

DESIGN AND IMPLEMENTATION OF A VARIABLE RECONFIGURABLE CRC CIRCUIT ARCHITECTURE

Sravankumar Sadasivuni and Sreedhar Tandle

Department of Electronics & Communication,
Sri Sivani College of Engineering, Chilakapalem, Srikakulam, Andhra Pradesh, India
E mail id: sra123999@gmail.com, sreedhartandleiete@yahoo.com

[Received-09/06/2012, Accepted-01/10/2012]

ABSTRACT

The design and implementation of a variable field programmable cyclic redundancy check (CRC) circuit architecture, suitable for deployment in network related and packet processing, security processing and storage related functions is presented. 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 implemented hardware of the CRC computation takes the form of a modulo-2 convolution of a data word against a binary version of the CRC polynomial. The design and implementation of a novel field programmable CRC computation circuit that is an ideal IP core for VLSI deployments. 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.

Index Terms—Cyclic redundancy check (CRC), error detection, field Programmable, network processing, reconfigurable.

I. INTRODUCTION

Cyclic Redundancy Codes (CRCs) are commonly used for error detection in embedded networks and other applications. But many applications employ CRCs that provide far less error detection capability than they might achieve for a given number of CRC bits. 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. 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. 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.

In this paper, we derive a fully variable field programmable, parallel architecture for a CRC computation circuit. The objective was to explore a domain specific programmable architecture capable of supporting 5 Gb/s line rates at a minimal area cost and robust error detection. The resulting architecture is able to support all types and sizes of CRC polynomial, for all types of protocols and data encryption. Furthermore, the circuit can handle a variable number of input octets in runtime for byte orientated variable sized protocols. An embedded self-reconfiguration controller allows any CRC function to be configured, while minimizing programming time and complexity. This paper explores the architecture and functions of the variable field programmable CRC computation circuit and analyses its performance.

II. CYCLIC REDUNDANCY CHECK

Data integrity is imperative for many network protocols, especially data-link layer protocols. 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 [1] 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. In such systems, it is

becoming common to offload TCP/IP operations to hardware. CRC computation faces similar computational overhead constraints in software and is thus an ideal candidate for offloading to specialized hardware. The ability of CRC implemented in hardware to be reconfigurable to handle new protocols will offer a key advantage in this fast developing area.

The fact that the reconfigurable CRC circuit that has been implemented can quickly switch between any polynomial gives it a key advantage over the other circuits referenced, in terms of flexibility and ease of upgrade for new and emerging applications and standards. Advanced NUs have to support most communications protocols, while maintaining a throughput performance of 1–10 Gb/s. Such performance cannot be achieved without a highly flexible CRC offload engine capable of parallel computation. iSCSI in particular is a good illustration of an application where new standards or more suitable polynomials can emerge. None of the publications referenced here offer the ability to reconfigure hardware in the field to operate with an entirely new CRC polynomial.

A. CRC Related Background

A large number of CRC polynomials of various lengths are available to use over a range of applications. Reference [2] investigates a total of 48 polynomials, ranging in length from 3- to 16-bits, that are suitable for embedded network applications utilizing CRC error detection. The paper shows how the various polynomials have been assessed for their ability to detect error patterns in messages. It shows that for different data word lengths, different CRC polynomials can be more suitable than others. This assessment is carried out based on maximum hamming distances. Similarly [3] investigates a number of 32-bit CRC polynomials, all suitable for network applications such as Ethernet and iSCSI.

CRC functions have been widely implemented in software using methods such as lookup tables [4] and shift and addition [5]. 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* [7] 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.

[8] Addresses a technique that allows pipelining to increase the circuit speed, independent of the underlying technology.

Reference [9] derives a VLSI implementation of a 32-bit CRC generator circuit based on Galois field arithmetic and look-ahead blocks. With an eighth-order look-ahead function this circuit can operate at 100 MHz despite the dated 0.6-micron technology. The circuits are flexible in terms of the number of input bits processed at a time, up to 32-bits, but they are restricted to using one CRC polynomial.

Reference [10] 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.

[11] Describes a pipelined and parallel implementation for an FPGA based CRC function. The level of parallelism can be varied between 8- to 32-bits and claims performance results of 1 to 4 Gb/s (depending on the level of

parallelism selected). Any polynomial can be selected before synthesis, but not after.

[12] Describes the derivation of VHDL code with a generic construct that allows a designer to synthesize 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 pre synthesis, after which the hardware is fixed and the CRC function is not configurable.

[13] Uses a recursive mathematical formula to derive parallel CRC circuits that can be generated automatically. The examples use MATLAB code to generate the VHDL code for the circuit. The polynomial and number of bits to be processed in parallel can be specified separately. The method is flexible and is likely to save both time and cost in the design phase, yet like the other circuits, this one will be fixed to a single polynomial as the circuit itself is inflexible post-synthesis.

Reference [14] 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.

Reference [15] presents a software implementation of the iSCSI protocol that includes implementing CRC error detection, which is recognized as the key bottleneck in the system. 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 [15] is likely to be very flexible, it comes at the expense of a Pentium processor. The next section outlines the derivation of a CRC circuit implementation that fulfils the outlined flexibility criteria.

III. DERIVATION AND IMPLEMENTATION OF THE FIELD PROGRAMMABLE CRC COMPUTATION CIRCUIT

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

$$G(x) = 1 + x + x^2 + x^4 + x^5 + x^7 + x^9 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

To find the FCS, first a number of zeroes equal to the number of FCS digits to be generated are appended to the message M(x). This is equivalent to multiplying M(x)

By 2^n , where n is the number of FCS digits. This value is then divided by the generator polynomial G(x). Which contains one more digit

than the FCS. The division uses modulo-2 arithmetic, where each digit is independent of its neighbor and numbers are not carried or borrowed, thus addition and subtraction are performed via an exclusive-OR (XOR) function. The remainder R(x) is appended to the end of the message before transmission. At the receiver, the message plus the FCS is divided by the same polynomial. If the remainder is zero then it can be assumed that no error has occurred.

The field programmable CRC design is based on the fundamentals established in [6], which derives the matrix D(1) that effectively forms the logic array of the CRC calculator circuit. Due to space restrictions the full derivation is not included here, but can be found in the reference

$$D = \begin{bmatrix} G \\ GT \\ GT^2 \\ \dots \\ GT^{j-1} \end{bmatrix} \tag{1}$$

where

$$T = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ g_{(0)} & g_{(1)} & g_{(2)} & \dots & g_{(k-1)} \end{bmatrix} \tag{2}$$

and G is the Generator polynomial (an example would be the polynomial used for Ethernet)

$$G = [g_{(0)}g_{(1)}g_{(2)} \dots g_{(k-1)}] \tag{3}$$

Each value in G is multiplied (ANDed) with the corresponding value in T. The results are XORed together, producing a D matrix of 0's or 1's. The position of the 1's in D determines the position of XOR gates within the logic array while j is the width of the input port in bits. Enabling programmability for parallel CRC computation requires the matrix D to be configurable for all known generator polynomials G(x). Furthermore, configuration logic for the configurable XOR array is required to adjust the input and CRC sizes.

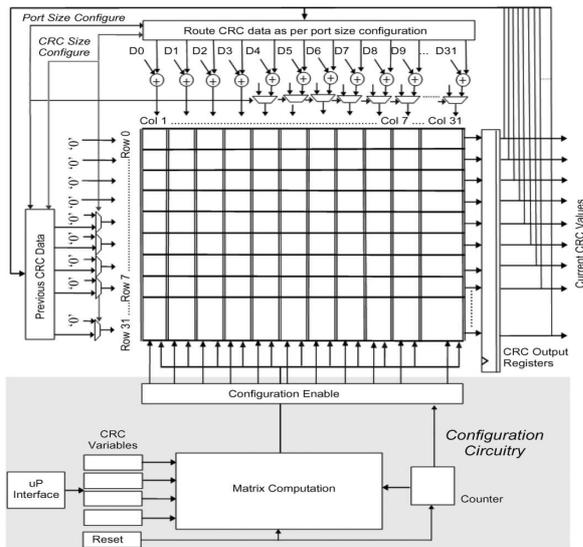


Fig. 1. Field programmable CRC architecture.

fully programmable, reconfigurable CRC circuit. The circuit is composed of six main components, the programmable input and feedback multiplexers, the configurable XOR array, the array configuration circuit and the CRC configuration processor. The top of the diagram shows the logic associated with the CRC cell array. The input data enters the array down the columns and the Fig. 1 shows a diagram illustrating the architecture of the outputs are formed along the rows. The current CRC value is held in a register at the array output, which is fed back and XORed with the input data of the next clock cycle as part of the CRC computation process. The outputs are then stored in the registers for the next clock cycle.

The CRC configuration parameters are passed via a microprocessor interface. The desired CRC polynomial $G(x)$ and the input port size are stored in registers. By selecting a signal *Generate Matrix*, a process is initiated that computes the configuration data and configures the CRC cell array with the required data. The configuration circuitry computes and configures one row of data every clock cycle. This reduces the memory required since the calculation of each row of the D matrix is based on the result of the calculation of the previous row. The configuration data is

broadcast to every column of the array, with one column enabled for configuration at a time, via one-hot encoding using a counter. When the CRC cells are fully configured, the configuration processor is not used. The *Port Size Configure* and *CRC Size Configure* signals control a set of multiplexers that enable/disable input and CRC feedback data to cater for the size of the CRC polynomial and the size of the input port. The *Port Size Configure* signal is also responsible for reconfiguring the circuit to process various input word sizes, e.g., if the port size is configured as 32-bit and the last cycle of the payload data contains only 16-bits to be processed, the *Port Size Configure* signal; switches input bits 16 to 31 to the “0” input of each multiplexer; switches the bottom 16 multiplexers to the previous CRC data input at the left-hand side of the array and finally routes bits 16 to 31 of the previous CRC data to rows 16 to 31 of the array.

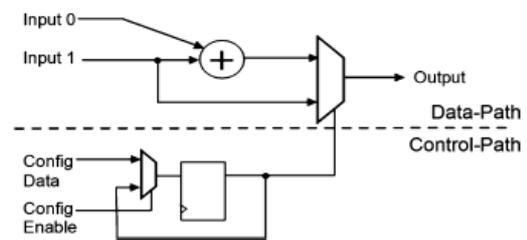


Fig. 2. Programmable CRC array cell.

The configurable XOR array is comprised of interconnected cells, corresponding to the D matrix. Each cell can be configured as an XOR gate (a “1” in D), or as a basic input to output connection (a “0” in D). Fig. 2 shows a diagram of one of the field programmable elements that facilitate this function in the array. The data-path can be configured so that the output will XOR the two inputs, or simply output *Input 1*. The control-path contains a configuration register which selects the data-path function and is programmed via the *Config Data* input when the *Config Enable* input is set high.

The computation of the D matrix is an iterative process, where the computation of each row is based on the result from the previous row.

Therefore, one row is computed every clock cycle and configuring the whole matrix requires 33 clock cycles. This is the minimum time possible due to the feedback required. Fig. 3 show the logic used to compute a D matrix row in one clock cycle for an example 4-bit CRC polynomial, using the T matrix data and the current D matrix row signal. The diagram shows how a single bit of the D matrix is computed on the first loop using the rows and columns shaded grey in the D and T matrices. Subsequent loops progress through the remaining rows and columns.

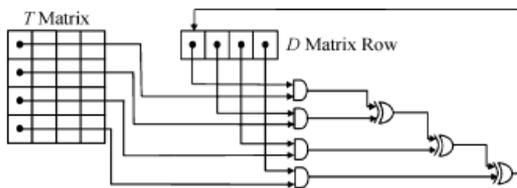


Fig. 3. D matrix row calculation, first loop.

For a CRC-64 or higher, two or more of the datapath circuits (top of Fig. 1) are combined in a diagonal cascade with additional feedback wiring, so that column and row 0 of the second circuit become column and row 32 of the CRC-64. Similarly for circuits smaller than 32-bit, e.g., 8-bit, the XOR array is configured so that four CRC-8's run diagonally from the top left. The XOR array and wiring is used to connect the four blocks together with the routing necessary to compute the CRC-8's. This means 32-bits can be calculated in parallel for all CRC sizes and all CRC sizes support the same line-speed.

Both the programmable architecture and an optimized implementation of the Ethernet CRC-32 polynomial were synthesized for FPGA, to allow the cost of full programmability to be established. The results showed that the programmable circuit operated at 117 MHz, while the Ethernet polynomial circuit operated at 233 MHz, nearly twice the frequency. The increase in area cost was approximately 700%, which is due to the introduction of feedback

logic, the configurable array and the reconfiguration controller. Overall this is a significant increase in hardware, but the circuit now has significantly improved capabilities to operate an extensive range of CRC polynomials and widths instead of just one.

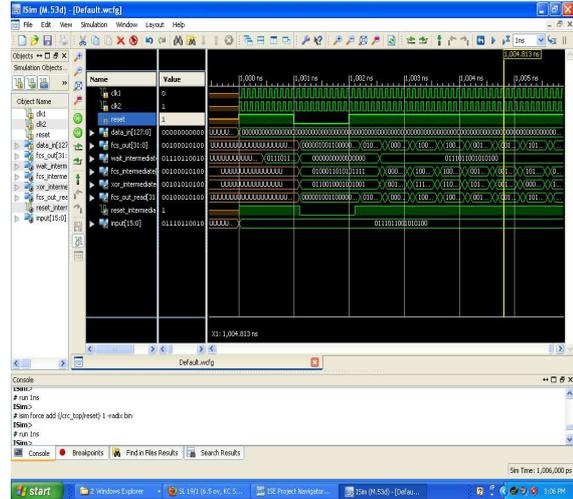
IV .SYNTHESIS RESULTS AND PERFORMANCE ANALYSIS

The circuit has been described in VHDL and synthesized for a 32×32 cell array using XILINX ISE, MODELSIM SIMULATOR EDAtools.

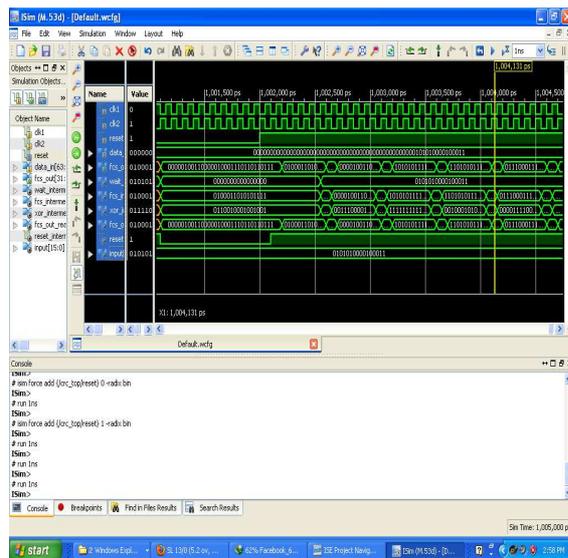
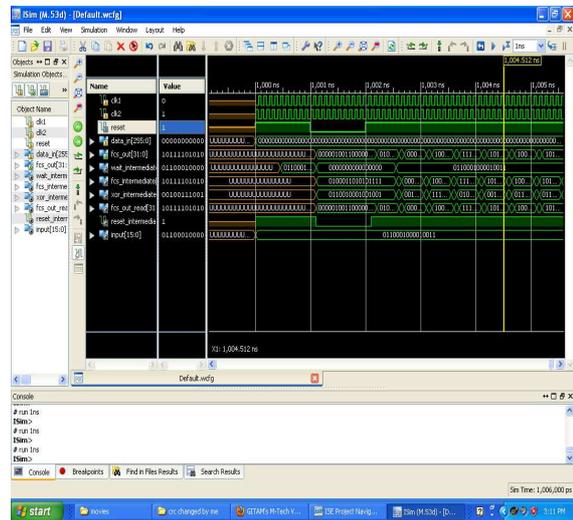
The flexibility of [12]–[14] is limited to configuration options in the design phase but the circuits themselves are not reprogrammable. [10] Allows flexibility by multiplexing between predefined CRC circuits, but does not allow full reconfigurability. [15] Utilizes software, which is likely to be fully reconfigurable in-field; which also represents a high cost in terms of power consumption. Given that CRC computation can account for 29% of the total computational cost in storage area networks [16], the power dissipation of the field programmable CRC circuit, which is less than 6 mW, can be considered very low. Unfortunately, power figures for the other references are not available for comparison.

Although it is difficult to directly compare performance and area parameters for different technologies, some valid comparisons can be made. Comparing [13] with our initial Ethernet polynomial implementation on FPGA, the number of LUTs used are of the same order, so it can be expected that the same trade-offs that apply in our test implementations can also be directly applied here. Furthermore, to add flexibility to [13] would require taking the FPGA offline for reconfiguration, whereas the field programmable CRC circuit can be reconfigured in less than 1 μ s, even on FPGA. In terms of area, the circuit of [10] is approximately 20 times larger than the programmable CRC circuit, with a similar throughout speed. The difference in area cost could be expected to be around 7:1 purely on

account of the different process technologies (350/130 nm). The difference in area cost is therefore comparatively insignificant, given that the programmable CRC has significantly more flexibility. The CRC circuit to be deployed as an add-on instruction in an NPU data-path or as a Standalone CRC accelerator IP core for dedicated network processing SoCs or ASSPs. For these in particular, flexibility, area cost, performance and power dissipation are the primary design considerations, making the proposed CRC architecture an ideal offload engine. In term of throughput performance, the field programmable CRC circuit, at 4.92 Gb/s, outperforms the only other truly reconfigurable option [15] which operates at 3.6 Gb/s. The variable field programmable CRC circuit also has the advantage of a much less costly technology platform, The circuit we have implemented does not support the high line speeds of [14] because this custom circuit (which is not reprogrammable) uses much higher levels of parallelism (64/256-bit). Here we can report the results for 64bits, 128bits, 256bits and corresponding output waveform results are generated. It’s the 64bit waveform.



It’s the 256bit waveform results



It’s the 128 bit waveform results

since a simple comparison shows that their 64-bit implementation is both double the bus width and double the line rate of the reconfigurable CRC circuit— assuming doubling our level of parallelism from 32- to 64-bit would also double the supported line speed, the result is 9.84 Gb/s compared to 10 Gb/s. Given all these considerations, the throughput of the field programmable CRC circuit compares favorably with the state of the art. Apart from the software/processor design, none of the other architectures offer a solution that is flexible in terms of supporting all possible combinations of polynomials with a wide variety

of port sizes. Objectives such as high speed or low cost have been considered in these implementations, but flexibility has not been fully addressed. Some present a degree of flexibility, but only in terms of preselecting variables in the design phase. Considering that versatile network processors have to address a wide range of applications, including frame packet processing, security processing and storage related functions; the alternative options presented give limited flexibility to support all these applications.

III. CONCLUSION

This paper presents the design and implementation of a variable novel field programmable CRC computation circuit that is an ideal IP core for VLSI deployment. The aim was to explore an architecture where all the CRC parameters were fully programmable. This was achieved by deriving an array of processing cells to implement a matrix based computation technique. The circuit uses an embedded configuration controller designed for both standalone and runtime programmability of the CRC circuit. 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, enabling support of throughput rates up to 40 Gb/s at 256-bits.

The tradeoff between flexibility, performance and cost has been taken further than those enabled by traditional heterogeneous architectures based on microprocessor, DSP and FPGA technology. Domain specific and field programmable processing cores, such as the presented CRC circuit, provide tailored flexibility while allowing high performance and low hardware cost. In conjunction with an embedded custom specific configuration controller, the programming task of the logic array is reduced to specific high-level instructions, executed by setting parameter registers.

Complex logic synthesis and place and route functions are not required to programmed the circuit, as is the case with traditional FPGA technologies, consequently real runtime programmability is possible with a much reduced reconfiguration time and cost. Embedded domain specific programmable architectures present an opportunity for enhancing SoC designs that face performance and flexibility issues, as they strive to meet emerging design challenges imposed by ICT convergence.

It is further anticipated that a physical oriented design methodology, such as a data-path compiler [17], can be used to optimize the regular structure of the programmable cell array, which could significantly increase the operational frequency while maintaining a low hardware cost. Such an optimized circuit represents an attractive hard macro for environments requiring low cost hardware flexibility, and in emerging areas such as iSCSI-based SANs, where the flexibility to adopt emerging protocols offers a key advantage to vendors.

REFERENCES

- [1] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "RFC 3720—Internet Small Computer Systems Interface (iSCSI)," RFC 3720, Apr. 2004.
- [2] P. Koopman and T. Charkravarty, "Cyclic Redundancy Code (CRC) polynomial selection for embedded networks," in *Proc. DSN*, pp. 145–154.
- [3] P. Koopman, "32-bit cyclic redundancy codes for internet applications," in *Proc. DSN*, pp. 459–472.
- [4] D. Sarwate, "Computation of cyclic redundancy checks via table lookup," *Commun. ACM*, vol. 31, no. 8, pp. 1008–1013, Aug. 1988.
- [5] D. Feldmeier, "Fast software implementation of error correcting codes," *IEEE Trans. Network.*, vol. 3, no. 6, pp. 640–651, Dec. 1995.
- [6] T. Bi-Pei and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Trans. Commun.*, vol. 40, no. 4, pp. 653–657, Apr. 1992.

- [7] M. Braun, J. Freidich, T. Grun, and J. Lember, "Parallel CRC computation in FPGAs," in *Proc. Workshop Field Program. Logic Appl.*, 1996, pp. 156–165.
- [8] J. H. Derby, "High-speed CRC computation using state-space transformations," in *Proc. Globecom*, Nov. 2001, pp. 166–170.
- [9] M.-D. Shieh, M.-H. Sheu, C.-H. Chen, and H.-F. Lo, "A systematic approach for parallel CRC computations," *J. Inf. Sci. Eng.*, vol. 17, pp. 445–461, 2001.
- [10] T. Henriksson and D. Liu, "Implementation of fast CRC calculation," in *Proc. ASP-DAC*, 2003, pp. 563–564.
- [11] F. Monteiro, A. Dandache, A. M'sir, and B. Lepley, "A fast CRC implementation on FPGA using a pipelined architecture for the polynomial division," in *Proc. ICECS*, 2001, vol. 3, pp. 1231–1234.
- [12] M. Sprachmann, "Automatic generation of parallel CRC circuits," *IEEE Des. Test Comput.*, vol. 18, no. 3, pp. 108–114, May/Jun. 2001.
- [13] G. Campobello, M. Russo, and G. Patanè, "Parallel CRC realization," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1312–1319, Oct. 2003.
- [14] Sarance Technologies, Ottawa, ON, Canada, "CRC-32 for 10 Gbps/ OC192 and 40 Gbps/OC768 Systems," 2006. [Online].
- [15] A. Joglekar, M. Kounavis, and F. Berry, "A scalable and high performance software iSCSI implementation," in *Proc. USENIX FAST*, 2005, pp. 267–280.
- [16] A. Crouch, "Technology developments favor IP storage growth," Communications Technology Lab, Intel, Apr. 2005. [Online]. Available: http://www.snwonline.com/tech_edge/technology_04-11-05.asp?article_id=535
- [17] O. Weiss, M. Gansen, and T. Noll, "A flexible datapath generator for physical oriented design," in *Proc. ESSCIRC*, Villach, Sep. 2001, pp.