参考資料

Application Note

*シンプルなブラウザベースの **CRC** カリキュレータ*

Joseph Wu

概要

TI の多くのデータ コンバータ デバイスでは、デバイスに対する読み取りおよび書き込みを検証するために、デジタル通信において巡回冗長性検査 (CRC) が使用されます。CRC は通常、固定バイト長を持つ通信で使用され、通信シーケンスから計算された結果に基づいてチェックが行われます。このアプリケーション ノートでは、JavaScript™ を使用してブラウザ上で実行できる CRC カリキュレータの例を紹介します。このノートではまず、CRC の基本的な説明と、その種類および特性について解説します。次に具体例を用いて、ブラウザ上で動作するシンプルな CRC カリキュレータの作成方法を示します。この例は、コーディングの経験が限られているさまざまな CRC タイプに対応するように、コードを簡単に変更する方法を示しています。

目次

図の一覧

商標

1 はじめに

CRC はエラー検出コードで、デジタル通信で一般的に使用されます。TI のデータ コンバータでは多くの場合、デバイスとマイコン間の通信を検証するために CRC が使用されます。CRC は送信データを使用し、初期値 (またはシード値) を付加します。次に、計算で生成多項式を使用して CRC コードを生成します。トランスミッタとレシーバが同じコードを生成した場合、データは良好です。同じコードが生成されない場合、データは不良であり、転送データにエラーがある可能性があります。

エラー検出にはさまざまな種類の CRC が使用されます。これらの CRC は、異なるビット長、初期値、および生成多項式を使用してコードを計算します。パリティ チェックは基本的にシングル ビット CRC 計算です。8 ビット、16 ビット、32 ビットの CRC コードが一般的であり、区別機能は初期値と生成多項式値です。

CRC の計算は、送信データを生成多項式で割るモジュロ 2 除算によって行われます。この除算はビット単位の XOR として実行されます。計算は、送信データ列の末尾にすべて 0 またはすべて 1 の初期値を付加することから開始します。除算を行い、その結果として得られる余りが CRC コードとなります。この計算によって得られた余りが受信側の値と一致しない場合、データ伝送エラーが発生したと判断されます。

データ送信は多くの場合、MSB ファーストで送信されるのが一般的ですが、他 (たとえば UART) は LSB ファーストで送信されます。計算の一部として実装されている場合、CRC チェックを行うときは、このビットを考慮する必要があります。一部の CRC アルゴリズムは、各バイトのビット順序を反転することで、入力データまたは出力 CRC コードを反映します。

多くのデバイス データシートには、デバイスで使用される CRC 関数と計算について詳細に説明されています。CRC 関数の使用方法の詳細については、『デルタ シグマ データ コンバータを用いたデータ インテグリティのための通信方法』または『巡回冗長性検査計算：TMS320C54x を使用した実装』を参照してください。

2 ブラウザベースの CRC カリキュレータの例

図 2-1 のサンプル コードを切り取って、テキスト エディタに貼り付けます。そのコードをハイパーテキスト マークアップ言語 (.html) ファイルとして保存します。

```
<!-- 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>
```

図 2-1. ブラウザベースの CRC カリキュレータ用の HTML コード

上の図のテキストをクリッピングして貼り付けると、各行のインデントが除去される場合があります。スペースを追加すると見やすくなりますが、コードはインデントなしで実行されます。

WordPad™ を使用して空のファイルを開き、テキストを貼り付けます。「File」(ファイル) メニューを選択して、「Save as」(名前を付けて保存) を使用します。「Save as」(名前を付けて保存) の種類に移動し、「All files」(すべてのファイル) (*.*) を選択します。エンコードは UTF-8 です。ファイル名 (crc.html) を入力し、「Save」(保存) をクリックして、選択したディレクトリに HTML ファイルを配置します。このファイルをコンピュータに保存した後、アイコンをクリックすると、HTML がデフォルトのブラウザで実行されます。ABC123 の 16 進文字列がデフォルト値です。ブラウザ ウィンドウは起動時、図 2-2 のように表示されます。

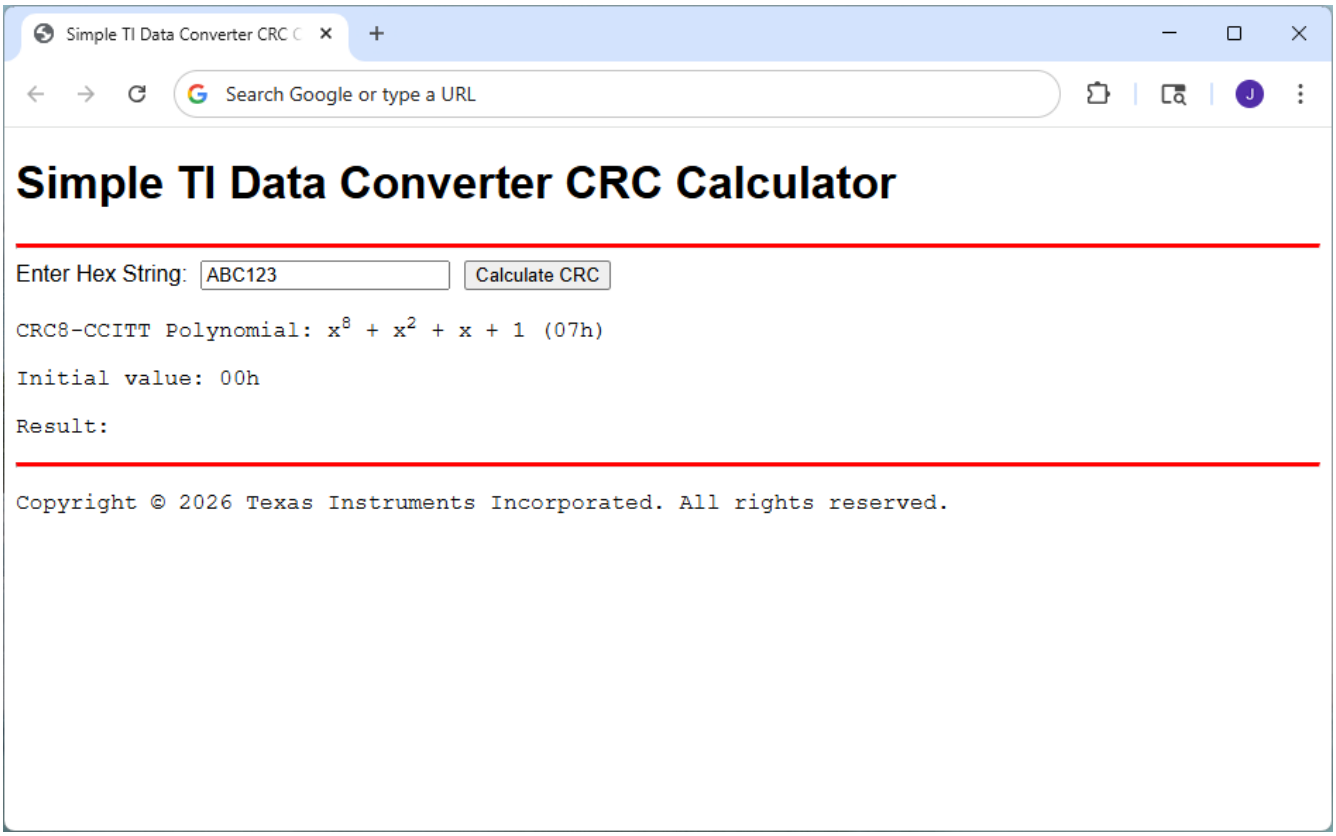

図 2-2. CRC カリキュレータのブラウザ ウィンドウ

カリキュレータを実行するには、デフォルトの 16 進文字列「ABC123」から始めて、「Calculate CRC」(CRC を計算する) ボタンをクリックすると、図 2-3 のように結果としてブラウザが B5 を返します。入力も出力も スペースなしの 16 進数で表示されます。

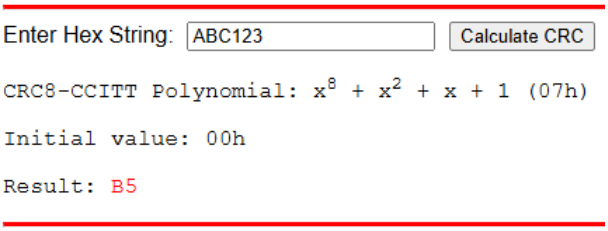

図 2-3. 0xABC123、CRC-8-CCITT、初期値 0x00 の CRC の結果

既存の 16 進文字列「ABC123」を置き換えて、新しい値を計算します。16 進文字列には、16 進数以外の文字を含めないようにしてください。大文字と小文字の両方が使用できます。下の例では、「020026」を使用しています。この文字列は、CRC を無効化するための AFE881H1 書き込みレジスタ コマンドです。「Calculate CRC」(CRC を計算する) をクリックすると、図 2-4 のように新しい結果が表示されます。

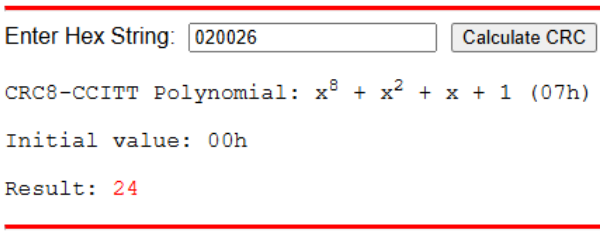

図 2-4. 0x020026、CRC-8-CCITT、初期値 0x00 の CRC の結果

0x020026 から計算された CRC 値は 0x24 です。デバイスの起動後、AFE881H1 の CRC をディスエーブルにするためのコマンドにこの値が追加されます。

2.1 CRC-8-CCITT、0x00 初期値

前のブラウザ ベースの例では、HTML ファイルを実行しているブラウザを介して入力されたバイトの 16 進文字列を使用しています。スクリプトは 16 進文字列をバイトの配列に変換します。このスクリプトは、0x00 の初期値で CRC-8-CCITT の計算を行い、結果をブラウザに報告します。コードは入力をバイト単位で処理するため、入力は偶数個の 16 進文字でなければなりません。

HTML 本体セクションは、デフォルト入力として ABC123 が設定された 16 進文字列を入力するためのボックスから始まります。本文セクションでは、CRC を生成するための「Calculate CRC」(CRC を計算する) ボタンも作成されます。CRC については、生成多項式と、計算に使用される初期値を示す 2 つのテキスト行で説明されます。結果は一番下の行に表示されます。

表 2-1 に、この例で使用されている JavaScript コードの関連する関数を一覧で示します。

表 2-1. CRC スクリプトの説明

関数	説明
calculateCRC()	入力ボックスからテキストを取得し、入力に 16 進文字しか含まれていないことを確認します。hexToBytes() および crc8CCITTZeroes() を呼び出します
hexToBytes()	入力文字をバイトの配列に変換します
crc8CCITTZeroes()	hexToBytes() から作成された配列から、初期値 0 を用いて CRC-8-CCITT を計算します

この例では、初期値として 0x00 を使用して CRC-8-CCITT を計算します。CRC-8-CCITTZeroes() 関数が例から抜粋され、図 2-5 の以下のコードに示されています。

```
        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;
        }
```

図 2-5. CRC-8-CCITT、初期値 0x00、計算関数

初期値は、関数の最初の行で定義されます。この CRC では、crc8CCITTZeroes() で 0x07 で表される $x^8 + x^2 + x + 1$ の多項式を使用します。多項式は、ビット単位の XOR (“^” で示される) を用いて、以下のコード スニペットに示されている CRC 値とともに設定されます。

表 2-2 に、CRC-8-CCITT の詳細を示します。

表 2-2. CRC-8-CCITT、初期値 0x00

CRC	多項式	初期値	デバイス	0xABC123 への 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

多項式と初期値が示されており、この特定の CRC を使用する TI のデバイスも一覧として記載されています。最後の列には、この CRC 関数を使用した ABC123 を用いた CRC 計算の結果も示しています。計算によって得られた値は、CRC コードの検証に使用できます。

2.2 CRC-8-CCITT、0xFF 初期値

CRC コードの例を変更して、他の CRC タイプを計算することもできます。その他の 8 ビット CRC 計算では、初期値と CRC 多項式を確認してください。

異なる CRC バージョンでは、初期値は 0x00 または 0xFF に設定されます。同じ多項式を確認するコードを変更するが、初期値として 0xFF を使用するには、関数の 2 行目の「let crc」の値を変更します。この関数名は crc8CCITTOnes() に変更され、異なる初期値が表示されます。図 2-6 に、新しい初期値 0xFF を持つコードを示します。

```
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;
}
```

図 2-6. CRC-8-CCITT、初期値 0xFF、計算関数

HTML の本文セクションでは、説明が新しい初期値 0xFF を反映するように変更されていることを確認してください。初期値 0xFF を持つこの新しい CRC-8-CCITTOnes() 関数は、calculateCRC() 関数の最後でも更新されています。

表 2-3 には、新しい CRC-8-CCITTOnes() と、この CRC アルゴリズムを使用している TI デバイスの一覧が示されています。最後の列は、コードをチェックするための ABC123 の計算結果を示します。

表 2-3. CRC-8-CCITT、初期値 0xFF

CRC	多項式	初期値	デバイス	0xABC123 への CRC
CRC-8-CCITT	$x^8 + x^2 + x + 1$ (0x07)	0xFF	ADS1261、ADS127L11、ADS127L14、ADS127L18、ADS127L21、ADS7066、ADS7067、TMAG5173-Q1、TMP114	0x9E

JavaScript を変更した後、CRC-8-CCITT 0xFF 初期値コードを実行します。CRC の新しい結果を図 2-7 に示します。

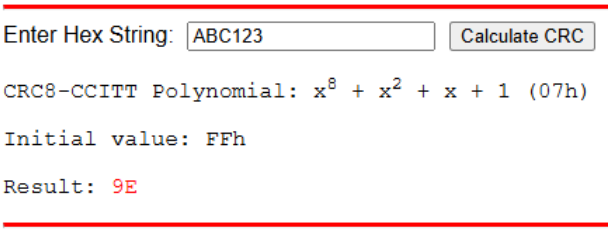

図 2-7. 0xABC123、CRC-8-CCITT、初期値 0xFF の CRC の結果

2.3 CRC-8-One-Wire、0xFF 初期値

多項式もスクリプト内で簡単に変更できます。たとえば、元の crc8CCITTZeroes() の例では、CRC 多項式に対応する IF 文が図 2-8 に示されています。

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

図 2-8. CRC 多項式コード

XOR の 0x07 は多項式であり、$x^8 + x^2 + x + 1$ の下位 3 つの項を表しています。x^8 の項は、XOR をトリガする MSB です。

アルゴリズムを CRC-8-One-Wire に変更する場合、新しい多項式 $x^8 + x^5 + x^4 + 1$ は 0x31 で表され、初期値は 0xFF となります。関数名を変更し、crc8OneWireOnes() 関数を作成するときは、冒頭で crc = 0xFF として開始します。また、多項式計算も新しい XOR に置き換え、図 2-9 に示す通りに変更します。

```
        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;
        }
```

図 2-9. CRC-8-OneWire、初期値 0xFF、計算関数

表 2-4 に、CRC-8-One-Wire アルゴリズムの詳細を示します。

表 2-4. CRC-8-One-Wire、初期値 0xFF

CRC	多項式	初期値	デバイス	0xABC123 への CRC
CRC-8-One-Wire	$x^8 + x^5 + x^4 + 1$ (0x31)	0xFF	LMP9007x, LMP90080, LMP9009x, LMP90100, HDC302x, HDC302x-Q1	0xF4

また、JavaScript を変更するときに、CRC 関数を記述する本文も変更します。次に、calculateCRC() 関数を変更して、新しい CRC 計算を呼び出します。結果として得られる CRC-8-One-Wire 計算が、図 2-10 に示すようにブラウザ ウィンドウに表示されます。

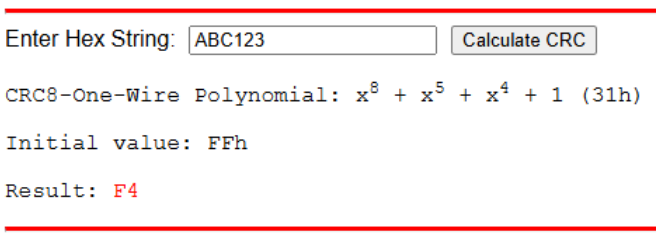

図 2-10. 0xABC123、CRC-8-OneWire、初期値 0xFF の CRC の結果

2.4 CRC-16-CCITT、0xFFFF 初期値

この CRC-16-CCITT アルゴリズムでは、初期値は 0xFFFF に設定されています。コードの変更により新しい計算が必要となり、その結果が図 2-11 に示されています。

```
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;
}
```

図 2-11. CRC-16-CCITT、初期値 0xFFFF、計算関数

この新しいコードでは、関数名が crc16CCITTOnes() に変更され、CRC の種類と初期値が識別できるようになっています。関数内では、初期値が 0xFFFF に設定され、入力データは最上位ビットを揃えるために 12 ビット左にシフトされます。XOR のための MSB のチェックは、0x80 から 0x8000 に変更されています。

CRC-16-CCITT は多項式 $x^{16} + x^{12} + x^5 + 1$ を使用するため、XOR は元のコードを 0x07 から 0x1021 に変更します。関数の最後では、CRC コードを 16 ビットに設定し、ビット単位の AND 演算を行った後、関数から返されます。HTML の本文部分では、新しい CRC 計算を説明するテキストの変更を修正します。次に、calculateCRC() を変更して crc16CCITTOnes() 関数を呼び出します。

表 2-5 には、CRC-16-CCITT の特性が示され、この CRC を使用している複数の TI デバイスも一覧化されています。

表 2-5. CRC-16-CCITT、初期値 0xFFFF

CRC	多項式	初期値	デバイス	0xABC123 への CRC
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (0x1021)	0xFFFF	ADS122C04、ADS122U04 (反映済み)、ADS131A04、ADS131M04、ADS131B04-Q1、AMC131M03、TMP126、TMP126-Q1	0xB095

新しい CRC-16-CCITT カリキュレータの結果が、図 2-12 に示すようにウィンドウに表示されます。

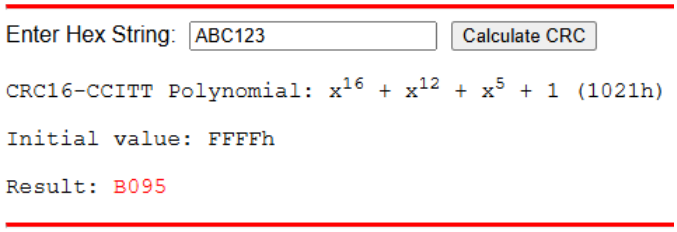

図 2-12. 0xABC123、CRC-16-CCITT、初期値 0xFFFF の CRC の結果

表 2-5 によると、ADS122U04 は CRC-16-CCITT 計算を反映したバージョンを使用します。次のセクションでは、CRC 計算における反映について説明します (ただし CRC-8-OneWire アルゴリズムを使用)。CRC 計算では、入力または出力の反映を簡単な関数で追加することができます。

2.5 入力および出力データの反映

一部の CRC アルゴリズムでは、入力または出力データが反映されます。このような場合、バイトは同じ順序で入力されますが、各バイトのビット順序を反転することでビットが反映されます。この反映は、計算の効率を向上させたり、シフト レジスタのハードウェア実装でコストを削減したりするために使用されることがあります。この例では、CRC-8-OneWire アルゴリズムが入力データと出力データの両方を反映するように変更されています。

前回の CRC-8-One-Wire で作成した例を基に、データを反映させるための reflect() 関数が作成されます。図 2-13 に示されている関数は、指定された幅を持つ値 (この例では 1 バイト) を受け取り、データを反映します。

```
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;
}
```

図 2-13. データ反映関数

reflect() 関数の結果は 0h から開始し、入力データのビットを反転した順序で 1 ビットずつ設定していきます。入力バイトから始めると、出力結果はそのバイトのビット反映となります。

新しい crc8OneWireZeroesInOutReflect() 関数は、前のセクションで示したのと同じ CRC 多項式 ($x^8 + x^5 + x^4 + 1$) を使用していますが、初期値 (0x00) は別ものが使用されています。その他の変更点は、入力および出力の反映を考慮して CRC を計算するために reflect() 関数を 2 回追加したことだけです。図 2-14 に、入力および出力の反映を持つ CRC コードを示します。

```
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
}
```

図 2-14. CRC-8-OneWire、初期値 0x00、入力および出力データを反映させた計算関数

各バイトが呼び出されると、入力バイトは関数の 4 行目に反映されます。crc8OneWireZeroesInOutReflect() の最後に、結果の CRC も反映されます。入力反映が必要ない場合。4 行目のバイト反映は削除できます。出力の反映が必要ない場合、反映された CRC ではなく、CRC が最後に返されます。

初期値 00h の CRC-8-One-Wire アルゴリズムの詳細と入力および出力の反映を表 2-6 に示します。

表 2-6. CRC-8-One-Wire、初期値 0x00、入力および出力を反映

CRC	多項式	初期値	デバイス	0xABC123 への CRC
CRC-8-One-Wire 入力を反映、出力を反映	$x^8 + x^5 + x^4 + 1$ (0x31)	0x00	TMP1826、TMP1827	0x86

前の例のように、JavaScript を変更する場合は、入力と出力の反映を含む CRC 関数を記述する本文も変更します。次に calculateCRC() 関数を変更して、新しい crc8OneWireZeroesInOutReflect() を呼び出します。

入力および出力の反映を適用した CRC-8-One-Wire の計算結果を図 2-15 に示します。

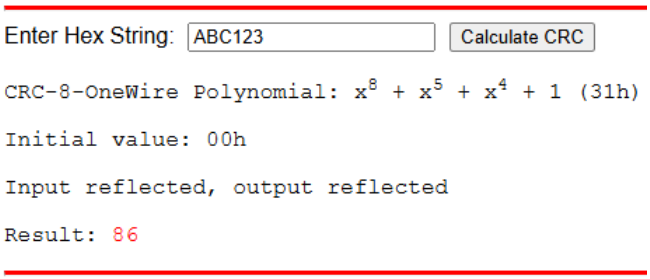

図 2-15. 0xABC123、CRC-8-OneWire、初期値 0x00、入力および出力データ反映の CRC の結果

3 まとめ

このアプリケーション ノートでは、サンプル コードを使用してシンプルな CRC カリキュレータを生成できます。このアプリケーション ノートからコードをクリッピングし、HTML ファイルに貼り付けることで、この CRC コードをブラウザで実行できます。元の例では、CRC-8-CCITT カリキュレータを実行しています。このカリキュレータは、多くの TI デバイスで使用しています。いくつかの小さな変更を加えるだけで、さまざまな初期値、生成多項式、CRC 長の CRC アルゴリズムにコードを適用できます。最後の例では、スクリプトに新しい関数を追加して、入力データまたは出力 CRC 計算にリフレクションを追加する方法も示しています。これらの変更は、他の多くの TI デバイスで使用されている CRC カリキュレータで使用できます。

4 参考資料

- テキサス インスツルメンツ、『デルタ シグマ データ コンバータを用いたデータ インテグリティのための通信方法』、アプリケーション ノート。
- テキサス インスツルメンツ、『巡回冗長性検査の計算：TMS320C54x を使用した実装』、アプリケーション ノート。
- テキサス インスツルメンツ、『ADS7066 における CRC の実装』、アプリケーション ノート。
- テキサス インスツルメンツ、『ADS7066 CRC カリキュレータ』、カリキュレータ。
- テキサス インスツルメンツ、『ADS7xx8 CRC カリキュレータ』、カリキュレータ。
- テキサス インスツルメンツ、『PADC デザイン カリキュレータ ツール』、カリキュレータ。
- テキサス インスツルメンツ、『CRC ツール、オンライン アナログ技術者向けカリキュレータ』、カリキュレータ。
- テキサス インスツルメンツ、『高精度 ADC Github』、Web ページ。

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月