参 考 資 料

Application Report
JAJA237

TMSx70　MCUのRTI(リアルタイム割り込み)を使用してオペレーティングシステムのTickを発生させる方法

Hari Udayakumar

要　約

このアプリケーションノートの目的は、TMSx70シリーズMCUのRTIモジュールの設定方法の一助となることである。

TIのTMSx70ファミリーのMCUはローパワー32ビットRISC MCUファミリーの一員で先進的なアーキテクチャ、豊富な周辺セットを備えている。

内　容

説明図

SPNA123　翻訳版

最新の英語版資料
http://www.ti.com/lit/spna123

1　イントロダクション

1.1　リアルタイム割り込み

RTIモジュールはOSのためのタイマー機能やコードのベンチマークに必要なタイマー機能を提供する。このモジュールはマルチプルカウンターを備えており、このマルチプルカウンターはOSのスケジューリングに必要なタイムベースを定義する。

このアプリケーションノートはTMSx70 RTI ユーザーガイドと併用して参照されるものである。

主な特長

- 2つの独立したカウンターブロックで構成（異なったタイムベースを生成）それぞれのブロックは次のもので構成されている
  - 32ビット　プリスケール・カウンタ
  - 32ビット　フリーラン・カウンタ
- システムあるいは周辺割り込みのために、2つまでのタイムスタンプ（キャプチャ）。それぞれのカウンタブロックに対して1つ
- フリーラン・カウンタ0は、内部プリスケール・カウンタあるいは外部イベントによりインクリメント可能
  （クロック監視を含むFlexRayネットワークでのアプリケーションの同期のため）
- 外部クロックが予め定義されたウィンドウの中でインクリメントされ故障と判断された場合は、オプショナルの外部クロック監視回路によって内部のプリスケール・カウンタ0に切り替え可能
- OS用のティックやDMAリクエストのための4つのコンフィギュアブル・コンペアレジスタ
  それぞれのイベントはカウンタブロック0あるいはカウンタブロック1でドライブされる
- 全てのコンペアレジスタの自動アップデート（コンペアマッチで周期割り込みを発生させるとき）
- 割り込みの高速なイネーブル/ディスエーブル
- どのようなクロックソースから生成されたRTIクロック入力でも、システムモジュールの中で選択可能

1.2 RTI モジュール・ブロック・ダイヤグラム

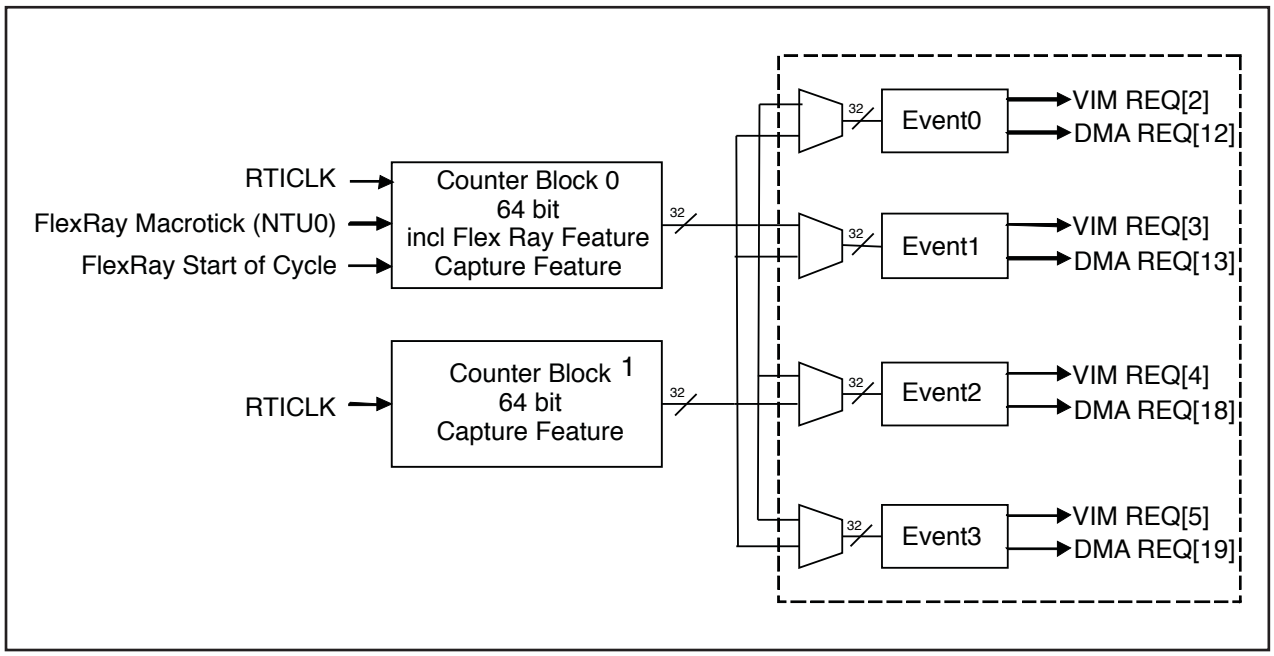

図 1. RTI Module Block Diagram

1.3 RTIカウンタ ブロック(x) ダイヤグラム

下図 (図2) はRTIカウンタブロック(X)の簡略化されたブロック図である。

RTICLKは32ビットRTIUPCxカウンタでプリスケールされる。

RTIUPC x カウンタはRTICPUC x レジスタの中の比較値が到達するまでカウントアップする。比較値が一致すると、RTIFRC x カウンタはインクリメントされる。

$$If\ CPUx = 0: \quad FRCx = \frac{RTICLK}{2^{32}}$$
$$If\ CPUx \neq 0: \quad FRCx = \frac{RTICLK}{CPUCx + 1} \qquad (1)$$

2 OSのためのティック生成

この項はある周期 (調整可能) でOSティック割り込みを発生するためのRTIモジュールの使い方を説明する。

必要なOSティックの生成と、それに相当するタスクを実行するために３つのコンペア割り込みが用いられる。タスクコールを解りやすくするために、独立したI/Oピンはサンプルコードの中でトグルされている。

異なったタイムベースを使ってタスクをトリガーするために、３つの異なった割り込みをどのように構成するかを、このアプリケーションノートは示している。(図3を参照)

2.1 フロー図

第3図は割り込みを用いて、異なる時間間隔でタスクを起動するRTIモジュールの使い方を示します。

- Compare 0 Interrupt – Task1 (GIO- PortA [0] Pin Toggle) - 10mS.
- Compare 1 Interrupt – Task2 (GIO- PortA [1] Pin Toggle) - 5mS.
- Compare 2 Interrupt – Task3 (GIO- PortA [2] Pin Toggle) - 1mS.

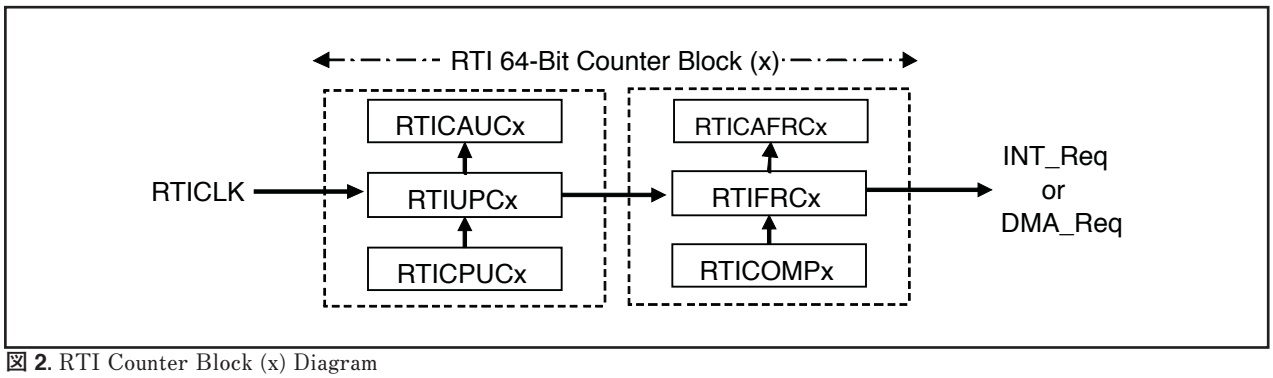

図 2. RTI Counter Block (x) Diagram

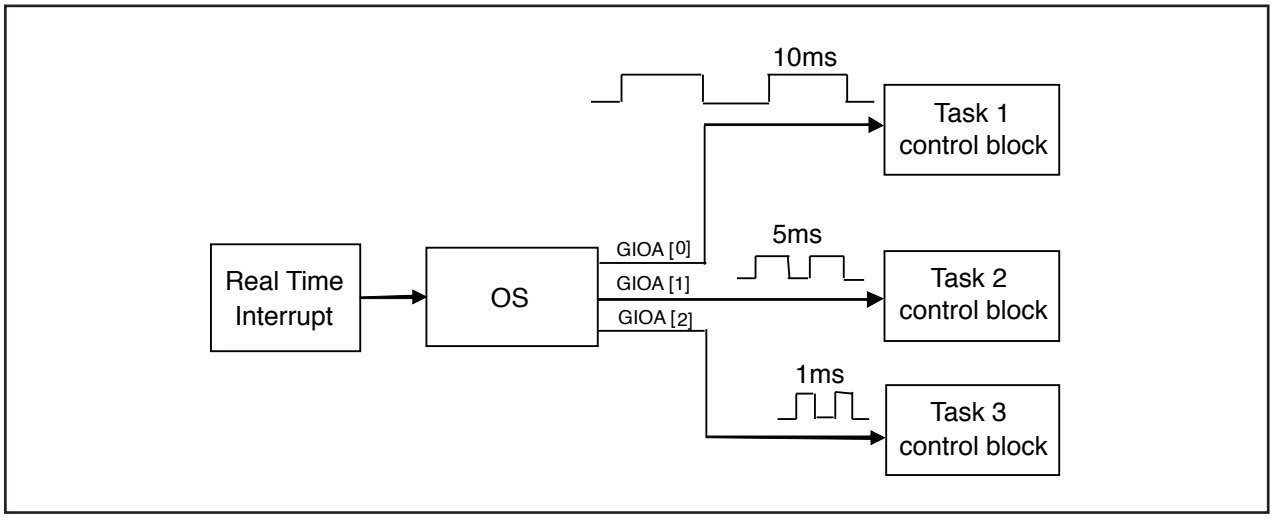

図 3. Operating System Tick Generation Block Diagram

2.2 周波数とTカウントの計算について

カウンタブロックへのRTICLKはアプリケーションの要求にしたがって、システムモジュールの中のRTICLKSRCレジスタをプログラム可能である。もし、RTICLK源がVCLKだけの場合、RTICLK源はVCLKよりも少なくとも３倍遅くなくてはならない。これはRTICLKSRCレジスタのRTI x DIVビットをコンフィグすることによって可能。図4はRTICLKSRCレジスタとRTICPUC x レジスタを使用してRTICLKの選択とプリスケールを設定することを現している。

RTICPUCxレジスタを設定してRTICLKを2分の1にプリスケールします。セクション1．3を参照してください。

$$\bullet\ RITCLK = VCLK = 8\ MHz$$

$$\bullet\ FRC0CLK = \frac{RTICLK}{CPUC0+1} = \frac{8\ MHz}{1+1} = 4\ MHz$$

$$\bullet\ Tcount = \frac{1}{FRC0CLK} = \frac{1}{4\ MHz} = 0.25\ \mu s \qquad (2)$$

2.3 カウント値の比較の計算法

周期的なOSのティックのためのコンペアレジスタのカウント値はTカウント周期から計算で求められる。RTICLKはUPカウンタ(x)でプリスケールされ、それぞれのフリーラン(x)カウンタに供給される。

- Task1 = T1period = 10ms
- Task2 = T2period = 5ms
- Task3 = T3period = 1ms

このように、INT0、INT1、INT2の割り込みのために周期的なティックを発生することができる。

$$\bullet\ Compare\ 0\ Count\ Value\ (Task1) = \frac{T1period}{Tcount} = \frac{10\ ms}{0.25\ \mu s} = 40000$$

$$\bullet\ Compare\ 1\ Count\ Value\ (Task2) = \frac{T2period}{Tcount} = \frac{5\ ms}{0.25\ \mu S} = 20000$$

$$\bullet\ Compare\ 2\ Count\ Value\ (Task3) = \frac{T3period}{Tcount} = \frac{1\ ms}{0.25\ \mu S} = 4000 \qquad (3)$$

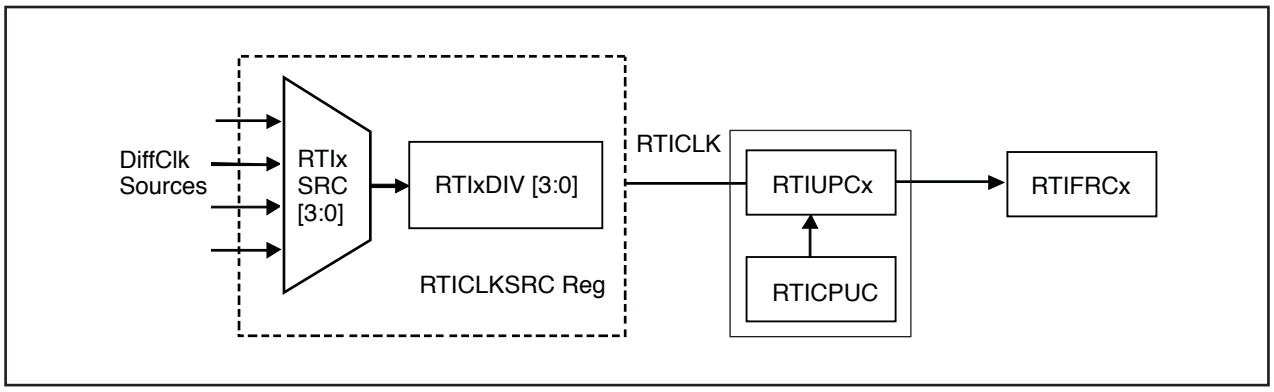

図 4. RTI Clock Source Selection and Pre-Scaling

2.4 OSティック出力波形

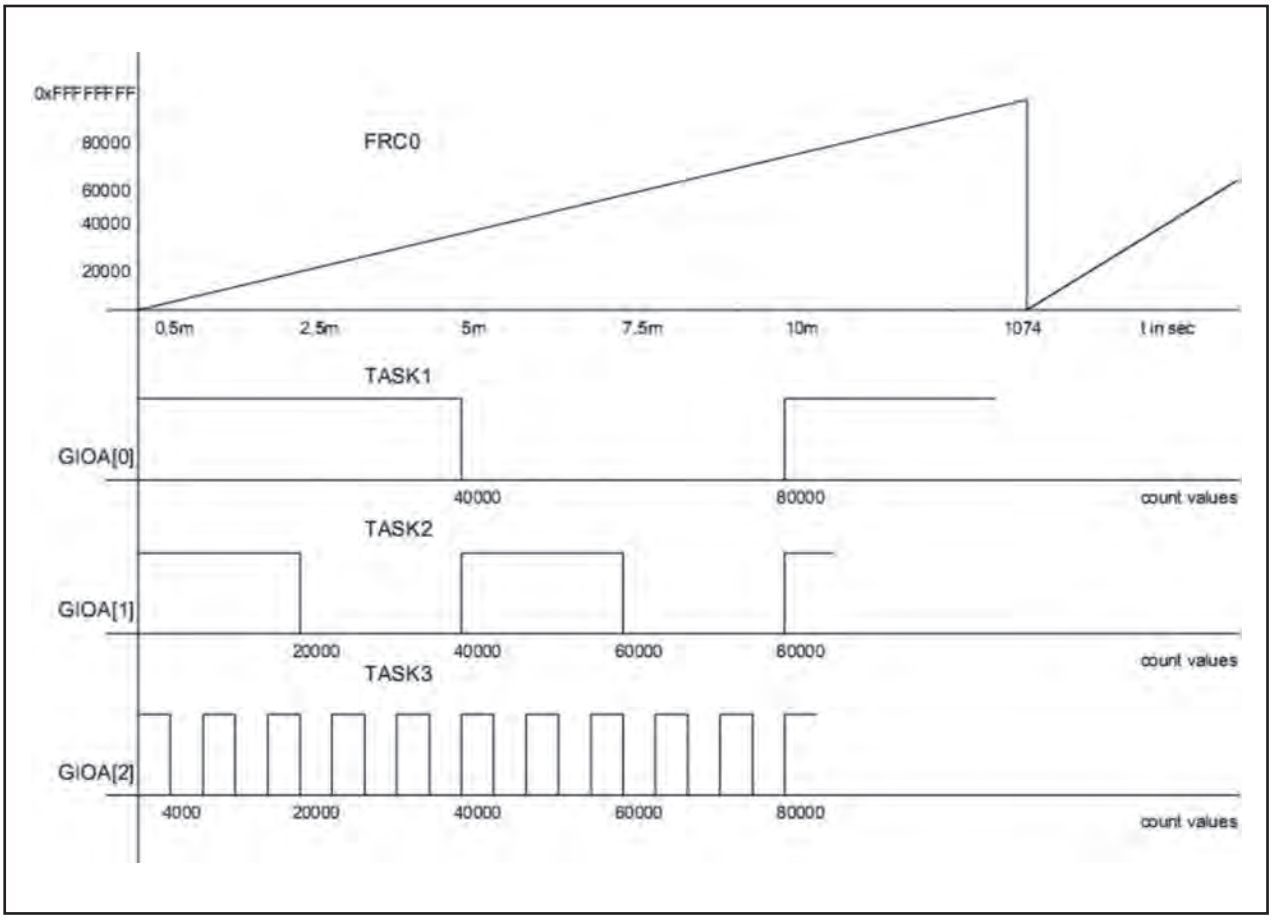

図 5. OSティック出力波形

2.5 OSティック生成のためのコンフィグレーション（手順）

- システムモジュールの中のRTICLKSRCレジスタを使用して、VCLKを選択する
- 次のクロック設定を選択：

$$\begin{aligned} \bullet\ HCLK &= OSCIN \\ \bullet\ VCLK &= \frac{HCLK}{2} \\ \bullet\ RTICLK &= VCLK \end{aligned} \qquad (4)$$

- RTICOMP（x）レジスタとそれぞれのRTIUDCP(x)レジスタを(x)カウント値を初期化するために設定
- GIOポートAを出力に設定
- RTIモジュール割り込みを設定
  例：RTISETINTとステータスレジスタ
  例：RTIINTFLAG
  — 割り込みのペンディングをクリア
  — コンペア0, 1, 2割り込みをイネーブル
- コンペア０割り込みサービスルーチンは次のことを行うように設定：
  — Comp0割り込みフラグはRTIINTFLAGレジスタのINT0ビットに１を書き込みことによってクリアされる
  — GIO ポートA — ピン０をトグル
- コンペア１割り込みサービスルーチンは次のことを行うように設定：
  — Comp1割り込みフラグはRTIINTFLAGレジスタのINT1ビットに1を書き込みことによってクリアされる
  — GIO ポートA — ピン1をトグル
- コンペア2割り込みサービスルーチンは次のことを行うように設定：
  — Comp2割り込みフラグはRTIINTFLAGレジスタのINT2ビットに1を書き込みことによってクリアされる
  — GIO ポートA — ピン2をトグル

2.6 Software Flow

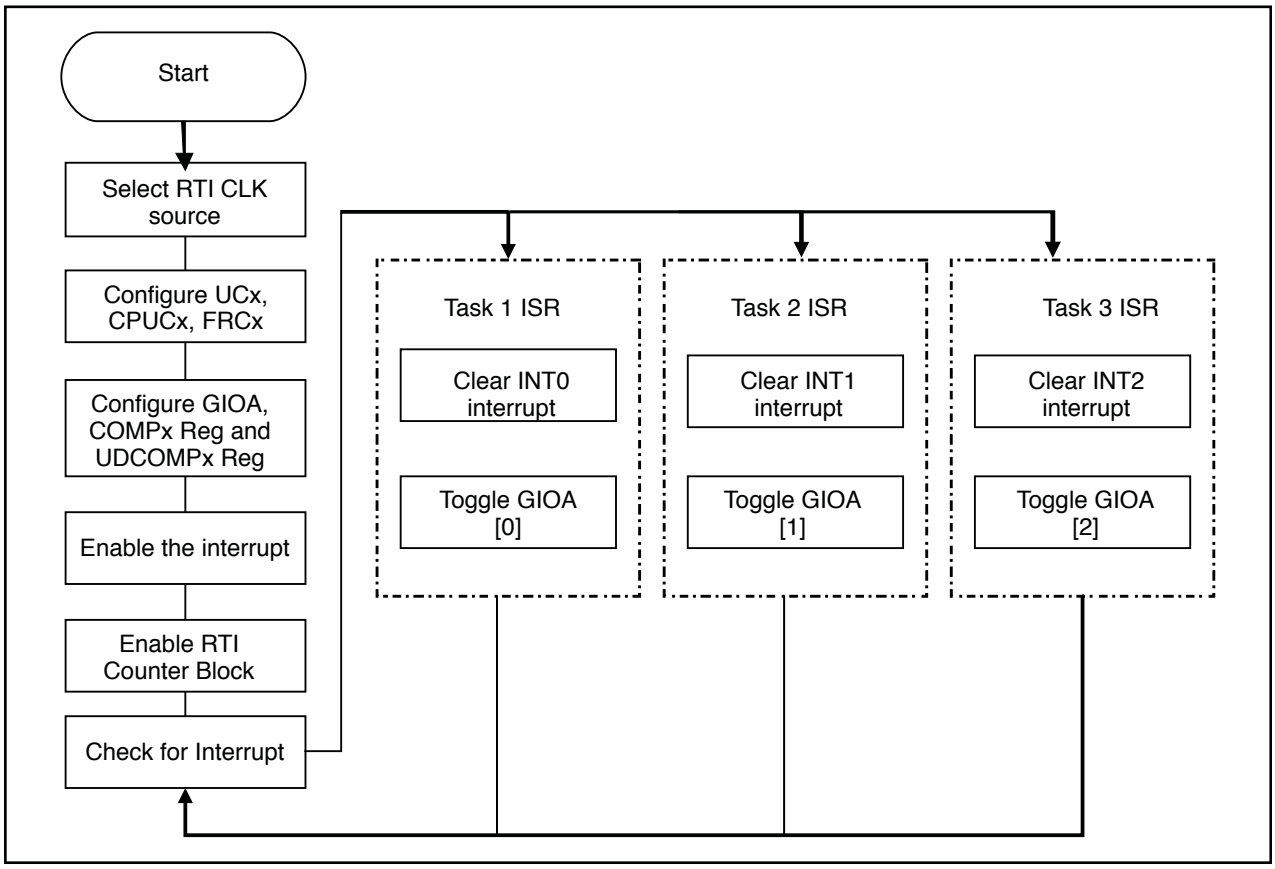

図 6. Software Flow Diagram

2.7 Captured Output Waveform

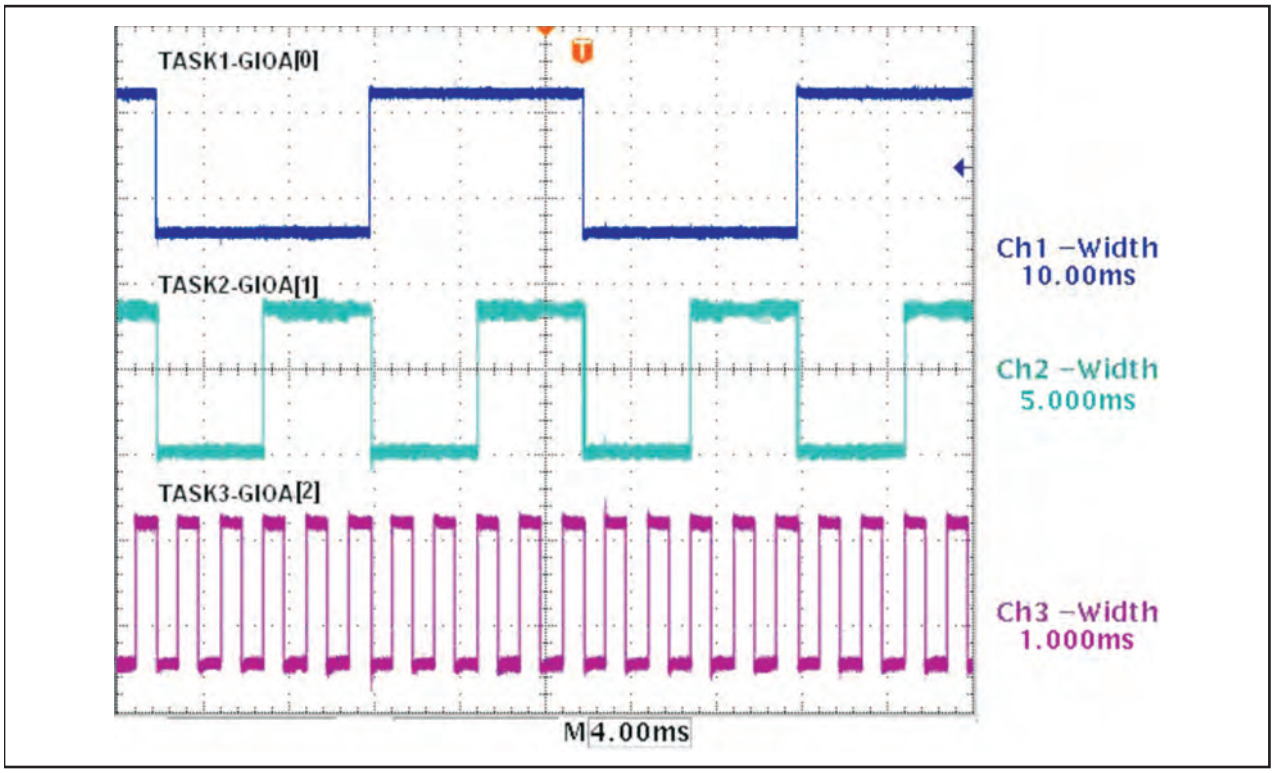

図 7. Output Waveform

3　付録A　ソフトウェアリスト

```
/*****************************************************************************************************
*/
/* file:RTI1_OST.c                                                  */
/*                                                 */
/* RTI generates Periodic operating system tick using compare interrupts                */
/*                                                 */
/* Purposes: Generate periodic operating system tick using compare interrupts            */
/* Conditions: PowerOnReset must be applied prior to run test                  */
/*                                                 */
/*****************************************************************************************************
*/
#include "rti.h"
#include "vim.h"
#include "pcr.h"
#include "system.h"
#include "swi_util.h"
#include "device.h"
#include "module.h"
#include "gio470.h"
SYSTEM_ST    *SYS_Ptr    = (SYSTEM_ST *) SYSTEM ;
PCR_ST        *PCR_Ptr    = (PCR_ST *) PCR;
VIM_ST        *VIM_Ptr    = (VIM_ST *) VIM;
VIM_RAM_ST    *VIM_RAM_Ptr    = (VIM_RAM_ST *) VIM_RAM;
RTI_ST        *RTI_Ptr    = (RTI_ST *) RTI;
GIO_ST        *GIO_Ptr    = (GIO_ST *) GIO1;
#define        TASK1        40000
#define        TASK2        20000
#define        TASK3        4000
#define        END_VALUE    5000
main()
{
/********************************** VIM Initialization*****************************************/
VIM_RAM_Init();                    //VIM initialization
RTI_irq_int_enable();                  //Enabling the RTI Interrupts in VIM
swi_enable_irq()    ;                //Int_irq_enable Enable only IRQ
/********************************** GIO PortA
Initialization*************************************/
GIO_Ptr->Gcr0_UN.Gcr0_ST.Reset_B1=1;            //enabling GIO
GIO_Ptr->Port_ST[0].Dir_UL=0x07;           //PortA.0,1,2 has output
GIO_Ptr->Port_ST[0].Dout_UL=0x00;
/**********************************RTI Clk Source
Initialization*********************************/
SYS_Ptr->RCLKSRC_UN.RCLKSRC_ST.RTI1SRC_B4 = 8;    //clock source 8 i.e VClk
SYS_Ptr->RCLKSRC_UN.RCLKSRC_ST.RTI1DIV_B2 = 0;    //div the clk source 0 by 1
/**********************************RTI
Initialization*************************************************/
RTI_Ptr->RTIGCTRL_UN.RTIGCTRL_UL= 0x00000000;    //Disable RTIUC0 and RTIUC1
RTI_Ptr->RTIUC0_UL= 0x00000000;            //Initialize up Counter
RTI_Ptr->RTIFRC0_UL= 0x00000000;            //Initialize Free Running Counter
RTI_Ptr->RTICPUC0_UL= 0x00000001;    //Set Compare Up Counter value to Prescale RTICLK.
/**********************************TASK
Initialization*****************************************/
RTI_Ptr->RTICOMP0_UL = TASK1;
RTI_Ptr->RTIUDCP0_UL = TASK1;
RTI_Ptr->RTICOMP1_UL = TASK2;
RTI_Ptr->RTIUDCP1_UL = TASK2;
RTI_Ptr->RTICOMP2_UL = TASK3;
RTI_Ptr->RTIUDCP2_UL = TASK3;
RTI_Ptr->RTIINTFLAG_UN.RTIINTFLAG_UL = 0x0000000F;    //Clearing the pending Interrupts Flags
RTI_Ptr->RTICOMPCTRL_UN.RTICOMPCTRL_UL = 0x00000000;    //Initializing the Compare Counter
register
RTI_Ptr->RTISETINT_UN.RTISETINT_UL = 0x00000007;    //enabling RTI Interrupt in RTI
RTI_Ptr->RTIGCTRL_UN.RTIGCTRL_UL = 0x00000001;    //Enable the RTIUC0
/*****************************************Infinite loop*************************************/
while(1);
}
/*****************************************ISR.c*****************************************
**/
#include "rti.h"
#include "vim.h"
#include "gio470.h"
#pragma INTERRUPT (Compare0_Handler, IRQ)
#pragma INTERRUPT (Compare1_Handler, IRQ)
#pragma INTERRUPT (Compare2_Handler, IRQ)
void Compare0_Handler(void);
void Compare1_Handler(void);
void Compare2_Handler(void);
void Compare0_Handler(void)
{
RTI_Ptr->RTIINTFLAG_UN.RTIINTFLAG_UL= 0x00000001;    //Clearing the Interrupt Flag 0
GIO_Ptr->Port_ST[0].Dout_UL ^= 0x01;            //Toggling the GIOA[0] Port
}
void Compare1_Handler(void)
{
RTI_Ptr->RTIINTFLAG_UN.RTIINTFLAG_UL= 0x00000002;    //Clearing the Interrupt Flag 1
GIO_Ptr->Port_ST[0].Dout_UL ^= 0x02;            //Toggling the GIOA[1] Port
}
void Compare2_Handler(void)
{
RTI_Ptr->RTIINTFLAG_UN.RTIINTFLAG_UL= 0x00000004;    //Clearing the Interrupt Flag 2
GIO_Ptr->Port_ST[0].Dout_UL ^= 0x04;            //Toggling the GIOA[2] Port
}
/*****************************************END*****************************************
/
```

ご注意

日本テキサス・インスツルメンツ株式会社（以下TIJといいます）及びTexas Instruments Incorporated（TIJの親会社、以下TIJないしTexas Instruments Incorporatedを総称してTIといいます）は、その製品及びサービスを任意に修正し、改善、改良、その他の変更をし、もしくは製品の製造中止またはサービスの提供を中止する権利を留保します。従いまして、お客様は、発注される前に、関連する最新の情報を取得して頂き、その情報が現在有効かつ完全なものであるかどうかご確認下さい。全ての製品は、お客様とTIJとの間に取引契約が締結されている場合は、当該契約条件に基づき、また当該取引契約が締結されていない場合は、ご注文の受諾の際に提示されるTIJの標準販売契約約款に従って販売されます。

TIは、そのハードウェア製品が、TIの標準保証条件に従い販売時の仕様に対応した性能を有していること、またはお客様とTIJとの間で合意された保証条件に従い合意された仕様に対応した性能を有していることを保証します。検査およびその他の品質管理技法は、TIが当該保証を支援するのに必要とみなす範囲で行なわれております。各デバイスの全てのパラメーターに関する固有の検査は、政府がそれ等の実行を義務づけている場合を除き、必ずしも行なわれておりません。

TIは、製品のアプリケーションに関する支援もしくはお客様の製品の設計について責任を負うことはありません。TI製部品を使用しているお客様の製品及びそのアプリケーションについての責任はお客様にあります。TI製部品を使用したお客様の製品及びアプリケーションについて想定されうる危険を最小のものとするため、適切な設計上および操作上の安全対策は、必ずお客様にてお取り下さい。

TIは、TIの製品もしくはサービスが使用されている組み合せ、機械装置、もしくは方法に関連しているTIの特許権、著作権、回路配置利用権、その他のTIの知的財産権に基づいて何らかのライセンスを許諾するということは明示的にも黙示的にも保証も表明もしておりません。TIが第三者の製品もしくはサービスについて情報を提供することは、TIが当該製品もしくはサービスを使用することについてライセンスを与えるとか、保証もしくは是認するということを意味しません。そのような情報を使用するには第三者の特許その他の知的財産権に基づき当該第三者からライセンスを得なければならない場合もあり、またTIの特許その他の知的財産権に基づきTI からライセンスを得て頂かなければならない場合もあります。

TIのデータ・ブックもしくはデータ・シートの中にある情報を複製することは、その情報に一切の変更を加えること無く、かつその情報と結び付られた全ての保証、条件、制限及び通知と共に複製がなされる限りにおいて許されるものとします。当該情報に変更を加えて複製することは不公正で誤認を生じさせる行為です。TIは、そのような変更された情報や複製については何の義務も責任も負いません。

TIの製品もしくはサービスについてTIにより示された数値、特性、条件その他のパラメーターと異なる、あるいは、それを超えてなされた説明で当該TI製品もしくはサービスを再販売することは、当該TI製品もしくはサービスに対する全ての明示的保証、及び何らかの黙示的保証を無効にし、かつ不公正で誤認を生じさせる行為です。TIは、そのような説明については何の義務も責任もありません。

TIは、TIの製品が、安全でないことが致命的となる用途ないしアプリケーション（例えば、生命維持装置のように、TI製品に不良があった場合に、その不良により相当な確率で死傷等の重篤な事故が発生するようなもの）に使用されることを認めておりません。但し、お客様とTIの双方の権限有る役員が書面でそのような使用について明確に合意した場合は除きます。たとえTIがアプリケーションに関連した情報やサポートを提供したとしても、お客様は、そのようなアプリケーションの安全面及び規制面から見た諸問題を解決するために必要とされる専門的知識及び技術を持ち、かつ、お客様の製品について、またTI製品をそのような安全でないことが致命的となる用途に使用することについて、お客様が全ての法的責任、規制を遵守する責任、及び安全に関する要求事項を満足させる責任を負っていることを認め、かつそのことに同意します。さらに、もし万一、TIの製品がそのような安全でないことが致命的となる用途に使用されたことによって損害が発生し、TIないしその代表者がその損害を賠償した場合は、お客様がTIないしその代表者にその全額の補償をするものとします。

TI製品は、軍事的用途もしくは宇宙航空アプリケーションないし軍事的環境、航空宇宙環境にて使用されるようには設計もされていませんし、使用されることを意図されておりません。但し、当該TI製品が、軍需対応グレード品、若しくは「強化プラスティック」製品としてTIが特別に指定した製品である場合は除きます。TIが軍需対応グレード品として指定した製品のみが軍需品の仕様書に合致いたします。お客様は、TIが軍需対応グレード品として指定していない製品を、軍事的用途もしくは軍事的環境下で使用することは、もっぱらお客様の危険負担においてなされるということ、及び、お客様がもっぱら責任をもって、そのような使用に関して必要とされる全ての法的要求事項及び規制上の要求事項を満足させなければならないことを認め、かつ同意します。

TI製品は、自動車用アプリケーションないし自動車の環境において使用されるようには設計されていませんし、また使用されることを意図されておりません。但し、TIがISO/TS 16949の要求事項を満たしていると特別に指定したTI製品は除きます。お客様は、お客様が当該TI指定品以外のTI製品を自動車用アプリケーションに使用しても、TIは当該要求事項を満たしていなかったことについて、いかなる責任も負わないことを認め、かつ同意します。

弊社半導体製品の取り扱い・保管について

半導体製品は、取り扱い、保管・輸送環境、基板実装条件によっては、お客様での実装前後に破壊/劣化、または故障を起こすことがあります。

弊社半導体製品のお取り扱い、ご使用にあたっては下記の点を遵守して下さい。

1. 静電気
   - 素手で半導体製品単体を触らないこと。どうしても触る必要がある場合は、リストストラップ等で人体からアースをとり、導電性手袋等をして取り扱うこと。
   - 弊社出荷梱包単位（外装から取り出された内装及び個装）又は製品単品で取り扱いを行う場合は、接地された導電性のテーブル上で（導電性マットにアースをとったもの等）、アースをした作業者が行うこと。また、コンテナ等も、導電性のものを使うこと。
   - マウンタやはんだ付け設備等、半導体の実装に関わる全ての装置類は、静電気の帯電を防止する措置を施すこと。
   - 前記のリストストラップ・導電性手袋・テーブル表面及び実装装置類の接地等の静電気帯電防止措置は、常に管理されその機能が確認されていること。
2. 温・湿度環境
   - 温度：0～40℃、相対湿度：40～85%で保管・輸送及び取り扱いを行うこと。（但し、結露しないこと。）
   - 直射日光があたる状態で保管・輸送しないこと。
3. 防湿梱包
   - 防湿梱包品は、開封後は個別推奨保管環境及び期間に従い基板実装すること。
4. 機械的衝撃
   - 梱包品（外装、内装、個装）及び製品単品を落下させたり、衝撃を与えないこと。
5. 熱衝撃
   - はんだ付け時は、最低限260℃以上の高温状態に、10秒以上さらさないこと。（個別推奨条件がある時はそれに従うこと。）
6. 汚染
   - はんだ付け性を損なう、又はアルミ配線腐食の原因となるような汚染物質（硫黄、塩素等ハロゲン）のある環境で保管・輸送しないこと。
   - はんだ付け後は十分にフラックスの洗浄を行うこと。（不純物含有率が一定以下に保証された無洗浄タイプのフラックスは除く。）

以上

2001.11