International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 3, March - 2013

Fpga Based High Speed Parallel Cyclic Redundancy Check

P. Harika [1], B. V. V .Satyanarayana [2]

[1][2]BVC Engineering College, Odalarevu, Dept.Of.Electronics and Communication Engineering

Andhra Pradesh-533210, INDIA

ABSTRACT

Error correction codes provides a mean to detect and correct errors introduced by the transmission channel.

This paper presents a high-speed parallel cyclic redundancy check (CRC) implementation based on unfolding, pipelining, and retiming algorithms. CRC architectures are first pipelined to reduce the iteration bound by using novel look-ahead pipelining methods and then unfolded and retimed to design high-speed parallel circuits. The study and implementation using Verilog HDL. Modelsim Xilinx Edition (MXE) will be used for simulation and functional verification. Xilinx ISE will be used for synthesis and bit file generation. The Xilinx Chip scope will be used to test the results on Spartan 3E 500K FPGA board.

KEYWORDS*: CRC, LFSR, Pipelining, Retiming, Unfolding, ISE, MXE, FPGA*

I.INTRODUCTION

A CRC (Cyclic Redundancy Check) is a popular error detecting code computed through binary polynomial division. To generate a CRC, the sender treats binary data as a binary polynomial and performs the modulo-2 division of the polynomial by a standard generator (e.g., CRC-32). The remainder of this division becomes the CRC of the data, and it is attached to the original data and transmitted to the receiver. Receiving the data and CRC, the receiver also performs the modulo-2 division with the received data and the same generator polynomial. Errors are detected by comparing the computed CRC with the received one. The CRC algorithm only adds a small number of bits (32 bits in the case of CRC-32) to the message regardless of the length of the original data, and shows good performance in detecting a single error as well as an error burst. Because the CRC algorithm is good at detecting errors and is simple to implement in hardware, CRCs are widely used today for detecting corruption in digital data which may have occurred during production, transmission, or storage. And CRCs have recently found a new application in universal mobile telecommunications system standard for message length detection of variable-length message communications.

In this paper, we present a fast cyclic redundancy check (CRC) algorithm that performs CRC computation for any length of message in parallel. For a given message with any length, the algorithm first chunks the message into blocks, each of which has a fixed size equal to the degree of the generator polynomial. Then it performs CRC computation using only lookup tables among the chunked blocks in parallel and the results are combined together by XOR operations. It was feedback in the traditional implementation that makes pipelining problematic. In the proposed algorithm, we solve this problem and implement a pipelined calculation of 32-bit CRC in SMIC 0.13 m CMOS technology. Our algorithm allows calculation over data that is not the full width of the input. Furthermore,

the pipeline latency is very short in our algorithm, and this method allows easy scaling of the parallelism while only slightly affecting timing.

Traditionally, the LFSR (Linear Feedback Shift Register) circuit is implemented in VLSI (Very-Large-Scale Integration) to perform CRC calculation which can only process one bit per cycle. Recently, parallelism in the CRC calculation becomes popular, and typically one byte or multiple bytes can be processed in parallel.A Common method used to achieve parallelism is to unroll the serial implementation. Unfortunately, the algorithms used for parallelism increase the length of the worst case timing path, which falls short of ideal speedups in practice. Furthermore, the required area and power consumption increases with the higher degree of parallelism. Therefore, we seek an alternative way to implement CRC hardware to speed up the CRC calculation while maintaining reasonable area and power consumption.

In summary, this paper proposes a hardware architecture for calculating CRC that offers a number of benefits. First of all, it calculates the CRC of a message in parallel to achieve better throughput. It does not use LFSRs and does not need to know the total length of the message before beginning the CRC calculation. While the algorithm is based on lookup tables, it adopts multiple small tables instead of a single large table so that the overall required area remains small.

A General linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The only linear function of single bits is xor, thus it is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle.

In computing, a **pipeline** is a set of data processing elements connected in series, so that the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements.

Retiming is the technique of moving the structural location of latches or registers in a digital circuit to improve its performance, area, and/or power characteristicscs in such a way that preserves its functional behavior at its outputs.[1]

Unfolding is a transformation technique of duplicating the functional blocks to increase the throughput of the DSP program in such a way that preserves its functional behavior at its outputs. Unfolding in general program is as known as Loop unrolling. Unfolding has applications in designing high-speed and low-power ASIC architectures. One application is to unfold the program to reveal hidden concurrency so that the program can be scheduled to a smaller iteration period, thus increasing the throughput of the implementation. Another application is parallel processing in word level or bit level. Therefore these transformed circuit could increase the throughput and decrease the power consumption.

II CYCLIC REDUNDANCY CHECK

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match. CRCs are so called because the *check* (data verification) value is a *redundancy* (it adds no information to the message) and the algorithm is based on *cyclic* codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

Cyclic codes are not only simple to implement but have the benefit of being particularly well suited for the detection of burst errors, contiguous sequences of erroneous data symbols in messages. This is important because burst errors are common transmission errors in many communication channels, including magnetic and optical storage devices. Typically an *n*-bit CRC applied to a data block of arbitrary length, will detect any single error burst not longer than *n* bits and will detect a fraction $1-2^{-n}$ of all longer error bursts. [7]

Specification of a CRC code requires definition of a so-called generator polynomial. This polynomial resembles the divisor in a polynomial long division, which takes the message as the dividend and in which the quotient is discarded and the remainder becomes the result, with the important distinction that the polynomial coefficients are calculated according to the carry-less arithmetic of a finite field. The length of the remainder is always less than the length of the generator polynomial, which therefore determines how long the result can be.

In practice, all commonly used CRCs employ the finite field GF(2). This is the field of two elements, usually called 0 and 1, comfortably matching computer architecture. The rest of this article will discuss only these binary CRCs, but the principles are more general. The simplest error-detection system, the parity bit, is in fact a trivial 1-bit CRC: it uses the generator polynomial $x+1$.

There are several techniques for generating check bits that can be added to a message. Perhaps the simplest is to append a single bit, called the "parity bit," which makes the total number of 1-bits in the code vector (message with parity bit appended) even (or odd). If a single bit gets altered in transmission, this will change the parity from even to odd (or the reverse). The sender generates the parity bit by simply summing the message bits modulo 2—that is, by exclusive oring them together. It then appends the parity bit (or its complement) to the message. The receiver can check the message by summing all the message bits modulo 2 and checking that the sum agree s with the parity bit. Equivalently, the receiver can sum all the bits (message and parity) and check that the result is 0 (if even parity is being used). This simple parity technique is often said to detect 1-bit errors. Actually it detects errors in any odd number of bits (including the parity bit), but it is a small comfort to know you are detecting 3-bit errors if you are missing 2-bit errors as shown in Fig1

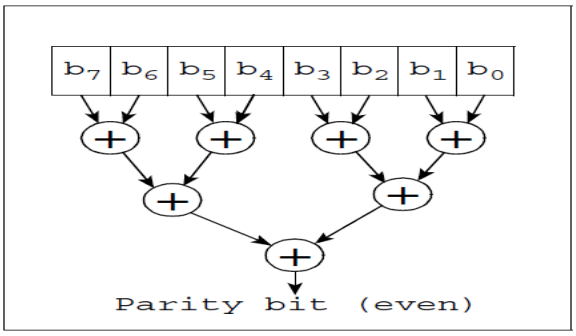


FIG 1. Exclusive or tree.

The CRC is based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial in GF(2) (Galois field with two elements) by another. It is a little like treating the message as a very large binary number, and computing the remainder on dividing it by a fairly large prime such as Intuitively, one would expect this to give a reliable checksum. A polynomial in GF(2) is a polynomial in a single variable x whose coefficients are 0 or 1. Addition and subtraction are done modulo 2—that is, they are both the same as the exclusive or operator. For example, the sum of the polynomials

X^3+X+1

$X^4+X^3+X^2$

is as is their difference. These polynomials are not usually written with minus signs, but they could be, because a coefficient of –1 is equivalent to a coefficient of 1. Multiplication of such polynomials is straightforward. The product of one coefficient by another is the same as their combination by the logical and operator, and the partial products are summed using exclusive or. Multiplication is not needed to compute the CRC checksum.

Division of polynomials over GF(2) can be done in much the same way as long division of polynomials over the integers.

$$
\begin{array}{r}
x^4 + x^3 + 1 \\
x^3+x+1\,\overline{)\,x^7 + x^6 + x^5 + \qquad\quad x^2 + x} \\
\underline{x^7 + \qquad x^5 + x^4} \\
\qquad x^6 + \qquad x^4 \\
\qquad \underline{x^6 + \qquad x^4 + x^3} \\
\qquad\qquad\qquad x^3 + x^2 + x \\
\qquad\qquad\qquad \underline{x^3 + \qquad\quad x + 1} \\
\qquad\qquad\qquad\qquad x^2 + \quad 1
\end{array}
$$

The reader might like to verify that the quotient of $x4 + x3 + 1$ multiplied by the divisor $x3 + x + 1$,of plus the remainder of $x2 + 1$,equals the dividend. For bit serial sending and receiving, the hardware to generate and check a single parity bit is very simple. It consists of a single exclusive or gate together with some control circuitry. For bit parallel transmission, an exclusive or tree may be used, as illustrated in Figure Efficient ways to compute the parity bit in software are given. Other techniques for computing a checksum are to form the exclusive or of all the bytes in the message, or to compute a sum with end-around carry of all the bytes. In the latter method the carry from each 8-bit sum is added into the least significant bit of the accumulator. It is believed that this is more likely to detect errors than the simple exclusive or, or the sum of the bytes with carry discarded. A technique that is believed to be quite good in terms of error detection, and which is easy to implement in hardware, is the cyclic redundancy check. This is another way to compute a checksum, usually eight, 16, or 32 bits in length, that is appended to the message. We will briefly review the theory and then give some algorithms for computing in software a commonly used 32-bit CRC checksum. [7]

III MATHEMATICAL BACKGROUND

We shall first introduce the binary field and binary polynomials that facilitate the definition of cyclic redundancy codes. In simple terms, a field is an algebraic system in which the operation of addition, subtraction, multiplication and division can be performed. The set of real numbers for example forms a field. Fields can be finite or infinite. The smallest finite field is the binary field that has just two elements denoted usually by 0 and 1.The table below defines the addition and multiplication operations in this field.

+	0	1
0	0	1
1	1	0

(a)

·	0	1
0	0	0
1	0	1

(b)

FIG 2. Addition and multiplication tables

From the addition table, we see that an EXOR gate is all that we need to perform the addition operation in the binary field. Moreover we see 0 and 1 to be their own additive inverses, and so subtraction in the binary field is the same as addition. Multiplication in the binary field can be performed simply by means of a AND gate. we must define division in this field single non zero element 1,and we do this trivially by noting that division by 1 leaves both 0 and 1 unchanged. [7]

A binary polynomial is a polynomial with coefficients from the binary field. For example, 0,1, x,1+x,x^2,1+x+x^2,are all binary polynomials in the dummy variable x. Given any sequence of bits, we can associate a binary polynomial with it by regarding the different bits representing the coefficients of the polynomial. For instance, with the sequence 101011, we can associate the fifth degree polynomial $1.x^0+0.x^1+1.x^2+0.x^3+1.x^4+1.x^5=1+x^2+x^4+x^5$.

According to the convention used here, the rightmost bit of a sequence represents the coefficients of the highest degree terms of the associated polynomial. A left-to-right shift sequence of bits by i positions (with the vacated positions filled with 0's), therefore, corresponds to multiplying the associated polynomial by xi. We perform operations involving binary polynomials in exactly the same manners we do with ordinary polynomials with real number coefficients. However, we manipulate the coefficients using the rules of the binary field.

IV LINEAR FEEDBACK SHIFT REGISTER

Register (LFSR) structures are widely used in digital signal processing and communication systems, such as BCH, CRC. Many current functions such as Scrambling, Convolutional Coding, CRC and even Cordic or Fast Fourier Transform can be derived as Linear Feedback Shift Registers (LFSR) In high-rate digital systems such as optical communication system, throughput of 1Gbps is usually desired. The serial input/output operation property of LFSR structure is a bottleneck in such systems and parallel LFSR architecture is thus required.This work presents a three-step high-speed VLSI architecture for LFSR structures, this paper proposes improved three-step LFSR architecture with both higher hardware efficiency and speed. This architecture can be applied to any LFSR structure for high-speed parallel implementation.[1]

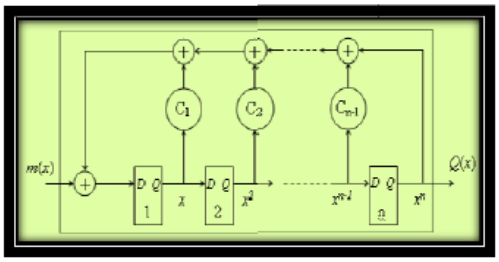


FIG 3 A General LFSR based CRC circuit

In computing, a **linear feedback shift register** (LFSR) is a shift register whose input bit is a linear function of its previous state.The most commonly used linear function of single bits is XOR. Thus, an LFSR is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value.

The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it musteventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle.

The mathematics of a cyclic redundancy check, used to provide a quick check against transmission errors, are closely related to those of an LFSR. [7]

LFSR IN CRC

In traditional hardware implementations, a simple circuit based on shift registers performs the CRC calculation by handling the message one bit at a time. A typical serial CRC using LFSRs is shown in Fig. 3.2.It illustrates one possible structure for CRC32. There are a total of 32 registers the middle ones are left out for brevity. The combinational logic operation in the figure is the XOR operation. One bit is shifted in at each clock pulse. This circuit operates in a fashion similar to manual long division. The XOR gates in Fig 4 hold the coefficients of the divisor corresponding to the indicated powers of x. Although the shift register approach to computing CRCs is usually implemented in hardware, this algorithm can also be used in software when bit-by-bit processing is adequate.

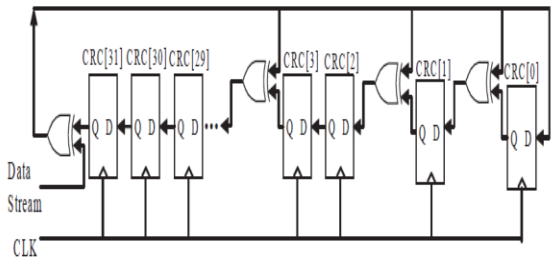


FIG 4 Serial CRC circuit using LFSRs

The proposed design starts from LFSR, which is generally used for serial CRC. An unfolding algorithm is used to realize parallel processing. However, direct application of unfolding may lead to a parallel CRC circuit with long iteration bound, which is the lowest achievable CP . Two novel look-ahead pipelining methods are developed to reduce the iteration bound of the original serial LFSR CRC structures; then, the unfolding algorithm is applied to obtain a parallel CRC structure with low iteration bound. The retiming algorithm is then applied to obtain the achievable lowest CP. [7]

V APPLICATIONS

The CRC can be applied to data storage devices, such as a disk drive in order to check bits in each block.
This can be applied in the both Transmitter and Receiver block in order to detect error in

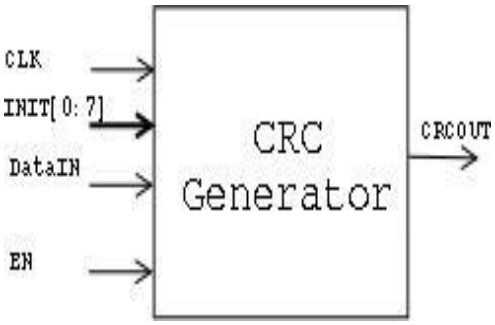
digital data. Easy to Implement in hardware. Efficient Error Detection in the digital data

VI RESULTS AND DISCUSSIONS

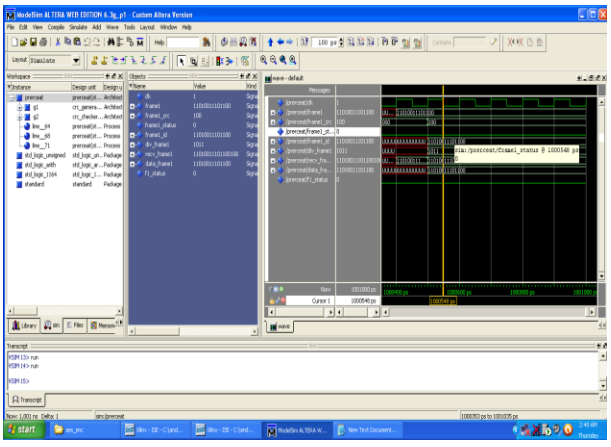
Each architecture is coded in Verilog and simulated. The simulation results and the net list simulation are verified for each architecture.

For the message bits: 11010011101100 and for the generator polynomial $1+Y+Y^2+Y^3$ i.e,1011.

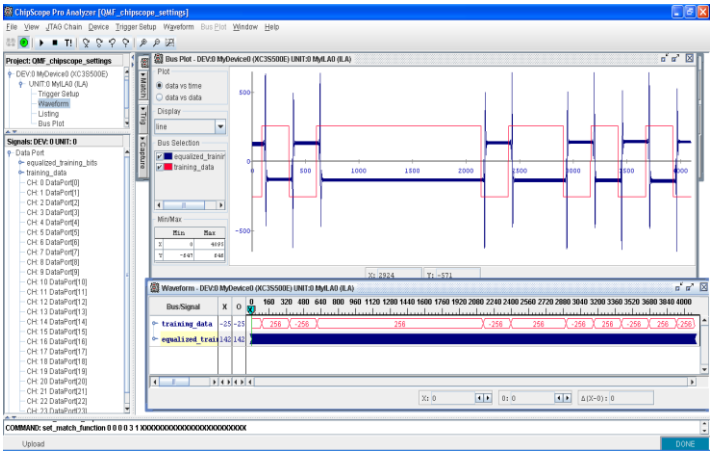
(A) Black Box View



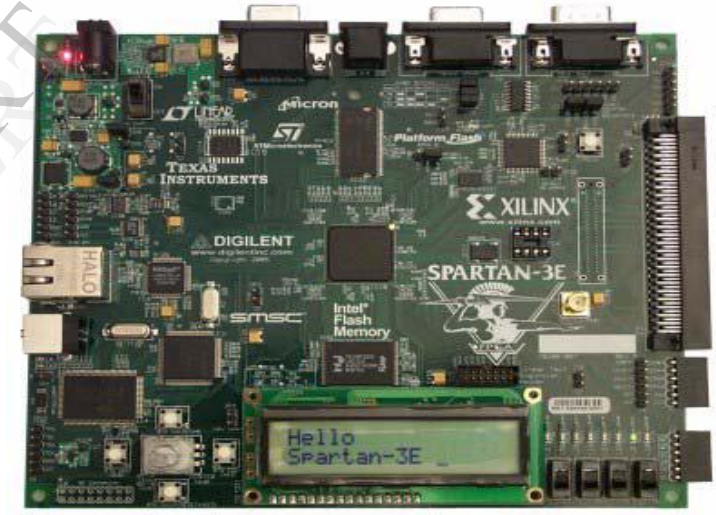
(B) Simulation Results



(C) Chip scope Result:



(d) SPARTAN_3E



VII CONCLUSION

Our parallel CRC design can efficiently reduce the CP and control or decrease the required hardware at the same time. Although the proposed design is not efficiently applicable for the LFSR architecture of any generator polynomial, it is very efficient for the generator polynomials with many zero coefficients between the second and third highest order nonzero coefficients, as shown in the

commonly used generator polynomials. A comparison on commonly used generator polynomials between the proposed design and previously proposed parallel CRC algorithms shows that the proposed design can increase the speed by up to 25% and control or even reduce hardware cost.

REFERENCES

[1] High speed parallel CRC based on unfolding, pipelining, retiming IEEE transactions on circuits and systems—ii: express briefs, vol. 53, no. 10, october 2006

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

[3] K. K. Parhi, "Eliminating the fan out bottleneck in parallel long BCH encoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 3, pp.512–516, Mar. 2004.

[4] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders," in *Proc. ACM Great Lakes Symp. VLSI*, Boston, MA,Apr. 2004, pp. 1–6.

[5] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*.
Hoboken, NJ: Wiley, 1999.

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

[7] 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.

[8] Basic VLSI design: principles and applications by Douglas A. Pucknell, Kamran Eshraghian