# A Novel Programmable Parallel CRC Circuit

Martin Grymel and Steve B. Furber

***Abstract*—A new hardware scheme for computing the transition and control matrix of a parallel cyclic redundancy checksum is proposed. This opens possibilities for parallel high-speed cyclic redundancy checksum circuits that reconfigure very rapidly to new polynomials. The area requirements are lower than those for a realization storing a precomputed matrix. An additional simplification arises as only the polynomial needs to be supplied. The derived equations allow the width of the data to be processed in parallel to be selected independently of the degree of the polynomial. The new design has been simulated and outperforms a recently proposed architecture significantly in speed, area, and energy efficiency.**

***Index Terms*—Cyclic redundancy checksum (CRC), digital logic, error detection, parallel, programmable.**

## I. Introduction

The integrity of data being stored or transmitted over noisy channels can be protected by means of codes for error detection. A widely adopted code is the cyclic redundancy checksum (CRC), first introduced by Peterson and Brown in 1961 [1]. The popularity of CRCs has led to a number of different software and hardware implementations [2], [3]. Speed requirements usually make software schemes impractical and demand dedicated hardware. The generic hardware approach uses an inexpensive linear feedback shift register (LFSR), which assumes serial data input. In the presence of wide data buses, the serial computation has been extended to parallel versions that process whole data words based on derived equations [4]–[7] and on cascading the LFSR [8]. Various optimization techniques have been developed that target resource reduction [9] and speed increase [10], [11].

The error detection capabilities of CRCs depend greatly on the polynomial used [12]–[14]. The performance of a certain polynomial is affected by the data, its length as well as the anticipated error patterns. Different applications might therefore favor different polynomials.

This work is motivated by SpiNNaker [15], a spiking neural supercomputer architecture, which aims to mimic aspects of the functionality of the human brain. The project will incorporate more than a million embedded processors. A single SpiNNaker chip will unite up to 20 ARM968 cores and provide access to a dedicated 1 Gb mobile DDR SDRAM memory. Each processor is accompanied by a DMA controller, managing transfers of neural connectivity data between the processor's local memory and the SDRAM. The data is protected by a CRC, which needs to be calculated without affecting the transfer. Restricting the CRC to a certain polynomial would imply a limitation of the performance as applications, and thus data, in SpiNNaker will vary.

In order to accommodate these requirements, a reconfigurable parallel CRC unit is desired that can easily adapt to new polynomials without slowing down the data communication. There is always a tradeoff between speed and area. In this particular scenario, there are two dimensions to the speed of a CRC calculation unit: the time necessary to process a data word, and the time that is required to reconfigure to a new polynomial.

Manuscript received March 22, 2010; revised June 03, 2010; accepted July 01, 2010. The work of M. Grymel was supported by an Engineering and Physical Sciences Research Council (EPSRC) studentship. The SpiNNaker Project was supported by EPSRC under Grant EP/D07908X/1 and Grant EP/G015740/1.

The authors are with the Advanced Processor Technologies Group, School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. (e-mail: mgrymel@cs.man.ac.uk; sfurber@cs.man.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2010.2058872

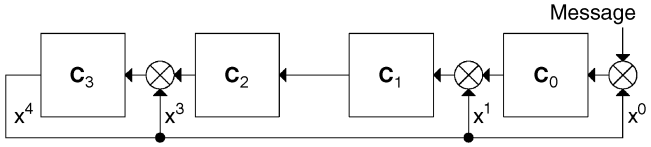


Fig. 1. LFSR for polynomial $P(x) = x^4 + x^3 + x^1 + x^0$.

This work directly extends the parallel CRC realization by Campobello *et al.* [6] based on state space representation in several ways. First, restrictions between the width of the data processed in parallel and the order of the polynomial are lifted. Second, a novel scheme is presented allowing the inexpensive computation of the CRC transition and control matrix in hardware. This leads to a programmable parallel CRC implementation that offers an improved balance between area and both dimensions of speed.

The structure of this paper is as follows. The basic concepts of the CRC are reviewed briefly in Section II. The formula for the parallel CRC realization is then derived in Section III. In Section IV, a new scheme is proposed that transforms the realization into a programmable version. Simulation results and the performance analysis of the new circuit are presented in Section V. Subsequently, the architecture is compared to previous work in Section VI, and conclusions are drawn in Section VII.

## II. CRC

The CRC is a short fixed-length datum (checksum) for an arbitrary data block. It will accompany the data and can be validated at an endpoint through recalculation. Differences between the two CRC values indicate a corruption in either the data or the received CRC itself.

A $k$-bit message can be considered as the coefficients of a polynomial $B(x) = b_{k-1}x^{k-1} + \cdots + b_1x^1 + b_0x^0$. The most significant bit $b_{k-1}$ leads the data stream. Furthermore, an $(m+1)$-bit generator polynomial $P(x) = x^m + p_{m-1}x^{m-1} + \cdots + p_1x^1 + p_0x^0$ of order $m$ is selected. Calculations are performed in modulo-2 arithmetic. The CRC is the remainder $R(x)$ of the division of $x^m B(x)$ by $P(x)$ and will be appended to the message. It can be verified that $x^m B(x) + R(x)$ is divisible by $P(x)$. If the receiver does not agree on this fact, data corruption must have occurred.

The CRC calculation can be realized in hardware with an LFSR in Galois configuration as shown in Fig. 1. The polynomial determines the size and the taps of the shift register. In order to obtain the CRC, the register needs to be cleared in a first step. Then, after injecting the message and $m$ additional zeros, the register will hold the desired CRC.

A receiver can verify the received message with its appended CRC by simply applying the same procedure, with the difference that the CRC will be shifted into the circuit instead of the zeros. If the register finally equals zero, no error has been detected.

The circuit can be modified according to Fig. 2, where the message is combined with the most significant register bit to form the feedback. An advantage arises as no zeros need to be shifted in at the end, and thus the CRC can be obtained $m$ clock cycles earlier. Following Campobello *et al.* [6] this circuit is referred to as LFSR2 in contrast to the first version, which is referred to simply as LFSR.

## III. From Serial to Parallel

Where systems use wide data buses, it is advantageous for the CRC circuits to operate on data words. Many approaches have been made to address this issue. Albertengo and Sisto [5] derived equations in 1990,

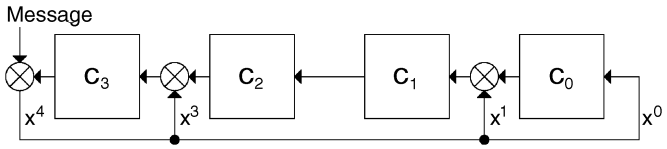


Fig. 2. LFSR2 for polynomial $P(x) = x^4 + x^3 + x^1 + x^0$.

for LFSR2 based on the Z-transform. A simpler method utilizing state space transformation leading to basically the same circuit was published two years later by Pei and Zukowski [4]. In 2003, Campobello *et al.* [6] developed a similar proof for LFSR, under the assumption that the order of the polynomial and the length of the message are both multiples of the number of bits to be processed in parallel, and reported a recursive formula for calculating powers of the state transition matrix.

In this section, the equations for the LFSR2 circuit are derived. The principle of the derivation is very similar to LFSR. Furthermore, the proof is extended in such a way that there will be no restriction on the order $m$ of the polynomial or the number of bits $w$ that are to be processed in parallel. Both parameters are unrelated. It is only assumed that the $k$-bit message can be split into data words of $w$ bits, which will usually be the case in computer systems.

The following considerations are made in Galois Field GF(2) where addition and multiplication correspond to the bitwise XOR and AND operators. As the LFSR2 circuit is a discrete time-invariant linear system, it can be expressed as

$$C[i+1] = FC[i] + Pu[i] \tag{1}$$

where

$$F = \left[ P \left| \frac{I_{m-1}}{0} \right. \right] = \begin{bmatrix} p_{m-1} & 1 & 0 & \cdots & 0 \\ p_{m-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_1 & 0 & 0 & \cdots & 1 \\ p_0 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{2}$$

$I_{m-1}$ denotes the identity matrix of size $m-1$ and $P = [p_{m-1}p_{m-2}\dots p_1 p_0]^T$ is the vector composed of the polynomial coefficients. $C = [c_{m-1}c_{m-2}\dots c_1 c_0]^T$ is the state of the system and corresponds to the CRC register value whereas $u$ is the scalar input. In each time step $i$, one message bit is shifted into the system, and thus $u[0] = b_{k-1}$, $u[1] = b_{k-2}$ and so forth. It can be verified that the solution for system (1) takes the following shape:

$$C[i] = F^i C[0] + [F^{i-1}P \cdots FP\ P]\,[u[0] \cdots u[i-1]]^T \tag{3}$$

with $C[0]$ being the initial state of the CRC register.

A simplified way exists to obtain $F^i$ from $F^{i-1}$

$$\begin{aligned} F^i &= F^{i-1}F \\ &= F^{i-1}\left[ P \left| \frac{I_{m-1}}{0} \right. \right] \\ &= \left[ F^{i-1}P \left| F^{i-1}\frac{I_{m-1}}{0} \right. \right] \end{aligned} \tag{4}$$

where $i$ starts from 2. Expanding $F$ to the power of $w$ with the help of (4) leads to

$$F^w = \begin{cases} \left[ F^{w-1}P \dots FP\ P \left| \frac{I_{m-w}}{0} \right. \right], & \text{if } w \le m \\ [F^{w-1}P \dots F^{w-m}P], & \text{otherwise.} \end{cases} \tag{5}$$

The columns of $F^w$ that drop out in the case where $w$ exceeds $m$ are combined in the auxiliary rectangular matrix

$$F_w := [F^{w-m-1}P \dots FP\ P].$$

In order to obtain the CRC register value after $w$ bits have been processed, $C[w]$ simply needs to be evaluated. For this purpose the $w$-dimensional data input vector $D[t] = [b_{k-1-t} \dots b_{k-w-t}]^T$ is introduced. Then (3) becomes

$$C[w] = F^w C[0] + [F^{w-1}P \cdots FP\ P]D[0]. \tag{6}$$

The following wo basic cases can be differentiated.

**Case $w \le m$:**

$$\begin{aligned} C[w] &= F^w C[0] + \left[ F^{w-1}P \cdots FP\ P \left| \frac{I_{m-w}}{0} \right. \right] \left[ \frac{D[0]}{0} \right] \\ &= F^w C[0] + F^w \left[ \frac{D[0]}{0} \right]. \end{aligned}$$

Considering additionally that the system is time-invariant, the behavior of the circuit can be described as

$$C[t+w] = F^w \left( C[t] + \left[ \frac{D[t]}{0} \right] \right). \tag{7}$$

The special case of $w = m$ leads to the compact form

$$C[t+w] = F^w \left( C[t] + D[t] \right). \tag{8}$$

**Case $w > m$:**

$$C[t+w] = F^w C[t] + [F^w | F_w] D[t]. \tag{9}$$

The result of (7) and (9) can be condensed into a single equation

$$C[t+w] = [F^w | F_w] \left( \left[ \frac{C[t]}{0} \right] + \left[ \frac{D[t]}{0} \right] \right). \tag{10}$$

As an example, polynomial $P(x) = x^4 + x^3 + x^1 + x^0$ is selected. It is intended to process 4 bits in parallel $(w = 4)$. Consequently

$$F = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \qquad F^4 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \tag{11}$$

According to (8), the necessary logic can be directly assembled with the help of (11). Matrix entries of $F^w$ are numbered from $m-1$ to 0, where the top left most element is denoted by $(m-1, m-1)$. Thus, an entry $f_{i,j}$ in matrix $F^w$ indicates that $c_j$ XOR $d_j$ is an input to the XOR forming the new value of $c_i$ one clock cycle later.

## IV. From Static to Programmable

The parallel CRC architecture from the previous section can be transformed into a programmable entity that is no longer bound to a specific CRC polynomial $P(x)$. A polynomial directly affects the transition and control matrix $[F^w | F_w]$ of system (10). Programmability can be achieved by introducing an AND gate with a controlling latch for each signal that may be a potential input to an XOR function as illustrated in Fig. 3. Flipflops can be utilized as well, but will have in general higher demands in terms of area, which may become crucial as $m\,w$ bits need to be stored.

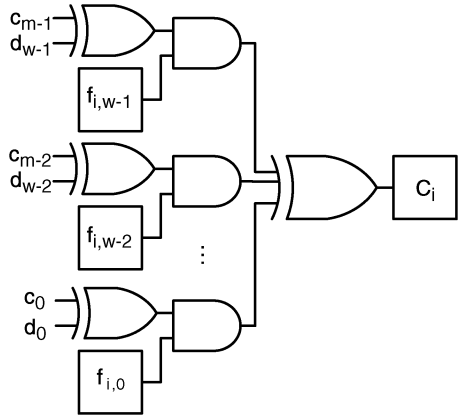


Fig. 3. Programmable parallel CRC circuit with $w = m$ for CRC bit $c_i$ utilizing control latches.

The derivation of the matrix necessary to set up all the latches can be performed in software. As the data bus width imposes a limitation on the transferable data, the matrix may need to be communicated line by line, which requires $m$ clock cycles. Additionally, the software function itself relies on a processor. Many scenarios may even imply a dedicated core for this task, if the polynomial needs to be changed frequently and faster than the matrix can be communicated over the data bus.

### A. Proposed Method

The chosen approach is to outsource the matrix derivation into hardware. As there is no computational effort involved in order to obtain the identity matrix in (5) for the case $w < m$, it is assumed that $w \geq m$, and thus

$$[F^w|F_w] = [F^{w-1}P \ldots FP \; P].$$

A new recursive formula for a column $F^iP$ with $i \geq 1$ is established

$$\begin{aligned} F^iP &= FF^{i-1}P \\ &= \left[P \left| \frac{I_{m-1}}{0} \right.\right] F^{i-1}P \\ &= Pe_1^T F^{i-1}P + \left[0 \left| \frac{I_{m-1}}{0} \right.\right] F^{i-1}P \end{aligned} \tag{12}$$

where $e_1 = [1\,0 \ldots 0]^T$ is the unit vector. Hence, each column element $(F^iP)_j$ can be easily computed with the help of the previous column $F^{i-1}P$ and $P$

$$(F^iP)_j = \begin{cases} p_j(F^{i-1}P)_{m-1} + (F^{i-1}P)_{j-1}, & \text{if } j \neq 0 \\ p_j(F^{i-1}P)_{m-1}, & \text{otherwise.} \end{cases} \tag{13}$$

For an implementation in hardware, an $(m-1)$-bit register needs to be provided to hold the coefficients of the polynomial. This register already forms the right-most column of $[F^w|F_w]$. Each other column can then be obtained with $m$ AND gates and $m-1$ XOR gates. The column element $(F^iP)_0$ of a column $i$ can be obtained through an AND gate that takes as inputs the polynomial coefficient $p_0$, and the column element $(F^{i-1}P)_{m-1}$ of the previous column. For every other element $(F^iP)_j$ of column $i$, polynomial coefficient $p_j$ and $(F^{i-1}P)_{m-1}$ need to be fed into an AND gate, before combining its result with column element $(F^{i-1}P)_{j-1}$ in an XOR gate.

It is possible to reduce the area with equivalent logic as illustrated in Fig. 4, which additionally shows the attached CRC circuitry. Apart from the column storing the polynomial, each other column requires $m-1$ NAND gates, $m-1$ XNOR gates and 1 AND gate. The controlling AND gates have been replaced with NAND gates under the assumption that $w$ is even. Inverting an even number of inputs to an XOR function does not affect the result of the function.

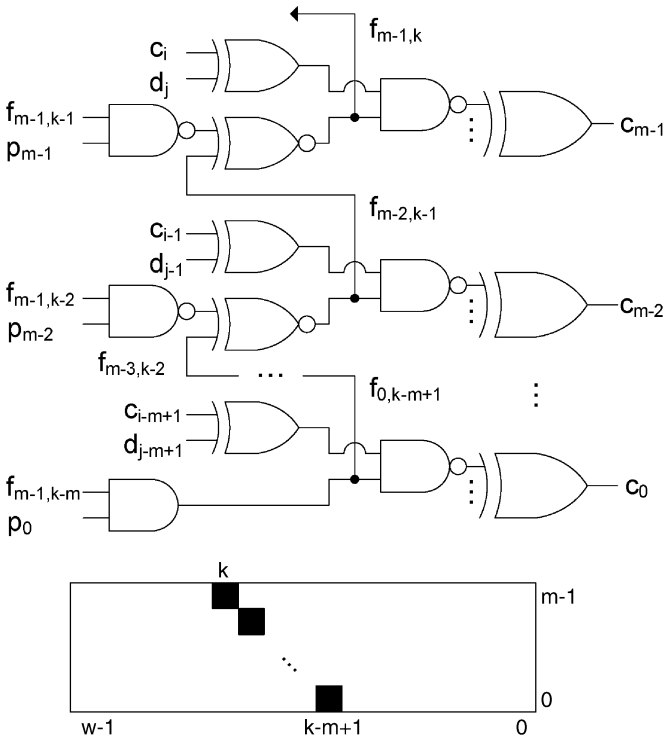


Fig. 4. Programmable parallel CRC circuit for the case $w > m$. Corresponding elements of $[F^w|F_w]$ are indicated with black pixels in the rectangular representing the matrix.

A circuit dimensioned for a certain polynomial degree $m$ can be used to calculate CRCs for a polynomial of smaller degree $n$. This can be achieved by providing the polynomial premultiplied by $x^{m-n}$. Additional multiplexing circuitry is required in order to switch between different data input widths, as the most significant bits of the data $D$ and the state of the system $C$ need to be aligned, when being combined by the bitwise XOR function according to (10).

## V. From Theory to Silicon

With the scheme from the previous section it is possible to replace each latch of the programmable CRC circuit (except for the first column that holds the polynomial) with a NAND and XNOR gate, which can compute the necessary value of the latch. The $w-1$ latches corresponding to the least significant CRC bit can be replaced with only an AND gate. Several 130-nm standard cell libraries indicate a saving of about 6%–7% in logic gate area for a NAND plus XNOR gate in comparison to a latch. Further savings arise from the much smaller AND gate area and irrelevant latch select logic.

In order to assess the performance of the new circuit in Fig. 4, it is necessary to consider two different paths. Assuming that the polynomial $P$ is already set up, the logic gate delay for the data (from $d$ to $c$) adds up to

$$T_{\mathrm{DC}} = (\lceil \log_2 w \rceil + 1)\, T_{\mathrm{XOR}} + T_{\mathrm{NAND}} \tag{14}$$

where $T$ is a logic gate or path delay. Changing the polynomial on the other hand, affects a longer path. The difference between $T_P$ and $T_{\mathrm{DC}}$ accounts for

$$\triangle T_P = T_{\mathrm{LATCH}} + (w-1)(T_{\mathrm{XNOR}} + T_{\mathrm{NAND}}) - T_{\mathrm{XOR}}. \tag{15}$$

TABLE I
IMPLEMENTATION COMPARISON

| | Cell Array [16] | Novel Circuit | Result |
|---|---|---|---|
| Clock Frequency | 154 MHz | 481 MHz | |
| Data Throughput | 4.92 Gbps | 15.38 Gbps | +212.70% |
| Reconfiguration | 33 clock cycles | 4 clock cycles | |
| | 214.29 ns | 8.32 ns | −96.12% |
| Core Area | 0.150 mm$^2$ | 0.033 mm$^2$ | −78.00% |
| Core Utilization | Not specified | 96.13% | |
| Total Power | 5.70 mW | 6.37 mW | |
| Internal Power | 3.42 mW | 3.69 mW | |
| Switching Power | 2.19 mW | 2.67 mW | |
| Leakage Power | 0.0896 mW | 0.0077 mW | |
| Energy[a] | 63 nJ/word | 14 nJ/word | −77.78% |

[a]Based on the setup of a polynomial with a subsequent CRC calculation for 47 data words.

Appending a CRC value to a message will typically require a data stream to stall for at least one clock cycle. Hence, this clock cycle can be used to provide a new polynomial for the subsequent message. This means that the polynomial has two clock cycles to propagate through the entire circuit.

The same behavior can be achieved on the message receiving side. Instead of inputting the received CRC as the last data word into the circuit, and checking the result for 0, the received and calculated CRC value can be compared directly. Again, this would free up an extra clock cycle that can be used to set up a new polynomial.

Without any further clock cycles, the frequency of the circuit would be limited to $f_{\mathrm{worst}} = 1/\max(T_{\mathrm{DC}}, \triangle T_P)$. In order to achieve best case frequency $f_{\mathrm{best}} = 1/T_{DC}$, additional $\lceil \triangle T_P f_{\mathrm{best}} \rceil - 1$ clock cycles would be necessary between messages to propagate $P$. Alternatively, the number of clock cycles can be reduced by providing $s$ columns of matrix $[F^w|F_w]$ instead of only one in order to split up $\triangle T_P$. This requires more area as these columns need to be stored in latches again. In the extreme case the individual paths have a length of $\triangle T_{Pi} = iT_{\mathrm{DC}}$ for $i = 1, \ldots, s$.

The design has been implemented for $m = w = 32$ targeting 130-nm high-speed standard cell technology. The simulation results in Table I were obtained assuming a typical-typical process corner and operating conditions of 1.2 V and 25 °C. These are compared to a previous design [16], which will be discussed in the following section. Better performance is anticipated with a full-custom design that will further exploit the regular structure of the circuit.

## VI. COMPARISON

A programmable parallel CRC architecture was recently proposed [16], which is referred to as the cell array architecture. It incorporates additional circuitry in order to switch between two different data input widths, which is considered in the following critical path and area analysis.

The main component of the cell array is a configurable array of $m \max(m, w)$ cells, each consisting of an XOR, two multiplexers, and a configuration register. A preliminary stage of XOR gates combines data with the current CRC value, which is then fed into the array. Furthermore, a configuration processor is integrated, which performs matrix multiplications in order to obtain the state transition matrix for a provided polynomial. The matrix is transferred row-wise into the configuration registers of the array.

For the basic CRC calculation, both architectures require the same number of two-input XOR gates. The present work however, also allows the utilization of wider and proportionally smaller XOR gates that will assemble $m$ trees each with $w$ inputs.

Considering the logic that is necessary for programmability, each cell in the array constitutes two multiplexers and one register. In the new design, this corresponds to two NAND and an XNOR gate, in $w - 1$ cases only one NAND plus one AND gate, and in $m$ cases one NAND gate and one latch. In all cases this is typically less than for a cell in the cell array design. More area is saved as there is no need for a processor.

The worst case data path in the cell array can be specified as follows:

$$T_{\mathrm{DC2}} = w(T_{\mathrm{XOR}} + T_{\mathrm{MUX}}).$$

This linear growth is inferior to the logarithmic growth of $T_{\mathrm{DC}}$ (14), which has also a reduced scaling factor by $T_{\mathrm{MUX}}$.

The reconfiguration time of the new circuit accounts for $\triangle T_P$ (15). For the cell array, a reconfiguration time of $w + 1$ clock cycles is indicated. The operating frequency of the circuit is limited to $f_{\mathrm{best2}} = 1/T_{\mathrm{DC2}}$. This means that

$$\triangle T_{P2} \approx w(w + 1)(T_{\mathrm{XOR}} + T_{\mathrm{MUX}}).$$

Consequently

$$\frac{\triangle T_{P2}}{\triangle T_P} \in \mathcal{O}(w).$$

This suggests that the new design reconfigures in the order of approximately $w$ times faster than the cell array with the configuration processor.

Both designs have been implemented targeting 130-nm standard cell technology, and are compared in Table I. The new design can be operated at a more than three times higher frequency than the cell array, and has a corresponding increased data throughput. It can reconfigure to a new polynomial 25 times faster than the cell array, while occupying only 22% of its area. Similarly, the energy consumption dropped by about 78%.

An alternative approach in realizing at least partial programmability, is to multiplex between several CRC modules dedicated to fixed polynomials. This method is beneficial if only a few polynomials come into consideration, for which each module can be specifically optimized in terms of speed. Beyond a certain number of different polynomials however, which depends on the polynomials and their realization, the area requirements will exceed those for the proposed architecture. Furthermore, the multiplexing overhead will offset the speed advantage if too many polynomials are involved.

## VII. CONCLUSION

An existing proof [6] for the derivation of parallel CRC circuits has been extended to any polynomial size $m$ and data width $w$. The proof has been conducted for the LFSR2 realization, which avoids inserting a final zero data word.

A simple method has been presented to incorporate programmability into the circuit through latches, allowing the polynomial to be changed during runtime.

Furthermore, a novel scheme has been proposed to compute the state transition and control matrix of the CRC circuit easily in hardware. The scheme is based on a new recursive formula and offers a range of advantages over existing techniques.

First of all, it is only necessary to provide the desired polynomial, instead of a complete matrix for the CRC core. A preliminary matrix calculation in software is no longer required. Second, the logic area requirements are lower than those for a realization that stores the matrix in latches. A recently proposed architecture [16] has significantly higher demands in terms of area as it incorporates a configuration processor, and more core logic in comparison to the latch variant. Third,

the data path grows only logarithmically with $w$ in contrast to the existing architecture where it grows linearly with $w$ with a higher scaling factor. This implies a faster CRC calculation. Most importantly however, the new circuit reconfigures approximately $w$ times faster than the previous circuit.

Implementation figures support the theoretical results showing a significant improvement in speed, area, and energy efficiency.

## REFERENCES

[1] W. Peterson and D. Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, no. 1, pp. 228–235, 1961.

[2] S. Lin and D. J. Costello, *Error Control Coding*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[3] T. Ramabadran and S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, vol. 8, no. 4, pp. 62–75, Aug. 1988.

[4] T.-B. Pei and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Trans. Commun.*, vol. 40, no. 4, pp. 653–657, Apr. 1992.

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

[6] G. Campobello, G. Patane, and M. Russo, "Parallel CRC realization," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1312–1319, Oct. 2003.

[7] 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, no. 3, pp. 445–461, May 2001.

[8] M. Sprachmann, "Automatic generation of parallel CRC circuits," *IEEE Des. Test Comput.*, vol. 18, no. 3, pp. 108–114, May 2001.

[9] M. Braun, J. Friedrich, T. Grün, and J. Lembert, "Parallel CRC computation in FPGAs," in *Proc. 6th Int. Workshop Field-Program. Logic, Smart Appl., New Paradigms Compilers (FPL)*, London, U.K., 1996, pp. 156–165, Springer-Verlag.

[10] C. Cheng and K. Parhi, "High-speed parallel CRC implementation based on unfolding, pipelining, and retiming," *IEEE Trans. Circuits Syst. II, Expr. Briefs*, vol. 53, no. 10, pp. 1017–1021, Oct. 2006.

[11] C. Kennedy and A. Reyhani-Masoleh, "High-speed parallel CRC circuits," in *Proc. 42nd Asilomar Conf. Signals, Syst. Comput.*, Oct. 2008, pp. 1823–1829.

[12] A. Tanenbaum, *Computer Networks*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[13] P. Koopman, "32-bit cyclic redundancy codes for internet applications," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Washington, DC, 2002, pp. 459–468.

[14] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Washington, DC, 2004, pp. 145–154.

[15] S. Furber and A. Brown, "Biologically-inspired massively-parallel architectures—Computing beyond a million processors," in *Proc. 9th Int. Conf. Appl. Concurrency Syst. Des. (ACSD)*, 2009, pp. 3–12.

[16] C. Toal, K. McLaughlin, S. Sezer, and X. Yang, "Design and implementation of a field programmable CRC circuit architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 8, pp. 1142–1147, Aug. 2009.