Application Report
SPNA123–July 2010

# Generating Operating System Tick Using RTI on a Hercules™ ARM™ Safety Microcontroller

*Hari Udayakumar*

**ABSTRACT**

The purpose of this application report is to provide help for setting up the RTI Module of Hercules Safety Microcontrollers. The Hercules family of microcontrollers from Texas Instruments is a family of 32-bit RISC microcontrollers with an advanced safety architecture and a rich peripheral set.

**Contents**



**List of Figures**

# 1 Introduction

## 1.1 Real Time Interrupt

The Real Time Interrupt Module (RTI) provides timer functionality for operating systems and for benchmarking code. The module incorporates multiple counters, which define the timebases needed for scheduling in the operating system.

This document needs to be referenced along with the Hercules Safety MCU (RM4x, TMS570LS31x/21x, TMS570LS20x/10x, TMS470M) RTI User's Guide found in the the appropriate Technical Reference Manual.

**Main Features**

- Up to two independent counter blocks for generating different timebases. Each block consists of:
  - One 32-bit prescale counter
  - One 32-bit free running counter
- Up to two time stamp (capture) functions for system or peripheral interrupts, one for each counter block
- Free running counter 0 can be incremented by either the internal prescale counter or by an external event (for application synchronization to FlexRay network including clock supervision)
- Optional External Clock supervising circuit to switch to internal prescale counter 0, if external clock source fails to increment in a predefined window
- Four configurable compare registers for generating operating system ticks or DMA requests. Each event can be driven by either counter block 0 or counter block 1
- Automatic update of all compare registers on compare match to generate periodic interrupts
- Fast enabling/disabling of interrupts
- RTI clock input derived from any of the available clock sources, selectable in the System Module

## 1.2 RTI Module Block Diagram

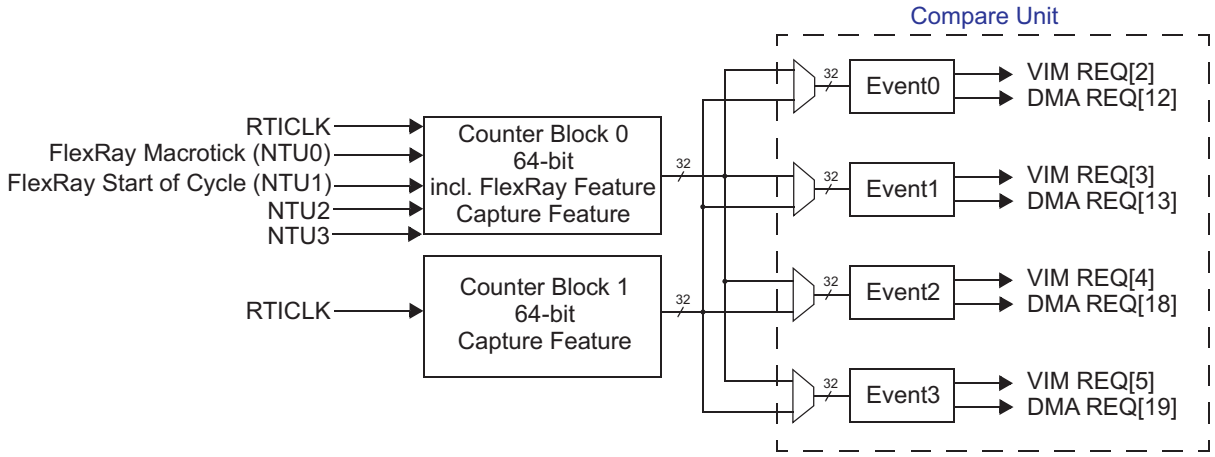


**Figure 1. RTI Module Block Diagram**

### 1.3 RTI Counter Block (x) Diagram

Figure 2 shows the simplified block diagram of the RTI Counter Block (x). The RTICLK is prescaled by 32 bit RTIUPCx Counter. The RTIUPCx Counter counts up until the compare value in the RTICPUCx register is reached. When the compare matches, RTIFRCx counter is incremented.

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

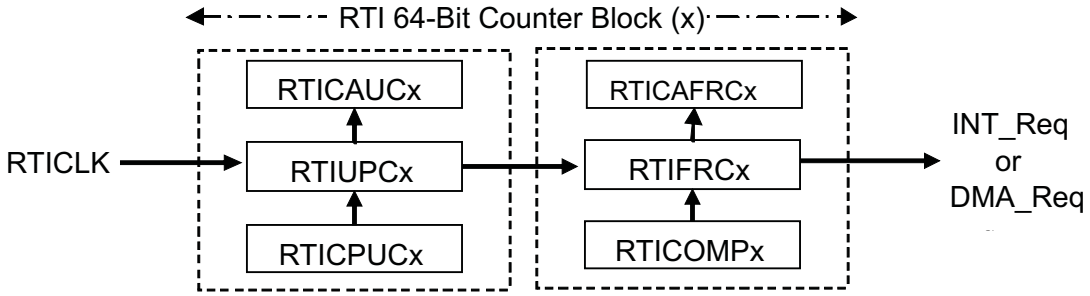


**Figure 2. RTI Counter Block (x) Diagram**

## 2 Operating System Tick Generation

This section describes how to use the RTI Module to generate Operating system tick interrupts with tunable period. Three compare interrupts are used to generate the required Operating system tick and the corresponding task is carried out. To visualize the task call, individual I/O pins are toggled in the example code. This application report shows how to configure three different interrupts to trigger tasks with different timebases.

### 2.1 Flow Diagram

Figure 3 shows how the RTI Module triggers tasks with different timebases using interrupts.

- 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.

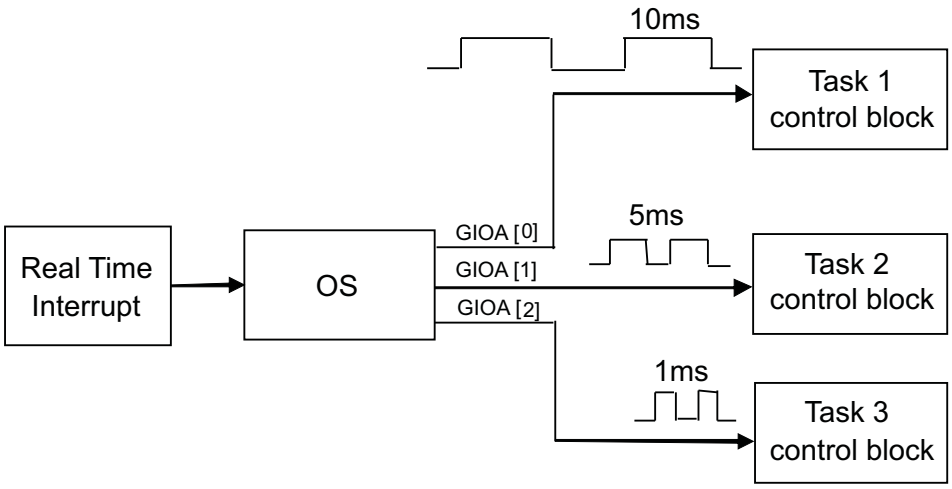


**Figure 3. Operating System Tick Generation Block Diagram**

## 2.2 Frequency and Tcount Calculation

The RTICLK to the counter block can be programmed using the RTICLKSRC register in the system module according to the application requirements. If RTICLK source is anything other than VCLK, then it needs to be at least three times slower than the VCLK. This can be achieved by configuring the RTIxDIV bits in RTICLKSRC register. Figure 4 shows RTICLK selection and pre-scaling using RTICLKSRC register and RTICPUCx register.

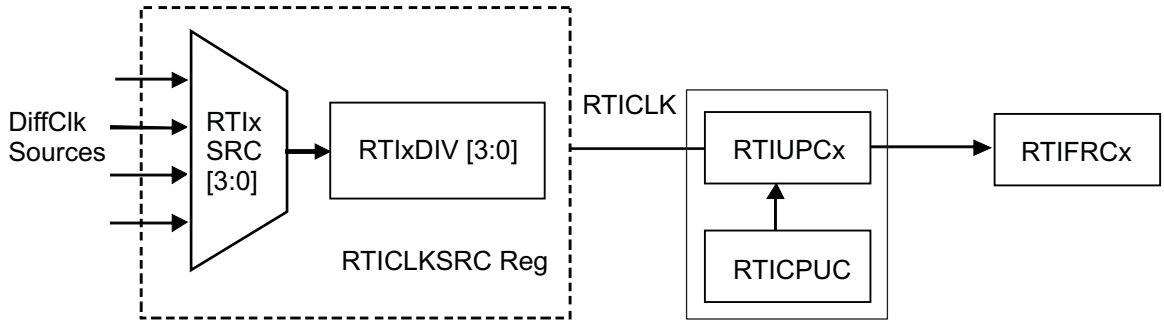


**Figure 4. RTI Clock Source Selection and Pre-Scaling**

In the following example, the RTICLK is pre-scaledby 2 using the RTICPUCx Register. For more information, see Section 1.3.

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

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

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

## 2.3 Compare Count Value Calculation

The Count values for the Compare register to generate Periodic Operating system tick is calculated with the knowledge of Tcount Period. The RTICLK is pre-scaled by the UP Counter (x) Register and fed to the respective Free Running (x) counter.

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

In the following example, interrupts (INT0, INT1, and INT2) are used for the generation for period system tick.

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

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

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

## 2.4 Operating System Tick Output Waveform

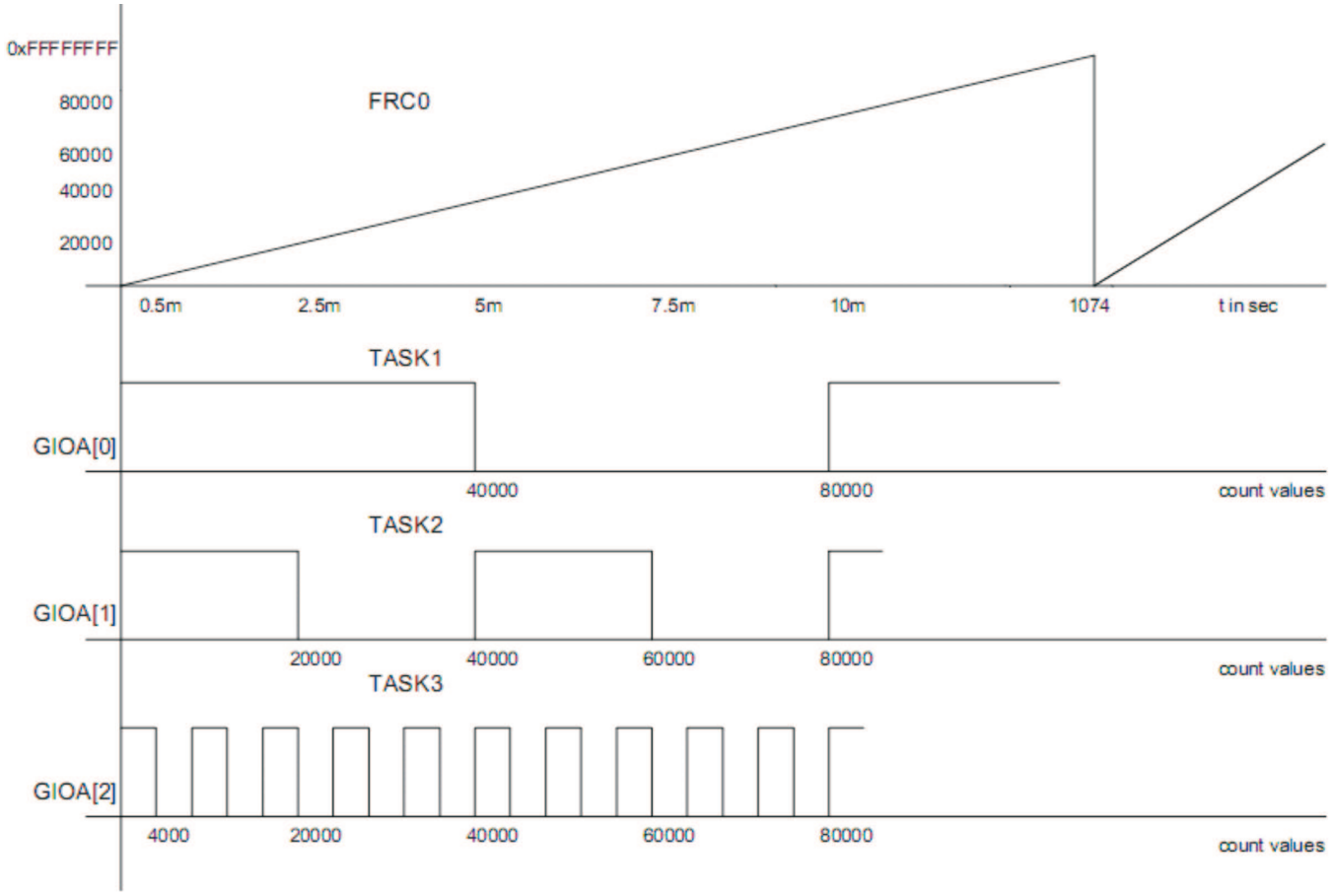


**Figure 5. Operating System Tick Output Waveform**

## 2.5 Configuration for Generating Operating System Tick

- Select VCLK using RTI clock source using RTICLKSRC register in system module.
- Select the following Clock Configurations.
  - $HCLK = OSCIN$
  - $VCLK = \frac{HCLK}{2}$
  - $RTICLK = VCLK$ (4)
- Configure RTICOMP(x) register and the respective Update RTIUDCP(x) Register to initialize with the Compare (x) count Values.
- Configure the GIO Port A as output port.
- Configure RTI module Interrupt i.e. RTISETINT and Status Register i.e. RTIINTFLAG.
  - Pending interrupts are cleared.
  - Compare 0, Compare 1 and Compare 2 interrupt are enabled.
- Configure Compare 0 Interrupt ISR to do the following:
  - Comp0 interrupt flag is cleared by writing 1 to INT0 bit in RTIINTFLAG register.
  - Toggle GIO Port A – Pin 0
- Configure Compare 1 Interrupt ISR to do the following:
  - Comp1 interrupt flag is cleared by writing 1 to INT1 bit in RTIINTFLAG register.
  - Toggle GIO Port A – Pin 1
- Configure Compare 2 Interrupt ISR to do the following:
  - Comp2 interrupt flag is cleared by writing 1 to INT2 bit in RTIINTFLAG register.
  - Toggle GIO Port A – Pin 2

## 2.6 Software Flow

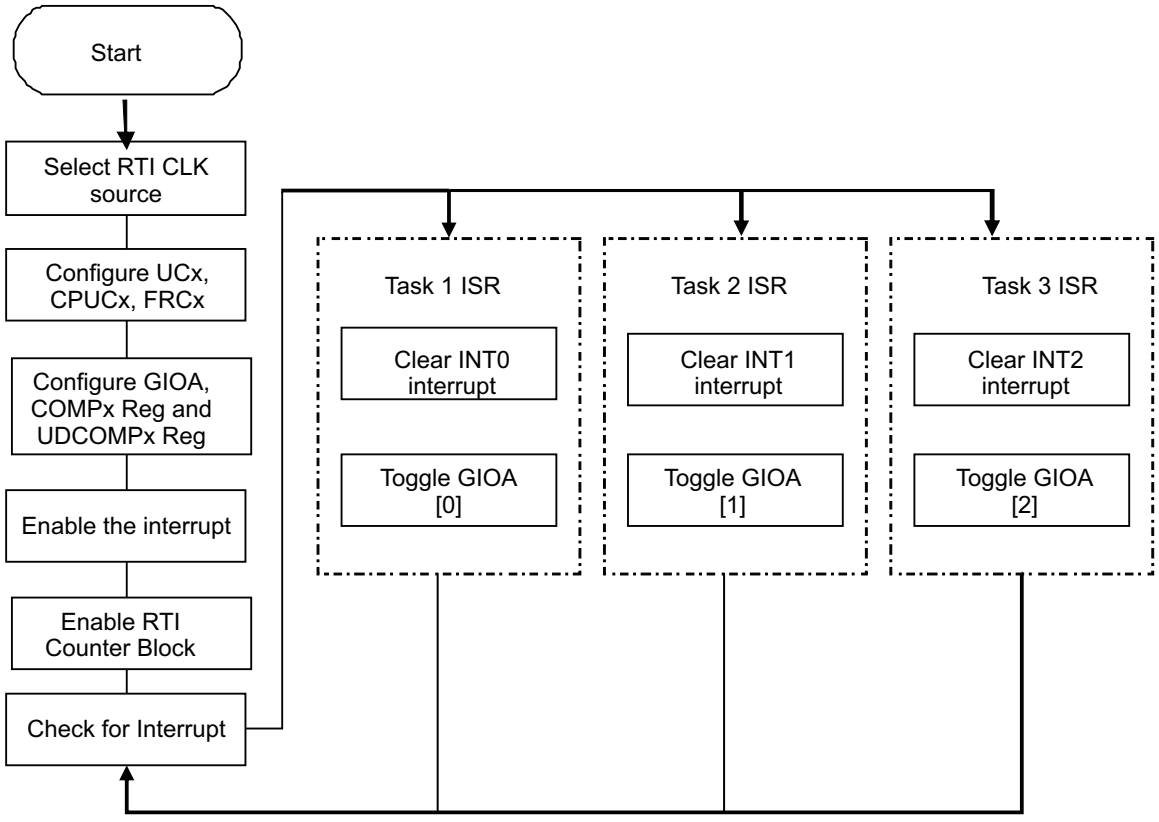


**Figure 6. Software Flow Diagram**

## 2.7 Captured Output Waveform

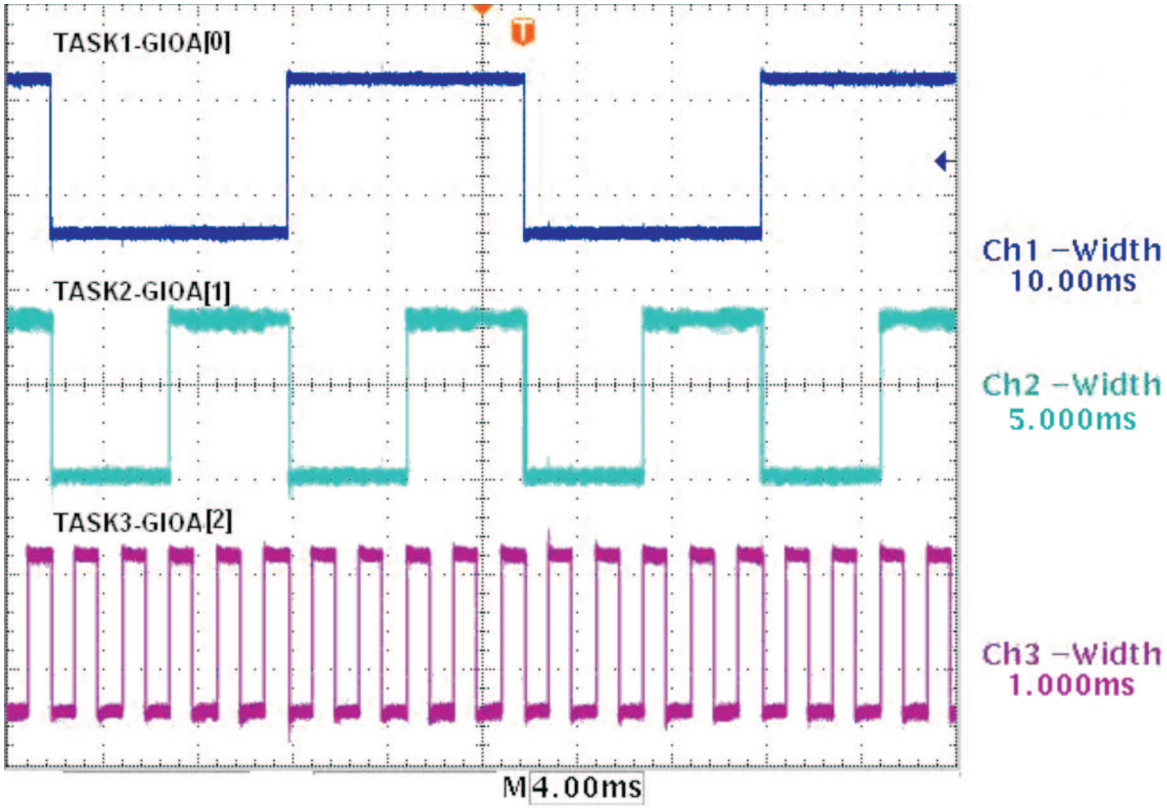


**Figure 7. Output Waveform**

## Appendix A Software Listing

```
/*****************************************************************************************
***/
/* 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*******************************************
**/
```

# IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265