Application Note

A Simple, Browser-Based, CRC Calculator

Joseph Wu

ABSTRACT

In many TI data converter devices, cyclic redundancy check (CRC) is used in digital communications to verify reads from and writes to the device. CRC is used in communication that typically has a fixed byte length, and the check is based on a resulting calculation from the communication sequence. This application note presents an example CRC calculator that can be run on a browser using JavaScript™. This note starts with a basic explanation and presentation of the different types and characteristics of CRC. Using an example, this note shows how to create a simple, browser-based, CRC calculator. The example is expanded to show how the code can be easily modified to cover a variety of CRC types with a limited amount of coding experience required.

Table of Contents

List of Figures

Trademarks

1 Introduction

CRC is an error-detection code that is commonly used in digital communication. CRC is often used in TI data converters to verify the communication between the device and the microcontroller. CRC uses the transmission data and appends an initial value (or seed value). Then a generator polynomial is used in a calculation to generate a CRC code. If the transmitter and receiver generate the same code, then the data is good. If the same code is not generated, the data is bad, and the transmission data likely has an error.

There are many types of CRC used for error detection. These CRCs use different bit lengths, initial values, and generator polynomials to calculate the code. Parity checks are basically a single bit CRC calculation. CRC codes of eight, 16, and 32 bits are common, and the distinguishing features are the initial values and the generator polynomial values.

The CRC calculation is a modulo-2 division of the data transmission by the generator polynomial. This division is performed as a bitwise XOR. The calculation starts with the transmission data string by appending an initial value of all zeros or all ones. The division is performed, and the resulting remainder is the CRC code. If the remainder of the calculation does not match the received value, then a transmission error of the data has occurred.

Data transmissions are often sent most significant bit first, while others (UART, for example) are sent as least significant bit first. Bit must be considered when making the CRC check when implemented as part of the calculation. Some CRC algorithms reflect the input data or output CRC code by reversing the bit order of each byte.

Many device datasheets give a detailed description of the CRC function and calculation used in the device. For more information about using CRC functions see *Communication Methods for Data Integrity Using Delta-Sigma Data Converters* or *Cyclic Redundancy Check Computation: An Implementation Using the TMS320C54x*.

2 Browser Based CRC Calculator Example

Clip out the example code in Figure 2-1 and paste into a text editor. Save the code as a hypertext markup language (.html) file.

```
<!-- Copyright (c) 2026 Texas Instruments Incorporated. All rights reserved. -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TI Data Converter CRC Calculator</title>
    <style>
        body {
            font-family: "Courier New", Courier, monospace;
        }
    </style>
</head>
<body>
    <h1><span style="font-family: 'Arial';">Simple TI Data Converter CRC Calculator</span></h1>
    <hr style="border-top: 3px solid red;">
    <label for="hexInput"><span style="font-family: 'Arial';">Enter Hex String:</span></label>
    <input type="text" id="hexInput" value="ABC123"> <!-- Enter hex value for calculation -->
    <button onclick="calculateCRC()">Calculate CRC</button> <!-- Calculate CRC button -->
    <p>CRC-8-CCITT Polynomial: x<sup>8</sup> + x<sup>2</sup> + x + 1 (07h)</p> <!-- Polynomial -->
    <p>Initial value: 00h</p> <!-- Initial value -->
    <p>Result: <span style="text-transform:uppercase; color:red" id="crcResult"></span></p>
    <hr style="border-top: 3px solid red;">
    <footer><p>Copyright &copy; 2026 Texas Instruments Incorporated. All rights reserved.</p></footer>
    <script>
        function calculateCRC() {
            const hexString = document.getElementById('hexInput').value;
            if (!/^[0-9a-fA-F]+$/.test(hexString)) {
                alert('Please enter a valid hex string.');
                return;
            }
            const buffer = hexToBytes(hexString);
            // Call CRC function
            document.getElementById('crcResult').innerText = crc8CCITTZeroes(buffer).toString(16);
        }
        function hexToBytes(hex) {
            const bytes = [];
            for (let i = 0; i < hex.length; i += 2) {
                bytes.push(parseInt(hex.substr(i, 2), 16));
            }
            return bytes;
        }
        function crc8CCITTZeroes(buffer) {
            let crc = 0x00; // Initial value
            for (let byte of buffer) {
                crc ^= byte;
                for (let j = 0; j < 8; j++) {
                    if (crc & 0x80) { // Checks if the MSB is set
                        crc = (crc << 1) ^ 0x07; // CRC-8-CCITT polynomial
                    } else {
                        crc <<= 1;
                    }
                }
                crc &= 0xFF; // Sets CRC to 8 bits long
            }
            return crc;
        }
    </script>
</body>
</html>
```

Figure 2-1. HTML Code for Browser-Based CRC Calculator

Clipping and pasting the text from the figure above can strip out the indents for each line. Add the spaces for easier viewing, but the code runs without the indents.

Using WordPad™, open an empty file and paste the text. Select the *File* menu and use *Save as*. Go to Save as type and select *All files* (*.*). The encoding is UTF-8. Enter a file name (crc.html) and click Save to place the html file in a directory you have chosen. After saving this file on the computer, click on the icon, and the html runs on the default browser. A hex string of ABC123 is the default value. The browser window looks like Figure 2-2 at start up.

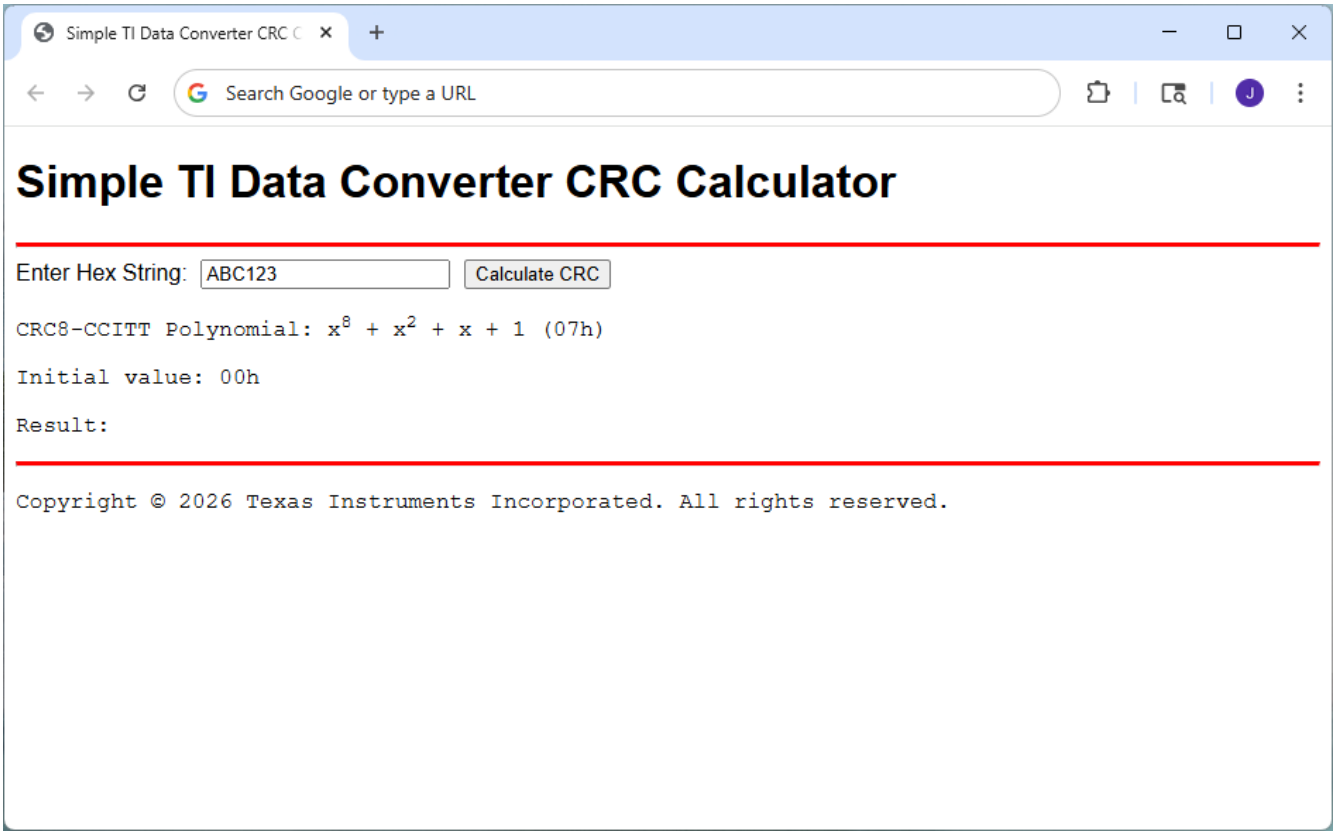

Figure 2-2. Browser Window for CRC Calculator

To run the calculator, start with the default hex string of ABC123 and click on the Calculate CRC button and the browser returns B5 as the result in Figure 2-3. Both the input and output are in hex, without spaces.

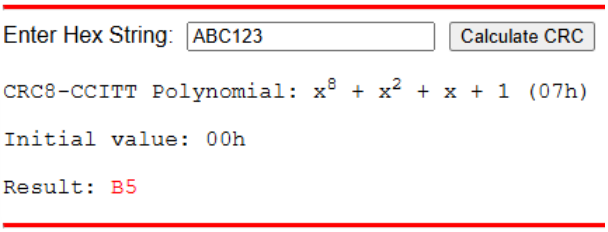

Figure 2-3. CRC Result for 0xABC123, CRC-8-CCITT, Initial Value 0x00

Replace the existing ABC123 hex string to calculate a new value. Use a hex string without non-hex characters. Both upper and lower case characters are accepted. A value of 020026 is selected in the example below. This string is the AFE881H1 write register command to disable CRC. Click on Calculate CRC and a new result appears as in Figure 2-4.

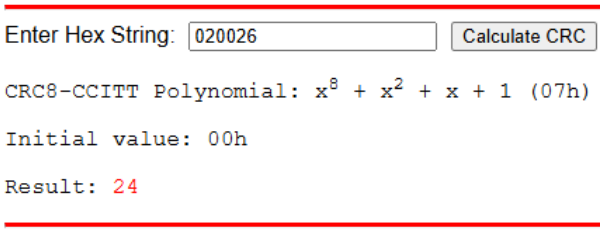

Figure 2-4. CRC Result for 0x020026, CRC-8-CCITT, Initial Value 0x00

The CRC value 0x24 is calculated from 0x020026. This value is appended to the command to disable the CRC in the AFE881H1 after startup of the device.

2.1 CRC-8-CCITT, 0x00 Initial Value

The previous browser-based example takes in a hex string of bytes entered through the browser running an HTML file. The script converts the hex string into an array of bytes. The script makes the calculation for CRC-8-CCITT with a 0x00 initial value and reports the result in the browser. The code processes the input as bytes, so the input must be an even number of hex characters.

The HTML main body section starts with a box to enter in a hex string with a default input of ABC123. The body section also creates the Calculate CRC button to generate the CRC. Two text lines describe the CRC, showing the generator polynomial and the initial value used for the calculation. The result is displayed on the bottom line.

Table 2-1 lists the relevant functions of the JavaScript code from the example.

Table 2-1. CRC Script Descriptions

Function	Description
calculateCRC()	Pulls in the text from the input box and verifies that the input has only hex characters, calls hexToBytes() and crc8CCITTZeroes()
hexToBytes()	Converts the input characters to an array of bytes
crc8CCITTZeroes()	Calculates CRC-8-CCITT with a zero initial value from the array created from hexToBytes()

The example calculates CRC-8-CCITT using 0x00 as an initial value. The CRC-8-CCITTZeroes() function is taken out of the example and shown in the following code in Figure 2-5.

```
function crc8CCITTZeroes(buffer) {
    let crc = 0x00; // Initial value
    for (let byte of buffer) {
        crc ^= byte;
        for (let j = 0; j < 8; j++) {
            if (crc & 0x80) { // Checks if the MSB is set
                crc = (crc << 1) ^ 0x07; // CRC-8-CCITT polynomial
            } else {
                crc <<= 1;
            }
        }
        crc &= 0xFF; // Sets CRC to 8 bits long
    }
    return crc;
}
```

Figure 2-5. CRC-8-CCITT, Initial Value 0x00, Calculation Function

The initial value is defined in the first line of the function. This CRC uses a polynomial of $x^8 + x^2 + x + 1$ represented as 0x07 in crc8CCITTZeroes(). The polynomial is set up in the bitwise XOR (indicated by the “^”) with the CRC value shown in the code snippet below.

Details of CRC-8-CCITT are described in Table 2-2.

Table 2-2. CRC-8-CCITT, Initial Value 0x00

CRC	Polynomial	Initial Value	Devices	CRC for 0xABC123
CRC-8-CCITT	$x^8 + x^2 + x + 1$ (0x07)	0x00	AFE881H1, AFE882H1, AFE88101, AFE88201, DAC8741H, DAC8742H, DAC8750, DAC8760, DAC8771, DAC8775, DAC80504, DAC80508, DAC81401, DAC81402, DAC81404, DAC81408, DAC81416, ADS124S08, ADS125H02, ADS1263, ADS127L01, ADS7028, ADS7038, ADS7128, ADS7138	0xB5

The polynomial and initial values are shown and TI devices that use this specific CRC are listed as well. The last column also shows the result of a CRC calculation using ABC123 using this CRC function. The resulting value can be used to check the code for the CRC.

2.2 CRC-8-CCITT, 0xFF Initial Value

The example CRC code can be changed to calculate other CRC types. For other 8-bit CRC calculations, check the initial value and the CRC polynomial.

In different CRC versions, the initial value is set as either 0x00 or 0xFF. To alter the code to check for the same polynomial, but using 0xFF as the initial value, change the value of *let crc* in the second line of the function. The function is renamed to crc8CCITTOnes() to show a different initial value. Figure 2-6 shows the code with a new initial value of 0xFF.

```
function crc8CCITTOnes(buffer) {
    let crc = 0xFF; // Initial value FFh
    for (let byte of buffer) {
        crc ^= byte;
        for (let j = 0; j < 8; j++) {
            if (crc & 0x80) { // Checks if the MSB is set
                crc = (crc << 1) ^ 0x07; // CRC-8-CCITT polynomial
            } else {
                crc <<= 1;
            }
        }
        crc &= 0xFF; // Sets CRC to 8 bits long
    }
    return crc;
}
```

Figure 2-6. CRC-8-CCITT, Initial Value 0xFF, Calculation Function

In the body section of the html, make sure that the description is changed to reflect the new initial value of 0xFF. This new CRC-8-CCITTOnes() function with 0xFF initial value is also updated at the end of the calculateCRC() function.

Table 2-3 lists the new CRC-8-CCITTOnes() with a list of TI devices that use this CRC algorithm. The last column provides the calculation for ABC123 to check your code.

Table 2-3. CRC-8-CCITT, Initial Value 0xFF

CRC	Polynomial	Initial Value	Devices	CRC for 0xABC123
CRC-8-CCITT	$x^8 + x^2 + x + 1$ (0x07)	0xFF	ADS1261, ADS127L11, ADS127L14, ADS127L18, ADS127L21, ADS7066, ADS7067, TMAG5173-Q1, TMP114	0x9E

After making changes to the JavaScript, run the CRC-8-CCITT 0xFF initial value code. Figure 2-7 shows the new result for the CRC.

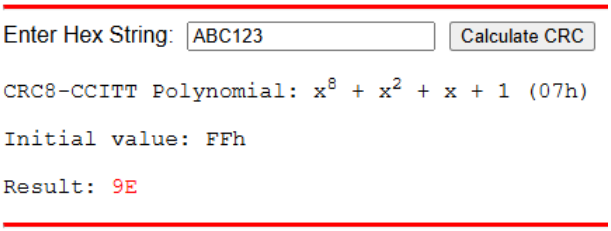

Figure 2-7. CRC Result for 0xABC123, CRC-8-CCITT, Initial Value 0xFF

2.3 CRC-8-One-Wire, 0xFF Initial Value

The polynomial can also be easily changed in the script. For example, the original crc8CCITTZeroes() example has the if statement for the CRC polynomial shown in Figure 2-8.

```
                if (crc & 0x80) { // Checks if the MSB is set
                    crc = (crc << 1) ^ 0x07; // CRC-8-CCITT polynomial
                } else {
                    crc <<= 1;
                }
```

Figure 2-8. CRC Polynomial Code

The 0x07 in the XOR is the polynomial and represents the bottom three terms of $x^8 + x^2 + x + 1$. The x^8 term is the MSB that triggers the XOR.

If the algorithm is changed to CRC-8-One-Wire, a new $x^8 + x^5 + x^4 + 1$ polynomial is represented by 0x31 and the initial value is 0xFF. Renaming the function, start the crc8OneWireOnes() function with the same *crc = 0xFF* at the beginning. Change the polynomial calculation with a new XOR as shown in Figure 2-9.

```
        function crc8OneWireOnes(buffer) {
            let crc = 0xFF; // Initial value
            for (let byte of buffer) {
                crc ^= byte;
                for (let j = 0; j < 8; j++) {
                    if (crc & 0x80) { // Checks if the MSB is set
                        crc = (crc << 1) ^ 0x31; // CRC-8-OneWire polynomial
                    } else {
                        crc <<= 1;
                    }
                }
                crc &= 0xFF; // Sets CRC to 8 bits long
            }
            return crc;
        }
```

Figure 2-9. CRC-8-OneWire, Initial Value 0xFF, Calculation Function

Details of the CRC-8-One-Wire algorithm are listed in Table 2-4.

Table 2-4. CRC-8-One-Wire, Initial Value 0xFF

CRC	Polynomial	Initial Value	Devices	CRC for 0xABC123
CRC-8-One-Wire	$x^8 + x^5 + x^4 + 1$ (0x31)	0xFF	LMP9007x, LMP90080, LMP9009x, LMP90100, HDC302x, HDC302x-Q1	0xF4

Again, when making changes to the JavaScript, also change in the body text describing the CRC function. Then change the calculateCRC() function to call the new CRC calculation. The resulting CRC-8-One-Wire calculation appears in the browser window as shown in Figure 2-10.

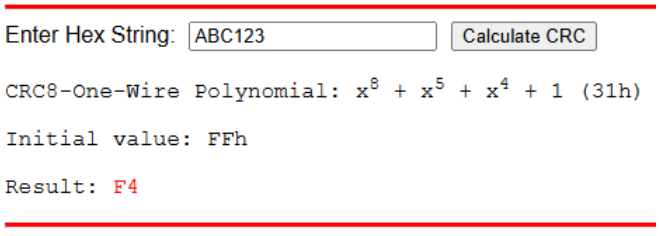

Figure 2-10. CRC Result for 0xABC123, CRC-8-OneWire, Initial Value 0xFF

2.4 CRC-16-CCITT, 0xFFFF Initial Value

In this CRC-16-CCITT algorithm, the initial value is set to 0xFFFF. The changes in the code required a new calculation, which can be seen in Figure 2-11.

```
function crc16CCITTOnes(buffer) {
    let crc = 0xFFFF; // Initial value, 16 bits
    for (let byte of buffer) {
        crc ^= (byte << 8); // Aligns byte to MSB
        for (let j = 0; j < 8; j++) {
            if (crc & 0x8000) { // Checks if the MSB is set, 16 bits
                crc = (crc << 1) ^ 0x1021; // CRC-16-CCITT polynomial
            } else {
                crc <<= 1;
            }
        }
        crc &= 0xFFFF; // Sets CRC to 16 bits long
    }
    return crc;
}
```

Figure 2-11. CRC-16-CCITT, Initial Value 0xFFFF, Calculation Function

In this new code, the function is renamed to crc16CCITTOnes(), identifying the type of CRC and the initial value. Within the function, the initial value is set to 0xFFFF and the input data is left shifted by 12 bits to align the most significant bit. The MSB check for the XOR is changed from 0x80 to 0x8000.

CRC-16-CCITT uses the polynomial $x^{16} + x^{12} + x^5 + 1$, so the XOR changes from 0x07 in the original code to 0x1021. At the end of the function, the CRC code is set to 16 bits long with a bitwise AND before being returned by the function. In the body part of the html, correct any changes to the text describing the new CRC calculation. Then change calculateCRC() to call the crc16CCITTOnes() function.

Table 2-5 lists the CRC-16-CCITT characteristics and lists several TI devices that use this CRC.

Table 2-5. CRC-16-CCITT, Initial Value 0xFFFF

CRC	Polynomial	Initial Value	Devices	CRC for 0xABC123
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (0x1021)	0xFFFF	ADS122C04, ADS122U04 (reflected), ADS131A04, ADS131M04, ADS131B04-Q1, AMC131M03, TMP126, TMP126-Q1	0xB095

The new CRC-16-CCITT calculator result appears as in the window as shown in Figure 2-12.

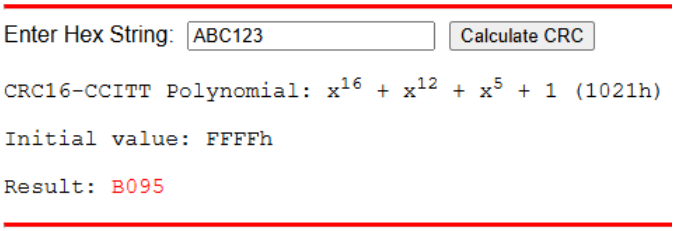

Figure 2-12. CRC Result for 0xABC123, CRC-16-CCITT, Initial Value 0xFFFF

From Table 2-5, the ADS122U04 uses a reflected version of the CRC-16-CCITT calculation. The following section describes reflection in the CRC calculation (albeit using the CRC-8-OneWire algorithm). Reflection can be added with a simple function for input or output reflection in the CRC calculation.

2.5 Input and Output Data Reflection

In some CRC algorithms, the input or output data are reflected. In these cases, bytes enter in same order but the bits are reflected by reversing the bit order of each byte. This reflection is sometimes used for improving efficiency in the calculation or in reducing cost in shift register hardware implementation. In this example, the CRC-8-OneWire algorithm is altered to reflect both the input and output data.

Starting with the previous example created with CRC-8-One-Wire, a reflect() function is created for reflecting the data. The function in Figure 2-13 takes in a value (in this example, a byte) with a defined width and reflects the data.

```
function reflect(value, width) {
    let result = 0;
    for (let r = 0; r < width; r++) { // Steps through the width of the input
        if (value & (1 << r)) {
            result |= (1 << (width - 1 - r)); // Sets bit in the reflected order from the input
        }
    }
    return result;
}
```

Figure 2-13. Data Reflection Function

The result of the reflect() function starts as 0h and steps through setting bits in the reflected order from the input data. Starting with an input byte, the output result is a bit reflection of the byte.

The new crc8OneWireZeroesInOutReflect() function has the same CRC polynomial ($x^8 + x^5 + x^4 + 1$) as shown in a previous section but uses a different initial value (0x00). The only other changes are two additions of the reflect() function for calculating the CRC with input and output reflection. CRC code with input and output reflection is shown in Figure 2-14.

```
function crc8OneWireZeroesInOutReflect(buffer) {
    let crc = 0x00; // Initial value
    for (let byte of buffer) {
        byte = reflect(byte, 8) // Reflect input
        crc ^= byte;
        for (let j = 0; j < 8; j++) {
            if (crc & 0x80) { // Checks if the MSB is set
                crc = (crc << 1) ^ 0x31; // CRC-8-OneWire polynomial
            } else {
                crc <<= 1;
            }
        }
        crc &= 0xFF; // Sets CRC to 8 bits long
    }
    return reflect(crc, 8); // Reflect output
}
```

Figure 2-14. CRC-8-OneWire, Initial Value 0x00, Input and Output Data Reflected Calculation Function

The input byte is reflected in the fourth line of the function as each byte is called. At the end of the crc8OneWireZeroesInOutReflect(), the resulting CRC also reflected. If the input reflection is not needed. The fourth line byte reflection can be removed. If the output reflection is not needed, the crc is returned at the end, not the reflected crc.

Details of the CRC-8-One-Wire algorithm with 00h initial value and input and output reflection listed in Table 2-6.

Table 2-6. CRC-8-One-Wire, Initial Value 0x00, Input and Output Reflected

CRC	Polynomial	Initial Value	Devices	CRC for 0xABC123
CRC-8-One-Wire Input reflected, output reflected	$x^8 + x^5 + x^4 + 1$ (0x31)	0x00	TMP1826, TMP1827	0x86

As in the previous examples, if a user is making changes to the JavaScript, also change in the body text describing the CRC function including the input and output reflection. Then change the calculateCRC() function to call the new crc8OneWireZeroesInOutReflect().

The resulting CRC-8-One-Wire calculation with input and output reflection appears in Figure 2-15.

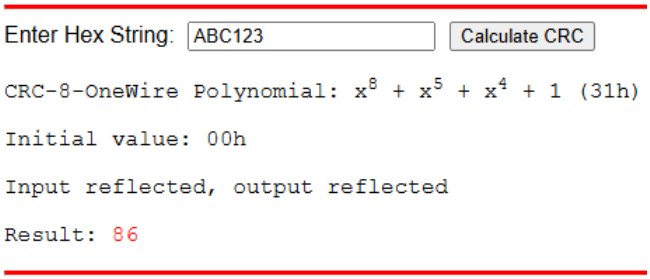

Figure 2-15. CRC Result for 0xABC123, CRC-8-OneWire, Initial Value 0x00, Input and Output Data Reflected

3 Summary

In this application note, the example code can be used to generate a simple CRC calculator. By clipping the code out of this applications note and pasting into an html file, this CRC code can be run in a browser. The original example runs a CRC-8-CCITT calculator, which is used in many TI devices. With a few small modifications, the code can be adapted to CRC algorithms of different initial values, generation polynomials, and CRC lengths. The last example also shows how to add reflection to the input data or the output CRC calculation by adding a new function to the script. These modifications can be used for CRC calculators used in many other TI devices.

4 References

- Texas Instruments, *Communication Methods for Data Integrity Using Delta-Sigma Data Converters*, application note.
- Texas Instruments, *Cyclic Redundancy Check Computation: An Implementation Using the TMS320C54x*, application note.
- Texas Instruments, *Implementation of CRC for ADS7066*, application note.
- Texas Instruments, *ADS7066 CRC Calculator*, calculator.
- Texas Instruments, *ADS7xx8 CRC Calculator*, calculator.
- Texas Instruments, *PADC Design Calculator Tool*, calculator.
- Texas Instruments, *CRC Tool, Online Analog Engineer's Calculator*, calculator.
- Texas Instruments, *Precision ADC Github*, webpage.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL 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 AND 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 skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025