AN-1186 Cyclic Redundancy Check Generator Unit for 1-wire Protocol

In any application where communications between devices are needed, data errors are one of the most important problems that must be analyzed in order to obtain a reliable communication system. To reduce data errors, there are different methods that can be used to detect and, sometimes, correct the errors.

One of the most used methods for detecting errors in binary digital communication channels and storage devices is called Cyclic Redundancy Check (CRC).

This technique is based on an error-detecting code calculated on the transmitter side based on the binary data to be transmitted. This code is appended at the end of the stream and is received as a part of it on the receiver side.

The calculation of the error code is repeated on the receiver and, if the received code and the calculated code do not match, an error is detected and the data is discarded or a corrective action can be taken.

This method is called redundant because a code is appended to the stream without adding more information to it.

One of the most important advantages of the CRC method, and the main reason for it being used in many digital communication systems, is that it is very simple to calculate and compare the error code in digital hardware.

In this application note, an 8-bit length CRC generator is implemented based on the specifications for a 1-wire communication protocol. For this purpose, SLG46533V GreenPAK™ is used.

The implemented system has a data and a clock input to obtain the binary data stream that will be transmitted, and from which the error-code must be calculated. It also has a reset input, which initializes the generator for a new data stream. The output of this system is the resulting code, as an 8-bit parallel output.

CRC Generation

The aim of error-codes based on Cyclic Redundancy Check (CRC) is based on adding a fixed-length check value to a variable length binary stream in order to obtain a method for verifying the integrity of the data. This technique is designed to account for burst errors, where contiguous sequences of erroneous bits are present in a message. This is important because burst errors are common transmission errors in many communication channels, including electromagnetic and storage devices.

From a mathematical point of view, a CRC code is specified by a *generator polynomial*. The CRC code generation can be thought as a binary polynomial division. The message (described as a polynomial) is the dividend and the generator polynomial is the divisor. On this operation, the remainder is the resultant error-code. The length of the remainder is always less than the length of the generator polynomial, determining the length of the error-code.

The message polynomial can be calculated by considering each bit (1 or 0) of the message as the coefficients of a complete polynomial with m-1 degree, where m is the length of the message.

A CRC code is commonly called n-bit CRC when the error code is n bits long. For a particular n value, multiple CRCs can be used, each with a different polynomial. If a polynomial has n-degree, it produces an error-code with n bits.

As mentioned earlier, one of the most important advantages of CRC is that it is very simple to calculate and compare the error code in digital hardware. This is because the binary division can be implemented in hardware with a shift register arrangement with feedback paths. These paths are the coefficients of the generator polynomial. The number of stages in the shift register for the hardware description is equal to the order of the generator polynomial.

The shift register and feedback can be implemented with a programmable logic device, so the GreenPAK ICs are ideal to generate the CRC code for any protocol.

In this application note, the 8-bit CRC for 1-Wire Communication Code is generated. The 1-Wire CRC is used for checking the 64-bit ROM code written into each 1-Wire product. This ROM code consists of an 8-bit family code written into the least significant byte, a unique 48-bit serial number written into the next 6 bytes, and a CRC value that is computed based on the preceding 56 bits of ROM and then written into the most significant byte.

CRC Code (MSB)	Serial number	Family Code (LSB)
1 byte	6 bytes	1 byte

Table 1. The ROM code bytes

This code is capable of detecting these error types:

- Any odd number of errors anywhere within the 64-bit number.
- All double-bit errors anywhere within the 64-bit number.
- Any cluster of errors that can be contained within an 8-bit "window" (1-8 bits incorrect).
- Most larger clusters of errors.

The generator polynomial for the CRC is:

$$x^8+x^5+x^4+1$$

In table 2, it can be seen an example of how to calculate the CRC of an 8-bit stream. The stream is 0x56 (01010110) and is entered into the CRC calculator from the LSB to the MSB.

CRC shift register	Input
00000000	0
00000000	1
10001100	1
11001010	0
01100101	1
00110010	0
00011001	1
00001100	0
00000110	

Table 2. CRC of a Stream

In table 2, the shift register state and the input at each step is shown. The state of the CRC shift register after the 8^{th} bit is entered is the result. So the result which corresponds to the CRC code for 0x56 is 0x06.

For a representation of how a CRC shift register works, we recommend watching this video.

Table 3 & 4 display an example of a packet of information and its CRC:

Stream	CRC
0423077C300000	2E

Table 3. CRC of a Stream

Stream	CRC
0423077C300001	70

Table 4. CRC of a Stream

Logic Implementation

As mentioned in the previous section of this application note, generating the CRC error code can be implemented with programmable logic based on Flip Flops arranged as a Linear Feedback Shift Register.

Because of the serial characteristics of a 1-wire communication, one bit from the binary stream is loaded in the shift register per clock cycle. Once the entire byte is loaded, the data is XOR'd with the output of the last stage of the shift register. This process can be seen in Figure 1.

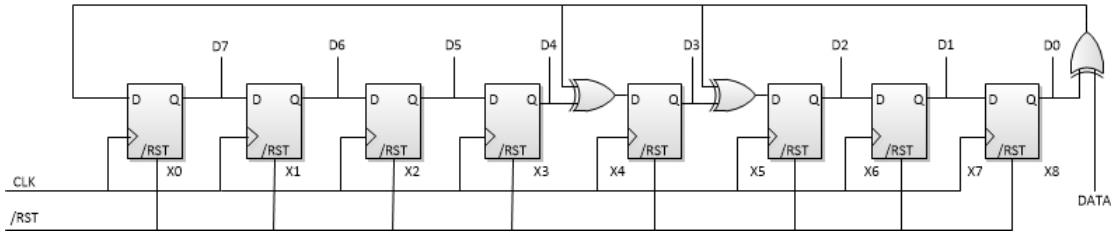

Figure 1. Logic Diagram of 1-Wire CRC Generator Unit

If the logic diagram shown in Figure 1 is analyzed from a mathematical point of view, the 8-stage shift register is the dividing circuit where the input data acts as the dividend, and the shift register data with feedback paths acts as the divisor, implementing the generator polynomial. The division result is discarded and the remainder is the CRC error code which will be appended to the binary stream.

The stages where the data is XOR'd with the last stage of the shift register corresponds to the non-zero coefficients of the generator polynomial.

As seen in Figure 1, the inputs to the system are the serial data input, the clock, and a reset signal. This signal initializes the shift register for a new binary stream by setting all of the flip-flops outputs to 0.

The result of the CRC Generator is a byte corresponding to the output of each of the Flip-Flops of the shift register.

The described characteristics of the 1-wire CRC error code can be extended to other CRC specifications for other communication protocols. This is because all of them are based on a shift register and feedback paths. If different protocols are analyzed, the number of stages and the feedback paths may change, based on the corresponding generator polynomial.

This logic diagram of a CRC generator unit is ideal to be implemented with a GreenPAK IC. This is because with only one integrated circuit, all of the flip-flops and exclusive-or gates can be used to implement a peripheral that may not be included in the microprocessor used to implement the communications.

An important characteristic of this implementation is the order of bits. This application note implements a CRC generator unit which expects to receive the least significant bit of the binary stream first. If the order of bits transmission is not correct, the result will be incorrect and the CRC will not correspond to the stream.

Implementation and Configuration

To implement the CRC Generator unit, we used a GreenPAK SLG46533V. This GreenPAK has 15 D-flip flops, so it's ideal to implement this 8-bit length CRC and other CRC codes for other communication protocols with longer error codes.

In Figure 2 the logic diagram of the 8-bit length shift register with feedback paths is shown, in accordance to the logic schematic from Figure 1.

Figure 2. 8-bit Length Shift Register

The feedback paths correspond to X8, X5, X4 and X0. It must be taken into account that the data is processed with the less significant bit first, so the order of the bits output must be reversed from the order of the generator polynomial. This is the reason why, e.g., X7 corresponds to output bit 1 of the error code.

Each DFF clock input of the shift register is connected to the clock input of the system. The output of the flip-flop is configured to be non inverted Q and connected to the next stage of the shift register, so the register and feedbacks are implemented together. Also, each flip flop nSet/nReset option is configured to be Reset, and it is connected to the reset input signal of the generator unit, so all of the flip flops can be initialized to generate a new CRC when the user sets the reset input to a low level.

The configuration of each flip flop can be seen in Figure 3.

In order to obtain the corresponding gates for the feedback paths, 2-bit LUTs are used. Each LUT has its truth table configured as an XOR gate and their inputs are connected to the 8 stage and the corresponding previous stage. In the case of the 2-bit LUT2, the input is connected to the data stream input because it corresponds to X8 term of the generator polynomial. The configuration of each LUT can be seen in Figure 4.

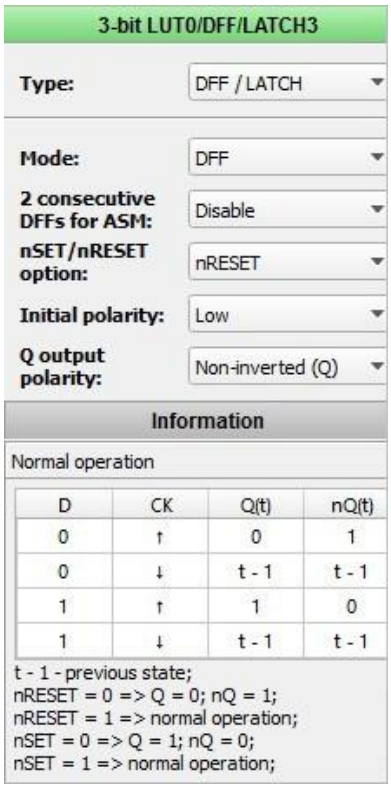

Figure 3. Flip Flop configuration

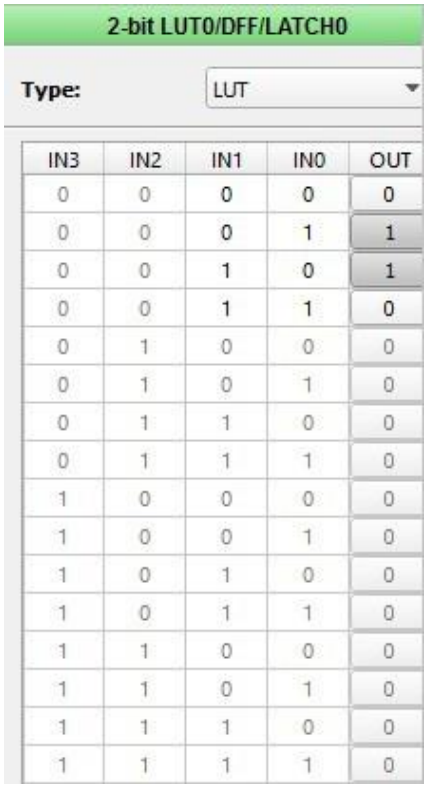

Figure 4. 2-bit LUT configuration

As mentioned earlier, the output of the CRC Generator Unit is the corresponding error code byte obtained from each flip flop of the shift register. Based on this idea, each output of the flip flops is connected to one output pin of the GreenPAK. The entire implementation is shown in Figure 5.

Tests and Conclusion

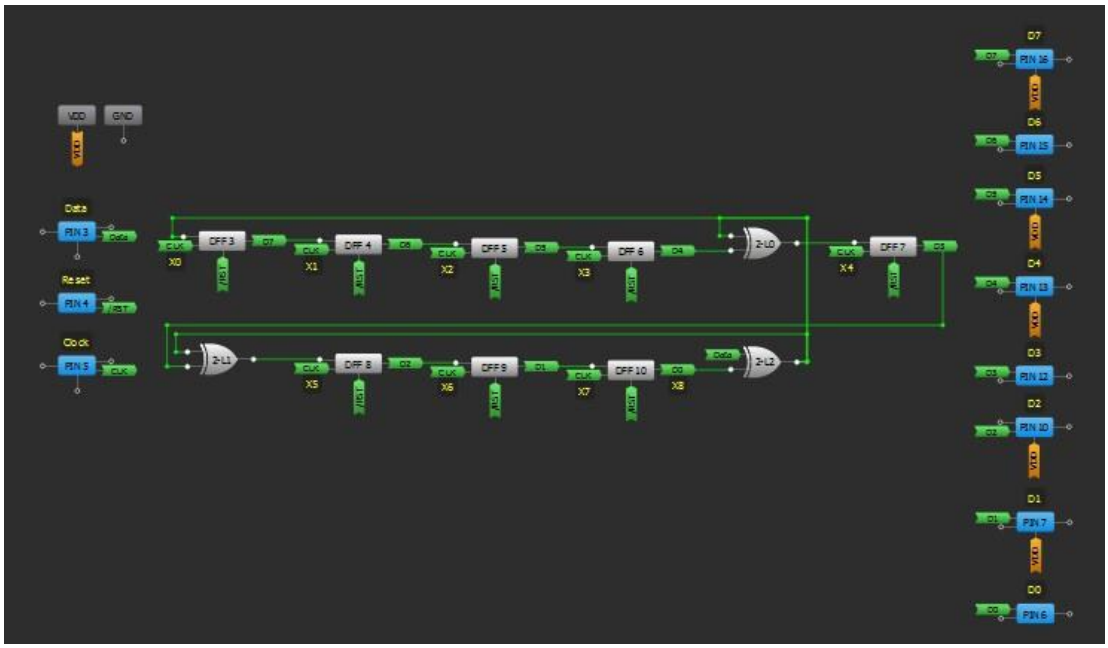

Figure 5. CRC Generator unit block diagram

To test the implementation, 16-bit binary streams were transmitted to the CRC Generation Unit to verify the calculated error code. The analyzed binary stream is:

0x0423 = 0000010000100011

This binary stream was inspected with a logic analyzer. In Figure 6, the transmitted binary stream and the clock signal are shown. The stream is transmitted with the LSB of each byte first.

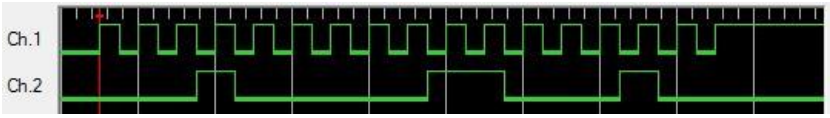

Figure 6. 16-bit binary stream

Channel 1 corresponds to the clock signal and Channel 2 corresponds to Data Input. The first clock cycle is used to initialize the generator with a low reset input.

The corresponding CRC for the stream is:

CRC = 0xFA = 11111010

In Figure 7, the eight outputs of the CRC generator Unit analyzed with the logic analyzer are shown. Channel 1 corresponds to the clock signal and Channel 2 to 9 correspond to bit 0 output to bit 7 output, that is D0 to D7. The first clock cycle is used to initialize the generator with a low reset input.

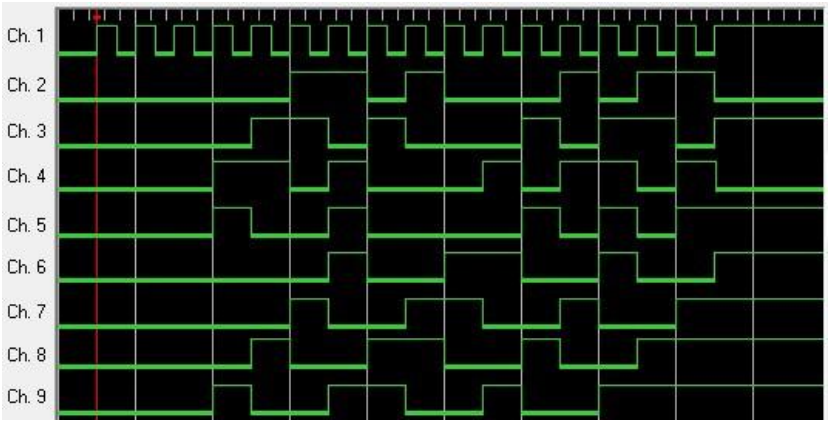

Figure 7. 8-bit 1-wire CRC error code output

After 16 clock cycles, the outputs correspond to the CRC error code as was expected.

In this application note, we implemented a CRC Generator Unit using SLG46533V. The user must provide the binary data stream to be processed by transmitting it to the generator synchronized to a clock signal. The user can initialize the CRC generation for a new stream with an active-low reset signal. The output of the system is a parallel output corresponding to the 8-bit length 1-wire CRC error code.

With this implementation, the user can use a GreenPAK IC to obtain an external peripheral which calculates the CRC for a transmitted/received binary stream that can be used with a microprocessor which doesn’t have an integrated CRC unit.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.