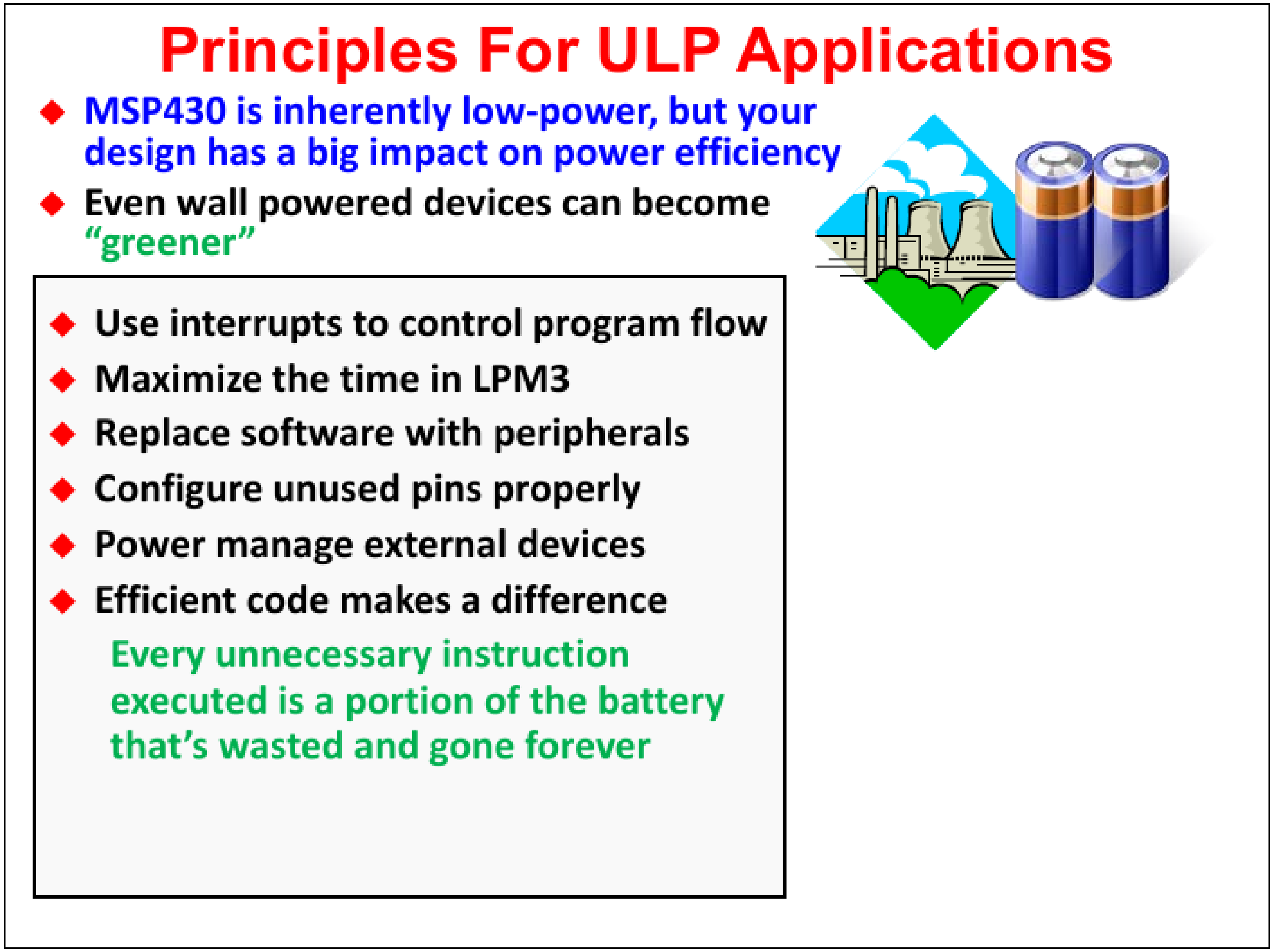
Low Power Concepts



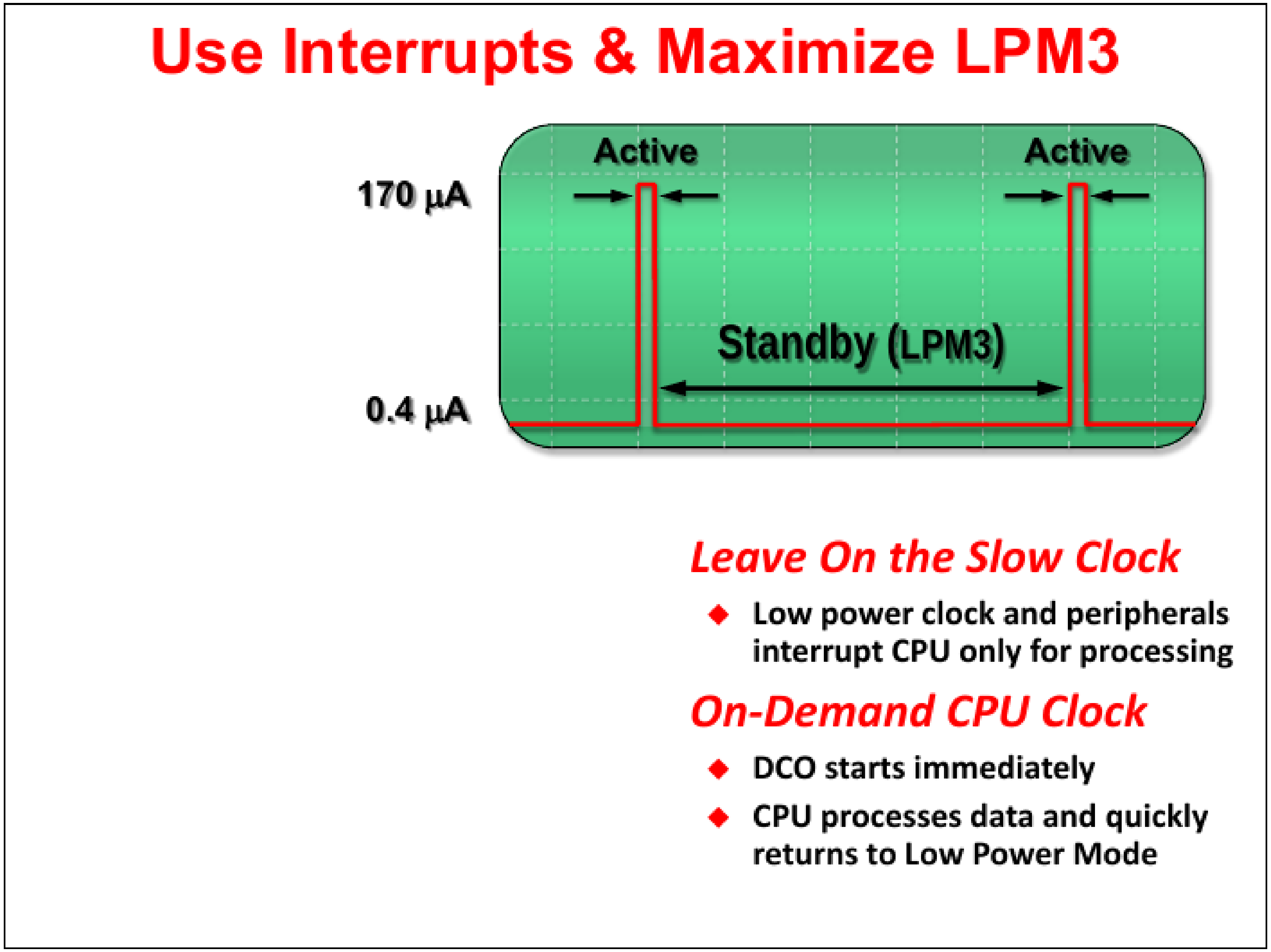
Principles For ULP Applications

- MSP430 is inherently low-power, but your design has a big impact on power efficiency
- Even wall powered devices can become "greener"

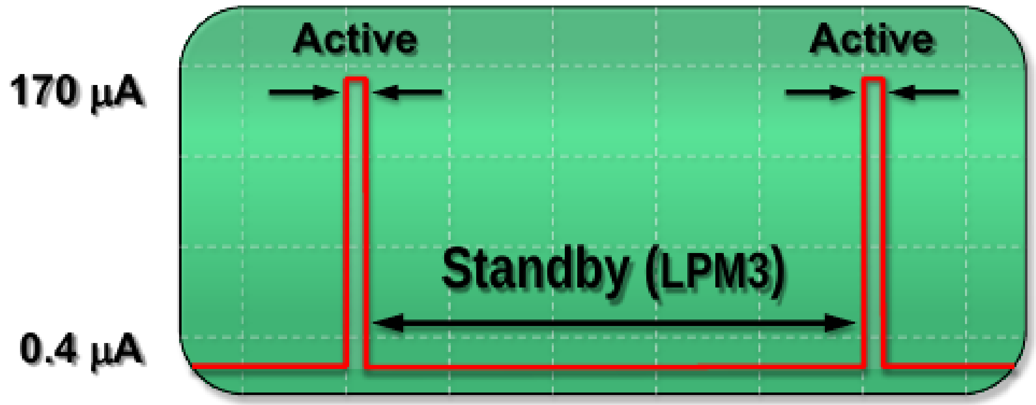
- Use interrupts to control program flow
- Maximize the time in LPM3
- Replace software with peripherals
- Configure unused pins properly
- Power manage external devices
- Efficient code makes a difference

  Every unnecessary instruction executed is a portion of the battery that's wasted and gone forever

Use Interrupts and Low-Power Modes



Use Interrupts & Maximize LPM3



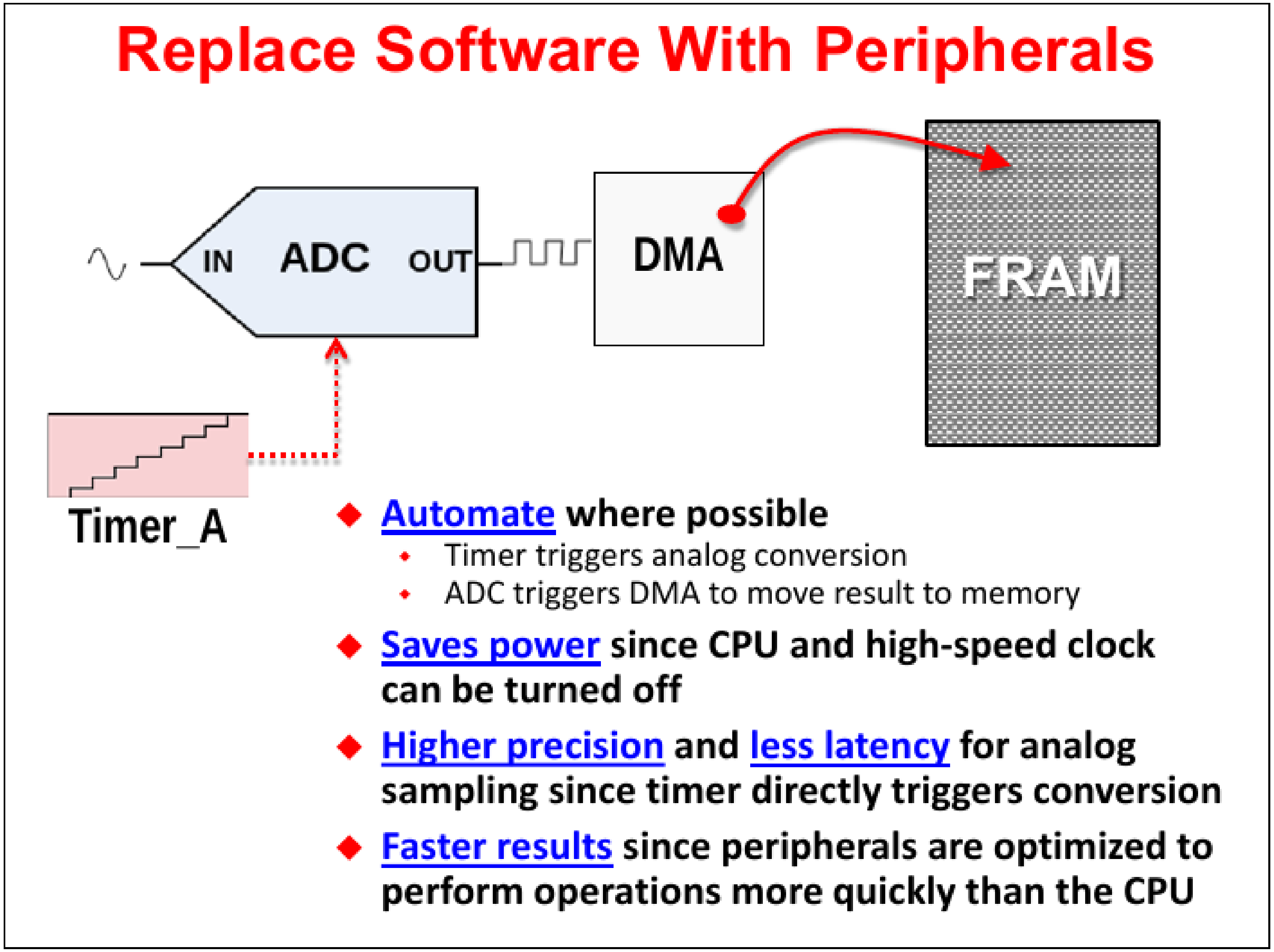
Leave On the Slow Clock

- Low power clock and peripherals interrupt CPU only for processing

On-Demand CPU Clock

- DCO starts immediately
- CPU processes data and quickly returns to Low Power Mode

Replace Software with Peripherals



Replace Software With Peripherals

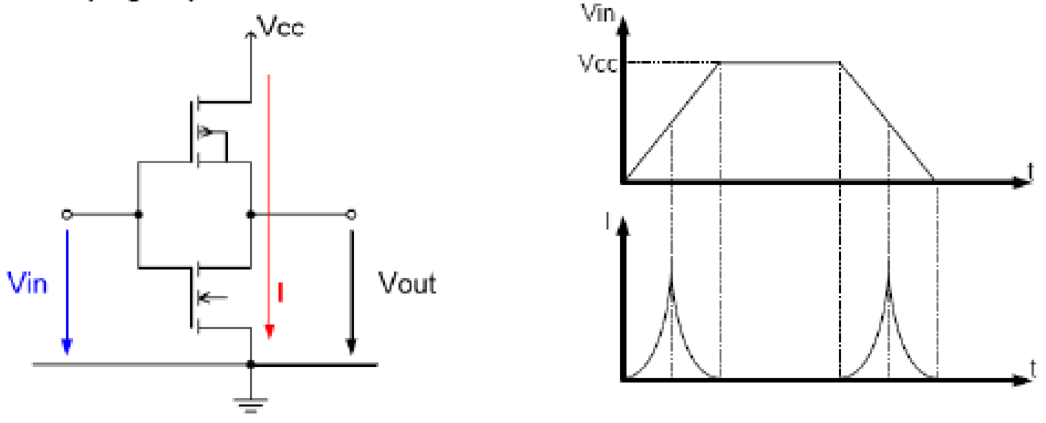
- **Automate where possible**
  - Timer triggers analog conversion
  - ADC triggers DMA to move result to memory
- **Saves power since CPU and high-speed clock can be turned off**
- **Higher precision and less latency for analog sampling since timer directly triggers conversion**
- **Faster results since peripherals are optimized to perform operations more quickly than the CPU**

Configure Unused Pins

Configure Unused Pins

- **Digital input pins subject to shoot-through current**
  - Input voltages between V_{IL} and V_{IH} cause shoot-through if input is allowed to “float” (left disconnected)
- **Port I/O’s should either:**
  1. Be driven to V_{cc} or ground by an external device
  2. Set as an input using the pull-up/down resistor
  3. Driven as an output

(Digital) CMOS Inverter



Efficient Code Makes a Difference

ULP "Sweet Spot"

- **Power dissipation increases with…**
  - CPU clock speed (MCLK)
  - Input voltage (V_{CC})
  - Temperature
- **Slowing MCLK reduces instantaneous power, but often increases active duty-cycle (how long the CPU stays on)**
  - Look for ULP 'sweet spot' to maximize performance with minimum current consumption per MIPS
    - ☞ Usually 8 MHz MCLK is the best tradeoff of power/performance
- **Use lowest input voltage possible**
  - 'F5529 lets you lower core voltage if full-speed operation is not required
  - 'FR5969 operates at full speed down to 1.8V
  - On some MSP430 devices, you need to take into consideration minimum Vcc for flash programming, etc.

Timer Counter Module

Challenge statement: - choose one or many

- Wake up after a time or
- Create timed events or
- Create events at regular intervals or
- Count events for a time or
- Count determined number of events or
- Capture time between events or
- Capture elapsed time or
- Start processes at determined times
- Use your imagination – loop back to start

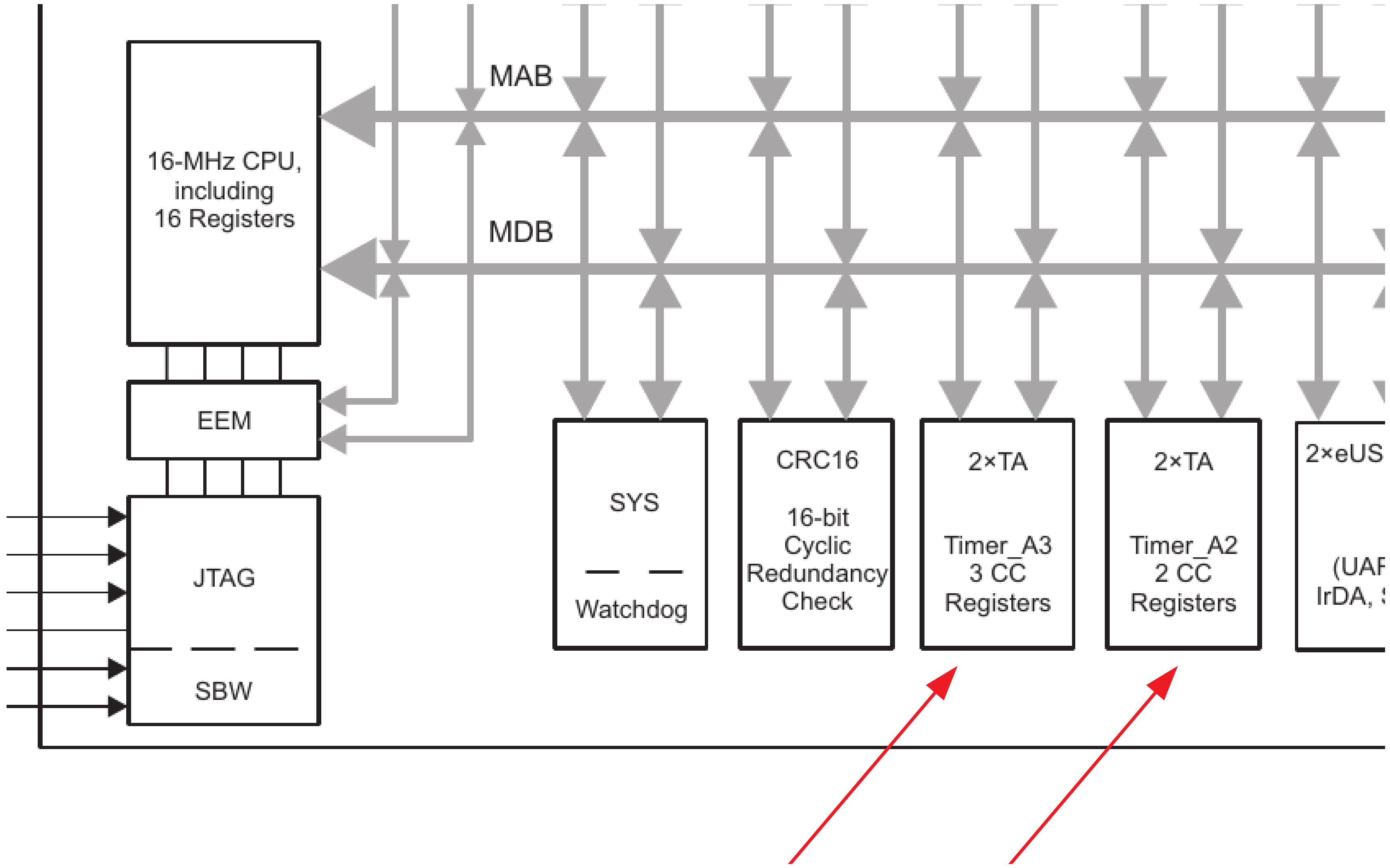
Count

Compare

Capture

Wake Up

Timer_A3 modules in MSP430FR2433



- Each Timer_A3 has three capture/compare registers, but only CCR1 and CCR2 are externally connected. CCR0 registers can be used only for internal period timing and interrupt generation.
- Each Timer_A2 has two capture/compare registers, but only CCR1 is a compare/capture functionality. CCR0 registers can be used only for internal period timing and interrupt generation.

As we will cover in great detail during this chapter, these timers contain one or more Capture and Compare Registers (CCR); these are useful for creating sophisticated timings, interrupts and waveforms. The more CCR registers a timer contains, the more independent waveforms that can be generated. To this end, the documentation often includes the number of CCR registers when listing the name of the timer. For example, if TIMER_A on a given device has 5 CCR registers, they often name it:

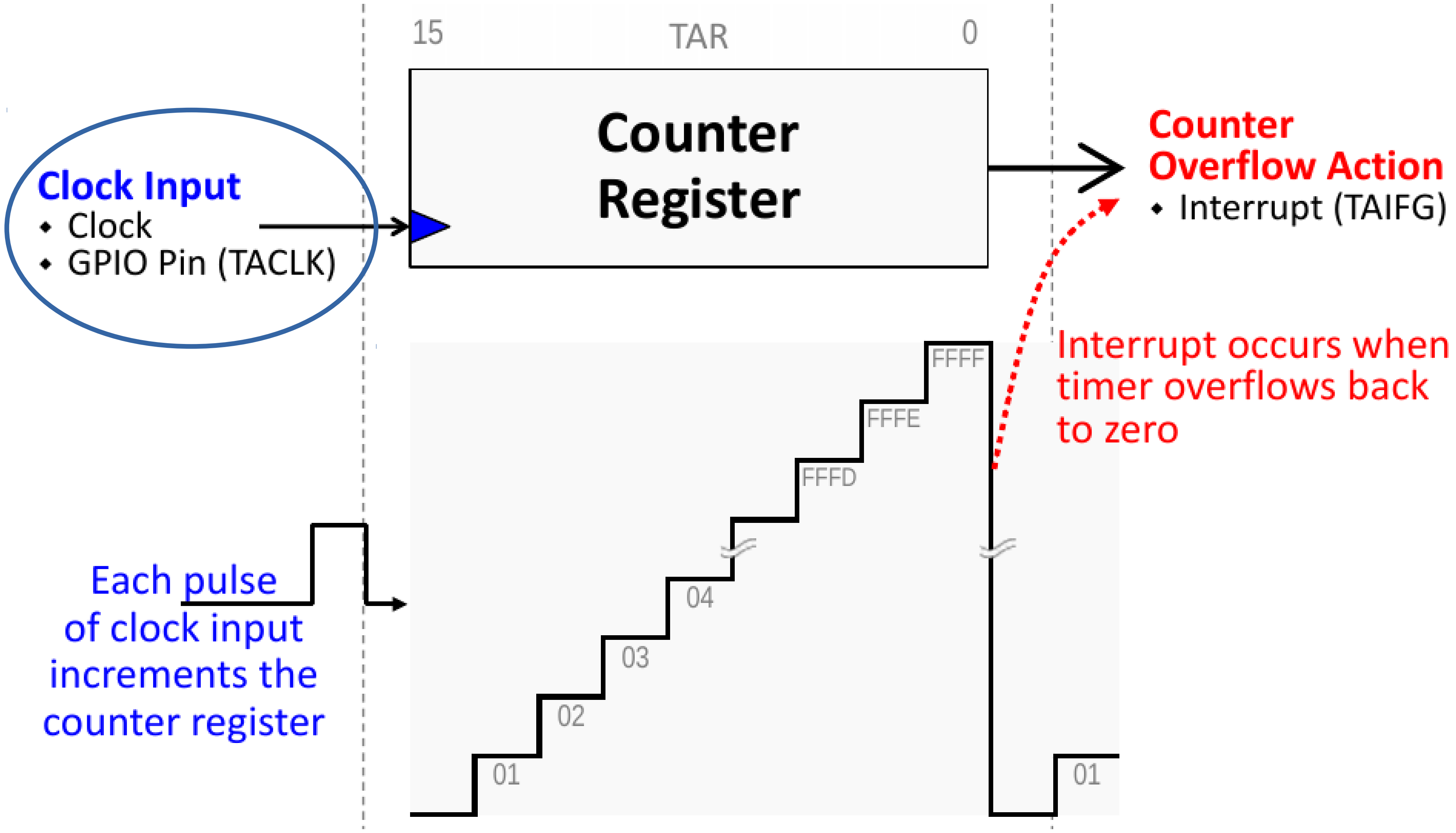
```
Timer_A5
```

But wait, that's not all. What happens when a device, such as the 'F5529 has more than one instance of TIMER_A? Each of these instances needs to be enumerated as well. This is done by appending the instance number after the word "Timer", as in Timer0.

To summarize, here's the long (and short) names for each of the 'F5529 TIMER_A modules:

Instance	Long Name	Short Name
0	Timer0_A5	TA0
1	Timer1_A3	TA1
2	Timer2_A3	TA2

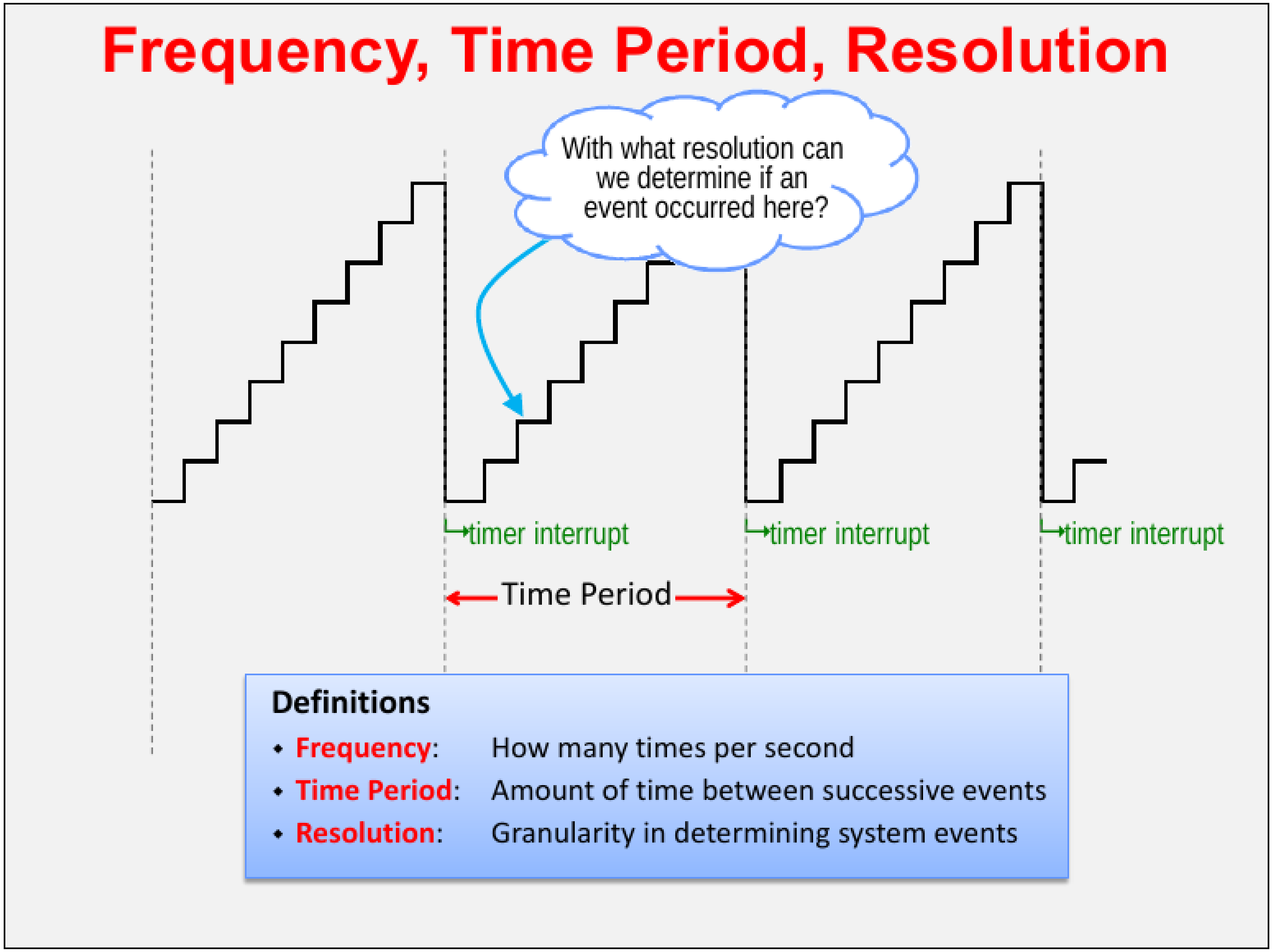
Timer/Counter Basics



Notes

- Timers are often called “Timer/Counters” as a counter is the essential element
- “Timing” is based on counting inputs from a known clock rate
- Actions don’t occur when writing value to counter

Can I 'capture' a count/time value?



Frequency, Time Period, Resolution

With what resolution can we determine if an event occurred here?
timer interrupt
timer interrupt
timer interrupt
Time Period

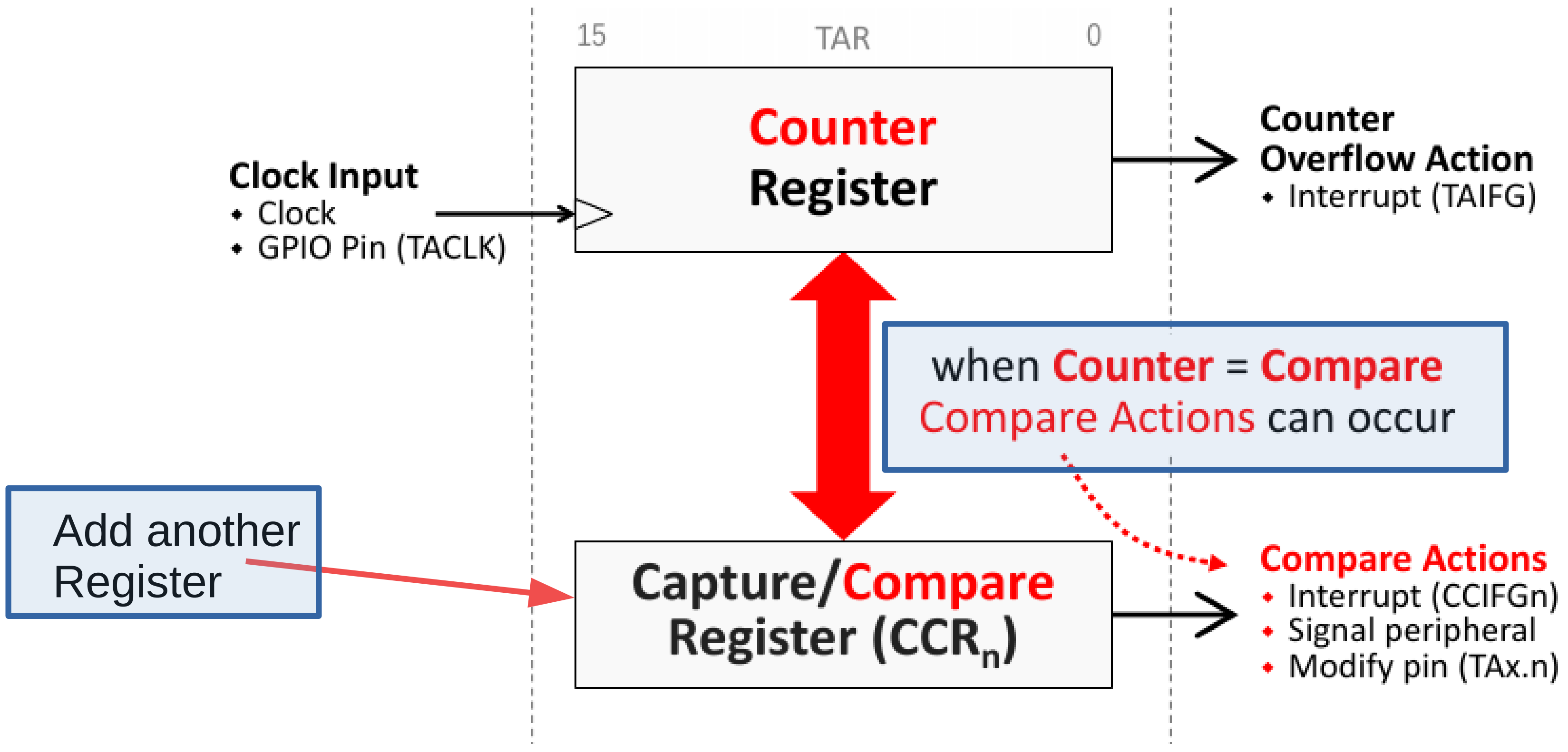
Definitions

• **Frequency**:	How many times per second
• **Time Period**:	Amount of time between successive events
• **Resolution**:	Granularity in determining system events

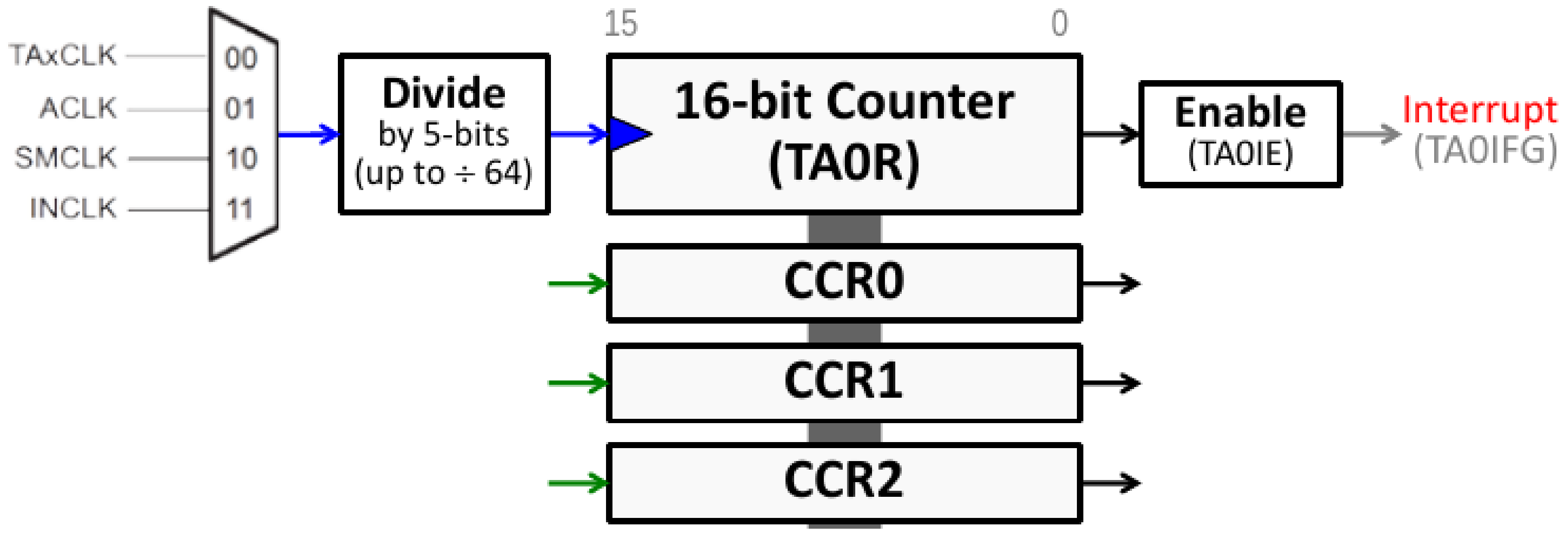
If a timer only consisted of a single counter, its *resolution* would be limited to the size of the counter.

If some event were to happen in a system – say, a user pushed a button – we could only ascertain if that event occurred within a time period. In other words, we can only determine if it happened between two interrupts.

The bottom portion of the diagram differs from the previous diagrams. In this case, rather than using the CCR register for capture, it's used as a *compare* register. In this mode, whenever a match between the Counter and Compare occurs, a compare action is triggered. The compare actions include generating an interrupt, signaling another peripheral (e.g. triggering an ADC conversion), or changing the state of an external pin.



The "modify pin" action is a very powerful capability. Using the timer's compare feature, we can create sophisticated PWM waveforms and/or trigger Analog/Digital Conversions



Remember:

• Timer0 means it's the first instance of Timer_A on the device.

• _A3 means that it's a Timer_A device and has 3 capture/compare registers (CCR's)

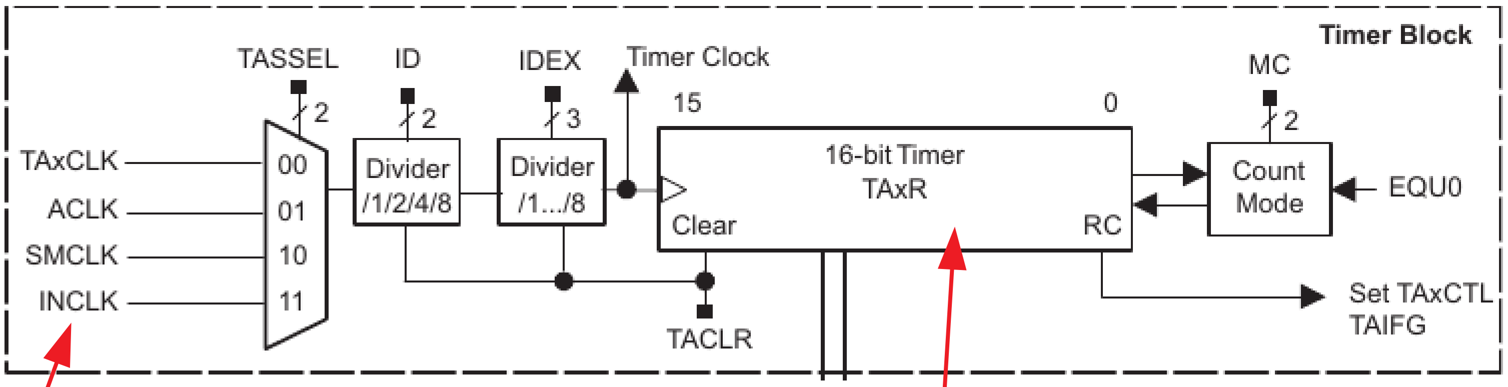
• The clock input, in this example, can be driven by a TACLK signal/pin, ACLK, SMCLK or another internal clock called INCLK.

• The clock input can be further divided down by a 5-bit scalar.

• The TA0IE interrupt enable can be used to allow (or prevent) an interrupt (TA0IFG) from reaching the CPU whenever the counter (TA0R) rolls over.

MSP430 Timer_A

- A 16-bit counter
- 4 modes of operation – Stop, Up, Continuous, Up/Down
- 3 capture/compare registers (CCRx)
- 2 interrupt vectors – TACCR0 and TAIV

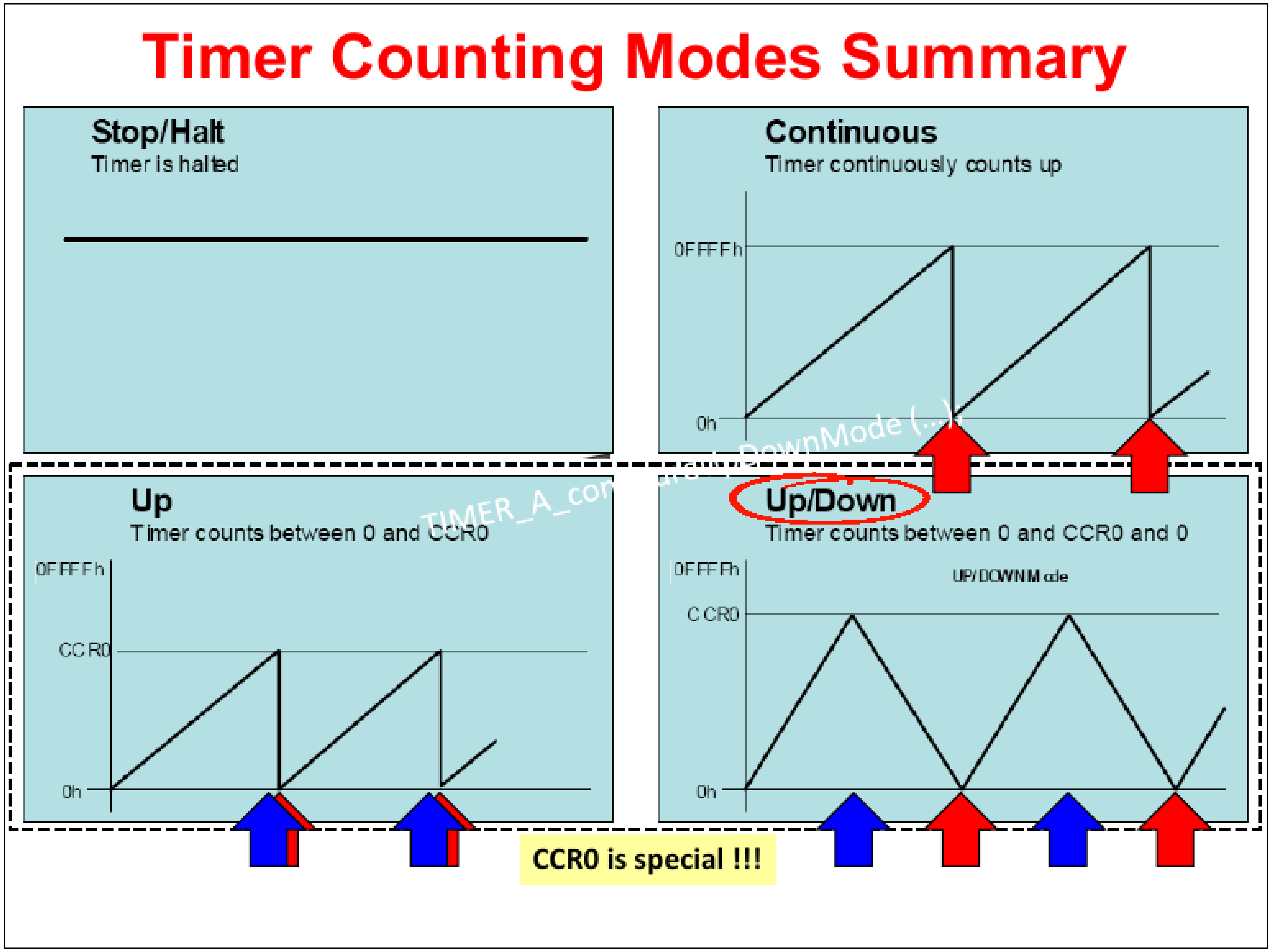


12.2.1 *16-Bit Timer Counter*

The 16-bit timer/counter register, TAxR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAxR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

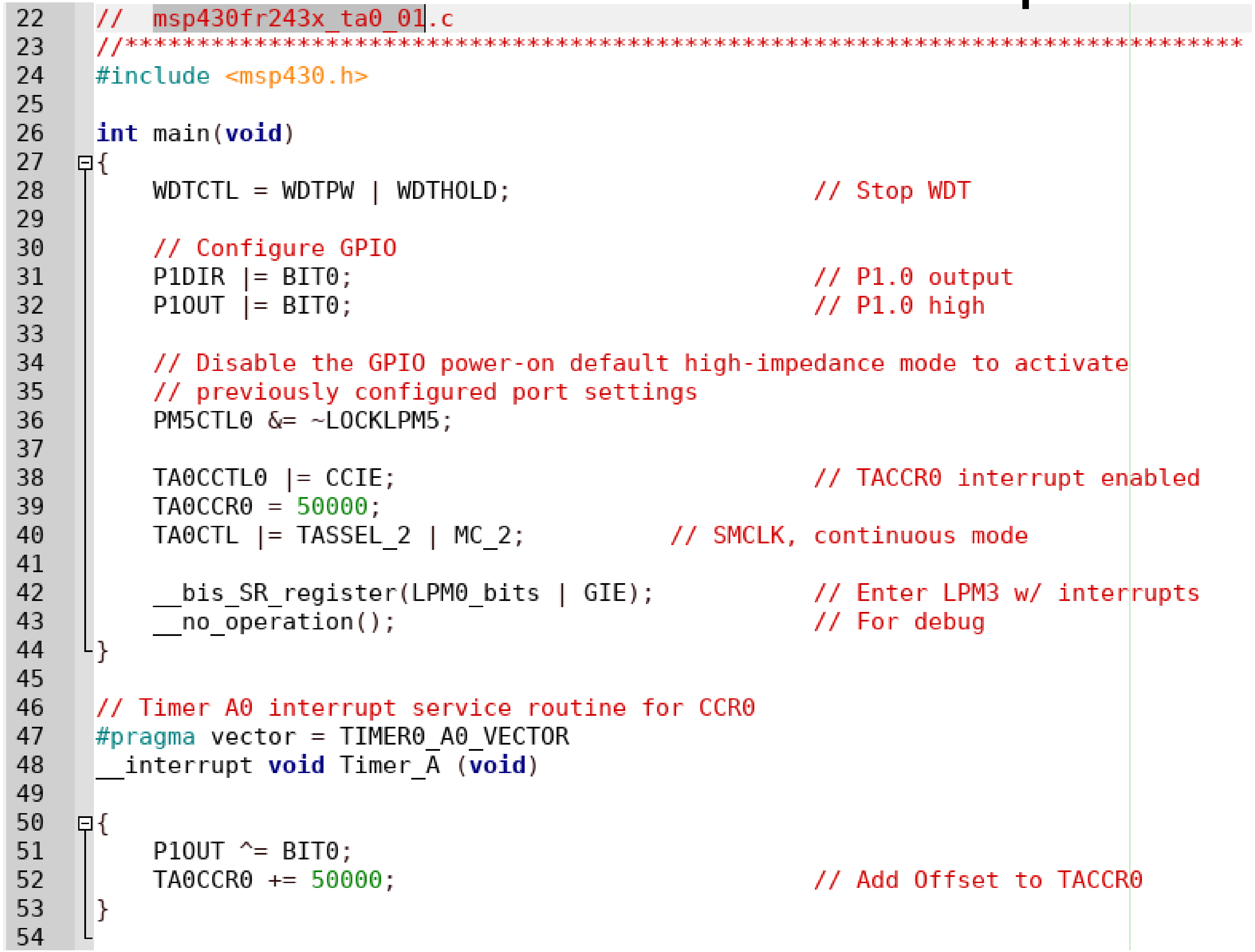
12.2.1.1 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally from TAxCLK or INCLK. The clock source is selected with the TASSEL bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the TAIDEX bits. The timer clock divider logic is reset when TACLR is set.

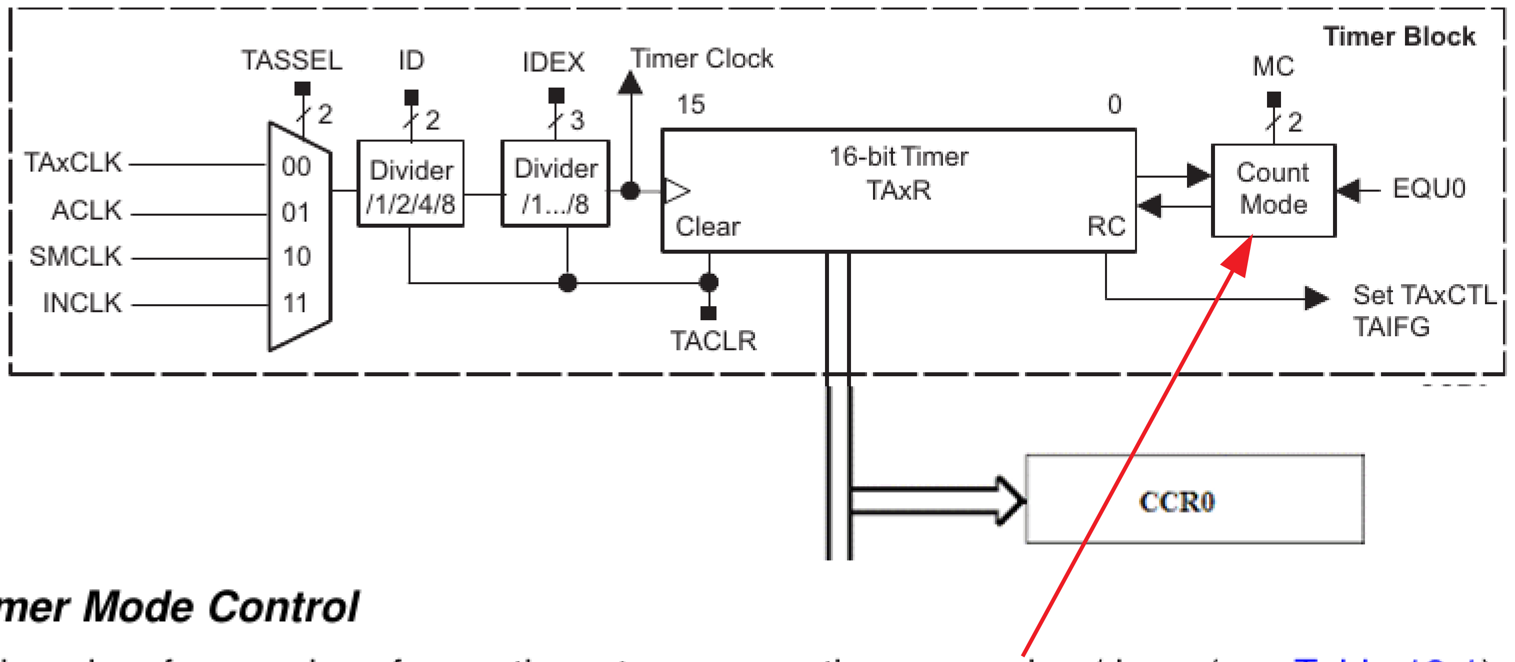


Using Compare mode (Up and Up/Down)

Continuous Mode Example: TA0



```c
//  msp430fr243x_ta0_01.c
//******************************************************************************
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                     // Stop WDT

    // Configure GPIO
    P1DIR |= BIT0;                                // P1.0 output
    P1OUT |= BIT0;                                // P1.0 high

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    TA0CCTL0 |= CCIE;                             // TACCR0 interrupt enabled
    TA0CCR0 = 50000;
    TA0CTL |= TASSEL_2 | MC_2;          // SMCLK, continuous mode

    __bis_SR_register(LPM0_bits | GIE);           // Enter LPM3 w/ interrupts
    __no_operation();                             // For debug
}

// Timer A0 interrupt service routine for CCR0
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)

{
    P1OUT ^= BIT0;
    TA0CCR0 += 50000;                             // Add Offset to TACCR0
}

```



12.2.3 *Timer Mode Control*

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 12-1). The operating mode is selected with the MC bits.

Table 12-1. Timer Modes

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

To move from one mode to another, first stop the timer by writing zero to the MC bits (MC = 0), then set the MC bits to the desired mode (see Table 12-1 for details).

Table 12-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 0 when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/Down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and always reads as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

12.2.3.2 Continuous Mode

In the Continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 12-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

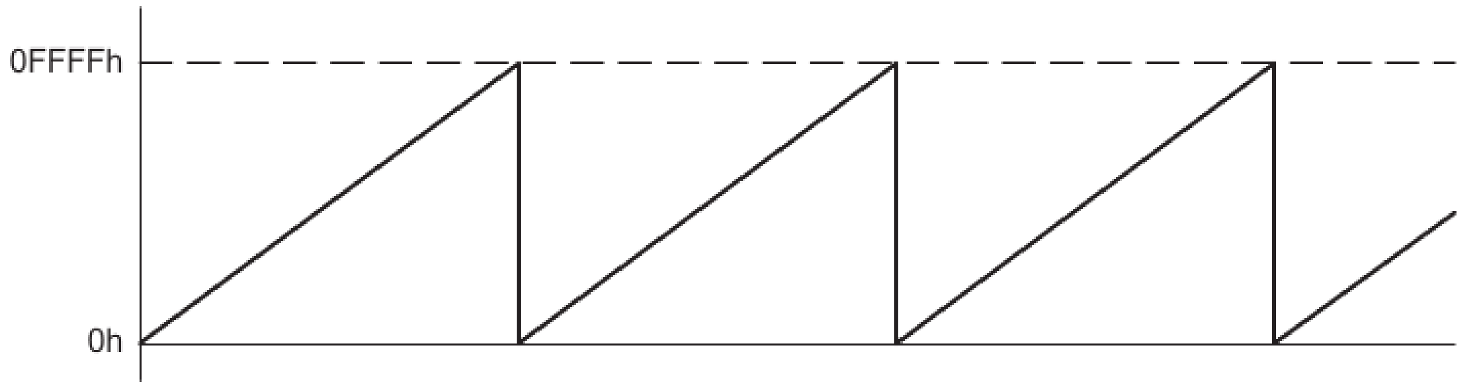
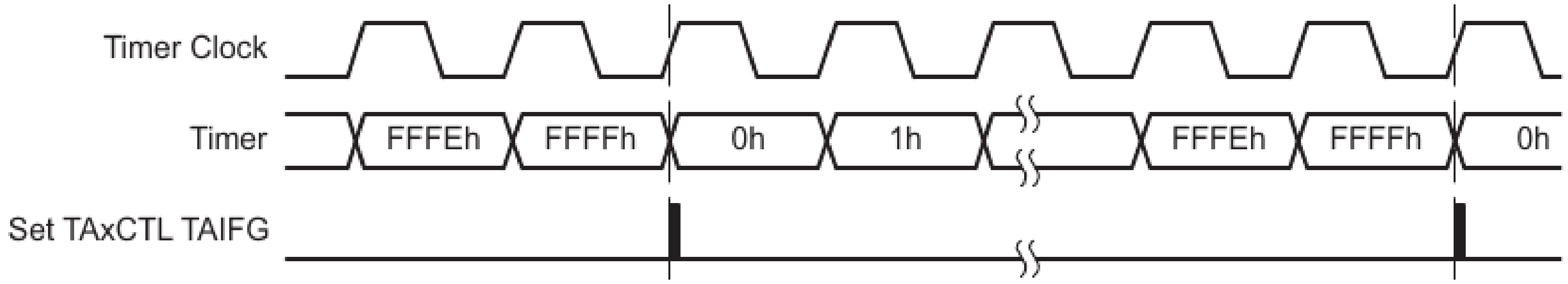
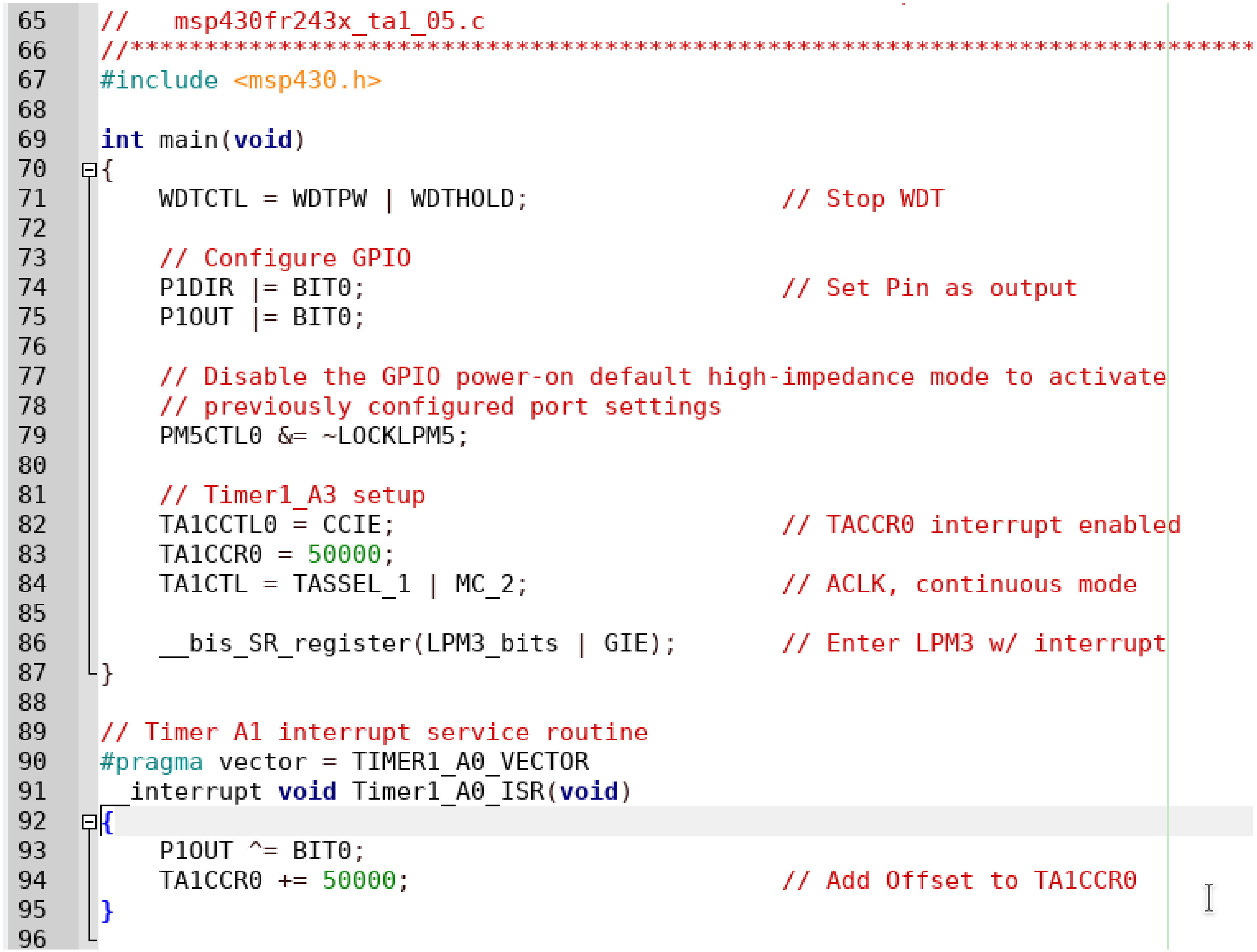


Figure 12-4. Continuous Mode

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 12-5 shows the flag set cycle.



Continuous Mode: Timer1_A3 – using TA1



```c
//   msp430fr243x_ta1_05.c
//******************************************************************************
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                 // Stop WDT

    // Configure GPIO
    P1DIR |= BIT0;                            // Set Pin as output
    P1OUT |= BIT0;

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    // Timer1_A3 setup
    TA1CCTL0 = CCIE;                          // TACCR0 interrupt enabled
    TA1CCR0 = 50000;
    TA1CTL = TASSEL_1 | MC_2;                 // ACLK, continuous mode

    __bis_SR_register(LPM3_bits | GIE);       // Enter LPM3 w/ interrupt
}

// Timer A1 interrupt service routine
#pragma vector = TIMER1_A0_VECTOR
__interrupt void Timer1_A0_ISR(void)
{
    P1OUT ^= BIT0;
    TA1CCR0 += 50000;                         // Add Offset to TA1CCR0
}
```

12.2.3.3 Use of Continuous Mode

The Continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TAxCCRn register in the interrupt service routine. Figure 12-6 shows two separate time intervals, t_0 and t_1, being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 6), independent time intervals or output frequencies can be generated using capture/compare registers.

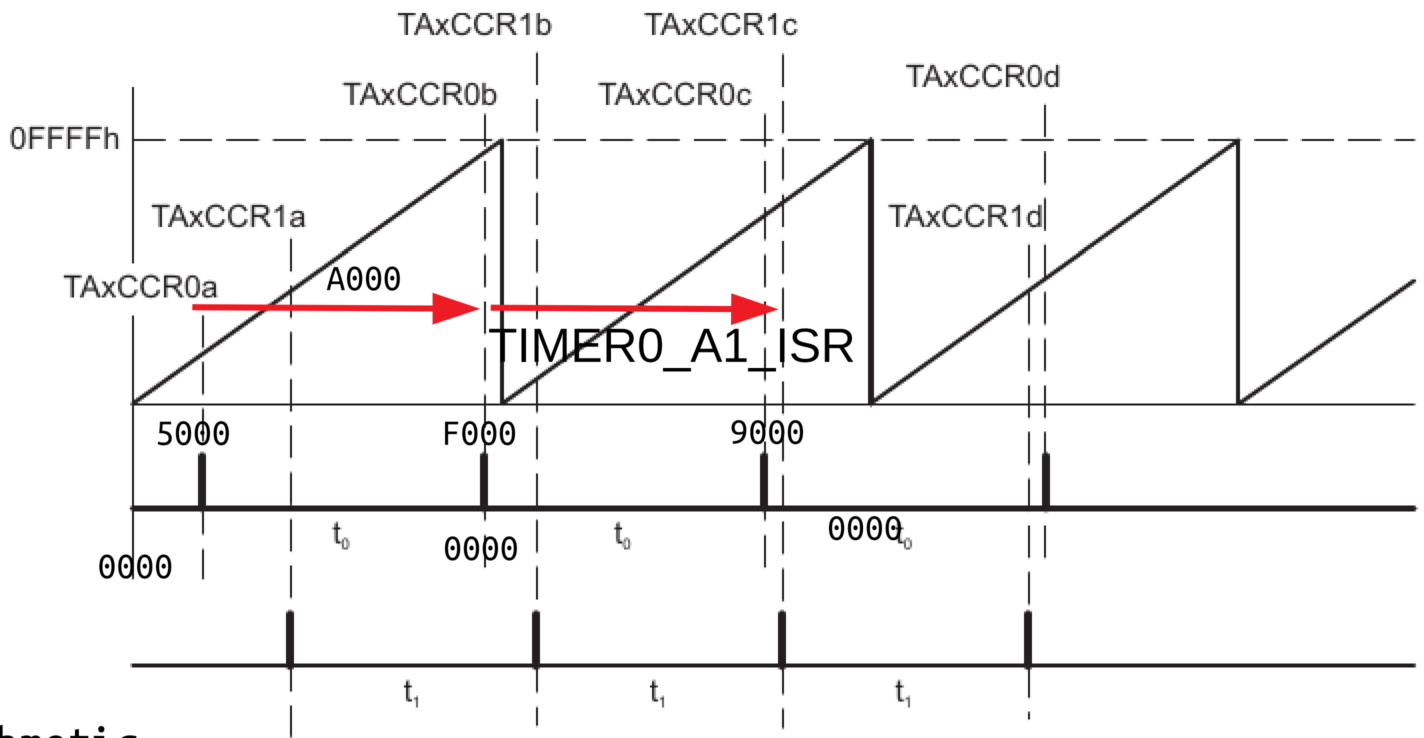


Figure 12-6. Continuous Mode Time Intervals

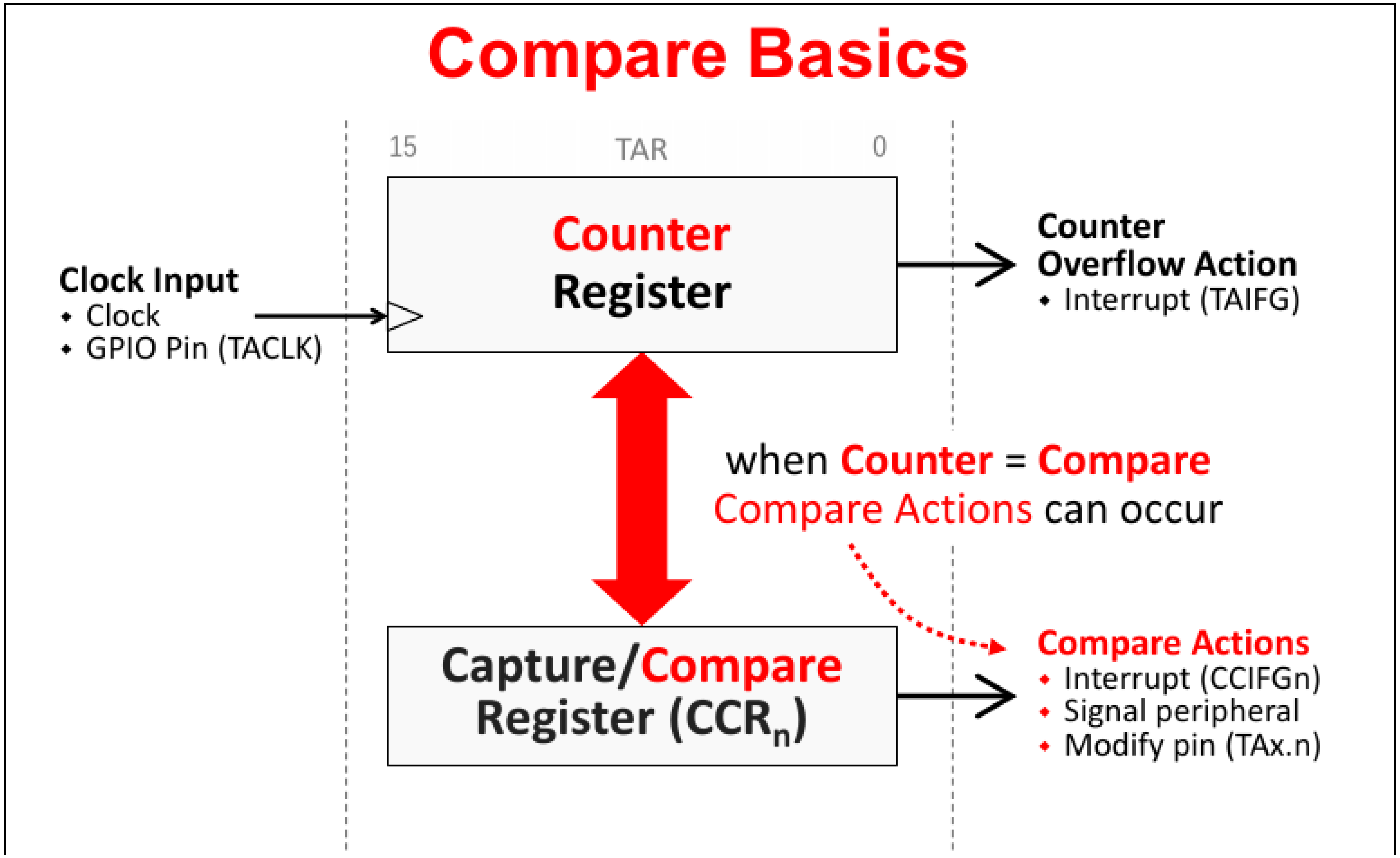
```
16 bit arithmetic

  F000
+ A000
 19000
  9000
```

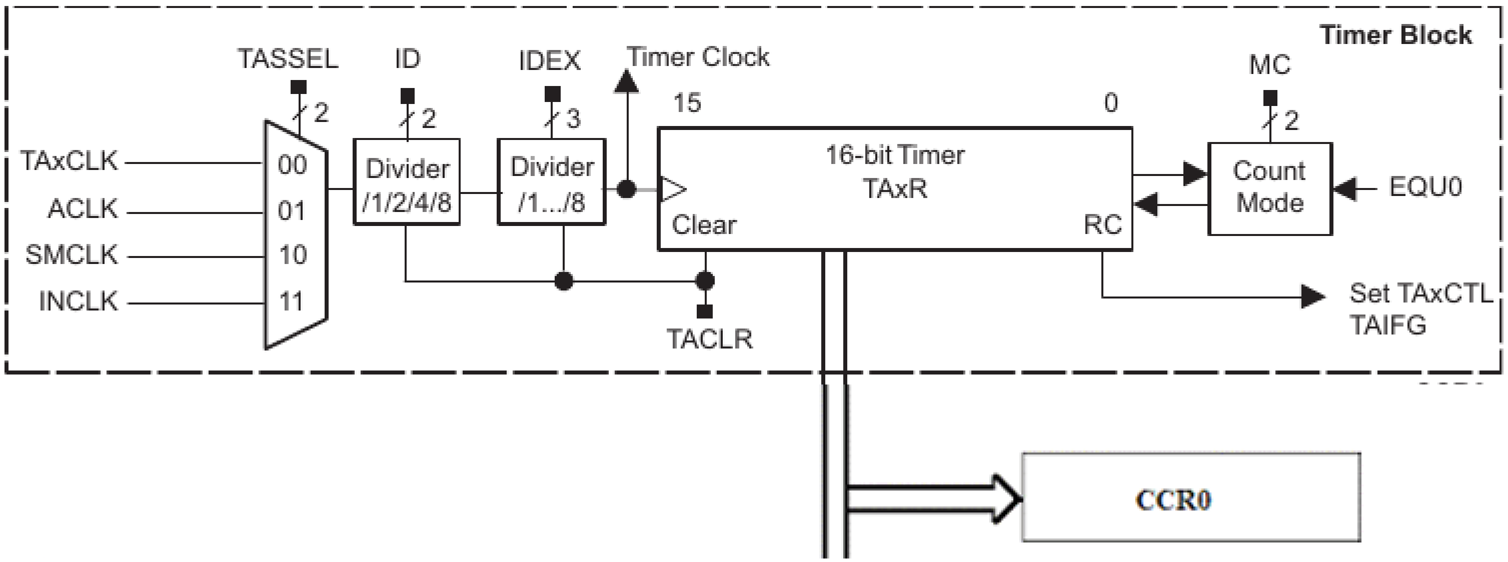
More than 16 bit result, so answer just ‘wraps around’
Final answer is 9000

Compare

A key feature for timers is the ability to create a consistent, periodic interrupts.

As we know, TIMER_A can do this, but the timer's frequency (i.e. time period) is limited to dividing the input clock by 2^{16}. So, while the timer may be *consistent*, but not very flexible. Thankfully, the ***Compare*** feature of TIMER_A (TIMER_B & TIMER_D) solves this problem.





The Up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TAxCCR0, which defines the period (see Figure 12-2). The number of timer counts in the period is TAxCCR0 + 1. When the timer value equals TAxCCR0, the timer restarts counting from zero. If Up mode is selected when the timer value is greater than TAxCCR0, the timer immediately restarts counting from zero.

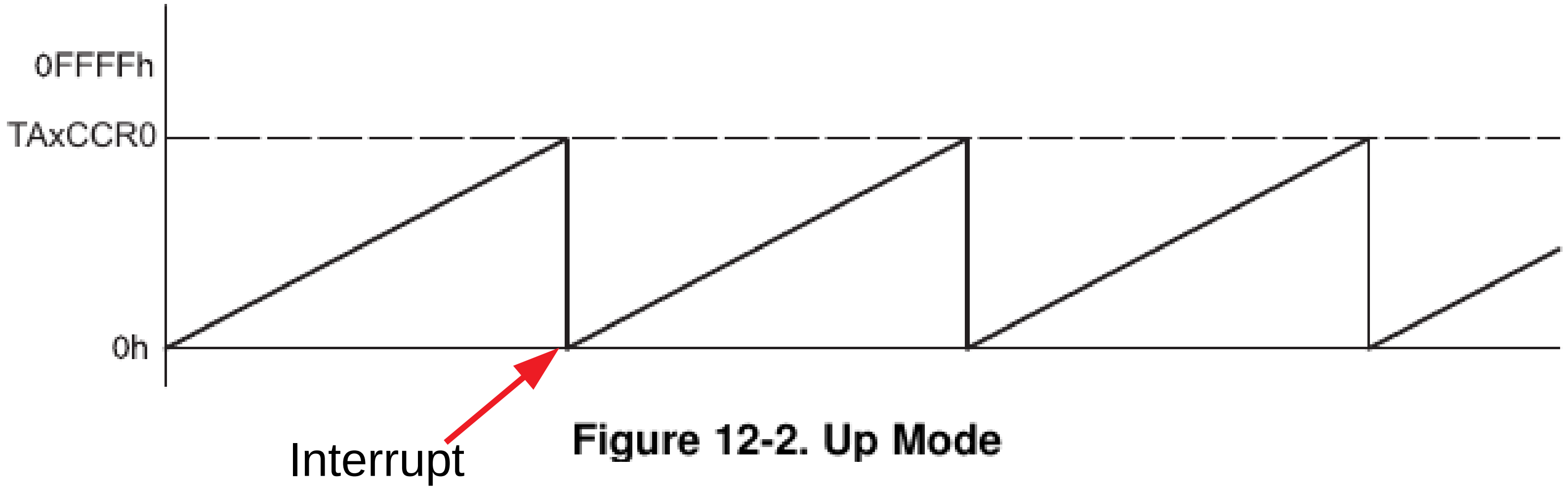
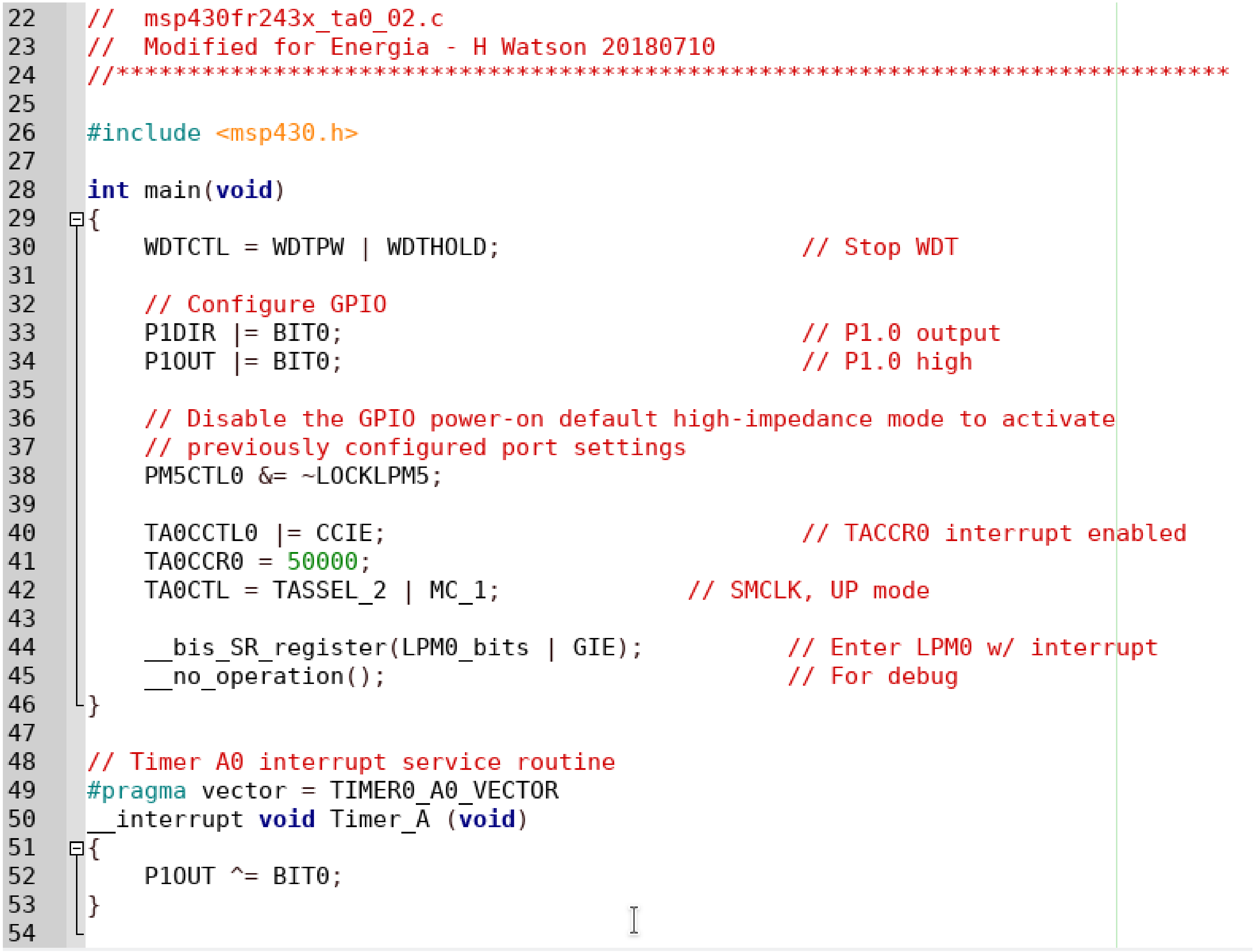


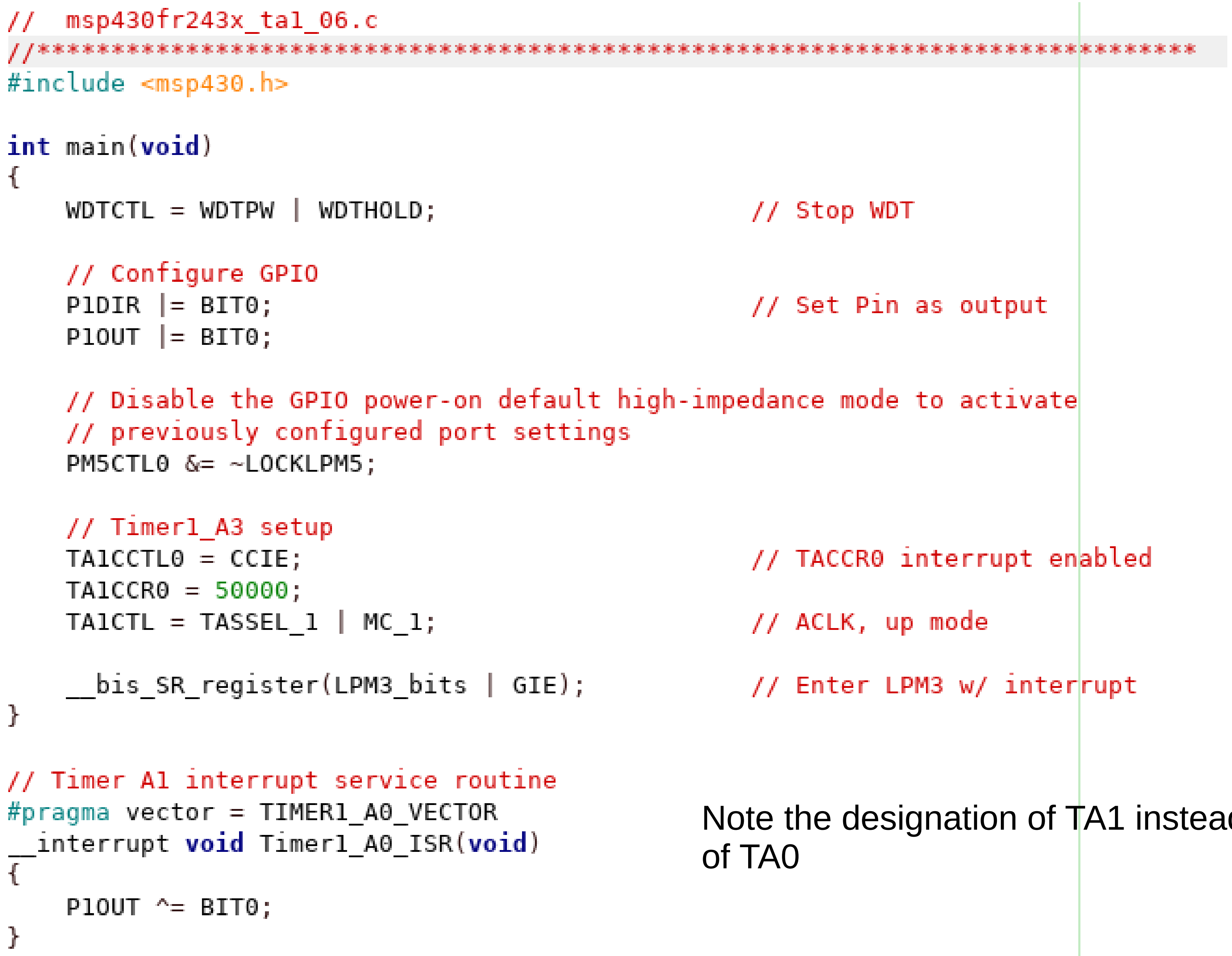
Figure 12-2. Up Mode

Count Up Mode – Interrupt & Zero when TA0 == CCR0



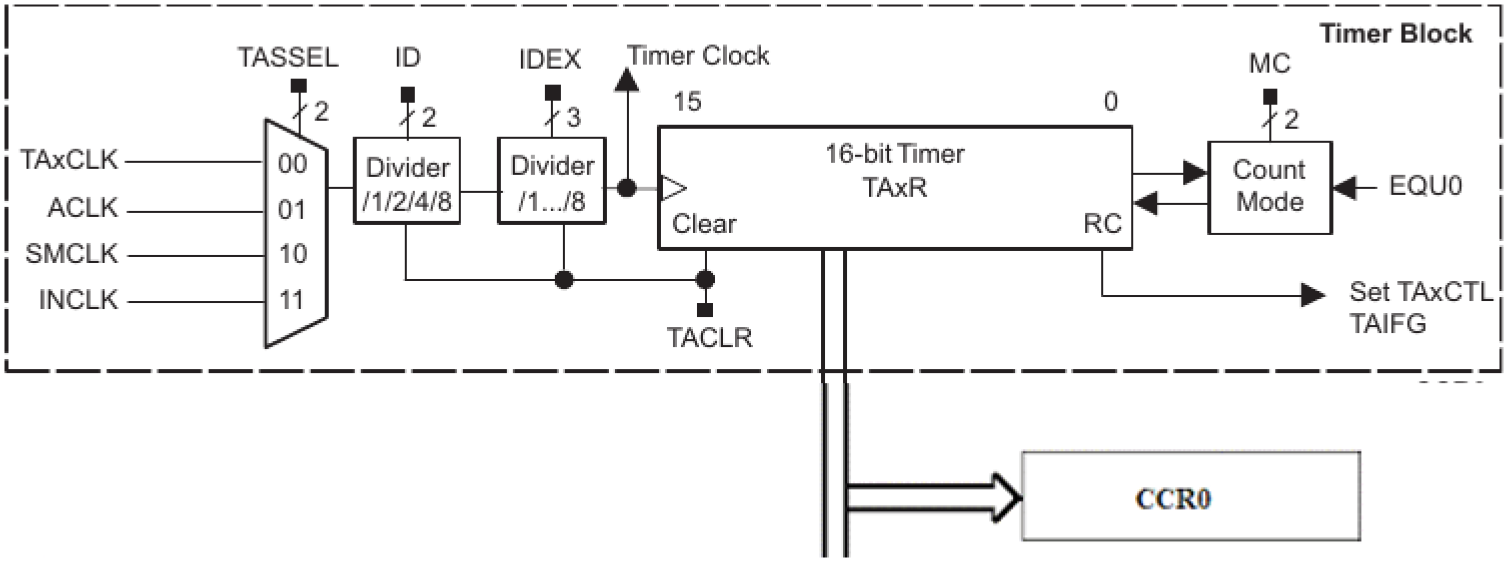
```c
//  msp430fr243x_ta0_02.c
//  Modified for Energia - H Watson 20180710
//******************************************************************************

#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                     // Stop WDT

    // Configure GPIO
    P1DIR |= BIT0;                                // P1.0 output
    P1OUT |= BIT0;                                // P1.0 high

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    TA0CCTL0 |= CCIE;                             // TACCR0 interrupt enabled
    TA0CCR0 = 50000;
    TA0CTL = TASSEL_2 | MC_1;             // SMCLK, UP mode

    __bis_SR_register(LPM0_bits | GIE);          // Enter LPM0 w/ interrupt
    __no_operation();                            // For debug
}

// Timer A0 interrupt service routine
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    P1OUT ^= BIT0;
}

```

Same: Count Up Mode – Interrupt when TA1 == CCR1



```
//  msp430fr243x_ta1_06.c
//******************************************************************************
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                     // Stop WDT

    // Configure GPIO
    P1DIR |= BIT0;                                // Set Pin as output
    P1OUT |= BIT0;

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    // Timer1_A3 setup
    TA1CCTL0 = CCIE;                              // TACCR0 interrupt enabled
    TA1CCR0 = 50000;
    TA1CTL = TASSEL_1 | MC_1;                     // ACLK, up mode

    __bis_SR_register(LPM3_bits | GIE);           // Enter LPM3 w/ interrupt
}

// Timer A1 interrupt service routine
#pragma vector = TIMER1_A0_VECTOR
__interrupt void Timer1_A0_ISR(void)
{
    P1OUT ^= BIT0;
}
```

Note the designation of TA1 instead of TA0



The Up/Down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TAxCCR0 and back down to zero (see Figure 12-7). The period is twice the value in TAxCCR0.

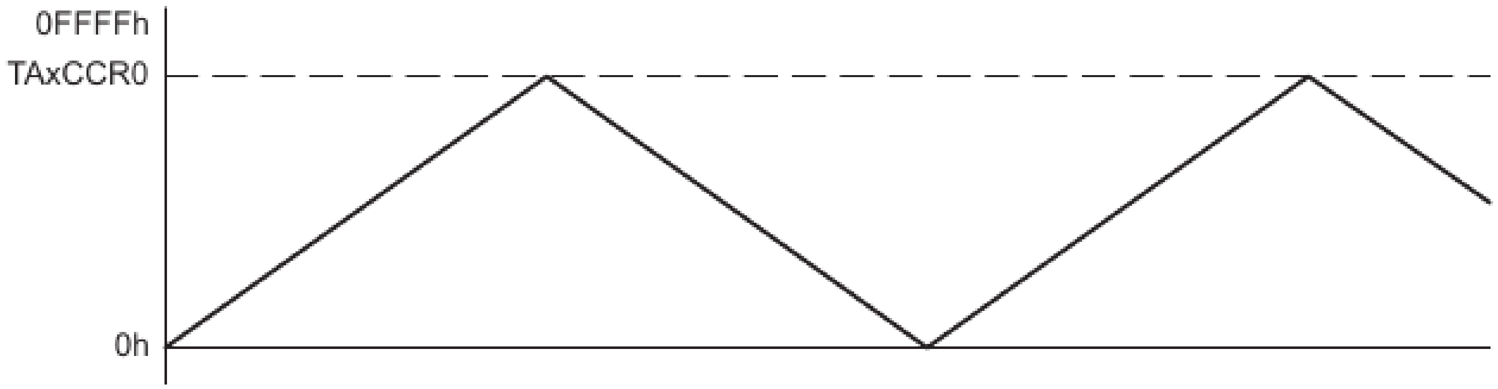
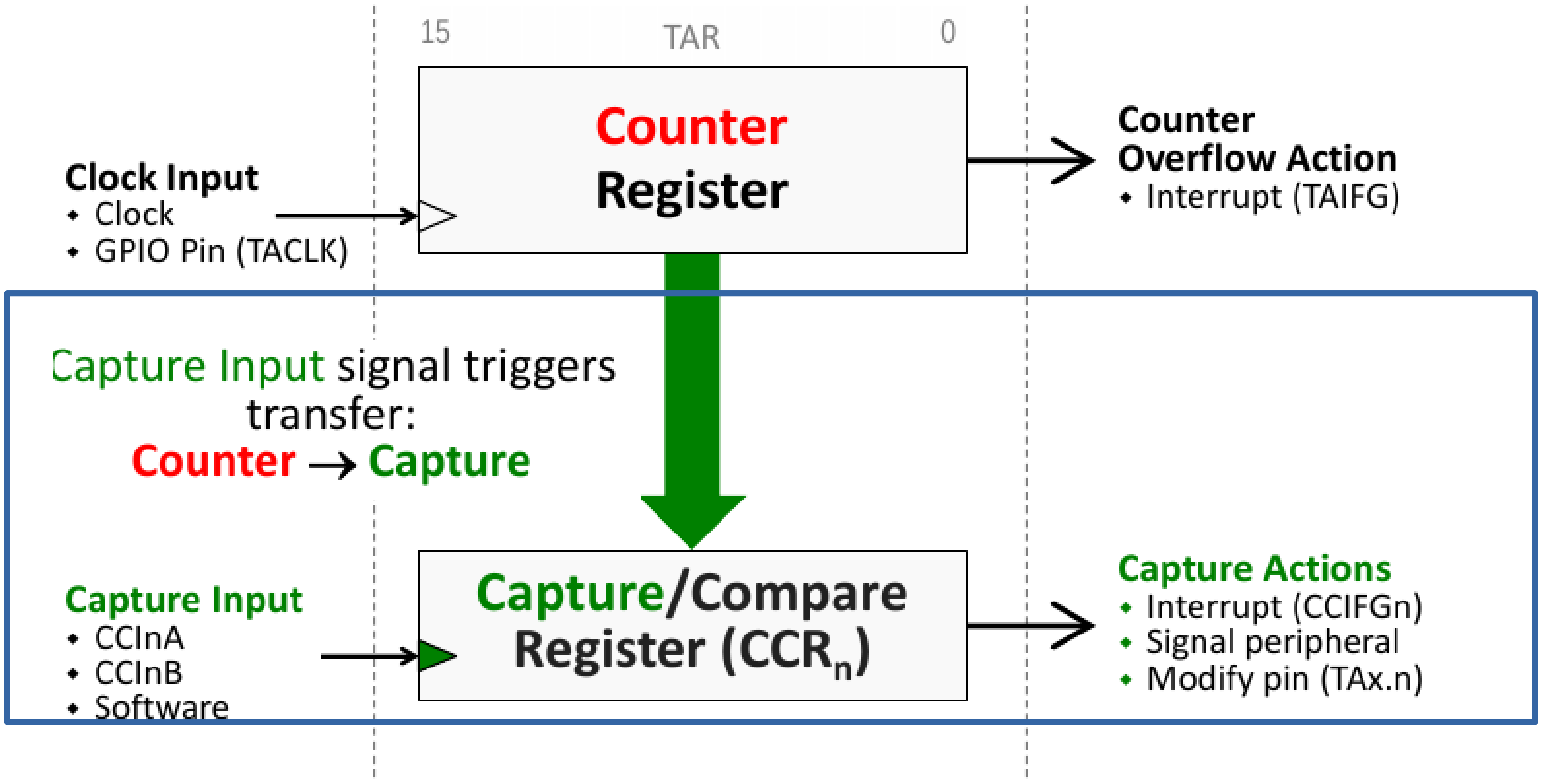


Figure 12-7. Up/Down Mode

Mainly with PWM

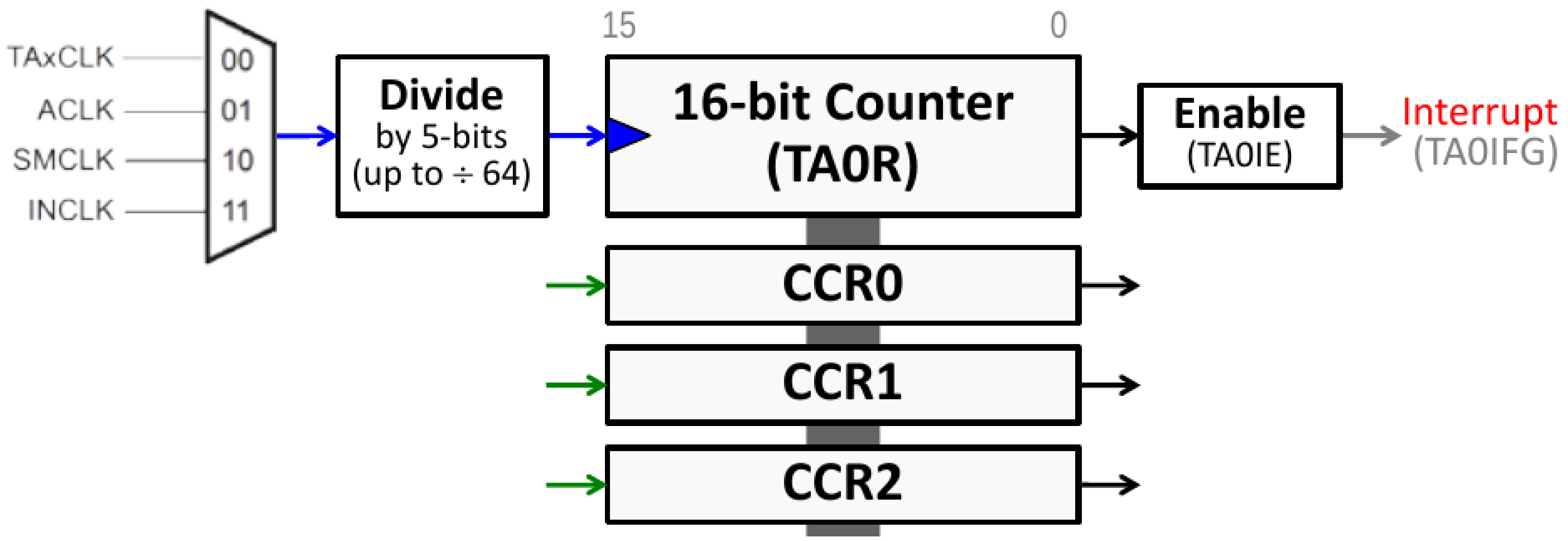
Using the second register with the **Capture** feature allows the contents of the **Counter** to be captured (the **Counter** value is copied into the **Capture** register with no latency and very low power)

When a **Capture Input** signal occurs, the value from the Counter Register (**TAR**) is copied into the capture register (i.e. **CCR**)



Notes
- Capture time (i.e. count value) when Capture Input signal occurs
- When capture is triggered, count value is placed in CCR and an interrupt is generated
- Capture Overflow (COV): indicates 2nd capture to CCR before 1st was read

Summary Example: Timer0_A3 Summary



Remember:

- Timer0 means it's the first instance of Timer_A on the device.

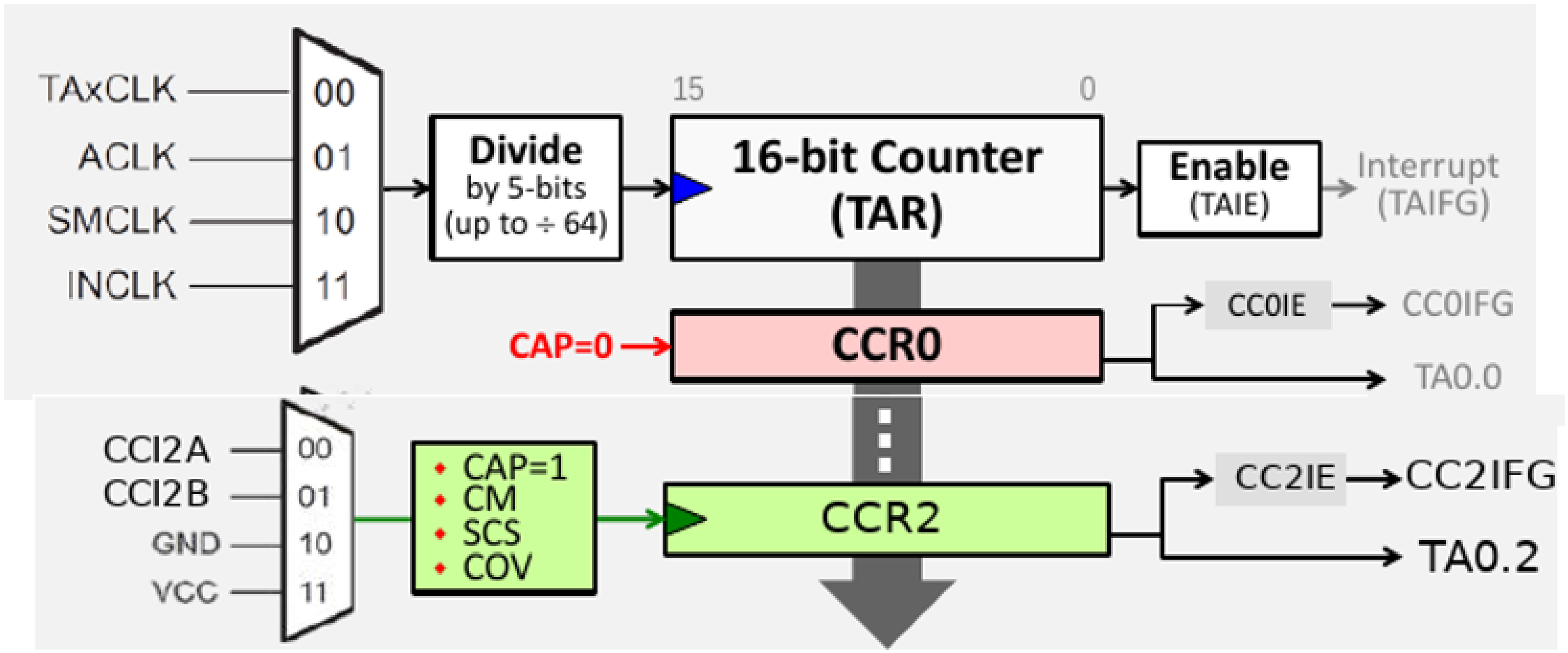
- _A3 means that it's a Timer_A device and has 3 capture/compare registers (CCR's)

- The clock input, in this example, can be driven by a TACLK signal/pin, ACLK, SMCLK or another internal clock called INCLK.

- The clock input can be further divided down by a 5-bit scalar.

- The TA0IE interrupt enable can be used to allow (or prevent) an interrupt (TA0IFG) from reaching the CPU whenever the counter (TA0R) rolls over.

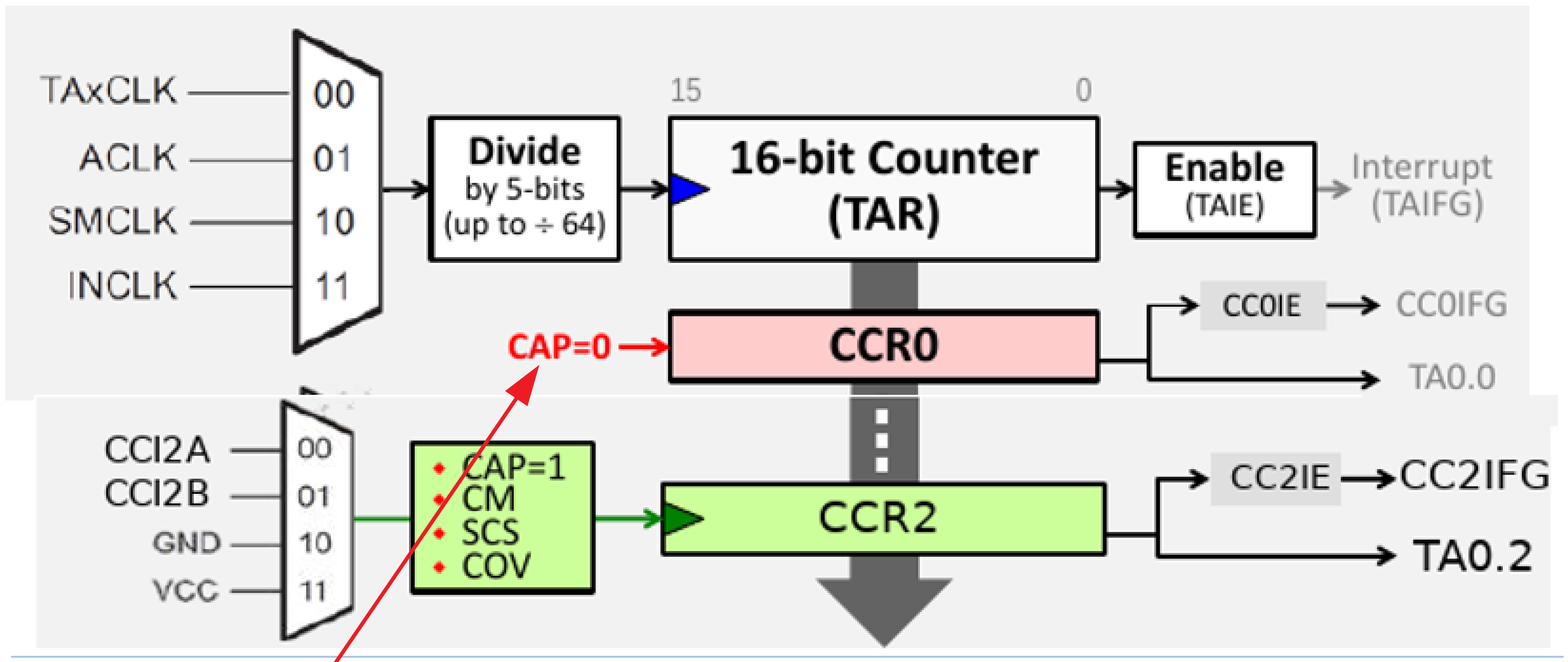
Timer_A3 Summary



Timer0_A3:
- Is the first instance (Timer0 or TA0) of Timer_A3 on the device
- _A3 means it has 3 Capture/Compare Registers (CCR's)

CCR registers can be configured for:
- Compare (set when CAP=0) generates interrupt (CCnIFG) and modifies OUT signal when TAR = CCRn
- Capture (when CAP=1) grabs the TAR value and sets an interrupt (CcnIFG) when triggered by the selected CCIx input

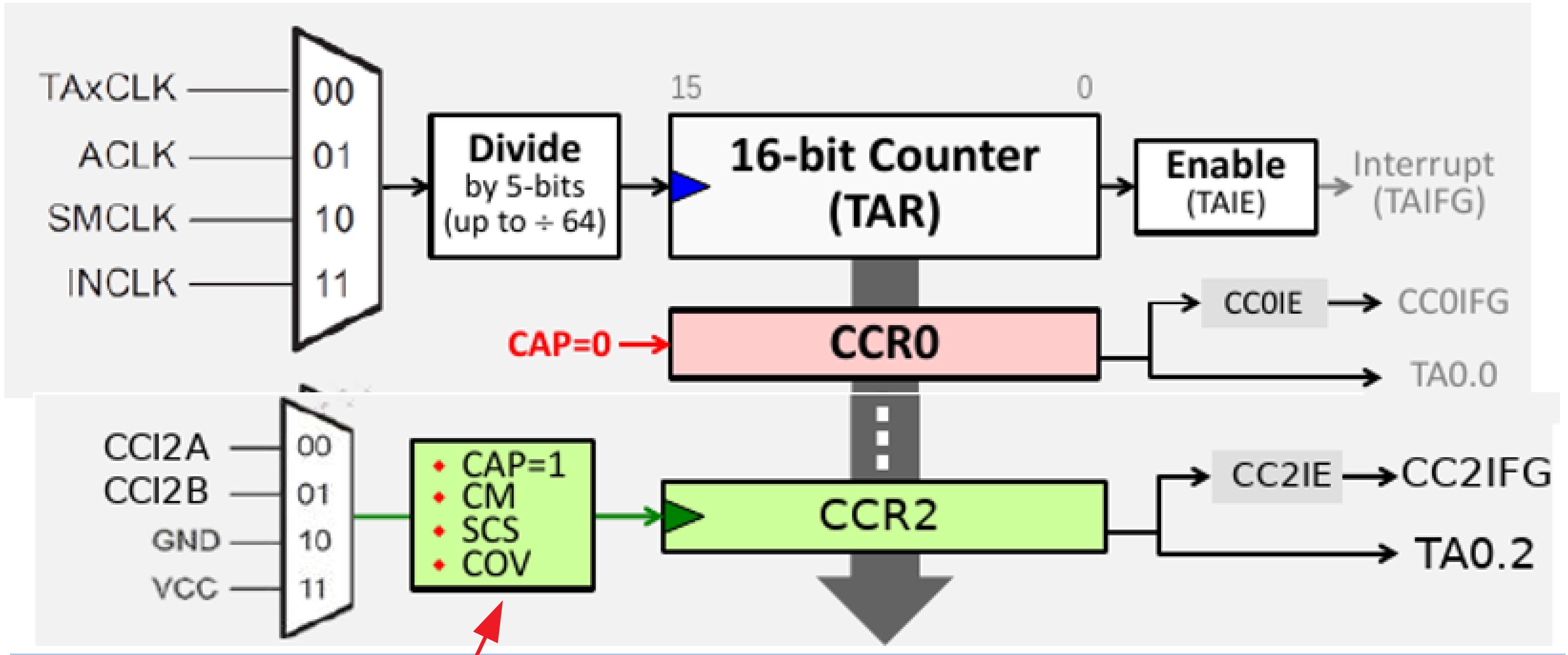
Timer_A3 Compare Mode Summary



CAP=0 (Capture off)

- Compare mode on
- If CCR = TAR (named EQU0):
  - Interrupt occurs (if enabled)
  - OUT is set/reset/toggled
- OUT can be:
  - Connected to pin (TA0.0)
  - Routed to peripherals
  - OUT bit can be polled
- Many OUT signal options (up to 7 options)

Timer_A3 Capture Mode Summary



- **Capture or Compare (CAP)**
  CAP=1 for capture
- **Which Edge (CM)**
  Rising, Falling, or Both
- **Sync'd to Clock (SCS)**
  Is capture sync or async?
- **Capture Overflow (COV)**
  Did you miss a capture?

Measure 'split' times

i.e. Capture the value of the TAR when Input signal occurs

There are two interrupt flags (CCIFG and TAIFG) and its corresponding two interrupt vectors (TACCR0 and TAIV) available for Timers in MSP430 as shown in the figure 1.3.

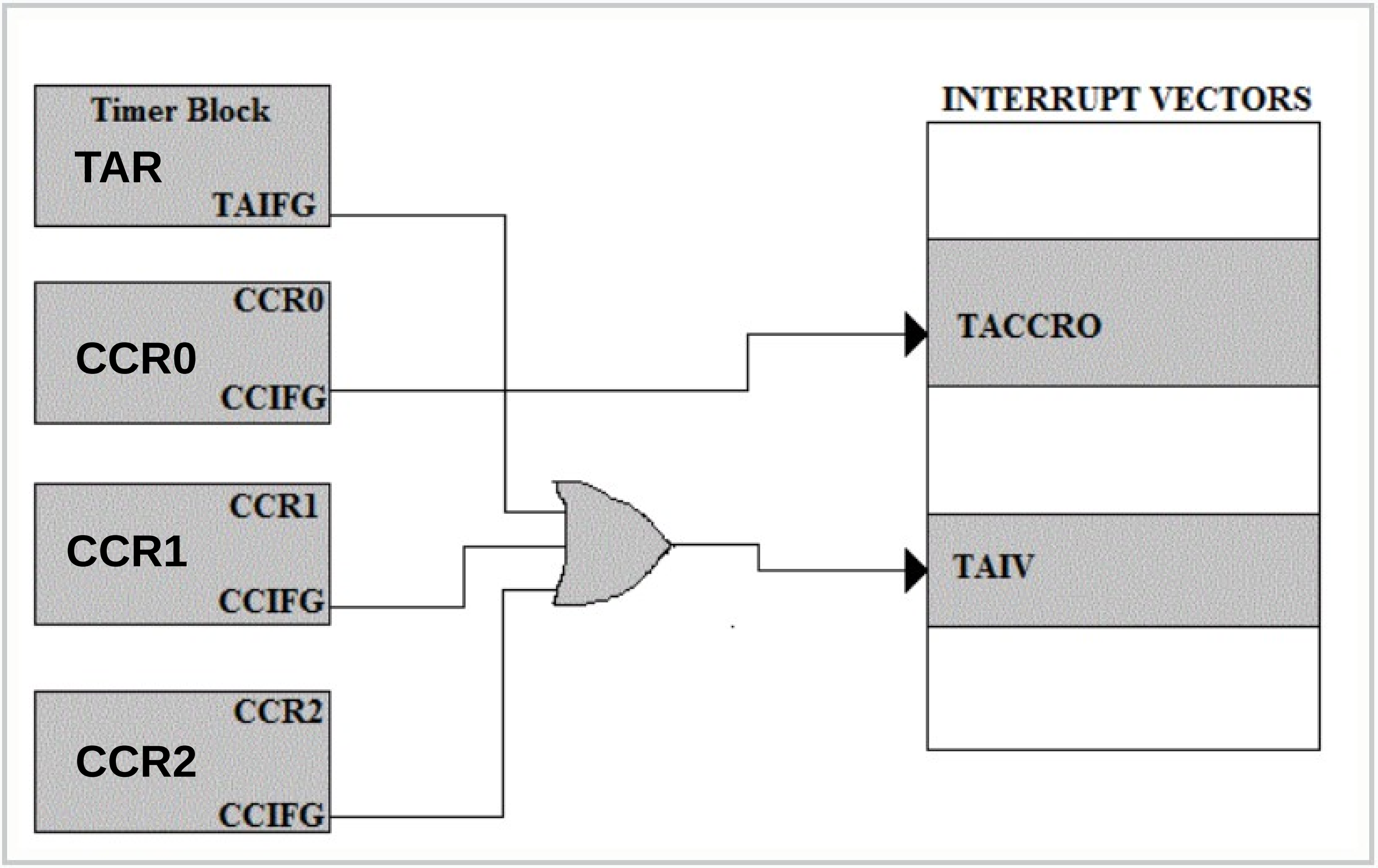
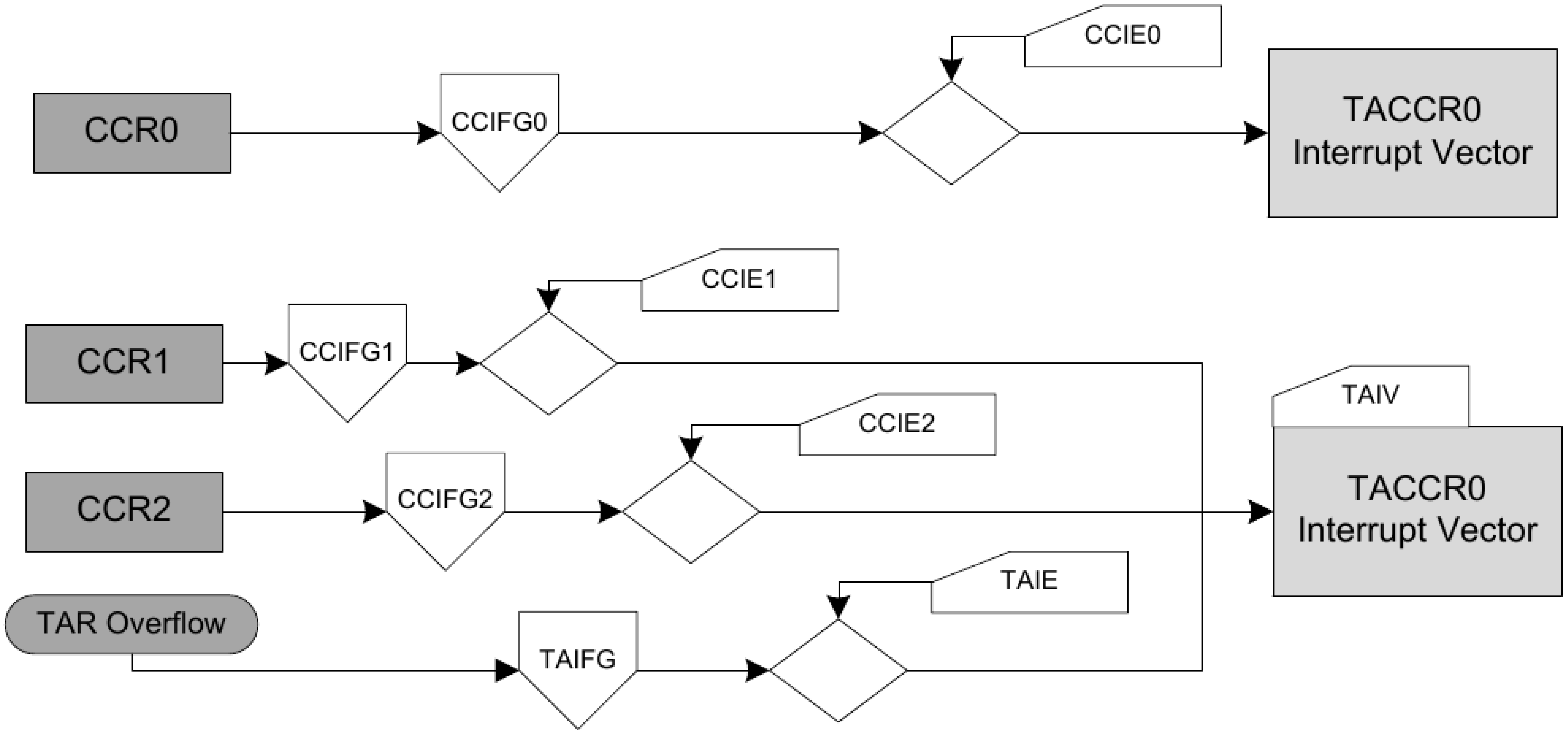
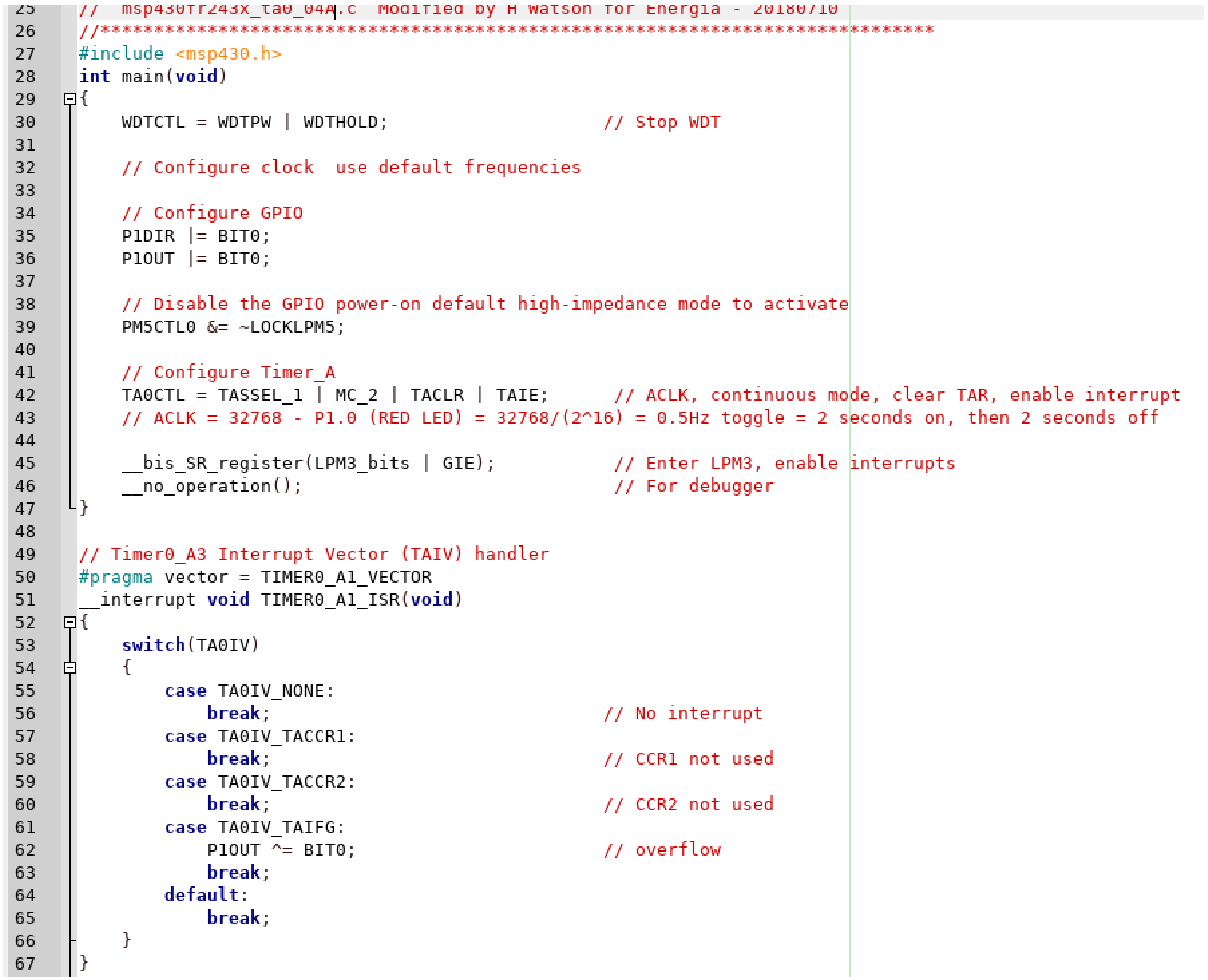


Figure 1.3 shows Timer interrupts and its corresponding interrupt vectors in MSP430

Timer_A Interrupt Vectors

- TACCR0 interrupt vector for CCIFG of CCR0
- TAIV interrupt vector for TAIFG and CCIFGs of CCR1,CCR2




```
//  msp430fr243x_ta0_04A.c  Modified by H Watson for Energia - 20180710
//******************************************************************************
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                    // Stop WDT

    // Configure clock  use default frequencies

    // Configure GPIO
    P1DIR |= BIT0;
    P1OUT |= BIT0;

    // Disable the GPIO power-on default high-impedance mode to activate
    PM5CTL0 &= ~LOCKLPM5;

    // Configure Timer_A
    TA0CTL = TASSEL_1 | MC_2 | TACLR | TAIE;      // ACLK, continuous mode, clear TAR, enable interrupt
    // ACLK = 32768 - P1.0 (RED LED) = 32768/(2^16) = 0.5Hz toggle = 2 seconds on, then 2 seconds off

    __bis_SR_register(LPM3_bits | GIE);           // Enter LPM3, enable interrupts
    __no_operation();                             // For debugger
}

// Timer0_A3 Interrupt Vector (TAIV) handler
#pragma vector = TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void)
{
    switch(TA0IV)
    {
        case TA0IV_NONE:
            break;                               // No interrupt
        case TA0IV_TACCR1:
            break;                               // CCR1 not used
        case TA0IV_TACCR2:
            break;                               // CCR2 not used
        case TA0IV_TAIFG:
            P1OUT ^= BIT0;                       // overflow
            break;
        default:
            break;
    }
}
```

Timer_Ax Interrupt Vector Register

Figure 12-20. TAxIV Register

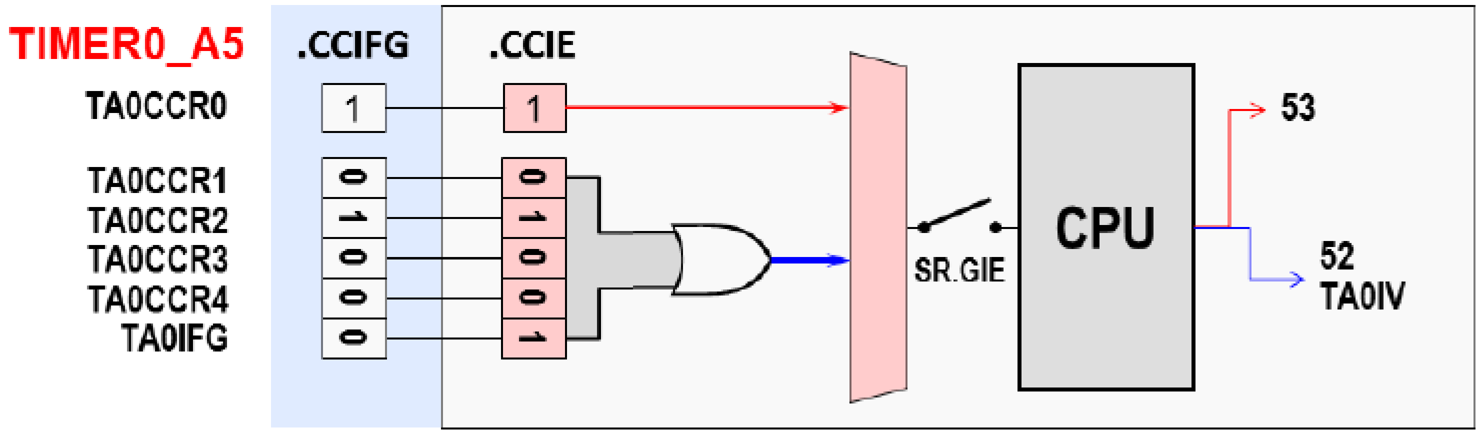
15	14	13	12	11	10	9	8
TAIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TAIV							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

Table 12-8. TAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	TAIV	R	0h	Timer_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest 04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest

Timer0_A5 Interrupts Review

INT Source	IFG	IV Register	Vector Address	Loc'n
Timer A (CCIFG0)	TA0CCR0.CCIFG	none	TIMER0_A0_VECTOR	53
Timer A	TA0CCR1.IFG1…TA0CCR4.IFG	TA0IV	TIMER0_A1_VECTOR	52



- In the interrupts chapter, we learned that most MPS430 interrupts are grouped together and share an interrupt vector, although a few have their own dedicated vector
- Timers A and B have two vectors: one for CCR0 and the other shared
- When the CPU responds to TIMER0_A0_VECTOR, the CCR0IFG is auto cleared
- In the TIMER0_A1_VECTOR ISR, reading **TA0IV** register returns the associated highest priority pending interrupt and clears it's IFG bit

TIMER0_A0_VECTOR is exclusively used for the CCR0 capture/compare event. Since CCR0 has the capability to reset the timer in up or up/down mode and therefore define the timer cycle, it has been given an own, higher priority interrupt. The CCIFG bit in CCTL0 is also automatically reset when entering this ISR, as there is only one possible reason for this interrupt.

On A1_VECTOR, you'll have to check what was causing the interrupt and manually clear the IFG bits (or any access, read or write, of the TAIV register automatically resets the highest pending interrupt flag). If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

The separate A0_VECTOR allows extremely short reaction times to the CCR0 event, which is necessary for controlling a PWM duty-cycle change or other situations.

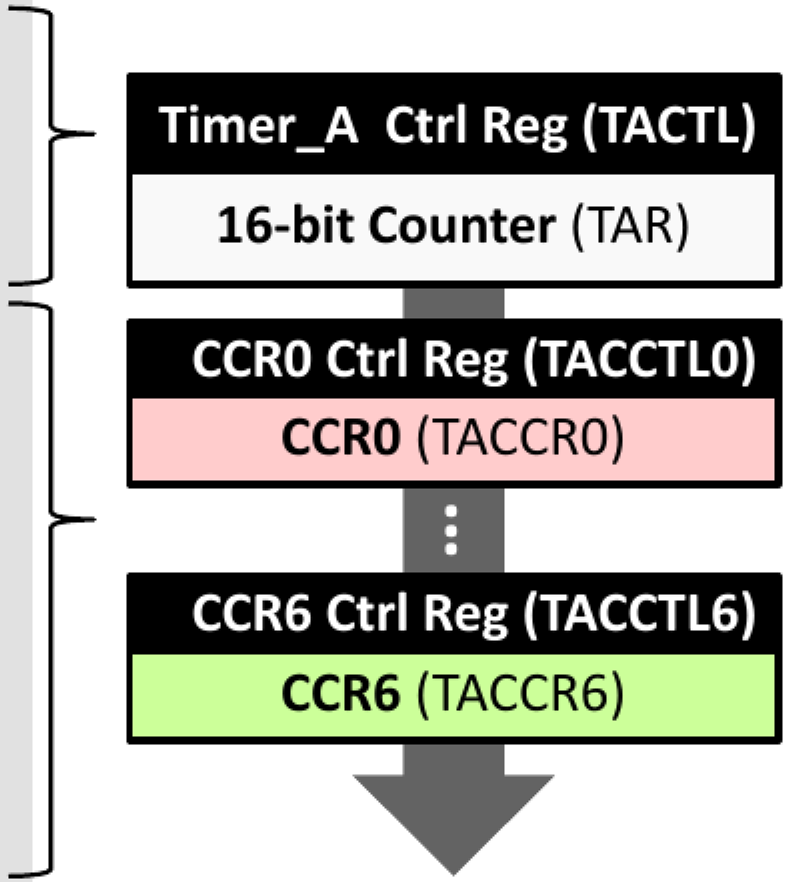
4 Steps to Program Timer_A

Timer Setup Code

1. Configure Timer/Counter (TACTL)
   - Clocking
   - Which Count Mode
   - Interrupt on TAR rollover?
2. Setup Capture and/or Compare Registers
   - Capture (TACCTL):
     - Input
     - Interrupt on Capture?
   - Compare (TACCTL, TACCR):
     - Compare-to Value
     - Output mode (How output signal changes at compare (EQU) events)
     - Interrupt on Compare?
3. Clear interrupt Flags & Start Timer

Timer Interrupt Service Routine(s)

4. Write 1-2 ISR's (CCR0, others)



Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	TAxCTL	Timer_Ax Control	Read/write	Word	0000h	Section 12.3.1
02h	TAxCCTL0	Timer_Ax Capture/Compare Control 0	Read/write	Word	0000h	Section 12.3.3
04h	TAxCCTL1	Timer_Ax Capture/Compare Control 1	Read/write	Word	0000h	Section 12.3.3
06h	TAxCCTL2	Timer_Ax Capture/Compare Control 2	Read/write	Word	0000h	Section 12.3.3
10h	TAxR	Timer_Ax Counter	Read/write	Word	0000h	Section 12.3.2
12h	TAxCCR0	Timer_Ax Capture/Compare 0	Read/write	Word	0000h	Section 12.3.4
14h	TAxCCR1	Timer_Ax Capture/Compare 1	Read/write	Word	0000h	Section 12.3.4
16h	TAxCCR2	Timer_Ax Capture/Compare 2	Read/write	Word	0000h	Section 12.3.4
2Eh	TAxIV	Timer_Ax Interrupt Vector	Read only	Word	0000h	Section 12.3.5
20h	TAxEX0	Timer_Ax Expansion 0	Read/write	Word	0000h	Section 12.3.6

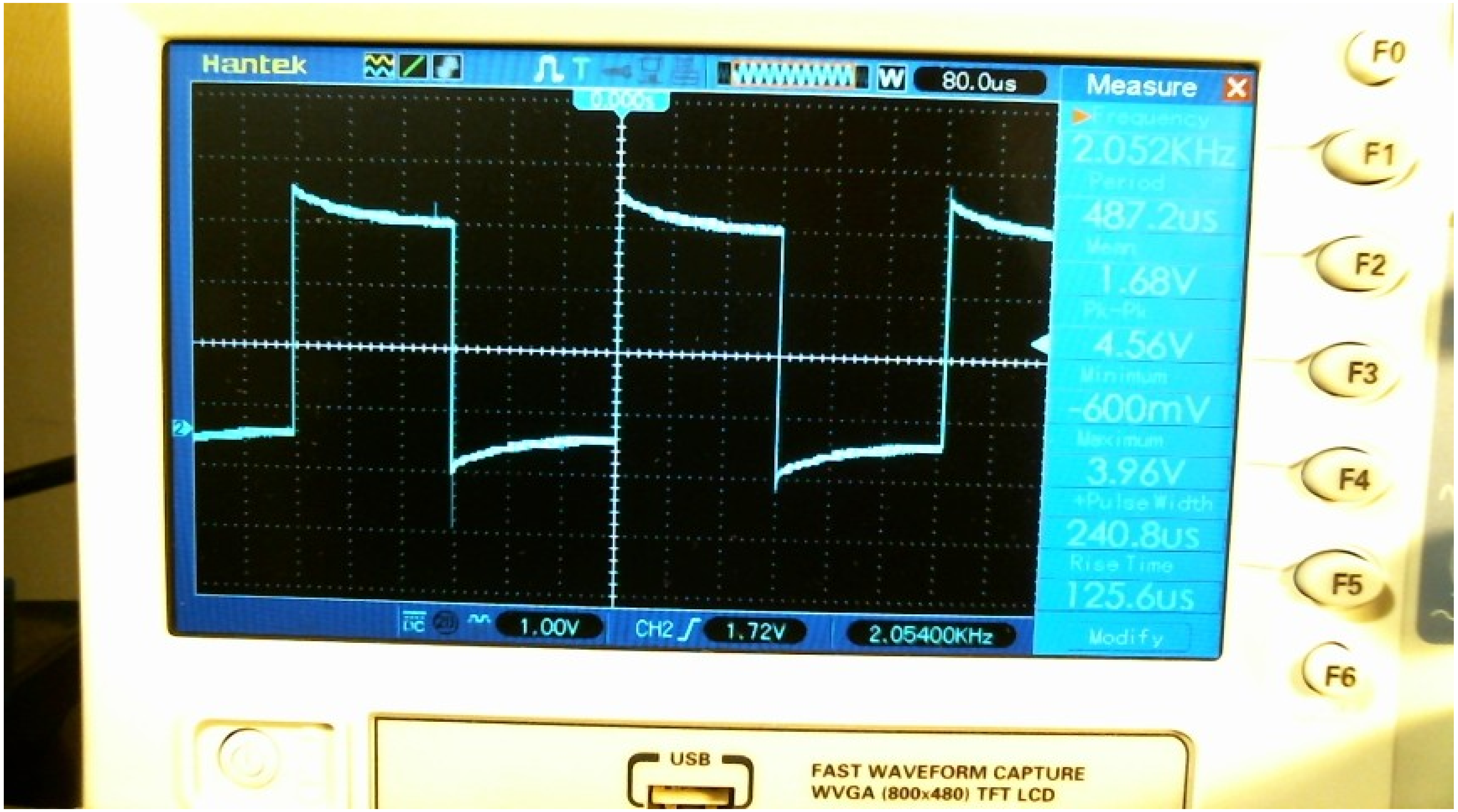
- TAxCTL register
  - Clock source select (TASSEL)
  - Input divider (ID)
  - Mode control (MC) (Note: Switching to Stop mode can be performed at any time)
  - Timer_A clear (TACLR)

TAR

- TAxCCTLn registers
  - Capture mode (CM) (Note: Switching to no capture mode can be performed any time)
  - Capture/compare input select (CCIS) (Note: Switching between GND an VCC can be performed at any time)
  - Synchronize capture source (SCS)
  - Capture mode (CAP)
  - Output mode (OUTMOD)

CCR

- TAxEX0 register
  - Input divider expansion (TAIDEX)



View Timer0_A3 Example: msp430fr243x_ta0_08A.c
M6V2 video