Application Note

一款基于浏览器的简单 CRC 计算器

Joseph Wu

摘要

在许多 TI 数据转换器器件中，循环冗余校验 (CRC) 在数字通信中用于验证对器件的读取和写入。CRC 用于通常具有固定字节长度的通信，并且校验基于通信序列的计算结果。本应用手册介绍了一个可使用 JavaScript™ 在浏览器上运行的 CRC 计算器示例。本应用手册首先对 CRC 的不同类型及特性进行了基本说明和表示。本应用手册通过示例说明了如何创建基于浏览器的简单 CRC 计算器。该示例进行了扩展，以显示如何轻松修改代码，从而涵盖各种具有有限编码经验的 CRC 类型。

内容

插图清单

商标

1 简介

CRC 是数字通信中常用的错误检测码。TI 数据转换器中通常使用 CRC 来验证器件及微控制器之间的通信。CRC 使用传输数据并且附加初始值（或种子值）。然后在计算中用生成器多项式来生成 CRC 代码。如果发送器和接收器生成相同的代码，则数据正常。如果未生成相同的代码，则数据损坏，传输数据可能有错误。

有诸多类型的 CRC 用于错误检测。这些 CRC 用不同的位长度、初始值和生成器多项式来计算代码。奇偶校验基本上采用 single-bit CRC 计算。8 位、16 位和 32 位的 CRC 代码很常见，显著特征是初始值和发生器多项式值。

CRC 计算通过生成器多项式来对数据传输进行模数 2 除法。该除法以按位 XOR 形式执行。该计算从传输数据字符串开始，方法是附加全零或全一的初始值。执行除法运算，得到的余数为 CRC 代码。如果计算的其余部分与接收到的值不匹配，则表示发生了数据传输错误。

通常会首先发送数据传输，而其他传输（例如 UART）则以最低有效位优先的形式发送。当 CRC 校验作为计算的一部分实现时，必须将位考虑在内。一些 CRC 算法通过反转每个字节的位顺序来反射输入数据或者输出 CRC 代码。

许多器件数据表详细说明了器件中所用的 CRC 函数和计算方法。有关使用 CRC 函数的更多信息，请参阅*使用 Delta-Sigma 数据转换器的数据完整性通信方法*或*循环冗余校验计算：使用 TMS320C54x 的实现*。

2 基于浏览器的 CRC 计算器示例

在图 2-1 中剪辑示例代码并粘贴到文本编辑器中。将代码另存为超文本标记语言 (.html) 文件。

```
<!-- 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>
```

图 2-1. 基于浏览器的 CRC 计算器的 HTML 代码

剪切和粘贴上图中的文本可去除每行的缩进。添加空格以便于查看，但代码运行时没有缩进。

使用 WordPad™ 打开一个空文件并且粘贴文本。选择 *File* 菜单并使用 *Save as*。转到 Save as 类型并选择 *All files* (*.*)。编码为 UTF-8。输入文件名 (crc.html)，然后点击 Save，以将 html 文件放置在您选择的目录中。将此文件保存到计算机上后，点击图标，html 将在默认浏览器上运行。ABC123 的十六进制字符串是默认值。启动时的浏览器窗口如图 2-2 所示。

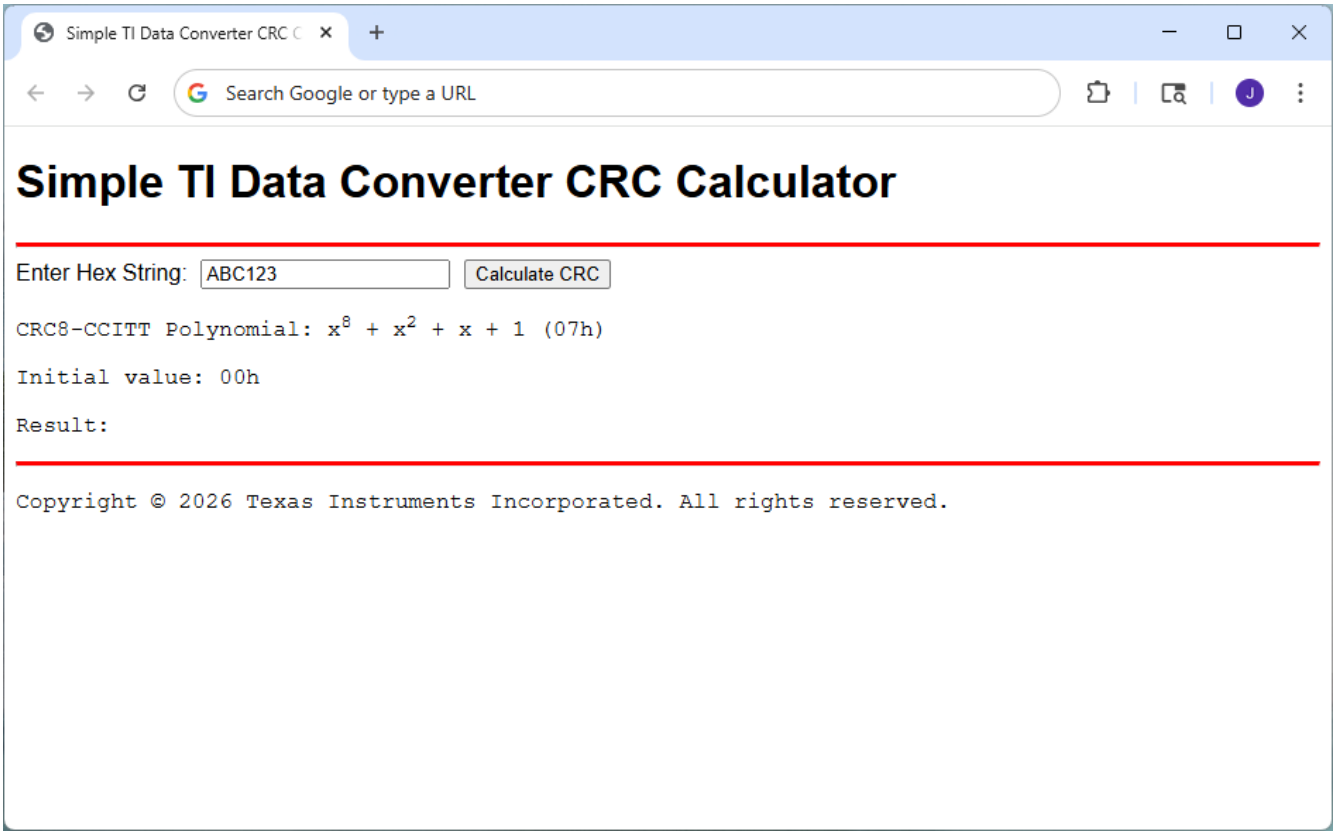

图 2-2. CRC 计算器的浏览器窗口

要运行计算器，请从 ABC123 的默认十六进制字符串开始，然后单击 Calculate CRC 按钮，浏览器将返回 B5 作为图 2-3 中的结果。输入和输出均为十六进制，没有空格。

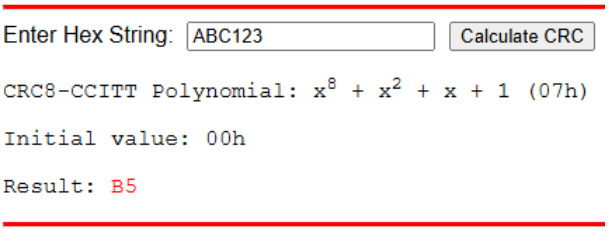

图 2-3. 0xABC123 的 CRC 结果，CRC-8-CCITT，初始值 0x00

替换现有 ABC123 十六进制字符串以计算新值。使用不含非十六进制字符的十六进制字符串。接受大小写两种字符。在以下示例中选择的值是 020026。该字符串是用于禁用 CRC 的 AFE881H1 写入寄存器命令。单击 Calculate CRC，将出现如图 2-4 所示的新结果。

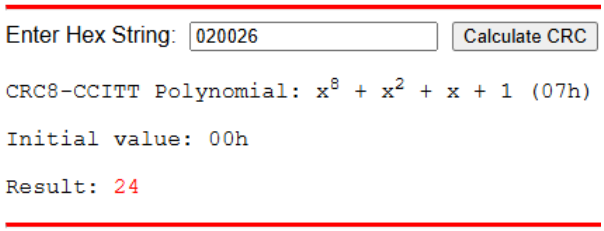

图 2-4. 0x020026 的 CRC 结果，CRC-8-CCITT，初始值 0x00

CRC 值 0x24 根据 0x020026 计算得出。此值附加到命令，以在器件启动后禁用 AFE881H1 中的 CRC。

2.1 CRC-8-CCITT，0x00 初始值

前面基于浏览器的示例采用通过运行 HTML 文件的浏览器而输入的十六进制字节字符串。该脚本将十六进制字符串转换成字节数组。该脚本会使用 0x00 初始值计算 CRC-8-CCITT，并在浏览器中报告结果。代码以字节的形式处理输入，因此输入必须是偶数个十六进制字符。

HTML 主体部分以一个框开头，以输入十六进制字符串，默认输入为 ABC123。正文部分还会创建 Calculate CRC 按钮以生成 CRC。两行文本描述了 CRC，其中显示了生成器多项式和用于计算的初始值。结果显示在底行上。

表 2-1 列出了该示例中 JavaScript 代码的相关函数。

表 2-1. CRC 脚本说明

功能	说明
calculateCRC()	从输入框中提取文本并验证输入只有十六进制字符，调用 hexToBytes() 和 crc8CCITTZeroes()
hexToBytes()	将输入字符转换成字节数组
crc8CCITTZeroes()	计算 CRC-8-CCITT，从 hexToBytes() 创建的数组中初始值为零

该示例使用 0x00 作为初始值来计算 CRC-8-CCITT。CRC-8-CCITTZeroes() 函数从示例中删除，并在图 2-5 中的以下代码中显示。

```
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;
}
```

图 2-5. CRC-8-CCITT，初始值 0x00，计算函数

初始值在函数第一行中定义。此 CRC 使用 crc8CCITTZeroes() 中表示为 0x07 的 $x^8 + x^2 + x + 1$ 的多项式。该多项式以按位 XOR (由“^”指示) 方式设置，其中 CRC 值显示在下面的代码片段中。

表 2-2 中介绍了 CRC-8-CCITT 的详细信息。

表 2-2. CRC-8-CCITT，初始值 0x00

CRC	多项式	初始值	器件	0xABC123 的 CRC
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

显示了多项式和初始值，还列出了使用此特定 CRC 的 TI 器件。最后一列还显示了使用此 CRC 函数用 ABC123 进行 CRC 计算的结果。所生成值可用于检查 CRC 的代码。

2.2 CRC-8-CCITT，0xFF 初始值

可以更改示例 CRC 代码以计算其他的 CRC 类型。对于其他 8 位 CRC 计算，请检查初始值和 CRC 多项式。

在不同的 CRC 版本中，初始值被设置为 0x00 或 0xFF。若要更改代码以检查相同的多项式，但使用 0xFF 作为初始值，请更改函数第二行中的 *let crc* 的值。将该函数重命名为 crc8CCITTOnes() 以显示不同的初始值。图 2-6 显示了新初始值为 0xFF 的代码。

```
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;
}
```

图 2-6. CRC-8-CCITT，初始值 0xFF，计算函数

在 html 的正文部分，确保更改说明以反射新的初始值 0xFF。这个初始值为 0xFF 的新 CRC-8-CCITTOnes() 函数也在 calculateCRC() 函数的末尾更新。

表 2-3 列出了新的 CRC-8-CCITTOnes() 以及使用此 CRC 算法的 TI 器件列表。最后一列提供 ABC123 检查代码的计算结果。

表 2-3. CRC-8-CCITT，初始值 0xFF

CRC	多项式	初始值	器件	0xABC123 的 CRC
CRC-8-CCITT	$x^8 + x^2 + x + 1$ (0x07)	0xFF	ADS1261、ADS127L11、ADS127L14、ADS127L18、ADS127L21、ADS7066、ADS7067、TMAG5173-Q1、TMP114	0x9E

更改 JavaScript 后，运行 CRC-8-CCITT 0xFF 初始值代码。图 2-7 展示了 CRC 的新结果。

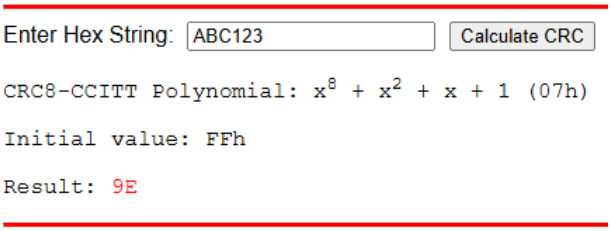

图 2-7. 0xABC123 的 CRC 结果，CRC-8-CCITT，初始值 0xFF

2.3 CRC-8-One-Wire、0xFF 初始值

也可以在脚本内轻松更改多项式。例如，原始 crc8CCITTZeroes() 示例具有图 2-8 中所示的 CRC 多项式的 if 语句。

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

图 2-8. CRC 多项式代码

XOR 中的 0x07 是多项式，代表 $x^8 + x^2 + x + 1$ 的底部三项。x^8 项是触发 XOR 的 MSB。

如果算法更改为 CRC-8-One-Wire，新的 $x^8 + x^5 + x^4 + 1$ 多项式由 0x31 表示，初始值为 0xFF。重命名该函数后，crc8OneWireOnes() 函数在开始时使用相同的 *crc = 0xFF* 启动。如图 2-9 中所示，使用新的 XOR 更改多项式计算。

```
        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;
        }
```

图 2-9. CRC-8-OneWire，初始值 0xFF，计算函数

表 2-4 中列出了 CRC-8-One-Wire 算法的详细信息。

表 2-4. CRC-8-One-Wire，初始值 0xFF

CRC	多项式	初始值	器件	0xABC123 的 CRC
CRC-8-One-Wire	$x^8 + x^5 + x^4 + 1$ (0x31)	0xFF	LMP9007x、LMP90080、LMP9009x、LMP90100、HDC302x、HDC302x-Q1	0xF4

同样，在更改 JavaScript 时，还对描述 CRC 函数的正文进行了更改。然后更改 calculateCRC() 函数以调用新的 CRC 计算。所得到的 CRC-8-One-Wire 计算结果将显示在浏览器窗口中，如图 2-10 所示。

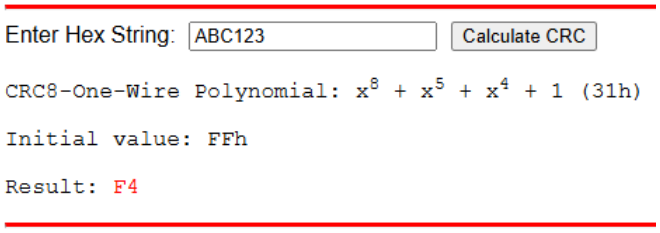

图 2-10. 0xABC123 的 CRC 结果，CRC-8-OneWire，初始值 0xFF

2.4 CRC-16-CCITT，0xFFFF 初始值

在此 CRC-16-CCITT 算法中，初始值设置为 0xFFFF。代码中的更改需要新的计算，如图 2-11 所示。

```
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;
}
```

图 2-11. CRC-16-CCITT，初始值 0xFFFF，计算函数

在这个新代码中，函数被重命名为 crc16CCITTOnes()，标识 CRC 的类型和初始值。在该函数内，初始值设置为 0xFFFF，输入数据左移 12 位以对齐最高有效位。XOR 的 MSB 检查从 0x80 更改成 0x8000。

CRC-16-CCITT 使用多项式 $x^{16} + x^{12} + x^5 + 1$，因此 XOR 从原始代码中的 0x07 更改为 0x1021。在函数结束时，CRC 代码被设置为 16 位长，并在函数返回之前按位 AND。在 html 的正文部分，更正对描述新 CRC 计算的文本所做的任何更改。然后更改 calculateCRC() 以调用 crc16CCITTOnes() 函数。

表 2-5 列出了 CRC-16-CCITT 特性，并列出了几个使用此 CRC 的 TI 器件。

表 2-5. CRC-16-CCITT，初始值 0xFFFF

CRC	多项式	初始值	器件	0xABC123 的 CRC
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (0x1021)	0xFFFF	ADS122C04、ADS122U04（反射型）、ADS131A04、ADS131M04、ADS131B04-Q1、AMC131M03、TMP126、TMP126-Q1	0xB095

新的 CRC-16-CCITT 计算器结果如图 2-12 中所示出现在窗口中。

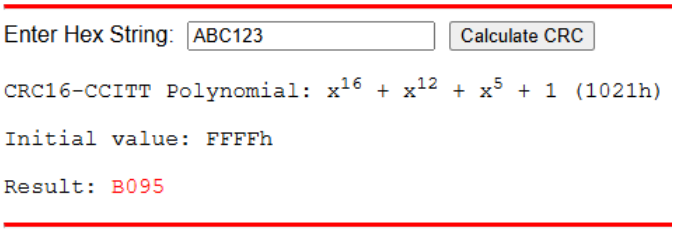

图 2-12. 0xABC123 的 CRC 结果，CRC-16-CCITT，初始值 0xFFFF

在表 2-5 中，ADS122U04 使用反射型 CRC-16-CCITT 计算。以下部分介绍了 CRC 计算内的反射（尽管使用 CRC-8-OneWire 算法）。可使用一个简单的函数添加反射，用于在 CRC 计算中进行输入或输出反射。

2.5 输入和输出数据反射

在一些 CRC 算法中，输入或输出数据均进行反射处理。在这些情况下，字节按相同的顺序输入，但通过反转每个字节的位顺序反射了这些位。这种反射有时用于提高计算效率或降低移位寄存器硬件实现的成本。在此示例中，CRC-8-OneWire 算法被修改为反射输入和输出数据。

从使用 CRC-8-One-Wire 创建的前一个示例开始，创建了一个用于反射数据的 reflect() 函数。图 2-13 中的函数接受定义宽度的值（本例中为一个字节）并反射数据。

```
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;
}
```

图 2-13. 数据反射函数

reflect() 函数的结果以 0h 开始，并按照输入数据的反转顺序逐步设置位。从输入字节开始，输出结果是对字节的位反射。

新的 crc8OneWireZeroesInOutReflect() 函数具有相同的 CRC 多项式 ($x^8 + x^5 + x^4 + 1$)，如上一节所示，但使用不同的初始值 (0x00)。唯一的其他更改是 reflect() 函数的两个加法，用于计算带有输入和输出反射的 CRC。带有输入和输出反射的 CRC 代码如图 2-14 所示。

```
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
}
```

图 2-14. CRC-8-OneWire、初始值 0x00、输入及输出数据反转计算函数

调用每个字节时，输入字节反射在函数的第四行中。在 crc8OneWireZeroesInOutReflect() 的末尾，产生的 CRC 也会进行反射处理。您可能不需要输入反射。可以删除第四行字节反射。如果不需要进行输出反射，则会在末尾返回 CRC，而不是反转的 CRC。

带 00h 初始值以及输入和输出反射的 CRC-8-One-Wire 算法的详细信息在表 2-6 中列出。

表 2-6. CRC-8-One-Wire，初始值 0x00，输入和输出均进行反射处理

CRC	多项式	初始值	器件	0xABC123 的 CRC
CRC-8-One-Wire 输入反射，输出反射	$x^8 + x^5 + x^4 + 1$ (0x31)	0x00	TMP1826、TMP1827	0x86

与前面的示例一样，如果用户正在更改 JavaScript，则还会更改描述 CRC 函数的正文（包括输入和输出反射）。然后更改 calculateCRC() 函数以调用新的 crc8OneWireZeroesInOutReflect()。

图 2-15 中显示了生成的带有输入和输出反射的 CRC-8-One-Wire 计算结果。

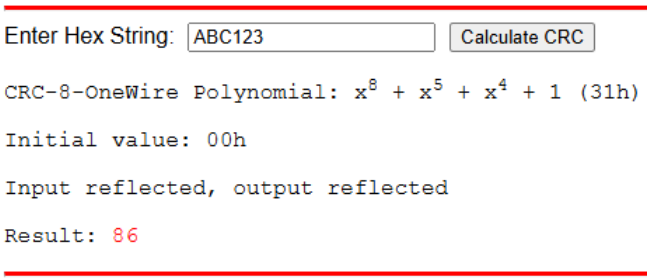

图 2-15. 0xABC123 的 CRC 结果，CRC-8-OneWire，初始值 0x00，反射的输入和输出数据

3 总结

在本应用手册中，示例代码可用于生成简单的 CRC 计算器。通过剪切本应用手册中的代码并粘贴到 html 文件中，可在浏览器中运行此 CRC 代码。原始示例运行 CRC-8-CCITT 计算器，该计算器用于许多 TI 器件。只需稍作修改，即可将代码调整为具有不同初始值、生成多项式和 CRC 长度的 CRC 算法。最后一个示例还展示了如何通过向脚本添加新函数来添加对输入数据或者输出 CRC 计算的反射。进行这些修改后，可用于许多其他 TI 器件中使用的 CRC 计算器。

4 参考资料

- 德州仪器 (TI)，*使用 Δ-Σ 数据转换器检查通信中数据完整性的方法*，应用手册。
- 德州仪器 (TI)，*循环冗余校验计算：使用 TMS320C54x 的实现*，应用手册。
- 德州仪器 (TI)，*ADS7066 的 CRC 实现*，应用手册。
- 德州仪器 (TI)，*ADS7066 CRC 计算器*，计算器。
- 德州仪器 (TI)，*ADS7xx8 CRC 计算器*，计算器。
- 德州仪器 (TI)，*PADC 设计计算器工具*，计算器。
- 德州仪器 (TI)，*CRC 工具、在线模拟工程师计算器*，计算器。
- 德州仪器 (TI)，*精密 ADC Github*，网页。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月