CONFIGURABLE CRC GENERATOR

Ulf Nordqvist, Tomas Henrikson and Dake Liu
Department of Electrical Engineering
Linköpings University, SE 58183 Linköping, Sweden
Phone: +46-1328-{2903, 8956 and 1256}
Email: {ulfnor, tomhe and dake}@isy.liu.se
Fax: +46(0)13 132599

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 novel architecture suitable for use as accelerating hardware in an protocol processor has been implemented and manufactured. It provides both high throughput and configurability for different protocols when used in a network terminal. Since the computation of the CRC is one of the most computational extensive operations performed by a network terminal the use of the accelerator presented here would reduce the workload of a host processor significantly.*

1 Introduction

Both computer and human communication networks, uses protocols with ever increasing demands on speed, cost and flexibility. There is also a strong development towards an increased use of network protocols for applications where other techniques where common earlier, e.g. voice and video. One reason is that packet based network protocols can normally handle a mixture of any kind of traffic. For network node components 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 users. 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 using 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]. Usage of standard, general purpose CPU:s, is expensive in terms of cost, space and power due to their lack of dedicated hardware. There is also an upper capacity limit, set by the I/O capacity and the instruction rate of the CPU.

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 in order to solve these problems. This kind of solution is also supported by [18], [19] and [14]. This architecture specifies that the data-path of a protocol processor should be implemented as configurable functional units. In paper [21] the authors of this paper compared different implementation methods based on simulations. In that

paper we stated that the without any doubt most computational extensive task, Cyclic Redundancy Check (CRC) [3], [20], should be implemented as configurable hardware, supporting buffering free in-line 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 configurability is also very important in order to supply for adaptability to different protocol standards. Instead of using multiple application specific CRC generators dedicated for one (or several) protocols that are switched between depending on which protocol that is currently in use, we would like to use one accelerator configurable for a wide area of protocols.

Different techniques for implementation of the CRC algorithm are presented in chapter 1.1. In chapter 2 a novel implementation is presented and in chapter 3 some results and test environment are discussed.

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, provided the constraint length is sufficient
- 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.

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

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) are the extra bits added to a message in order to provide redundancy so that errors during transmission can be detected. The length of the S(x) is denoted the constraint length. The constraint length of the most commonly used CRC polynomials are between 8 and 32 bits. S(x) is computed according to equation 3.

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

$$\frac{X^{n-k}U(x)}{g(x)} = a(x) + \frac{S(x)}{g(x)} \tag{3}$$

S(x) is by other words the reminder resulting from a division of the data stream and a generator polynomial g(x). Since all codewords are divisible with g(x), the remainder S(x) of the left hand side of (3) has to be zero for a real codeword.

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.

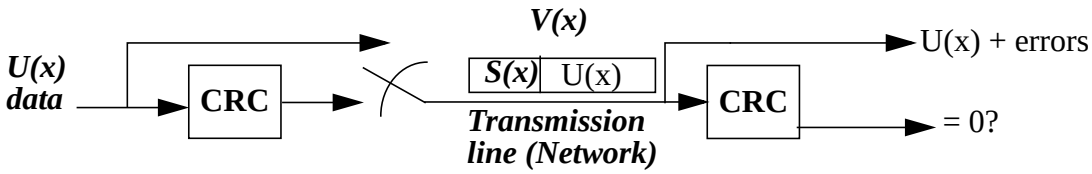

FIGURE 1. Principle of error detection using the CRC algorithm.

As can be seen in figure 1 the receiving NT perform a CRC-check on the incoming message and if the result (S(x)) 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.

1.2 Traditional implementations

This section introduces the commonly used and presents one new architecture for implementation of the CRC algorithm. It should be noted that pipelining which is a common technique for increasing of the throughput, is impossible to use for CRC because of the loop in all generators.

Software (SW) Solution [3], [1]: The CRC algorithm can always be implemented as an software algorithm on a standard CPU or general purpose DSP, with all the flexibility reprogramming then offers. Since there in most communication network terminals exists a CPU or DSP, 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. One general purpose RISC CPU need roughly 30 instructions / byte to perform CRC generation. The processing load might also be a problem for the host processor in many applications.

Serial ASIC 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. Several loop-connections schemes and reset alternatives exist.

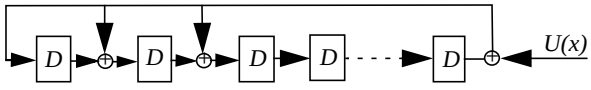

FIGURE 2. Linear Shift Serial Data Feed

Parallel ASIC 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 there-

fore 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. That would be very expensive. The lack of flexibility makes this architecture non suitable for use in a protocol processor.

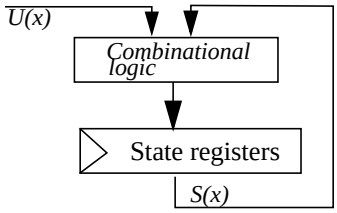

FIGURE 3. Parallel Fixed Logic Implementation

LUT based solution

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.

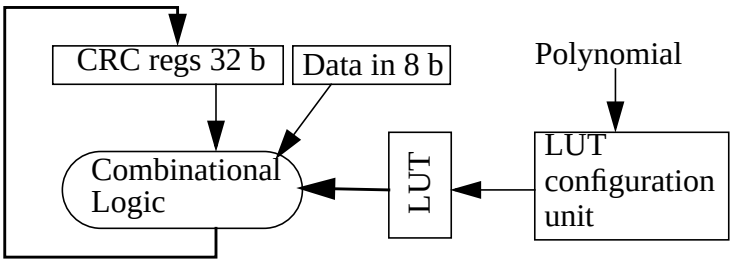

FIGURE 4. Look Up Table based configurable hardware.

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. In [21] we stated that the LUT solution gives a rather high throughput have limitations in terms of configurability. The LUT implementation enables some configurability since it is possible to change the polynomial by changing the content of the LUT memory. However there is no possibility to adjust for different constraint lengths if the combinational logic not is reconfigurable. Further any change of polynomial and/or constraint length would take a significantly number of clockcycles to perform since all the contents of the LUT have to be recalculated and replaced. This problem may force the terminal to buffer the incoming data and therefore making in-line processing impossible.

2 Implementation

Another, novel implementation method is the **Radix-32 Configurable CRC Unit**, has been implemented and manufactured. The architecture combines configurable and parallel hardware which makes it suitable for use in a protocol processor. The architecture will not compete with the traditional methods presented in chapter 1 but rather replace them in an novel application area. The throughput will be in the order of 30 to 50% lower than if a LUT implementation would have been used. This is ilustrated by figure 5.

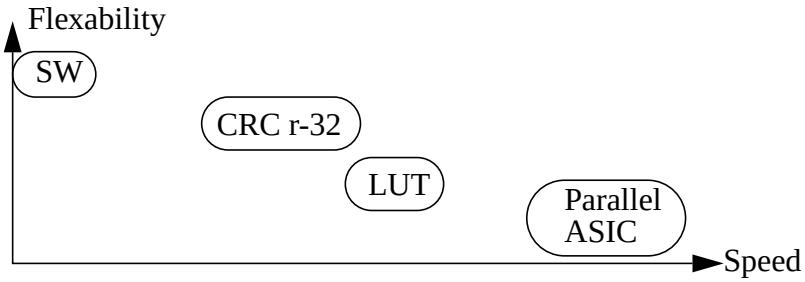

FIGURE 5. Speed and flexability differenses between different solutions.

2.1 Configuration

By noticing that any polynomial of a fixed length can be represented by implementing the CRC using a LSR with switches on the loop back as illustrated by figure 6, a configurable hardware can be implemented using NAND-gates to implement the switches.

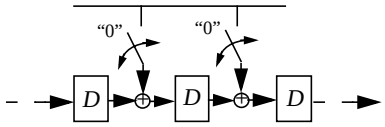

FIGURE 6. Configuration by use of switches in the circuit reconnecting wire.

This allows us to change the polynomial g(x) of a given length L by storing a bit description of the polynomial in a register. However some protocols uses CRC with different constraint length L. This can be solved by using reset-signals for the flip-flops not used as illustrated in figure 7. This also enables a programable shut-down feature that saves power when the CRC is in-active. All flip-flops also needs a preset input since before the start of the computation all flip-flops must be set to "one".

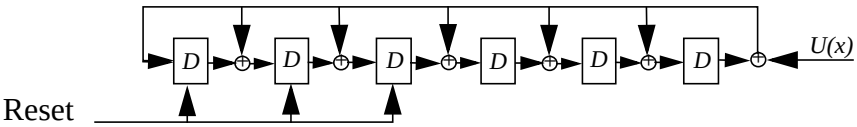

FIGURE 7. Changing constraint length using reset-signals. In this particular example the constraint length can be changed between 3 and 6 bits.

2.2 Parallelism

In order to improve the speed of the Radix-32 Configurable CRC, a 8 - bit wide input data stream is used as can be seen in figure 8.

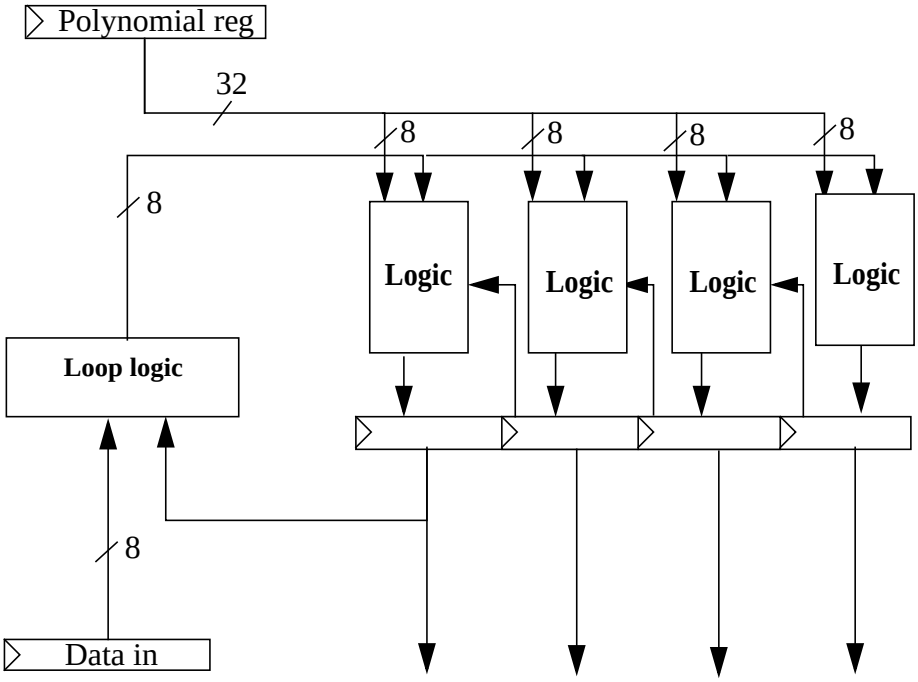

FIGURE 8. Bytewise calculations enhances the throughput of the CRC.

The resulting bit in each position k in the CRC register then depends on the value of the k-8 CRC bit, the last eight CRC bits, the polynomial bit description and the input bits. The logic, which consists mainly of XOR and NAND-functions provides the necessarily configurability. The logic is further explained by (4), note that the sum is modulo-2 (implemented as XOR).

$$NEXTCRC(k) = CRC(k-8) \oplus \sum_{i=7}^{0} Polynomial(k-i) \cdot LoopData(i) \tag{4}$$

For a fixed polynomial CRC, the throughput gets higher the wider inputs are used. The speedup in a configurable CRC is however not that significant. In simulations we have found it to be less than 20% when doubling the input width from 8 to 16 bits. Further it is clear that every network seen today are byte-based. This means that a multi-byte calculation of the CRC must handle the last bytes in a transmission using some special algorithm which will increase the overall complexity and thereby reduce the gain. That is why a 8 bit-wide input has been chosen.

The polynomial registers makes it possible to implement any given CRC algorithm of a given size. Using shut-down logic on parts of the circuit enables the CRC to be configured for any polynomial with a constraint length up to 32 bits. This means that for example CRC polynomials for protocols such as HIPERLAN, UMTS, ATM and Ethernet is manageable. The CRC unit can be reconfigured for a new protocol within on clock cycle and then process an incoming packet without using buffers or memory.

3 Measured results

The **Radix-32 Configurable CRC Unit Radix-32 Unit** was implemented using a synthesis tool called Build Gates from Cadence. Place and Route tool was Silicon Ensemble, also from Cadence. The standard cells used comes from AMS 0.35 m 3.3 V process.

The chip manufactured contained 2 additional designs and one Parallel/Serial-converter for speedup of the input data feeding. In order to use a minimum number of pads on the chip the polynomial register implemented using a shift register. Also the output is bit-serial out shifted from the chip.

Due to the limiting pad numbers the controllability of the chip is limited. Therefor testing on a limited set of random data has been used in order to verify the functionality. Another test limitation is the fact that the measurement equipment (pattern generator) available only allows stepwise increment of the frequency. However one can by reducing the supply voltage theoretically calculate the maximum throughput. Using this method the measured maximum frequency for the design is 189 MHz which gives a throughput of 1512 MB/s. The simulated result from static timing analysis suggested 186 Mhz which is very close.

4 Conclusions and further work

This paper incudes an overview of different implementations of the CRC algorithm from a throughput and flexibility point of view. Further a solution suitable for use in an protocol processor has been presented.

With a 8 bit wide input Configurable Radix -32 CRC running on 189 MHz a throughput of about 1.5 GB/s which is sufficient today. In a few years the dominating terminal protocol will probably be 10 GEthernet which means that the speedup required is under 7 times. That is obviously within reach without changing the architecture used by just taking advantage of scaling. If the CRC is used as a accelerating functional unit in a protocol processor such as the one presented in [6] it would significantly reduce the workload of the host processor. Since it operates on wire-speed the need for buffering of incoming data is removed and power consumption will be reduced significantly. Multiple protocol processors can then be used as port processors on a system-on-chip (SoC) in order to provide really high data-rates.

A reasonable continuation of this work is to integrate the CRC with a configurable encoder and decoder for convolutional codes since they are both used together in many protocols. Using such an accelerator the need for buffering and scheduling handled by a host processor would be relaxed in many wireless applications.

5 Acknowledgments

This work has been sponsored by the graduate school ECSEL, a part of the Swedish Strategic Research foundation.

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”, 3nd 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

[21] U. Nordqvist, T. Henriksson, D. Liu, “CRC Generation for Protocol Processing",, NORCHIP 2000, Turku, Finland