AN3289

tinyAVR® 1系での時間管理

要点

- 時間と日付の両方が提供される2つの異なる時間管理算法
- 2つの時間管理法の比較とﾍﾞﾝﾁﾏｰｷﾝｸﾞ(比較参照更新)
- 8ﾋﾞｯﾄ基本構造でのより速い除算用の逆乗算法の提示
- Atmel STARTで利用可能な4つのｺｰﾄﾞ例
- 例の機能を試験するためのUSART命令ｲﾝﾀｰﾌｪｰｽ用ｺｰﾄﾞ
- 自動閏年計算と警報機能用ｺｰﾄﾞ
- 計時の基本とﾀｲﾏ/ｶｳﾝﾀの概要

序説

著者: Henrik Nyholm, Microchip Technology Inc.

8ﾋﾞｯﾄ系での時間管理は度々32や64ﾋﾞｯﾄ基本構造ほど簡単ではありません。32や64ﾋﾞｯﾄ系での時間管理は代表的に時代や時刻印のために単一時間単位の経緯を保つ32や64ﾋﾞｯﾄの変数を使うことによって実行されます。これらのｼｽﾃﾑは通常、それらの命令一式の一部として高速乗除算ﾙｰﾁﾝを持ちます。この理由のため、単変数と人が可読な時間との間の変換は命令周期数やﾒﾓﾘ使用に関して厳しくありません。けれども、8ﾋﾞｯﾄ基本構造について、単変数法は常に最良の選択ではありません。他の代替は時、分、秒のように分離された時間単位の経緯を保つ多変数を使うことです。2つの方法の利点と欠点は「1. 時間と日付での時間管理」章で検討されます。加えて、2つの方法の実装が提供されます。

単変数と多変数の両方法の性能が実行時間とﾒﾓﾘ使用に関して提供されます。単変数法の性能を改善するため、定数の逆数や逆数での乗算の解決策が提供されます。この方法は32ﾋﾞｯﾄ変数の除算ﾙｰﾁﾝに対する必要性を無くすことによって実行時間を改善してﾌﾗｯｼｭ ﾒﾓﾘ使用を減らします。多変数法はﾌﾗｯｼｭ ﾒﾓﾘでのかなりの減少を与え、人が可読な形式が度々必要とされる場合に単変数法と比べてより速い実行時間を持ちます。この方法では単一時間単位から人が可読な形式に変換する必要もありません。

この応用記述で検討された時間管理を実装するATtiny817 Xplained Pro用にAtmel STARTで利用可能な4つの例ﾌﾟﾛｼﾞｪｸﾄがあります。単変数と多変数の両方法は時間専用と時間/日付との独立した例で実装されます。永久暦(自動閏年計算)は多変数と単変数の両例ﾌﾟﾛｼﾞｪｸﾄで利用可能です。例の機能を試験するために、万能同期/非同期送受信器(USART:Universal Synchronous Asynchronous Receiver Transmitter)命令ｲﾝﾀｰﾌｪｰｽが含められています。このｲﾝﾀｰﾌｪｰｽを通して、使用者は時間や日付の設定と取得の両方ができます。加えて、多変数例は警報機能を持ち、命令ｲﾝﾀｰﾌｪｰｽを通して現在時間を継続的に表示することができます。例で紹介される概念はRTCとUSARTを持つどのAVRに対しても有効です。例ﾌﾟﾛｼﾞｪｸﾄは「4. Atmel STARTからのｿｰｽ ｺｰﾄﾞ取得」章で一覧にされます。

tinyAVR 1系のﾀｲﾏ/ｶｳﾝﾀに対する短い紹介が「3. 計時の基本」章で与えられます。「2. ﾊｰﾄﾞｳｪｱ要件」章はATtiny817 Xplained Pro上のﾃﾞﾊﾞｲｽに外部32kHzｸﾘｽﾀﾙ発振子を接続する方法を説明します。外部ｸﾘｽﾀﾙはそのままで例を走らせるのに必要で、正確な時間の経緯を保ちます。2.章はｸﾘｽﾀﾙが接続される時にﾏｲｸﾛUSBﾎﾟｰﾄを通してUSART命令ｲﾝﾀｰﾌｪｰｽを使う方法も説明します。代替手段はｷｯﾄ変更なしに例を走らせることを含みますが、時間管理が著しく低い精度です。

目次

1. 時間と日付での時間管理

実時間計数器(RTC)は時間管理に使う最も自然なﾀｲﾏ/ｶｳﾝﾀ(TC)です。これはRTCがそれのｸﾛｯｸ元として32.768kHzｸﾘｽﾀﾙを使うことができ、RTC前置分周器が1Hz刻時を提供する32,768まで入力周波数を分周することができるからです。RTCはｽﾀﾝﾊﾞｲ休止動作で走ることができ、ﾃﾞﾊﾞｲｽに対する起き上がり元として使われます。これは低電力応用に対してそれを素晴らしい選択にします。Atmel STARTで利用可能な例では、RTCが秒毎に割り込みを生成するように構成設定されます。

周期的な1秒割り込み刻時は時間の経緯を保つためにｿﾌﾄｳｪｱ変数を増やすのに使うことができます。これを行う最も効率的な方法はRTCﾚｼﾞｽﾀが溢れる時毎に呼ばれる割り込み処理ﾙｰﾁﾝ(ISR:Interrupt Service Routine)を使うことによります。このｿﾌﾄｳｪｱ変数を人が可読な時間と日付へ変換する時に、暦に関する複雑さが考慮されなければなりません。

日付と暦

時間に加えて日付の経緯を保つ時に、ｸﾞﾚｺﾞﾘｵ歴に関する或る複雑さが考慮されなければなりません。ｸﾞﾚｺﾞﾘｵ歴は太陽を周回する地球に関して365.256日かかると言う事実に対して補償するために閏年を使います。閏年は年数が4で完全に割ることができるけれども、400でも割ることができる時を除き、100で割ることができない時の全ての年で起きます。例えば、1900年は閏年ではなく、一方で2000年は閏年です。各閏年は2月の月の日数を28から29に増やします。

閏秒は地球の緩やかな回転減少に対して調整する不定期に置かれる秒です。閏秒は不定期に置かれ、それらを計算するのに利用可能な式はありません。手動時間調整、または全地球測位ｼｽﾃﾑ(GPS:Global Positioning System)やｲﾝﾀｰﾈｯﾄのような時間供給元から時間を更新することが閏秒に対して自動的に補償する唯一の方法です。

実時間の経緯を保つ時に、閏年と閏秒が考慮に入れられるべきです。けれども、閏秒はこの応用記述の範囲外です。

発振器精度

全てのｸﾛｯｸ元はいくつかの周波数不正確性に左右されます。ｸﾘｽﾀﾙ発振子はそれがﾃﾞﾊﾞｲｽ内の内部RC発振器と比べて周波数出力と安定性の面で全般的により正確なため、一般的に時間管理に使われます。内部RC発振器は低い精度を必要とする低電力応用により有用で、ﾊﾟﾙｽ幅変調(PWM:Pulse Width Modulation)やｼﾘｱﾙ通信のような周辺機能用のｸﾛｯｸ元として適します。

ｸﾘｽﾀﾙ発振子の精度は複数の要素に依存します。最も顕著なのは以下です。

- 以下から成る総負荷容量:
  - PCB寄生容量
  - 追加負荷容量とそれらの製造業者許容誤差
  - ﾃﾞﾊﾞｲｽでの容量
- ｸﾘｽﾀﾙそれ自身の製造業者許容誤差
- 温度のための公称周波数からの変動

相対的に安い32.768kHzｸﾘｽﾀﾙで一般的に25℃で±20 PPM内の設計を得ることが可能ですが、温度での変動はｸﾘｽﾀﾙ毎に異なります。「AN2711 – AVR MCUでの実時間ｸﾛｯｸの校正と補償」応用記述はこれらの変動に対する補償方法を記述します。tinyAVR 1系でのｸﾛｯｸとｸﾛｯｸ ｼｽﾃﾑのより多くの情報については関連ﾃﾞﾊﾞｲｽ用のﾃﾞｰﾀｼｰﾄで「CLKCTRL – ｸﾛｯｸ制御器」章をご覧ください。

1.1. 単変数時間

情報: 時間の経緯を保つのに単変数を使い、時/分/秒への変換のものと日/時/分/秒への変換のものの2つの例ﾌﾟﾛｼﾞｪｸﾄが利用可能です。後者は前章で記述された閏年計算も考慮します。これらの例へのﾘﾝｸは「4. Atmel STARTからのｿｰｽｺｰﾄﾞ取得」章に置かれます。

単変数使用は時間の経緯を保つ最も一般的な方法です。(64ﾋﾞｯﾄ環境でさえ)32ﾋﾞｯﾄで、この方法は最も簡単にし易く、或る程度まで時間管理実装の最も効率的な方法です。8ﾋﾞｯﾄ基本構造では、この方法を使う選択は以下の部分で検討されるいくつかの要素に依存します。

単変数法で、時間の経緯維持は単一の32ﾋﾞｯﾄ変数に時間の1単位(この場合、1秒)を追加するのと同じ位簡単です。応用が与えられた時間に警報を設定する可能性を含む必要がある場合、計数器値を32ﾋﾞｯﾄ比較値と比べることによって容易に達成することができます。けれども、8ﾋﾞｯﾄ ﾃﾞﾊﾞｲｽが一度に8ﾋﾞｯﾄ ﾃﾞｰﾀしか操作できないため、32ﾋﾞｯﾄ値での加算と比較の命令の実行は多数のｸﾛｯｸ周期を必要とします。例え、Cｺｰﾄﾞが簡単とは言え、ｺﾝﾊﾟｲﾙされた機械語はより多くの時間と電力を消費します。

より長い期間に渡って時間の経緯を保つため、単変数計数器を時、分、秒に変換するのに32ﾋﾞｯﾄ数の除算が必要です。8ﾋﾞｯﾄ基本構造では、ｺﾝﾊﾟｲﾙされたｺｰﾄﾞの実行周期数と大きさの両方の話になると、除算は資源に厳しいです。現在の秒数を人が可読な形式への時刻印に変換する時に、変数はいくつかの異なる定数で除算されなければなりません。これらの定数の逆数を計算することにより、ｺｰﾄﾞに於いて除算の代わりに乗算とﾋﾞｯﾄ移動を利用する算法を使うことができます。

定数の逆数による除算

実行時間中に除算を行うことなくAと名付けられたNﾋﾞｯﾄ変数がﾌﾟﾛｸﾞﾗﾑで定数Cによって乗算されるべきと考えてください。A/C=A×1/Cのため、ｺｰﾄﾞをｺﾝﾊﾟｲﾙして代わりに乗算を行う前に、Cの逆数と呼ばれる1/Cを計算することが可能です。問題は逆数を正確に表現するため、浮動または固定小数点数の表現を使わなければならないことです。浮動または固定小数点数での計算が資源に厳しいため、この手段は好ましくありません。実行時間中に整数での乗算とﾋﾞｯﾄ移動だけを使うもっと効率的な方法が以降の順序一覧で提示されます。一覧の各点はC=3600とN=32を使うことによって例示されます。これらの値は単変数時間例で秒の経緯を保つ変数が32ﾋﾞｯﾄで秒を時に変換するために3600によって除算されるので、使われます。表1–1.は時間変換に有用な各種定数に対してこの算法を使った後の結果値を示します。

1. より正確な浮動小数点表現のために1/Cを2NのQ数形式を持つ固定小数点2進数表現に変換します。

   例: 1/3600の浮動小数点値は0.00027777777777777777です。この値を$2^{2N}=2^{64}$で乗算して結果からの小数を無視することにより、5,124,095,576,030,431を得ます。数値を符号付き2の補数の2進数に変換することにより、以下の64ﾋﾞｯﾄ数値を得ます。

   0b0000000000001001000110100010101100111100010011010101111100110111111

2. 最上位ﾋﾞｯﾄ(MSB)が1になるまで2進数を左移動し、移動数をSとします。換言すると、2Nﾋﾞｯﾄ数内で最初の1に先行する0の数を左から右に数えます。

   例: 2進数で最初の1の前に11個の0があり、従って、S=11です。

3. S個左移動した2進数の上位側N+1ﾋﾞｯﾄを取ってこの数に値1を加えます。2進数をNﾋﾞｯﾄに切り詰めることにより、数値の丸め(0捨1入)が達成されます。

   例: 移動した2進数の上位33ﾋﾞｯﾄは0b100100011010001010110011110001001です。
   これへの数値0x1加算は0b100100011010001010110011110001010に帰着します。
   32ﾋﾞｯﾄへの切り詰めは0b10010001101000101011001111000101を生成します。

4. Nﾋﾞｯﾄ2進数をKとして示されるN/4桁16進数に変換します。

   例: 32ﾋﾞｯﾄ2進数の変換は16進数K=0x91A2B3C5を生成します。

5. Cﾌﾟﾛｸﾞﾗﾐﾝｸﾞで、uintN_t R = A/C;の計算はuintN_t R = (((uint2N_t)A * (uint2N_t)K) >> N) >> S;と等価です。ここで除算ﾙｰﾁﾝは必要とされません。

   例: この例ではhours(時)を得るため定数C=3600によってseconds(秒)を除算することを求めます。secondsが32ﾋﾞｯﾄ変数で、N=32のため、S=11、K=0x91A2B3C5であることを見つけ、結果のｺｰﾄﾞは次のようになります。

```
uint32_t hours = (((uint64)seconds * (uint64_t)0x91A2B3C5) >> 32) >>11;
```

この方法が正しく使われることを確認するため、「5. 追補A – 逆数乗算検査」章で検査ﾌﾟﾛｸﾞﾗﾑが提供されます。検査関数は入力ﾊﾟﾗﾒｰﾀとしてC、S、K値を使い、固定小数点乗算の結果が対応する整数除算と同じ結果か調べます。変数Aは可能な全ての値を検査するために正の整数範囲全体に渡って増やされます。表1-1.は単変数日付と時間例での時間変換で使われる有用な定数について対応するKとSの値を一覧にします。これらの値は除算ﾙｰﾁﾝを使うことなく、定数の1つによって32または16ﾋﾞｯﾄの変数の除算を達成するために下の情報枠内のｺｰﾄﾞ上に直接挿入することができます。

表1-1. 除算なしでの時間変換に使われる定数用のSとKの値

説明	C	S	K	Nﾋﾞｯﾄ除算
1分内の秒数	60	5	0x8889	16
1時間内の秒数	3600	11	0x91A2B3C5	32
1日内の時間数	24	4	0xAAAAAAAB	32
1年内の日数	265	8	0xB38D	16

情報: uintN_t R = A/C;はuintN_t R = (((uint2N_t)A * (uint2N_t)K) >> N) >> S;と等価です。

単変数法での閏年

閏年は1年で季節が変動しないことを保証するのに最適な一方で、これらは日付計算を複雑にします。この文書の執筆時点で、4年則に対する最初の例外は2100年です。今日の多くの応用について、100年と400年の例外は関係ないと思うかもしれません。単変数日付と時間例では、変換関数がこの例外を扱わず、代わりにｺﾝﾊﾟｲﾗ警告を与えます。ｺｰﾄﾞが100と400年則を扱う場合、必然的に追加の逆数乗算が必要です。単変数手法での閏年計算追加はいくつかの余分な計算と直ぐに複雑になる論理を付加し、従って、単変数の各変換に余分な実行周期数を追加します。

単変数ｺｰﾄﾞのﾍﾞﾝﾁﾏｰｸ(比較参照更新)

下表は単変数時間と単変数日付と時間の例のｺﾝﾊﾟｲﾙの大きさと実行時間を示します。ｺﾝﾊﾟｲﾙの大きさ測定は(どのUSART構成設定もなしで)必要なAtmel START構成設定と時間管理に最低限必要なｺｰﾄﾞだけをｺﾝﾊﾟｲﾙすることによって生成されました。これは正確に実際の例のように動かないことを意味します。比較のために、表は同じ構成設定を持つﾌﾟﾛｸﾞﾗﾑのｺﾝﾊﾟｲﾙの大きさを示しますが、時間管理関数がなく、その他は空のmain()です。大きさ測定がどう行われるかのより多くの情報については「6. 追補B – ｺｰﾄﾞの大きさ測定の詳細」章をご覧ください。

表1-2. ｺﾝﾊﾟｲﾙの大きさと実行時間

ﾌﾟﾛｸﾞﾗﾑ	GCC -O3最適化		GCC -Os最適化		ｸﾛｯｸ周期数
	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	
空のISRで構成されたRTCと休止	444ﾊﾞｲﾄ	0ﾊﾞｲﾄ	262ﾊﾞｲﾄ	0ﾊﾞｲﾄ	該当なし
単変数時間	1184ﾊﾞｲﾄ	11ﾊﾞｲﾄ	1000ﾊﾞｲﾄ	11ﾊﾞｲﾄ	589
単変数日付と時間	1598ﾊﾞｲﾄ	27ﾊﾞｲﾄ	1372ﾊﾞｲﾄ	27ﾊﾞｲﾄ	1370～1880

表の単変数時間例の周期数測定は逆数時間変換関数呼び出し後に使われた周期数を数えることによって実行されました。比較すると、除算時間変換関数は954周期を使い、よりﾌﾗｯｼｭ ﾒﾓﾘを消費しました。これから、実行時間が最優先の場合、大きな数の通常の除算が避けられるべきことを推測することができます。

単変数日付と時間例で使われる周期数は逆数での時間と日付の両変換関数呼び出しによって測定されました。日付変換で使われた周期数は現在の日付に依存して変わります。これは正しい日付を見つけるためにその月で日にちを減算する間、月を通してｺｰﾄﾞが繰り返すからです。加えて、実行時間に於いて、閏年計算を行うことは現在の年が閏年か否かに応じて変わります。

1.2. 多変数への時間分割

多変数への時間分割は8ﾋﾞｯﾄ基本構造でいくつかの恩恵を持ちます。多変数法のため、単変数法を使うのと比べて、ｺﾝﾊﾟｲﾙされたｺｰﾄﾞの大きさがより小さく、実行時間はより短くなります。これは単変数からより人が可読な多変数表現に変換する必要がないからです。従って、32ﾋﾞｯﾄ変数の乗算や除算の必要がなく、閏年計算も更により簡単にされます。

この方法ではRTC ISRが1秒刻時ごとにﾌﾗｸﾞを設定します。主関数の無限while繰り返しではこのﾌﾗｸﾞが設定されたかを知るための調査が実行されます。ﾌﾗｸﾞが設定されているかどうかに応じて、CPUは休止へ行くか、またはseconds変数を1つ増やす関数を呼びます。その後にseconds変数は60の値と比較されます。secondsが60と等しいと、これは0に設定され、minutes変数が増されます。同じ方法がhoursに対して使われます。

月と年のことになると、処理は少し違います。月と年はどちらも常に同じ長さではありません。閏年に対して追加される例外付きで月用の参照表(LUT:Look Up Table)がこれを解決する容易な方法です。これは月に対応する指標での配列とその月内の日にちに対応する指標を使うことによって達成することができます。

単一秒を8ﾋﾞｯﾄ値に追加するための平均周期数は大きくありません。時間の殆どは1つの8ﾋﾞｯﾄ加算と1つの8ﾋﾞｯﾄ比較が必要とされるだけです。比較が真の時にだけより周期数が必要とされます。seconds変数が60になった時には必ず、平均で1つの余分な8ﾋﾞｯﾄの加算と比較が実行されなければならず、以下同様です。例え最悪の実行時間がかなり大きいとは言え、平均実行時間は非常に小さくなります。警報機能が追加される場合、多くの追加比較が必要です。より多くの情報については「1.3. 警報」項をご覧ください。

閏年

この方法で閏年計算を行うのは簡単です。ｺｰﾄﾞへの閏年追加は多くのﾌﾗｯｼｭ ﾒﾓﾘやRAM使用を追加しません。現在の年が閏年か否かのどちらかかの計算は年で一度必要なだけで、計算それ自体も多くの周期数を消費しません。従って多変数例は閏年計算用の2つのﾏｸﾛを含みます。最初の任意選択は既定で許可され、完全な閏年計算を実行します、他の任意選択は注釈にされ、2100年まで正しいだけです。

多変数ｺｰﾄﾞのﾍﾞﾝﾁﾏｰｸ(比較参照更新)

下表は多変数時間と多変数日付と時間の例のｺﾝﾊﾟｲﾙの大きさと実行時間を示します。ｺﾝﾊﾟｲﾙの大きさ測定は(どのUSART構成設定もなしで)必要なAtmel START構成設定と時間管理に最低限必要なｺｰﾄﾞだけをｺﾝﾊﾟｲﾙすることによって生成されました。これは正確に実際の例のように動かないことを意味します。比較のために、表は同じ構成設定を持つﾌﾟﾛｸﾞﾗﾑのｺﾝﾊﾟｲﾙの大きさを示しますが、時間管理関数がなく、その他は空のmain()です。大きさ測定がどう行われるかのより多くの情報については「6. 追補B – ｺｰﾄﾞの大きさ測定の詳細」章をご覧ください。

表1-3. ｺﾝﾊﾟｲﾙの大きさと実行時間

ﾌﾟﾛｸﾞﾗﾑ	GCC -O3最適化		GCC -Os最適化		平均実行時間
	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	
空のISRで構成されたRTCと休止	444ﾊﾞｲﾄ	0ﾊﾞｲﾄ	262ﾊﾞｲﾄ	0ﾊﾞｲﾄ	該当なし
多変数時間	600ﾊﾞｲﾄ	6ﾊﾞｲﾄ	416ﾊﾞｲﾄ	6ﾊﾞｲﾄ	～8周期/秒
多変数日付と時間	1598ﾊﾞｲﾄ	27ﾊﾞｲﾄ	1372ﾊﾞｲﾄ	27ﾊﾞｲﾄ	～8周期/秒

比較参照された上のｺｰﾄﾞでは必要とされる変換がないため、実行時間は現在時間に秒を追加する平均実行時間です。

1.3. 警報

事象を合図するために時間の経緯を保つ応用で警報機能が必要とされ得ます。

多変数手法でこれを実装する簡単な方法は入れ子にしたif行で秒毎にseconds変数とalarm_seconds変数、minutes変数とalarm_minutes変数、以下同様に比較することです。例え事によると秒毎に複数比較が必要とされても、比較は単変数手法での時の32ﾋﾞｯﾄ比較に代わり2つの8ﾋﾞｯﾄ値間だけです。

秒毎に警報を調べる時に最初にhours、次にminutes、そしてsecondsを比較するか、または逆順かでするかは必ずしも明らかではありません。seconds比較が分毎に1度真なので、日毎に実行される比較の量を減らすため、hours比較で始めるようにしてみたくなるかもしれません。けれども、hours比較が真の時に日毎に1時間の間、全ての秒で常に追加の比較が実行されます。結局、最初に秒を比較することは日毎に2000回少ない比較に帰着することが分かります。従って、平均実行時間がより短くなるだけでなく、1日の時間に依存しないため、より予測可能にもなります。最初の比較は4周期かかります。従って、平均周期数は4近くです。そして最悪実行時間は警報が時刻だけか、または日付と時刻かのどちらに対して設定されるかに応じて4×3または4×6になります。使用者が特定時刻に警報を設定する場合、例え使用者が後で時計を設定することによって時刻を変更しても、特別な考慮は必要ありません。

多変数例では、簡単化のために警報関数が秒毎に呼ばれ、日付ではなく時刻でだけ起動します。hoursの値が増される時に一度alarm_hoursをhoursと比較して増加関数内で一致が見つかった場合にﾌﾗｸﾞを設定することだけを必要とするより高度な解決策が可能です。使用者はその後に分と秒に対して同じことを行うことができますが、劇的に比較数を減らすためにﾌﾗｸﾞが設定されているかどうかに応じて各種増加関数を呼んでください。欠点はﾌﾗｯｼｭ ﾒﾓﾘ使用量増加、ｺｰﾄﾞに於ける増加時間と警報関数の混ぜ合わせです。

「1.1. 単変数時間」項で言及されたように、単一変数に対して警報を作るのは、秒毎にseconds変数を別のalarm_seconds変数と比較することが必要なだけなので、一見したところかなり簡単です。欠点はこの比較が8ビット比較に対して多数の追加周期を必要とすることです。単一32ビット変数法では、変数全体が毎回考慮されなければならないため、走行時間は一定です。換言すると、avr-gccは比較に於いて4バイトのどれかが偽の場合にその比較を停止するような32ビット比較に最適化しません。32ビット等価検査はAVRで13周期を使います。

けれども、alarm_seconds変数の値を設定する方法は明らかではありません。1つの可能性は現在時間から将来での既知の秒の量に警報を設定することで、これは素直(な方法)です。或る時刻に警報時間を設定するには現在のseconds変数の値を秒毎に人が可読な時間に変換するか、または人が可読な警報時間を秒に変換して秒毎に比較を行うかのどちらかでなければなりません。これら2つの代替案からいくつかの複雑さが生じます。最初の代替案について、変換後、単に多変数手法でのように、警報が起動される場合を知るためにsecondsをalarm_secondsと、minutesをalarm_minutesと、以下同様に比較することができます。けれども、表1-2.から、単変数手法での秒毎のseconds変数の人が可読な時間への変換が多変数手法に比べて約70～200倍の周期がかかることを知ります。従って、人が可読な警報時間を秒に変換することは2つの32ビット変数を比較する不利益にも関わらず、最良の手法のように思えます。

この手法で、警報が日毎に同じ時間に起動されるべきなら、応用は警報起動時毎に1日に相当する秒数をalarm_seconds変数に追加しなければなりません。応用は警報を許可または禁止するためのフラグも使うことができます。これは32ビット比較を避けることによって電力消費を減らし、起動制御の層を追加します。一方で、警報が以前に禁止されて1日後に再許可された場合、警報を起動するための時刻を応用が憶えるべき場合の複雑さを追加します。これは禁止された場合、その後にコードが日毎にalarm_secondsをもはや自動的に更新しないからです。最後に、使用者が時刻値を変更する場合、コードはalarm_seconds変数も更新しなければなりません。前述の全ての考慮は完全に実装することが可能な一方で、警報機能は明快さとコード可読性のため単変数例で省略されました。

1.4. 2つの方法の比較

コードの大きさに関して2つの方法を比較すると、多変数法は最小のフラッシュ使用量を持ちます。それにも関わらずコードの別の部分が32ビット乗算を必要とする場合、コンパイラが乗算サブルーチンを共用することができるため、定数の逆数の乗算を持つ単変数法が更に多くのフラッシュ メモリを消費しないかもしれないことに注意してください。

単変数法は多変数法よりも多くの周期を使います。従って、(それが32ビットのため)時間単位変数の増加時と、特にそれを人が可読な時間への変更時の両方が平均でより遅くなり、これらは多変数手法で必要ではありません。多変数法は実行時間が一定ではありませんが、平均は8周期/秒を僅かに超えます。8ビット基本構造で、取得と格納の命令を考慮すると、32ビット変数の増加は8ビット変数を増加するのに比べて最低2倍の周期の量に帰着します。単変数法の主な利点は時間増加の実行時間が固定されることで、一方で多変数手法はかなりより大きな最悪実行時間(WCET:Worst-Case Execution Time)を持ちます。例え多変数手法のWCETが日や年を通して稀にしか起きないとしても、WCETに基づく厳しい期限を持つ実時間応用の計画で問題を起こし得ます。単変数例での人が可読な時間への変換が固定された実行時間を持たないことに注意してください。

一般的に、単変数法は(特に逆数での乗算使用時に)より影響を及ぼされ、(暦と閏年が考慮される時に)実装をより複雑にし、一方で多変数手法は簡単です。これは使用者が時間と日付を秒に変換するための余分な関数を含められなければならない、人が可読な形式で時刻と日付を設定することを望む場合、特に真実です。警報が追加される場合、「1.3. 警報」項で検討されるように更なる複雑さを生じます。単変数法に対する最良の使用事例は応用が(始動や外部割り込みなどのような)事象から時間単位の経緯を保つことだけが必要な時です。この場合では秒から人が可読な時間への変換とその逆も不要で、警報機能は時間管理変数を別の警報変数と比較するのと同じ位簡単です。この時間単位が小さくて時間管理間隔が短い場合、「3. 計時の基本」章で記述されるように正しい周波数を持つデバイスでどれかのタイマ/カウンタの計数(CNT)レジスタと割り込みを使うことでより効率的にできます。加えて、単変数法は実時間応用が上で検討したように考慮される時により有用かもしれません。

結論として、多変数手法は実装が最も容易で、一般的に8ビット基本構造により適切です。例え人が可読な形式から秒への変換が必要でも、平均でより速く、除算の使用を必要としません。主な欠点は時刻を増す実行時間が固定でなく、従って割り込みルーチンで実行されるべきでないことです。主while繰り返しが大きくて長い実行時間を持つ場合、現在の時間は秒毎で瞬時に増されません。実行に対して主繰り返しが秒よりもっと使うかもしれない場合、結果としてフラグが2度設定され、時間増加関数が1度だけ呼ばれるために秒が取り損なわれないことを保証するのにいくつかの追加論理が必要とされます。警報機能は簡単ですが、これもまた固定された実行時間を持ちません。

1.5. AVR Libcとtime.hの使用

AVR LibcはAVRマイクロ コントローラとGCCに適合する無料のCライブラリです。これはMicrochip AVR 8ビットRISCマイクロ コントローラ用に独自化された標準Cライブラリの部分集合を含み、標準time.hライブラリを含みます。標準Cライブラリと同様に、時間管理はUNIXやY2Kの起点時間から時間単位の経緯を保つ単一32ビット変数で実装されます。この変数は(資料で秒に指定される)時間単位毎にsystem_tick()関数を呼ぶことによって増され、現在時間はその後に人が可読な時間を含む各種形式に変換することができます。時間と日付に対して標準の取得と設定の関数に加えて、夏時間、時間帯、それと特に与えられた地理的位置に基づく太陽と月の事象に対する機能を持ちます。このライブラリはこの応用記述で検討した例より明らかにもっと高度で、時間の経緯を保つ容易で迅速な方法を必要とする応用に対する良い候補です。けれども、実装は「1.4. 2つの方法の比較」項で検討された単変数手法に対するのと同じ利点と欠点を持ちます。加えて、これは逆数による乗算の代わりに標準除算ルーチンを使い、これは応用内の何処か他の場所で定数による他の32ビット変数の除算が必要とされない限り、プログラム メモリ使用を増して実行をより遅くし得ます。メモリと電力の効率が最優先なら、この文書で検討された例に基づく時間管理関数を書くことが有益になり得ます。

2. ハードウェア要件

「発振器精度」項で記述されるように、正確な時間管理は内部RCタイマ/カウンタの代わりに外部クリスタル発振子が必要とされます。ATtiny817 Xplained Proキットは基板上にATtiny817デバイスでの時間管理例にクロック元として使われる32.768kHzクリスタル発振子を持ちます。このクリスタルはデバイスのクリスタル用ピンがUSART単位部と共用されるため、既定でデバイスに接続されないことに注意してください。従って、(可能な限り正しくするために発振器が校正を必要とすることも除いて)正確な時間管理で例をそのまま使うにはいくつかのキット変更が必要です。これを行う方法とマイクロUSBを通して未だUSARTを使うことを可能にする方法の記述が下の部分に続きます。時間管理の精度が重要でない試験目的用にキット変更なしで例を走らせるには、「2.3. キット変更なしでの例走行」項で提供される指示に従ってください。

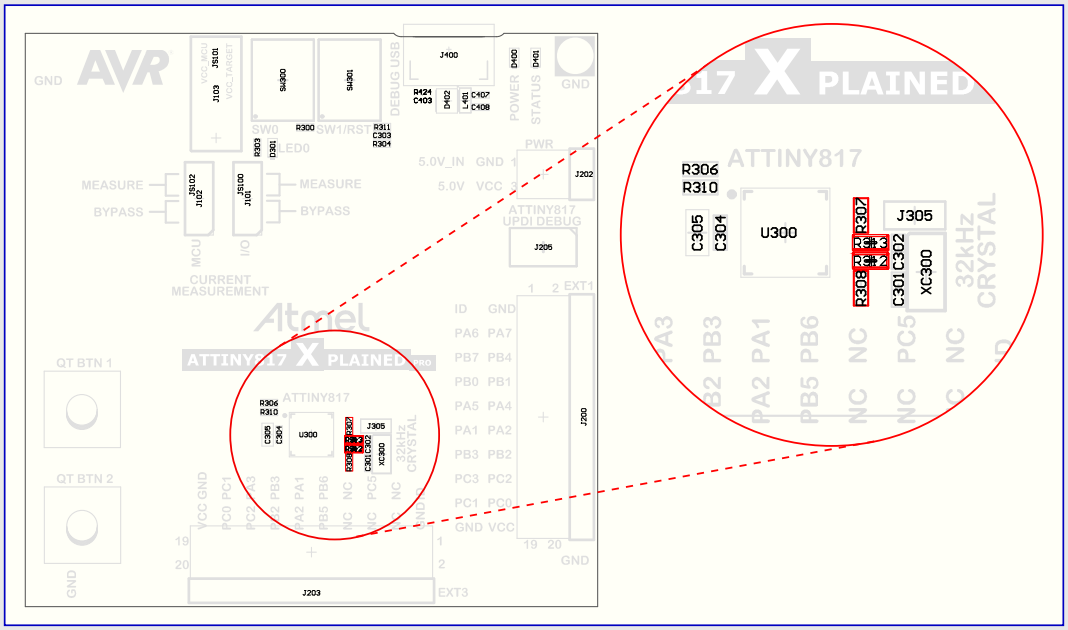
2.1. 32kHzクリスタルの接続

ATtiny817 Xplained Pro基板はキット上に実装された32.768kHzクリスタルを持ちます。既定で、このクリスタルはデバイス上でTOSCピンとして接続されず、このピンは拡張ヘッダへのUSART通信と組み込みデバッガ(EDBG:Embedded Debugger)仮想COMポートに使われます。クリスタルを使うには、デバイス上のピンが他の全てから切断され、クリスタル発振子回路に接続されなければなりません。

クリスタルを接続するには、R307とR308の0Ω抵抗を取り外し、それらをR312とR313用の取り付け位置に配置しなければなりません。部品の位置を見つけるには下の組立図をご覧ください。

情報: クリスタルでATtiny817を動かすには半田ごてを用いてキット上で物理的な変更が必要です。

図2-1. 組立図、表面



2.2. マイクロUSBを通すUSARTでの外部クリスタル

例応用では、クリスタル発振子が使われ、従ってUSARTとUSB仮想COMポート間の接続は切断されます。時間や日付の設定と現在の時間や日付のような現在情報の受け取りのために指令を送るのに命令インターフェース用にUSARTも使います。接続されたクリスタル発振子と共に機能するこれのためにここで対策が提示されます。

ATtiny817の既定USARTピンは受信(RX)データ用のPB3と送信(TX)データ用のPB2です。ATtiny817 Xplained Pro回路図で示されるように、ATtiny817デバイスのPB2とPB3は回路基板上のPB2とPB3(EXT1)拡張ヘッダに接続されます。両ピンと拡張ヘッダはEDBGの仮想COMポートに直接接続されます。この接続のため、マイクロUSBケーブルを通してUSARTデータを送ることが可能です。0Ω抵抗が取り去られると、PB2とPB3のピンとEDBG CDC間の接続が切断されますが、拡張ヘッダは未だEDBG CDCに接続されています。

ATtiny817はRX用のPA2とTX用のPA1のUSART通信用代替ピンを持ちます。これらは同様に基板上でPA1とPA2(EXT3)拡張ヘッダのそれら自身に接続されます。問題はこれらがEDBGの仮想COMポートに接続されていないことです。けれども、PB2とPB3の拡張ヘッダはEDBGの仮想COMポートに接続されています。これは2つのジャンパ線で代替USARTピンを既定USARTピンの拡張ヘッダに接続することが可能なことを意味します。

換言すると、そのまま例を走らせるには、外部32kHzクリスタル発振子を許可するために2つの0Ω抵抗が移動されなければなりません。時間管理機能を確認するためにマイクロUSBを通してUSARTインターフェースで指令を送ってデータを受け取るため、以下の拡張ヘッダピンを、例えば、2つの雌ジャンパ線で、接続してください。

- PA1(EXT3)をPB2(EXT1)
- PA2(EXT3)をPB3(EXT1)

代わりに、その他の方法によって直接PA2とPA3の拡張ヘッダを通してRXとTXのデータを送受信することが可能です。

2.3. キット変更なしでの例走行

どんなキット変更も行うことなしに例を走らせることが可能ですが、コード自体で小さな変更が必要です。外部32kHzクリスタル発振子を使う代わりに代替として内部RCタイマ/カウンタが使われます。時間共に、このクロックは時間管理が不正確であるように実時間からかなり外れます。4つの例がAtmel STARTを使って準備されるため、例をダウンロードする前にAtmel STARTで直接必要な変更を行うことが可能です。別の任意選択は例の1つの主ファイルで関数呼び出しを注釈解除することです。後者の任意選択はその関数がAtmel STARTによって作成された構成設定関数後に呼ばれるため次善策です。換言すると、標準構成設定が最初に実行され、その後に特定関数が或るレジスタを上書きすることによっていくつかの構成設定をやり直します。けれども、これは確かに実装が最速です。

任意選択A – Atmel STARTを通す例変更

1. Atmel STARTで4つの例の1つを選び、OPEN SELECTED EXAMPLE(選んだ例を開く)をクリックしてください。
2. USART_0をクリックしてCOMPONENT SIGNALS部を見つけてください。RXD:とTXD:傍らの引き落としメニューでPB3とPB2を選んでください。
3. RTC_0をクリックしてCLOCK CONFIGURATION部を見つけてください。RTC Clock Source Selection:(RTCクロック元選択:)傍らの引き落としメニューで32kHz Internal Ultra Low Power Oscillator (OSCULP32K)(32kHz内部超低電力発振器(OSCULP32K))を選んでください。
4. CLKCTRLをクリックして32.786kHz CRYSTAL OSCILLATOR CONFIGURATION部を見つけてください。ENABLE: 32.768kHz Crystal Oscillator Enable:(32.768kHzクリスタル発振器許可:)とRUNSTDBY: Run Standby:(スタンバイで走行:)の両方で傍らの枠のチェックを外してください。
5. CLKCTRLの内側で32kHz INTERNAL ULP OSCILLATOR CONFIGURATION部を見つけてください。RUNSTDBY: Run Standby:(スタンバイで走行:)傍らの枠のチェックを確実にしてください。
6. EXPORT PROJECT⇒DOWNLOAD PACKをクリックして.atzipファイルを保存してください。
7. File(ファイル)⇒Import(インポート)⇒Atmel Start Project(Atmel STARTプロジェクト)をクリックすることによって.atzipファイルをAtmel Studio 7にインポートしてください。

任意選択B – 例コードでの関数呼び出し注釈解除

1. 4つの例のどれかを開いてください。
2. mian.cファイルを開いてください。
3. alternate_START_setup();に対する関数呼び出しを注釈解除してください。

3. 計時の基本

いくつかの周期的な処理だけを必要とする応用には複数の任意選択があります。CPUで遅延関数を使うことが可能ですが、常により良い手段ではありません。これを行うことはそれが行われるまで遅延を実行するためにCPUを束縛します。加えて、CPUが遅延を実行している間に割り込みが起こる場合、ISRを実行する周期が遅延関数によって考慮されないため、総遅延時間が増えます。それに割り込みを生成させることによって遅延と計時にタイマ/カウンタ(TC)を使うことは往々にしてより良いことです。その方法ではCPUがプログラムの他の部分を実行することや、または節電のため休止へ行くことさえ自由で、加えて同時に計時の精度がしばしば改善されます。

タイマ/カウンタA型(TCA)、タイマ/カウンタB型(TCB)、タイマ/カウンタD型(TCD)、実時間計数器(RTC)は計時と遅延に全て使うことができます。これらの周辺機能に対して考慮する主な点のいくつかはTCに対するクロック選択、クロックを分周する全能力、ビット数での計数器の大きさです。

計時器の最大期間はクロック周波数、前置分周器設定、計時器の大きさの結果です。クロックの周波数増加時に時間分解能は増しますが、(計数)期間の最大長は減ります。期間の長さは計数器の大きさと共に増します。けれども、TCの最大期間に帰着するその設定の使用が或る応用に対して充分でないかもしれません。このような場合、良い任意選択はソフトウェア計数器変数を使うことです。TCをソフトウェア計数器と組み合わせる時に、TCからの周期的割り込みはISRでソフトウェア変数を増やすのに使うことができます。その後にソフトウェア変数は他の或る活動が取られるべき時の経緯を保つのに使われる変数や定数と比較することができます。この方法では計時の長さを与えられたどれかのタイマ/カウンタの最大期間を超えて増やすことができます。

3.1. RTC

RTCは32.768kHzクリスタル発振器、内部32kHz RC発振器、または別の外部クロック元から走行することができます。RTCの前置分周器は入力周波数を最大32768で分周することができます。32.768kHzの入力が与えられると、結果は1Hz刻時になります。計数器の大きさは16ビットです。低周波数クロックと大きな分周器は容易に長い計時時間を達成させます。

周期的割り込み計時器(PIT:Periodic Interrupt Timer)をRTCの前置分周器から動くように構成設定することも可能です。PIT使用はRTC機能の残りを変更せず、PITは独立した割り込みベクタを持ちます。PITは可能な周期値の全てが4、8、16、～32,768までのクロック周回から成る2のべき乗である周期的な割り込みを与えます。この特徴は作業を行うのにより高度なタイマ/カウンタを使うことなく、或る周期的な処理を必要とする応用に使うことができます。また、最低の電力消費を持つパワーダウン動作(休止動作)で動くことができ、割り込みでデバイスを起こすことができる唯一のタイマ/カウンタです。

3.2. TCA

TCAは周辺機能ｸﾛｯｸ(CLK_PER)を使い、16ﾋﾞｯﾄの計数器の大きさで最大1024の前置分周器を持ちます。3.3MHzの既定ｸﾛｯｸ設定で、最大計時期間は2^{16}/(3.3MHz/1024)=23秒です。

CLK_PERのｸﾛｯｸ周期計数に加え、TCA計数事象を持つことが可能です。この特徴で、使用者は事象入力ﾋﾟﾝの1つにｸﾛｯｸ信号を提供する能力がある外部回路からTCAを動かすことができます。他のﾀｲﾏ/ｶｳﾝﾀと周辺機能からの内部事象を使うこともできます。

TCAと組み合わせた外部事象の例はGPS単位部からの1秒ﾊﾟﾙｽを使うことです。この刻時を計数することによって非常に正確な計時を達成することができます。RTCやTCBの溢れ事象のような内部事象を計数するようにTCAを設定することも可能です。

3.3. TCB

TCBは周辺機能ｸﾛｯｸ(CLK_PER)を使い、16ﾋﾞｯﾄの計数器の大きさで最大2の前置分周器を持ちます。3.3MHzの既定ｸﾛｯｸ設定で、最大計時期間は2^{16}/(3.3MHz/2)=39.7msです。

TCBは一般的に事象を実行して信号を捕獲するために作られています。TCAと異なり、TCBは事象を数えることができません。

3.4. TCD

TCDは周辺機能ｸﾛｯｸ(CLK_PER)、外部ｸﾛｯｸから、または20MHz発振器から直接動くことができます。TCDは12ﾋﾞｯﾄの計数器の大きさで最大32の前置分周器を持ちます。TCDが3.3MHzの既定ｸﾛｯｸ設定でCLK_PERから走行する場合、最大計時期間は2^{12}/(3.3MHz/32)=636msです。

TCDは電動機制御とLED駆動のような電力応用のために設計されています。けれども、この計時器は基本的な計時応用に使うこともできます。TCBと同様に、TCDは事象を数えることができません。

4. Atmel STARTからのｿｰｽ ｺｰﾄﾞ取得

ｺｰﾄﾞ例は画像使用者ｲﾝﾀｰﾌｪｰｽ(GUI)を通して応用ｺｰﾄﾞの形態設定を許すｳｪﾌﾞに基づくAtmel STARTを通して利用可能です。ｺｰﾄﾞは下の直接ｺｰﾄﾞ例ﾘﾝｸまたはAtmel START先頭頁のBROWSE EXAMPLES(例検索)釦経由Atmel Studio 7.0とIAR Embedded Workbench®の両方に対してﾀﾞｳﾝﾛｰﾄﾞすることができます。

Atmel STARTｳｪﾌﾞ ﾍﾟｰｼﾞ : Atmel START

ｺｰﾄﾞ例

- 単変数時間例(Single variable time example):
  - http://start.atmel.com/#examples/single/variable/time
- 単変数時間と日付の例(Single variable time and date example):
  - http://start.atmel.com/#examples/single/variable/date/time
- 多変数時間の例(Multivariable variable time example):
  - http://start.atmel.com/#examples/multivariable/time
- 多変数時間と日付の例(Multivariable variable time and date example):
  - http://start.atmel.com/#example/multivariable/date/time
- 多変数時間と日付、警報の例(Multivariable variable time, date and alarm example):
  - http://start.atmel.com/#examples/multivariable/alarm

例ﾌﾟﾛｼﾞｪｸﾄについての詳細と情報に関してはAtmel STARTでUser guide(使用者の手引き)をｸﾘｯｸしてください。User guide釦はAtmel STARTﾌﾟﾛｼﾞｪｸﾄ形態設定部内の一覧画面でﾌﾟﾛｼﾞｪｸﾄ名をｸﾘｯｸすることにより、例閲覧部で見つけることができます。

Atmel Studio

DOWNLOAD SELECTED EXAMPLE(選んだ例をﾀﾞｳﾝﾛｰﾄﾞ)をｸﾘｯｸすることにより、Atmel STARTで例閲覧部からAtmel Studio用.atzipﾌｧｲﾙとしてｺｰﾄﾞをﾀﾞｳﾝﾛｰﾄﾞしてください。Atmel START内からﾌｧｲﾙをﾀﾞｳﾝﾛｰﾄﾞするには、EXPORT PROJECT(ﾌﾟﾛｼﾞｪｸﾄをｴｸｽﾎﾟｰﾄ)に続いてDOWNLOAD PACK(一括ﾀﾞｳﾝﾛｰﾄﾞ)をｸﾘｯｸしてください。

ﾀﾞｳﾝﾛｰﾄﾞした.atzipﾌｧｲﾙをﾀﾞﾌﾞﾙ ｸﾘｯｸしてください。ﾌﾟﾛｼﾞｪｸﾄがAtmel Studio 7.0に導入されます。

IAR Embedded Workbench

IAR Embedded Workbenchでﾌﾟﾛｼﾞｪｸﾄをｲﾝﾎﾟｰﾄする方法の情報についてはAtmel START使用者の手引きを開き、Using Atmel Start Output in External Tools(外部ﾂｰﾙでAtmel START出力を使用)とIAR Embedded Workbenchを選んでください。Atmel START使用者の手引きへのﾘﾝｸは共に頁の右上隅に置かれたAtmel START先頭頁からHelp(手助け)またはﾌﾟﾛｼﾞｪｸﾄ構成設定部内のHelp And Support(手助けと支援)をｸﾘｯｸすることによって見つけることができます。

5. 追補A – 逆数乗算検査

以下のﾌﾟﾛｸﾞﾗﾑは使われる逆数乗算値の正当性を調べるのに使うことができます。「1.1. 単変数時間」項で記述される方法が表1-1.の表で示されるのと違う定数に対して結果を正しく生じることを確認するのに有用で有り得ます。これはAVRではなく、ｺﾝﾋﾟｭｰﾀで実行されることが意味されます。整数の全範囲(32ﾋﾞｯﾄに対して0～2^{32}、16ﾋﾞｯﾄに対して0～2^{16})に対して逆数乗算を対応する除算と比較する2つの関数を含みます。

```
#include <stdio.h>
#include <stdint.h>

/* 32ﾋﾞｯﾄ除算検査 */
void verify_reciprocal_mult_32(int C, int S, uint32_t K)
{
    printf("Reciprocal multiplication verification started for C = %i.¥n", C);
    for (uint32_t A = 0; A < 0xFFFFFFFF; A++)
    {
        if ((((A * (uint64_t)K) >> 32) >> S) != (uint32_t)(A/(C)))
        {
            printf("Division failed with A = %u.¥n", A);
            return;
        }
    }
    printf("Test passed!¥n");
}

/* 16ﾋﾞｯﾄ除算検査 */
void verify_reciprocal_mult_16(int C, int S, uint16_t K)
{
    printf("Reciprocal multiplication verification started for C = %i.¥n", C);
    for (uint16_t A = 0; A < 0xFFFF; A++)
    {
        if ((((A * (uint32_t)K) >> 16) >> S) != (uint16_t)(A/(C)))
        {
            printf("Division failed with A = %u.¥n", A);
            return;
        }
    }
    printf("Test passed!¥n");
}

int main() {
    verify_reciprocal_mult_32(3600, 11, 0x91A2B3C5);     // 1時間内の秒数
    verify_reciprocal_mult_32(24, 4, 0xAAAAAAAB);        // 1日内の時間数
    verify_reciprocal_mult_16(60, 5, 0x8889);            // 1分内の秒数
    verify_reciprocal_mult_16(365, 8, 0xB38D);           // 1年内の日数
    return 0;
}
```

このﾌﾟﾛｸﾞﾗﾑはC標準ﾗｲﾌﾞﾗﾘからの関数だけを使い、これを検査する時にGCCでｺﾝﾊﾟｲﾙされましたが、おそらくどのCｺﾝﾊﾟｲﾗでもｺﾝﾊﾟｲﾙして同様に上手く動くでしょう。

6. 追補B – ｺｰﾄﾞの大きさ測定の詳細

単変数例

下表は単変数時間と単変数日付と時間の例のｺﾝﾊﾟｲﾙの大きさと実行時間を示します。ｺﾝﾊﾟｲﾙの大きさ測定は(どのUSART構成設定もなしで)必要なAtmel START構成設定と秒ごとに変数を増してこれが呼ばれる時毎にﾌﾗｸﾞを設定するRTC ISRだけをｺﾝﾊﾟｲﾙすることによって生成されました。加えて、主関数はﾌﾗｸﾞが設定されているかを調べます。ﾌﾗｸﾞが設定されている場合、主関数はﾌﾗｸﾞを解除し、秒を人が可読な時間に変換して時刻または時刻と日付の構造でこれを格納する相互変換関数を呼びます。ﾌﾗｸﾞが立ってない場合、main()関数はCPUをｽﾀﾝﾊﾞｲ休止に設定します。比較のため、下表は空のRTC ISRを持つRTC構成設定とISR完了後にCPUをｽﾀﾝﾊﾞｲ休止に設定するmainだけを持ちます。

表6-1. ｺﾝﾊﾟｲﾙの大きさと実行時間

ﾌﾟﾛｸﾞﾗﾑ	GCC -O3最適化		GCC -Os最適化		ｸﾛｯｸ周期数
	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	
空のISRで構成されたRTCと休止	444ﾊﾞｲﾄ	0ﾊﾞｲﾄ	262ﾊﾞｲﾄ	0ﾊﾞｲﾄ	該当なし
単変数時間	1184ﾊﾞｲﾄ	11ﾊﾞｲﾄ	1000ﾊﾞｲﾄ	11ﾊﾞｲﾄ	589
単変数日付と時間	1598ﾊﾞｲﾄ	27ﾊﾞｲﾄ	1372ﾊﾞｲﾄ	27ﾊﾞｲﾄ	1370～1880

多変数例

下表は多変数時間と多変数日付と時間の例のｺﾝﾊﾟｲﾙの大きさと実行時間を示します。ｺﾝﾊﾟｲﾙの大きさ測定は(どのUSART構成設定もなしで)必要なAtmel START構成設定と秒毎にﾌﾗｸﾞを設定するRTC ISRだけをｺﾝﾊﾟｲﾙすることによって生成されました。加えて、main()関数はﾃﾞﾊﾞｲｽをｽﾀﾝﾊﾞｲ休止に設定するのとﾌﾗｸﾞが設定されているかを調べます。ﾌﾗｸﾞが設定されている場合、main()関数はﾌﾗｸﾞを解除し、時間増加関数を呼び出し、必要な比較と必要とされる時に他の時間に関連する変数の増加を行います。変数は時刻と日付の構造で格納されます。

表6-2. ｺﾝﾊﾟｲﾙの大きさと実行時間

ﾌﾟﾛｸﾞﾗﾑ	GCC -O3最適化		GCC -Os最適化		平均実行時間
	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ	ﾃﾞｰﾀ ﾒﾓﾘ	
空のISRで構成されたRTCと休止	444ﾊﾞｲﾄ	0ﾊﾞｲﾄ	262ﾊﾞｲﾄ	0ﾊﾞｲﾄ	該当なし
多変数時間	600ﾊﾞｲﾄ	6ﾊﾞｲﾄ	416ﾊﾞｲﾄ	6ﾊﾞｲﾄ	～8周期/秒
多変数日付と時間	1598ﾊﾞｲﾄ	27ﾊﾞｲﾄ	1372ﾊﾞｲﾄ	27ﾊﾞｲﾄ	～8周期/秒

7. 改訂履歴

資料改訂	日付	注釈
A	2019年12月	初版資料公開

Microchipｳｪﾌﾞ ｻｲﾄ

Microchipはhttp://www.microchip.com/で当社のｳｪﾌﾞ ｻｲﾄ経由でのｵﾝﾗｲﾝ支援を提供します。このｳｪﾌﾞ ｻｲﾄはお客様がﾌｧｲﾙや情報を容易に利用可能にするのに使われます。利用可能な情報のいくつかは以下を含みます。

- **製品支援** – ﾃﾞｰﾀｼｰﾄと障害情報、応用記述と試供ﾌﾟﾛｸﾞﾗﾑ、設計資源、使用者の手引きとﾊｰﾄﾞｳｪｱ支援資料、最新ｿﾌﾄｳｪｱ配布と保管されたｿﾌﾄｳｪｱ
- **全般的な技術支援** – 良くある質問(FAQ)、技術支援要求、ｵﾝﾗｲﾝ検討ｸﾞﾙｰﾌﾟ、Microchip設計協力課程会員一覧
- **Microshipの事業** – 製品選択器と注文の手引き、最新Microchip報道発表、ｾﾐﾅｰとｲﾍﾞﾝﾄの一覧、Microchip営業所の一覧、代理店と代表する工場

製品変更通知ｻｰﾋﾞｽ

Microchipの製品変更通知ｻｰﾋﾞｽはMicrochip製品を最新に保つのに役立ちます。加入者は指定した製品系統や興味のある開発ﾂｰﾙに関連する変更、更新、改訂、障害情報がある場合に必ず電子ﾒｰﾙ通知を受け取ります。

登録するにはhttp://www.microchip.com/pcnへ行って登録指示に従ってください。

お客様支援

Microchip製品の使用者は以下のいくつかのﾁｬﾈﾙを通して支援を受け取ることができます。

- 代理店または販売会社
- 最寄りの営業所
- 組み込み解決技術者(ESE:Embedded Solutions Engineer)
- 技術支援

お客様は支援に関してこれらの代理店、販売会社、またはESEに連絡を取るべきです。最寄りの営業所もお客様の手助けに利用できます。営業所と位置の一覧はこの資料の後ろに含まれます。

技術支援はhttp://www.microchip.com/supportでのｳｪﾌﾞ ｻｲﾄを通して利用できます。

Microchipﾃﾞﾊﾞｲｽ ｺｰﾄﾞ保護機能

Microchipﾃﾞﾊﾞｲｽでの以下のｺｰﾄﾞ保護機能の詳細に注意してください。

- Microchip製品はそれら特定のMicrochipﾃﾞｰﾀｼｰﾄに含まれる仕様に合致します。
- Microchipは意図した方法と通常条件下で使われる時に、その製品系統が今日の市場でその種類の最も安全な系統の1つであると考えます。
- ｺｰﾄﾞ保護機能を破るのに使われる不正でおそらく違法な方法があります。当社の知る限りこれらの方法の全てはMicrochipのﾃﾞｰﾀｼｰﾄに含まれた動作仕様外の方法でMicrochip製品を使うことが必要です。おそらく、それを行う人は知的財産の窃盗に関与しています。
- Microchipはそれらのｺｰﾄﾞの完全性について心配されているお客様と共に働きたいと思います。
- Microchipや他のどの半導体製造業者もそれらのｺｰﾄﾞの安全を保証することはできません。ｺｰﾄﾞ保護は当社が製品を”破ることができない”として保証すると言うことを意味しません。

ｺｰﾄﾞ保護は常に進化しています。Microchipは当社製品のｺｰﾄﾞ保護機能を継続的に改善することを約束します。Microchipのｺｰﾄﾞ保護機能を破る試みはﾃﾞｼﾞﾀﾙ ﾐﾚﾆｱﾑ著作権法に違反するかもしれません。そのような行為があなたのｿﾌﾄｳｪｱや他の著作物に不正なｱｸｾｽを許す場合、その法律下の救済のために訴権を持つかもしれません。

法的通知

ﾃﾞﾊﾞｲｽ応用などに関してこの刊行物に含まれる情報は皆さまの便宜のためにだけ提供され、更新によって取り換えられるかもしれません。皆さまの応用が皆さまの仕様に合致するのを保証するのは皆さまの責任です。**Microchipはその条件、品質、性能、商品性、目的適合性を含め、明示的にも黙示的にもその情報に関連して書面または表記された書面または黙示の如何なる表明や保証もしません。**Microchipはこの情報とそれの使用から生じる全責任を否認します。生命維持や安全応用でのMicrochipﾃﾞﾊﾞｲｽの使用は完全に購入者の危険性で、購入者はそのような使用に起因する全ての損害、請求、訴訟、費用からMicrochipを擁護し、補償し、免責にすることに同意します。他に言及されない限り、Microchipのどの知的財産権下でも暗黙的または違う方法で許認可は譲渡されません。

商標

品質管理ｼｽﾃﾑ

Microchipの品質管理ｼｽﾃﾑに関する情報についてはhttp://www.microchip.com/qualityを訪ねてください。

日本語© *HERO* 2019.

本応用記述はMicrochipのAN3289応用記述(DS00003289A-2019年12月)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。

世界的な販売とサービス

米国	亜細亜/太平洋	亜細亜/太平洋	欧州
本社 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 技術支援: http://www.microchip.com/support ウェブ アドレス: http://www.microchip.com **アトランタ** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **オースチン** TX Tel: 512-257-3370 **ボストン** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **シカゴ** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **ダラス** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **デトロイト** Novi, MI Tel: 248-848-4000 **ヒューストン** TX Tel: 281-894-5983 **インディアナポリス** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **ロサンゼルス** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **ローリー** NC Tel: 919-844-7510 **ニューヨーク** NY Tel: 631-435-6000 **サンホセ** CA Tel: 408-735-9110 Tel: 408-436-4270 **カナダ - トロント** Tel: 905-695-1980 Fax: 905-695-2078	**オーストラリア - シドニー** Tel: 61-2-9868-6733 **中国 - 北京** Tel: 86-10-8569-7000 **中国 - 成都** Tel: 86-28-8665-5511 **中国 - 重慶** Tel: 86-23-8980-9588 **中国 - 東莞** Tel: 86-769-8702-9880 **中国 - 広州** Tel: 86-20-8755-8029 **中国 - 杭州** Tel: 86-571-8792-8115 **中国 - 香港特別行政区** Tel: 852-2943-5100 **中国 - 南京** Tel: 86-25-8473-2460 **中国 - 青島** Tel: 86-532-8502-7355 **中国 - 上海** Tel: 86-21-3326-8000 **中国 - 瀋陽** Tel: 86-24-2334-2829 **中国 - 深圳** Tel: 86-755-8864-2200 **中国 - 蘇州** Tel: 86-186-6233-1526 **中国 - 武漢** Tel: 86-27-5980-5300 **中国 - 西安** Tel: 86-29-8833-7252 **中国 - 廈門** Tel: 86-592-2388138 **中国 - 珠海** Tel: 86-756-3210040	**インド - バンガロール** Tel: 91-80-3090-4444 **インド - ニューデリー** Tel: 91-11-4160-8631 **インド - プネー** Tel: 91-20-4121-0141 **日本 - 大阪** Tel: 81-6-6152-7160 **日本 - 東京** Tel: 81-3-6880-3770 **韓国 - 大邱** Tel: 82-53-744-4301 **韓国 - ソウル** Tel: 82-2-554-7200 **マレーシア - クアラルンプール** Tel: 60-3-7651-7906 **マレーシア - ペナン** Tel: 60-4-227-8870 **フィリピン - マニラ** Tel: 63-2-634-9065 **シンガポール** Tel: 65-6334-8870 **台湾 - 新竹** Tel: 886-3-577-8366 **台湾 - 高雄** Tel: 886-7-213-7830 **台湾 - 台北** Tel: 886-2-2508-8600 **タイ - バンコク** Tel: 66-2-694-1351 **ベトナム - ホーチミン** Tel: 84-28-5448-2100	**オーストリア - ヴェルス** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **デンマーク - コペンハーゲン** Tel: 45-4450-2828 Fax: 45-4485-2829 **フィンランド - エスポー** Tel: 358-9-4520-820 **フランス - パリ** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **ドイツ - ガルヒング** Tel: 49-8931-9700 **ドイツ - ハーン** Tel: 49-2129-3766400 **ドイツ - ハイルブロン** Tel: 49-7131-72400 **ドイツ - カールスルーエ** Tel: 49-721-625370 **ドイツ - ミュンヘン** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **ドイツ - ローゼンハイム** Tel: 49-8031-354-560 **イスラエル - ラーナナ** Tel: 972-9-744-7705 **イタリア - ミラノ** Tel: 39-0331-742611 Fax: 39-0331-466781 **イタリア - パドバ** Tel: 39-049-7625286 **オランダ - デルーネン** Tel: 31-416-690399 Fax: 31-416-690340 **ノルウェー - トロンハイム** Tel: 47-72884388 **ポーランド - ワルシャワ** Tel: 48-22-3325737 **ルーマニア - ブカレスト** Tel: 40-21-407-87-50 **スペイン - マドリード** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **スウェーデン - イェーテボリ** Tel: 46-31-704-60-40 **スウェーデン - ストックホルム** Tel: 46-8-5090-4654 **イギリス - ウォーキンガム** Tel: 44-118-921-5800 Fax: 44-118-921-5820