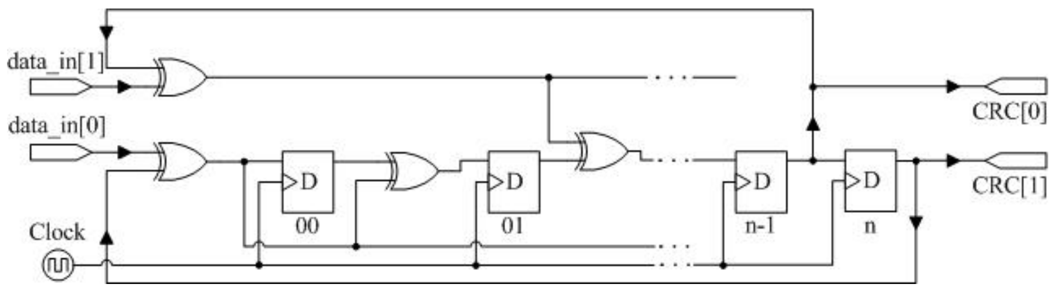


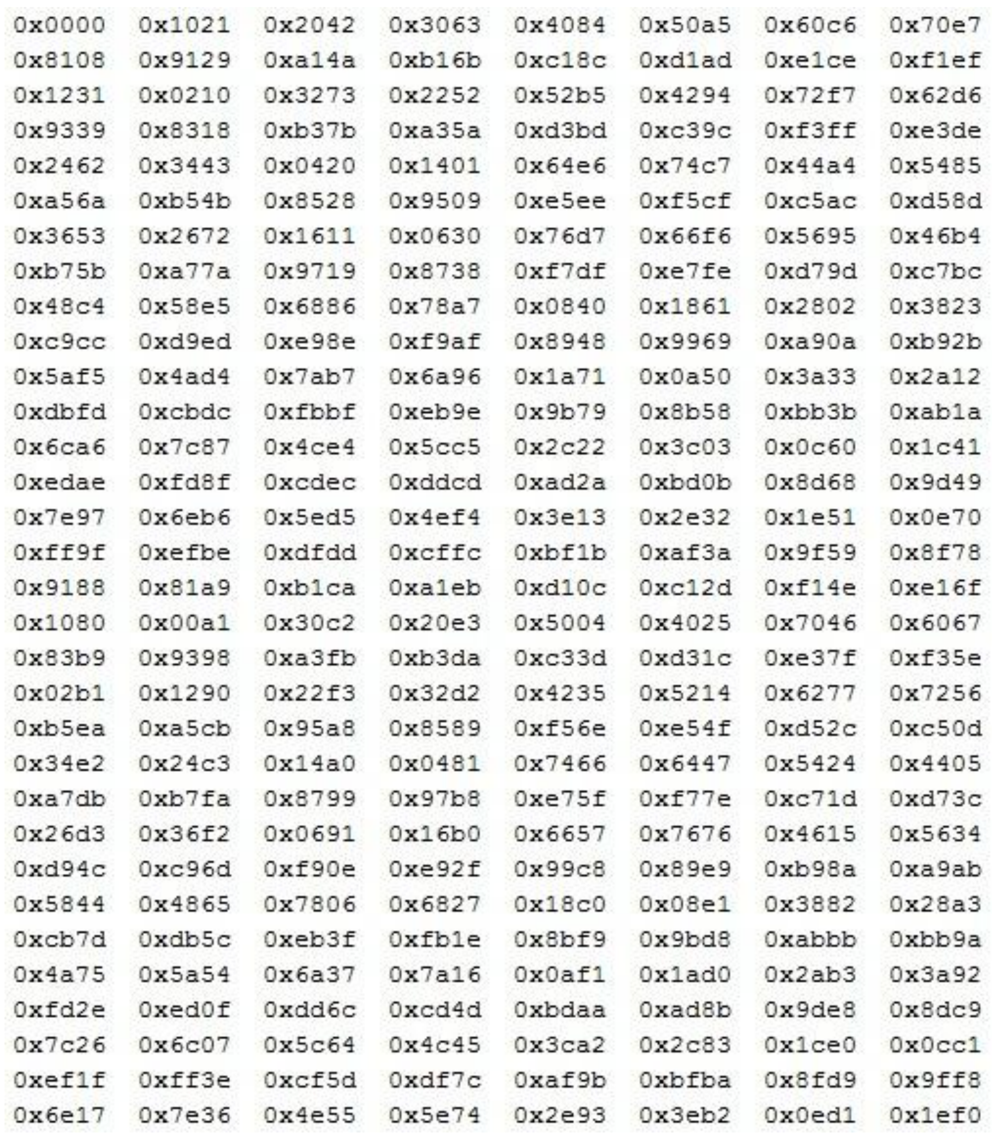
Figure 2-4 A Parallel (2-bits) CRC Circuit

In Figure 2-4, the parallel circuit implements CRC calculation where `data_in`[1:0] and `CRC`[1:0] are the inputs and outputs of the circuit respectively. In each cycle, 2-bits of CRC is computed for a given input data sequence. So, the 2-bit parallel implementation is almost 2 times faster than the serial one. Logic utilization of parallel implementation is more than the serial one but the rate of increase is due to the polynomial.

2.2.3 Table-Based Implementation

Lastly, table-based CRC implementation is another alternative for hardware-based CRC computation when the data rates are considerably high. However, table-based implementations may be too expensive for small-scale systems when the table-sizes are considerably high.

For all of the possible input data bytes, the CRC values can be pre-computed and stored in a table. In this manner, a table with 256-entries is required. In Figure 2-5, a LUT’s content is shown. This LUT is generated for CRC-16 by using the generator polynomial “0x1021”. Each entry has 16-bit values corresponds to the input byte value.



0x0000	0x1021	0x2042	0x3063	0x4084	0x50a5	0x60c6	0x70e7
0x8108	0x9129	0xa14a	0xb16b	0xc18c	0xd1ad	0xe1ce	0xf1ef
0x1231	0x0210	0x3273	0x2252	0x52b5	0x4294	0x72f7	0x62d6
0x9339	0x8318	0xb37b	0xa35a	0xd3bd	0xc39c	0xf3ff	0xe3de
0x2462	0x3443	0x0420	0x1401	0x64e6	0x74c7	0x44a4	0x5485
0xa56a	0xb54b	0x8528	0x9509	0xe5ee	0xf5cf	0xc5ac	0xd58d
0x3653	0x2672	0x1611	0x0630	0x76d7	0x66f6	0x5695	0x46b4
0xb75b	0xa77a	0x9719	0x8738	0xf7df	0xe7fe	0xd79d	0xc7bc
0x48c4	0x58e5	0x6886	0x78a7	0x0840	0x1861	0x2802	0x3823
0xc9cc	0xd9ed	0xe98e	0xf9af	0x8948	0x9969	0xa90a	0xb92b
0x5af5	0x4ad4	0x7ab7	0x6a96	0x1a71	0x0a50	0x3a33	0x2a12
0xdbfd	0xcbdc	0xfbbf	0xeb9e	0x9b79	0x8b58	0xbb3b	0xab1a
0x6ca6	0x7c87	0x4ce4	0x5cc5	0x2c22	0x3c03	0x0c60	0x1c41
0xedae	0xfd8f	0xcdec	0xddcd	0xad2a	0xbd0b	0x8d68	0x9d49
0x7e97	0x6eb6	0x5ed5	0x4ef4	0x3e13	0x2e32	0x1e51	0x0e70
0xff9f	0xefbe	0xdfdd	0xcffc	0xbf1b	0xaf3a	0x9f59	0x8f78
0x9188	0x81a9	0xb1ca	0xa1eb	0xd10c	0xc12d	0xf14e	0xe16f
0x1080	0x00a1	0x30c2	0x20e3	0x5004	0x4025	0x7046	0x6067
0x83b9	0x9398	0xa3fb	0xb3da	0xc33d	0xd31c	0xe37f	0xf35e
0x02b1	0x1290	0x22f3	0x32d2	0x4235	0x5214	0x6277	0x7256
0xb5ea	0xa5cb	0x95a8	0x8589	0xf56e	0xe54f	0xd52c	0xc50d
0x34e2	0x24c3	0x14a0	0x0481	0x7466	0x6447	0x5424	0x4405
0xa7db	0xb7fa	0x8799	0x97b8	0xe75f	0xf77e	0xc71d	0xd73c
0x26d3	0x36f2	0x0691	0x16b0	0x6657	0x7676	0x4615	0x5634
0xd94c	0xc96d	0xf90e	0xe92f	0x99c8	0x89e9	0xb98a	0xa9ab
0x5844	0x4865	0x7806	0x6827	0x18c0	0x08e1	0x3882	0x28a3
0xcb7d	0xdb5c	0xeb3f	0xfb1e	0x8bf9	0x9bd8	0xabbb	0xbb9a
0x4a75	0x5a54	0x6a37	0x7a16	0x0af1	0x1ad0	0x2ab3	0x3a92
0xfd2e	0xed0f	0xdd6c	0xcd4d	0xbdaa	0xad8b	0x9de8	0x8dc9
0x7c26	0x6c07	0x5c64	0x4c45	0x3ca2	0x2c83	0x1ce0	0x0cc1
0xef1f	0xff3e	0xcf5d	0xdf7c	0xaf9b	0xbfba	0x8fd9	0x9ff8
0x6e17	0x7e36	0x4e55	0x5e74	0x2e93	0x3eb2	0x0ed1	0x1ef0

Figure 2-5 Look-Up Table of CRC-16 (0x1021 Polynomial)

In Figure 2-5, the values in the table are generated for input data ranging from 0 to 255 and increasing first by column and then by row. For this byte-wise table, a 2048-bit memory element is required. Computation of CRC is occurred in a manner that one byte input data is processed in one computation cycle.

The memory requirement for CRC table is can be reduced by decreasing the parallelization level. In other words, if the input length of table is shortened, the size of the table is reduced.

A nibble-wise approach reduces the memory requirement of table from 2048-bits to 256-bits. However, decreasing the input data size reduces the throughput. In nibble-wise approach, 4-bit data is processed in a computation cycle. So, the size of table can be chosen due to the memory and area limitations of the platform which the CRC calculator is implemented on it.

2.3 Performance Metrics

All of the proposed hardware implementation techniques have their own advantages and disadvantages in terms of some specific performance metrics. These metrics are namely resource utilization, throughput, polynomial length and reconfiguration time.

2.3.1 Resource Utilization

Resource utilization is usage of the hardware elements on a platform for implementing the design. Resource types can vary due to implementation technique and the hardware platform. The basic elements of a CRC calculator are registers, XOR gates and memory blocks. In addition, the number of used unit logic blocks on an FPGA is also the resource utilization parameter.

2.3.2 Throughput

The throughput of a CRC calculator is the maximum processed bits of input data in unit time. In other words, the maximum length of the calculated data sequence in unit time is the throughput of the CRC calculator. The throughput requirement of CRC calculator is determined by the communication protocols in data communication systems. In [13], CRC has been described as the biggest bottleneck in iSCSI protocol processing. So, the throughput is an important performance metric for various communication protocols.

In a similar way, several data storage systems require different access speeds on their interfaces. Therefore, the throughput requirement of the CRC calculator in a data storage system is adjusted by the interface speed of that system.

2.3.3 Polynomial Length

Since the CRC computation is based-on polynomial division, the length of the polynomial is an important parameter. The resource utilization of a calculator varies by the polynomial length. Also, the throughput is affected by the polynomial length indirectly. For example, the polynomial length determines the number of required registers in a serial implementation. Accordingly, the throughput of the calculator changes by the length of logic paths.

The polynomial length is determined by the CRC standards as mention before.

2.3.4 Reconfiguration Time

The dynamically reconfigurable CRC calculators have another parameter which is called reconfiguration time. It is the time of reconfiguring the polynomial of the calculator for different CRC standards.

As an example, a system which has multiple communication interfaces may use only one instance of a dynamically reconfigurable CRC calculator. To communicate over an interface, the system has to configure the calculator for that interface. Then, the system may communicate over another interface that has another CRC standard. Herein, a run-time reconfiguration of the calculator is required for communication over the new interface. After a time period which is named as the reconfiguration time the calculator can be used for new interface.

2.4 Previous Work on Dynamically Reconfigurable CRC Implementations

In this section, we present previous studies on dynamically reconfigurable CRC implementations on hardware. The architectures of these designs and some important properties of them are described.

In [4], a 32-bit parallel field programmable CRC implementation is proposed. The design was implemented on ASIC technology using 130-nm UMC standard cell. They proposed a cell-array based architecture which is run-time programmable. There are two main blocks in the architecture which is shown in Figure 2-6. First block is the calculator of the design and the second one is the *configuration circuitry* which is responsible for configuring the calculator.

The *configuration circuitry* of this design has a microprocessor interface. The polynomial of the CRC calculator can be changed over this interface by a microprocessor. The *matrix computation* block of the *configuration circuitry* computes the cell-array values of the calculator. The configuration takes place by writing the calculated values to the CRC array cells. After the configuration, the calculator can be used for CRC computation with the new polynomial.

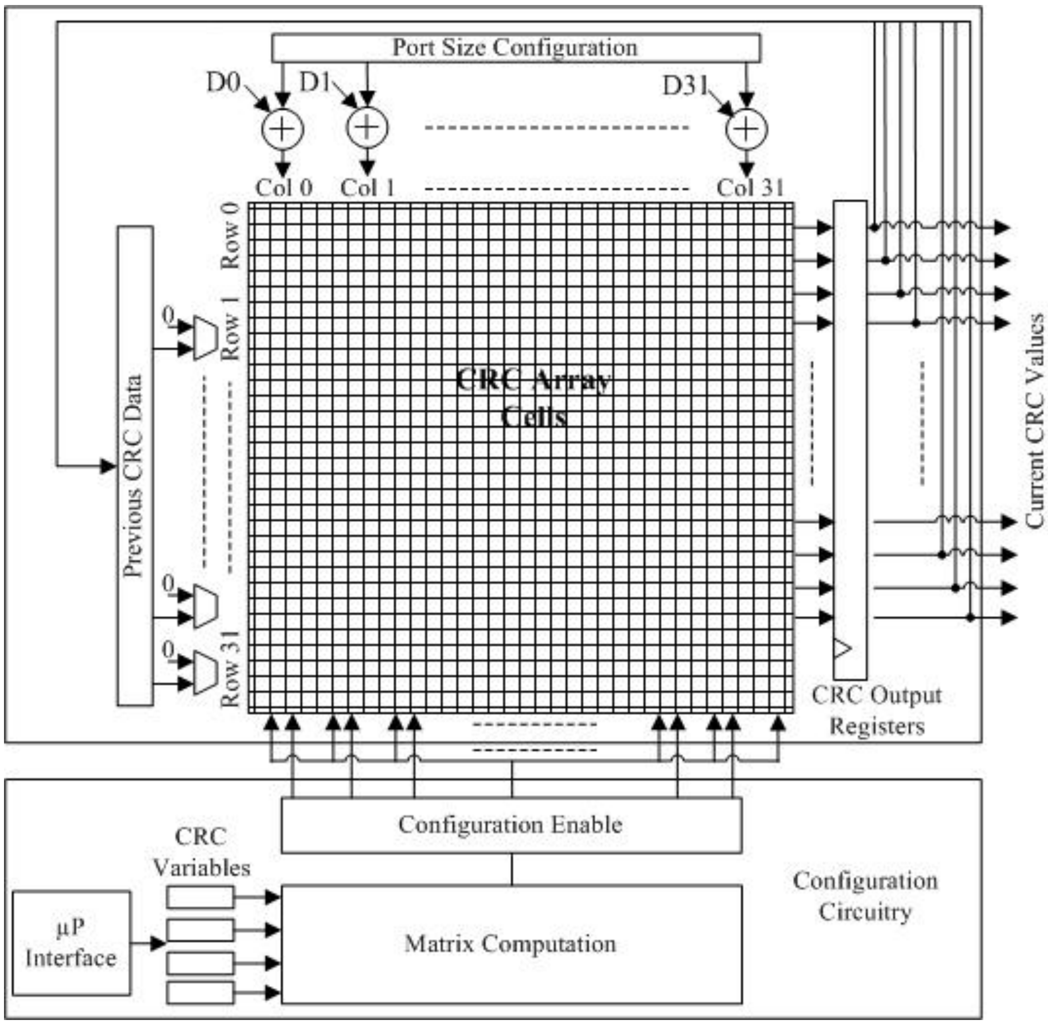


Figure 2-6 The Architecture of Field Programmable CRC Design (adapted from [4])

The reconfigurable block of the architecture is the *programmable CRC array cell* which is shown in Figure 2-7. This cell consists of two parts which are the *data-path* and the *control-path*. The *control-path* has a configuration register that is used for selecting the *data-path* function. The configuration register is configured by the `Config Data` when the `Config Enable` input goes high. The data-path can be configured in two different ways. The `Input 1` or the result of the XOR operation between two inputs is derived to the `Output`.

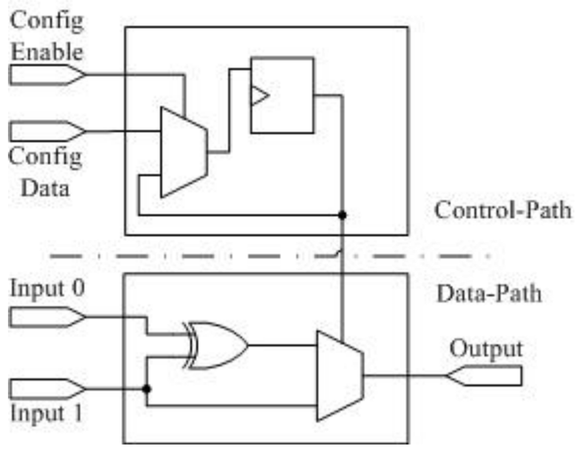


Figure 2-7 Programmable CRC Array Cell (adapted from [4])

The *programmable CRC array cell*s in the architecture which proposed in [4] provide the ability of reconfiguration. The main approach of this design is locating XOR gates for all possible combinations of input paths and choosing the required ones for the selected polynomial. The disadvantage of this design is that the critical path and the number of multiplexers increase rapidly in proportion to the parallelization level.

Another hardware-based CRC architecture is presented in [5]. This programmable parallel CRC circuit was implemented on ASIC with the 130-nm standard cell technology as so the previous study in [4].

There is a similar approach in study [5]. First difference between these architectures is the basic components of the logics. In [5], their preference is using the XOR and AND gates with latches instead of XOR gates, multiplexers and registers. The basic logic diagram for computing a CRC bit in this study is shown in Figure 2-8. This general diagram describes the outline of their study where the c_x is CRC bits, the d_x is the data input and f_x is the latches used for polynomial select.

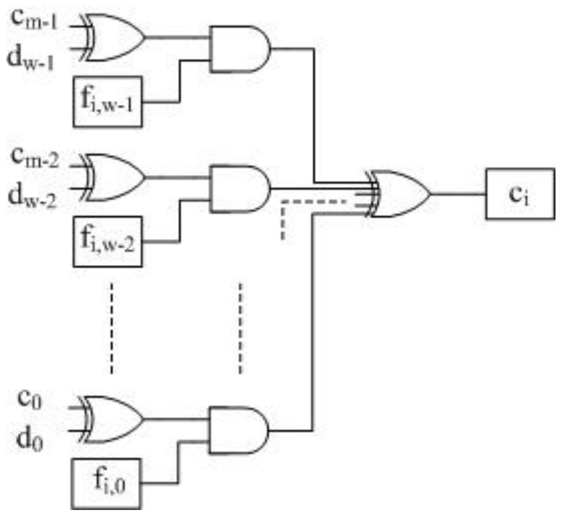


Figure 2-8 Programmable Parallel CRC Circuit for CRC bit c_i (adapted from [5])

Programmable parts of the design are the latches which they controls the XOR inputs by driving the AND gates. Similarly the approach in [4], the latch values can be reprogrammed at run-time. So, the polynomial of CRC circuit is dynamically reconfigurable.

Second improvement of this study is using a XNOR and a NAND gate instead of a latch in the logic. The purpose of this update is decreasing the area utilization and the configuration time. Due to utilization of XNOR and NAND gates in ASIC standard cells, there is approximately %6 area saving in comparison to a latch [5]. In addition, there is no need for calculating the latch values in reconfiguration process.

Lastly, a table-based approach is presented in [6]. Different than the previous run-time reconfigurable studies, the table-based implementation technique is used on FPGA platform which is named Xilinx Virtex-6 LX550T. There are two main blocks in the architecture which are the *Table Generation Module* and the *CRC Module*. A general block diagram of this design is shown in Figure 2-9.

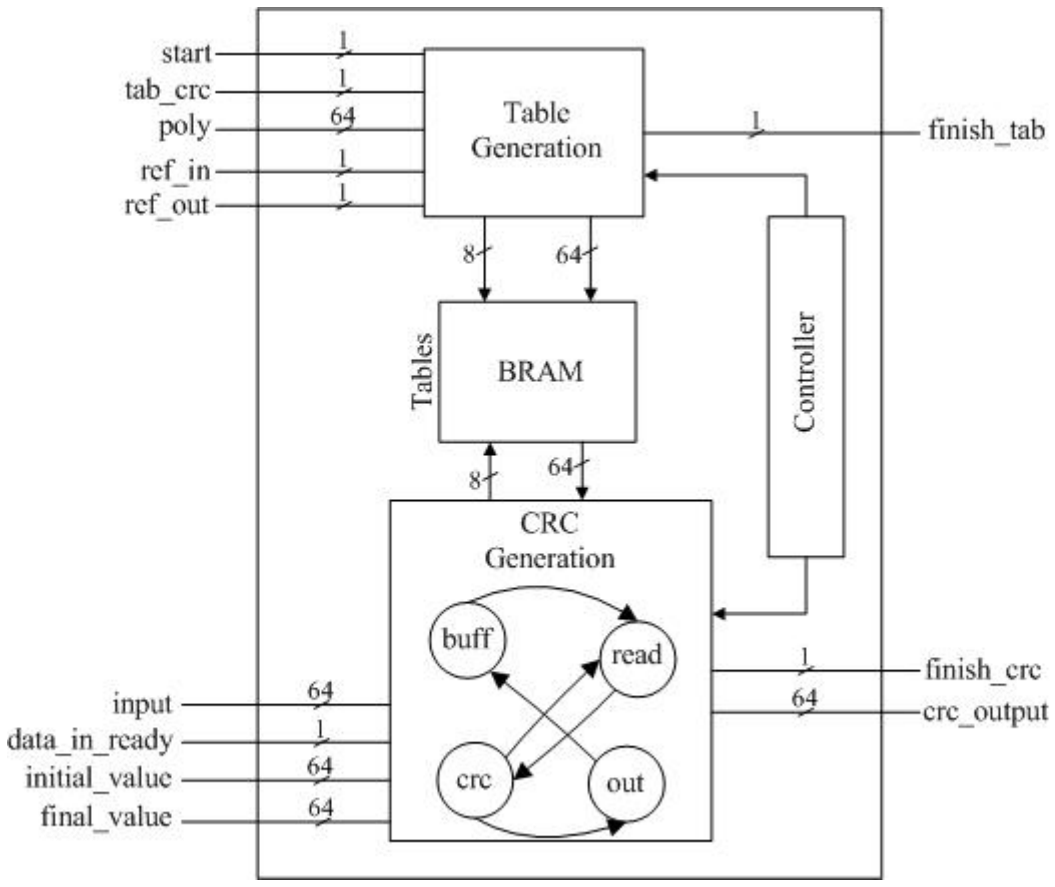


Figure 2-9 General Block Diagram of Table-based CRC (adapted from [6])

Table Generation Module is the module that generates the pre-computed CRC values for given polynomial through the `poly` input. While it is generating the CRC values, it stores these computed values to the tables which consist of the BRAMs or LUTs. After the completion of table generation process, the *CRC Generation Module* can compute the CRC values for given input data frames from the `input` port.

The design proposed in [6] can be extended in terms of polynomial length and the parallel input data length. However, these extensions are only possible at the synthesizing phase before the implementation. The architecture relatively requires a huge number of resources in terms of BRAM and unit logic block. For example, 64-bit parallel CRC generation requires 3398 Slice LUTs and 288Kb BRAM for BRAM-based implementation and 5571 Slice LUTs for logic-based implementation. The throughput is high with a cost of high resource utilization. Increasing the throughput is the main goal of this study.

We observe that, there is generally a tradeoff between the resource utilization and the throughput. So, one of the designs may have the maximum throughput, but at the same time, the resource utilization of it may be the highest. The mentioned run-time reconfigurable designs have some advantages with respect to each other. On the one hand, some of them have higher parallelization level than the others. On the other hand, some of them are more flexible than others in terms of input data width selection. However, their common motivation is that achieving very high throughput by increasing the parallelization level. The mentioned designs are compared in Table 2-5.

Table 2-5 Implementation Comparison

	Cell Array [4]	**Programmable Parallel [5]**	**Table-based (logic) [6]**
Platform	ASIC - 130-nm standard cell technology	ASIC - 130-nm standard cell technology	FPGA – Xilinx Virtex 6 LX550T
Core Area Utilization (mm^2)	0.150	0.033	NA
Slice LUTs Utilization	NA	NA	5571
Parallelization Level	32	32	64
Clock Frequency (MHz)	154	481	443.9
Throughput (Mbps)	4920	15380	28410
Throughput (Mbps) / Slice LUTs	NA	NA	5.1
Reconfiguration Time	33 clock cycle 214 ns	4 clock cycle 8 ns	320 clock cycle 720 ns

CHAPTER 3

DAROC: DYNAMICALLY RECONFIGURABLE AND AREA OPTIMIZED CRC ARCHITECTURE AND FPGA IMPLEMENTATION

3.1 DAROC Architecture

DAROC is an area-optimized CRC architecture that is designed to perform calculations for all 16-bit CRC standards. Its CRC standard is dynamically reconfigurable. In other words, the CRC polynomial, the initial value and the final XOR value of the CRC calculator can be reprogrammed at run-time through the configuration ports. A general block diagram of *DAROC* is depicted in Figure 3-1.

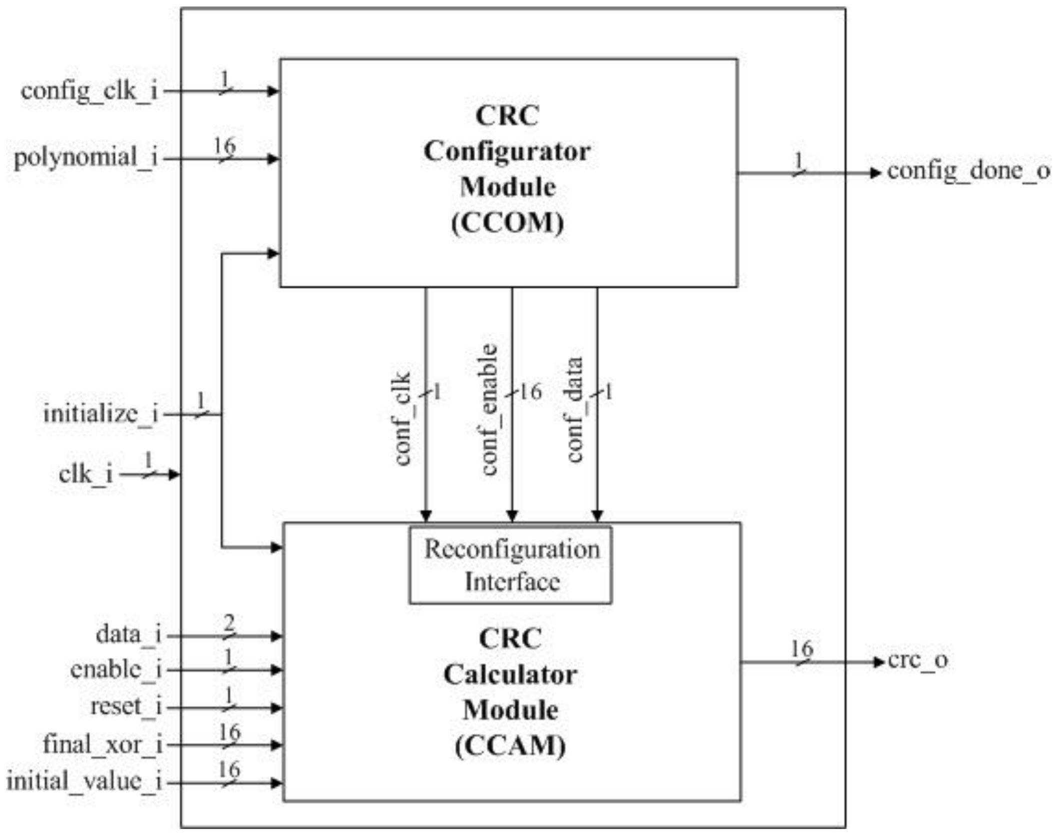


Figure 3-1 General Block Diagram of DAROC

The main component of *DAROC* is the *CRC Calculator Module* (*CCAM*). *CCAM* is responsible for calculating CRC values of input data which is driven to it. *CCAM* can be configured by the *CRC Configurator Module* (*CCOM*) to change the parameters of calculation such as polynomial and initial value of CRC. The inputs and outputs of the DAROC's top module are defined in Table 3-1. The internal signals are defined in the following subsections.

Table 3-1 DAROC Inputs and Outputs

Port	Width	Type	Function
`config_clk_i`	1	input	Configuration clock
`polynomial_i`	16	input	Polynomial input for reconfiguring the calculator with driven polynomial value
`initialize_i`	1	input	1 : For reconfiguring the calculator with the values at configuration ports, 0 : Otherwise
`clk_i`	1	input	System clock signal
`data_i`	2	input	Data input of CRC calculation
`enable_i`	1	input	1 : For data ready notification, 0 : Otherwise
`reset_i`	1	input	1 : For reset request, 0 : Otherwise
`final_xor_i`	16	input	CRC result is XOR operated with this value
`initial_value_i`	16	input	Initial value of the CRC result
`config_done_o`	1	output	1 : Configuration is completed, 0 : Otherwise
`crc_o`	16	output	CRC calculation result

CRC computation process is started when the `enable_i` input goes to high together with the valid input data at `data_i` input. *CCAM* computes the CRC whenever the `enable_i` input is at high. The result of the computation can be read from the `crc_o` output at any moment. If a new computation is requested, the `reset_i` input have to be driven to high. So, the CRC computation process starts with the initial value of CRC.

Reconfiguration process is started when the `initialize_i` input goes to high. At this moment, the configuration parameters are read from the configuration input ports which are `the polynomial_i`, the `final_xor_i` and the `initial_value_i`.

CRC computation and the configuration processes are presented by a general flowchart which is showed in Figure 3-2.

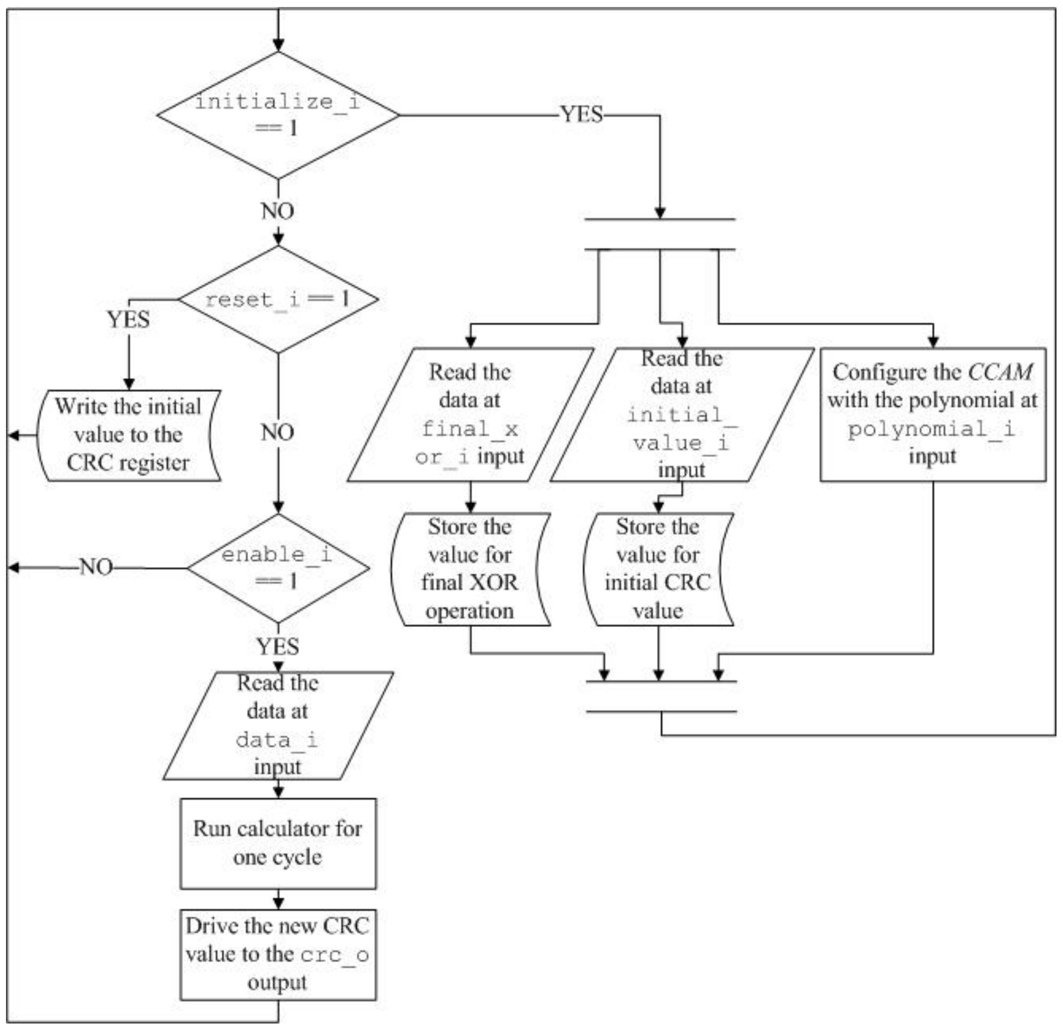

initialize_i == 1
YES
NO
reset_i == 1
YES
Write the initial value to the CRC register
NO
enable_i == 1
NO
YES
Read the data at data_i input
Run calculator for one cycle
Drive the new CRC value to the crc_o output
Read the data at final_xor_i input
Read the data at initial_value_i input
Configure the CCAM with the polynomial at polynomial_i input
Store the value for final XOR operation
Store the value for initial CRC value

Figure 3-2 Flowchart of DAROC

3.1.1 CRC Calculator Module (CCAM)

In this section, CCAM architecture is defined in detail. CRC computation process is presented step by step. Furthermore, the configuration interface is described.

By extending a basic serial LFSR-based one-polynomial CRC calculator, a generalized architecture can be built so that the calculator can be used with various CRC polynomials. The architecture proposed in the Serial Implementation section can be transformed into a reconfigurable structure that does not bound to a specific CRC standard.

The reconfiguration ability is acquired by adding a multiplexer and a XOR gate between all the consecutive registers which keep the CRC bit values. The first input of the multiplexers is the preceding CRC bit value. Second one is the XOR operation result of the preceding CRC bit value and the input of first CRC bit register. Due to this modification, any polynomials can be implemented for the CRC computation by selecting the bits to be XOR operated. In Figure 3-3, the modified architecture which is dynamically reconfigurable is shown.

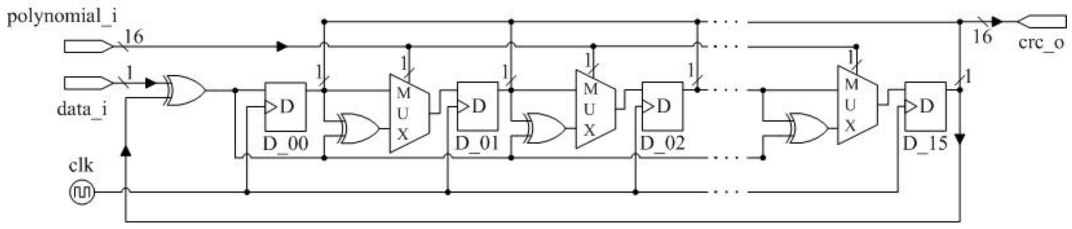


Figure 3-3 Serial Reconfigurable CRC Circuit with Multiplexers

The combinational logic blocks consist of XOR gates and multiplexers. According to the binary representation of the polynomial (`polynomial_i` input in Figure 3-3), multiplexers select the bits that are to be XOR operated. This makes the CRC circuit configurable for any CRC polynomial which has 16-bits length.

To illustrate the polynomial selection, a CRC calculator for 4-bits polynomial is shown in Figure 3-4. The polynomial *G(x)* that specified in equation in (9) is driven to the `polynomial_i` input of the circuit.

$$G(x) = x^4 + x^1 + 1 \tag{9}$$

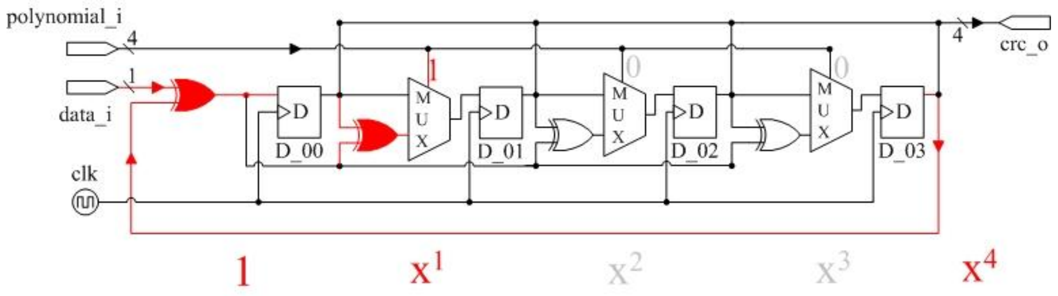


Figure 3-4 Serial Reconfigurable CRC Circuit for 4-bits Polynomials

The serial implementation of CRC processes the incoming data by one bit per clock cycle. To increment the throughput of CRC computation circuit, parallelization can be applied. As mentioned before, the parallelization method of CRC implementation is unrolling the serial circuit. This unrolling method is illustrated in Figure 3-5 by using a commonly used polynomial “0x1021”.

Polynomial (hexadecimal) : 0x1021

Polynomial (binary) : 0001'0000'0010'0001

STEP-1

1	next_1_crc_o[0]	=	crc_o[15] **XOR** data_i[0]
0	next_1_crc_o[1]	=	crc_o[0]
0	next_1_crc_o[2]	=	crc_o[1]
0	next_1_crc_o[3]	=	crc_o[2]
0	next_1_crc_o[4]	=	crc_o[3]
1	next_1_crc_o[5]	=	crc_o[4] **XOR** crc_o[15] **XOR** data_i[0]
0	next_1_crc_o[6]	=	crc_o[5]
0	next_1_crc_o[7]	=	crc_o[6]
0	next_1_crc_o[8]	=	crc_o[7]
0	next_1_crc_o[9]	=	crc_o[8]
0	next_1_crc_o[10]	=	crc_o[9]
0	next_1_crc_o[11]	=	crc_o[10]
1	next_1_crc_o[12]	=	crc_o[11] **XOR** crc_o[15] **XOR** data_i[0]
0	next_1_crc_o[13]	=	crc_o[12]
0	next_1_crc_o[14]	=	crc_o[13]
0	next_1_crc_o[15]	=	crc_o[14]

$\downarrow$

STEP-2

next_2_crc_o[0]	=	next_1_crc_o[15] **XOR** data_i[1]
next_2_crc_o[1]	=	next_1_crc_o[0]
next_2_crc_o[2]	=	next_1_crc_o[1]
next_2_crc_o[3]	=	next_1_crc_o[2]
next_2_crc_o[4]	=	next_1_crc_o[3]
next_2_crc_o[5]	=	next_1_crc_o[4] **XOR** next_1_crc_o[15] **XOR** data_i[1]
next_2_crc_o[6]	=	next_1_crc_o[5]
next_2_crc_o[7]	=	next_1_crc_o[6]
next_2_crc_o[8]	=	next_1_crc_o[7]
next_2_crc_o[9]	=	next_1_crc_o[8]
next_2_crc_o[10]	=	next_1_crc_o[9]
next_2_crc_o[11]	=	next_1_crc_o[10]
next_2_crc_o[12]	=	next_1_crc_o[11] **XOR** next_1_crc_o[15] **XOR** data_i[1]
next_2_crc_o[13]	=	next_1_crc_o[12]
next_2_crc_o[14]	=	next_1_crc_o[13]
next_2_crc_o[15]	=	next_1_crc_o[14]

$\downarrow$ (put the right-handside values of next_1_crc_o at step-1 into equations at step-2)

STEP-3

next_2_crc_o[0]	=	crc_o[14] **XOR** data_i[1]
next_2_crc_o[1]	=	crc_o[15] **XOR** data_i[0]
next_2_crc_o[2]	=	crc_o[0]
next_2_crc_o[3]	=	crc_o[1]
next_2_crc_o[4]	=	crc_o[2]
next_2_crc_o[5]	=	crc_o[3] **XOR** crc_o[14] **XOR** data_i[1]
next_2_crc_o[6]	=	crc_o[4] **XOR** crc_o[15] **XOR** data_i[0]
next_2_crc_o[7]	=	crc_o[5]
next_2_crc_o[8]	=	crc_o[6]
next_2_crc_o[9]	=	crc_o[7]
next_2_crc_o[10]	=	crc_o[8]
next_2_crc_o[11]	=	crc_o[9]
next_2_crc_o[12]	=	crc_o[10] **XOR** crc_o[14] **XOR** data_i[1]
next_2_crc_o[13]	=	crc_o[11] **XOR** crc_o[15] **XOR** data_i[0]
next_2_crc_o[14]	=	crc_o[12]
next_2_crc_o[15]	=	crc_o[13]

Figure 3-5 Unrolling the Serial CRC Circuit to Achieve Parallelization

The parallelization method that is illustrated in Figure 3-5 can be applied for all polynomials. The polynomial “0x1021” which has 16-bits length is used in this illustration. At the first step, the `next_1_crc_o` which is the CRC result for the first bit of input data sequence is computed. Then, the computation of CRC for the second bit of input data sequence is given at the second step. The `next_2_crc_o` is the CRC result for two bits of data. At the last step, the equation of 2-bits parallel computation is represented by putting the right hand-side equivalent of the `next_1_crc_o` to equalization in step-2.

By this method, the dynamically reconfigurable CRC calculator which is shown in Figure 3-3 can be upgraded from serial to parallel. A 2-bits parallel and dynamically reconfigurable CRC calculator which is achieved by applying the proposed unrolling method is shown in Figure 3-6. The proposed multiplexer-based design is implemented on Xilinx XC6SLX45T platform. Number of slice register utilization is 33 out of 54576. It utilizes 63 out of 27288 slice LUTs.

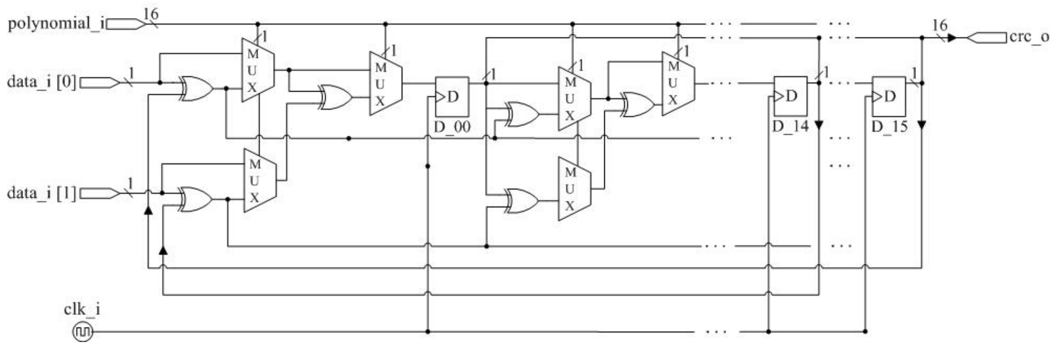


Figure 3-6 Parallel (2-bits) Reconfigurable CRC Circuit with Multiplexers

The CRC calculator which is shown in Figure 3-6 can be used for all the polynomials which have 16-bits length. There are two main parts in each stage of this circuit which are the combinational logic part and a register part. We can say the programmable part is the combinational part. If we make an abstraction on the programmable part, the combinational part of the circuit can be assumed as a configurable block. So, the CRC calculator consists of configurable blocks and registers.

The configurable blocks are assumed as LUT blocks considering the target platform of this study is FPGA, because the combinational logic operations are implemented by using programmable LUTs on FPGAs. The design transforms into a LUT-based entity. A high-level view of this 2-bits parallel calculator is shown in Figure 3-7.

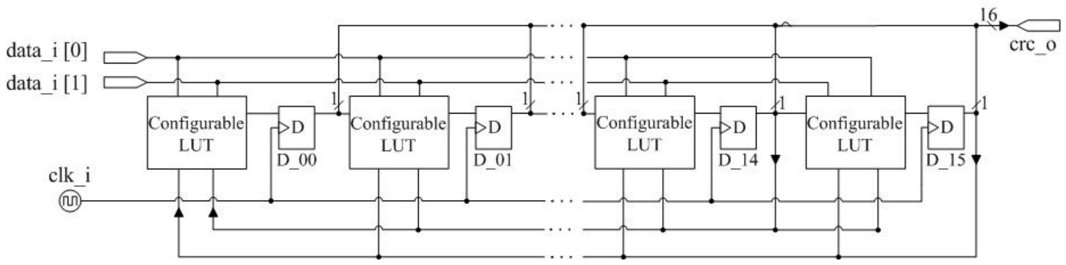


Figure 3-7 Parallel Reconfigurable CRC Circuit with Configurable LUTs

In Figure 3-7, the modified version of the previous 2-bits parallel and dynamically reconfigurable CRC circuit is shown. Polynomial selection is made by configuring the *Configurable LUT blocks* in proposed architecture. This generalized CRC calculator architecture in Figure 3-7 can be implemented with LUT resources on an FPGA. If these LUT blocks of the FPGA are dynamically reconfigurable, the architecture allows run-time switching the CRC polynomial for different CRC standards. To that end, this thesis proposes a generalized 2-bit parallel architecture for all CRC polynomials of a given length. A structural view of this architecture which is named *CCAM* is depicted in Figure 3-8.

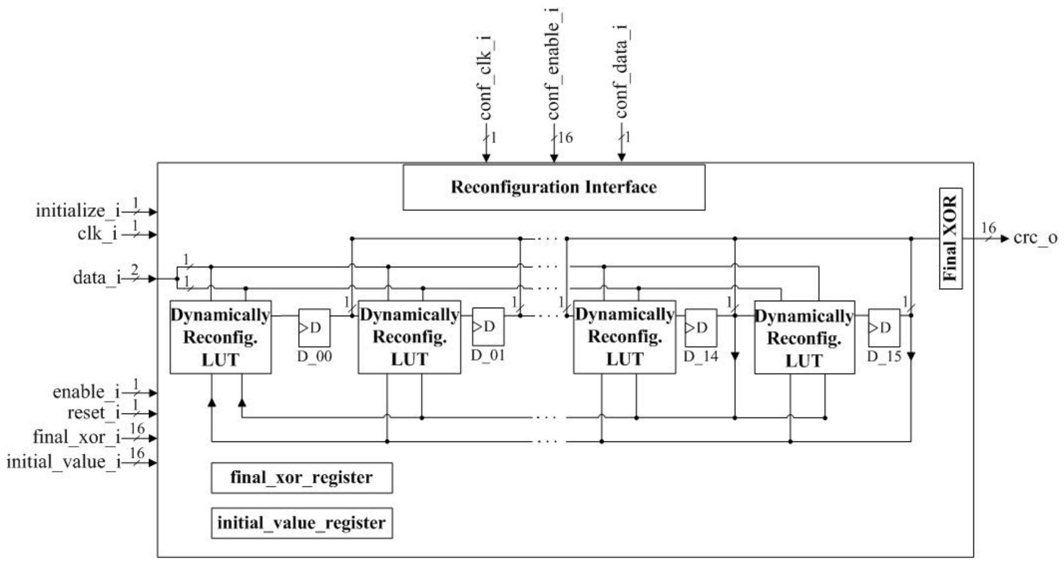


Figure 3-8 Architecture of CRC Calculator Module (CCAM)

CCAM implements CRC calculation where `data_i` and `crc_o` are the input and output of the *CCAM*, respectively. To change the polynomial of the *CCAM*, dynamically reconfigurable LUTs' contents can be replaced through the *reconfiguration interface* of the *CCAM*.

Before a new CRC computation start, the initialization process have to be executed unless it has not already be done for intended CRC standard. The initialization process starts by driving the initial value of CRC to the `initial_value_i` input and the value, which is used in final XOR operation, to the `final_xor_i` input. These values are customized by the CRC standards. Then, the `initialize_i` input is driven to high for registering these values. The reconfiguration takes place in this initialization process. The dynamically reconfigurable LUTs are updated through the reconfiguration interface of the *CCAM*. This interface has three inputs which are the `config_clk_i`, the `config_enable_i` and the `config_data_i`. The configuration data input ports of all the reconfigurable LUTs are combined with the purpose of simplifying the configuration interface and decreasing the resource utilization. So, the LUTs are configured in a serial way starting from the first LUT which calculates the least significant bit value of CRC. The mentioned inputs and outputs of the *CCAM* are described in Table 3-2.

The computation takes place by driving the 2-bits data to the `data_i` input when the `enable_i` input is at high. The `enable_i` input has to be held in high through one clock cycle for each 2-bits of the data sequence which we want to calculate the CRC for. The CRC computation of a given data sequence is completed by processing the last 2-bits of it and driving the `enable_i` input to low. The CRC of the given data sequence is produced through the `crc_o` output.

After a computation, a reset operation has to be executed for subsequent computations. If we want to resume calculating the CRC, the reset operation is not executed. The computation continues with the last value in CRC register in such a case. Otherwise, a reset operation is a must for a new computation. The reset operation can be executed by driving the `reset_i` input of *CCAM* to high for one clock cycle. After the reset, the initial value which was stored in last initialization process is written to the CRC register. So, the *CCAM* can be utilized for a new computation.

Table 3-2 Inputs and Outputs of CCAM

Port	Width	Type	Function
`conf_clk_i`	1	input	Clock input for dynamic reconfiguration
`conf_enable_i`	16	input	Enable/disable control input for reconfigurable LUTs. Each bit enables/disables the corresponding LUTs starting from the LSB.
`conf_data_i`	1	input	Data input for dynamic reconfiguration
`initialize_i`	1	input	1 : For registering the initial and final XOR values, 0 : Otherwise
`clk_i`	1	input	System clock signal
`data_i`	2	input	Data input of CRC calculation
`enable_i`	1	input	1 : For data ready notification, 0 : Otherwise
`reset_i`	1	input	1 : For reset request, 0 : Otherwise
`final_xor_i`	16	input	CRC result is XOR operated with this value
`initial_value_i`	16	input	Initial value of the CRC register
`crc_o`	16	output	Calculated CRC value

3.1.2 CRC Configurator Module (CCOM)

The proposed CRC calculator *CCAM* can be configured through the reconfiguration interface. Any microcontroller or microprocessor which has a compatible interface can configure the *CCAM*. However, a hardware-based module which can configure the *CCAM* was developed in this study.

This module which is named as *CCOM-CRC COnfigurator Module* determines the table contents of *CCAM* for a given CRC standard and updates the tables with these values. The table contents are determined due to the utilized dynamically reconfigurable block of the target platform for a given CRC polynomial.

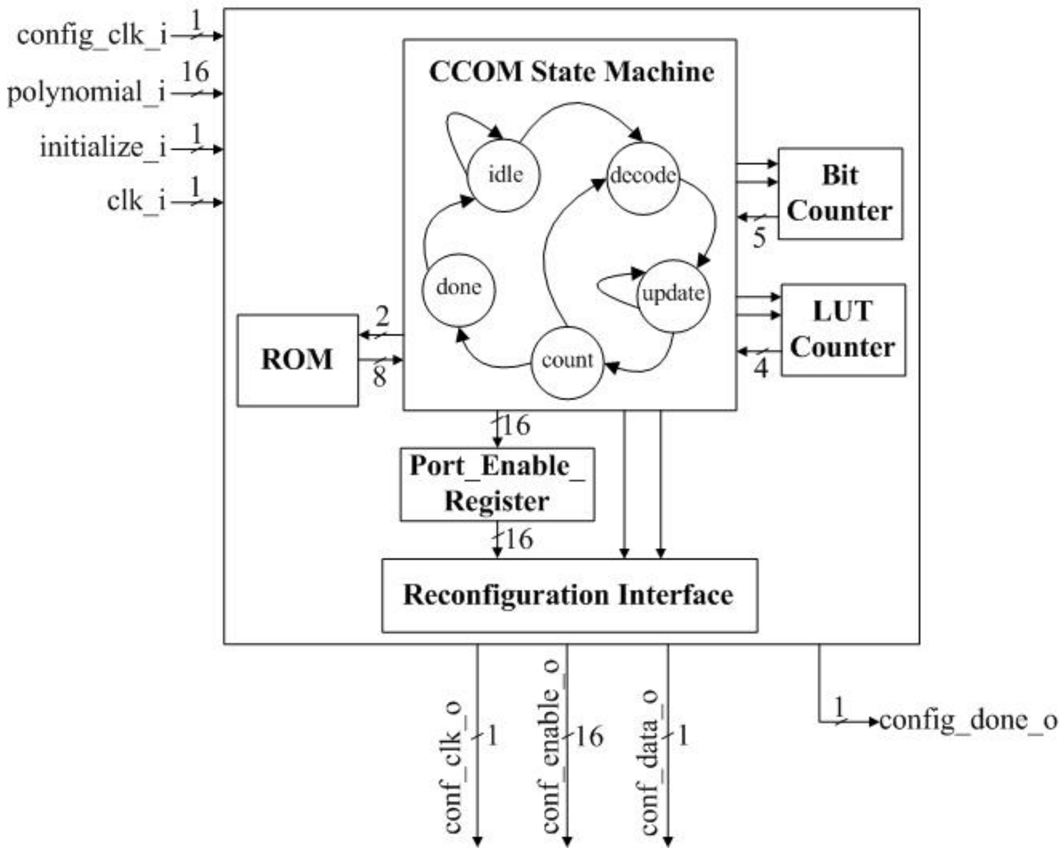


Figure 3-9 CCOM-CRC Configurator Module

The general structure of the *CCOM* is shown in Figure 3-9. It has a simple state machine which organizes the configuration process. There is a reconfiguration interface that provides to communicate with the *CCAM*. There are also some counters which are utilized for counting the sent bits of a table's content and counting the updated tables. Lastly, CCOM has a ROM which the LUT configuration values stored in it.

The configuration process is started by the rising edge of `initialize_i` input. Then, reconfigurable LUTs' contents are replaced with pre-calculated values for different CRC polynomial computations over the `conf_data_i` input of *CCAM*. The configuration data transmission takes place in synchronization with the `conf_clk_o` clock signal. Each LUT is controlled by the corresponding `conf_enable_o` signal. At the end of the configuration, `config_done_o` output of the *CCOM* goes to logic high. In Table 3-3, the inputs and outputs of the CCOM are described.

Table 3-3 CCOM Inputs and Outputs

Port	**Width**	**Type**	**Function**
`config_clk_i`	1	input	Clock input for dynamic reconfiguration
`polynomial_i`	16	input	Polynomial input for reconfiguring the calculator with driven polynomial value
`initialize_i`	1	input	1 : For reconfiguration start request 0 : Otherwise
`clk_i`	1	input	System clock signal
`config_done_o`	1	output	1 : Configuration is completed, 0 : Otherwise
`conf_clk_o`	1	output	Clock output for dynamic reconfiguration
`conf_enable_o`	16	output	Enable/disable control output for reconfigurable LUTs. Each bit enables/disables the corresponding LUTs starting from the LSB.
`conf_data_o`	1	output	Data output for dynamic reconfiguration

3.2 FPGA Implementation of DAROC Architecture

DAROC architecture can be implemented on any hardware platform which has dynamically reconfigurable LUTs. Xilinx Spartan-6 series FPGAs are appropriate for *DAROC* implementation. The first reason is that Xilinx Spartan-6 series FPGAs have a dynamically reconfigurable component which is named CFGLUT5. CFGLUT5 is a 5-input Look-Up Table. The second reason for implementing DAROC on Xilinx Spartan-6 is that the Spartan-6 is relatively a cost effective selection when considered the target systems.

The details of CFGLUT5 component and the *DAROC* implementation on FPGA by utilizing this component are described in following subsections.

3.2.1 CFGLUT5

The logic / Boolean functions are generally implemented by function generators on FPGAs. These function generators are implemented by look-up tables in Spartan-6 FPGAs [11].

CFGLUT5 is an element of Xilinx Spartan-6 FPGAs which is runtime, dynamically reconfigurable, 5-input look-up table. During the circuit operation, the logical function of this 5-input look-up table can be changed [12]. A general view of this element is shown in Figure 3-10.

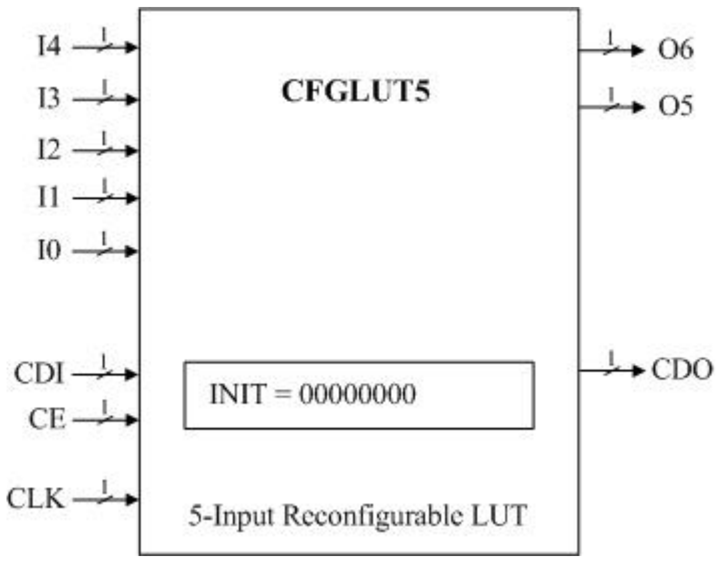


Figure 3-10 Reconfigurable CFGLUT5 Element

The inputs which are utilized in logical function are `I0`, `I1`, `I2`, `I3` and `I4`. The outputs of the logical function are driven through `O6` and `O5` ports of CFGLUT5. The inputs and the outputs of the CFGLUT5 are described in Table 3-4.

Table 3-4 CFGLUT5 Inputs and Outputs

Port	Width	Type	Function
`O6`	1	output	Output of 5-input look-up table
`O5`	1	output	Output of 4-input look-up table
`I0, I1, I2, I3, I4`	1	input	Look-up table inputs
`CDO`	1	output	Cascaded reconfiguration data output (For reconfiguring multiple CFGLUT5s in a chain)
`CDI`	1	input	Serial reconfiguration data input
`CLK`	1	input	Clock signal for reconfiguration
`CE`	1	input	Clock enable signal for CLK (Active high)

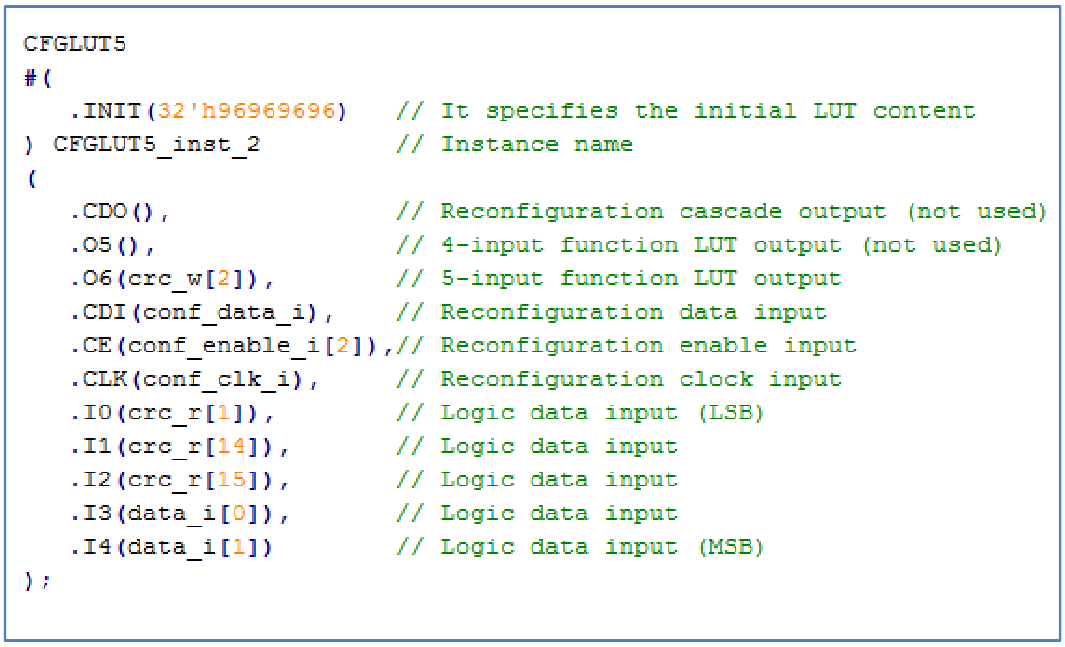
There are two outputs of the logical function which is implemented on CFGLUT5. `O6` is the output of the function that has 5 inputs. On the other hand, `O5` can be used as an output of 4-input function that is a subset of 5-input function. CFGLUT5 has an attribute which is named as *INIT*. The value loaded into *INIT* determines the logical function of the CFGLUT5 instance. The truth table of this reconfigurable element based on the current *INIT* value is shown in Table 3-5.

Reconfiguration of CFGLUT5 is executed by using three configuration ports of it, which are `CDI`, `CE` and `CLK`. The method of reconfiguration is loading the values of intended logical function into the *INIT* attribute. The reconfiguration process starts with shifting the MSB (INIT [31]) of data into the LUT. The data is transmitted through the `CDI` input synchronously with the reconfiguration clock signal `CLK`. Clock enable signal should be held at high during the reconfiguration. The logical function of the CFGLUT5 is updated as new INIT value which is shifted into it. This process can be occurred any time during circuit operation.

Table 3-5 CFGLUT5 Truth Table

Inputs					**Outputs**	
I4	**I3**	**I2**	**I1**	**I0**	**O6**	**O5**
1	1	1	1	1	INIT [31]	INIT [15]
1	1	1	1	0	INIT [30]	INIT [14]
…	…	…	…	…	…	…
1	0	0	0	1	INIT [17]	INIT [1]
1	0	0	0	0	INIT [16]	INIT [0]
0	1	1	1	1	INIT [15]	INIT [15]
0	1	1	1	0	INIT [14]	INIT [14]
…	…	…	…	…	…	…
0	0	0	0	1	INIT [1]	INIT [1]
0	0	0	0	0	INIT [0]	INIT [0]

A CFGLUT5 instance can be implemented by using the VHDL (**V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage) or the Verilog HDL (**H**ardware **D**escription **L**anguage) instantiation template of it. In Figure 3-11, a CFGLUT5 instantiation in Verilog HDL is shown.



```
CFGLUT5
#(
    .INIT(32'h96969696)    // It specifies the initial LUT content
) CFGLUT5_inst_2           // Instance name
(
    .CDO(),                // Reconfiguration cascade output (not used)
    .O5(),                 // 4-input function LUT output (not used)
    .O6(crc_w[2]),         // 5-input function LUT output
    .CDI(conf_data_i),     // Reconfiguration data input
    .CE(conf_enable_i[2]),// Reconfiguration enable input
    .CLK(conf_clk_i),      // Reconfiguration clock input
    .I0(crc_r[1]),         // Logic data input (LSB)
    .I1(crc_r[14]),        // Logic data input
    .I2(crc_r[15]),        // Logic data input
    .I3(data_i[0]),        // Logic data input
    .I4(data_i[1])         // Logic data input (MSB)
);
```

Figure 3-11 CFGLUT5 Instance in Verilog HDL

3.2.2 DAROC CFGLUT5-Based FPGA Implementation

Due to the run-time reconfiguration property of CFGLUT5 element, Xilinx Spartan-6 series FPGAs are appropriate for the implementation of *DAROC*. The combinational logic blocks of the *DAROC* can be implemented by utilizing this reconfigurable element.

To start with the simple CRC structure, a serial implementation of *DAROC* for 16-bits CRC polynomials is shown in Figure 3-12. In this structure, the CRC of a data sequence is computed in such a way that one bit is processed at one clock cycle. So, the data is driven serially through the `data_i` input during the CRC computation.

To reconfigure the *DAROC* for a new polynomial that has 16-bits length, the initialization operations are executed. The initialization starts by driving the new values to the input ports which are the `polynomial_i`, the `final_xor_i` and the `initial_value_i`. Then, the `initialize_i` input is driven to high. *CCOM* starts to update the contents of *CCAM*'s CFGLUT5 blocks through the reconfiguration interface. After the completion of reconfiguration, the `config_done_o` output goes to high for "configuration is done" notification.

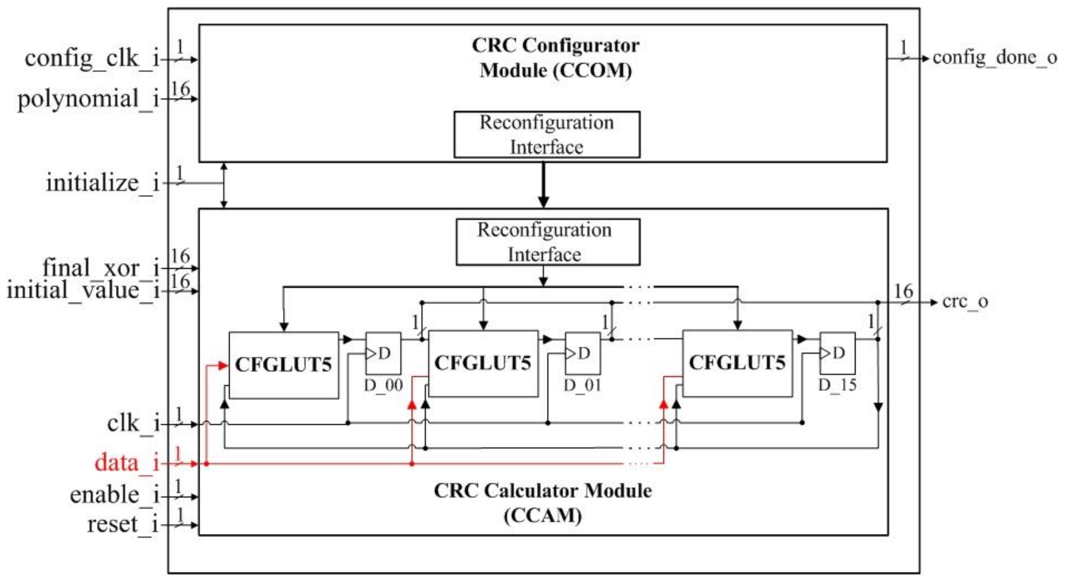


Figure 3-12 Serial Implementation of DAROC on Xilinx Spartan-6

CCAM implementation, which is shown in Figure 3-12, is the minimal CRC calculator for 16-bits polynomial CRC standards in terms of resource utilization. The proposed design is implemented on Xilinx Spartan-6 XC6SLX45T platform. Number of slice register utilization is 32 out of 54576. It utilizes 37 out of 27288 slice LUTs.

The parallelization method of *DAROC* which is proposed before is implemented to increase the CRC computation throughput as much as possible. It is straightforward to increase the parallelization level. However, the motivation of this study is to design a dynamically reconfigurable CRC calculator for small scale systems. Therefore, the minimal resource utilization is one of the most important goals of this study. Considering the tradeoff between the throughput and the resource utilization, the optimum parallelization level can be determined by the requirements of target systems.

It can be noticed that the maximum input utilization of CFLUT5 is three in serial DAROC implementation which is shown in Figure 3-12. These inputs are the `data_i` signal, the output of the former CRC register and the output of the last CRC register (CRC [15] in the 16-bits design). Considering the input number of CFGLUT5 element, 2 out of 5 input ports are not used. In such a case, the maximum operation capacity of the element is not used in implemented logical function. Consequently, a more efficient design can be investigated in terms of resource utilization.

The previously proposed unrolling method is used to parallelize the CRC computation. By the simplicity of this unrolling method, it can be realized that each step, which increases the parallelization level by one bit, adds extra two inputs. These inputs are added to all LUT blocks which are corresponded to CRC bits. The reason for adding these extra inputs is handling all possible CRC polynomials for computation.

The CFGLUT5 elements have two unused input ports in the previous implementation. If the design is updated from serial form to 2 bit parallel form, the 5-input reconfigurable look-up tables would be utilized in an efficient way. Because, all the input ports of CFGLUT5s would be used in the logical functions. In other words, there is no missing input port of a CFGLUT5 and there is no redundant field in the look-up table.

The number of required CFGLUT5 per 1-bit of *CCAM* changes by parallelization level in the way that is given in Figure 3-13.

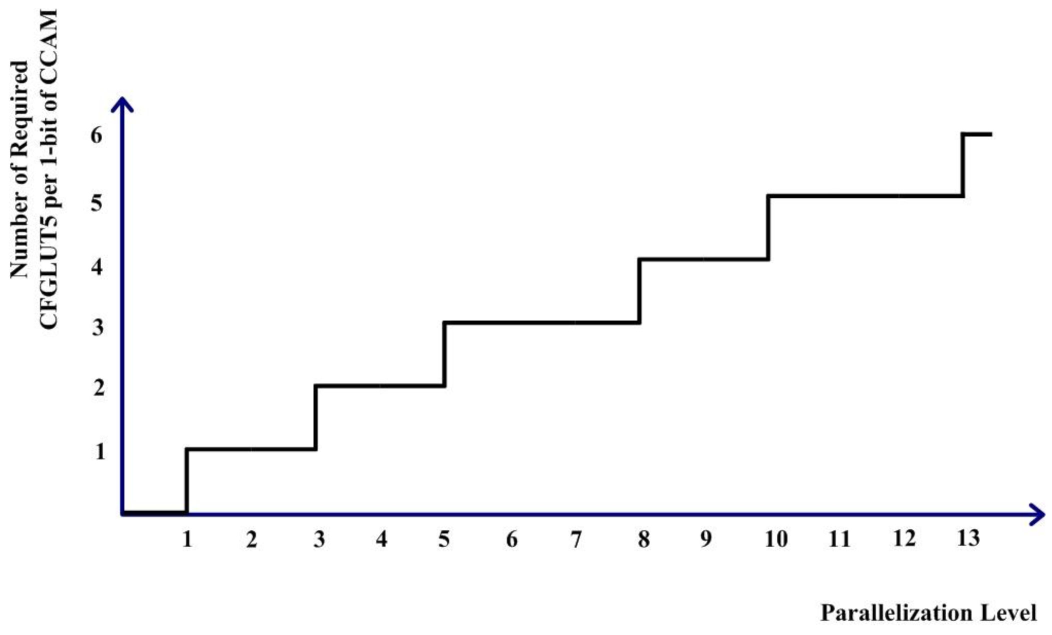


Figure 3-13 CFGLUT5 Utilization with increasing Parallelization Level

A structural view of 2-bits parallel *DAROC* implementation on Xilinx Spartan-6 XC6SLX45T is shown in Figure 3-14. The proposed architecture is implemented by utilizing CFGLUT5 elements in place of the *Dynamically Reconfigurable LUTs* which are shown in Figure 3-8.

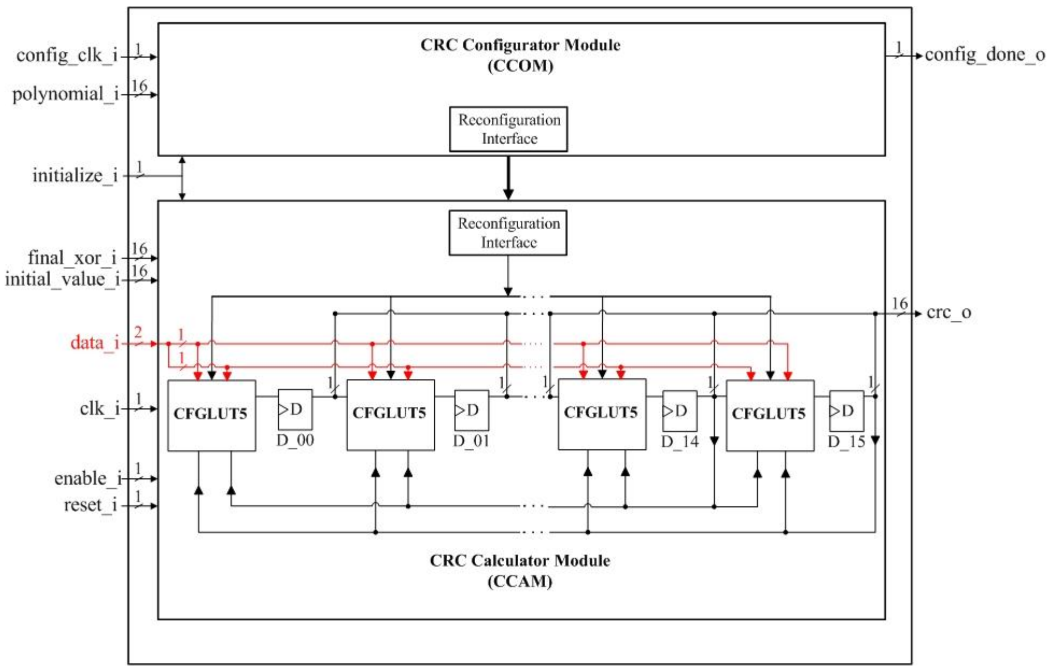


Figure 3-14 2-Bits Parallel Implementation of DAROC on Xilinx Spartan-6

The resource utilization of this 2-bit parallel implementation is as low as the serial implementation of DAROC although the throughput is doubled. It is achieved by utilizing the unit logical function generators of the Spartan-6 in an efficient way. There are no more elements than the serial implementation. Also, the throughput is doubled properly due to the fact that the critical path is not changed.

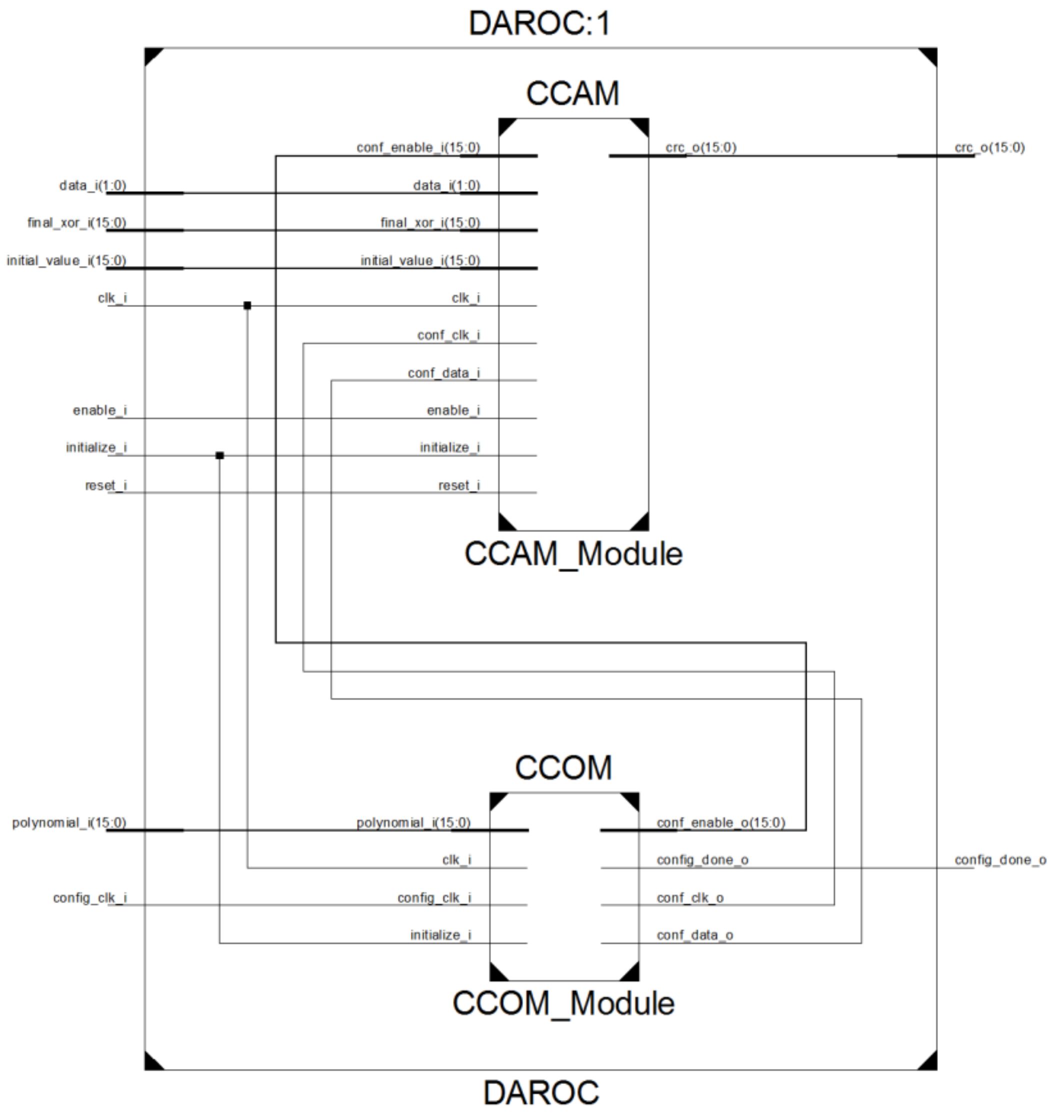


Figure 3-15 DAROC Top Level RTL Schematic

DAROC was designed and implemented in Verilog HDL. Xilinx ISE 14.4 design environment was used for all parts of the design flow including “the synthesis”, “the mapping” and “the place and route”. *DAROC* top level RTL schematic which is generated in Xilinx ISE 14.4 platform is shown in Figure 3-15.

CHAPTER 4

PERFORMANCE EVALUATION

The proposed architecture is tested and verified in simulations. During the simulations, the functional analysis of *DAROC* is performed. Also, timing simulations are executed for performance measurement. After the simulations, the HardWare - In - the - Loop (HWIL) tests are performed for precise evaluations. In this chapter, performance evaluations of *DAROC* are presented.

4.1 Simulations

Simulations were executed for functional tests of the proposed architecture. Xilinx ISE Design Suite 14.4 has a Hardware Description Language simulator for behavioral and timing simulations [16].

Before the simulations, the test benches have been designed for testing the DAROC with full coverage. These test benches have been written in Verilog HDL. Then the behavioral and timing simulations were performed. In Figure 4-1, a simple simulation for CRC calculation is shown.

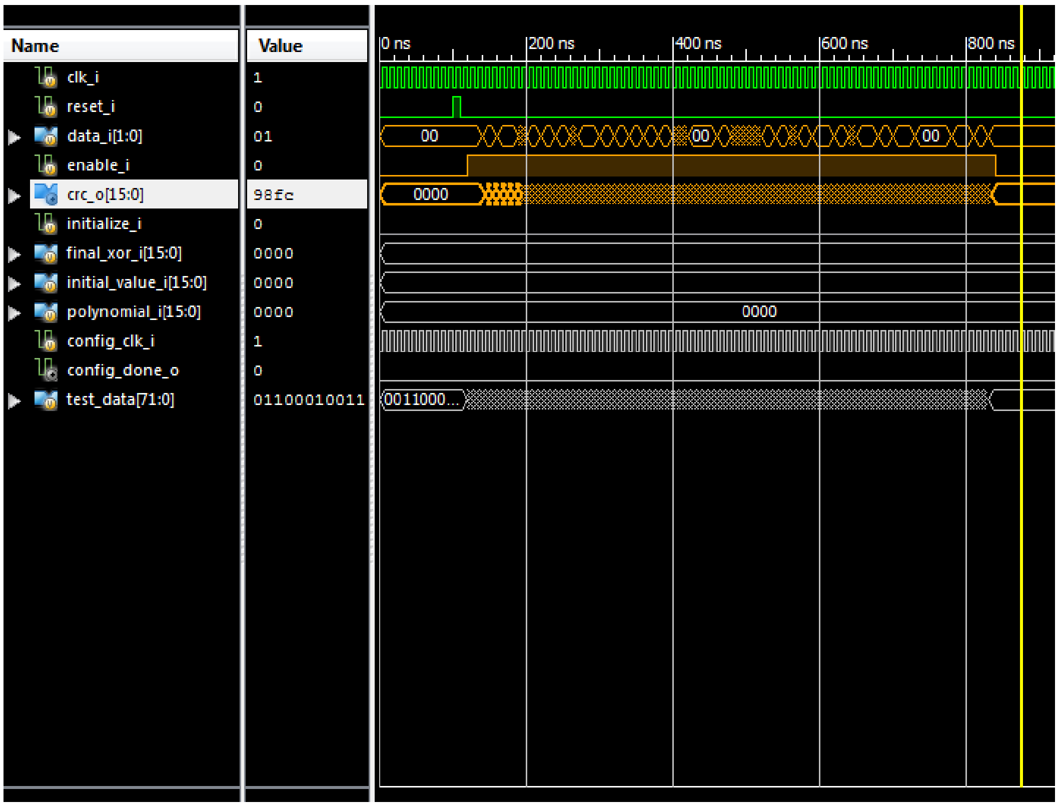


Figure 4-1 Simulation for CRC Computation of DAROC

In this simulation, the basic flow of CRC computation is shown. First of all, a reset operation has to be executed to start a new computation by driving the `reset_i` input. It is simulated by driving the `reset_i` input from low to high at the time of 100 ns in simulation shown in Figure 4-1. Then, reset is completed by driving the input from high to low. The computation of CRC can be started anymore.

CRC computation is started by driving the `enable_i` input from low to high. Starting point of the computation is approximately at 120ns of the simulation which is shown in Figure 4-1. During a period of time, a data sequence is driven from the `data_i` input. At the end, the `enable_i` input goes to low which means the computation is halted. If the data sequence is completed at this point, the CRC result can be read from the `crc_o` output. In this simulation, the calculated CRC is “0x98FC” at the halt point.

4.2 Test Setup

The test setup of *DAROC* implementation consists of a computer which the Xilinx ISE Design Suite 14.4 Evaluation Platform is installed on it, interface cables for FPGA configuration and serial communication between the FPGA and the computer, a Spartan-6 evaluation board and a two channel oscilloscope. This setup is shown in Figure 4-2.

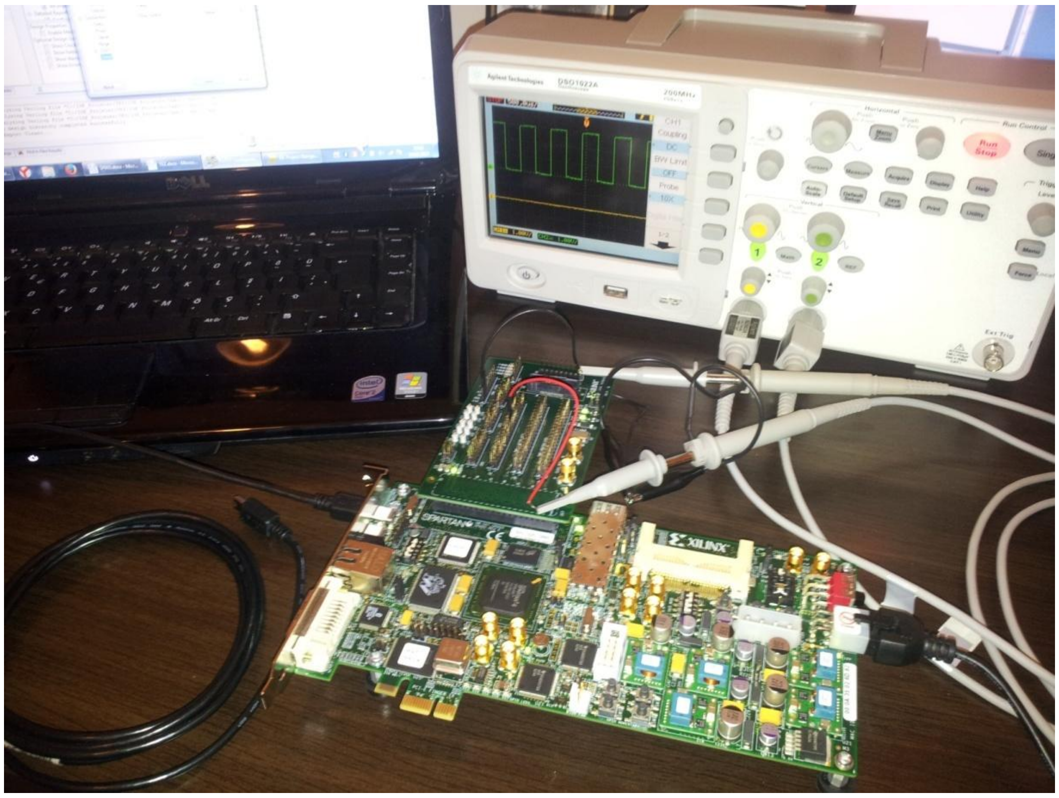


Figure 4-2 Test Setup

Xilinx Spartan-6 FPGA SP605 Evaluation Kit is used for hardware implementation tests. There is a Xilinx Spartan-6 XC6SLX45T FPGA on the board. Four FPGA configuration options are available. During the tests, 8 MB Quad SPI flash memory is used for FPGA configuration. 27 MHz on board user clock is used as the system clock signal of DAROC.

The Silicon Labs CP2103GM USB to UART Bridge component is available for serial communication on SP605 board [15]. By using a terminal program on PC, the test messages are transmitted to and received from FPGA over this UART interface. There are three types of test message. First one is used for configuration of CRC standard. The second is the message that resets the CRC computation. Lastly, there is a data sequence message to make the CCAM calculate the CRC of it. In Table 4-1, the details of these message commands are shown.

Table 4-1 Command List of Test Setup

Command Name	**ID**	**Length (byte)**	**Function**	**Message Reply**
CONFIGURE	0x01	7	Used for reconfiguration of CRC calculator at runtime. Message Format: Command ID (1 – byte), Polynomial (2 - byte), Initial Value (2 - byte), Final XOR Value (2 – byte).	“ACK”
RESET	0x02	1	Used for resetting the CRC calculator for new calculations. After reset, the stored initial value is written to CRC register. Message Format: Command ID (1 – byte).	“ACK”
COMPUTE	0x03	2	CRC computation of given data byte is occurred by this command. The computation continues with the last value in CRC registers for new COMPUTE commands. Message Format: Command ID (1 – byte), Data byte (1 – byte).	The calculated 2 byte CRC value is returned.

For the purpose of performing the DAROC tests an environment designed on FPGA that covers the DAROC. The design has an instance of DAROC and a UART that is used for command transmission. Also, a central state machine is designed for fetching and executing the commands. The block diagram of the test design which is implemented on FPGA is shown in Figure 4-3.

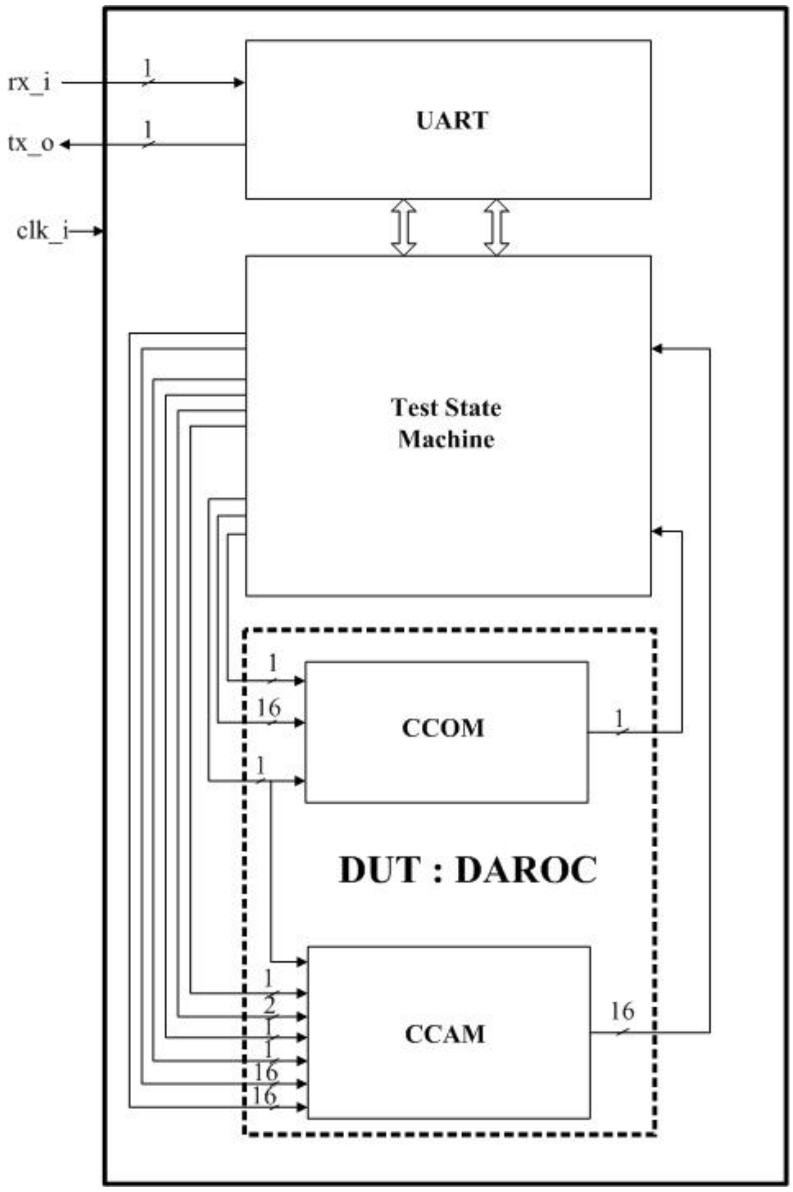


Figure 4-3 Block Diagram of DAROC Test System

4.3 Evaluation Results

DAROC implementation is simulated in ISIM platform. The simulation results are verified by the HWIL tests executed on Xilinx Spartan-6 FPGA SP605 Evaluation Kit. The results are presented in Table 4-2.

Table 4-2 DAROC Implementation Results

	DAROC Implementation
Platform	Spartan-6 XC6SLX45T
Slice LUTs Utilization	37
Parallelization Level	2
Clock Frequency (MHz)	353
Throughput (Mbps)	705
Throughput (Mbps) / Slice LUTs	19.1
Reconfiguration Time	512 clock cycle 1450 ns

Resource utilization details of DAROC on Xilinx Spartan-6 XC6SLX45T are shown in Table 4-3.

Table 4-3 DAROC Device Utilization Summary on Xilinx XC6SLX45T

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	70	54,576	1%
Number used as Flip Flops	70		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	87	27,288	1%
Number used as logic	67	27,288	1%
Number using O6 output only	47		
Number using O5 output only	0		
Number using O5 and O6	20		
Number used as ROM	0		
Number used as Memory	16	6,408	1%
Number used as Dual Port RAM	0		
Number used as Single Port RAM	0		
Number used as Shift Register	16		
Number using O6 output only	16		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used exclusively as route-thrus	4		
Number with same-slice register load	4		
Number with same-slice carry load	0		

Table 4-3 (continued)

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number with other load	0		
Number of occupied Slices	46	6,822	1%
Number of MUXCYs used	0	13,644	0%
Number of LUT Flip Flop pairs used	96		
Number with an unused Flip Flop	42	96	43%
Number with an unused LUT	9	96	9%
Number of fully used LUT-FF pairs	45	96	46%
Number of unique control sets	22		
Number of slice register sites lost to control set restrictions	122	54,576	1%
Number of bonded IOBs	72	296	24%
IOB Flip Flops	16		
Number of RAMB16BWERs	0	116	0%
Number of RAMB8BWERs	0	232	0%
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of BUFG/BUFGMUXs	2	16	12%
Number used as BUFGs	2		
Number used as BUFGMUX	0		
Number of DCM/DCM_CLKGENs	0	8	0%
Number of ILOGIC2/ISERDES2s	16	376	4%
Number used as ILOGIC2s	16		
Number used as ISERDES2s	0		
Number of IODELAY2/ IODRP2/ IODRP2_MCBs	0	376	0%
Number of OLOGIC2/OSERDES2s	0	376	0%
Number of BSCANs	0	4	0%
Number of BUFHs	0	256	0%

Table 4-3 (continued)

Slice Logic Utilization	Used	Available	Utilization
Number of BUFPLLs	0	8	0%
Number of BUFPLL_MCBs	0	4	0%
Number of DSP48A1s	0	58	0%
Number of GTPA1_DUALs	0	2	0%
Number of ICAPs	0	1	0%
Number of MCBs	0	2	0%
Number of PCIE_A1s	0	1	0%
Number of PCILOGICSEs	0	2	0%
Number of PLL_ADVs	0	4	0%
Number of PMVs	0	1	0%
Number of STARTUPs	0	1	0%
Number of SUSPEND_SYNCs	0	1	0%
Average Fan-out of Non-Clock Nets	2.78		

Module level utilization is presented in Table 4-4.

Table 4-4 Module Level Resource Utilization

Module Name	Slices	Slice Reg	LUTs	LUTRAM
CCAM	29	32	37	16
CCOM	17	38	50	0

4.4 Discussion

While the parallelization level is 2, resource utilization is as low as in serial implementation. This area optimization is succeeded by considering the input size of configurable logic unit in Spartan-6's. The aim is achieving the maximum throughput by minimum resource utilization. So, while the minimum resource requirement is obvious according to the serial implementation, an optimized design is achieved by increasing the throughput using this resource limitation.

The propagation delay through a LUT does not change due to the implemented function in it [11]. Therefore, there is not an increment in critical path delay while the parallelization level goes from 1 to 2.

In addition, DAROC is more advantageous than the other table-based design proposed in [6] in terms of reconfiguration port size. DAROC has serial reconfiguration ports for CRC LUTs. On the contrary, the proposed table-based design requires wider reconfiguration ports for memory interfaces. Therefore, the resource utilization is reduced also by serializing the reconfiguration interface.

DAROC has a considerably high *Throughput/Slice LUTs* value in comparison to the proposed table-based design.

Although the throughput is lower than the mentioned run-time reconfigurable studies, the performance of DAROC is remarkably sufficient for targeted systems. In Table 4-5, some communication protocols which have CRC polynomials in 16-bits length are listed. Also, the line rates of these interfaces are presented.

Table 4-5 Communication Protocols That Uses 16-bits CRC

Communication Protocol	**Line Rate**
DECT	32 kbps
ANSI X3.28	14.4 kbps
MODBUS	19.2 kbps
USB 1.0	12 Mbps
USB 2.0	480 Mbps

16-bits CRC has a wide range of utilization in communication systems in addition to the listed protocols. It is used in various serial communication applications which have RS-232, RS-422 or RS-485 interfaces. DAROC can be implemented in these applications that require 16-bits CRC for reducing the resource utilization.

CHAPTER 5

CONCLUSIONS

In this thesis we presented an area minimized and dynamically reconfigurable hardware CRC calculator for 16-bits CRC standards.

We add multiplexers to the standard serial implementation of calculator to achieve the capability of switching between the CRC standards. Then, the combinational parts of the calculator which consist of multiplexers and XOR gates are implemented by reconfigurable logic blocks for dynamically changing the CRC polynomial during run time. We parallelize the serial architecture to 2 bits to fully utilize the available reconfigurable LUT resources. So, the throughput is doubled while the resource utilization remains the same. In addition, the resource utilization is reduced by using a serial reconfiguration interface with respect to the other table-based architectures.

The proposed architecture is implemented on Xilinx Spartan-6 XC6SLX45T platform. The design is simulated by using ISIM - Xilinx ISE Design Suite 14.4 Simulator. We execute the HWIL tests on Xilinx SP605 Evaluation Kit. The results show that it works properly. We achieved 705 Mbps throughput with 32 out of 54576 slice register and 37 out of 27288 slice LUTs utilization on Xilinx Spartan-6 XC6SLX45T for 16-bits CRC.

In this thesis, a 2-bits parallel CRC calculator is implemented for 16-bits CRC polynomials. In the future, the proposed architecture might be extended for wider CRC polynomials. A flexible structure might be designed in terms of polynomial length. In addition, the reconfiguration interface of the calculator might be reduced by cascading the LUTs.

One of the most important metrics for a dynamically reconfigurable CRC calculator is the reconfiguration time in the systems that have multiple communication interfaces. Therefore, decreasing the reconfiguration time of a CRC calculator is suggested as a further study.

REFERENCES

[1] Walma, M., "Pipelined Cyclic Redundancy Check (CRC) Calculation," *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on* , pp.365,370, 13-16 Aug. 2007

[2] Akagic, A.; Amano, H., "Performance evaluation of multiple lookup tables algorithms for generating CRC on an FPGA," *Access Spaces (ISAS), 2011 1st International Symposium on* , pp.164,169, 17-19 June 2011

[3] Shukla, S.; Bergmann, N.W., "Single bit error correction implementation in CRC-16 on FPGA," *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on* , pp.319,322, 6-8 Dec. 2004

[4] Toal, C.; McLaughlin, K.; Sezer, S.; Xin Yang, "Design and Implementation of a Field Programmable CRC Circuit Architecture," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.17, no.8, pp.1142,1147, Aug. 2009

[5] Grymel, M.; Furber, S.B., "A Novel Programmable Parallel CRC Circuit," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.19, no.10, pp.1898,1902, Oct. 2011

[6] Akagic, A.; Amano, H., "Performance analysis of fully-adaptable CRC accelerators on an FPGA," *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on* , pp.575,578, 29-31 Aug. 2012

[7] Ramabadran, T.V.; Gaitonde, S.S., "A tutorial on CRC computations," Micro, IEEE , vol.8, no.4, pp.62,75, Aug. 1988

[8] Campobello, G.; Patane, G.; Russo, M., "Parallel CRC realization," *Computers, IEEE Transactions on* , vol.52, no.10, pp.1312,1319, Oct. 2003

[9] Albertengo, G.; Sisto, R., "Parallel CRC generation," *Micro, IEEE* , vol.10, no.5, pp.63,71, Oct. 1990

[10] Yan Sun; Min Sik Kim, "A Table-Based Algorithm for Pipelined CRC Calculation," *Communications (ICC), 2010 IEEE International Conference on* , pp.1,5, 23-27 May 2010

[11] http://www.xilinx.com/support/documentation/user_guides/ug384.pdf (last visited on 18.08.2013)

[12] http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/spartan6_hdl.pdf (last visited on 18.08.2013)

[13] Joglekar, A.; Kounavis, M.E.; Berry. F.L., “A Scalable and High Performance Software iSCSI Implementation,” *File and Storage Technologies (FAST’05) , Proceedings of 4th USENIX Conference on* , Vol.4. USENIX Dec, 2005

[14] Peterson, W.W.; Brown, D.T., "Cyclic Codes for Error Detection," *Proceedings of the IRE* , vol.49, no.1, pp.228,235, Jan. 1961

[15] http://www.xilinx.com/support/documentation/boards_and_kits/ug526.pdf (last visited on 19.08.2013)

[16] http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/plugin_ism.pdf (last visited on 19.08.2013)