AN1089

Parallel Cyclic Redundancy Check (CRC) for Hotlink®

Associated Project: No
Associated Part Family: CY7B923 / CY7B933
Software Version: NA
Related Application Notes: AN1274

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN1089.

AN1089 discusses using CRC codes to insure data integrity over high-speed serial links, such as Fibre Channel, ESCON and other standards supported by Cypress's CY7B923 and CY7B933 HOTLink devices. It also shows why parity is not useful and then describes the most common CRC codes (CRC-16 and CRC-32) used in high-speed communications systems.

Introduction

This application note discusses using CRC codes to ensure data integrity over high-speed serial links, such as Fibre Channel, ESCON™ and other interfaces supported by Cypress's CY7B923/CY7B933 HOTLink® devices. It also shows why parity and Hamming codes are not useful, and describes common CRC codes used in high-speed communications systems. Finally, algorithms for parallel calculation of CRC–16 and CRC–32 are presented.

Why Not Parity (or Why Some Parallel Interface Practices Don't Apply in the Serial World)?

Some systems go to great lengths to detect data errors. Parity is often used with parallel forms of data, on buses or memories, to detect some errors. It provides a small measure of robustness by detecting certain bit errors with minimal redundancy. However, while parity can detect single-bit errors, it can detect only half of all multiple-bit errors.

Other systems go further, employing Hamming codes to not only detect, but in many instances correct, bit errors. Both of these approaches are applied to data in its parallel form. Unfortunately, the use of a Hamming code requires many more bits of redundancy, per character or word, than parity.

For transmission of data on high-speed serial channels, the most prevalent errors are multi-bit bursts. These multi-bit errors make parity worthless, and severely limit the effectiveness of single-bit correcting Hamming codes.

The large amount of redundancy in a Hamming code (7 bits to protect a 32-bit word) also makes it a poor choice to protect data across a serial link. Transmission of the redundant bits in each word can easily consume a fifth of the available link bandwidth, or require operation of the link at a 20% faster transfer rate to carry the redundant bits.

In reality, bit errors of any type are quite rare in these links (<< 1 in 10^{12} bits). Since these errors cannot generally be corrected by a Hamming code or detected by character parity, the transmission overhead of these types of detection/correction bits becomes a poor use of link bandwidth. In systems where data is sent serially across a link, the data integrity of the link can be much better verified using Cyclic Redundancy Check (CRC) codes.

CRC Codes

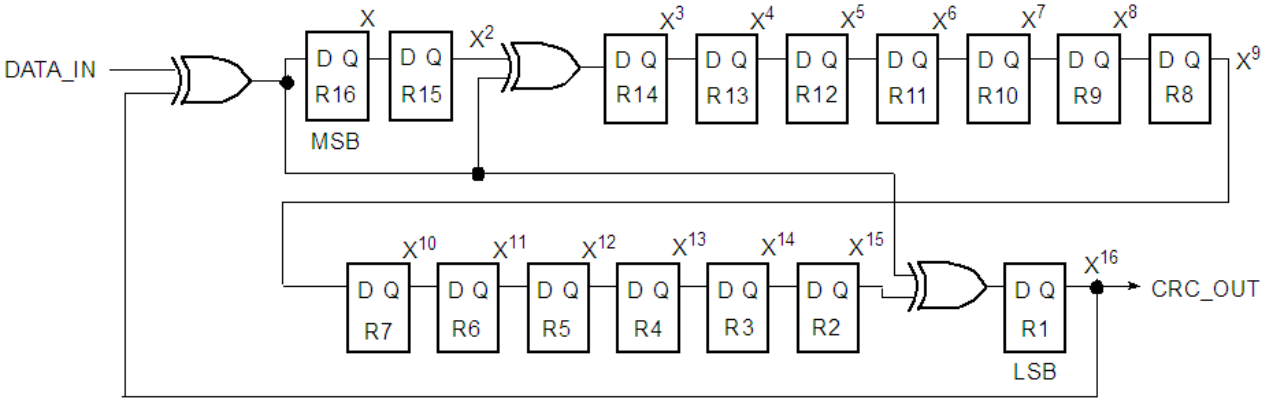
CRC codes make use of a Linear Feedback Shift Register (LFSR) to generate a signature based on the contents of any data passed through it. This signature can be used to detect the modification or corruption of bits in a serial stream.

CRC–16 and CRC–32

In general CRC codes are able to detect:

- All single- and double-bit errors.
- All odd numbers of errors.
- All burst errors less than or equal to the degree of the polynomial used.
- Most burst errors greater than the degree of the polynomial used.

Figure 1. Linear Feedback Implementation of CRC–16



CRC codes have been used for years to detect data errors on interfaces, and their operation and capabilities are well understood. Two codes that have found wide use are CRC–16 and CRC–32. As the names imply, CRC–16 makes use of a 16-bit LFSR, while CRC–32 uses a 32-bit LFSR. Additional information on CRC codes can be found in the references at the end of this application note.

The generator polynomial for CRC–16 is listed in Equation 1, and the polynomial for CRC–32 is listed in Equation 2. These CRC codes are traditionally calculated on the serial data stream using a Linear Feedback Shift Registers (LFSR) built from flip-flops and XOR gates, as shown in Figure 1. The structure for the CRC–32 polynomial is similar to Figure 1, but with twice the number of flip-flops.

$$G(x) = x^{16} + x^{15} + x^{2} + 1 \quad \text{Equation 1}$$

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \quad \text{Equation 2}$$

In these equations, the superscripts identify the tap location in the shift register. The order of the polynomial is identified by the highest order term, and specifies the number of flip-flops in the shift register. Since these polynomials are for modulo-2 arithmetic, each bit-shift is equivalent to a multiply by 2.

Development of a Parallel Implementation

When used with high-speed serial data, especially data which is encoded in the serial domain, it becomes quite difficult to implement the CRC calculations using a shift register. However, it is possible to convert a serial implementation into a parallel form that accumulates multiple bits in each clock cycle. The following paragraphs and tables describe how the CRC–16 polynomial is converted to calculate eight bits at a time (i.e., a byte basis). The CRC–32 polynomial is converted using a similar procedure, with the results calculated 16 bits at a time (on a half-word basis). The results for CRC–32 are presented in Table 5 and Table 6, but without the intermediate calculations. The generation of these intermediate equations are left as an exercise for the reader.

Implementation

First, a few notes:

- Ri is the *i* th bit of the CRC register.
- Ci is the contents of *i* th bit of the initial CRC register, before any shifts have taken place.
- R1 is the least significant bit (LSB).
- The entries under each CRC register bit indicate the values to be XORed together to generate the content of that bit in the CRC register.
- Di is the data input, with LSB input first.
- D8 is the MSB of the input byte, and D1 is the LSB.
- A substitution is made to reduce the table size, such that Xi = Di XOR Ci.

The results of the CRC are calculated one bit at a time and the resulting equations for each bit are examined. The CRC register prior to any shifts is shown in Table 1. The CRC register after a single bit shift is shown in Table 2. The CRC register after two shifts is shown in Table 3.

This process continues until eight shifts have occurred.

Table 4 lists the CRC register contents after eight shifts. Xi was substituted for the various Di XOR Ci combinations. The following properties were used to simplify the equations:

- Commutativity (A XOR B = B XOR A).
- Associativity (A XOR B XOR C = A XOR C XOR B).
- Involution (A XOR A = 0).

A study of Table 4 reveals two interesting facts:

The most-significant byte (bits R16–R9) of the CRC register is only dependent on XOR combinations of the initial low-order byte of the CRC register and the input byte.

The least-significant byte (bits R8–R1) of the CRC register is dependent on the XOR combination of the initial lower eight bits of the CRC register, the input data byte, and the initial contents of the high-order bits of the CRC register.

This allows the next value of the CRC register to be calculated as an XOR of the input data character bits, and a constant determined by the present contents of the CRC register. For example, calculating a new value for R9 is accomplished by calculating X3 and X2 and exclusive-ORing them together.

Table 1. CRC–16 Register prior to any shifts

R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
C16	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1

Table 2. CRC–16 Register after One Shift

R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
C1 D1	C16	C15 C1 D1	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2 C1 D1

Table 3. CRC–16 Register after Two Shifts

R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
C2	C1	C16	C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3
D2	D1	C2	C1												C2
C1		D2	D1												D2
D1		C1													C1
		D1													D1

Table 4. CRC–16 Register after Eight Shifts

R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
X8	X7	X8	X7	X6	X5	X4	X3	C15	C15	C14	C13	C12	C11	C10	C9
X7	X6	X7	X6	X5	X4	X3	X2	X2	X1						X8
X6	X5							X1							X7
X5	X4														X6
X4	X3														X5
X3	X2														X4
X2	X1														X3
X1															X2
															X1

Description of CRC–32 Parallel Algorithm

The parallel algorithm for CRC–32 is derived in the same manner as CRC–16. The differences here are that data is now handled 16 bits (a half-word) at a time, the CRC register is now 32 bits in length, and a different polynomial is used.

Table 5 contains the XOR information for the least-significant half-word (LSHW) of the CRC–32 register after 16 shifts, and Table 6 contains the XOR information for the most-significant half-word (MSHW) of the CRC–32 register after 16 shifts. Again, note that the MSHW only depends on XOR combinations of the initial lower-order bits of the CRC–32 register and the input data. The LSHW depends on XOR combinations of the initial lower-order bits of the CRC–32 register, the input data, and the initial MSHW of the CRC–32 register.

Table 5. CRC–32 Register (LSW) after 16 Shifts with Xi Substitution

R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
C32	C31	C30	C29	C28	C27	C26	C25	C24	C23	C22	C21	C20	C19	C18	C17
X1	X2	X1	X5	X4	X3	X2	X1	X6	X5	X6	X5	X4	X3	X2	X1
X3	X7	X6	X7	X6	X5	X5	X7	X9	X8	X10	X9	X8	X7	X6	X5
X8	X9	X8	X8	X7	X6	X7	X10	X14	X13	X12	X11	X10	X9	X8	X7
X10	X10	X9	X9	X8	X7	X16	X15	X15	X14	X13	X12	X11	X10	X9	X8
X11	X11	X10	X13	X12	X11		X16			X16	X15	X14	X13	X12	X11
X12	X15	X14													
X16															

Table 6. CRC–32 Register (MSW) after 16 Shifts

R32	R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17
X4	X3	X2	X1	X1	X3	X2	X1	X4	X3	X2	X1	X1	X1	X1	X2
X6	X4	X3	X2	X4	X6	X5	X6	X5	X4	X3	X2	X2	X2	X3	X4
X7	X5	X7	X6	X5	X8	X8	X8	X6	X5	X4	X3	X3	X4	X5	X5
X10	X7	X8	X7	X8	X10	X9	X9	X8	X7	X6	X4	X5	X6	X6	X7
X16	X9	X9	X8	X10	X11	X10	X11	X12	X11	X10	X5	X7	X7	X8	X8
	X10	X10	X9	X12	X12	X11	X13	X13	X12	X11	X6	X8	X9	X9	X9
	X15	X14	X13	X13	X13	X12	X14	X15	X14	X13	X7	X10	X10	X10	X12
	X16	X15	X14	X14	X15	X14	X16	X16	X15	X14	X9	X11	X11	X13	X13
		X16	X15	X16	X16	X15					X12	X12	X14	X14	
											X13	X15	X15		
											X16	X16			

Summary

This application note shows how to calculate a parallel implementation of any CRC polynomial, and the equations for CRC–16 and CRC–32 are provided. Both designs easily operate at the fastest character rate supported by Cypress's HOTLink devices.

Additional information on usage of CRC polynomials may be found in the following references and in the Cypress application note titled "Drive ESCON With HOTLink."

References

A. Perez, "Byte-wise CRC Calculations," IEEE MICRO, June 1983, pp. 40-50.

A. K. Pandeya and T. J. Cassa, "Parallel CRC Lets Many Lines Use One Circuit," Computer Design, Sept. 1975, pp 87-91. R. Swanson, "

Understanding Cyclic Redundancy Codes," Computer Design, Nov. 1975, pp. 93-99.

Document History

Document Title: AN1089 - Parallel Cyclic Redundancy Check (CRC) for HOTLink®

Document Number: 001-27960

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1467003	FRE	10/11/2007	New application note.
*A	3396139	SAAC	10/05/2011	Updated Abstract. Removed “Implementation Issues for CRC-16 Parallel Algorithm”. Removed “Implementation Issues for CRC-32 Parallel Algorithm”. Updated to new template.
*B	4573006	YLIU	11/18/2014	Updated to new template. Completing Sunset Review.
*C	5879945	AESATMP9	09/11/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709