Timer A (0 and 1) and PWM

EE3376

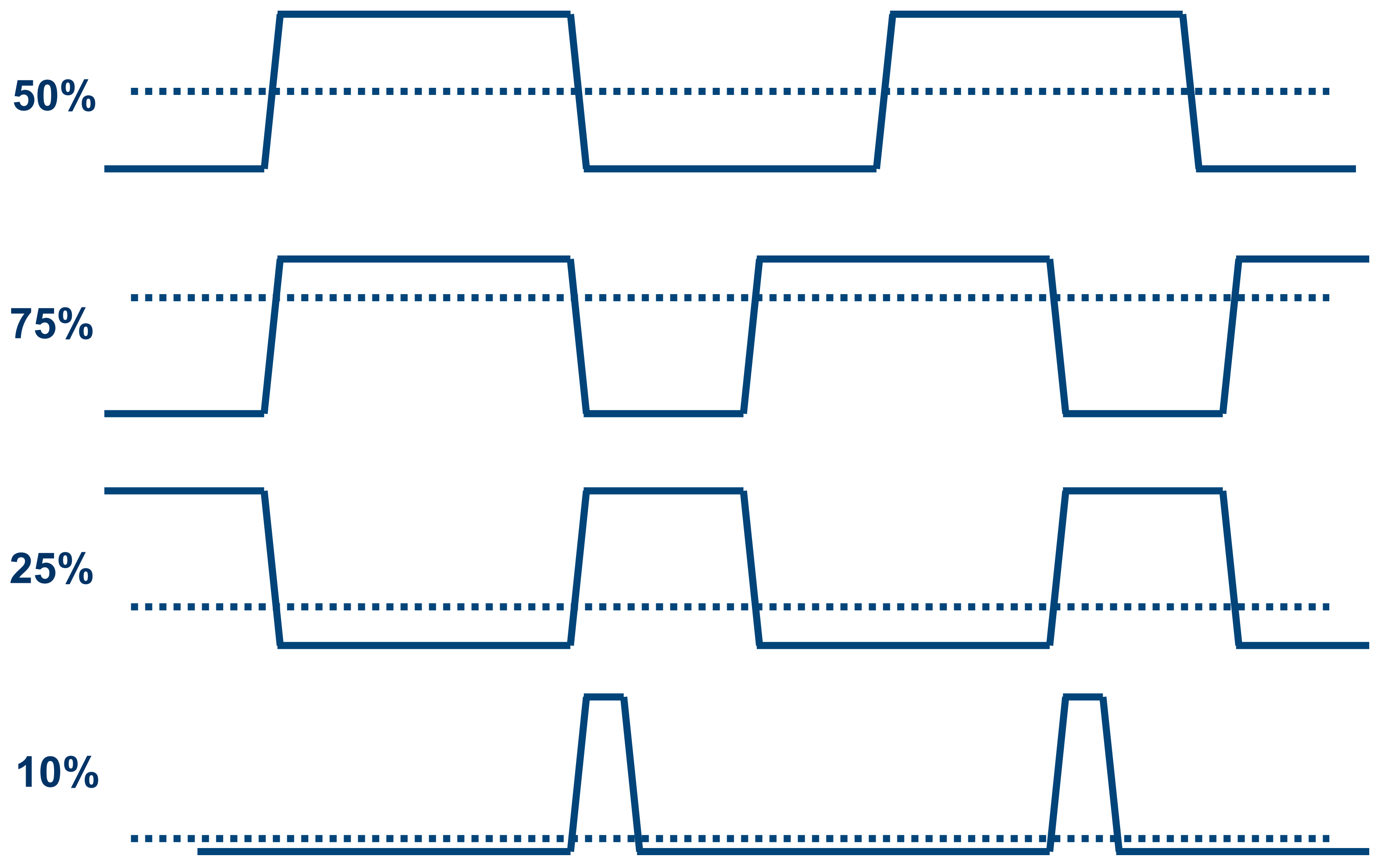
General Peripheral Programming Model

- Each peripheral has a range of addresses in the memory map
  - peripheral has base address (i.e. 0x00A0)
  - each register used in the peripheral has an offset from the base
- some registers are for control
    - bits to enable the peripheral
    - bits to configure specific modes of operation
    - bits to determine level of clock division
- some registers are for status which is generally read-only
    - error conditions are represented with a bit in status register
    - completion status
- some registers are for data
    - data to be transmitted is written to data registers
    - data received can be read from data registers
    - data measured (i.e. ADC) can be read from data registers

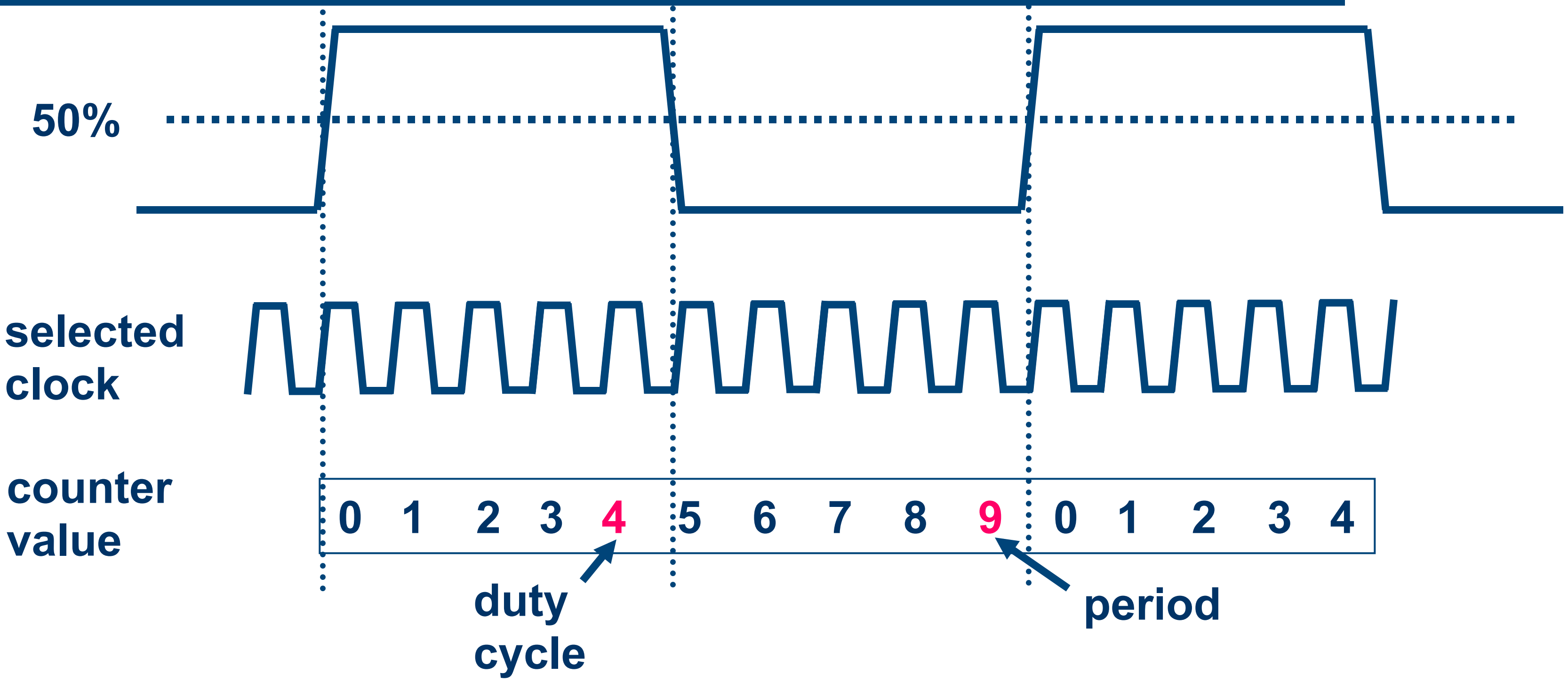
Pulse Width Modulation

- Most commonly used for motor control
  - switching mode for efficiency with transistor drive circuit
- One time configuration – stand alone operation
- Pulse Width Modulation
  - like a poor man's version of Digital to Analog Converter
    - take average value of square wave with variable duty cycle
    - low power output – must buffer with driving circuit for high power applications (motors, etc)
    - can change analog value, but much slower than D/A
- Generates a square wave
  - control of frequency
  - control of duty cycle
  - control of polarity - starts high, ends low OR starts low, ends high
  - control of alignment – left vs. center
  - 8 independent channels on Port P (P for PWM)

Pulse Width Modulation



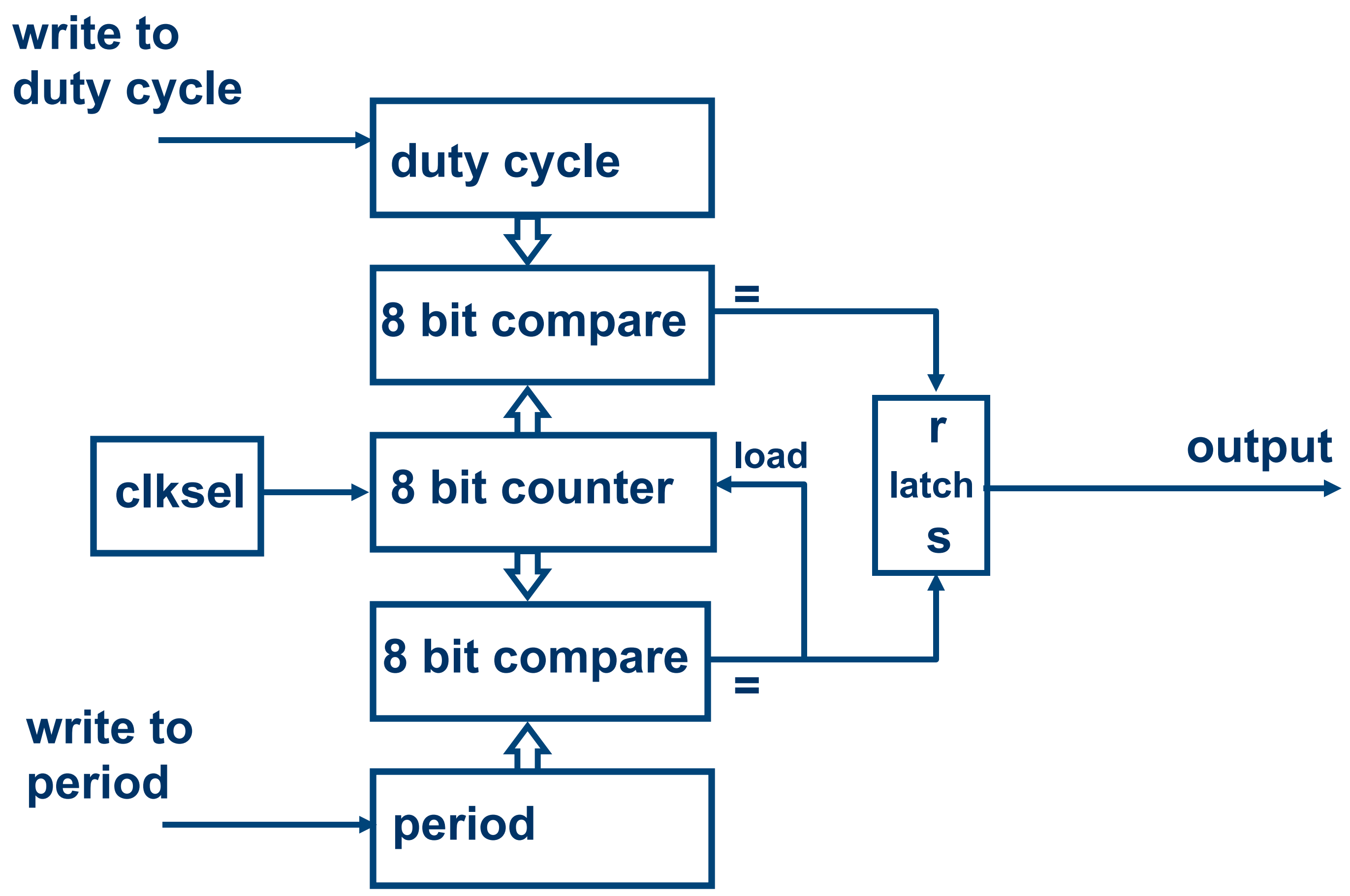
Pulse Width Modulation



PWM frequency = F_{sc} / (period)
= 100KHz / 10 = 10 KHz

PWM duty cycle = ((period – duty cycle) / (period)) * 100%
= ((10 – 5) / 10) * 100% = 50%

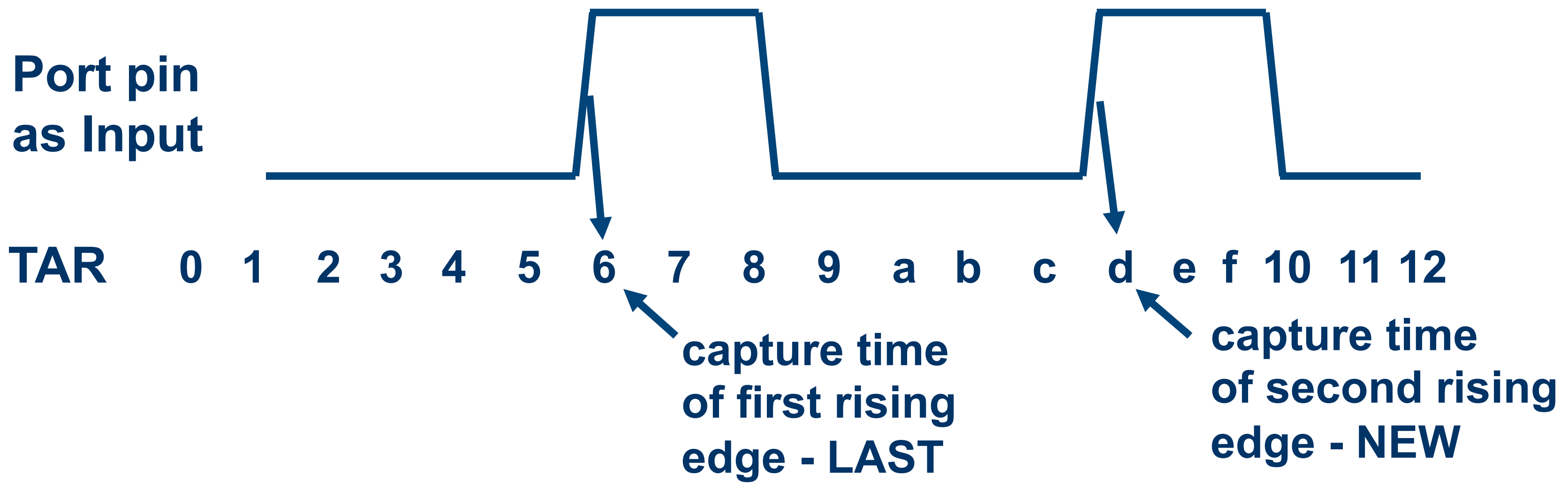
PWM Simple Design Example



Functions of Timers

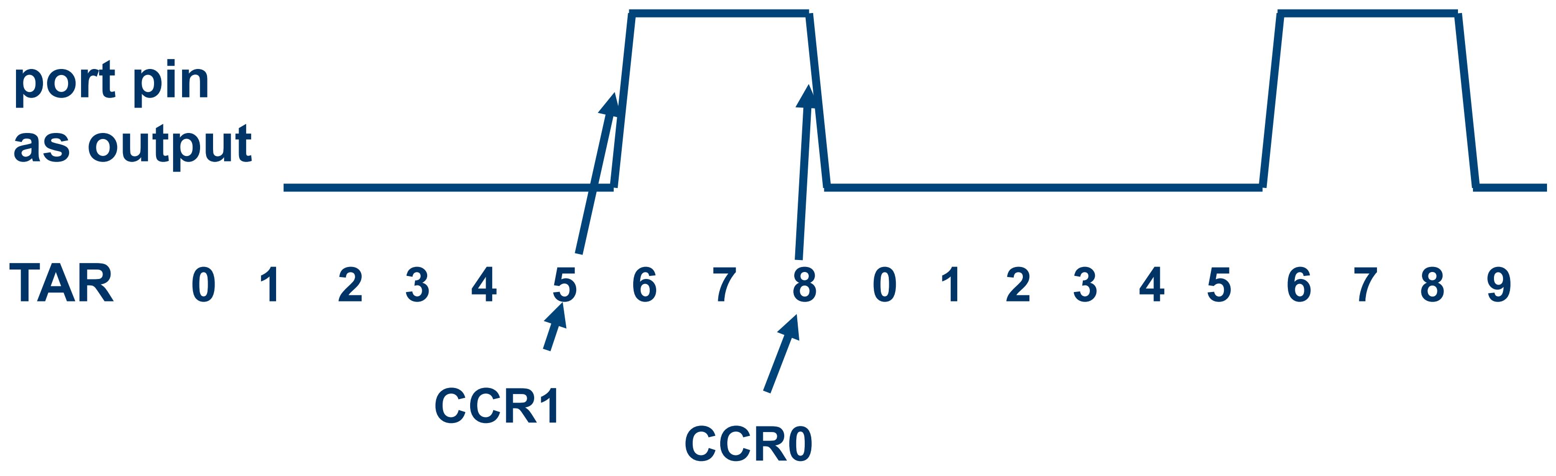
- stop watch
- captures time of external events
  - for instance rising edge of input pin
  - allows for period and pulse width measurements
- creates output waveform
  - rising edge programmed for specific time
  - falling edge programmed for specific time
- pulse accumulations
  - count cycles while input is asserted
  - count the number of rising edges on input
- creates periodic interrupts

Input Capture Mode



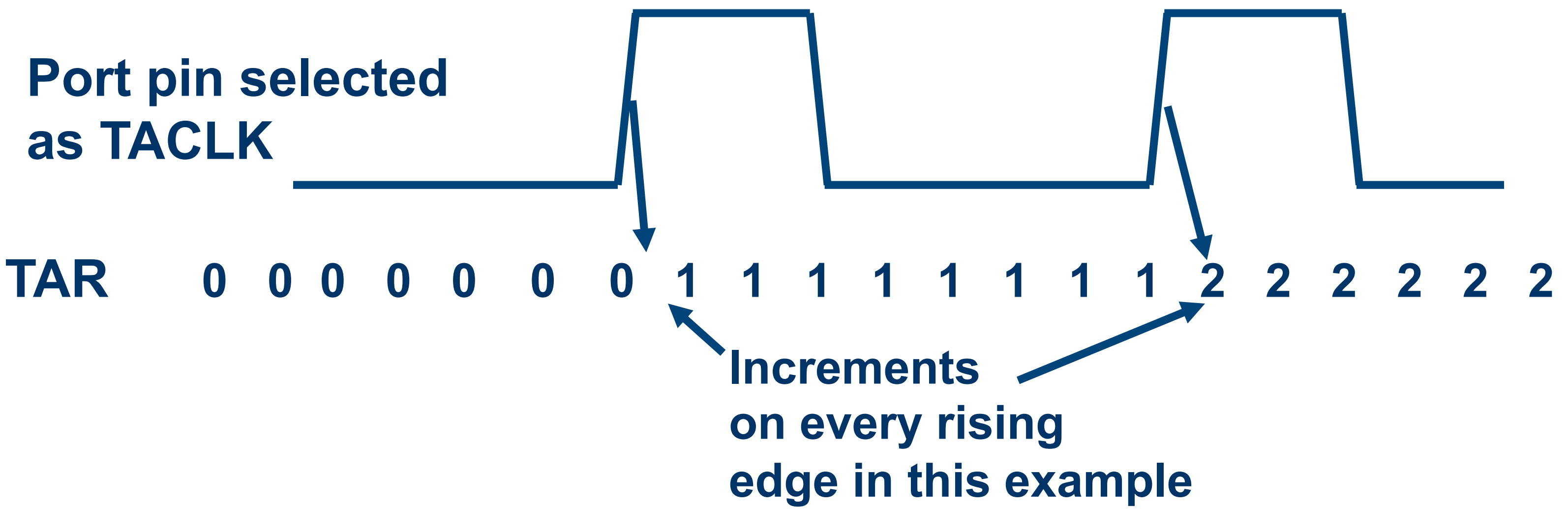
- input pin can be programmed in variety of ways
- In this example,
  - Port input interrupt is enabled
  - Input rising edge causes interrupt – which captures time on TAR
  - TAR is recorded and compared against previous captured value
  - LAST value is subtracted from NEW to get period of waveforms

Output Compare Mode (8 channels)



- Can program edges relative to time in TAR
- Can generate periodic interrupts with same mechanism
- Set OUTMODE if output is used, don't if only needing periodic interrupts.

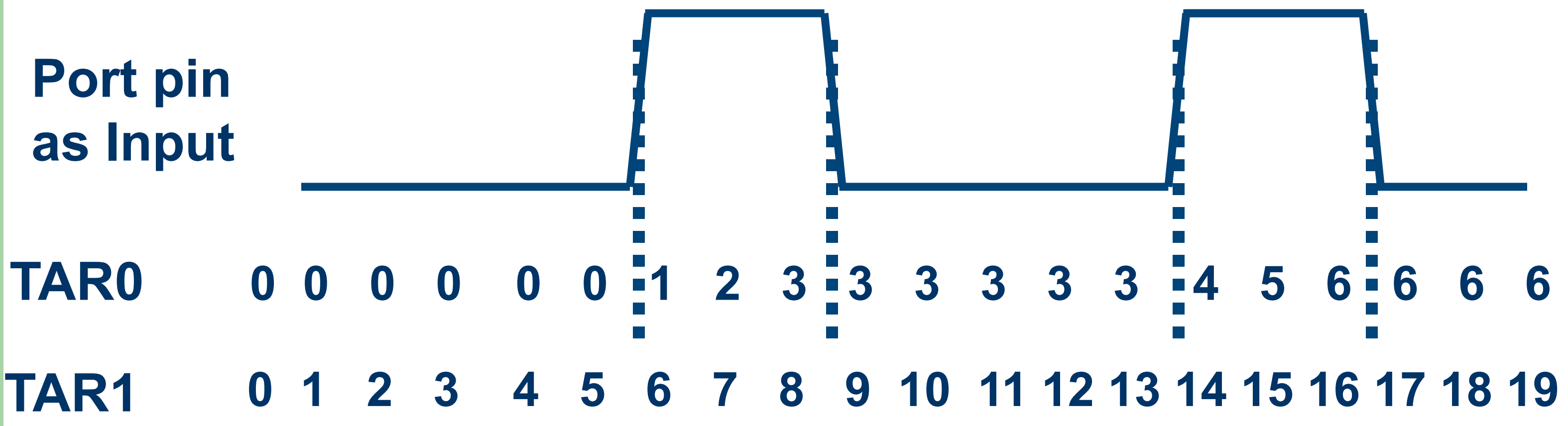
Pulse Accumulation (edge based)



Port pin selected as TACLK
TAR 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2
Increments on every rising edge in this example

- Counts the number of rising edges over time
- For a fixed time, can calculate the average frequency

Pulse Accumulation (gated time)



Port pin as Input
TAR0 0 0 0 0 0 0 1 2 3 3 3 3 3 3 4 5 6 6 6 6
TAR1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

- Interrupt for rising and falling edges of port 1
- One interrupt enables TAR and one disables
- Use other timer to measure straight time for comparison
- TAR0 / TAR1 is ratio of on versus off.

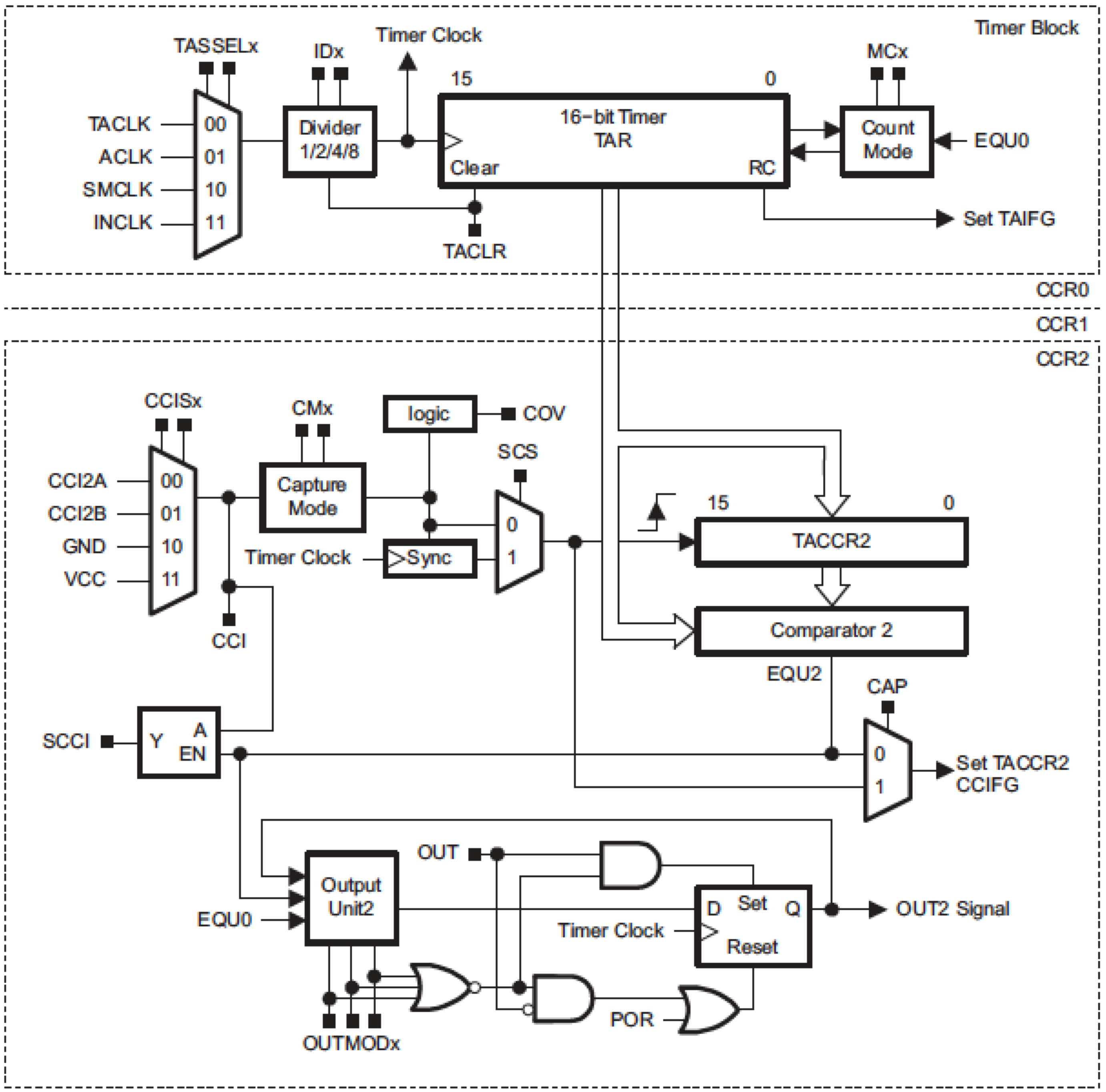


Figure 12-1. Timer_A Block Diagram

MSP430 User’s Manual

Table 12-3. Timer_A3 Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control	TACTL	Read/write	0160h	Reset with POR
Timer_A counter	TAR	Read/write	0170h	Reset with POR
Timer_A capture/compare control 0	TACCTL0	Read/write	0162h	Reset with POR
Timer_A capture/compare 0	TACCR0	Read/write	0172h	Reset with POR
Timer_A capture/compare control 1	TACCTL1	Read/write	0164h	Reset with POR
Timer_A capture/compare 1	TACCR1	Read/write	0174h	Reset with POR
Timer_A capture/compare control 2	TACCTL2(1)	Read/write	0166h	Reset with POR
Timer_A capture/compare 2	TACCR2(1)	Read/write	0176h	Reset with POR
Timer_A interrupt vector	TAIV	Read only	012Eh	Reset with POR

(1) Not present on MSP430 devices with Timer_A2 like MSP430F20xx and other devices.

12.3.1 TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Unused	Bits 15-10	Unused	
TASSELx	Bits 9-8	Timer_A clock source select	
		00	TACLK
		01	ACLK
		10	SMCLK
		11	INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock.	
		00	/1
		01	/2
		10	/4
		11	/8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.	
		00	Stop mode: the timer is halted.
		01	Up mode: the timer counts up to TACCR0.
		10	Continuous mode: the timer counts up to 0FFFFh.
		11	Up/down mode: the timer counts up to TACCR0 then down to 0000h.
Unused	Bit 3	Unused	
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.	
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request.	
		0	Interrupt disabled
		1	Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag	
		0	No interrupt pending
		1	Interrupt pending

Select clock to run counter

Clock divider of selected clock

Counter mode

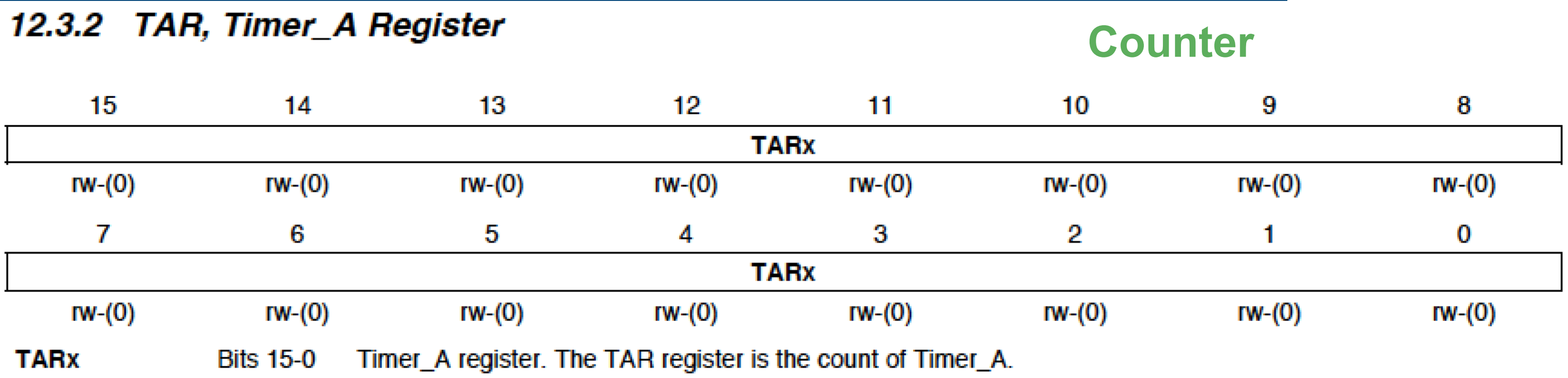
Interrupt flag and enable
Must read TAIV to determine which is responsible and to clear the bit. IRQ occurs if TAR == 0

MSP430 User’s Manual

In continuous mode, TAR overflow details

Source	Freq	Divide	Res	Freq	Period
SMCLK	16MHz	1	1/16 uS	240 Hz	4ms
SMCLK	1MHz	1	1 uS	240 Hz	66 mS
SMCLK	1MHz	8	8 uS	2 Hz	0.5 S
ACLK	32KHz	1	31 uS	½ Hz	2 S
ACLK	32KHz	8	240 uS	1/16 Hz	16 S

MSP430 User's Manual



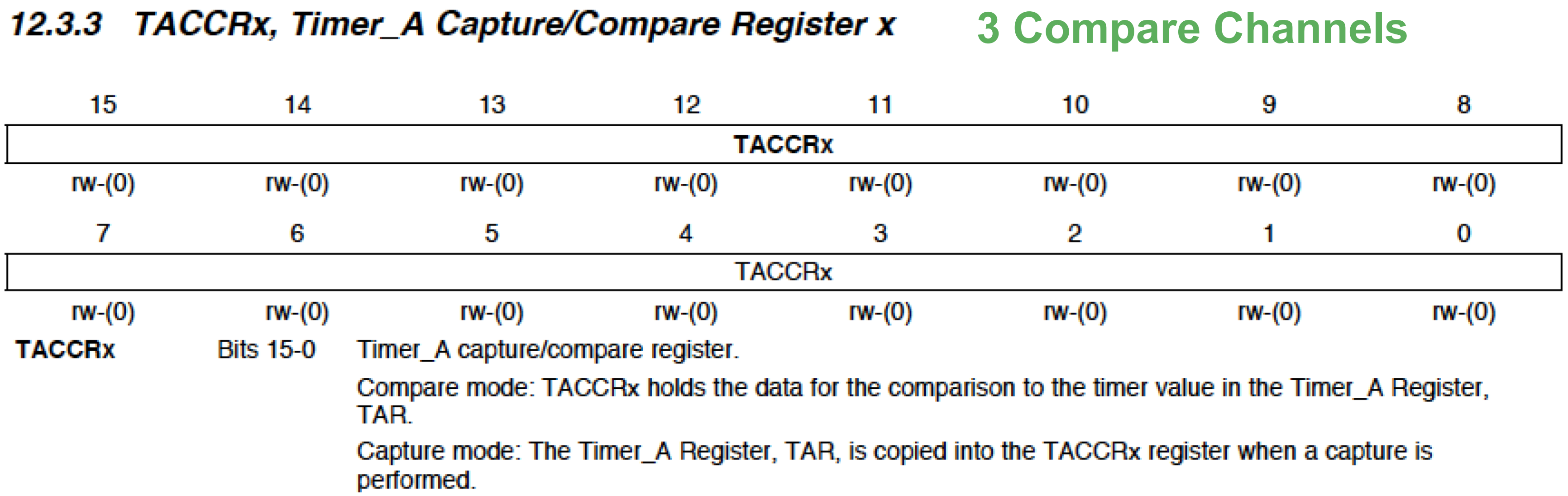
12.3.2 TAR, Timer_A Register

Counter

15	14	13	12	11	10	9	8
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TARx	Bits 15-0	Timer_A register. The TAR register is the count of Timer_A.



12.3.3 TACCRx, Timer_A Capture/Compare Register x

3 Compare Channels

15	14	13	12	11	10	9	8
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TACCRx	Bits 15-0	Timer_A capture/compare register. Compare mode: TACCRx holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TACCRx register when a capture is performed.

12.3.4 TACCTLx, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)

7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CMx	Bit 15-14	Capture mode	
		00	No capture
		01	Capture on rising edge
		10	Capture on falling edge
		11	Capture on both rising and falling edges
CCISx	Bit 13-12	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.	
		00	CCIxA
		01	CCIxB
		10	GND
		11	V_{CC}
SCS	Bit 11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.	
		0	Asynchronous capture
		1	Synchronous capture
SCCI	Bit 10	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit	
Unused	Bit 9	Unused. Read only. Always read as 0.	
CAP	Bit 8	Capture mode	
		0	Compare mode
		1	Capture mode

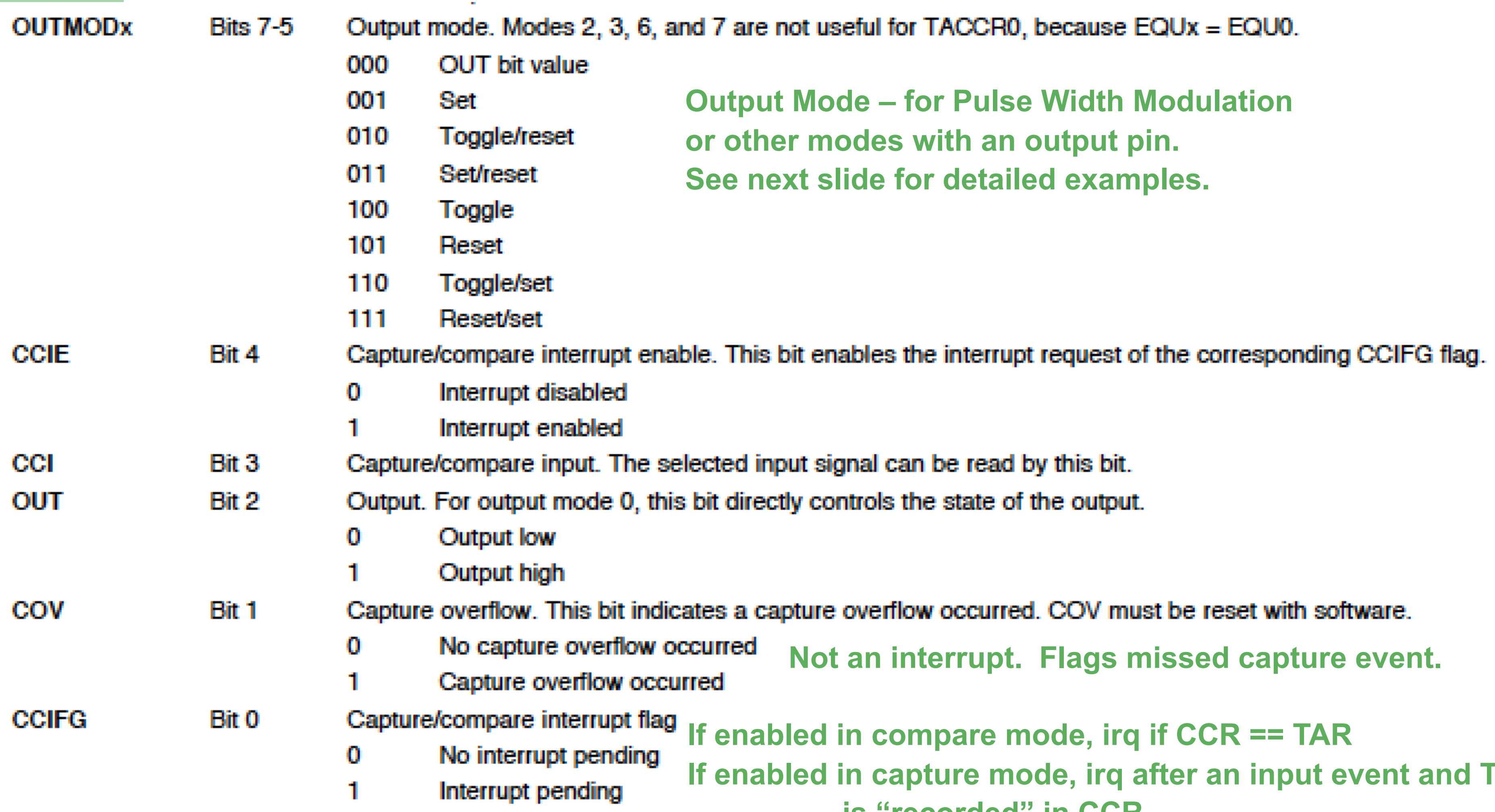
Capture mode allows TAR to be captured at a specific event – example paddle wheel rotation for wind speed calculation

Capture mode input select – two IO possible. Gnd and Vdd are for software enabled capture.

Input should always be synchronized

Capture Vs. Compare mode select

TACCTLx continued



OUTMODx	Bits 7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0.	
		000	OUT bit value
		001	Set
		010	Toggle/reset
		011	Set/reset
		100	Toggle
		101	Reset
		110	Toggle/set
		111	Reset/set
CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.	
		0	Interrupt disabled
		1	Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.	
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output.	
		0	Output low
		1	Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.	
		0	No capture overflow occurred
		1	Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag	
		0	No interrupt pending
		1	Interrupt pending

Output Mode – for Pulse Width Modulation or other modes with an output pin.
See next slide for detailed examples.

Not an interrupt. Flags missed capture event.

If enabled in compare mode, irq if CCR == TAR
If enabled in capture mode, irq after an input event and TAR is “recorded” in CCR.

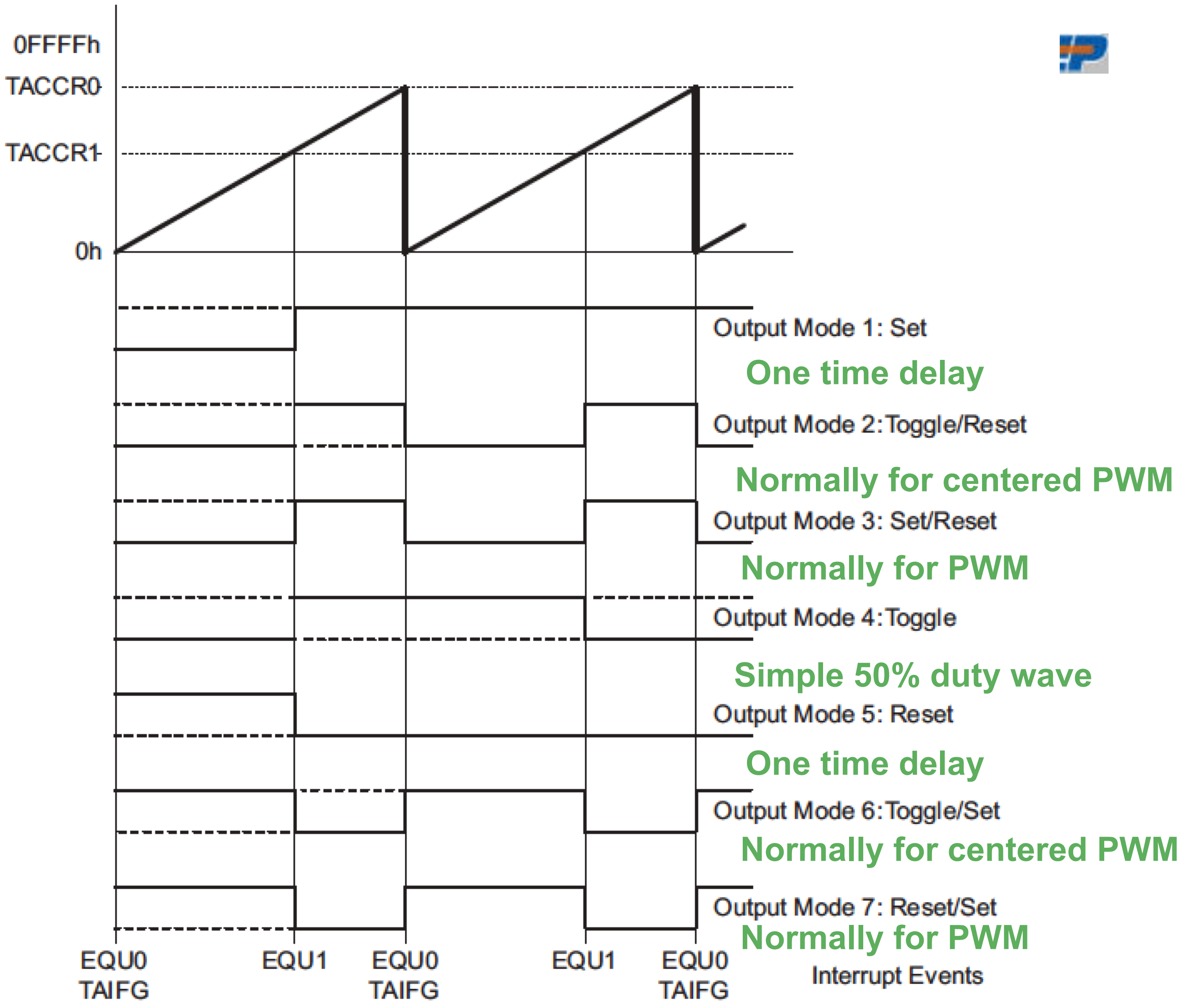


Figure 12-12. Output Example—Timer in Up Mode

Up down mode

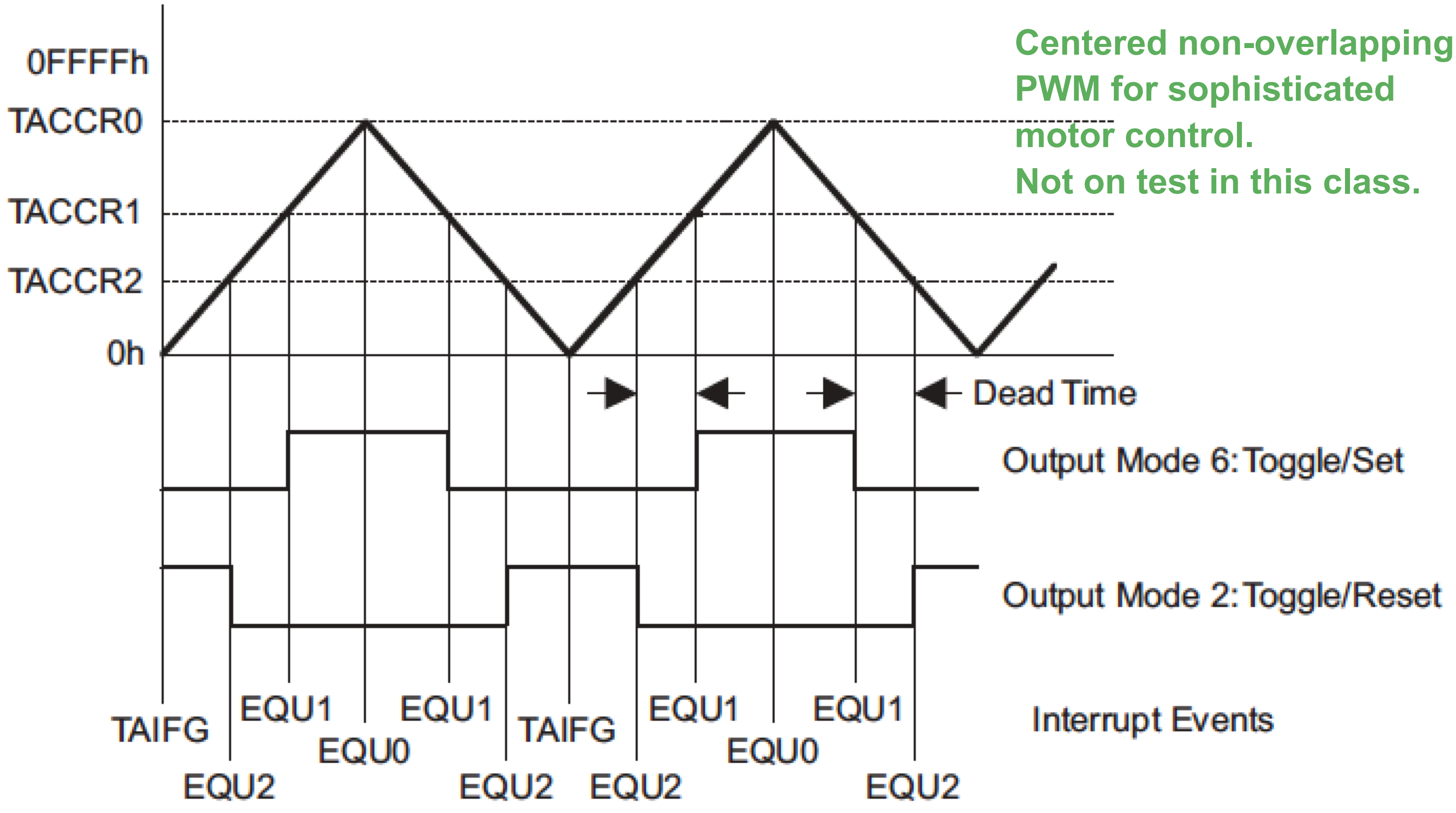
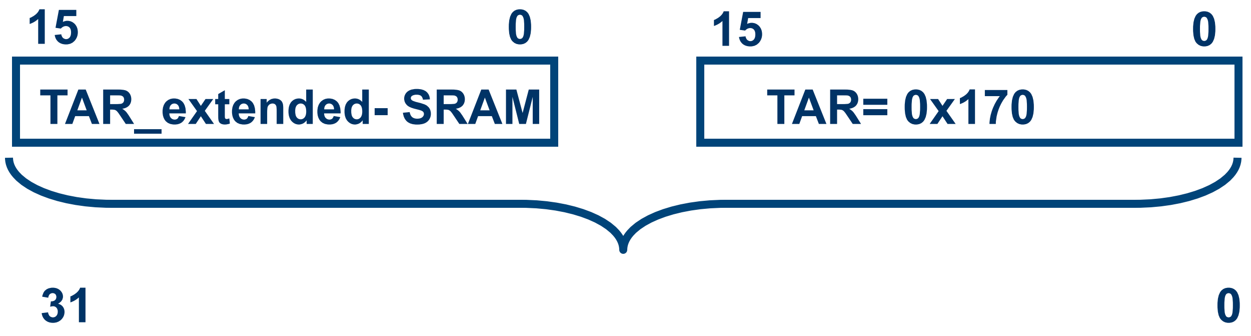


Figure 12-9. Output Unit in Up/Down Mode

Centered non-overlapping PWM for sophisticated motor control.
Not on test in this class.

32 bit counter extension example

- If 16 bit counter is running at 1 MHz, rollover occurs every 2^{16} * (1 / 1MHz) = 64 milliseconds
- By extending to 32 bits, rollover period is much longer – 2^{32} * (1 / 1MHz) ~ 4000 seconds
- Set overflow interrupt service routine to increment a global variable - 16 bit TAR_extended - which represents the upper 16 bits of a 32 bit word.

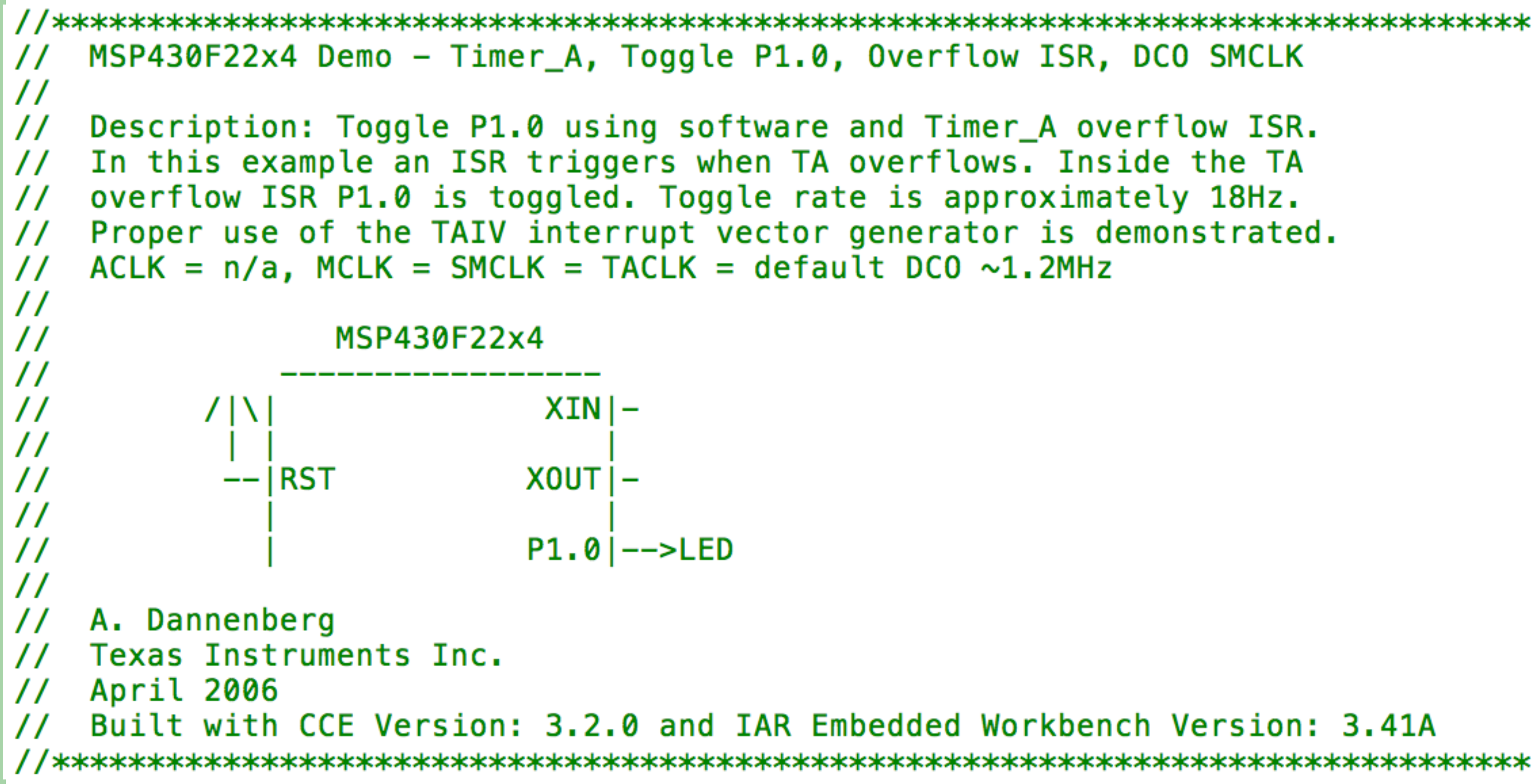


Note on programming MSP430

```
P1DIR  |= 0x41;                    // P1.0 and P1.6 to output
P1SEL  |= 0x41;
P1SEL2 |= 0x01;

CCR0 = 1000;            //PWM Period - Freq = (SMCLK freq)/(CCR0 value)
                          //eg,SMCLK = 1MHZ so 1MHZ/1000 = 1Khz is the PWM Freq
CCTL1 = OUTMOD_7;         //CCR1 toggle/set
CCTL2 = OUTMOD_7;         //CCR2 toggle/reset
CCR1 = 0;                 //CCR1 PWM duty cycle
TACTL = TASSEL_2 + MC_1;
```

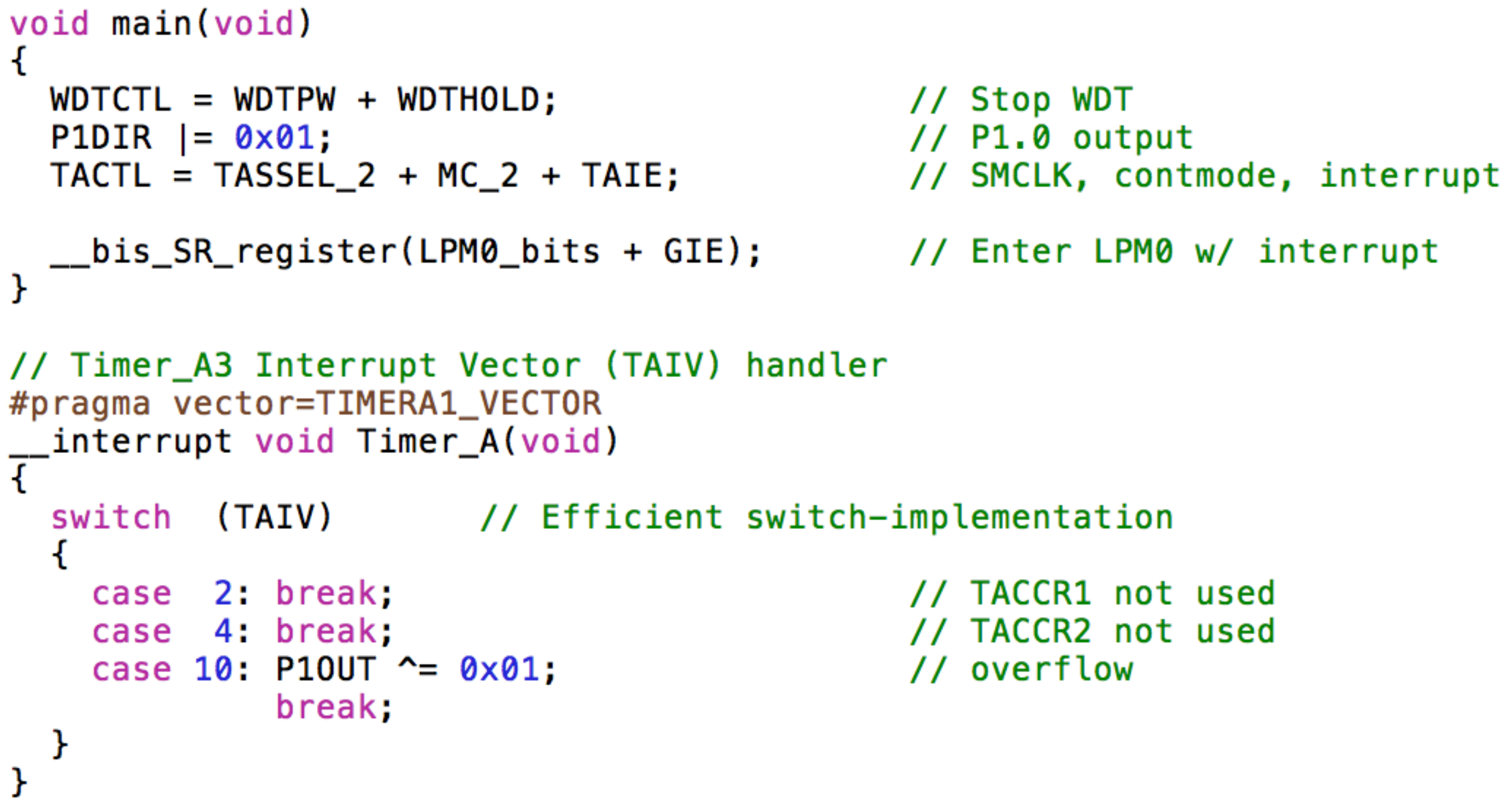
Typical programming style is to use header file abbreviations for registers and bit patterns.
TACTL is the TA0CTL register at 0x160
TASSEL_2 is (2*0x0100u)) = 0x0200 unsigned – picks the SMCLK
MC_1 = (0x0020) – picks "up to CCR0" mode.
TASSEL_2 | MC_1 = 0x0220 which programs both the mode and clock.

msp430x22x4_ta_03



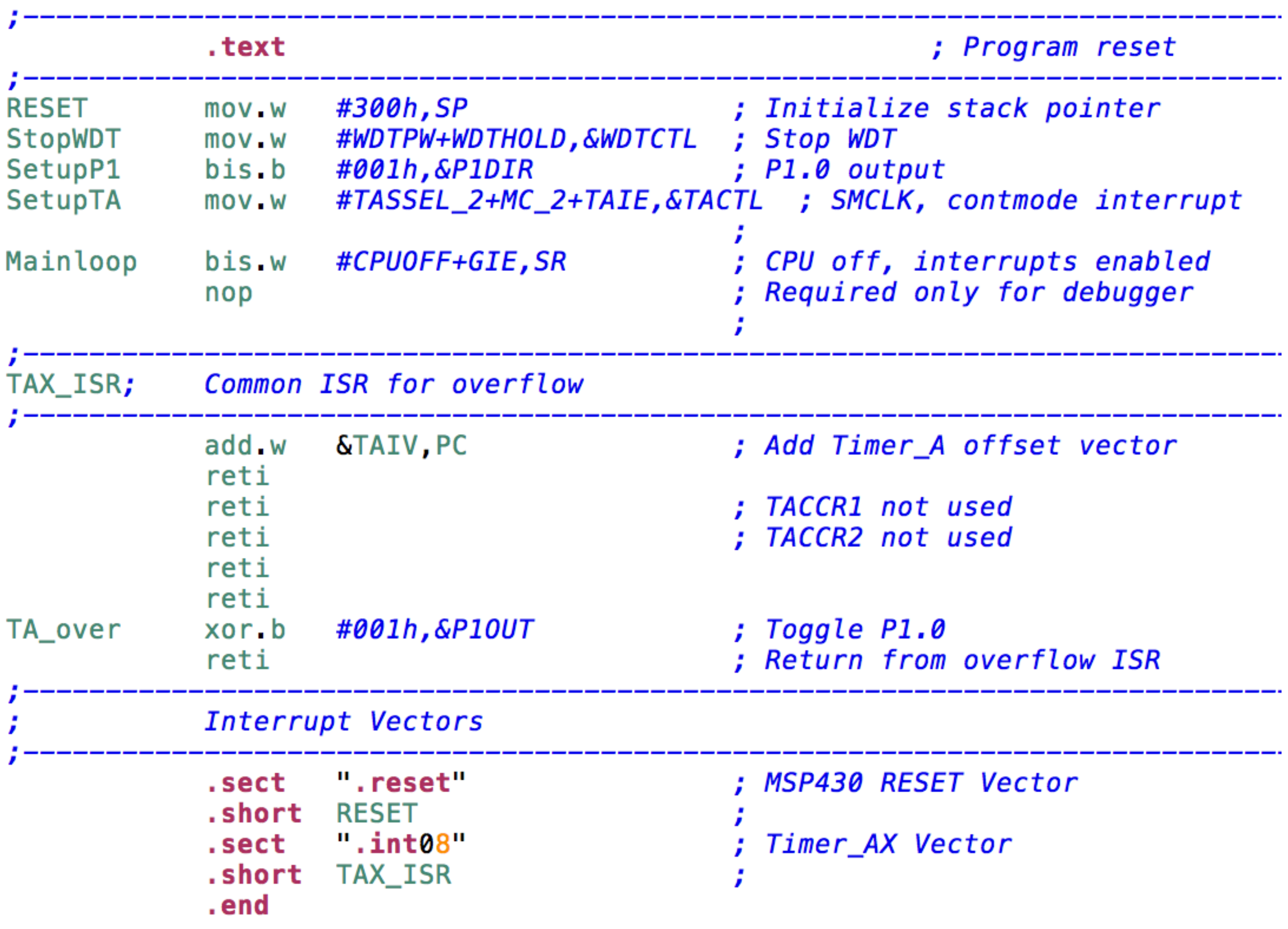
```
//******************************************************************************
//  MSP430F22x4 Demo - Timer_A, Toggle P1.0, Overflow ISR, DCO SMCLK
//
//  Description: Toggle P1.0 using software and Timer_A overflow ISR.
//  In this example an ISR triggers when TA overflows. Inside the TA
//  overflow ISR P1.0 is toggled. Toggle rate is approximately 18Hz.
//  Proper use of the TAIV interrupt vector generator is demonstrated.
//  ACLK = n/a, MCLK = SMCLK = TACLK = default DCO ~1.2MHz
//
//                MSP430F22x4
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |             P1.0|-->LED
//
//  A. Dannenberg
//  Texas Instruments Inc.
//  April 2006
//  Built with CCE Version: 3.2.0 and IAR Embedded Workbench Version: 3.41A
//******************************************************************************
```

msp430x22x4_ta_03



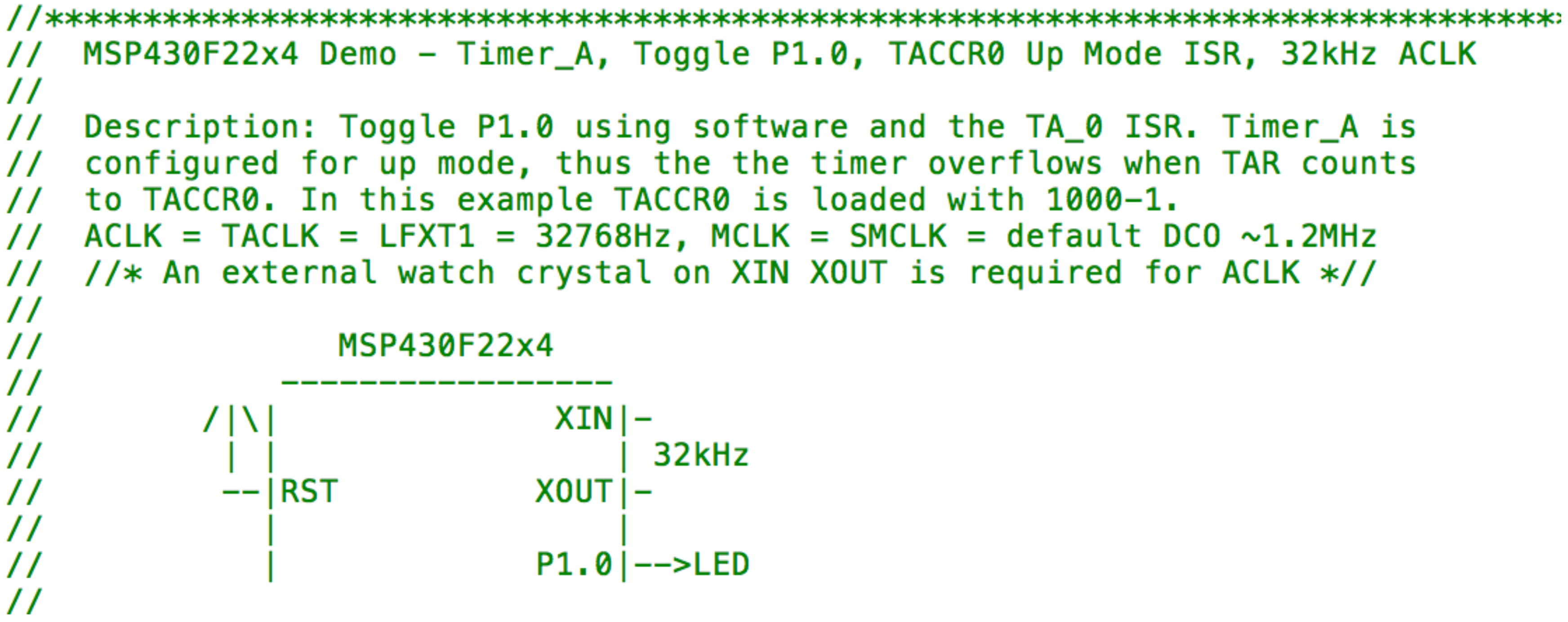
```c
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x01;                            // P1.0 output
  TACTL = TASSEL_2 + MC_2 + TAIE;           // SMCLK, contmode, interrupt

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0 w/ interrupt
}

// Timer_A3 Interrupt Vector (TAIV) handler
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
  switch  (TAIV)      // Efficient switch-implementation
  {
    case  2: break;                         // TACCR1 not used
    case  4: break;                         // TACCR2 not used
    case 10: P1OUT ^= 0x01;                 // overflow
             break;
  }
}
```

msp430x22x4_ta_03



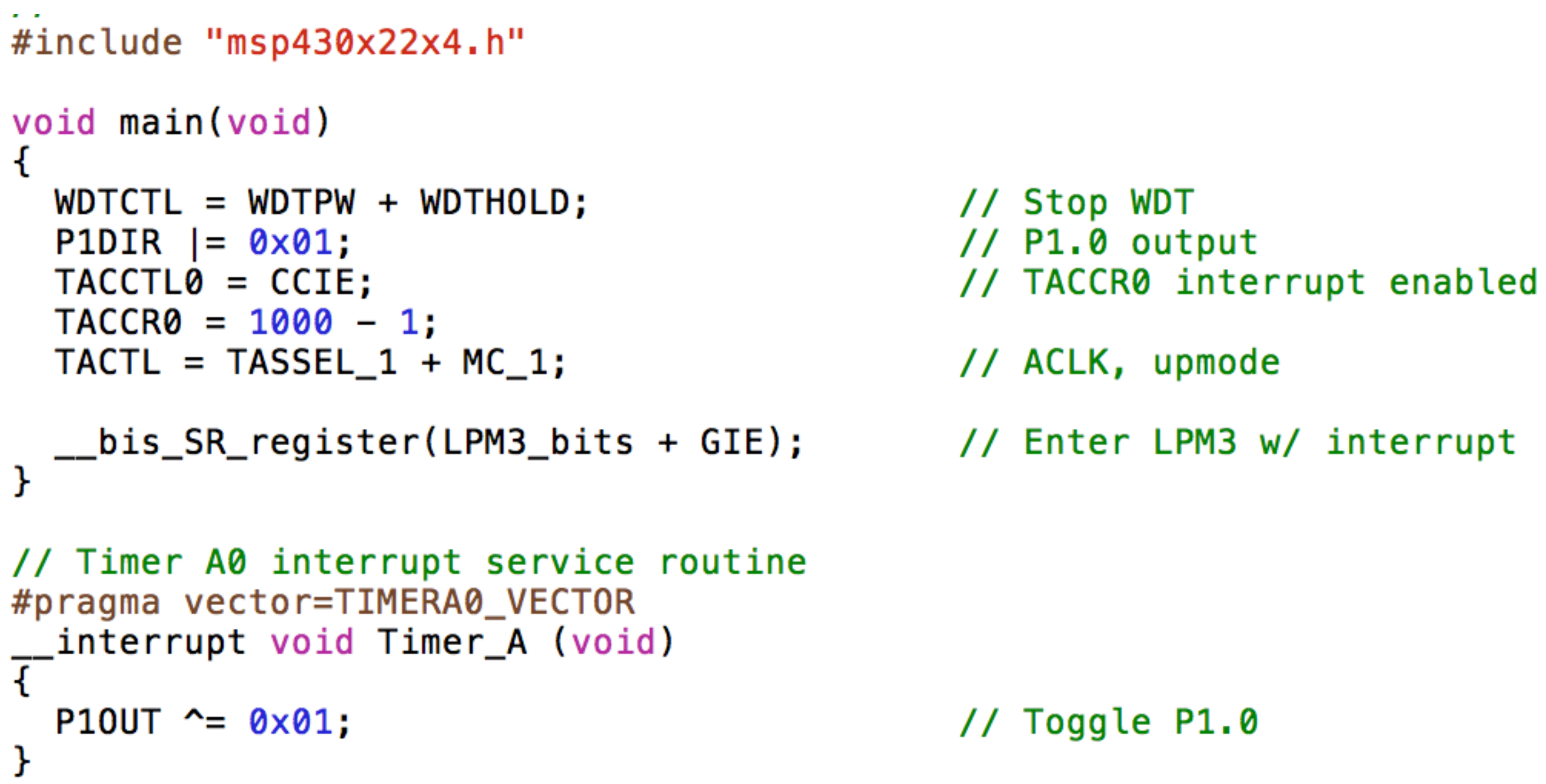
```
;-------------------------------------------------------------------------------
            .text                                           ; Program reset
;-------------------------------------------------------------------------------
RESET       mov.w   #300h,SP                        ; Initialize stack pointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; Stop WDT
SetupP1     bis.b   #001h,&P1DIR                    ; P1.0 output
SetupTA     mov.w   #TASSEL_2+MC_2+TAIE,&TACTL   ; SMCLK, contmode interrupt
                                                    ;
Mainloop    bis.w   #CPUOFF+GIE,SR                  ; CPU off, interrupts enabled
            nop                                     ; Required only for debugger
                                                    ;
;-------------------------------------------------------------------------------
TAX_ISR;    Common ISR for overflow
;-------------------------------------------------------------------------------
            add.w   &TAIV,PC                        ; Add Timer_A offset vector
            reti
            reti                                    ; TACCR1 not used
            reti                                    ; TACCR2 not used
            reti
            reti
TA_over     xor.b   #001h,&P1OUT                    ; Toggle P1.0
            reti                                    ; Return from overflow ISR
;-------------------------------------------------------------------------------
;           Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                        ; MSP430 RESET Vector
            .short  RESET                           ;
            .sect   ".int08"                        ; Timer_AX Vector
            .short  TAX_ISR                         ;
            .end
```

msp430x22x4_ta_05



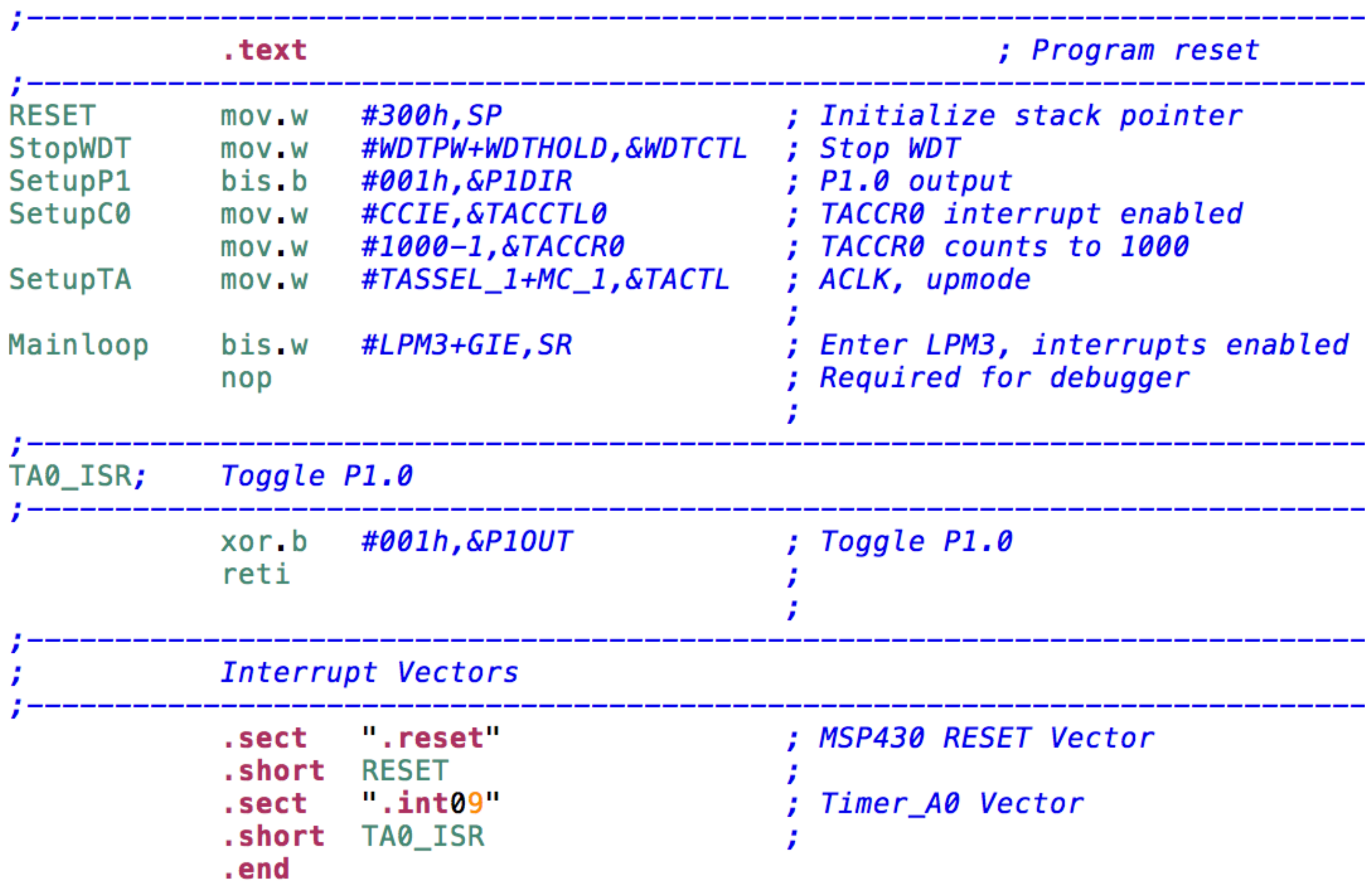
```
//*****************************************************************************
//  MSP430F22x4 Demo - Timer_A, Toggle P1.0, TACCR0 Up Mode ISR, 32kHz ACLK
//
//  Description: Toggle P1.0 using software and the TA_0 ISR. Timer_A is
//  configured for up mode, thus the the timer overflows when TAR counts
//  to TACCR0. In this example TACCR0 is loaded with 1000-1.
//  ACLK = TACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO ~1.2MHz
//  //* An external watch crystal on XIN XOUT is required for ACLK *//
//
//               MSP430F22x4
//            -----------------
//        /|\|              XIN|-
//         | |                 | 32kHz
//         --|RST          XOUT|-
//           |                 |
//           |             P1.0|-->LED
//
```

msp430x22x4_ta_05



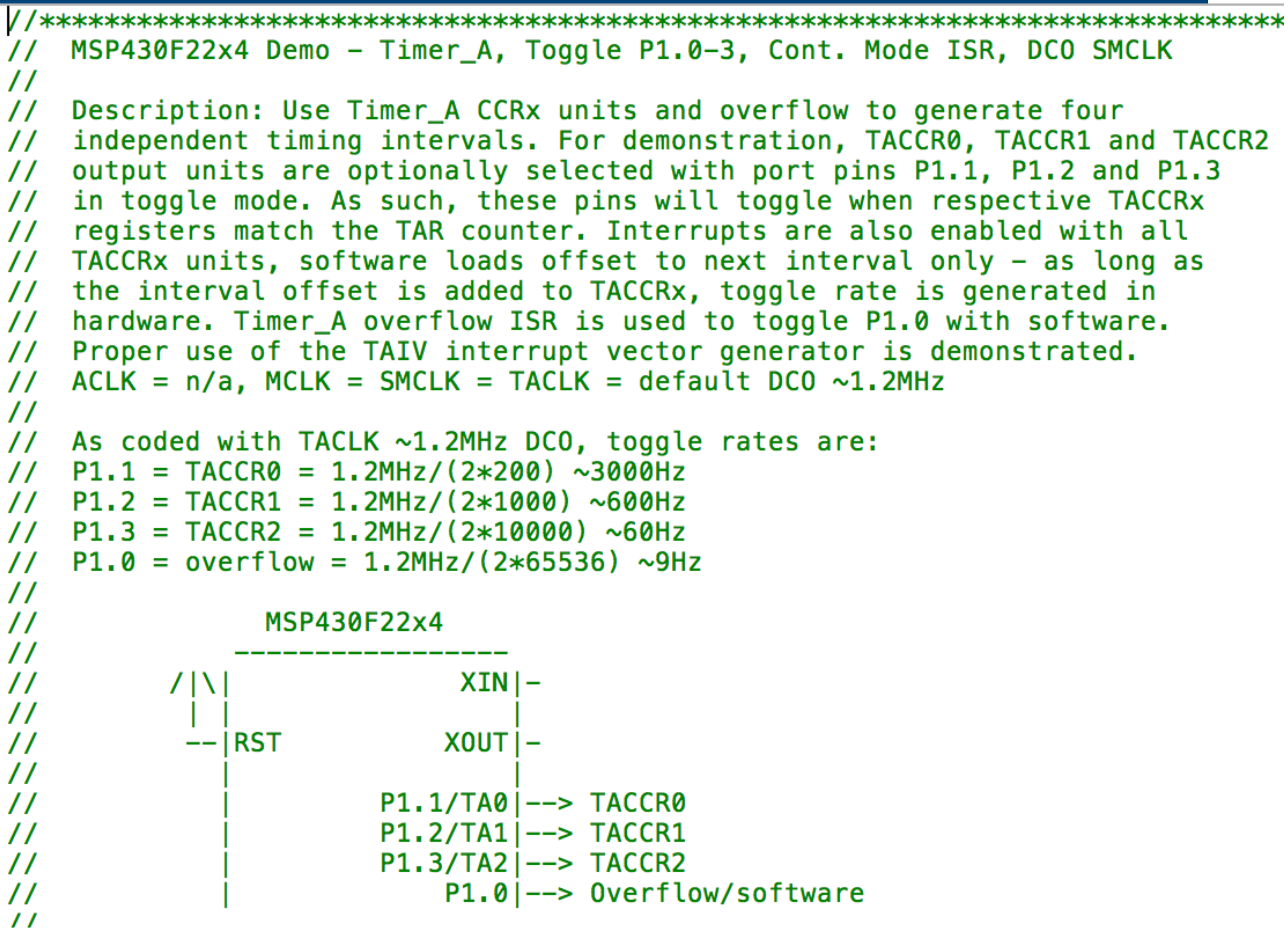
```c
#include "msp430x22x4.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x01;                            // P1.0 output
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 1000 - 1;
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

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

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                            // Toggle P1.0
}
```

msp430x22x4_ta_05



```
;-------------------------------------------------------------------------------
            .text                                   ; Program reset
;-------------------------------------------------------------------------------
RESET       mov.w   #300h,SP                        ; Initialize stack pointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; Stop WDT
SetupP1     bis.b   #001h,&P1DIR                    ; P1.0 output
SetupC0     mov.w   #CCIE,&TACCTL0                  ; TACCR0 interrupt enabled
            mov.w   #1000-1,&TACCR0                 ; TACCR0 counts to 1000
SetupTA     mov.w   #TASSEL_1+MC_1,&TACTL           ; ACLK, upmode
                                                    ;
Mainloop    bis.w   #LPM3+GIE,SR                    ; Enter LPM3, interrupts enabled
            nop                                     ; Required for debugger
                                                    ;
;-------------------------------------------------------------------------------
TA0_ISR;    Toggle P1.0
;-------------------------------------------------------------------------------
            xor.b   #001h,&P1OUT                    ; Toggle P1.0
            reti                                    ;
                                                    ;
;-------------------------------------------------------------------------------
;           Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                        ; MSP430 RESET Vector
            .short  RESET                           ;
            .sect   ".int09"                        ; Timer_A0 Vector
            .short  TA0_ISR                         ;
            .end
```

msp430x22x4_ta_07

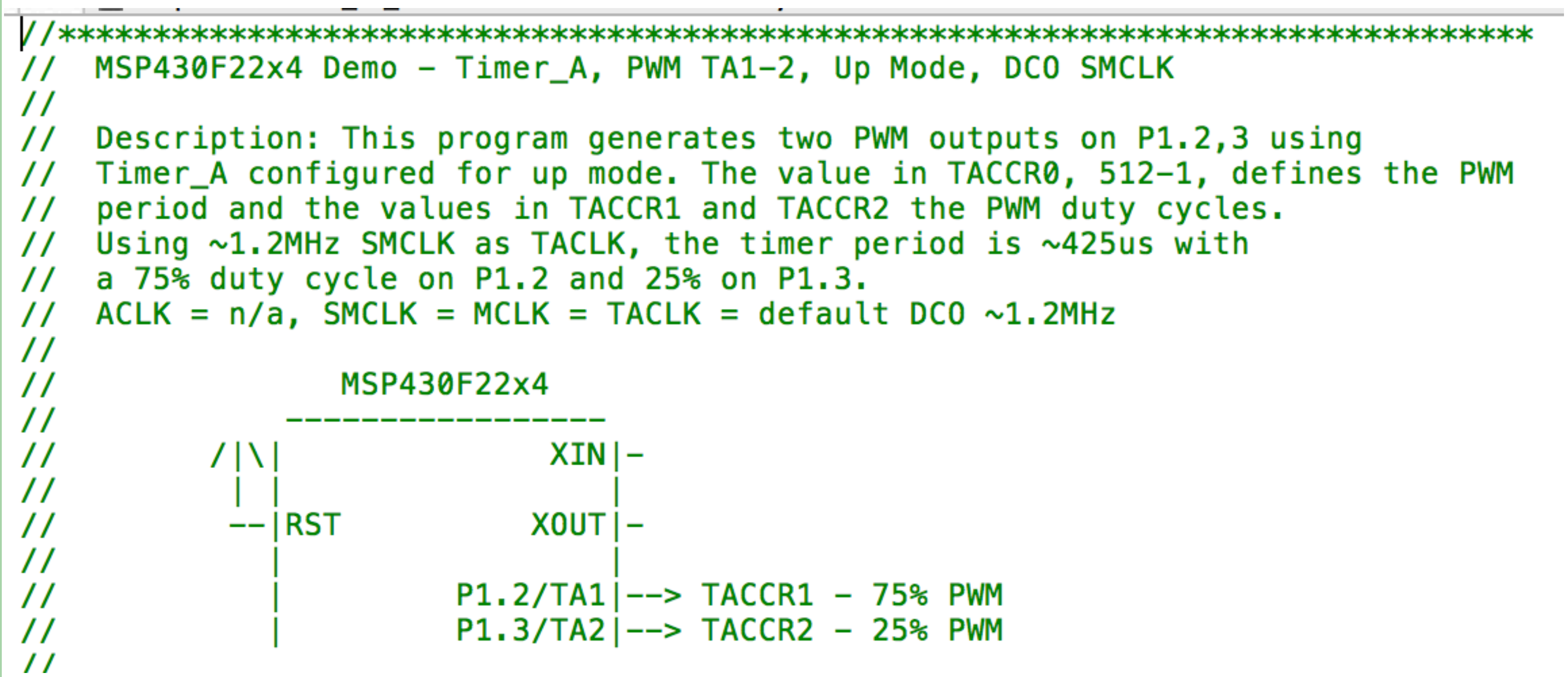


```
//*****************************************************************************
//  MSP430F22x4 Demo - Timer_A, Toggle P1.0-3, Cont. Mode ISR, DCO SMCLK
//
//  Description: Use Timer_A CCRx units and overflow to generate four
//  independent timing intervals. For demonstration, TACCR0, TACCR1 and TACCR2
//  output units are optionally selected with port pins P1.1, P1.2 and P1.3
//  in toggle mode. As such, these pins will toggle when respective TACCRx
//  registers match the TAR counter. Interrupts are also enabled with all
//  TACCRx units, software loads offset to next interval only - as long as
//  the interval offset is added to TACCRx, toggle rate is generated in
//  hardware. Timer_A overflow ISR is used to toggle P1.0 with software.
//  Proper use of the TAIV interrupt vector generator is demonstrated.
//  ACLK = n/a, MCLK = SMCLK = TACLK = default DCO ~1.2MHz
//
//  As coded with TACLK ~1.2MHz DCO, toggle rates are:
//  P1.1 = TACCR0 = 1.2MHz/(2*200) ~3000Hz
//  P1.2 = TACCR1 = 1.2MHz/(2*1000) ~600Hz
//  P1.3 = TACCR2 = 1.2MHz/(2*10000) ~60Hz
//  P1.0 = overflow = 1.2MHz/(2*65536) ~9Hz
//
//              MSP430F22x4
//            -----------------
//        /|\|              XIN|-
//         | |                 |
//         --|RST          XOUT|-
//           |                 |
//           |         P1.1/TA0|--> TACCR0
//           |         P1.2/TA1|--> TACCR1
//           |         P1.3/TA2|--> TACCR2
//           |             P1.0|--> Overflow/software
//
```

msp430x22x4_ta_07

```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                    // Stop WDT
  P1SEL |= 0x0E;                               // P1.1 - P1.3 option select
  P1DIR |= 0x0F;                               // P1.0 - P1.3 outputs
  TACCTL0 = OUTMOD_4 + CCIE;                   // TACCR0 toggle, interrupt enabled
  TACCTL1 = OUTMOD_4 + CCIE;                   // TACCR1 toggle, interrupt enabled
  TACCTL2 = OUTMOD_4 + CCIE;                   // TACCR2 toggle, interrupt enabled
  TACTL = TASSEL_2 +  MC_2 + TAIE;             // SMCLK, Contmode, int enabled

  __bis_SR_register(LPM0_bits + GIE);          // Enter LPM0 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A0(void)
{
  TACCR0 += 200;                               // Add Offset to TACCR0
}

// Timer_A3 Interrupt Vector (TAIV) handler
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1(void)
{
  switch (TAIV)        // Efficient switch-implementation
  {
    case  2: TACCR1 += 1000;                   // Add Offset to TACCR1
             break;
    case  4: TACCR2 += 10000;                  // Add Offset to TACCR2
             break;
    case 10: P1OUT ^= 0x01;                    // Timer_A3 overflow
             break;
  }
}
```

msp430x22x4_ta_16

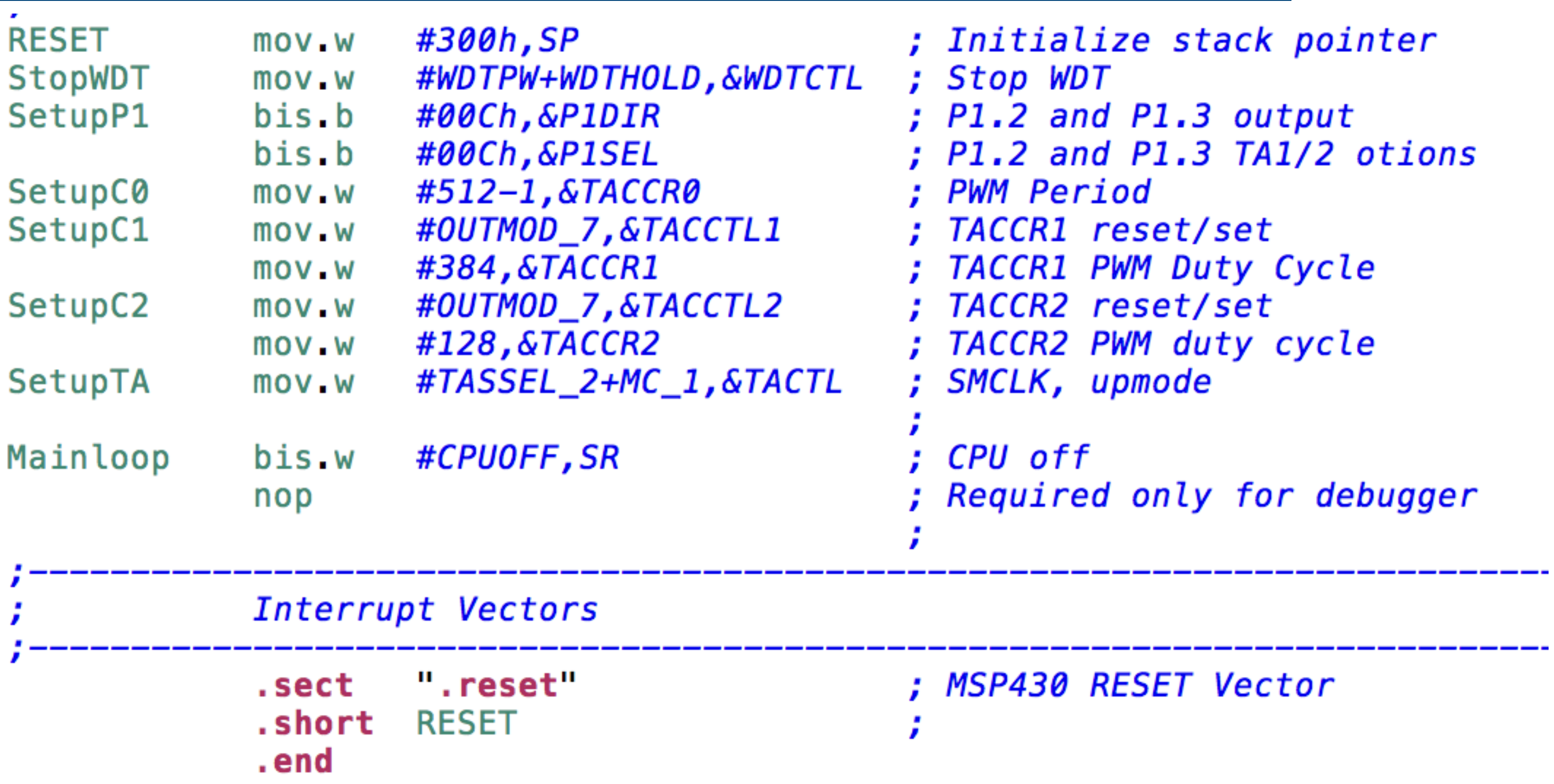


```
//******************************************************************************
//  MSP430F22x4 Demo - Timer_A, PWM TA1-2, Up Mode, DCO SMCLK
//
//  Description: This program generates two PWM outputs on P1.2,3 using
//  Timer_A configured for up mode. The value in TACCR0, 512-1, defines the PWM
//  period and the values in TACCR1 and TACCR2 the PWM duty cycles.
//  Using ~1.2MHz SMCLK as TACLK, the timer period is ~425us with
//  a 75% duty cycle on P1.2 and 25% on P1.3.
//  ACLK = n/a, SMCLK = MCLK = TACLK = default DCO ~1.2MHz
//
//                MSP430F22x4
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |         P1.2/TA1|--> TACCR1 - 75% PWM
//            |         P1.3/TA2|--> TACCR2 - 25% PWM
//
```

msp430x22x4_ta_16

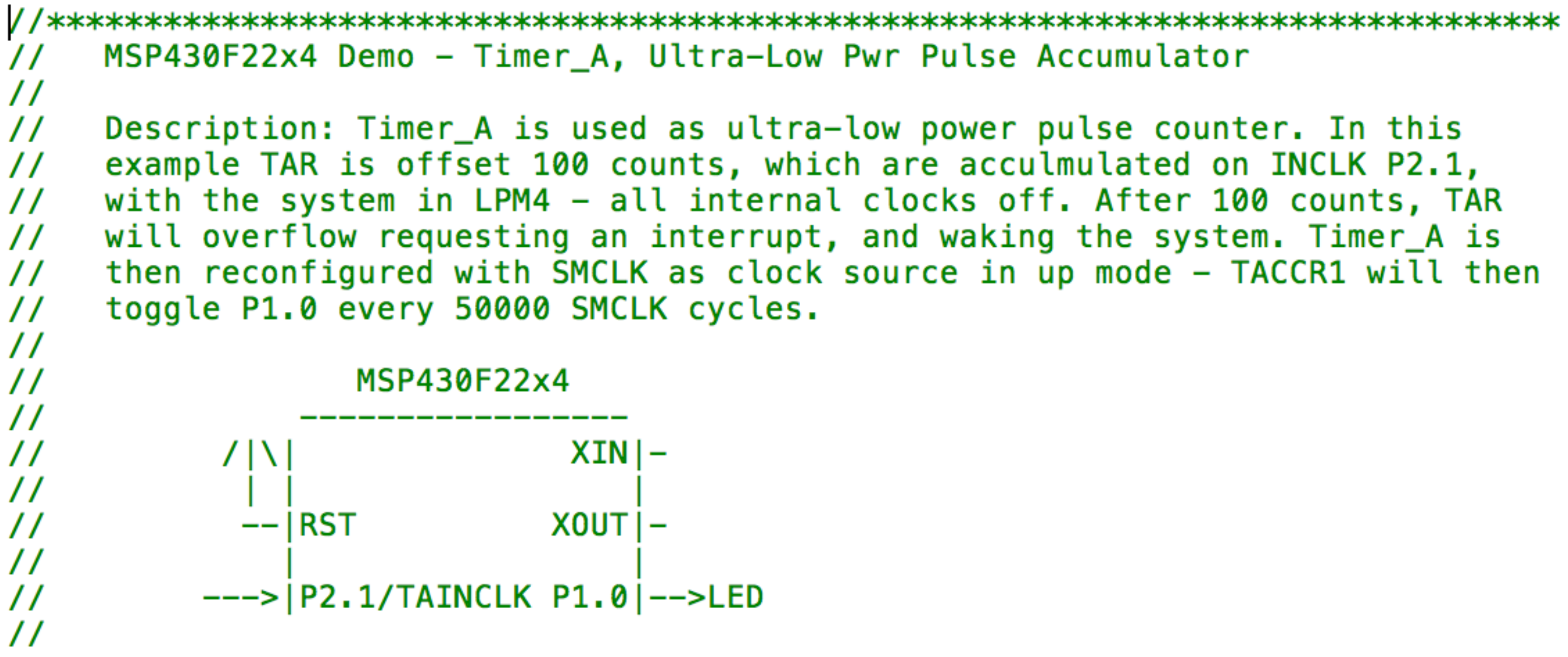
```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                // Stop WDT
  P1DIR |= 0x0C;                           // P1.2 and P1.3 output
  P1SEL |= 0x0C;                           // P1.2 and P1.3 TA1/2 otions
  TACCR0 = 512 - 1;                        // PWM Period
  TACCTL1 = OUTMOD_7;                      // TACCR1 reset/set
  TACCR1 = 384;                            // TACCR1 PWM duty cycle
  TACCTL2 = OUTMOD_7;                      // TACCR2 reset/set
  TACCR2 = 128;                            // TACCR2 PWM duty cycle
  TACTL = TASSEL_2 + MC_1;                 // SMCLK, up mode

  __bis_SR_register(CPUOFF);               // Enter LPM0
}
```

msp430x22x4_ta_16



```
;
RESET       mov.w   #300h,SP                  ; Initialize stack pointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL    ; Stop WDT
SetupP1     bis.b   #00Ch,&P1DIR              ; P1.2 and P1.3 output
            bis.b   #00Ch,&P1SEL              ; P1.2 and P1.3 TA1/2 otions
SetupC0     mov.w   #512-1,&TACCR0            ; PWM Period
SetupC1     mov.w   #OUTMOD_7,&TACCTL1        ; TACCR1 reset/set
            mov.w   #384,&TACCR1              ; TACCR1 PWM Duty Cycle
SetupC2     mov.w   #OUTMOD_7,&TACCTL2        ; TACCR2 reset/set
            mov.w   #128,&TACCR2              ; TACCR2 PWM duty cycle
SetupTA     mov.w   #TASSEL_2+MC_1,&TACTL     ; SMCLK, upmode
                                              ;
Mainloop    bis.w   #CPUOFF,SR                ; CPU off
            nop                               ; Required only for debugger
                                              ;
;-----------------------------------------------------------------------------
;           Interrupt Vectors
;-----------------------------------------------------------------------------
            .sect   ".reset"                  ; MSP430 RESET Vector
            .short  RESET                     ;
            .end
```

msp430x22x4_ta_22



```
//******************************************************************************
//   MSP430F22x4 Demo - Timer_A, Ultra-Low Pwr Pulse Accumulator
//
//   Description: Timer_A is used as ultra-low power pulse counter. In this
//   example TAR is offset 100 counts, which are acculmulated on INCLK P2.1,
//   with the system in LPM4 - all internal clocks off. After 100 counts, TAR
//   will overflow requesting an interrupt, and waking the system. Timer_A is
//   then reconfigured with SMCLK as clock source in up mode - TACCR1 will then
//   toggle P1.0 every 50000 SMCLK cycles.
//
//                MSP430F22x4
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//        --->|P2.1/TAINCLK P1.0|-->LED
//
```

ta_22

```c
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                        // Stop WDT
  P2DIR = 0xFD;                                    // All but P2.1 outputs
  P2SEL = 0x02;                                    // P2.1 TAINCLK option select
  P2OUT = 0;                                       // All P2.x reset
  TACTL = TASSEL1 + TASSEL0 + TACLR + TAIE;        // Ext. INCLK, interrupt
  TAR = 0xFFFF - 100;                              // Offset until TAR overflow
  TACTL |= MC1;                                    // Start Timer_A continuous mode
  __bis_SR_register(LPM4_bits + GIE);              // Enter LPM4 w/ interrupts

  while (1)
  {
    P1OUT ^= 0x01;                                 // P1.0 = toggle
    __bis_SR_register(LPM0_bits);                  // CPU is not required
  }
}

#pragma vector = TIMERA1_VECTOR
__interrupt void TA1_ISR(void)
{
  switch (TAIV)         // Efficient switch-implementation
  {
    case 2:
      TACCR1 += 50000;                             // Add Offset to TACCR1
      __bic_SR_register_on_exit(LPM0_bits);        // CPU active on reti
      break;
    case 10:
      TACTL = TASSEL1 + TACLR;                     // SMCLK, clear TAR
      TACCTL1 = CCIE;                              // TACCR1 interrupt enabled
      TACCR1 = 50000;
      TACTL |= MC1;                                // Start Timer_A in continuous
      __bic_SR_register_on_exit(LPM4_bits);        // Exit LPM4 on reti
      break;
  }
}
```