Module 8. Timers: Theory and Practice

Topics

- Intro to Interrupts
- Intro to Timers

But first… what is a timer?

Most microcontrollers have timers in some form. Timers can be used to generate **interrupts** at particular intervals, generate PWM signals, measure frequency of input signals, and more! In this course, we will focus on the generation of timer interrupts, which tell the CPU that a certain amount of time has passed.

Fundamental timer counting modes

Most timers have a number of *counting modes*:
Unidirectional mode (called "Up mode" on the MSP430)
Count from 0 to a *programmer set* maximum count value (which we call MAX_CNT).

Continuous mode
Count from 0 to full count of timer (8, 12, 16 bits, etc.) For a 16 bit timer, this means:

Up/Down Mode
Counts from 0 to *programmer set* maximum count, then back down to zero

In each mode, most timers (like those on the MSP430) will trigger an interrupt when the count transitions back to 0.
Most timer peripherals have two "operating modes", which control how they use the counter:

- **Capture mode**: Records the counter value when a certain input changes

- **Compare mode**: Performs an operation when the counter value reaches a certain value

Interrupts

Interrupt: A signal sent to the CPU from a peripheral or external source

- Typically, an interrupt is either a request for the CPU to do something or a notification that the peripheral has something (ie, data) available for the CPU to use.
- The CPU can choose to accept (or to "service") the interrupt, or ignore it. Certain interrupts, called "non-maskable interrupts" (NMIs) cannot be ignored.

Interrupts on the CPU are handled by a special function called an **Interrupt Service Routine (ISR)**.

Note: Interrupts are not just for timers!
Many peripherals on the MSP430 can generate interrupts for different reasons:

What does the CPU do when it receives interrupts?

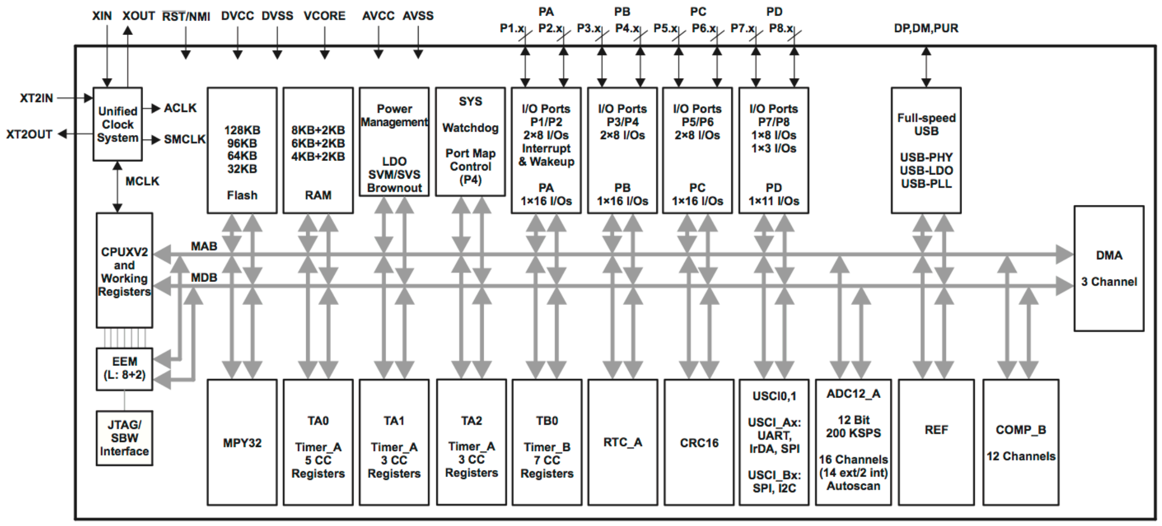
Arrival of interrupts is *asynchronous* to the program's execution.

How do interrupts work internally?

Peripherals that can trigger interrupts so do by issuing a request to the CPU's *interrupt controller*, often over a dedicated wire called an interrupt request line. The interrupt controller decides how the CPU processes each interrupt:

Timers on the MSP430

The MSP430F5529 has a number of timer peripherals, as shown in the system block diagram:


XIN
XOUT
RST/NMI
DVCC
DVSS
VCORE
AVCC
AVSS
PA
PB
PC
PD
P1.x
P2.x
P3.x
P4.x
P5.x
P6.x
P7.x
P8.x
DP,DM,PUR
XT2IN
XT2OUT
Unified Clock System
ACLK
SMCLK
MCLK
128KB 96KB 64KB 32KB Flash
8KB+2KB 6KB+2KB 4KB+2KB RAM
Power Management LDO SVM/SVS Brownout
SYS Watchdog Port Map Control (P4)
I/O Ports P1/P2 2×8 I/Os Interrupt & Wakeup PA 1×16 I/Os
I/O Ports P3/P4 2×8 I/Os PB 1×16 I/Os
I/O Ports P5/P6 2×8 I/Os PC 1×16 I/Os
I/O Ports P7/P8 1×8 I/Os 1×3 I/Os PD 1×11 I/Os
Full-speed USB USB-PHY USB-LDO USB-PLL
CPUXV2 and Working Registers
MAB
MDB
DMA 3 Channel
EEM (L: 8+2)
JTAG/ SBW Interface
MPY32
TA0 Timer_A 5 CC Registers
TA1 Timer_A 3 CC Registers
TA2 Timer_A 3 CC Registers
TB0 Timer_B 7 CC Registers
RTC_A
CRC16
USCI0,1 USCI_Ax: UART, IrDA, SPI USCI_Bx: SPI, I2C
ADC12_A 12 Bit 200 KSPS 16 Channels (14 ext/2 int) Autoscan
REF
COMP_B 12 Channels

MSP430s have two main types of timers, Timer A, and Timer B; both types function in very similar ways, but have some subtle differences. Each chip can have multiple Timer A's and B's, as shown in the block diagram.

The MSP430F5529 has the following timers:
Timer B: Has 7 capture/compare units, can generate PWM signals
Timer A0: Multiple capture compare modules, can generate PWM
Timer A1: Functionally the same as A0
Timer A2: 3 capture compare registers

Additionally, the MSP430F5529 has the following other peripherals that contain timers:

- A Basic Timer, which has some real-time clock features
- The **Watchdog timer** (WDT)
    - When the WDT is on, it must continuously have its count reset within the program
    - If the count reaches zero, it **resets the MSP430!**

Why does the WDT exist? To prevent your program from getting stuck in some kind of unrecoverable state. We don't want to deal with the WDT in your labs, which is why the first line of every program we write is:

```
WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
```

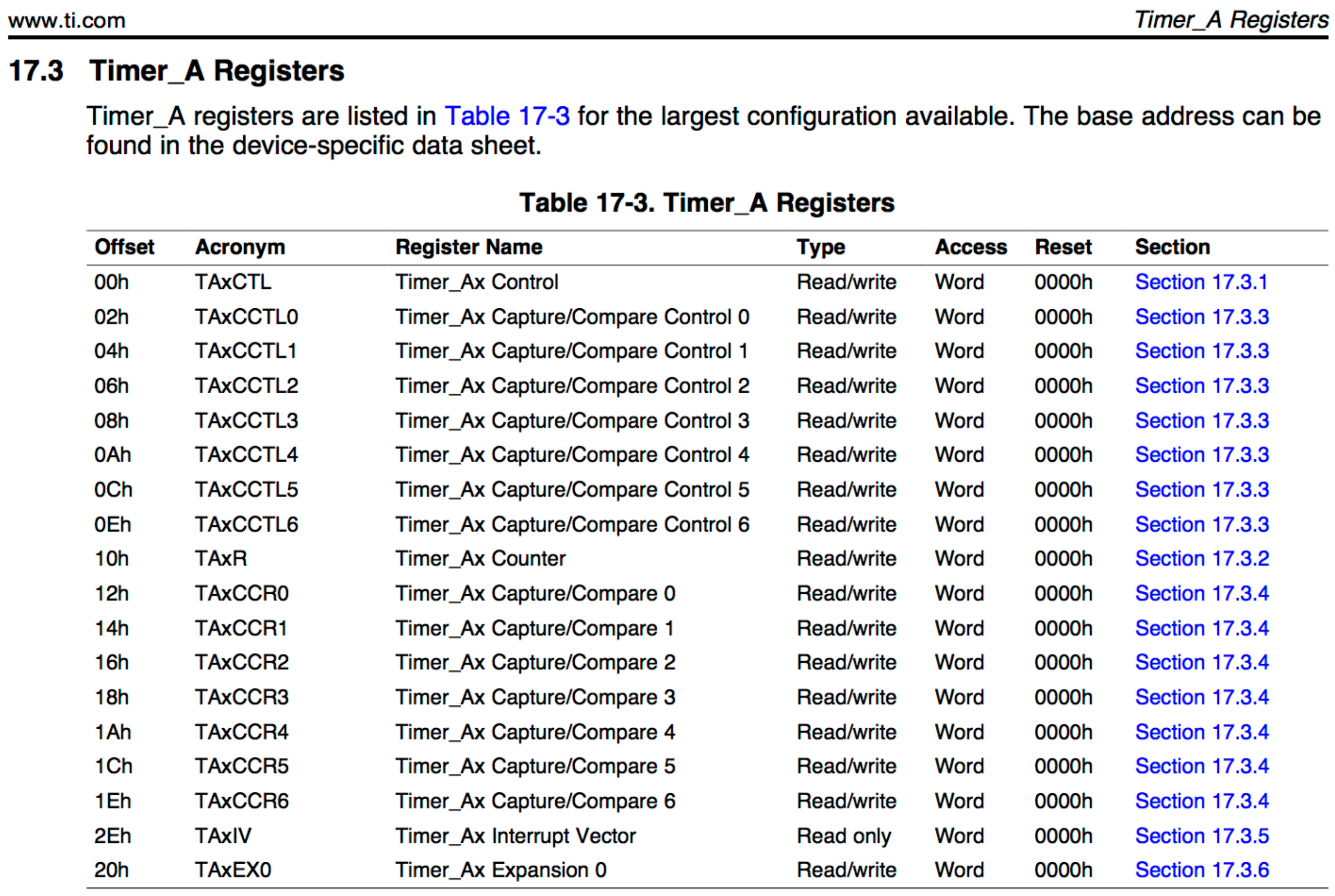
This disables the watchdog timer. If you wait too long to stop the watchdog timer, your program will reset only a few milliseconds after startup!

Configuring timers on the MSP430

Like the UCS module, timers are highly configurable!

We will stick to the basic configurations and **only add complexity when we need it**! This is a good design practice, and also makes our lives easier!

Timer A has the following registers:



www.ti.com *Timer_A Registers*

17.3 Timer_A Registers

Timer_A registers are listed in Table 17-3 for the largest configuration available. The base address can be found in the device-specific data sheet.

Table 17-3. Timer_A Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TAxCTL	Timer_Ax Control	Read/write	Word	0000h	Section 17.3.1
02h	TAxCCTL0	Timer_Ax Capture/Compare Control 0	Read/write	Word	0000h	Section 17.3.3
04h	TAxCCTL1	Timer_Ax Capture/Compare Control 1	Read/write	Word	0000h	Section 17.3.3
06h	TAxCCTL2	Timer_Ax Capture/Compare Control 2	Read/write	Word	0000h	Section 17.3.3
08h	TAxCCTL3	Timer_Ax Capture/Compare Control 3	Read/write	Word	0000h	Section 17.3.3
0Ah	TAxCCTL4	Timer_Ax Capture/Compare Control 4	Read/write	Word	0000h	Section 17.3.3
0Ch	TAxCCTL5	Timer_Ax Capture/Compare Control 5	Read/write	Word	0000h	Section 17.3.3
0Eh	TAxCCTL6	Timer_Ax Capture/Compare Control 6	Read/write	Word	0000h	Section 17.3.3
10h	TAxR	Timer_Ax Counter	Read/write	Word	0000h	Section 17.3.2
12h	TAxCCR0	Timer_Ax Capture/Compare 0	Read/write	Word	0000h	Section 17.3.4
14h	TAxCCR1	Timer_Ax Capture/Compare 1	Read/write	Word	0000h	Section 17.3.4
16h	TAxCCR2	Timer_Ax Capture/Compare 2	Read/write	Word	0000h	Section 17.3.4
18h	TAxCCR3	Timer_Ax Capture/Compare 3	Read/write	Word	0000h	Section 17.3.4
1Ah	TAxCCR4	Timer_Ax Capture/Compare 4	Read/write	Word	0000h	Section 17.3.4
1Ch	TAxCCR5	Timer_Ax Capture/Compare 5	Read/write	Word	0000h	Section 17.3.4
1Eh	TAxCCR6	Timer_Ax Capture/Compare 6	Read/write	Word	0000h	Section 17.3.4
2Eh	TAxIV	Timer_Ax Interrupt Vector	Read only	Word	0000h	Section 17.3.5
20h	TAxEX0	Timer_Ax Expansion 0	Read/write	Word	0000h	Section 17.3.6

We will use a subset of these registers:

- **TA2CTL**: Control register for Timer A2
- **TA2CCTLx**: Control register for a capture/compare block
- **TA2CCRx**: Capture/compare register (data register)

We will discuss how to use these registers in detail using an example.

Timer configuration example: A stopwatch

Example: Implement a stopwatch that measures seconds and hundredths of seconds on our development board.

First, how do we measure the passage of 0.01 seconds? By counting clock ticks. We will do this by configuring a timer for the job. But how do we start?

In our labs, we can break any problem involving timers into a set of steps:

1. Select a timer to use: How about Timer A2?

2. Map desired behavior to an operating mode (Up, Continuous, Up/Down)

3. Select a clock source and configure registers appropriately

How do we configure the TimerA2 control registers?

Using this information, we can write the register configuration:

Parameters we know	Relevant register field

We can use this to write:

```
TA2CTL =


TA2CCR0 =


TA2CCTL0 =
```

Step 4: Write Interrupt Service Routine (ISR) and enable interrupts

… how do we write interrupts in our code, anyway?

An ISR for Timer A2 looks like this:

```
// Example syntax for TimerA2 ISR
#pragma vector=TIMER2_A0_VECTOR
__interrupt void TIMER_A2_ISR(void)
{
    // Do something
    // ...
}
```

In addition, in your `main()` you must **enable interrupts** to tell the CPU to handle them:

```
// Using pre-defined macros in msp430.h

_BIS_SR(GIE);          // Global interrupt enable
// ... OR ...
__enable_interrupt();
```

(The above macros are equivalent. You will see both of them in example code and notes in this class.)

Back to the example: what does it mean when we get an interrupt from Timer A2?
What should the ISR do?

Each interrupt means that the timer has reached MAX_CNT, meaning that 328 ticks of ACLK ~= 0.01s have elapsed.

Thus, the ISR should count how many interrupts have occurred… and do nothing else:

```
// Global count of clock ticks
unsigned long int timer = 0;

#pragma vector=TIMER2_A0_VECTOR
__interrupt void TIMER_A2_ISR(void)
{



}
```

Important note: ALWAYS keep your ISRs short. Why?
What happens if your ISR hasn't completed before the next ISR arrives?

In general, NEVER do any of the following in an ISR:

- Write to the display
- Flush the display
- Do floating point math
- Call expensive functions like `sin()` or `sprintf()`

Examples: Using the timer variable

Stopwatch

Now that our ISR is properly configured, what does the variable timer represent? How do you use it to actually display the time?

The timer represents the number of 0.01 second intervals that have elapsed since Timer A2 was started. To use it, we need to convert this to minutes and seconds in order to display it.

How do we do this? Note that we want to do it using *integer math*, since floating point is slow and we eventually want to put this information on the display.

Timer accuracy

How accurate will our stopwatch be? Is that accuracy acceptable?
The duration of one ACLK tick = 1/32768 Hz = 3.05e-5 seconds

Since our stopwatch will run slow, how long until it is off by 0.01 second?

Here is how we can add a leap count for this example:

```
#pragma vector=TIMER2_A0_VECTOR
__interrupt void TIMER_A2_ISR(void)
{









}
```

When using leap counting, we can use the following general rule:

In our example, is our leap counting solution perfect? For how long will it be accurate to 0.01 sec?

Configuration example: 1ms resolution

What if we wanted our stopwatch to have 1ms resolution? What would need to change? We would need to change the value of MAX_CNT. Would we need leap counting?

Configuration example

Configure Timer A2 for 0.5 sec resolution. Do you need to use leap counting?

Another configuration example

What if you wanted 0.0001 second resolution? What do we do now?