# Draft

# **Zero-Latency Interrupts and TI-RTOS on C2000 Devices**

**Todd Mullanix TI-RTOS Apps Manager** Feb, 11, 2018





# Agenda

Here's the high-level view of what we will cover in this presentation:

- Typical kernel interrupt 1.
- What is a zero-latency interrupt? 2.
- 3. When to use a zero-latency interrupt
- 4. Steps to add a zero-latency interrupt into a TI-RTOS based F28379D example.



### **TEXAS INSTRUMENTS**

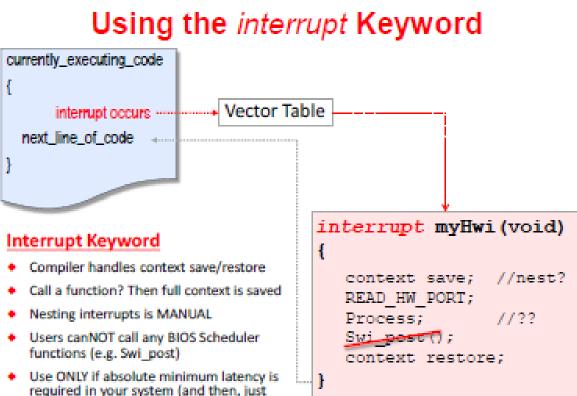
# Managed Interrupts in TI-RTOS Kernel

When the TI-RTOS kernel (aka SYS/BIOS or sometimes just BIOS) manages an interrupt, it is executed via the Hwi Dispatcher as opposed to using the *interrupt* keyword.

### Using the BIOS Hwi Dispatcher currently\_executing\_code Vector Table Hwi Dispatcher: interrupt occurs Context Save next line of code Context Restore 'Smart" Return **BIOS Hwi Dispatcher** void myHwi(arg) Easy to use, simple, RECOMMENDED Turned ON for every BIOS Hwi READ HW PORT; Swi post(); Slight increase in latency due to full context save/restore Allows BIOS Scheduler function calls Saves code space (all INTs share common) save/restore routine)

Performs "smart return" – returns to highest priority pending thread

maybe)





Texas Instruments

# What is a Zero-Latency Interrupt?

You can have an interrupt not be managed by the kernel. The interrupt will run independently of the kernel. We call this a "zero-latency interrupt"\*.

There are four key points about zero-latency interrupts.

- The **kernel adds no overhead** when the interrupt runs (thus the name zero-latency).
- The kernel will **never disable** a zero-latency interrupt. 2.
- The zero-latency interrupt cannot make any calls into the kernel that would impact the 3. **scheduler** (.e.g. Semaphore post() is not allowed in a zero-latency interrupt).
- There is a **slightly negative performance impact** on the kernel with C28 devices when there 4. is a zero-latency interrupt in the system. This is because the kernel has to use the mask when disabling/enabling the interrupts it manages.

\* Some people prefer the name "unmanaged interrupt"



# When to use a Zero-Latency Interrupt?

When the TI-RTOS kernel manages an interrupt, an overhead is incurred. The overhead is detailed in the SYS/BIOS Release Notes->Sizing and Timing Benchmarks->Target. For example, to the right shows some of the timing benchmarks for a TMS320F280049M device (with hard FP). Note: the SYS/BIOS User Guide discusses these benchmarks in more details.

There are two key areas to consider about interrupt timing

- Maximum duration the kernel ever disables interrupts: This value is 1. denoted by the "Interrupt Latency" benchmark. During this interval, an asserted interrupt will not run. If this value might have a negative effect on one of your interrupts, you should consider making it a zero-latency interrupt.
- **Overhead incurred by having the kernel manage the interrupt**: The 2. kernel adds some overhead to the running of an ISR. This value is denoted by the "Hwi dispatcher" value. It is mostly comprised on the "Hwi dispatcher" prolog" and "Hwi dispatcher epilog". If this value might have a negative effect on one of your interrupts, you should consider making it a zero-latency interrupt.

Please note: you timing numbers may vary based on compiler settings, flash wait-states, etc.

### C28x with hard FP Timing Benchmarks

Tool Chain Version: 16.9.1

BIOS Version: bios 6 52 00 11 eng

XDCTools Version: xdctools 3 50 03 33 core

Benchmark Interrupt Latency Hwi restore() Hwi disable()

Hwi dispatcher prol

Hwi dispatcher epilo

Hwi dispatcher

Hardware Interrupt

Hardware Interrupt



### Target Platform: ti.platforms.tms320x28:TMS320F280049M:1

	Cycles
	153
	19
	13
og	210
og	155
	366
to Blocked Task	588
to Software Interrupt	420

5

## Lab

For this presentation we are going to first play with two different examples on the F28379D controlCARD (or LaunchPad).

- 1. ControlSuite's cpu\_timer Example: Non-RTOS based application that configures and runs 3 timers.
- 2. SYS/BIOS' Task Mutex Example: Simple application where the kernel has a couple tasks accessing a shared resource. Each task sleeps for a bit and then tries to get the resource via a semaphore.

The goal is to add another timer into the Task Mutex Example. The timer will run at 30us, so we are going to make it a zero-latency interrupt. We'll freely steal some of the code from the cpu\_timer example to accomplish this.



# Import and Build cpu\_timers example

In CCS, **import** the cpu\_timer project in controlSUITE. controlSUITE\device support\F2837xD\v210\F2837xD examples Cpu1\cpu timers

If you are using the F28379D LaunchPad (instead of the controlCard), please add the following compiler predefined symbol LAUNCHXL F28379D

The LaunchPad has a slower external oscillator than the controlCARD.

### Build the project.

This project configures 3 timers to run every second and increment a unique counter.

Automatically import ref

✓ ₽ cpu timers cpu01

- > Sinaries
- Includes
- > 🗁 CPU1 RAM
- > b targetConfigs
- kg cpu\_timers\_cpu01.c

- F2837xD Gpio.c
- > 🗟 F2837xD lpc.c
- F2837xD\_PieCtrl.c
- F2837xD PieVect.c
- F2837xD\_SysCtrl.c
- F2837xD usDelay.asm



💱 Import CCS Eclipse Proj	ects							
Select CCS Projects to I Select a directory to search	<b>mport</b> n for existing CCS Eclipse projects.							
<ul> <li>Select search-directory:</li> <li>Select archive file:</li> </ul>	C:\ti\controlSUITE3_4_7\device_supp	port\F2837xD\v210\F	Browse Browse					
Discovered projects:								
✓ <sup>©</sup> cpu_timers_cpu01	[C:\ti\controISUITE3_4_7\device_sup	port\F2837xD\v210\I	Select All Deselect All Refresh					
Automatically import referenced projects found in same search-directory Copy projects into workspace Open <u>Resource Explorer</u> to browse a wide selection of example projects								
?		Finish	Cancel					

# F2837xD CodeStartBranch.asm F2837xD CpuTimers.c F2837xD DefaultISR.c F2837xD GlobalVariableDefs.c

7

### **Run cpu\_timers example**

Load and run the cpu\_timers .out file on CPU1. After N number of seconds, pause the core and look at the CpuTimer0, CpuTimer1, and CpuTimer2 variables. You'll see that their InterruptCount corresponds to the number of seconds you have ran the application (in this case 5 seconds).

(x)= Variables في Expression	ns 🖾 🕮 Registers		
Expression	Туре	Value	
🕆 🥭 CpuTimer0	struct CPUTIMER_VARS	{RegsAddr=0x00	
> 🔹 RegsAddr	struct CPUTIMER_REGS *	0x00000C00 {TIN	
⇔ InterruptCount	unsigned long	5	
⇔ CPUFreqInMHz	float	200.0	
⇔ PeriodInUSec	float	100000.0	
🕆 🥭 CpuTimer1	struct CPUTIMER_VARS	{RegsAddr=0x0	
> 🔹 RegsAddr	struct CPUTIMER_REGS *	0x00000C08 {TIN	
⇔ InterruptCount	unsigned long	5	
⇔ CPUFreqInMHz	float	200.0	
⇔ PeriodInUSec	float	100000.0	
🕆 🍃 CpuTimer2	atmust CDUITINED MADE	{RegsAddr=0x00	
• 🖉 Cputimerz	struct CPUTIMER_VARS	{RegsAddr=0x0	
> • RegsAddr	struct CPUTIMER_VARS		
	_	{RegsAddr=0x00 0x00000C10 {TI 5	
> ➡ RegsAddr	struct CPUTIMER_REGS *	0x00000C10 {TI	



# **Build and Run SYS/BIOS Mutex Example**

In CCS, **import** Task Mutex example via TI Resource Explorer Classic.

Build and load the project onto CPU1. ✓ <sup>™</sup> task\_TMS320F28379D > <sup>™</sup> Binaries > <sup>™</sup> Includes > <sup>™</sup> Debug > <sup>™</sup> mutex.c > <sup>™</sup> TMS320F28379D.cmd > <sup>™</sup> makefile.defs

When you **run** the example, you'll see tasks ping-pong back and forth and then terminate.

🗳 Console 🛛	
F283779D.ccx	ml:CIO
Running ta	sk2 function
Running ta	sk1 function
Running ta	sk2 function
Running ta	sk1 function
Running ta	sk2 function
Running ta	sk1 function
Running ta	sk2 function
Running ta	sk1 function
Running ta	sk2 function
Sem blocke	d in task2
Calling BI	OS_exit from task2

mutex.cfg



# SYS/BIOS Users Guide Generic Devices 28004x Piccolo 2801x Fixed Point 2802x Piccolo 2803x Piccolo

- > III 2805x Piccolo
- > III 2806x Piccolo
- > IIII 2807x Piccolo
- > 10 280x Fixed Point
- > IV 281x Fixed Point
- > IV 2823x Fixed Point
- > III 2833x Delfino
- > IIII 2834x Delfino
- 🗸 🕅 2837xD Delfino
- > # TMS320F28374D
- > # TMS320F28375D
- > IMS320F28376D
- > 🕅 TMS320F28377D
- > # TMS320F28378D
- TMS320F28379D
- 👻 🔮 TI Target Examples
  - 🗟 Minimal
  - 🖶 Typical
  - > 12 28x Specific Examples
  - ✓ ♥ Generic Examples
    - 🗟 Benchmark Example
    - ☐ C++ Example (bigtime)
    - 🗟 Clock Example
    - 🗟 Error Example
    - 🗟 Event Example
    - 🗟 Hello Example
    - Example
    - 🖶 Memory Example
    - 🗟 Small Example
    - 🗟 Static Example
    - 🗟 Swi Example
    - Task Mutex Example

9

# **SYS/BIOS Mutex Example Overview**

### Here's a look at the pseudo-code for the two tasks

```
Task1 // (lower priority)
   while (1) {
      Semaphore pend(sem, BIOS_WAIT_FOREVER)
      //simulate doing real work ...
      resource++;
      Semaphore post (sem)
      Task sleep(10)
```

```
Task2 // (higher priority)
   while (1) {
      Semaphore pend (sem, BIOS WAIT FOREVER)
      //simulate doing real work ...
      resource++;
      Semaphore post (sem)
      Task sleep(10)
      finishCount++
      if (finishCount == 5) {
         BIOS exit(0);
```

The semaphore makes sure the higher priority task does not preempt the lower priority task when it is updating resource. Let's look a little more how the Task sleep() works...



### **TEXAS INSTRUMENTS**

# **Timing in SYS/BIOS**

The Clock module by default grabs a timer to use for the Clock instances and for driving timing mechanism like Task sleep and Semaphore pend (with a timeout).

Here's the default configuration (from mutex.cfg in the Task Mutex example)



As you can see, ANY is specified. This means the kernel will grab a timer that is **not already used** in the .cfg file. For this lab, will fix this to a specific timer since we want to avoid collisions.

Also note the **default period is 1000us** (1ms). The kernel will do book-keeping during this interrupt (e.g. wake-up tasks whose Task\_sleep has expired, call Clock functions that are due to run, etc.)

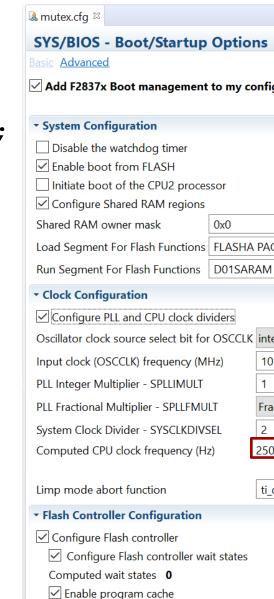
**Task\_sleep's argument is in ticks**. So Task\_sleep(10) means to sleep for 10 ticks, or when the Clock's period is 1ms, sleep for 10ms.



# **CPU Speed in SYS/BIOS**

The .cfg file can be used to configure the CPU speed on the C28
devices. Here is the graphical view of the Boot module (after
var Boot =
xdc.useModule('ti.catalog.c2800.initF2837x.Boot');
was added into the .cfg).

By default, the kernel sets the CPU speed to 2.5MHz. This can be changed by setting the clock source, multiplers, etc., but for this lab **we'll leave it at 2.5MHz**. We'll need this when configuring the new zero-latency timer.



Enable data cache



S
nfiguration
PAGE = 0
M PAGE = 0
nternal oscillator 2 (default on reset) $$
10
1
Fractional multiplier is 0 🛛 🗸
2
500000
ti_catalog_c2800_initF2837x_Boot_defaultLimpAbortFunction

### èxas Instruments

## **Task Mutex Example Enhancment: Overview**

Let's say we want to add something that needs to be checked every 30us. Here are three options:

- 1. Make the SYS/BIOS Clock module's timer run at 30us.
- 2. Use a different timer, via the SYS/BIOS API Timer create(), that runs every 30us.
- 3. Use a different timer that is not managed by the kernel (aka zero-latency interrupt).

Option 1 is a bad idea because the kernel does lots of book-keeping on a Clock tick. 30us is simply too fast of a period.

Option 2 might work. It takes a minimum of 366 cycles to run an empty Hwi (refer to the SYS/BIOS) Release Notes for timing benchmarks). Let's say we bump this up to 400 to have it actually do something. So if you are running faster than 13.3MHz it will work, but you are using lots of cycles.

CPU Speed	uS/cycle	# cycles/interrupt	uS required for each 30us Interrupt
2.5MHz	.4	400	160
13.3MHz	.07519	400	30
50MHz	.02	400	8
200MHz	.005	400	2

That do option 3! Let's look to see how to add a timer (in this case Timer1) as a zero-latency interrupt...



13

# **Task Mutex Example Enhancment: Configuration File Changes**

First let's make two changes to the kernel configuration file (.cfg). We'll edit the mutex.cfg as a text file (instead of graphically).

1. The default is to let the kernel select a non-used timer. Since it does not know about the zero-latency timer at build time, we will specify Timer2 as the timer to be used by the **kernel**. This is to avoid any conflict with the zero-latency timer interrupt we are going to create. Please add this to the bottom of the .cfg file.

```
Clock.timerId = 2;
```

2. Tell the kernel that **interrupt 13 (Timer1) will be a zero-latency interrupt**. Please add this to the bottom of the .cfg file.

Hwi.zeroLatencyIERMask = 0x1000; // note: the 13<sup>th</sup> bit is set





14

### **Task Mutex Example Enhancment: Plugging Interrupt**

During runtime, we need to **plug in the ISR**. Please add the following **bolded** code to the 3. mutex.c file. The next slides will add the variables and necessary functions.

```
#include <ti/sysbios/family/c28/Hwi.h>
```

```
interrupt void cpu timer1 isr(void)
    CpuTimer1.InterruptCount++;
Int main()
    ...
    tsk2 = Task create (task2, &taskParams, NULL);
    /* Plug in the zero-latency interrupt */
    Hwi plug(13, cpu timer1 isr);
   BIOS start(); /* does not return */
```



**TEXAS INSTRUMENTS** 

# **Task Mutex Example Enhancment: Plugging Interrupt**

4. Now we need to **configure the timer and enable the interrupt**. Please add the following **bolded** code.

```
Int main()
   ...
   tsk2 = Task create (task2, &taskParams, NULL);
   /* Plug in the zero-latency interrupt */
   Hwi plug(13, cpu timer1 isr);
   myInitCpuTimers();
   ConfigCpuTimer(&CpuTimer1, 2.5, 30); // CPU is 2.5MHz, timer is 30us
   CpuTimer1Regs.TCR.all = 0x4000;
   Hwi enableIER(0x1000);
```

BIOS start(); /\* does not return \*/



**TEXAS INSTRUMENTS** 

### **Task Mutex Example Enhancment: Initialize Timer1**

5. Please add the following timer initialization code into clock.c (somewhere above main()). This is modified version of the function in the cpu\_timers project. We only need to initialize Timer1.

```
#include "F28x Project.h"
void myInitCpuTimers(void)
    CpuTimer1.RegsAddr = &CpuTimer1Regs;
    CpuTimer1Regs.PRD.all = 0xFFFFFFF;
    CpuTimer1Regs.TPR.all = 0;
    CpuTimer1Regs.TPRH.all = 0;
    CpuTimer1Regs.TCR.bit.TSS = 1;
    CpuTimer1Regs.TCR.bit.TRB = 1;
    CpuTimer1.InterruptCount = 0;
```



**TEXAS INSTRUMENTS** 

# Task Mutex Example Enhancment: Steal code from cpu\_timers

- Copy and paste these two files from the cpu\_timers project and add them into the Task 6. Mutex project
  - F2837xD\_CpuTimers.c —
  - F2837xD\_GlobalVariableDefs.c —

We'll use global variables and functions from these files instead of redoing them.

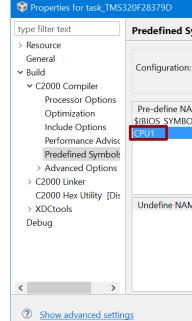
- Let's add a linker file from controlSUITE that supports SYS/BIOS. Please add the 7. controlSUITE\device\_support\F2837xD\v210\F2837xD\_headers\cmd\F2837xD\_Headers\_BIOS\_cpu1.cmd file
  - task TMS320F28379D
    - > Sinaries
    - Includes
    - > >> Debug
    - > 🗟 F2837xD\_CpuTimers.c
    - F2837xD GlobalVariableDefs.c
    - > Image: F2837xD Headers BIOS cpu1.cmd
    - > in mutex.c
    - TMS320F28379D.cmd
    - > 🗡 src
      - la makefile.defs
      - Image: Market Market



Texas Instruments

# **Task Mutex Example Enhancment: Linker File**

The controlSUITE files depend on a **CPU1 define**. So add it 8. into the project.



Finally, let's add includes paths for controlSUITE. Namely (this is assuming you have a 9. linked resource for controlSUITE in CCS). "\${INSTALLROOT\_F2837XD}/F2837xD headers/include" "\${INSTALLROOT\_F2837XD}/F2837xD\_common/include"

Properties for task_TMS3	20F28379D	- 🗆 X
type filter text	Include Options	↓ ↓ ↓ ↓
<ul> <li>&gt; Resource</li> <li>General</li> <li>&gt; Build</li> <li>&gt; C2000 Compiler</li> <li>Processor Options</li> <li>Optimization</li> <li>Include Options</li> <li>Performance Advisc</li> <li>Predefined Symbols</li> <li>&gt; Advanced Options</li> <li>&gt; C2000 Linker</li> <li>C2000 Hex Utility [Dis</li> <li>&gt; XDCtools</li> <li>Debug</li> </ul>	Configuration: Debug [Active] Add dir to #include search path (include_path, -I) \$(BIOS_INCLUDE_PATH) == \$(PROJECT_ROOT) == \$(CS_TOOL_ROOT)/== \$(CS_TOOL_ROOT)/F2837XD_headers/include" == "\$(INSTALLROOT_F2837XD)/F2837xD_headers/include" == "\$(INSTALLROOT_F2837XD)/F2837xD_common/include" == \$(Specify a preinclude file (preinclude)	<ul> <li>Manage Configurations</li> <li>କା ଲା ଲା ନା ନା</li> </ul>
Show advanced setting	22	OK Cancel



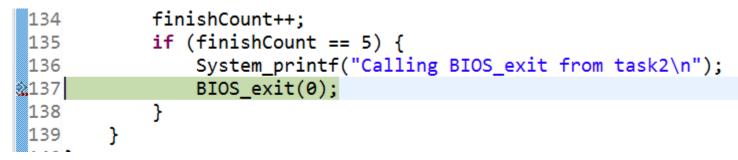
		—		×
l Symbols			⇔ ▼	⇒ ▼ ▼
on: Debug [Active]	~	Manage	e Configura	tions
NAME (define, -D) IBOLS}			le	🗟 🖗 🏷
19613)				
IAME (undefine, -U)			<b>i</b>	월 상I 상I
	OK		Cance	1
	2.1			

# Task Mutex Example Enhancment: Build/Load/RUN!

### 10. Let's **build the project**. You'll probably get a warning We have an open bug report on this. It can be ignored.

"C:/ti/controlSUITE3\_4\_7/device\_support/F2837xD/v210/F2837xD\_headers/include/F2837xD\_device.h", line 246: warning #303-D: typedef name has already been declared (with same type) "C:/ti/controlSUITE3\_4\_7/device\_support/F2837xD/v210/F2837xD\_headers/include/F2837xD\_device.h", line 247: warning #303-D: typedef name has already been declared (with same type)

11. Now load the application, set a breakpoint at the BIOS exit() call in task2, and run.



12. If you look at the tick count (ROV->Clock) and the CpuTimer.InterruptCount, they make sense! 113 ticks \* 1000us / 30us = ~3766 for InterruptCount. Since we halted after a Clock tick boundary, it's expected that InterruptCount is slightly higher than calculated.

■ RTOS Object View (ROV) 🛛				<sup>(x)=</sup> Variables	🛠 Expression	ns 🛛	<sup>IIII</sup> Registers	
<ul><li>☆ Favorites</li><li>✓ ▲ task TMS320F28379D.out</li></ul>	^		le Raw tickSource	Expression	ner1	Type struc		Value {RegsAddi
<ul> <li>✓          <sup>™</sup> Viewable Modules         <ul> <li>● BIOS</li> </ul> </li> </ul>			ti.sysbios.ki	>	Addr	struc	t CPUTIMER_REGS *	0x00000C
Boot					ruptCount FregInMHz	float	gned long	3799 2.5
Clock				⇔= Perio	dInUSec	float		30.0



20

# Advanced

- 1. You can comment out the BIOS\_exit and run the application for a longer period. Or you can even run past the breakpoint. The zero-latency interrupt will continue even though the kernel has exited.
- 2. If the zero-latency interrupt **needed to tell the kernel about something**, it could call Hwi post() on an interrupt that is managed by the kernel. That interrupt can call Semaphore post(), etc.



Texas Instruments

# **Additional Resources**

- <u>http://processors.wiki.ti.com/index.php/SYS/BIOS\_for\_the\_28x</u>
- SYS/BIOS User Guide (inside the docs directory on an install SYS/BIOS product).



**TEXAS INSTRUMENTS** 



©Copyright 2017 Texas Instruments Incorporated. All rights reserved.

This material is provided strictly "as-is," for informational purposes only, and without any warranty. Use of this material is subject to TI's Terms of Use, viewable at TI.com