CRC Generation for Protocol Processing

Ulf Nordqvist, Tomas Henrikson and Dake Liu
Department of Physics and Measurement Technology
Linköpings University, SE 58183 Linköping, Sweden
Phone: +46-1328-{8916, 1256 and 8956}
Email: {ulfnor, tomhe and dake}@ifm.liu.se
Fax: +46-13132285

***Abstract**:*

In order to provide error detection in communication networks a method called Cyclic Redundancy Check has been used for almost 40 years. This algorithm is widely used in computer networks of today and will continue to be so in the future. The implementation methods has on the other hand been constantly changing.

A comparative study of different implementation strategies for computation of Cyclic Redundancy Checks has been done in this paper. 10 different implementation strategies was examined. A novel architecture suitable for use as an IP in an protocol processor is presented. As conclusion, different implementation techniques have been divided into application areas according to their speed, flexibility and power-consumption.

1.0 INTRODUCTION

Both computer and human communication networks, uses protocols with ever increasing demands on speed, cost and flexibility. In the market segment of hardware for network nodes such as routers, switches and bridges, the performance needs can be fulfilled by using Application Specific Integrated Circuits (ASIC) or Application Specific Standard Products (ASSP). This will probably be the case also in the future due to there relatively cost-insensitive costumers. In order to let the end-user take advantage of the bandwidth enhancement in today networks, tomorrows Network Terminal (NT) hardware must support transmission speeds of Gbit/s [10]. Hardware for such NT components is on the other hand sold on a cost-sensitive market share with high demands on flexibility and usability.

Traditionally NT has been implemented as ASIC:s for the lower layers in the OSI-Reference Model [17] with an CPU-RISC based SW implementation of the upper layers [8], or completely implemented in software [1], [3], [17]. In [6], [7] we presented a new architecture for configurable protocol processing that supports programmability on the upper layers and gives both configurability and high performance on the lower layers. This kind of solution is also supported by [18], [19] and [14]. This architecture specifies that, the without any doubt most computational extensive task, Cyclic Redundancy Check (CRC) [3], [20], should be implemented as configurable hardware, supporting buffering free processing.

The speed requirement is very important since a protocol processor must buffer incoming data if jobs are not completed at wire-speed. This leads to high costs in terms of power consumption, area and manufacturing costs due to the usage of buffers.

The aim of this paper is to compare different implementations of CRC computational units in order to specify a suitable one for protocol processors.

1.1 The CRC algorithm

Cyclic Redundancy Check is a way of providing error control coding in order to protect data by introducing some redundancy in the data in an controlled fashion. It is a commonly used and very effective way of detecting transmission errors during transmissions in various networks. Common CRC polynomials can detect following types of errors:

- All single bit error
- All double bit errors
- All odd number of errors
- Any burst error for which the burst length is less than the polynomial length
- Most large burst errors

The CRC encoding procedure can be described by equation 1.

$$V(x) = S(x) + x^{n-k}U(x) \qquad \text{(EQ 1)}$$

V(x) is the n bit long data word transmitted and it consists of the original data and U(x) followed by a codeword S(x) called the CRC-sum. S(x) is computed according to equation 2.

$$X^{n-k}U(x) = a(x)g(x) + S(x) \qquad \text{(EQ 2)}$$

S(x) is by other words the reminder resulting from a division of the data stream and a generator polynomial g(x).

The actual coding-procedure is the same on both the receiving and transmitting end of the line. The CRC encoding/decoding principle is illustrated by figure 1.

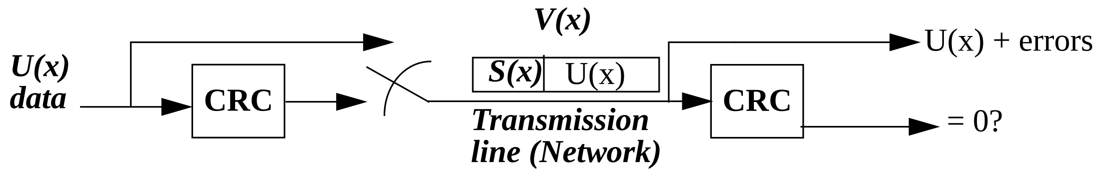

FIGURE 1. Principle of error detection using the CRC algorithm.

As can be seen in figure 1 the receiving NT perform a CRC-check on the incoming message and if the result is zero, the transmission was error free. One more practical way of solving this is to compute the CRC only for the first part of the message U(x), and then do a bitwise 2-complements addition with the computed checksum S(x) on the transmission side. If the result is non-zero the receiver will order a retransmission from the sender.

2.0 IMPLEMENTATION THEORY

This section introduces the commonly used and presents one new architecture for implementation of the CRC algorithm.

- **Software(SW) Solution** [3]**,** [1]: The CRC algorithm can always be implemented as an software algorithm on a standard CPU, with all the flexibility reprogramming then offers. Since there in most communication network terminals exists a CPU, the SW-solution will be cheap or free in terms of hardware cost. The drawback is obviously the computational speed since no general purpose CPU can achieve the same troughput as dedicated hardware.

- **Traditional Hardware Solution**: Linear Shift Register (LSR) with serial data feed [20] has been used since the sixties to implement the CRC algorithm, see figure 2. As all hardware implementations, this method simply perform a division and then the reminder which is the resulting CRC checksum, is stored in the registers (delay-elements) after each clock cycle. The registers can then be read by use of enabling signals. Simplicity and low power dissipation are the main advantages. This method gives much higher throughput than the SW solution but still this implementation can not fulfill all the speed requirements of today network nodes. Since fixed logic is used there is no possibility of reconfigure the architecture and change the generator polynomial using this implementation.

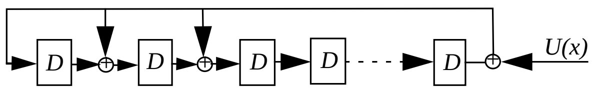

FIGURE 2. Linear Shift Serial Data Feed

- **Parallel Solution**: In order to improve the computational speed in CRC generating hardware, parallelism has been introduced [2], [4], [5], [9], [11], [12]. The speed-up factor is between 4 and 6 when using a parallelism of 8. By using fixed logic, implemented as parallelised hardware, this method can supply for CRC generation at wire speed and therefore it is the pre-dominant method used in computer networks. The parallel hardware implementation is illustrated by figure 3. If the CRC polynomial is changed or a new protocol is added, new changed hardware must be installed in the network terminal. The lack of flexibility makes this architecture non suitable for use in a protocol processor.

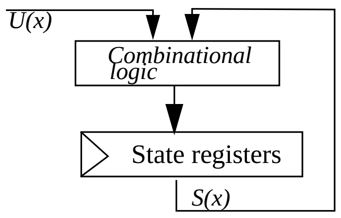

FIGURE 3. Parallel Fixed Logic Implementation

Configurable Hardware: One way of implementing configurable hardware is by using Look-Up-Tables (LUT) as proposed by [3], [12] and [2]. The architecture is illustrated by figure 4.

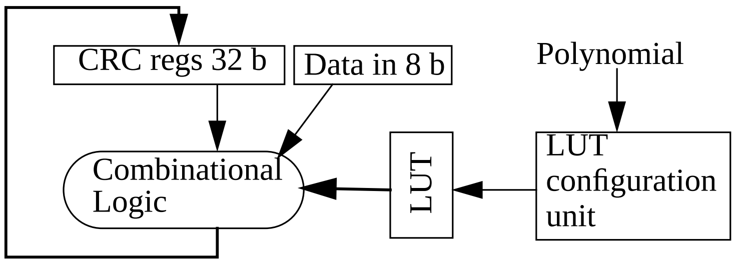

FIGURE 4. Look Up Table based configurable hardware.

This implementation can be modified by using a larger or smaller LUT. If the size of the LUT is reduced the hardware-cost in terms of power consumption and area will be reduced but in the same time the Combinational Network will be increased so the effect will be cancelled. The optimal solution has not been derived.

Another, novel implementation method is the ***Radix-16 Configurable CRC Unit***, which is presented for the first time in this paper. By noticing that any polynomial of a fixed length can be represented by implementing the CRC using a LSR with switches on the reconnecting wires as illustrated by figure 5, a configurable hardware can be implemented using NAND-gates to represent the switches.

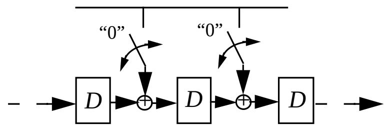

FIGURE 5. Configuration by use of switches in the circuit reconnecting wire.

In order to improve the speed of the Radix-16 Configurable CRC, a 4 - bit wide input data stream is used as can be seen in figure 6. The resulting bit in each position *k* in the CRC register then depends on the value of the *k-4* CRC bit, the last four CRC bits, the polynomial bit description and the input bits. The logic, which consists mainly of XOR and NAND-functions provides the necessarily configurability.

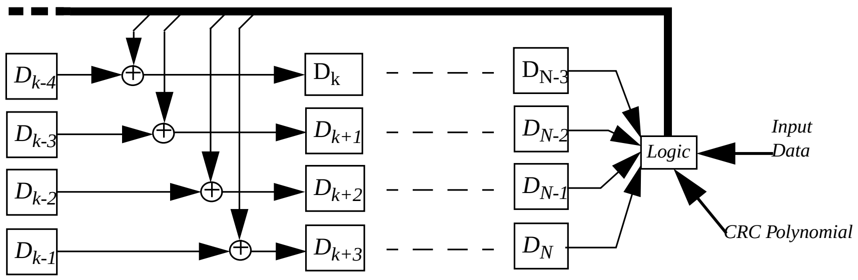

FIGURE 6. Radix-16 Configurable CRC engine

The polynomial input makes it possible to implement any given CRC algorithm of a given size. Using shut-down logic on parts of the circuit enables N to be configured for 16, 24 or 32 bit polynomials. This means that for example CRC polynomials for protocols such as HIPERLAN and Ethernet is manageable.

3.0 EXPRIMENTAL RESULTS

10 different implementations of the CRC algorithm, including one CPU RISC based SW-implementation, have been examined. They have been described using Mentor Graphics Renoir and VHDL, synthesized and optimized using Build Gates from Cadence and the Place & Route was done using Ensemble P&R from Cadence. The technology used is AMS 0.35 µm.

Since most network protocols are bytebased, there is no meaning in investigating a parallelism of more than 8 even if the other parts of a protocol processor might run on other clock frequencies using for example a 32 bit wide input stream.

As seen in table 1 the fixed logic and parallel input implementation is the fastest. That is in the order of what have been reported in earlier work. We can also see that the LUT based method gives about twice the speed as the Configurable Radix 16 implementation at the cost of a 4.5 times higher area. A big part of the area in the LUT based architecture is the LUT registers, but the power consumption will anyway be considerably higher than the power consumption in the Radix-16 implementation. In many upcoming applications such as HIPERLAN [15], [16], the power consumption will be crucial. The speed supported by the Radix-16 implementation exceeds 0.6 Gbit/s, which is sufficient since today NT applications do not demand higher troughput. Since the logic in that specific implementation dominates and the connection delay is quite small, there will be a considerable increase of the speed powered by down-scaling in future technologies. The speed-up factor due to scaling s will be up to s^2 which means that even protocols as 10-GEthernet which will come in the future can be supported by the Radix-16 implementation [13] thanks to scaling.

TABLE 1. Comparison between different CRC implementations. The Pads are not included in the area computation.

CRC implementation	**Polyn. Length**	**Area** [mm^2]	**Max Clk freq. [MHz]**	**Max Speed [Mbit/s]**
Serial Input - fixed Ethernet Polynomial	32	0.014	413	413
Serial Input - any polynomial	32	0.017	369	369
Serial Input - any polynomial	16	0.011	355	355
Parallel(8) Input - any polynomial	32	0.061	109	875
Parallel(8) Input - any polynomial	16	0.038	130	1039
Parallel(8) Input - fixed Ethernet Polynomial	32	0.035	208	1663
Parallel(8) Input LUT Based	32	0.225	169	1358
Configurable Radix-16 CRC - any polynomial	**32**	**0.050**	**166**	**663**
Configurable Radix-16 CRC - any polynomial	**16,24,32**	**0.052**	**153**	**612**
SW Pure RISC (43893 clk cycles / 1500 Bytes)	any		600	164

Conflict with other processes makes interlayer processing difficult, not to say impossible when using the SW algorithm run on a CPU. This means that even if the SW-algorithm alternative can be implemented on a high-performance CPU that provides the speed that is needed, it is not suitable for protocol processors such as those described by [6] and [7].

4.0 CONCLUSIONS

Because of the superior performance of a parallel ASIC implementation, it will be used for implementation of network-node-components. The concept of using several ASIC implementation as Functional Units in a protocol processor and just letting the processor turn on the CRC that is currently used, as in VLIW architechtures, might also be of interest although you then have no configurability for supporting new protocols.

Software solutions for low speed protocols will also be used for low-speed applications, but a increasing area of applications demands high-speed configurable protocol processing, including CRC generation. An hardware architecture that can fulfill this specifications is the Look-Up table -based structure proposed in [1] and implemented in this paper.

A novel architecture for this application area has also been presented, which has a superior power-delay product. The architecture implemented can be configured for CRC encoding/decoding using any 16, 24 or 32 bit polynomial. Power consumption will be kept low using shut-down logic. The architecture support the speed requirements of today protocol processing in NT:s. For upcoming protocols used in NT network processing, scaling will provide necessarily speed-enhancements.

5.0 REFERENCES

[1] A. Perez, “Byte-wise CRC Calculations”, *IEEE Micro*, Vol. 3, No. 3, June 1983, pp. 40-50
[2] G. Albertango and R. Sisto, “Parallel CRC Generation”, *IEEE Micro*, Vol. 10, No. 5, October 1990, pp. 63-71
[3] T. V. Ramabadran and S. S. Gaitonde, “A Tutorial on CRC Computations”, *IEEE Micro*, Vol.8, No. 4, August 1988, pp. 62-75
[4] T. B. Pei and C. Zukowski, “High-speed parallel CRC cicuits in VLSI”, *IEEE Transaction Communication*, Vol. 40, No. 4, pp. 653-657, 1992.
[5] R. F. Hobson and K. L, Cheung, “A High-Performance CMOS 32-Bit Parallel CRC Engine”, *IEEE Journal Solid State Circuits*, Vol. 34, No. 2, Feb 1999
[6] D. Liu, U. Nordqvist, and C. Svensson, “Configuration-Based Architecture for High-Speed and General-Purpose Protocol-Processing”, *Proceedings of IEEE Signal Processing Systems 1999*, pp. 540-547, Taipei
[7] T. Henrikson, U. Nordqvist, and D. Liu, “Specification of a Configurable General-Purpose Protocol-Processor”, *Proceedings of CSNDSP 2000*, Bournemouth
[8] C. J. Georgiou and C.-S. Li, “Scalable Protocol Engine for High-Bandwidth Communications”, *IEEE International Conference on Communications. ICC’97 Montreal,* Volume: 2, 1997, Page(s): 1121 -1126 vol.2
[9] R. Nair, G. Ryan, and F. Farzaneh, “A Symbol Based Algorithm for Implementation of Cyclic Redundancy Check (CRC)”, Proceedings *VHDL International Users’ Forum*, 1997, Page(s): 82 -87
[10] J. Kadambi et al, “Gigabit Ethernet”, *Prentice Hall PRT, ISBN 0-13-913286-4*, 1998
[11] G. Castagnoli, S. Brauer, and M. Herrmann, “Optimization of cyclic redundancy-check codes with 24 and 32 parity bits”, *IEEE Transactions on Communications,* Volume: 41 6, June 1993, Page(s): 883 -892
[12] R. J. Glaise, X. Jacquart, “Fast CRC calculation”, *1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1993, Page(s): 602-605
[13] A. P. Chandrakasan, R. W. Brodersen, “Minimizing power consumption in digital CMOS circuits”, *Proceedings of the IEEE*, Vol, 83 No. 4, pp. 498 -523, April 1995
[14] M. Yang, A. Tantawy, “A design methodology for protocol processors”, *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 1995, pp. 376 -381
[15] “Technical Specification of BRAN and Hiperlan-2. Common part.”, *ETSI TS 101 493 - 1, V1.1.1*, 2000
[16] “Technical Specification of BRAN and Hiperlan-2. Ethernet Service Specific Convergence Sublayer.”, *ETSI TS 101 493 - 2, V1.1.1*, 2000
[17] A. S. Tannenbaum, “Computer Networks”, *3rd Edition*, *Prentice Hall PRT,* ISBN 0-13-349945-6, 1996
[18] “Building Next Generation Network Processors”, *White paper, Sep 1999, Agere Inc*., http://www.agere.com/support/non-nda/docs/Building.pdf
[19] D. Husak, “Network Processors: A Definition and Comparison”, *White paper, C-PORT,* http://www.cportcorp.com/solutions/docs/netprocessor_wp5-00.pdf
[20] W. W. Peterson and D. T. Brown “Cyclic Codes for Error Detection”, *Proc. IRE,* Jan 1961, pp. 228-235