Comparison of Serial Data-Input CRC and Parallel Data-Input CRC Design for CRC-8 ATM HEC employing MLFSR

Avipsa S. Panda
School of Electronics, KIIT University
desiringclouds@gmail.com

G. L. Kumar
Asst. Professor(I), School of Electronics, KIIT University
ganesh.moganti@gmail.com

Abstract – **Cyclic Redundancy check (CRC) is primarily used in the physical layer of transmission protocols (viz., Ethernet and Bluetooth). CRC computation can be done in two ways; either serial computation or parallel computation. CRC computation implement linear feedback shift registers (LFSRs). The proposed paper gives an insight of serial and parallel implementation of CRC 8- CCITT widely used in Header Error Control in Asynchronous Transfer Mode (ATM HEC).**

Keywords – LFSR, CRC, MLFSR, serial architecture of CRC, parallel architecture of CRC

I. INTRODUCTION

The role of data-link layer in networking systems is to convert potentially unreliable physical link between two networks into very reliable link, by including redundant pieces of information in each transmitted frame. CRC is widely used redundancy check scheme. Usage of parity-check is not widely accepted, as parity can only detect single-bit errors and half of all multi-bit errors. Multi-bit burst errors are very common in high speed transmission channels and parity check is almost worthless [1]. CRC codes bring into play LFSRs for generation of a signature, based on the contents of data passed through it. This signature can be employed to detect the modification or corruption of bits. Common generator polynomials for CRC are:

- CRC-8 ATM-HEC : $x^8 + x^2 + x + 1$
- CRC-8 1-wire bus : $x^8 + x^7 + x^3 + x^2 + 1$
- CAN bus : $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$
- CRC-16 : $x^{16} + x^{12} + x^2 + 1$
- CRC-32 (Ethernet) : $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

CRC encoding and decoding can be done either via software or hardware and hardware implementation uses LFSRs. Traditional LFSRs generate 1-bit of output per clock cycle. Owing to high-speed application demands, the proposed design uses concept of parallel LFSR.

The paper is organized as follows. Section II contains the background of LFSRs. Section III explains serial and parallel CRCs and Section IV includes experimental results obtained. Section V brings the paper to a close.

II. BACKGROUND

A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The only linear functions of single bits are XOR and inverse 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. The LFSR has certain finite states which are eventually repeated. A linear feedback shift register can be formed by performing exclusive-OR on the outputs of two or more of the flip-flops together and feeding those outputs back into the input of one of the flip-flops.

LFSRs can be realized in either serial or parallel architectures. In a serial LFSR, per clock cycle a flip-flop is clocked and 1-bit of output is generated. In a parallel LFSR, also called *i*-output LFSR, output of *i*-successive clock cycles are generated in a single clock cycle. Parallel architecture is optimal for transmission systems, as most IEEE 802.3 transceivers communicate via a 4-bit bus [2]. Moreover, the sequence generated by serial LFSR is more or less predictable. Serial LFSRs do generate multiple-bit random numbers, but a new random number keeps most bits of the old random number, only 1-bit of information is new. An m-independent LFSR can generate m-independent random numbers.

Also, the power consumption in case of parallel LFSR is less, as switching activity gets reduced.

III. SERIAL AND PARALLEL CRC

One of the widely accepted application of LFSRs is computation of CRCs. CRC is basically a sophisticated check-sum. In general, all CRC codes are capable of detecting the following:

- All single and double-bit errors
- All burst errors of length less than the degree of the generator polynomial
- Most burst errors having length greater than the degree of the generator polynomial[3]

Mathematically, calculation of an *n*-bit CRC would be: $CRC = Rem\ [\ M(x) * (x^n/G(x))\ ]$; where M(x) is the message polynomial, G(x) is the generator polynomial and n is the degree of polynomial G(x).

Computation of CRC can be either serial computation or parallel computation. Figure 1(a) shows the serial data input hardware implementation. Din is the data message input, clk stands for clock given to the circuits. XOR gates are employed before the input of each flip-flop. The final output can be obtained from any input or output wire of any flip-flop.

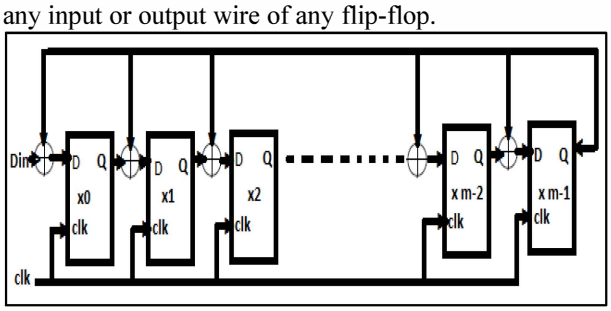


Fig. 1(a): Serial implementation of CRC

Figure 1(b) shows the parallel implementation of CRC. The data message input is to be XOR-ed with a calculated input. The calculated input can be obtained by using matrix method [4].

State equation for LFSRs can be written as: $X(i+1) = F^m.X(i) + H.D(i)$; where X(i) is the ith state of register and X(i+1) is the $(i+1)^{th}$ state of the register, D(i) is the ith serial input bit, F^m is a m x m matrix and H is a 1 x m matrix. Let us consider the generator polynomial $G = \{g_m, g_{m-1}, - - -, g_0\}$.

$F^m =$

$$\begin{bmatrix} g_{m-1} & 1 & 0 & - & - & 0 \\ g_{m-2} & 0 & 1 & - & - & 0 \\ | & - & - & - & - & - \\ | & - & - & - & - & - \\ g_1 & 0 & 0 & - & - & 1 \\ g_0 & 0 & 0 & - & - & 0 \end{bmatrix}$$

$H = [\ 0\ 0\ - - -\ 0\ 1]^T$

We obtain $x'_{m-1} = (g_{m-1} \cdot x_{m-1}) \oplus x_{m-2}$

$x'_{m-2} = (g_{m-2} \cdot x_{m-1}) \oplus x_{m-3}$

$x'_1 = (g_1 \cdot x_{m-1}) \oplus x_0$

$x'_0 = (g_0 \cdot x_{m-1}) \oplus d$

The above equations are valid for serial computation. For parallel computation, all variables used have the same meaning, only the process of calculation differs.

F^m matrix is computed, F(i)(j) indicates the value at i^{th} row and j^{th} column.

$x'_{m-1} = (F^m(m-1)(m-1).x_{m-1}) \oplus (F^m(m-1)(m-2).x_{m-2})$ - - - - - $\oplus$ $(F^m(m-1)(0).x_0) \oplus d_{m-1}$

$x'_{m-2} = (F^m(m-2)(m-1).x_{m-1}) \oplus (F^m(m-2)(m-2).x_{m-2})$ - - - - - $\oplus$ $(F^m(m-2)(0).x_0) \oplus d_{m-2}$

and so on.

$x'_0 = (F^m(0)(m-1).x_{m-1}) \oplus (F^m(0)(m-2).x_{m-2})$ - - - - - $\oplus$ $(F^m(0)(0).x_0) \oplus d_0$

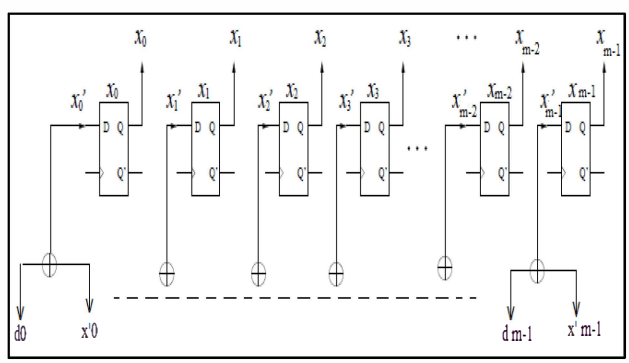


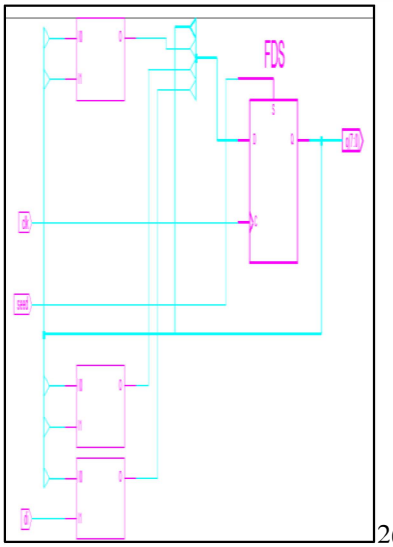
Fig. 1(b): Parallel implementation of CRC

IV. RESULTS

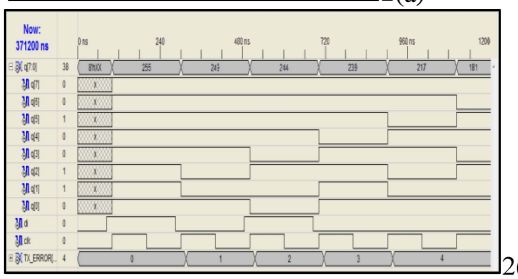
This paper shows the comparison between the serial and parallel implementation of CRC circuits. The generator polynomial used is CRC-8 ATM HEC polynomial: $x^8 + x^2 + x + 1$. The design is implemented in Verilog hardware description language and simulated using Xilinx on Spartan-3

and synthesized using Cadence Encounter tool. Both the implementations employ multi-output parallel LFSR.

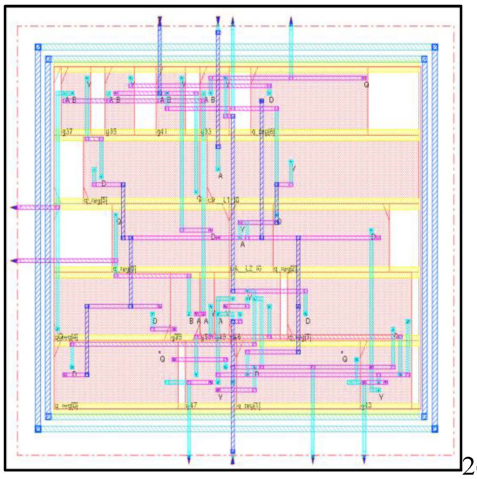
Figure 2 shows the results obtained for Serial CRC implementation and Figure 3 shows the results obtained for Parallel CRC implementation for the above mentioned polynomial. Table-1 shows the comparison of results.



2(a)

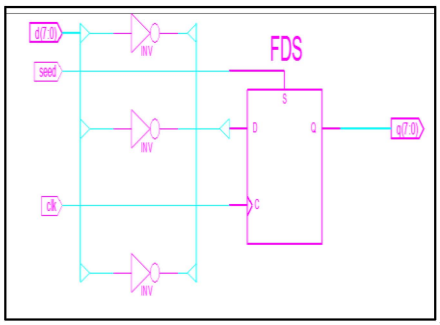


2(b)

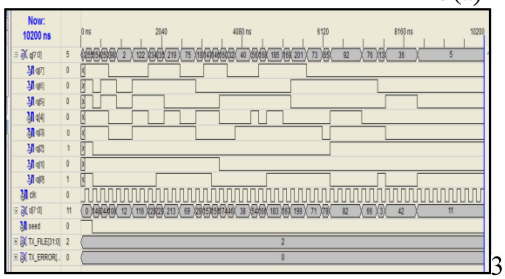


2(c)

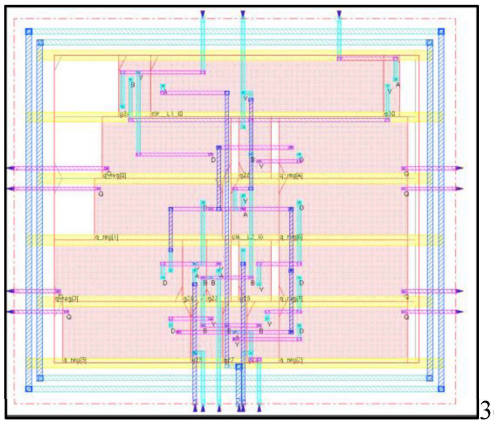
Fig. 2(a): RTL Schematic for Serial CRC
Fig. 2(b): Simulation Result
Fig. 2(c): Layout of Serial CRC



3(a)



3(b)



3(c)

Fig. 3(a) RTL Schematic of Parallel CRC
Fig. 3(b): Simulation
Fig. 3(c): Layout of Parallel CRC

Table-1: Comparison of Results obtained

	Serial CRC implementati-on	**Parallel CRC implementatio -n**
Area (No. of Instances) (μm^2)	585 (18)	529 (17)
Total Power (nW)	108005.943	57212.539
Timing slack (ps)	4809	15130

It is evident from the above table that the area, power is reduced and the timing slack is improved to a great extent. The parallel CRC implementation employing multi-output LFSR is better.

V. CONCLUSION

From the above experimental results, it is evident that the parallel data-input implementation of CRC using multi-output LFSR is better in terms of area and

power consumption. The parallel architecture is faster also. The parallelism introduction does increase the critical path, which can be improved by introducing pipelining stages in the design.

ACKNOWLEDGEMENT

The authors extend their gratitude to School of Electronics, KIIT University for providing all necessary facilities.

REFERENCES

[1] Parallel Cyclic Redundancy Check for HOTLink, www.cypress.com

[2] IEEE 802.3 Cyclic Redundancy Check, Xilinx, XAPP209 (v1.0) March 23, 2001

[3] "*Fpga Based High Speed Parallel Cyclic Redundancy Check*"; Harika, P.; Satyanarayana, B.V.V; IJERT, Vol. 2 Issue 3, March – 2013.

[4] "*A BDD-Based Approach to Constructing LFSRs for Parallel CRC Encoding*"; Dubrova, E.; Mansouri, S.S.; IEEE 42nd International Symposium on Multiple-Valued Logic, 2012.