IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 2, Issue 5 (May. – Jun. 2013), PP 66-72
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

VLSI Implementation of Parallel CRC Using Pipelining, Unfolding and Retiming

Sangeeta Singh[1], S. Sujana[2], I. Babu[3], K. Latha[4]
[1 2](Associate Professor, Department of ECE, Vardhaman College of Engineering, Hyderabad, A.P, India)
[3 4](Assistant Professor, Department of ECE, Vardhaman College of Engineering, Hyderabad, A.P, India)

Abstract : *Error detection is important whenever there is a non-zero chance of data getting corrupted. A Cyclic Redundancy Check (CRC) is the remainder, or residue, of binary division of a potentially long message, by a CRC polynomial. This technique is ubiquitously employed in communication and storage applications due to its effectiveness at detecting errors and malicious tampering. The hardware implementation of a bit-wise CRC is a simple linear feedback shift register. Such a circuit is very simple and can run at very high clock speeds, but it requires the stream to be bit-serial. This means that 'n' clock cycles will be required to calculate the CRC values for an n-bit data stream. This latency is intolerable in many high speed data networking applications where data frames need to be processed at high speed and hence implementation of CRC generation and checking on a parallel stream of data becomes desirable. This paper presents implementation of parallel Cyclic Redundancy Check (CRC) based upon DSP algorithms of pipelining, retiming and unfolding. The architectures are first pipelined to reduce the iteration bound by using novel look-ahead techniques and then unfolded and retimed to design high speed parallel circuits. This paper presents the comparison between the parallel implementation of CRC-9 and its serial implementation. It also shows that parallel implementation uses less number of clock cycles than the serial implementation of CRC-9 thereby increasing the speed of the architecture. This paper is implemented using Verilog hardware description language, simulated using Xilinx ISE tools and synthesized using Cadence tools.*

Keywords - *Cyclic Redundancy Check (CRC), Pipelining, Retiming, Unfolding*

I. Introduction

Error correction codes provides a mean to detect and correct errors introduced by the transmission channel. Two main categories of codes exist: block codes and convolution codes. They both introduce redundancy by adding parity symbols to the message data. Cyclic redundancy check (CRC) codes are the subset of the cyclic codes that are also a subset of linear block codes. Cyclic Redundancy Check (CRC) is widely used to detect errors in data communication and storage devices. CRC is a very powerful and easily implemented technique to obtain data reliability. The CRC technique is used to verify the integrity of blocks of data called Frames. In this technique, the transmitter appends an extra n bit sequence to every frame called Frame Check Sequence (FCS). FCS holds redundant information about the frame that helps the receiver detect errors in the frame. When the transmission is received or the stored data is retrieved, the CRC residue is regenerated and confirmed against the appended residue.

For high-speed data transmission, the general serial implementation cannot meet the speed requirement. Parallel processing is a very efficient way to increase the throughput rate. Although parallel processing increases the number of message bits that can be processed in one clock cycle, it can also lead to a long critical path (CP). Thus, the increase of throughput rate that is achieved by parallel processing will be reduced by the decrease of circuit speed.

Another issue is the increase of hardware cost caused by parallel processing, which needs to be controlled. The parallel CRC algorithm in [1] processes an m -bit message in (m+k)/L clock cycles, where k is the order of the generator polynomial and L is the level of parallelism. However, in [2], m message bits can be processed in m/L clock cycles. High speed architectures for parallel long encoders are based on the multiplication and division computations on generator polynomial are efficient in terms of speeding up the parallel linear feedback shift register (LFSR) structures.

The proposed design achieves shorter critical path for parallel CRC circuits leading to high processing speed than commonly used generator polynomial. The proposed design starts from LFSR, which is generally used for serial CRC. An unfolding algorithm [3] 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[1]. 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.

II. Pipelining, Unfolding And Retiming

The implementation of CRC check generation circuit can be done with the use of linear feedback circuit. The CRC architecture for generator polynomial $G(y)=1+y+y^8+y^9$ is shown in Fig.1.

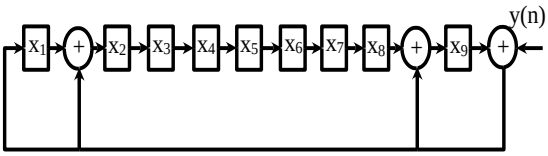

Figure 1: Serial CRC

2.1 Pipelining: It reduces the effective critical path by introducing pipelining latches along the critical data path either to increase the clock frequency or sample speed or to reduce power consumption at the same speed. It is done using a look-ahead pipelining algorithm to reduce the iteration bound. Iteration bound is defined as the maximum of all the loop bounds. Loop bound is defined as t/w, where 't' is the computation time of the loop and 'w' is the no. of delay elements in the loop. The iteration bound for the circuit shown in Fig.1 is $2T_{XOR}$. The largest iteration bound of a general serial CRC architecture is also $2T_{XOR}$.

The critical loop is described by

$$a(n+1) = a(n) + y(n) + b(n+1) \quad \text{----------} \quad (1)$$

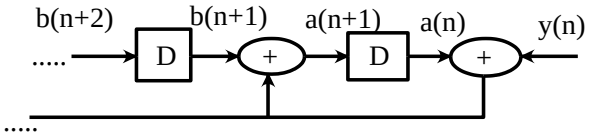

Figure 2: Pipelining to reduce iteration bound

The largest iteration bound of a general serial architecture is also $2T_{XOR}$. For example, the serial architectures of commonly used generator polynomials CRC-16 and CRC-12 have the iteration bound of $2T_{XOR}$ because they have terms $y^{15}+y^{16}$ and $y^{11}+y^{12}$ in their generator polynomials respectively.

In proposed look-ahead pipelining, 2-level pipelining is given by

$$a(n+2) = a(n+1) + y(n+1) + b(n+2)$$

$$a(n+2) = a(n) + y(n) + b(n+1) + b(n+2) + y(n+1) \quad \text{-----(2)}$$

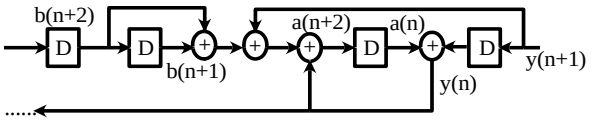

Figure 3: First level pipelining

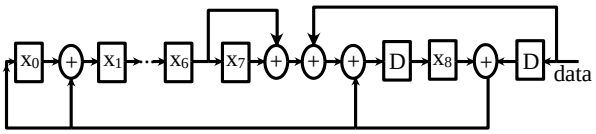

Figure 4: Two level pipelining

Fig.4 shows that the loop bound for the circuit in Fig.2 has been reduced from 2Txor to T_{XOR} at the cost of two XOR gates and two flip flops. Also the loop bounds of loop1 and loop2 are T_{XOR} and (5/8) T_{XOR} respectively. So, the iteration bound of the two level pipelined CRC architecture is T_{XOR}.

For improved look ahead pipelining consider the polynomial $G(y)= 1+y+y^7+y^9$

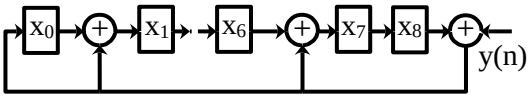

Figure 5: LFSR for G(y)

The loop equation can be given as:

$$a(n+2) = a(n) + y(n) + b(n+2) \quad \text{-------} \quad (3)$$

Four level pipelined structure can be obtained with application of improved look ahead pipelining technique.

$$a(n+4) = a(n+2) + y(n+2) + b(n+4)\ a(n+4) \text{ ------- (4)}$$

$$a(n+4) = a(n) + y(n) + b(n+2) + y(n+2) + b(n+4) \text{ ---- (5)}$$

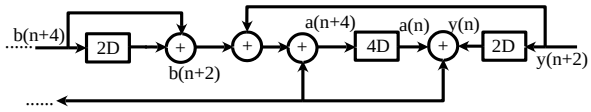

Figure 6: Loop bound of T_{XOR} /2

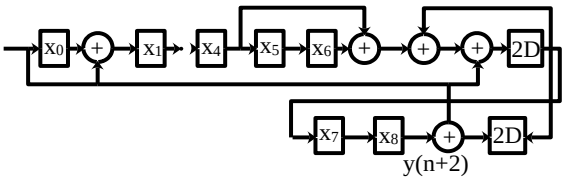

Figure 7: Four level pipelined

2.2 Retiming: Retiming is used to change the locations of delay elements in a circuit without affecting the input/output characteristics of the circuit. It reduces the critical path of the system by not altering the latency of the system. Retiming has many applications in synchronous circuit design. These applications include reducing the clock period of the circuit, reducing the number of registers in the circuit, decreasing the power consumption of the circuit and logic synthesis. It can be used to increase the clock rate of a circuit by reducing the computation time of the critical path. Critical path is the path with the longest computation time among all paths that contain zero delays, and its computation time is the lower bound on the clock period of the circuit. The two factors affecting the frequency of operation is critical path and iteration bound. Retiming is done by applying the algorithm presented in [3] and using Floyd Warshall algorithm.

2.3 Unfolding: It's a transformation technique that can be applied to DSP program to create a new program describing more than one iteration of the original program. Unfolding a DSP program by an unfolding factor J creates a new program that describes J consecutive iterations of the original program. It increases the sampling rate by replicating hardware so that several inputs can be processed in parallel and several outputs can be produced at the same time. The lower bound on the iteration period of a recursive DSP program is termed as iteration bound. An implementation of the DSP program can never achieve an iteration period less than the iteration bound, even when infinite processors are made available. In some cases, the DSP program cannot be implemented with the iteration bound equal to the iteration bound without the use of unfolding. In general, for a given DFG, when unfolding algorithm is applied with unfolding factor J, the iteration bound of the resultant DFG is J times that of the original DFG.

III. IMPLEMENTATION

CRC-9 architectures are first pipelined to reduce the iteration bound by using novel look-ahead pipelining methods and then retimed and unfolding to design high speed parallel circuits. The proposed design starts from LFSR, which is generally used for serial CRC-9.

$G(y)=1+y+y^8+y^9$

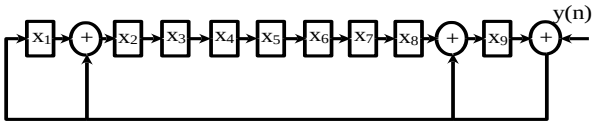

Figure 8: Serial CRC-9

The serial CRC-9 is input is having the 1 bit data and the correct output is obtained after 9 clock cycles. CRC-9 architectures are first pipelined to reduce the iteration bound.

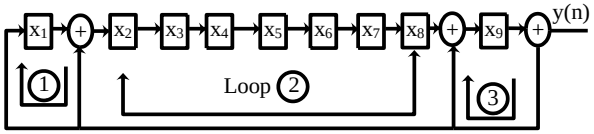

Figure 9: LFSR of $G(y)=1+y+y^8+y^9$

Loop bound for this 1st = t/w = T_{XOR} (t=1T_{XOR}; w=1)

2nd =(2/7)T_{XOR}.

3rd =2 T_{XOR}.

Retiming algorithm is then applied to the pipelined architecture and based on the algorithm data flow graph for the pipelined circuit of Fig.10 is drawn. Here the critical path obtained is of magnitude 2.

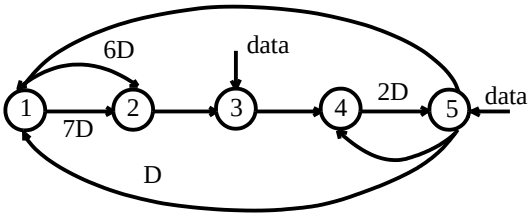

Figure 10: DFG for pipelined block

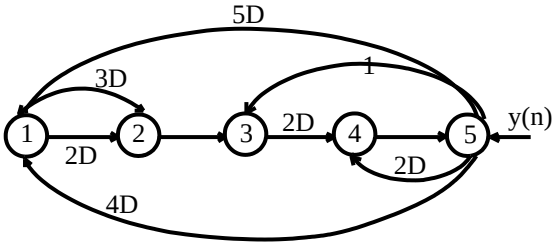

Figure 11: Retimed Graph

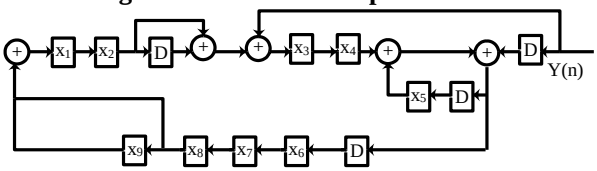

Figure 12: LFSR for retimed CRC

The following are the steps to be applied for unfolding after application of retiming algorithm. For each node U in the original DFG, draw J node $U_0,U_1,\ldots U_{J-1}$. For each edge U→V with W delays in the original DFG, draw the J edges $U_i \rightarrow V_{(i+w)\%J}$ with (floor(i+w)/J) delays for i=0,1…J-1. Where 'i' is index of the node, 'w' is weight of the edges, 'J' is unfolding factor. Fig.13 and Fig.14 show the DFG and LFSR representation for 2-level unfolding.

The correct output is obtained after 5 clock cycles. Application of the unfolding algorithm increases the speed of the architecture.

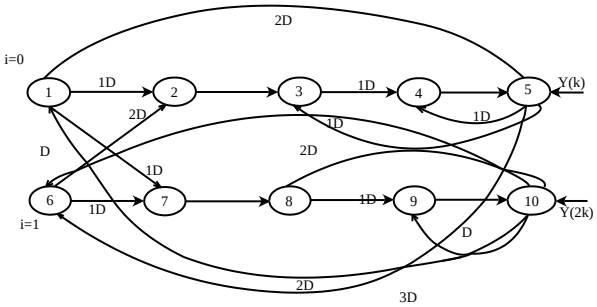

Figure 13: Data flow graph for 2-level unfolding

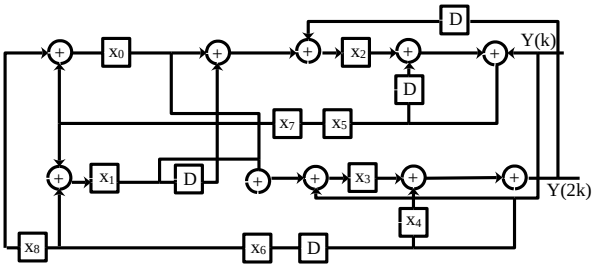

Figure 14: LFSR representation for 2-level unfolding

IV. RESULTS

This paper shows the comparison between the serial and parallel implementation of CRC circuits using pipelining, retiming and unfolding techniques. The generator polynomial used is given by $G(Y)=1+y+y^8+y^9$. The input message sequence polynomial is: 1100000011. The design is implemented in verilog hardware description language and simulated using Xilinx on Spartan-3 and synthesized using cadence tools. Fig.15 shows the results obtained for the serial implementation of CRC. The correct output is obtained after 9 clock cycles. Table 1 shows the timing report for serial CRC. After application of the pipelining, retiming and then unfolding algorithm to the same design, the output is generated after 5 clock cycles which is shown in figure 19. Hence it reduces the clock cycles and thus increases the speed of the circuit. Fig.16 to Fig.18 depict the respective RTL schematics.

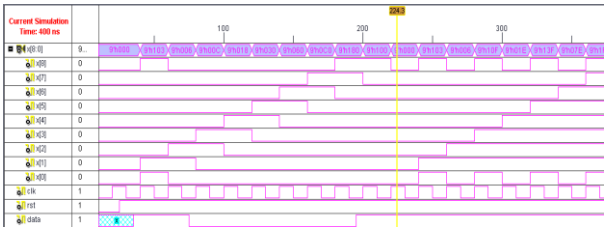

Figure 15: Serial CRC

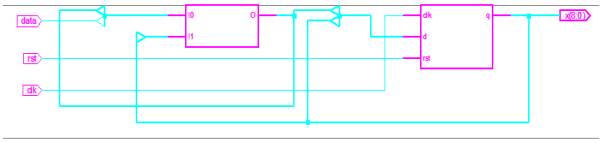

Figure 16: Serial CRC- RTL schematic

Table 1: Timing report for serial CRC

Timing Summary Speed Grade: -5	
Minimum period:	1.962ns (Maximum Frequency: 509.697MHz)
Minimum input arrival time before clock:	3.567ns
Maximum output required time after clock:	4.182ns

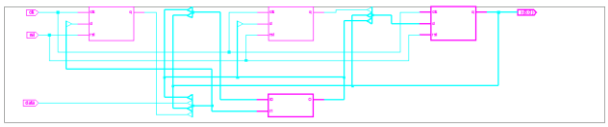

Figure 17: RTL schematic of pipelined CRC

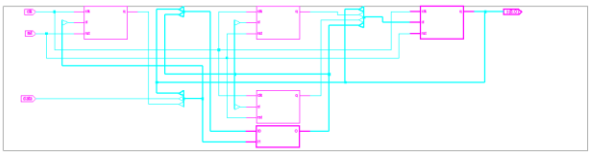

Figure 18: RTL schematic of retimed CRC

Figure 19: CRC output based on pipelining, retiming and unfolding

The following are the results obtained for CRC implementation using generator polynomial $G(Y)=1+y+y^8+y^9$ and input message sequence 1100000011. The required number of clock cycles, iteration bound and critical path is noted down for different architectures. These results are depicted in Table 2 to 4.

Table 2: No. of clock cycles

Architecture	No. of clock cycles
Original Architecture	9
2-level pipelined	10
4-level pipelined	12
Retiming after 2level pipelining	10
Unfolding the 2-level pipelined	5
Retiming the unfolded architecture	6

Table 3: Iteration Bound

Architecture	Iteration Bound
Original architecture	$2\ T_{XOR}$
2-level pipelined	T_{XOR}
4-level pipelined & retiming	$(7/8)T_{XOR}$

Table 4: Critical Path

Architecture	Critical Path
2-level pipelined	$2\ T_{XOR}$
Retiming the 2-level pipelined	T_{XOR}

V. CONCLUSION

This paper implements pipelining method for high-speed parallel CRC circuits. Pipelining has decreased the iteration bound of the architecture effectively. Applying unfolding technique to pipelined architecture increased the throughput of the circuit and thereby applying retiming to the architecture reduced the critical path delay. So applying pipelining, unfolding and retiming to the CRC has increased the throughput to achieve high speed design. This brief has proposed two pipelining methods for high speed parallel CRC hardware implementation. Proposed look ahead pipelining has a simpler structure, parallel CRC design can efficiently reduce the critical path. 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. The serial implementation of CRC-9 uses 9 clock cycles, iteration bound is $2T_{XOR}$ and critical path is $2T_{XOR}$. Whereas the parallel implementation of CRC-9 using pipelining, retiming and unfolding uses only 5 clock cycles, its iteration bound is T_{XOR} and critical path is T_{XOR}. Thus increasing the speed of the architecture. This can be extended to achieve the high speed CRC circuit without increasing the hardware resources such that area occupied by the design is minimized. Parallel CRC architecture implemented here is having high speed compared to previous algorithms and also hardware cost is controlled.

REFERENCES

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

[2] 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.

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

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

[5] 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.

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

[7] W. Peterson and D. Brown, "Cyclic codes for error detection," Proc. IRE, vol. 49, no. 1, pp. 228–235, 1961.
[8] G. Albertengo and R. Sisto, "Parallel CRC generation," IEEE Micro, vol. 10, no. 5, pp. 63–71, Oct. 1990.
[9] M. D. Shieh, M.-H. Sheu, C.-H. Chen, and H.- F. Lo, "A systematic approach for parallel CRC computations," J. Inf. Sci. Eng., vol. 17, no. 3, pp. 445–461, May 2001.
[10] A. Tanenbaum, Computer Networks, 4th ed. Englewood Cliffs, NJ-Prentice Hall, 2003.
[11] 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.
[12] M. Sprachmann, "Automatic generation of parallel CRC circuits," IEEE Des. Test Comput., vol. 18, no. 3, pp. 108–114, May 2001.
[13] Martin Grymel and Steve B. Furber, " A novel programmable parallel CRC circuit," IEEE Trans. VLSI Systems,. 19, vol. no. 10, October 2011.
[14] S. Lin and D. J. Costello, Error Control Coding. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[15] R.Lee, "Cyclic Codes Redundancy," Digital Design, July 1977.
[16] D. Feldmeier, "Fast Software Implementation of Error Detection Codes," IEEE Trans. Networking, Dec. 1995.
[17] M.C. Nielson, "Method for High Speed CRC Computation," IBM Technical Disclosure Bull., vol. 27,no. 6, pp. 3572-3576, Nov. 1984.
[18] T. C. Denk & K. K. Parhi, "Exhaustive Scheduling and Retiming of Digital Signal Processing Systems", IEEE Trans. On Circuits and Systems-II: Analog and DSP, vol. 45, no.7, pp. 821-838, July 1998.