Parallel CRC Realization

Giuseppe Campobello , Giuseppe Patanè , Marco Russo

Abstract

This paper presents a theoretical result in the context of realizing high speed hardware for parallel CRC checksums. Starting from the serial implementation widely reported in literature, we have identified a recursive formula from which our parallel implementation is derived. In comparison with previous works, the new scheme is faster and more compact and is independent of the technology used in its realization. In our solution, the number of bits processed in parallel can be different from the degree of the polynomial generator. Lastly, we have also developed high level parametric codes that are capable of generating the circuits autonomously, when only the polyonomial is given.

Index Terms

parallel CRC, LFSR, error-detection, VLSI, FPGA, VHDL, digital logic

I. Introduction

Cyclic Redundancy Check (CRC) [1]–[5] is widely used in data communications and storage devices as a powerful method for dealing with data errors. It is also applied to many other fields such as the testing of integrated circuits and the detection of logical faults [6]. One of the more established hardware solutions for CRC calculation is the Linear Feedback Shift Register (LFSR), consisting of a few flip-flops (FFs) and

Giuseppe Campobello is with the Department of Physics, Uniersity of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, ITALY and INFN Section of Catania, 64, Via S.Sofia, I-95123 Catania, ITALY; e-mail: gcampo@ai.unime.it; Tel: +39 (0)90 6765231

Giuseppe Patanè is with the Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, ITALY and INFN Section of Catania, 64, Via S.Sofia, I-95123 Catania, ITALY; e-mail: gpatane@ai.unime.it; Tel: +39 (0)90 6765231

Marco Russo (corresponding author) is with the Department of Physics, University of Catania, 64, Via S.Sofia, I-95123 Catania, ITALY and INFN Section of Catania, 64, Via S.Sofia, I-95123 Catania, ITALY; e-mail: marco.russo@ct.infn.it

logic gates. This simple architecture processes bits serially. In some situations, such as high-speed data communications, the speed of this serial implementation is absolutely inadequate. In these cases, a parallel computation of the CRC, where successive units of w bits are handled simultaneously, is necessary or desirable.

Like any other combinatorial circuit, parallel CRC hardware could be synthetized with only two levels of gates. This is defined by laws governing digital logic. Unfortunately, this implies a huge number of gates. Furthermore, the minimization of the number of gates is an NP-hard optimization problem. Therefore when complex circuits must be realized, one generally use heuristics or seeks customized solutions.

This paper presents a customized, elegant, and concise formal solution for building parallel CRC hardware. The new scheme generalizes and improves previous works. By making use of some mathematical principless, we will derive a recursive formula that can be used to deduce the parallel CRC circuits. Furthermore, we will show how to apply this formula and to generate the CRC circuits automatically. As in modern synthesis tools, where it is possible to specify the number of inputs of an adder and automatically generate necessary logic, we developed the necessary parametric codes to perform the same tasks with parallel CRC circuits. The compact representation proposed in the new scheme provides the possibility of saving hardware significantly and reaching higher frequencies in comparison to previous works. Finally, in our solution, the degree of the polynomial generator, m, and the number of bits processed in parallel, w, can be different.

The article is structured as follows: Sect. II illustrates the key elements of CRC. In Sect. III we summarize previous works on parallel CRCs to provide appropriate background. In Sect. IV we derive our logic equations and present the parallel circuit. In addition, we illustrate the performance by some examples. Finally, in Sect. V we evaluate our results by comparing them with those presented in previous works. The codes implemented are included in appendix.

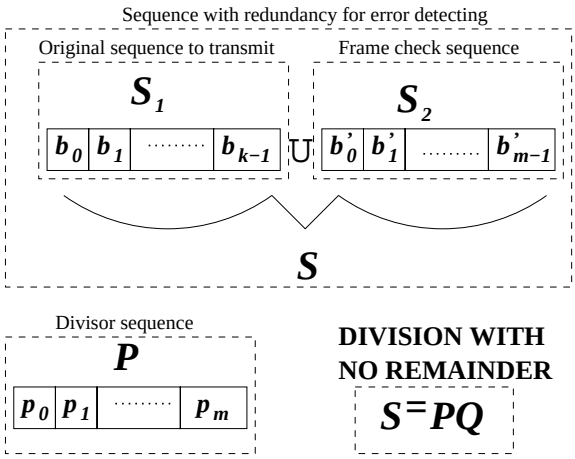

Fig. 1. CRC description. S is the sequence for error detecting, P is the divisor and Q is the quotient. S_1 is the original sequence of k bits to transmit. Finally, S_2 is the FCS of m bits.

II. Cyclic Redundancy Check

As already stated in the introduction, CRC is one of the most powerful error-detecting codes. Briefly speeking, CRC can be described as follows. Let us suppose that a transmitter, **T**, send a sequence, S_1, of k bits $\{b_0, b_1, \cdots, b_{k-1}\}$, to a receiver, **R**. At the same time, **T** generates another sequence, S_2, of m bits $\{b'_0, b'_1, \cdots, b'_{m-1}\}$, to allow the receiver to detect possible errors. The sequence S_2 is commonly known as a Frame Check Sequence (FCS). It is generated by taking into account that the fact that the complete sequence, $S = S_1 \cup S_2$, obtained by concatenating of S_1 and S_2, has the property that it is divisible (following a particular arithmetic) by some predetermined sequence P, $\{p_0, p_1, \cdots, p_m\}$, of m+1 bits. After **T** sends S to **R**. **R** divides S (i.e. the message and the FCS) by P, using the same particular arithmetic, after it receives the message. If there is no remainder, **R** assumes there was no error. Fig. 1 illustrates how this mechanism works.

A modulo 2 arithmetic is used in the digital realization of the above concepts [3]: the product operator is accomplished by a bitwise AND, whereas both the sum and subtraction are accomplished by bitwise XOR operators. In this case, a CRC circuit (modulo 2 divisor) can be easily realized as a special shift register, called LFSR. Fig. 2 shows a typical architecture. It can be used by both the transmitter and the receiver. In the case of the transmitter, the dividend is the sequence S_1 concatenated with a sequence of

m zeros to the right. The divisor is P. In the simpler case of a receiver, the dividend is the received sequence and the divisor is the same P.

In Fig. 2 we show that m FFs have common clock and clear signals. The input x'_i of the ith FF is obtained by taking a XOR of the $(i-1)$th FF output and a term given by the logical AND between p_i and x_{m-1}. The signal x'_0 is obtained by taking a XOR of the input d and x_{m-1}. If p_i is zero, only a shift operation is performed (i.e. XOR related to x'_i is not required); otherwise the feedback x_{m-1} is XOR-ed with x_{i-1}. We point out that the AND gates in Fig. 2 are unnecessary if the divisor P is time-invariant. The sequence S_1 is sent serially to the input d of the circuit starting from the most significant bit, b_0. Let us suppose that the k bits of the sequence S_1 are an integral multiple of m, the degree of the divisor P. The process begins by clearing all FFs. Then, all k bits are sent, once per clock cycle. Finally, m zero bits are sent through d. In the end, the FCS appears at the output end of the FFs.

Another possible implementation of the CRC circuit [7] is shown in Fig. 3. In this paper we will call it LFSR2. In this circuit, the outputs of FFs (after k clock periods) are the same FCS computed by LFSR. It should be mentioned that, when LFSR2 is used, no sequence of m zeros has to be sent through d. So, LFSR2 computes FCS faster than LFSR. In practice, the message length is usually much greater than m; so LFSR2 and LFSR have similar performance.

III. Related Works

Parallel CRC hardware is attractive because, by processing the message in blocks of w bits each, it is possible to reach a speed-up of w with respect to the time needed by the serial implementation. Here we report the main works in literature. Later, in Section V we compare our results with those presented in literature.

- As stated by Albertengo and Sisto [7] in 1990, previous works [8]–[10] *"dealt empirically with the problem of parallel generation of CRCs"*. Furthermore, *"the validity of the results is in any case restricted to a particular generator polynomial"*. Albertengo and Sisto [7] proposed an interesting analytical approach. Their idea was to apply the digital filter theory to the classical CRC circuit.

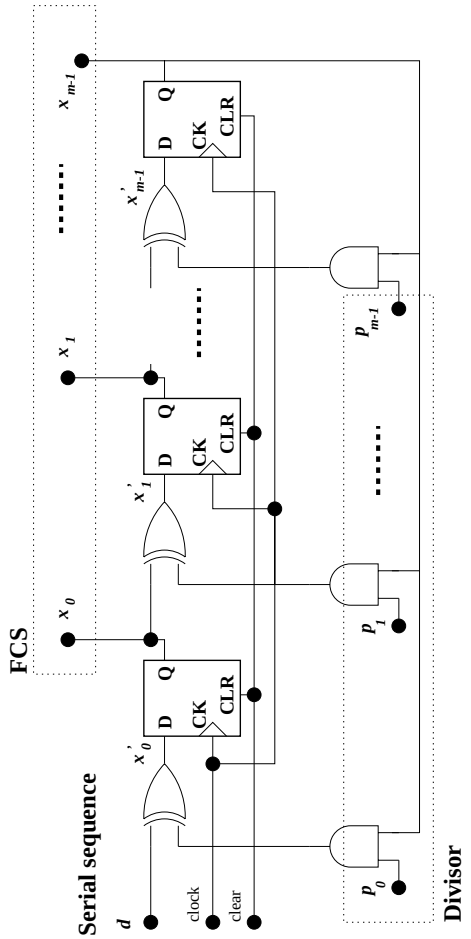

Fig. 2. One of the possible LFSR architectures.

They derived a method for determining the logic equations for any generator polynomial. Their formalization is based on a z-trasform. To obtain logic equations, many polynomial divisions are needed. Thus, it is not possible to write a synthesizable VHDL code that automatically generates the equations for parallel CRCs. The theory they developed is restricted to cases where the number of bits processed in parallel is equal to the polynomial degree ($w = m$).

- In 1996, Braun et al. [11] presented an approach suitable for FPGA implementation. A very complex analytical proof is presented. They developed a special heuristic logic minimization to compute CRC checksums on FPGA in parallel. Their main results are a precise formalism and a set of proofs to derive the parallel CRC computation starting from the bit-serial case. Their work is similar to our work but their proofs are more complex.

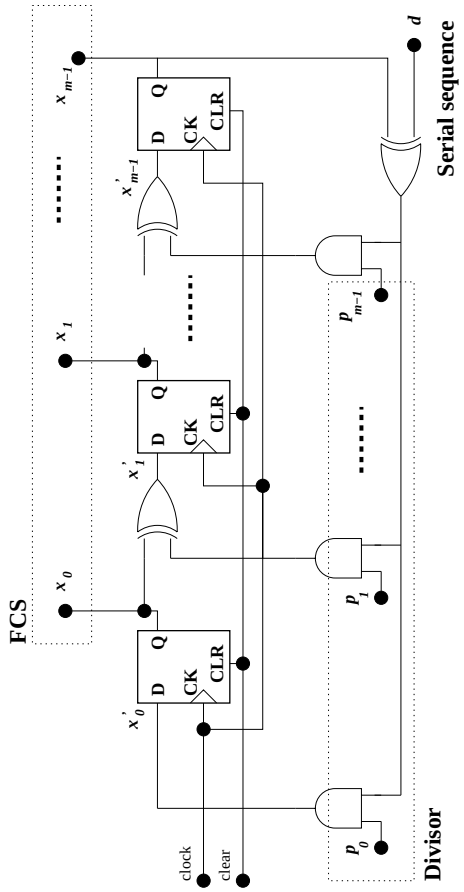

Fig. 3. LFSR2 architecture

- Later, McCluskey [12] developed a high speed VHDL implementation of CRC suitable for FPGA implementation. He also proposed a parametric VHDL code accepting the polynomial and the input data width, both of arbitrary length. This work is similar to ours, but the final circuit has a worse performance.
- In 2001 Sprachmann [13] implemented parallel CRC circuits of LSFR2. He proposed interesting VHDL parametric codes. The derivation is valid for any polynomial and data-width w, but equations are not so optimized.
- In the same year, Shieh et al. [14] proposed another approach based on the theory of the Galois field. The theory they developed is quite general like those presented in our paper (i.e. w may differ from m). Howerver their hardware implementation is strongly based on lookahead techniques [15], [16]; Thus their final circuits require more area and elaboration time. The possibility to use several smaller

look-up tables (LUTs), is also shown, but the critical path of the final circuits grows substantially. Their derivation method is similar to ours but, as in [13], equations are not optimized (see Section V).

IV. Parallel CRC computation

Starting from the circuit represented in Fig. 2 we have developed our parallel implementation of the CRC. In the following, we assume that the degree of polynomial generator (m) and the length of the message to be processed (k) are both multiples of the number of bits to be processed in parallel (w). This is typical in data transmission where a message consists of many bytes and the polynomial generator, as desired parallelism, consist of a few nibbles.

In the final circuit that we will obtain, the sequence S_1 plus the zeros are sent to the circuit in blocks of w bits each. After $\frac{k+m}{w}$ clock periods, the FFs output give the desired FCS.

From linear systems theory [17] we know that a discrete-time, time-invariant linear system can be expressed as follows:

$$\begin{cases} X(i+1) &= FX(i) + GU(i) \\ Y(i) &= HX(i) + JU(i) \end{cases} \tag{1}$$

where X is the state of the system, U the input and Y the output. We use F, G, H, J to denote matrices, and use X, Y, and U to denote column vectors.

The solution of the first equation of the system (1) is:

$$X(i) = F^i X(0) + [F^{i-1}G \cdots FG\ G][U(0) \cdots U(i-1)]^T \tag{2}$$

We can apply eq. (2) to the LFSR circuit (Fig. 2). In fact, if we use $\oplus$ to denote the XOR operation, and the symbol $\cdot$ to denote bitwise AND, and B to denote the set $\{0,1\}$, it is easy to demonstrate that the structure $\{B, \oplus, \cdot\}$ is a ring with identity (Galois Field GF(2) [18]). From this consideration the solution of the system (1)(expressed by (2)) is valid even if we replace multiplication and addition with the AND

and XOR operators respectively. In order to point out that the XOR and AND operators must be also used in the product of matrices, we will denote their product by $\otimes$.

Let us consider the circuit shown in Fig. 2. It is just a discrete-time, time-invariant linear system for which: the input $U(i)$ is the i-th bit of the input sequence; the state X represents the FFs output and the vector Y coincides with X, i.e. H and J are the identity and zero matrices respectively. Matrix F and G are chosen according to the equations of serial LFRS. So, we have:

$$X = [x_{m-1} \cdots x_1 \; x_0]^T$$

$$H = I_m \quad \text{The identity matrix of size } m \times m$$

$$J = [0 \; 0 \; \cdots \; 0]^T$$

$$U = d;$$

$$G = [0 \; 0 \; \cdots \; 1]^T;$$

$$F = \begin{bmatrix} p_{m-1} & 1 & 0 & \cdots & 0 \\ p_{m-2} & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_1 & 0 & 0 & \cdots & 1 \\ p_0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

where p_i are the bits of the divisor P (i.e. the coefficients of the generator polynomial).

When i coincides with w, the solution derived from eq. (2) with substitution of the operators, is:

$$X(w) = F^w \otimes X(0) \oplus [0 \cdots 0 | d(0) \cdots d(w-1)]^T, \tag{3}$$

where $X(0)$ is the initial state of the FFs. Considering that the system is time-invariant, we obtain a recursive formula:

$$X' = F^w \otimes X \oplus D \tag{4}$$

where, for clarity, we have indicated with X' and X, respectively the next state and the present state of the system, and $D = [d_{m-1} \cdots d_1 \; d_0]^T$ assumes the following values: $[0 \cdots 0 | b_0 \cdots b_{w-1}]^T$, $[0 \cdots 0 | b_w \cdots b_{2w-1}]^T$, etc., where b_i are the bits of the sequence S_1 followed by a sequence of m zeros.

This result implies that it is possible to calculate the m bits of the FCS by sending the $k+m$ bits of the message S_1 plus the zeros, in blocks of w bits each. So, after $\frac{k+m}{w}$ clock periods, X is the desired FCS.

Now, it is important to evaluate the matrix F^w. There are several options, but it is easy to show that the matrix can be constructed recursively, when i ranges from 2 to w:

$$F^i = \left[\begin{array}{c|c} F^{i-1} \otimes \begin{bmatrix} p_{m-1} \\ \cdots \\ p_1 \\ p_0 \end{bmatrix} & \begin{array}{c} \text{the first } m\text{-1} \\ \text{columns of } F^{i-1} \end{array} \end{array} \right] \tag{5}$$

This formula permits an efficient VHDL code to be written as we will show later.

From eq. (5) we can obtain F^w when F^m is already available. If we indicate with P' the vector $[p_{m-1} \cdots p_1 \; p_0]^T$ we have:

$$F^w = \left[F^{w-1} \otimes P' \cdots F \otimes P' \; P' \middle| \frac{I_{m-w}}{0} \right] \tag{6}$$

where I_{m-w} is the identical matrix of order $m-w$. Furthermore, we have:

$$F^m = [F^{m-1} \otimes P' | \cdots | F \otimes P' | P'] \tag{7}$$

So, F^w may be obtained from F^m as follows: the first w columns of F^w are the last w columns of F^m. The upper right part of F^w is I_{m-w} and the lower right part must be filled with zeros.

Let us suppose, for example, we have P={1,0,0,1,1}. It follows that:

$$F = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Then, if $w = m = 4$, after applying eq. (5) we obtain:

$$F^4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Finally, if we use $[x'_3\ x'_2\ x'_1\ x'_0]^T$, $[x_3\ x_2\ x_1\ x_0]^T$, and $[d_3\ d_2\ d_1\ d_0]^T$ to denote the three column vectors X', X, and D in equation (4) respectively, we have:

$$x'_3 = x_2 \oplus x_1 \oplus x_0 \oplus d_3$$

$$x'_2 = x_3 \oplus x_2 \oplus d_2$$

$$x'_1 = x_3 \oplus x_2 \oplus x_1 \oplus d_1$$

$$x'_0 = x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus d_0$$

As indicated above, having F^4 available, a power of F of lower order is immediately obtained. So, for example:

$$F^2 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{array}\right]$$

The same procedure may be applied to derive equations for parallel version of LFSR2. In this case the matrix G is $G = P'$; and equation (4) becomes:

$$X' = F^w \otimes (X \oplus D) \tag{8}$$

where D assumes the values $[b_0 \cdots b_{w-1}|0 \cdots 0]^T$, $[b_w \cdots b_{2w-1}|0 \cdots 0]^T$, etc..

A. Hardware realization

A parallel implementation of the CRC can be derived from the above considerations. Yet again, it consists of a special register. In this case the inputs of the FFs are the exclusive sum of some FF outputs and inputs. Fig. 4 shows a possible implementation. The signals $e_{r,c}$ (**Enables**) are taken from F^w. More precisely, $e_{r,c}$ is equal to the value in F^w at the rth row and cth column. Even in the case of Fig. 4, if the divisor P is fixed, then the AND gates are unnecessary. Furthermore, the number of FFs remains unchanged. We recall that if $w < m$ then inputs $d_{m-1} \cdots d_w$ are not needed. Inputs $d_{w-1} \cdots d_0$ are the bits of dividend (Sect. II) sent in groups of w bits each. As to the realization of the LFSR2, by considering eq. (8) we have a circuit very similar to that in Fig. 4 where inputs d are XORed with FF outputs and results are fed back.

In the appendix, the interested reader can find the two listings that generate the correct VHDL code for the CRC parallel circuit we propose here.

Actually, for synthesizing the parallel CRC circuits, using a Pentium II 350 MHz with 64 MB of RAM, less than a couple of minutes are necessary in the cases of CRC-12, CRC-16, and CRC-CCITT. For CRC-32 several hours are required to evaluate F^m by our synthesis tool; We have also written a

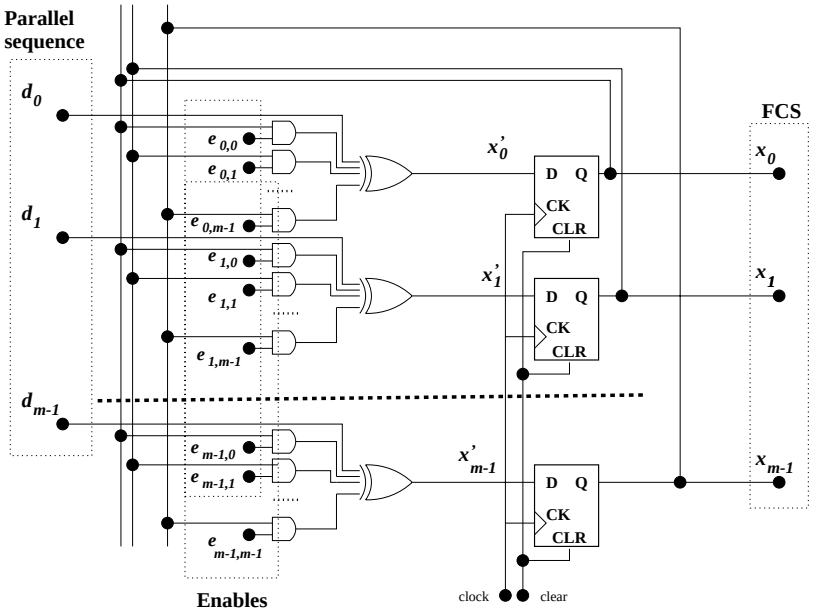

Fig. 4. Parallel CRC architecture

MATLAB code that is able to generate a VHDL code. The code produces logic equations of the desired CRC directly; Thus it is synthesized much faster than the previous VHDL code.

B. Examples

Here, our results, applied to four commonly used CRC polynomial generators are reported. As we stated in the previous paragraph, F^w may be derived from F^m, so we report only the F^m matrix. In order to improve the readability, matrices F^m are not reported as matrices of bits. They are reported as a column vector in which each element is the hexadecimal representation of the binary sequence obtained from the corresponding row of F^m, where the first bit is the most significant. For the example reported above we have $F^4 = [7\ \mathrm{C}\ \mathrm{E}\ \mathrm{F}]^T$.

- **CRC-12:** $P = \{1,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1\}$

  $F^{12}_{\text{CRC-12}} = [\text{CFF 280 140 0A0 050 028 814 40A 205 DFD A01 9FF }]^T$

- **CRC-16:** $P = \{1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1\}$

  $F^{16}_{\text{CRC-16}} = [\text{DFFF 3000 1800 0C00 0600 0300 0180 00C0 0060 0030 0018 000C 8006 4003 7FFE}$

BFFF $]^T$

- **CRC-CCITT:** $P = \{1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1\}$

  $F^{16}_{\text{CRC-CCITT}} = [$0C88 0644 0322 8191 CC40 6620 B310 D988 ECC4 7662 3B31 9110 C888 6444 3222 1911 $]^T$

- **CRC-32:** $P = \{1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1\}$

  $F^{32}_{\text{CRC-32}} = [$FB808B20 7DC04590 BEE022C8 5F701164 2FB808B2 97DC0459 B06E890C 58374486 AC1BA243 AD8D5A01 AD462620 56A31310 2B518988 95A8C4C4 CAD46262 656A3131 493593B8 249AC9DC 924D64EE C926B277 9F13D21B B409622D 21843A36 90C21D1B 33E185AD 627049F6 313824FB E31C995D 8A0EC78E C50763C7 19033AC3 F7011641 $]^T$

V. Comparisons

- Albertengo and Sisto [7] based their formalization on z-transform. In their approach, many polynomial divisions are required to abtain logic equations. This implies that it is not possible to write synthesizable VHDL codes that automatically generate CRC circuits. Their work is based on the LFSR2 circuit. As we have already shown in Sect. IV, our theory can be applied to the same circuit. However eq. (8) shows that, generally speaking, one more level of XOR is required with respect to the parallel LFSR circuit we propose. This implies that our proposal is, generally, faster. Further considerations can be made if FPGA is chosen as the target technology. Large FPGAs are, generally, based on look-up-tables (LUTs). A LUT is a little SRAM which usually has more than two inputs (tipically four or more). In the case of high speed LFSR2 there are many two-input XOR gates (see [7] page 68). This implies that, if the CRC circuit is realized using FPGAs, many LUTs are not completely utilized. This phenomenon is less critical in the case of LFSR. As a consequence, parallel

LFSR realizations are cheaper than LFSR2 ones. In order to give some numerical results to confirm our considerations, we have synthesized the CRC32 in both cases. With LFSR we needed 162 LUTs to obtain a critical path of 7.3 ns, whereas, for LFSR2, 182 LUTs and 10.8 ns are required.

- The main difference between our work and Braun et al's [11] is the dimension of the matrices to be dealt with and the complexity of the proofs. Our results are simpler, i.e., we work with smaller matrices and our proofs are not so complex as those present in [11].
- McCluskey [12] developed a high speed VHDL implementation of CRC suitable for FPGA implementation. The results he obtained are similar to ours (i.e. he started from the LFSR circuit and derived an empirical recursive equation). But he deals with matrices a little greater than the ones we use. Even in this case only XOR are used in the final representation of the formula. Accurate results are reported in the paper dealing with two different possible FPGA solutions. One of them offers us the possibility of comparing our results with those presented by McCluskey. The solution suitable for our purpose is related to the use of the ORCA 3T30-6 FPGA. This kind of FPGA uses a technology of 0.3-0.35μm containing 196 Programmable Function Units (PFUs) and 2436 FFs. Each PFU has 8 LUTs each with 4 inputs and 10 FFs. One LUT introduces a delay of 0.9 ns. There is a flexible input structure inside the PFUs and a total of 21 inputs per PFU. The PFU structure permits the implementation of a 21-input XOR in a single PFU. We have at our disposal the MAX+PLUSII ver.10.0 software[1] to synthesize VHDL using ALTERA FPGAs. We have synthesized a 21-input XOR and have realized that 1 Logic Array Block (LAB) is required, for synthesizing both small and fast circuits. In short, each LAB contains some FFs and 8 LUTs each with 4 inputs. We have synthesized our results regarding the CRC-32. The technology target was the FLEX-10KATC144-1. It is smaller than ORCA3T30. The process technology is 0.35 μm, and the typical LUT delay is 0.9 ns. When $m = w = 32$, the synthesized circuit needed 162 logic cells (i.e., a total of 162 LUTs) with a maximum operating frequency of $\sim$137 MHz. McCluskey, in this case, required 36 PFUs (i.e.,

[1]available at www.altera.com

a total of 288 LUTs) with a speed of 105.8 MHz. So, our final circuit requires less area ($\sim 60\%$) and has greater speed ($\sim 30\%$). The better results obtained with our approach are due to the fact that matrices are different (i.e. different starting equations) and, what is more, McCluskey's matrix is larger.

- We have compiled VHDL codes reported in [13] using our Altera tool. The implementation of our parallel circuit usually requires less area (70-90$\%$) and has higher speed (a speedup of $\sim$4 is achieved). For details see Table I. There are two main motives that explain these results: the former is the same mentioned at the beginning of this section regarding the differences between LFSR and LFSR2 FPGA implementation. The latter is that our starting equations are optimized. More precisely, in our equation of x_i', term x_j appears only once or not at all, while, in the starting equation of [13] x_j may appear more (up to m times), as it is possible to observe in Fig.4 in [13]. Optimizations like $x_j \oplus x_j = 0$ and $x_j \oplus x_j \oplus x_j = x_j$ must be processed from a synthesis tool. When m grows, many expressions of this kind are present in the final equations. Even if very powerful VHDL synthesis tools are used, it is not sure that they are able to find the most compact logical form. Even when they are able to, more synthesis time is necessary with respect to our final VHDL code.
- In [14] a detailed performance evaluation of the CRC-32 circuit is reported. For comparison purposes we take results from [14] when $m = w = 32$. For the matrix realization they start from a requirement of 448 2-input XORs and a critical path of 15 levels of XORs. After Synopsys optimization they obtain 408 2-input XORs and 7 levels of gates. We evaluated the number of required 2-input XORs starting from the matrix F^w, counting the ones and supposing the realization of XORs with more than two inputs with binary tree architectures. So, for example, to realize a 8-input XOR, 3 levels of 2-input XORs are required with a total of 7 gates. This approach gives 452 2-input XORs and only 5 levels of gates before optimization. This implies that our approach produces faster circuits, but the circuits are a little bit larger. However, with some simple manual tricks it is possible to obtain hardware savings. For example, identifying common XOR sub-expressions and realizing them only

TABLE I

PERFORMANCE EVALUATION FOR A VARIETY OF PARALLEL CRC CIRCUITS. RESULTS ARE FOR $w = m$. LUTS IS THE NUMBER OF LOOK-UP TABLES USED IN FPGA; CPD IS THE CRITICAL PATH DELAY; t_1 AND t_2 ARE THE SYNTHESIS TIMES, THE FIRST TIME REFERS TO THE CRCGEN.VHD AND THE SECOND TIME REFERS TO THE VHDL GENERATED WITH MATLAB. FOR [13] t_1 IS THE SYNTHESIS TIME OF THE VHDL CODE REPORTED IN THIS PAPER USING OUR SYNTHESIS TOOL.

Polynomial	LUTs	CPD	t_1,t_2
CRC12 [our]	21	6 ns	20,7
CRC12 [13]	27	24.2 ns	7
CRC16 [our]	28	7.2 ns	100,8
CRC16 [13]	31	28.1 ns	9
CRC-CCITT [our]	39	6.9 ns	113,8
CRC-CCITT [13]	30	10.2 ns	10
CRC32 [our]	162	7.3 ns	long,15
CRC32 [13]	220	30.5 ns	360

once, the number of required gates decreases to 370. With other smart tricks it is possible to obtain more compact circuits. We do not have at our disposal the Synopsys tool, so we do not know which is the automatic optimization achievable starting from the 452 initial XORs.

VI. ACKNOWLEDGMENTS

The authors wish to thank Chao Yang of the ILOG, Inc. for his useful suggestions during the revision process. The authors wish also to thank the anonimous reviewers for their work, a valuable aid that contributed to the improvement of the quality of the paper.

APPENDIX

A. VHDL code

In Figs. 5 and 6 we present two VHDL listings, named respectively *crcpack.vhd* and *crcgen.vhd*. They are the codes we developed describing our parallel realization of both LFSR and LFSR2 circuits. It is necessary to assign to the constant CRC the divisor P, in little endian form; to CRCDIM the value of

```
library ieee;
use ieee.std_logic_1164.all;
package crcpack is
constant CRC16: std_logic_vector(16 downto 0):=
            "11000000000000101";
constant CRCDIM: integer:=16;
constant CRC: std_logic_vector(CRCDIM downto 0):=
                CRC16;
constant DATA_WIDTH: integer range 1 to CRCDIM:=16;
type matrix is array(CRCDIM-1 downto 0) of
     std_logic_vector(CRCDIM-1 downto 0);
end crcpack;
```

Fig. 5. The package: *crcpack.vhd*

```
library ieee;
use ieee.std_logic_1164.all; use work.crcpack.all;

entity crcgen is
port(res,clk: std_logic;
  Din: std_logic_vector(DATA_WIDTH-1 downto 0);
  Xout: out std_logic_vector(CRCDIM-1 downto 0));
end crcgen;

architecture rtl of crcgen is
signal X,X1,X2,Dins:
  std_logic_vector(CRCDIM-1 downto 0);
begin

process(Din)
variable Dinv:std_logic_vector(CRCDIM-1 downto 0);
begin
  Dinv:=(others=>'0');
  Dinv(DATA_WIDTH-1 downto 0):=Din; --LFSR
  -- LFSR2
  -- Dinv(CRCDIM-1 downto CRCDIM-DATA_WIDTH):=Din;
  Dins<=Dinv;
end process;
X2<=X; --LFSR
-- X2<=X xor Dins; --LFSR2

process(res,clk)
begin
  if res='0' then X<=(others=>'0');
  elsif rising_edge(clk) then X<=X1 xor Dins;--LFSR
                        -- then X<=X1; --LFSR2
  end if;
end process;
Xout<=X;

-- This process build matrix M=F^w
process(X2)
variable Xtemp,vect,vect2:
  std_logic_vector(CRCDIM-1 downto 0);
variable M,F: matrix;
begin
-- Matrix F
F(0):=CRC(CRCDIM-1 downto 0);
for i in 0 to CRCDIM-2 loop
  vect:=(others=>'0'); vect(CRCDIM-i-1):='1';
  F(i+1):=vect;
end loop;
-- Matrix M=F^w
M(DATA_WIDTH-1):=CRC(CRCDIM-1 downto 0);
for k in 2 to DATA_WIDTH loop
vect2:=M(DATA_WIDTH-k+1); vect:=(others=>'0');
for i in 0 to CRCDIM-1 loop
  if vect2(CRCDIM-1-i)='1' then vect:=vect xor F(i);
  end if;
end loop;
M(DATA_WIDTH-k):=vect;
end loop;
for k in DATA_WIDTH-1 to CRCDIM-1 loop
  M(k):=F(k-DATA_WIDTH+1);
end loop;
-- Combinational logic equations: X1 = M (x) X
Xtemp:=(others=>'0');
for i in 0 to CRCDIM-1 loop
  vect:=M(i);
  for j in 0 to CRCDIM-1 loop
    if vect(j)='1' then
    Xtemp(j):=Xtemp(j) xor X2(CRCDIM-1-i);
    end if;
  end loop;
end loop;
X1<=Xtemp;
end process;
end rtl;
```

Fig. 6. The code: *crcgen.vhd*

```
function [S,Fk,Dimxor,FH]=crcgen(CRC,DATAWIDTH,opt)
CRC16=[1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1];

S='-- CRC= ';
if (isstr(CRC)==0), p=CRC; conver=1;
elseif ((findstr(CRC,'CRC')==1) & exist(CRC)),
   conver=1; p=eval(CRC);
elseif (exist('sym2poly')==2),
   p=sym2poly(CRC); conver=0;
else error('There is not Symbolic toolbox');
end;

n=size(p,2);
if conver==1,
   for i=1:n,
      if p(1,i)==1,S=[S,'x^',num2str(n-i),' + '];end;
   end;
    S=S(1:size(S,2)-3);
else S=[S,CRC];
end;

%First bit of CRC is not important: it is always 1
P=p(2:n); n=n-1; F=zeros(n); F(1:n,1)=P';
F(1:n-1,2:n)=eye(n-1); k=DATAWIDTH; Fk=rem(F^k,2);
for i=1:n,
   str=['X1(',num2str(n-i),')<= '];
   for j=1:n,
      if Fk(i,j)==1,
        str=[str,'X2(',num2str(n-j),') xor '];
      end;
   end;
   str=[str(1:size(str,2)-5),';'];
   S=str2mat(S,str);
   Dimxor(i)=nnz(Fk(i,:));
end;

['Maximal number of xor input is ',...
    num2str(max(Dimxor))]

for i=1:n,
   Dec(i)=0;
   for j=1:n,Dec(i)=Dec(i)+2^(n-j)*Fk(i,j); end;
   zerofill=zeros(n/4-size(dec2hex(Dec(i)),2),1)';
   str=[num2str(zerofill) dec2hex(Dec(i))];
   FH(i,:)=str;
end;

if (nargin==3),
if (strcmp(opt,'vhdl')==1),
%Make VHDL-file named crcgen.vhd
echo off;
Package=str2mat('package crcpack is',...
['constant CRCDIM: integer:=',num2str(n),';'],...
['constant DATA_WIDTH: integer:=',num2str(k),';'],...
'end crcpack;');
if exist('crcgen.vhd'), delete crcgen.vhd;end;
diary crcgen.vhd
disp(Package)
type crcgen.txt
disp(S)
disp('end rtl;')
diary off;
echo on;
end;
end;
```

Fig. 7. The matlab file: *crcgen.m*

m; and to DATAWIDTH the desired processing parallelism (w value). The codes reported are ready to generate the hardware for the CRC-16 where $m = w = 16$.

B. Matlab code

In Fig. 7 we report the Matlab code, named *crcgen.m*, used to directly produce the VHDL listing of the desired CRC where only the logical equations of the CRC are present. This code is synthesized much faster than the previous one. In order to work correctly, the *crcgen.m* file needs another file, named *crcgen.txt*; this file contains the first 34 rows of *crcgen.vhd*.

REFERENCES

[1] W.W.Peterson, D.T.Brown, "Cyclic Codes for Error Detection," *Proc. IRE*, Jan. 1961.

[2] A.S.Tanenbaum, *Computer Networks*. Prentice Hall, 1981.

[3] W.Stallings, *Data and Computer Communications*. Prentice Hall, 2000.

[4] T.V. Ramabadran and S.S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, Aug. 1988.

[5] N.R.Sexana, E.J.McCluskey, "Analysis of Checksums, Extended Precision Checksums and Cyclic Redundancy Checks," *IEEE Transactions on Computers*, July 1990.

[6] M.J.S.Smith, *Application-Specific Integrated Circuits*. Addison-Wesley Longman, Jan. 1998.

[7] G.Albertengo, R.Sisto, "Parallel CRC Generation," *IEEE Micro*, Oct. 1990.

[8] R.Lee, "Cyclic Codes Redundancy," *Digital Design*, July 1977.

[9] A.Perez, "Byte-wise CRC Calculations," *IEEE Micro*, June 1983.

[10] A.K.Pandeya, T.J.Cassa, "Parallel CRC Lets Many Lines Use One Circuit," *Computer Design*, Sept. 1975.

[11] M.Braun et al., "Parallel CRC Computation in FPGAs," in *Workshop on Field Programmable Logic and Applications*, 1996.

[12] J.McCluskey, "High Speed Calculation of Cyclic Redundancy Codes," in *Proc. of the 1999 ACM/SIGDA seventh Int. Symp. on Field Programmable Gate Arrays*, p. 250, ACM Press New York, NY, USA, 1999.

[13] M.Spachmann, "Automatic Generation of Parallel CRC Circuits," *IEEE Design & Test of Computers*, May 2001.

[14] M.D.Shieh et al., "A Systematic Approach for Parallel CRC Computations," *Journal of Information Science and Engineering*, May 2001.

[15] G.Griffiths and G.Carlyle Stones, "The Tea-Leaf Reader Algorithm: An Efficient Implementation of CRC-16 and CRC-32," *Communications of the ACM*, July 1987.

[16] D.V. Sarwate, "Computation of Cyclic Redundancy Checks via Table Look-Up," *Communications of the ACM*, Aug. 1988.

[17] Gene F.Franklin et al., *Feedback Control of Dynamic Systems*. Addison Wesley, 1994.

[18] J.Borges and J.Rifá, "A Characterization of 1-Perfect Additeive Codes," *Pirdi-1/98*, 1998.

Giuseppe Campobello was born in Messina (Italy) in 1975. He received the "Laurea" (MS) degree in Electronic Engineering from the University of Messina (Italy) in 2000. Since 2001 he has been a Ph.D. student in Information Technology at the University of Messina (Italy). His primary interests are reconfigurable computing, VLSI design, micropocessor architectures, computer networks, distributed computing and soft computing. Currently, he is also associated at the National Institute for Nuclear Physics (INFN), Italy.

Giuseppe Patanè was born in Catania (Italy) in 1972. He received the “Laurea” and the Ph.D. in Electronic Engineering from the University of Catania (Italy) in 1997 and the University of Palermo (Italy) in 2001, respectively. In 2001, he joined the Department of Physics at the University of Messina (Italy) where, currently, he is a Researcher of Computer Science. His primary interests are soft computing, VLSI design, optimization techniques and distributed computing. Currently, he is also associated at the National Institute for Nuclear Physics (INFN), Italy.

Marco Russo was born in Brindisi in 1967. He received the M.A. and Ph.D. degrees in electronical engineering from the University of Catania, Catania, Italy, in 1992 and 1996. Since 1998, he has been with the Department of Physics, University of Messina, Messina, Italy as an Associate Professor of Computer Science. His primary interests are soft computing, VLSI design, optimization techniques and distributed computing. He has more than 90 technichal publications appearing in internationl journals, books and conferences. He is coeditor in the book *Fuzzy Learning and Applications* (CRC Press, Boca Raton, FL). Currently, he is in charge of Research at the National Institute for Nuclear Physics (INFN).