# AVR236：ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘのCRC検査

## 要点

- ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘのCRC生成と検査
- LPM命令を持つ全てのAVR®を支援
- 簡潔なｺｰﾄﾞ量、44語(CRC生成とCRC検査)
- SRAMの必要条件なし
- EEPROM格納のﾁｪｯｸｻﾑ
- 実行時間: 90ms(AT90S8515/8MHz)
- 16ﾋﾞｯﾄ実装、32ﾋﾞｯﾄへの容易な変更
- CRC-16規格支援、CRC-CCITT,CRC-32への容易な変更

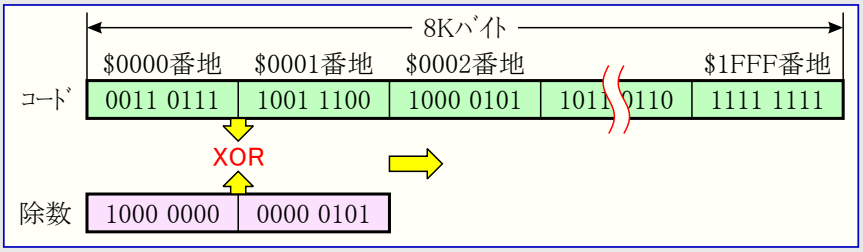
## 序説

この応用記述はCRC(Cyclic Redundancy Check,巡回冗長検査)の原理とAtmel AVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗのﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘでの誤りを検出するためのCRC実装を記述します。

CRCは雑音の多い経路を伝わって伝達されるﾒｯｾｰｼﾞでの誤りを検出する方法に広く用いられています。安全なﾏｲｸﾛ ｺﾝﾄﾛｰﾗ応用のための新しい規格、ﾏｲｸﾛ ｺﾝﾄﾛｰﾗのﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘでの誤り検出の方法としてCRCが導入されました。実際の応用で使うMCU資源をより解放するので、ﾃﾞｰﾀ記憶ﾒﾓﾘに対する低い必要条件と簡潔なｺｰﾄﾞでのCRC計算実装は望ましいことです。

この応用記述で用いるCRCの実装は最小ｺｰﾄﾞ量とﾚｼﾞｽﾀ使用に最適化されています。

図1. 16ﾋﾞｯﾄ除数を使うﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘのCRC検査




## 動作の理屈

ﾁｪｯｸｻﾑは雑音の多い経路を通る通信で始めて使われました。数値(ﾁｪｯｸｻﾑ)は送出ﾒｯｾｰｼﾞの関数として計算されます。受信側はﾁｪｯｸｻﾑを計算するのに同じ関数を用い、その計算された結果を送信側からの受信値と比較します。

この応用記述ではﾁｪｯｸｻﾑがｺｰﾄﾞの関数として構築されて内部EEPROMに格納されます。ﾏｲｸﾛ ｺﾝﾄﾛｰﾗは後でｺｰﾄﾞのﾁｪｯｸｻﾑを計算するのに同じ関数を用い、そして追加されたﾁｪｯｸｻﾑと比較することができます。

例: ｺｰﾄﾞの数値の総和によって計算されたﾁｪｯｸｻﾑ (以下、同様記述は16進数)

- ｺｰﾄﾞ列 : 04 29 06
- ﾁｪｯｸｻﾑ付きｺｰﾄﾞ列 : 04 29 06 39

このﾁｪｯｸｻﾑはｺｰﾄﾞの数値の単純な総和です。

ｺｰﾄﾞ内の第2ﾊﾞｲﾄが$29から$23へ不正に変えられた場合、元のﾁｪｯｸｻﾑが計算されたﾁｪｯｸｻﾑと比較された時に誤りが検出されます。

- 元のﾁｪｯｸｻﾑ付きｺｰﾄﾞ列 : 04 29 06 39
- 誤りを持つｺｰﾄﾞ列 : 04 23 06 39 (33) ← 誤(後計算ﾁｪｯｸｻﾑ不一致)

ｺｰﾄﾞ内の先頭ﾊﾞｲﾄが$04から$10へ、それと第2ﾊﾞｲﾄが$29から$23へ不正に変えられた場合、ﾁｪｯｸｻﾑは誤りを検出しません。

- 元のﾁｪｯｸｻﾑ付きｺｰﾄﾞ列 : 04 29 06 39
- 誤りを持つｺｰﾄﾞ列 : 10 23 06 39 (39) ← 正(後計算ﾁｪｯｸｻﾑ一致、実際は誤)

8-bit AVR®
マイクロ コントローラ

応用記述

本書は一般の方々の便宜のため有志により作成されたもので、Atmel社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

Rev. 1143B-05/02, 1143BJ3-03/21

このチェックサムが持つ問題はそれが単純すぎることです。それはコード内の複数の誤りを検出できないかも知れず、そしてチェックサムそれ自身での誤りを検知できないかもしれません。

この例は加算が誤り検出に充分ではないことを示します。CRC計算はコードに対する計算に加算の代わりに除算を用います。原理は同様ですが、除算を使うことにより、複数ビットの誤りと集中誤りが検出されます。

CRC算法は巨大な2進値としてプログラム メモリを扱い、そしてそれは別の固定2進数で除算されます。この除算の剰余がチェックサムです。マイクロ コントローラは後で同じ除算を実行して剰余を(元々)計算されたチェックサムと比較します。

この除算は(剰余2の)多項式演算を使い、そしてそれは桁溢れを全く使わないことを除き、通常の2進数演算と同じです。多項式演算が持つ数値の加算は単にデータをXORするだけです。

例: 多項式演算に於ける加算

```
  1011 0110
+ 1101 0011
  0110 0101
```

この加算は2つの数のXORと等価です。

多項式演算に関するいくつかの特性を定義しましょう。

M(x) = kビットの数値 (検査すべきコード)
G(x) = (n+1)ビットの数値 (除数または多項式)
R(x) = k>nとなるようなnビットの数値 (剰余またはチェックサム)

$$\frac{M(x)\times 2^n}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \quad ;\text{ここのQ(x)は商}$$

今やQ(x)は次のように記述することができます。

$$Q(x) = \frac{M(x)\times 2^n + R(x)}{G(x)} \quad ;\text{M(x)}\times 2^n\text{はコードの最後にn個の0を追加するのに等しい}$$

$\frac{M(x)\times 2^n}{G(x)}$ が次式に於いて置換されると、

$$\frac{M(x)\times 2^n + R(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} + \frac{R(x)}{G(x)} = Q(x)$$

そしてこれは除数と剰余が同じ数値で、それ自身への加算がXORと同じで0を返すため、Q(x)に等しくなります。

## CRC除算の例

16進数の$6A(2進数の0110 1010)が除数$9(2進数の1001)で除算されます。チェックサムは1001での0110 1010除算操作の剰余です。

最初に元のメッセージ(ビット列)の最後にW個の0を追加してください(ここでのWは除数の幅(ビット数)です)。

```
01101010 0000 ÷ 1001 = 01100      ・・・・・・ 商は無視されます。
0000                              ・・・・・・ 被除数の今回分MSBが0なので除算なし。
 1101
 1001                             ・・・・・・ 被除数の今回分MSBが1なので除算有り。
  1000
  1001                            ・・・・・・ 被除数の今回分MSBが1なので除算有り。
   0011
   0000                           ・・・・・・ 被除数の今回分MSBが0なので除算なし。
    0110
    0000                          ・・・・・・ 被除数の今回分MSBが0なので除算なし。
     1100
     1001                         ・・・・・・ 被除数の今回分MSBが1なので除算有り。
      1010
      1001                        ・・・・・・ 被除数の今回分MSBが1なので除算有り。
       0110
       0000                       ・・・・・・ 被除数の今回分MSBが0なので除算なし。
        1100
        1001                      ・・・・・・ 被除数の今回分MSBが1なので除算有り。
        0101 =5=剰余=チェックサム
```

元のメッセージ(ビット列)の最後にこのチェックサムが追加されます。結果のコードは$6A5です。このコードが検査される時にコードと(追加された)チェックサムが除数で除算されます。誤りが全く起きていなければこの除算の剰余は0で、さもなければ0以外です。

(訳補) 上記計算例で、最初に追加した'0000'がチェクサム'0101'に置き換わることなので、当然結果は0になります。そもそもこの計算は仮に追加した'0000'に相当する剰余を求めるのですから・・・。

CRC検出に関して今日様々な規格が用いられています。除数の種類は8～32と一様でなく、誤り検出能力は使われる除数の幅で変わります。一般的に使われるいくつかのCRCの除数は以下です。

CRC-16 = 1 1000 0000 0000 0101 = $8005

CRC-CCITT = 1 0001 0000 0010 0001 = $1021

CRC-32 = 1 0000 0100 1100 0001 0001 1101 1011 0111 = $04C11DB7

16ﾋﾞｯﾄ除数に於ける実際のﾋﾞｯﾄ数は17で、32ﾋﾞｯﾄ除数に於ける実際のﾋﾞｯﾄ数が33であることに注意してください。最上位ﾋﾞｯﾄ(MSB)は常に1です。

## ソフトウェアの説明

### 主ﾌﾟﾛｸﾞﾗﾑ

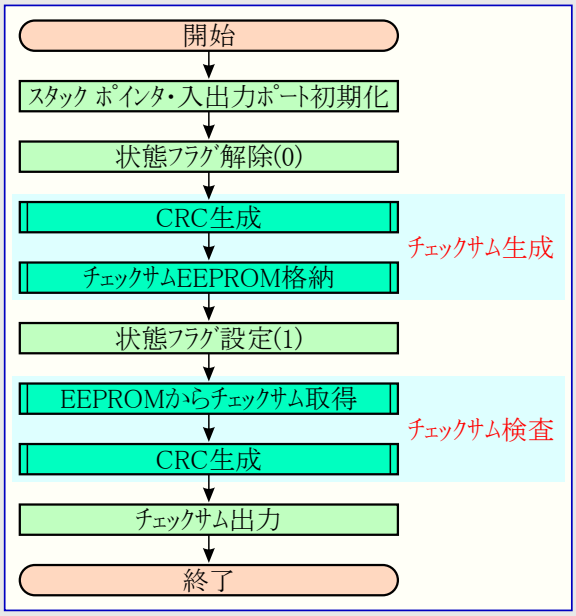
主ﾌﾟﾛｸﾞﾗﾑはCRC生成とCRC検査の両操作を示すために提供されます。生成されたﾁｪｯｸｻﾑは内部EEPROMに格納され、CRC検査が実行される前に読み戻されます。

殆どの応用では書き込み器によってﾁｪｯｸｻﾑが生成され、そしてﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘの最終番地に置かれます。

ｺｰﾄﾞに対する新しいﾁｪｯｸｻﾑを生成するために、主ﾌﾟﾛｸﾞﾗﾑはﾘｾｯﾄ後に状態ﾌﾗｸﾞ ﾚｼﾞｽﾀ=$00でcrc_genｻﾌﾞﾙｰﾁﾝを呼び出します。生成されたﾁｪｯｸｻﾑはEEPROMに格納されます。

CRCﾁｪｯｸｻﾑを検査するために、状態ﾌﾗｸﾞ ﾚｼﾞｽﾀ=$FFまたは$00以外の何れかの値でcrc_genｻﾌﾞﾙｰﾁﾝが呼び出されます。

図2. 主ﾌﾟﾛｸﾞﾗﾑ用流れ図




### CRCﾁｪｯｸｻﾑ生成

この操作はﾋﾞｯﾄ単位でﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ全体を回転する原理に基きます。MSBがｷｬﾘｰ ﾌﾗｸﾞへ移されます。ｷｬﾘｰ ﾌﾗｸﾞが1なら、語が除数とXORされます。ｷｬﾘｰ ﾌﾗｸﾞへ移されたﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘのMSBも除数のMSBとXORされることに注意してください。これらが両方とも1で結果が常に0なので、この除算(XOR)は無視されます。

ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘの最後で、16個の0がｺｰﾄﾞに追加されます。ﾁｪｯｸｻﾑはXOR操作完了の結果値です。

### CRCﾁｪｯｸｻﾑ検査

生成と同じ原理が適用されますが、生成されたﾁｪｯｸｻﾑがｺｰﾄﾞに追加され、0を置き換えます。追加されたﾁｪｯｸｻﾑを含む計算の結果は誤りが起きていなければ0で、さもなければ0以外です。

(既に)ﾁｪｯｸｻﾑがﾌﾟﾛｸﾞﾗﾑ ｺｰﾄﾞに含まれている場合、ﾌﾟﾛｸﾞﾗﾑ ｺｰﾄﾞに於いて計算の検査部分だけが行われることが必要です。

CRC生成とCRC検査の両方に同じﾙｰﾁﾝが使われます。全域状態ﾌﾗｸﾞ ﾚｼﾞｽﾀはCRC生成を実行するための関数呼び出しで$00を格納されます。この状態ﾌﾗｸﾞ ﾚｼﾞｽﾀが関数呼び出しで$00と違うどの値を可能されても、関数はCRCﾁｪｯｸｻﾑ検査を実行します。
流れ図はCRC生成とCRC検査の両方を含むcrc_genﾙｰﾁﾝの流れを示します。

図3.と図4.の流れ図はcrc_genｻﾌﾞﾙｰﾁﾝの操作を記述します。

図3. crc_genサブルーチン

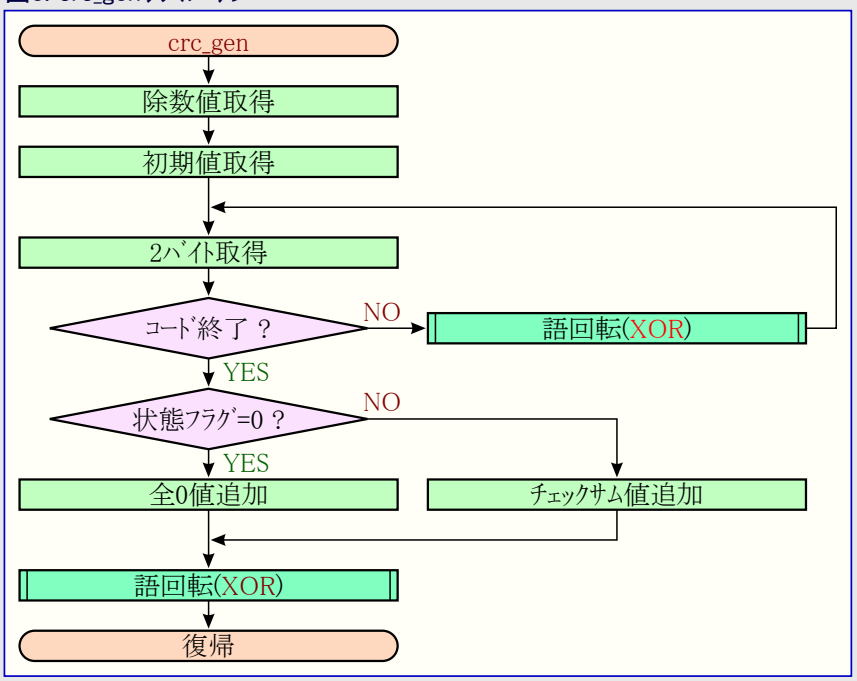
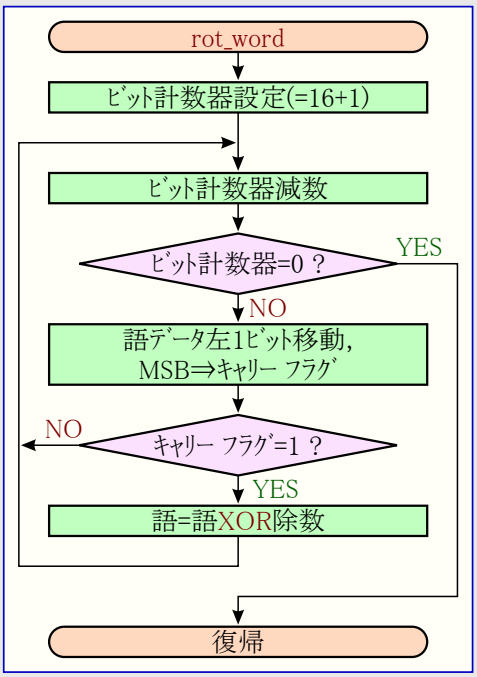



図4. 回転サブルーチン




## 変更

コード゙例はCRC-16計算に16ビットのチェックサムを実行します。このコード゙はコード゙緩衝部の容量を32から64ビットへ増やして除数の容量を16から32ビットへ増やすことにより、32ビットのチェックサムを支援するように簡単に変更できます。

チェックサムが書き込み器によって生成され、そして最終メモリ位置に置かれる場合、チェックサムを検査するためのコード゙だけがプログラムに含められることを必要とします。ルーチンの最後の部分のコード゙を削除することができます。コード゙内の注釈をご覧ください。

いくつかのCRC算法は$00と異なる初期値を持つデータ レジスタが必要です。他の値が使われる場合、最初の2つのLPM命令を置き換えて、そのレジスタ内に初期値を格納することができます。より多くの情報についてはコード゙内の注釈をご覧ください。

CRC算法が反転される場合、これはMSBの代わりにバイトのLSBが最初に移動されることを意味し、ルーチンはLSL(論理的ビット左移動)とROL(ビット左回転)の命令をLSR(論理的ビット右移動)とROR(ビット右回転)の命令で置き換えることによってこれを支援することができます。

より速い速度の既存CRC計算の別の実装の殆どは操作の速度を増すために参照表を用います。このような応用に対するSRAMの必要条件がそれらをもっと複雑なシステムに適応させます。

## 資源

表1. CPUとメモリの使用

| 関数 | コード゙量(語) | 周期数 | 使用レジスタ | 割り込み | 説明 |
|---|---|---|---|---|---|
| main | 36 | - | R2,R3,R16,R22～R25 | - | 初期化と例プログラム |
| crc_gen | 44 | 概ね 700,000 | R0～R3,R17～R22, R30,R31 | - | CRCチェックサムの生成と検査 |
| eewrite | 7 | 13 | R16,R23～R25 | - | EEPROMへCRCチェックサム書き込み |
| eeread | 4 | 8 | R16,R23～R25 | - | EEPROMからCRCチェックサム読み込み |
| 合計 | 91 | - | | - | |

表2. 周辺機能の使用

| 周辺機能 | 説明 |
|---|---|
| EEPROM (2バイト) | CRC値格納 |
| ポートBの8つのI/Oピン | LED用にCRCの下位バイト出力 |

## 参考文献

Fred Halsall "Data Communication, Computer Networks and Open Systems" 1992 Addison-Wesley Publishers

RossN.Williams "The Painless Guide to Error Detection Algorithms" ftp://ftp.rocksoft.com/papers/crc_v3.txt

## 本社

*Atmel Corporation*
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

## 国外営業拠点

*Atmel Asia*
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

*Atmel Europe*
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines
Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

*Atmel Japan*
104-0033 東京都中央区
新川1-24-8
東熱新川ﾋﾞﾙ 9F
ｱﾄﾒﾙ ｼﾞｬﾊﾟﾝ株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

## 製造拠点

*Memory*
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

*Microcontrollers*
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3
France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

*ASIC/ASSP/Smart Cards*
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR
Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

*RF/Automotive*
Theresienstrasse 2
Postfach 3535
74025 Heilbronn
Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

*Biometrics*
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-76-58-47-50
FAX (33) 4-76-58-47-60

**文献請求**
www.atmel.com/literature

© Atmel Corporation 2002.

Atmel製品は、ｳｪﾌﾞｻｲﾄ上にあるAtmelの定義、条件による標準保証で明示された内容以外の保証はありません。本製品は改良のため予告なく変更される場合があります。いかなる場合も、特許や知的技術のﾗｲｾﾝｽを与えるものではありません。Atmel製品は、生命維持装置の重要部品などのような使用を認めておりません。

本書中の®、™はAtmelの登録商標、商標です。
本書中の製品名などは、一般的に商標です。

© *HERO* 2021.

本応用記述はAtmelのAVR236応用記述(doc1143.pdf Rev.1143B-05/02)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。