THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

CPE 323: MSP430 Timers

Aleksandar Milenkovic

Electrical and Computer Engineering
The University of Alabama in Huntsville

milenka@ece.uah.edu

http://www.ece.uah.edu/~milenka

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Outline

- Watchdog Timer
- TimerA

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

MSP430xG461x Microcontroller

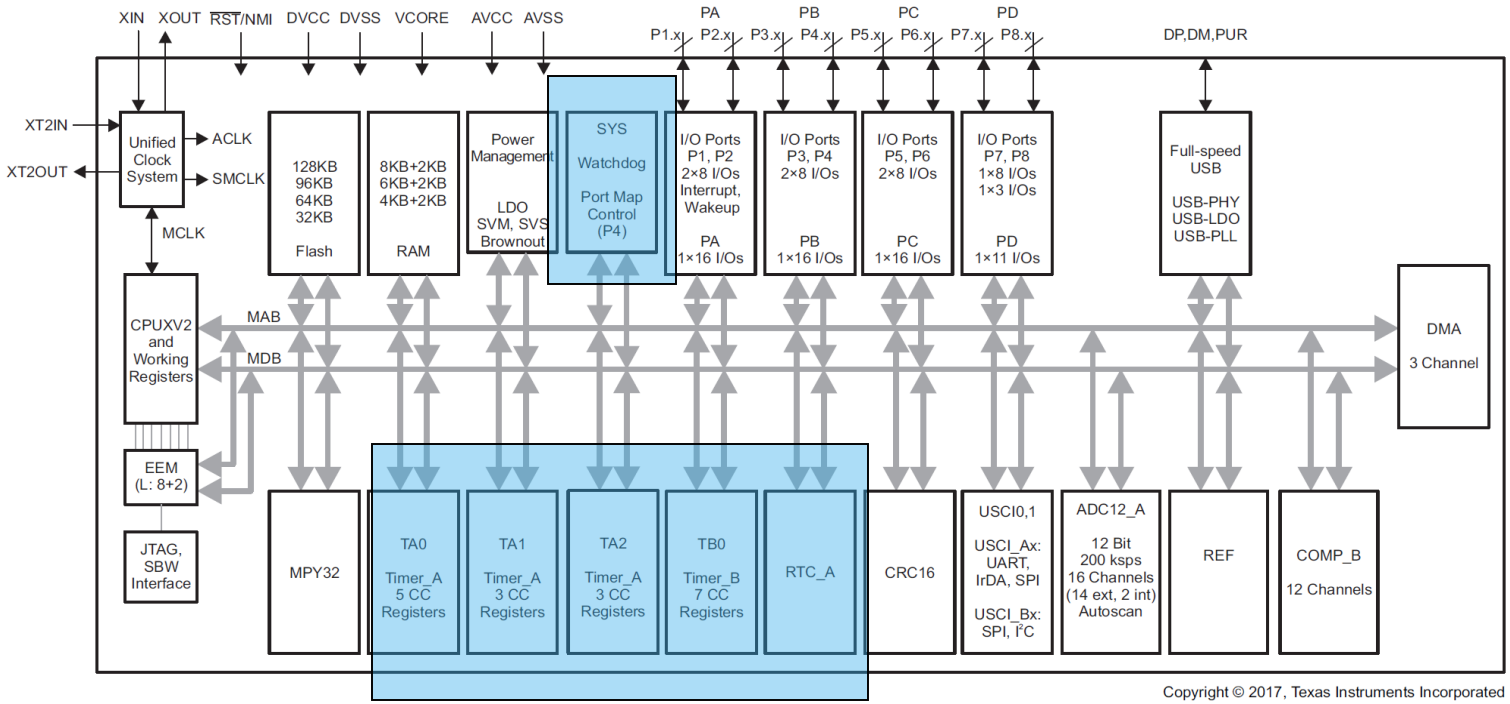


Figure 1-1. Functional Block Diagram – MSP430F5529IPN, MSP430F5527IPN, MSP430F5525IPN, MSP430F5521IPN

LaCASA

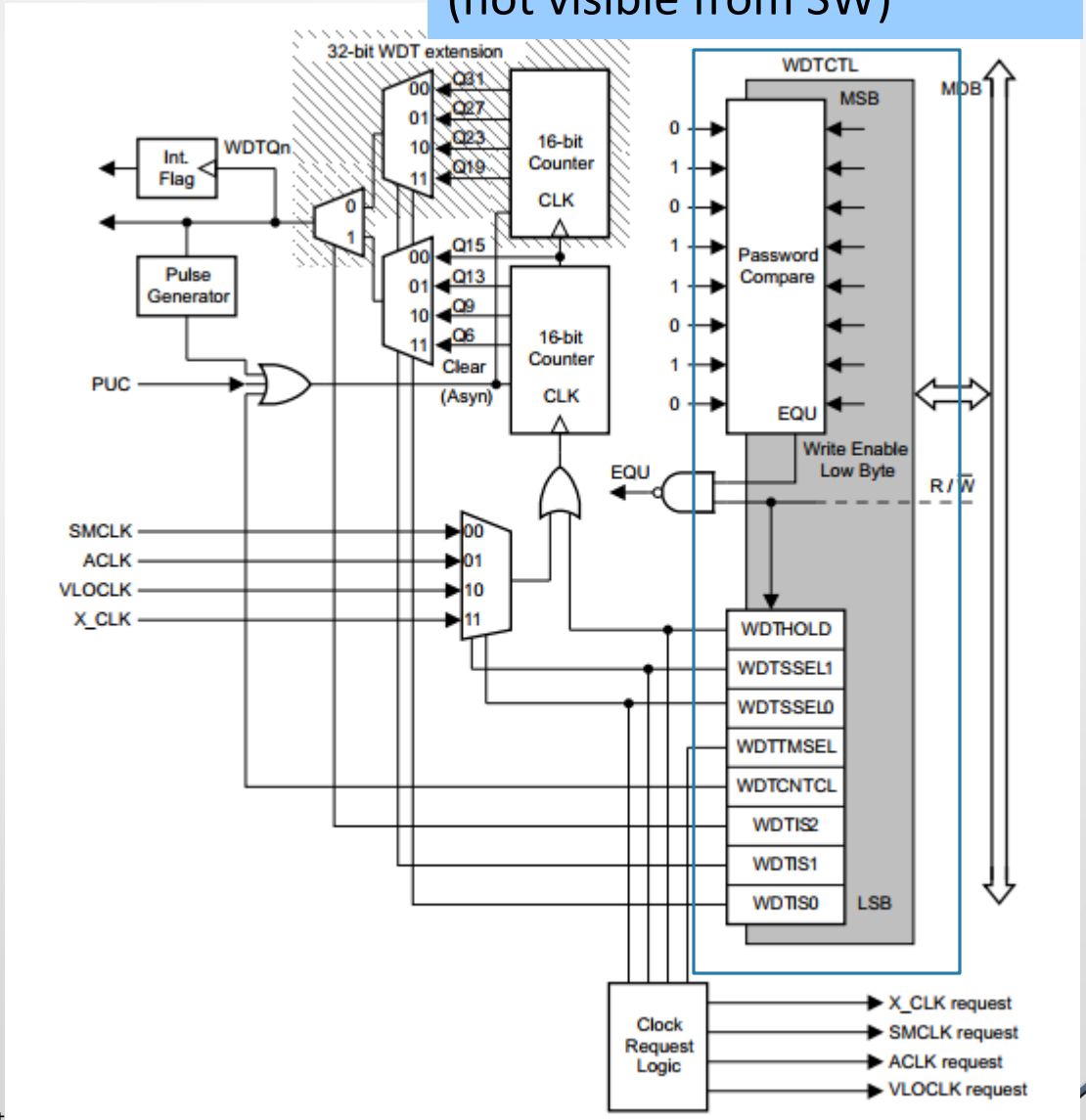
Watchdog Timer

- Primary function: WDT operation
  - Performs a controlled-system restart after a software problem occurs
  - If the selected time interval expires, a system reset is generated
- Secondary function
  (if WDT functionality is not needed)
  - Can work as an interval timer, to generate an interrupt after the selected time interval

WTD Block Di

WDTCNT – 32-bit counter (not visible from SW)

Clock sources: ACLK, SMCLK



WDT StartUp Conditions

- Important: It Powers Up Active
- After a PUC, WDT is automatically configured in the watchdog mode with an initial 32768 clock cycle reset interval using the DCOCLK
- => User must setup or halt the WDT prior to the expiration of the initial reset interval

Watchdog Mode

- Expiration of the selected time interval, sets WDTIFG and triggers a PUC
  => reset vector interrupt is sourced, and WDT goes to its default configuration
- Security key violation does the same
- WDTIFG can be used by the reset ISR to determine source of reset
  - If the flag is set, then WDT initiated the reset condition either by timing out or by a security key violation
  - If WDTIFG is cleared, the reset was caused by a different source

Interval Mode

- Interval mode: WDTTMSEL is set to 1
- In this mode, the WDT provides periodic interrupts
- WDTIFG is set at the expiration of the selected time interval (no PUC is generated this time)
- If the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt
  - The interrupt vector address in interval timer mode is different from that in watchdog mode
- The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software

WDT_A Registers

Register	Short Form	Register Type	Address	Initial State
Watchdog timer control register	WDTCTL	Read/write	0120h	06900h with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC†

† WDTIFG is reset with POR

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	WDTCTL	Watchdog Timer Control	Read/write	Word	6904h	Section 16.3.1
00h	WDTCTL_L		Read/write	Byte	04h	
01h	WDTCTL_H		Read/write	Byte	69h	

WDT Control Register

- Contains control bits to configure WDT plus the RST/NMI pin
- It is a 16-bit password-protected read/write register (WORD PERIPHERAL)
  - Writes must include the write password 05Ah in the upper byte
  - Otherwise it is a security key violation and triggers a PUC system reset regardless of timer mode
  - Reads of WDTCTL reads 069h in the upper byte

15	14	13	12	11	10	9	8
WDTPW							
7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

WDTCTL Register

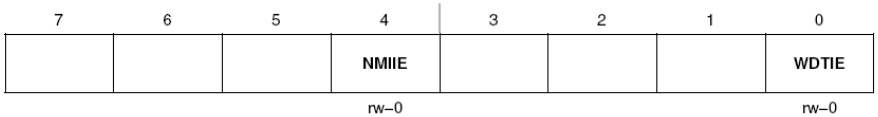
15	14	13	12	11	10	9	8
WDTPW							
7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

Bit	Field	Type	Reset	Description
15-8	WDTPW	RW	69h	Watchdog timer password. Always read as 069h. Must be written as 5Ah; if any other value is written, a PUC is generated.
7	WDTHOLD	RW	0h	Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power. 0b = Watchdog timer is not stopped. 1b = Watchdog timer is stopped.
6-5	WDTSSEL	RW	0h	Watchdog timer clock source select 00b = SMCLK 01b = ACLK 10b = VLOCLK 11b = X_CLK; VLOCLK in devices that do not support X_CLK
4	WDTTMSEL	RW	0h	Watchdog timer mode select 0b = Watchdog mode 1b = Interval timer mode
3	WDTCNTCL	RW	0h	Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset. 0b = No action 1b = WDTCNT = 0000h
2-0	WDTIS	RW	4h	Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC. 000b = Watchdog clock source /(2^{31}) (18h:12m:16s at 32.768 kHz) 001b = Watchdog clock source /(2^{27}) (01h:08m:16s at 32.768 kHz) 010b = Watchdog clock source /(2^{23}) (00h:04m:16s at 32.768 kHz) 011b = Watchdog clock source /(2^{19}) (00h:00m:16s at 32.768 kHz) 100b = Watchdog clock source /(2^{15}) (1 s at 32.768 kHz) 101b = Watchdog clock source /(2^{13}) (250 ms at 32.768 kHz) 110b = Watchdog clock source /(2^{9}) (15.625 ms at 32.768 kHz) 111b = Watchdog clock source /(2^{6}) (1.95 ms at 32.768 kHz)

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

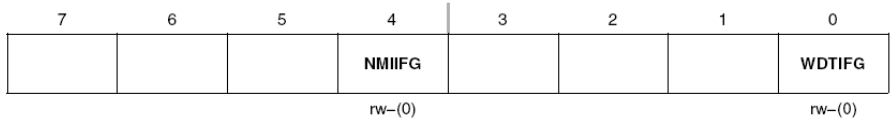
IE1, IFG1

IE1, Interrupt Enable Register 1



	Bits 7-5	These bits may be used by other modules. See device-specific data sheet.
NMIIE	Bit 4	NMI interrupt enable. This bit enables the NMI interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 3-1	These bits may be used by other modules. See device-specific data sheet.
WDTIE	Bit 0	Watchdog timer interrupt enable. This bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled

IFG1, Interrupt Flag Register 1



	Bits 7-5	These bits may be used by other modules. See device-specific data sheet.
NMIIFG	Bit 4	NMI interrupt flag. NMIIFG must be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear NMIIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 No interrupt pending 1 Interrupt pending
	Bits 3-1	These bits may be used by other modules. See device-specific data sheet.
WDTIFG	Bit 0	Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or it can be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear WDTIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 No interrupt pending 1 Interrupt pending

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

WDT Demo: Blinking LED1 (Lab7_D1.c)

```
/*-------------------------------------------------------------------------------
 * File:         Lab7_D1.c (CPE 325 Lab7 Demo code)
 * Function:     Blinking LED1 using WDT ISR (MPS430F5529)
 *
 * Description: This C program configures the WDT in interval timer mode,
 *              clocked with the ACLK clock. The WDT is configured to give an
 *              interrupt for every 1s. LED1 is toggled in the WDT ISR
 *              by xoring P1.0. The blinking frequency of LED1 is 0.5Hz.
 *
 * Board:       MSP-EXP430F5529 (includes 32-KHZ crystal on XT1 and
 *                               4-MHz ceramic resonator on XT2)
 *
 * Clocks:      ACLK = XIN-XOUT = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                          MSP430F5529
 *                      -----------------
 *                  /|\|              XIN|-
 *                   | |                 | 32kHz crystal
 *                   --|RST          XOUT|-
 *                     |                 |
 *                     |             P1.0|-->LED1(RED)
 *                     |                 |
 * Input:       None
 * Output:      LED1 blinks at 0.5Hz frequency
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *              Prawar Poudel
 * Date:        December 2008
 *-------------------------------------------------------------------------------*/
```


LaCASA

WDT Demo: Blink LED (WDT Interval)

```
#include <msp430.h>

void main(void) {
    WDTCTL = WDT_ADLY_1000;           // 1 s interval timer
    P1DIR |= BIT0;                    // Set P1.0 to output direction
    SFRIE1 |= WDTIE;                  // Enable WDT interrupt
    _BIS_SR(LPM0_bits + GIE);         // Enter LPM0 w/ interrupt
}

// Watchdog Timer Interrupt Service Routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    P1OUT ^= BIT0;                    // Toggle P1.0 using exclusive-OR
}
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

WDT Demo: Blinking LED1 (Lab7_D2.c)

```
/*-------------------------------------------------------------------------------
 * File:         Lab7_D2.c (CPE 325 Lab7 Demo code)
 *
 * Function:     Toggling LED1 using WDT ISR (MPS430F5529)
 *
 * Description: This C program configures the WDT in interval timer mode,
 *              clocked with SMCLK. The WDT is configured to give an
 *              interrupt for every 32ms. The WDT ISR is counted for 32 times
 *              (32*32.5ms ~ 1sec) before toggling LED1 to get 1 s on/off.
 *              The blinking frequency of LED1 is 0.5Hz.
 *
 * Clocks:       ACLK = XT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *               An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                           MSP430xF5529
 *                       -----------------
 *                   /|\|              XIN|-
 *                    | |                 | 32kHz
 *                    --|RST           XOUT|-
 *                      |                 |
 *                      |             P1.0|-->LED1(RED)
 *                      |                 |
 * Input:        None
 * Output:       LED1 blinks at 0.5Hz frequency
 * Author:       Aleksandar Milenkovic, milenkovic@computer.org
 *               Prawar Poudel
 * Date:         December 2008
 *-------------------------------------------------------------------------------*/
```


LaCASA

WDT Demo: Blinking LED1 (Lab7_D2.c)

```
#include <msp430.h>

void main(void)
{
    WDTCTL = WDT_MDLY_32;           // 32ms interval (default)
    P1DIR |= BIT0;                  // Set P1.0 to output direction
    SFRIE1 |= WDTIE;                // Enable WDT interrupt

    _BIS_SR(LPM0_bits + GIE);       // Enter LPM0 with interrupt
}

// Watchdog Timer interrupt service routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    static int i = 0;
    i++;
    if (i == 32) {                  // 31.25 * 32 ms = 1s
        P1OUT ^= BIT0;              // Toggle P1.0 using exclusive-OR
                                    // 1s on, 1s off; period = 2s, f = 1/2s = 0.5Hz
        i = 0;
    }
}
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Blinking LED1 Using SW Polling (Lab7_D3.c)

```
/*-------------------------------------------------------------------------------
 * File:         Lab7_D3.c (CPE 325 Lab7 Demo code)
 * Function:     Blinking LED1 using software polling.
 * Description: This C program configures the WDT in interval timer mode and
 *               it is clocked with ACLK. The WDT sets the interrupt flag (WDTIFG)
 *               every 1 s. LED1 is toggled by verifying whether this flag
 *               is set or not. After it is detected as set, the WDTIFG is cleared.
 * Clocks:       ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (2^20 Hz)
 *               An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                          MSP430F5529
 *                      -----------------
 *                  /|\|              XIN|-
 *                   | |                 | 32kHz
 *                   --|RST          XOUT|-
 *                     |                 |
 *                     |             P1.0|-->LED1(RED)
 *                     |                 |
 * Input:        None
 * Output:       LED1 blinks at 0.5Hz frequency
 *
 * Author:       Aleksandar Milenkovic, milenkovic@computer.org
 * Revised by:   Prawar Poudel
 *-------------------------------------------------------------------------------*/
```


LaCASA

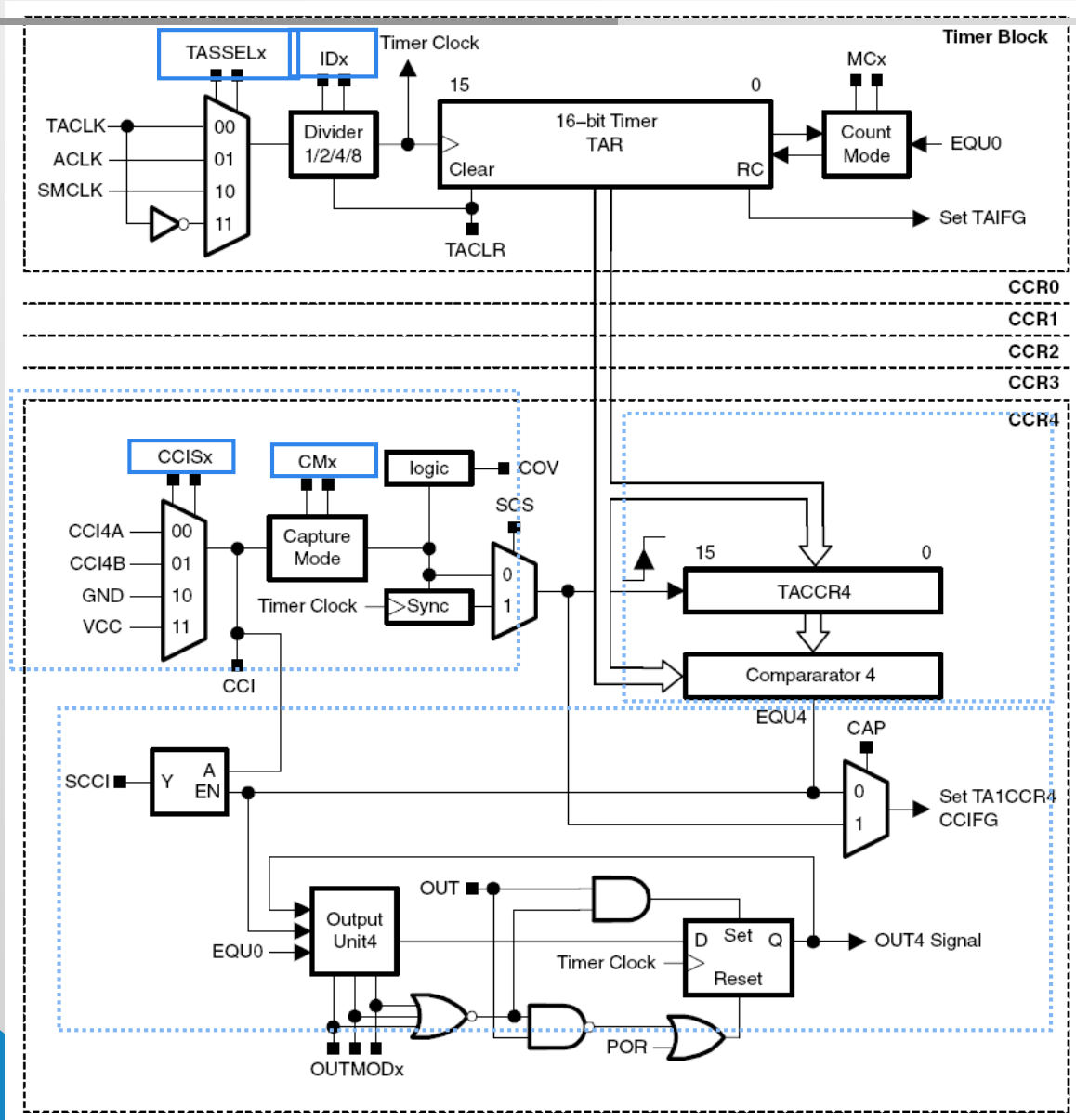
Blinking LED1 Using SW Polling

```
#include <msp430.h>

void main(void)
{
    WDTCTL = WDT_ADLY_1000;             // 1 s interval timer
    P1DIR |= BIT0;                      // Set P2.2 to output direction

    for (;;) {
        // Use software polling
        if ((SFRIFG1 & WDTIFG) == 1) {
            P1OUT ^= BIT0;
            SFRIFG1 &= ~WDTIFG;         // Clear bit WDTIFG in IFG1
        }
    }
}
```

Timer_A

- General-purpose timer in MSP430
- Features
  - 16-bit counter with 4 operating modes
  - Selectable and configurable clock source
  - Three (five, seven) independently configurable *capture/compare registers with configurable inputs*
  - Three (five, seven) individually configurable *output modules with 8 output modes*
- Multiple, simultaneous, timings; multiple capture/compares; multiple output waveforms such as PWM signals; and any combination of these
- Interrupt capabilities
  - Each capture/compare block individually configurable

Timer_A Block Diagram



Timer Block

(TAR)

Capture & compare channels

(TACCRx)

Timer_A Organization

- Timer block (TAR)
  - up/down counter with a choice of clock sources that can be prescaled (divided)
  - TAIFG is raised when the counter returns to 0
- Capture & compare channel
  - Capture: we capture an input, which means record the “time” (value in TAR) at which the input changes in TACCRn; the input can be internal (from another peripheral or SW) or external
  - Compare: the current value of TAR is compared to the value stored in TACCRn and the output is updated when they match; the output can be either internal or external
  - Request an interrupt on either capture or compare or by setting its CCIFG flag (e.g., from SW)
  - Sample an input at a compare event; useful if TimerA is used for serial communication

Timer_A Organization (cont'd)

- Single Timer block, multiple Capture&Compare channels
  - We may have multiple Timer_A modules that can operate on independent time bases
- Use HW (TimerA) for more precise timing and reserve software for the less critical tasks
- TACCR0 is special
  - Used for UP and UP/DOWN mode and cannot be used for usual functions
  - Has its own interrupt vector with a higher priority than the other interrupts from TimerA, which all share a common vector

Timer_A Modes

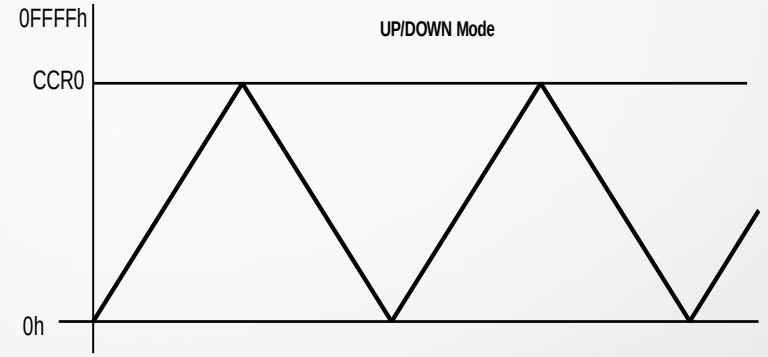
MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0.
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and back down to zero.

Stop/Halt Mode

Timer is halted with the next +CLK

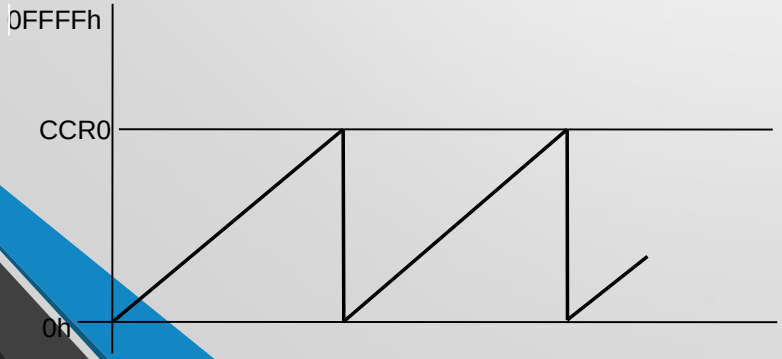
UP/DOWN Mode

Timer counts between 0 and CCR0 and 0



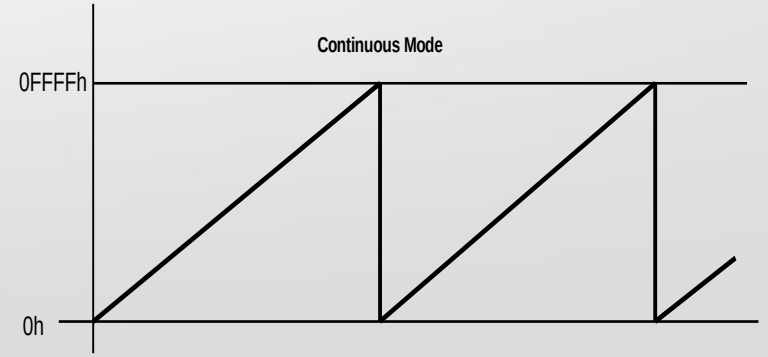
UP Mode

Timer counts between 0 and CCR0



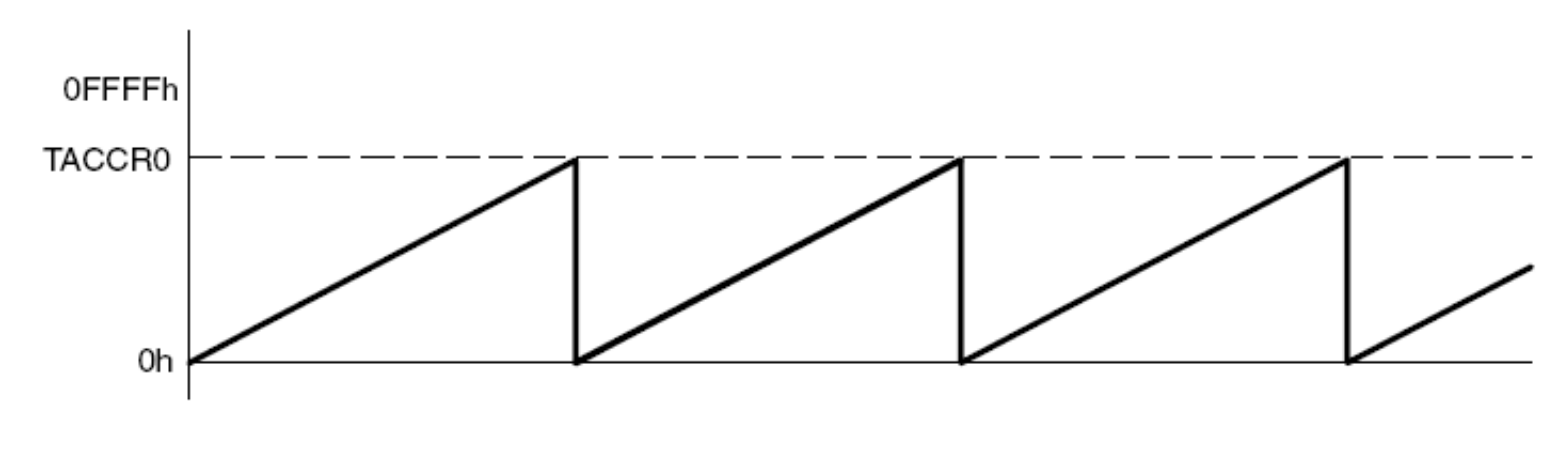
Continuous Mode

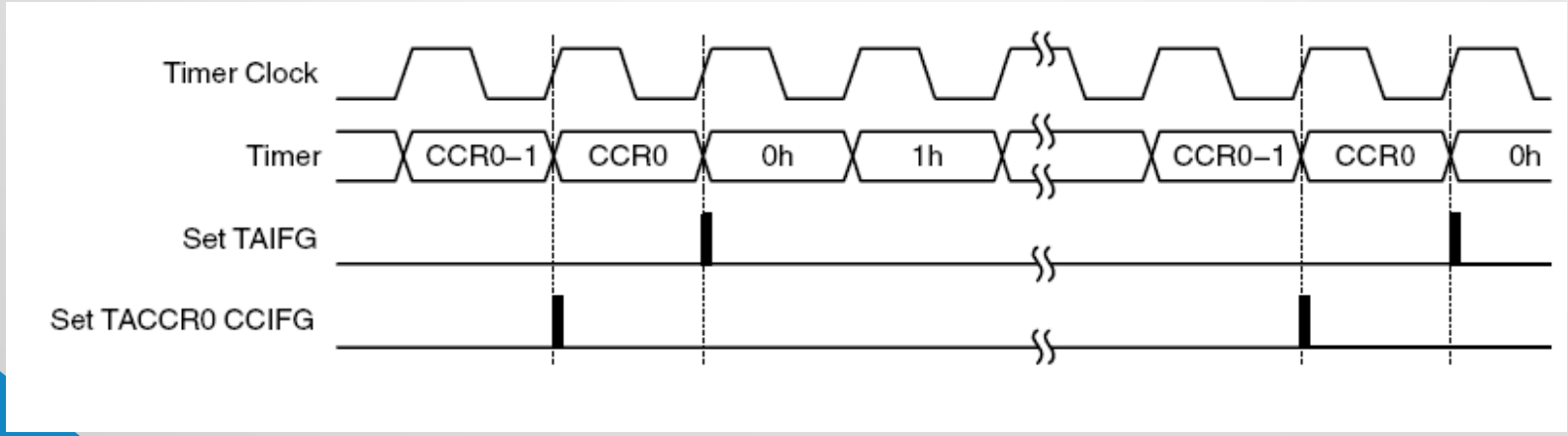
Timer continuously counts up



UP Mode

Counts up: 0, 1, …TACCR0, 0, 1 … (period is (TACCR0+1)Tclk

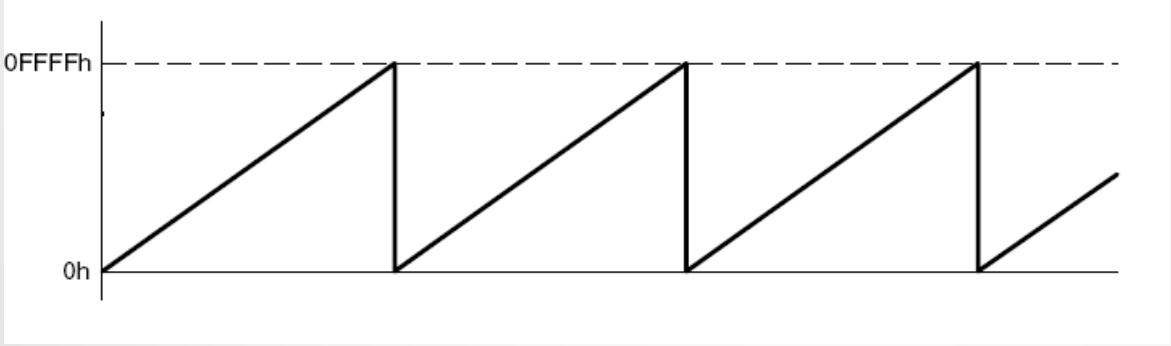


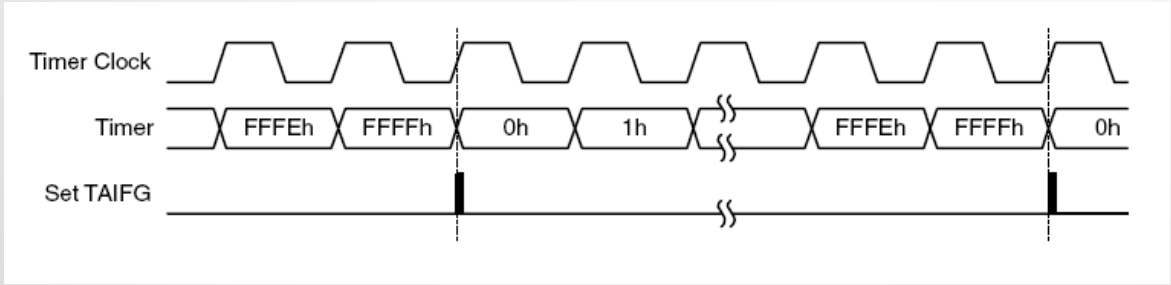


THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

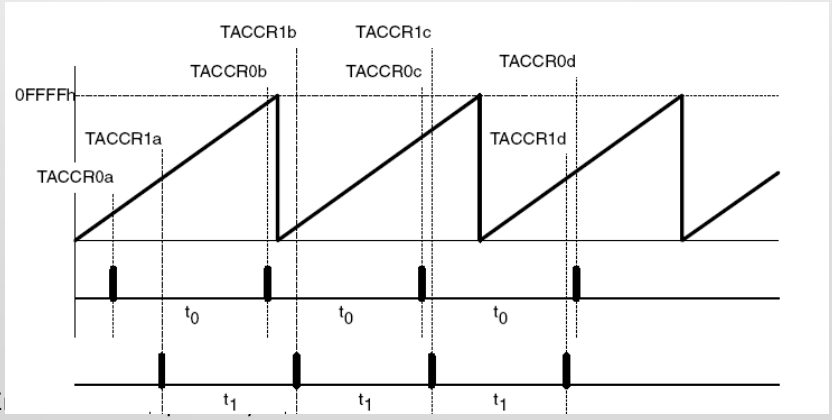
Continuous Mode

Counts up: 0, 1, ...0xFFFF, 0, 1 ... (period is (2^16)Tclk





Use of CM

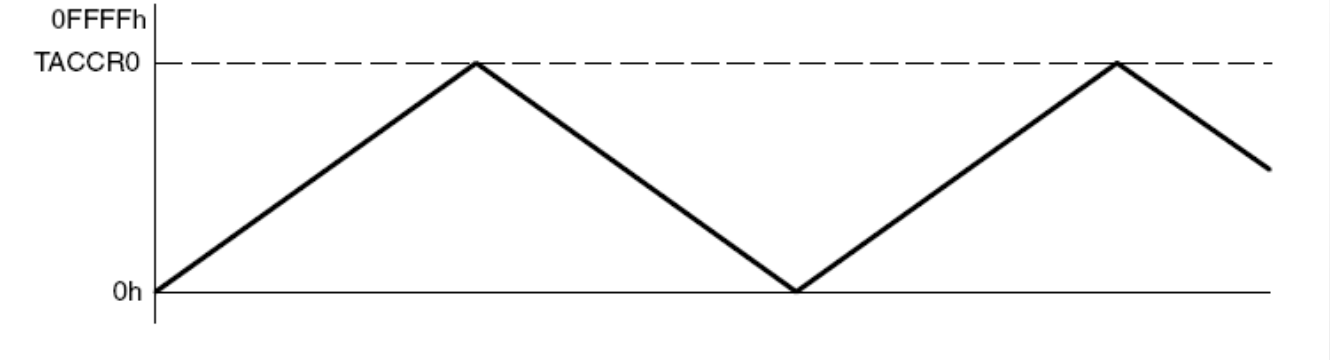


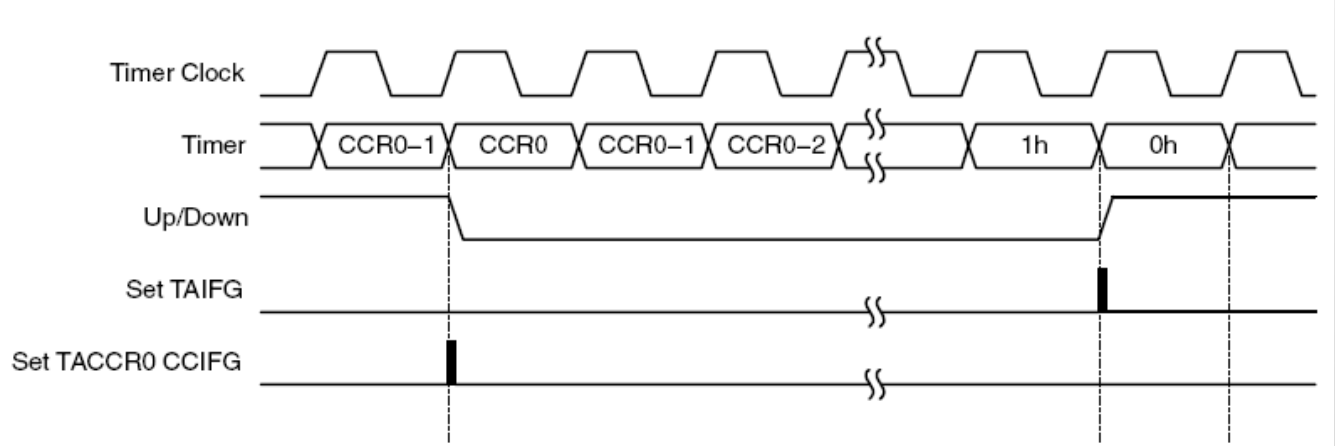
LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Up/Down Mode

Counts up: 0, 1, …TACCR0, TACCR0-1, … 2, 1, 0, 1, …
(period is (2*TACCR0)Tclk

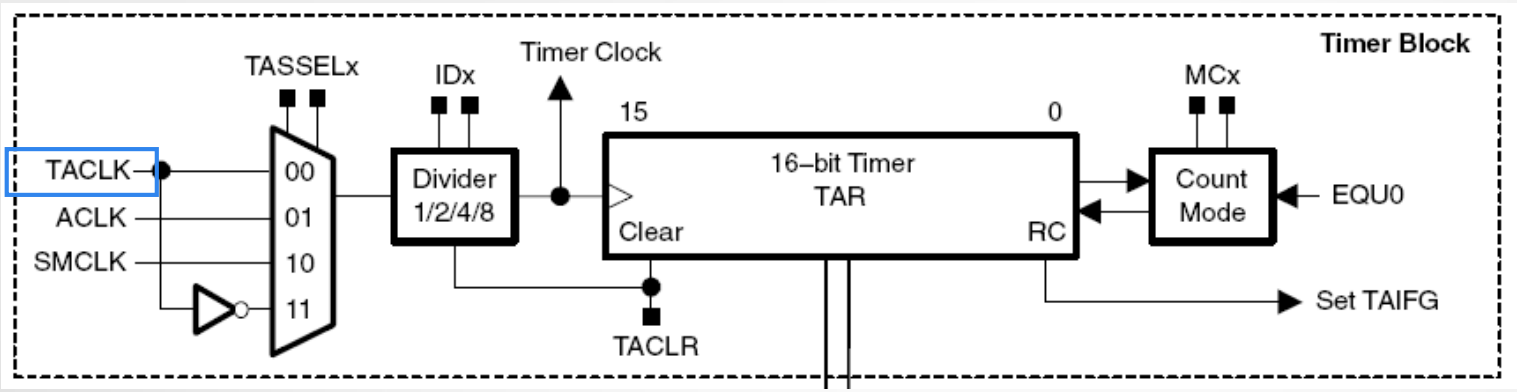




LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Counter



TACTL

160h

15														0
unused						Input Select		Input Divider		Mode Control		un-used	CLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	(w)-(0)	rw-(0)	rw-(0)

MC1 MC0

0 0 Stop Mode
0 1 Up Mode
1 0 Continuous Mode
1 1 Up/Down Mode

ID1 ID0

0 0 1/1, Pass
0 1 1/2
1 0 1/4
1 1 1/8

SSEL1 SSEL0

0 0 TACLK
0 1 ACLK
1 0 MCLK
1 1 INCLK (often = #TACLK)

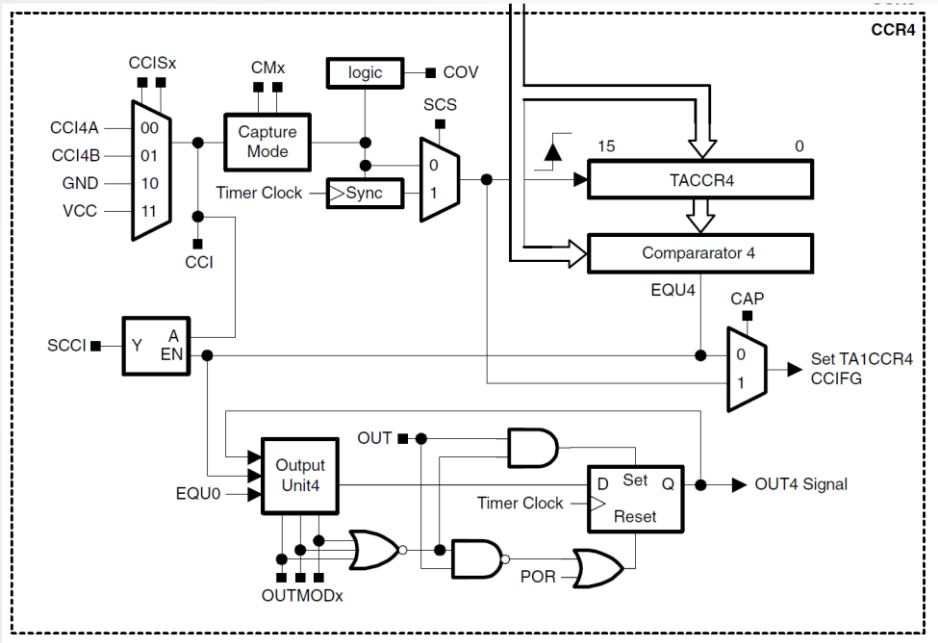
TACLR – clears TAR and resets the direction of counting (it clears automatically itself)

TAIFG – set when the timer counts to 0; a maskable interrupt is requested if TAIE bit is set

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Capture and Compare Block



CCRx
0172h
to
017Eh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CCTLx
162h
to
16Eh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPTURE MODE		INPUT SELECT		SCS	SCCI	un-used	CAP	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

LaCASA

TACCTLn: Capture Control

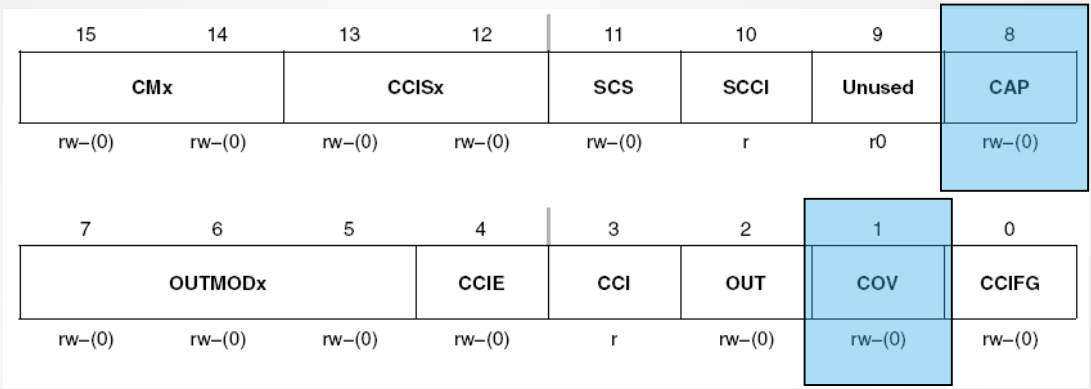
- **CMx** (Capture Mode)
  - 00 – disabled
  - 01 – positive edge
  - 10 – negative edge
  - 11 – both edges
- **CCISx** (Capture Input Select)
  - 00 – CCInA (outside timer)
  - 01 – CCInB (outside timer)
  - 10 – Gnd (pointless, but allows captures from SW)
  - 11 – Vdd (pointless, but allows captures from SW)
  - (for SW-triggered captures: use CMx=11, set CCIS1=1, and toggle CCIS0)
- **SCS** – synchronizer bit ensures synchronization with the timer clock (SHOULD always be set)
  - Race conditions: the selected input changes at the same time as the timer clock
- **CCI** –the state of the selected input can be read at any time from SW
- **OUT**- For output mode 0, this bit directly controls the state of the output

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)

TACCTLn: Capture Control

- CAP: Capture mode
  - 0 - Compare mode
  - 1 - Capture mode



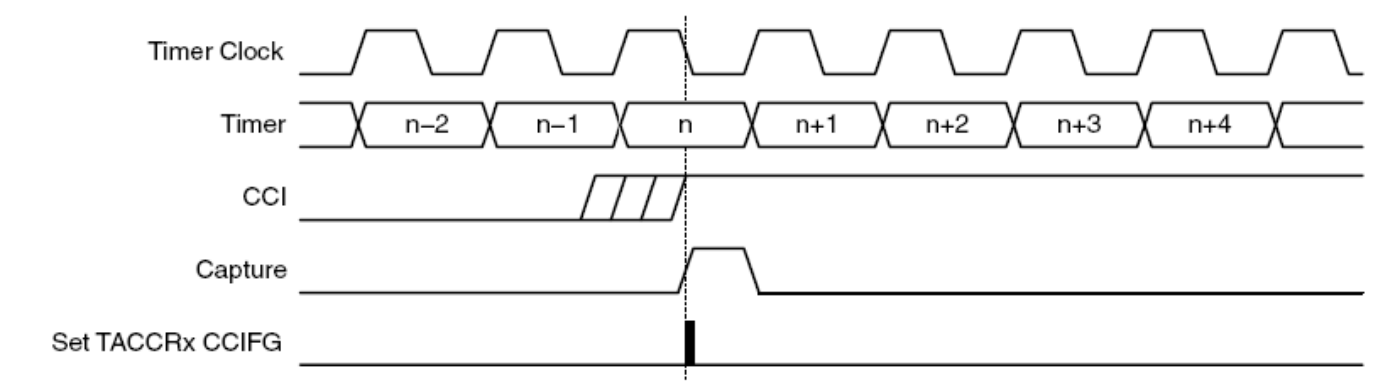
15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	r0	rw–(0)

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)

- Capture: TAR is copied into TACCRn, the channel flag CCIFGn is set, and a maskable interrupt is requested if bit CCIE in TACCTLx is set
- COV: Capture Overflow (next capture occurs before the TACCRn has been read following the previous event)

Capture Signal Synchronization

- The capture signal can be asynchronous to the timer clock and cause a race condition
- => Setting the SCS bit synchronizes the capture with the next timer clock



TACCTLn: Compare Mode

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)

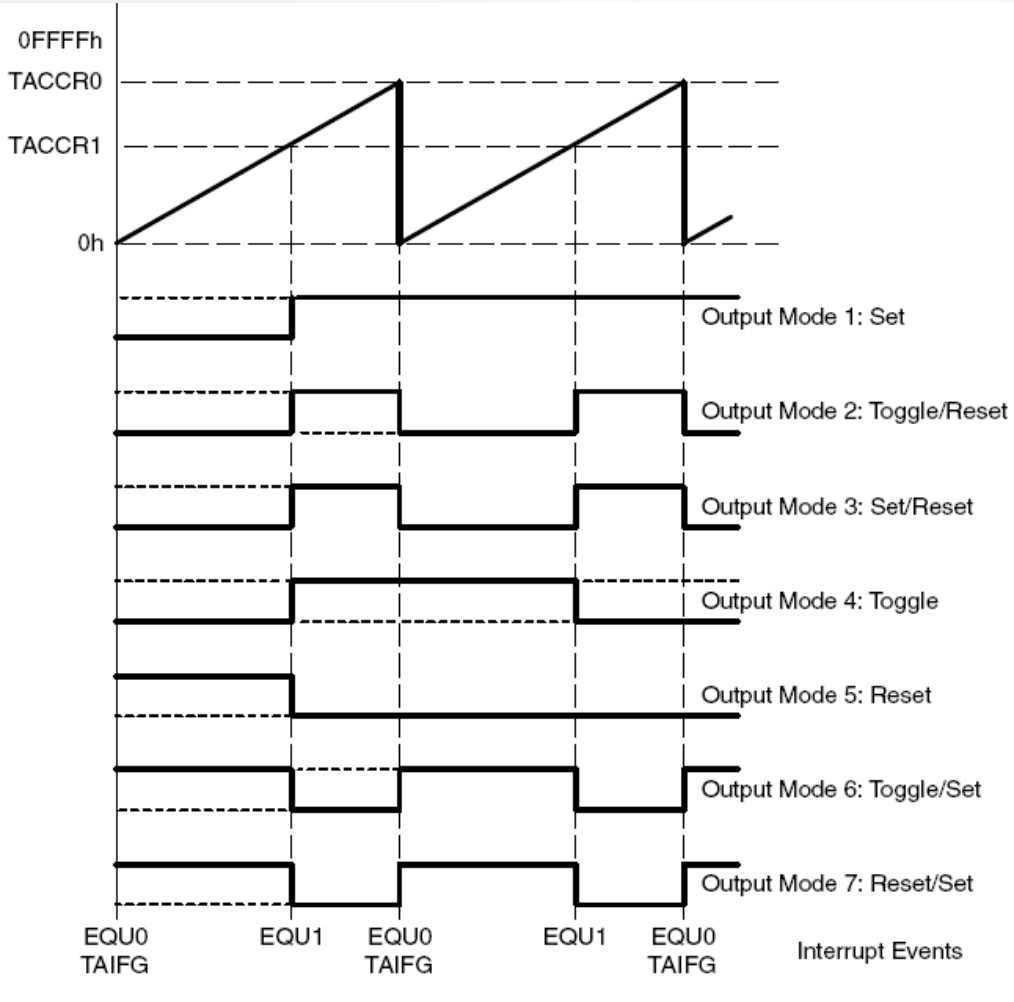
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	rw–(0)	rw–(0)	rw–(0)

- Compare mode: produces an output and an interrupt at the time stored in TACCRn
- Actions when TAR reaches value in TACCRn
  - Internal EQU is set
  - CCIFGn flag is set and an interrupt is requested if enabled
  - Output OUTn is changed according to the mode set in OUTMODx bits in TACCTLn
  - Input signal to the capture HW, CCI, is latched into the SCCI bit
- Use compare mode to trigger periodic events on other peripherals (e.g., DAC, ADC)

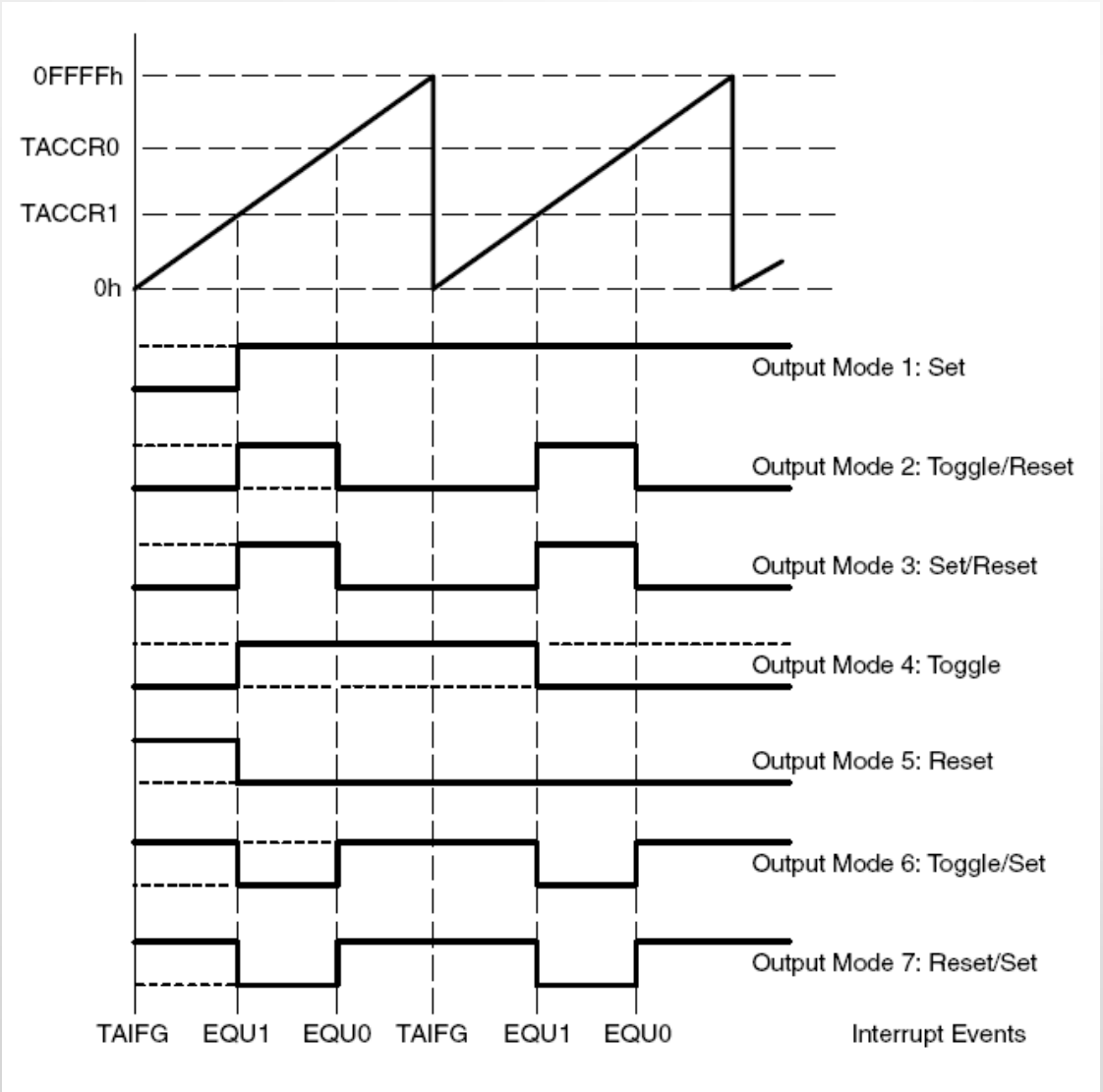
Output Mode

- OUTMODx: Output mode
  - 000 OUT bit value
  - 001 Set
  - 010 Toggle/reset
  - 011 Set/reset
  - 100 Toggle
  - 101 Reset
  - 110 Toggle/set
  - 111 Reset/set

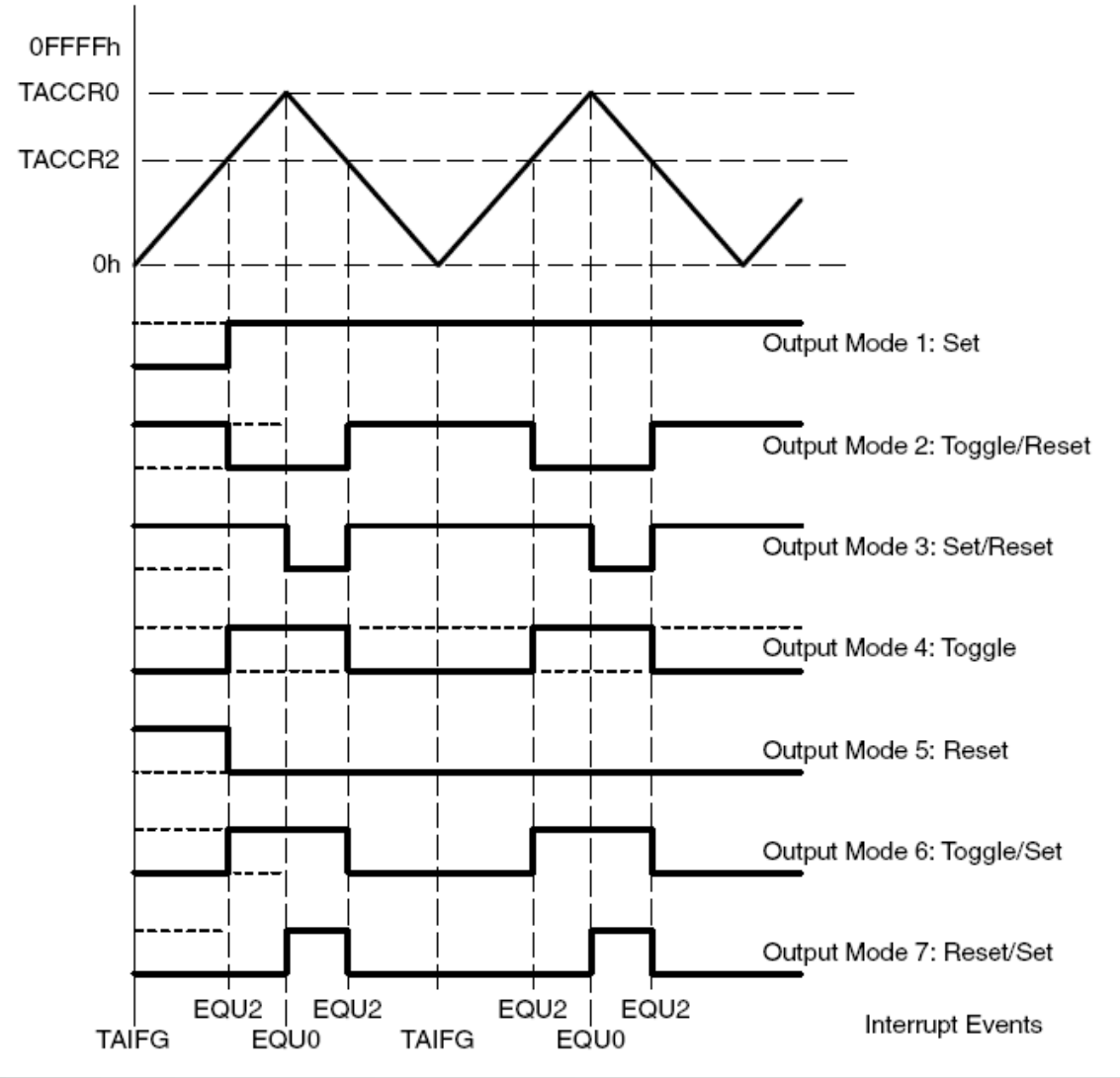
Output Modes (UP Counter Mode)



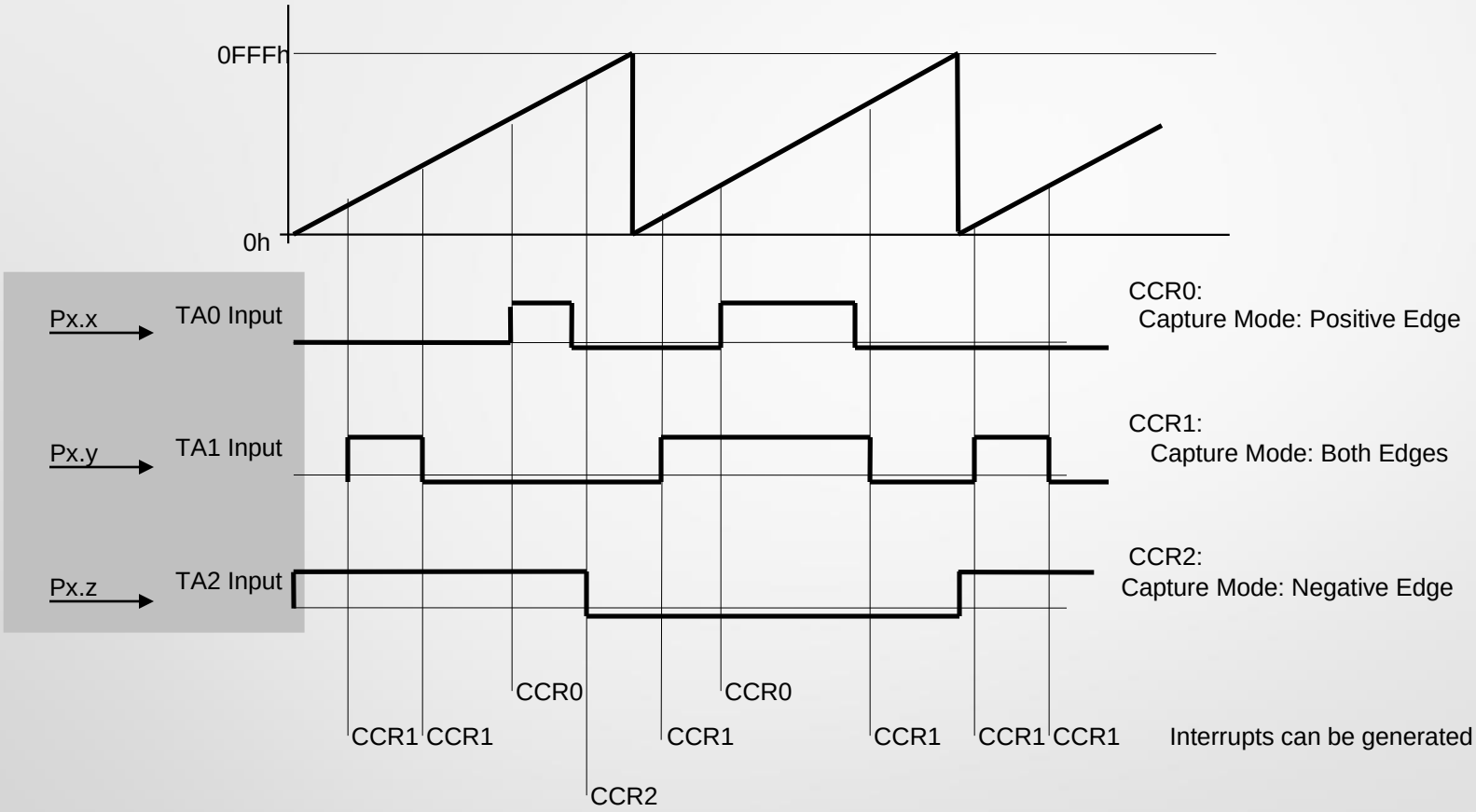
Output Modes (CONT Counter Mode)



Output Modes (UP/DOWN Counter Mode)

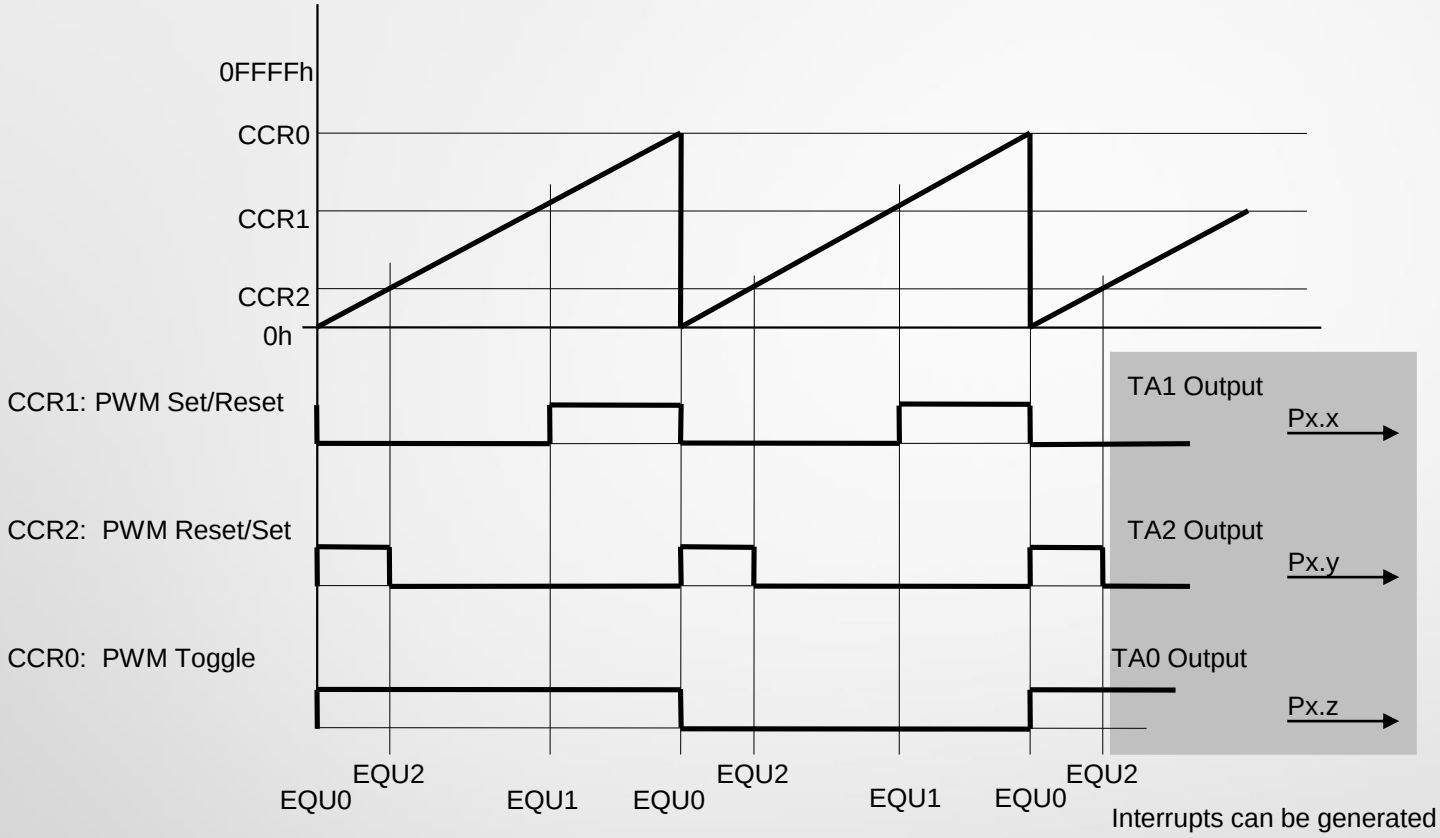


Capture Example (CONT Mode)



Example shows three independent HW event captures.
CCRx "stamps" time of event - Continuous-Mode is ideal.

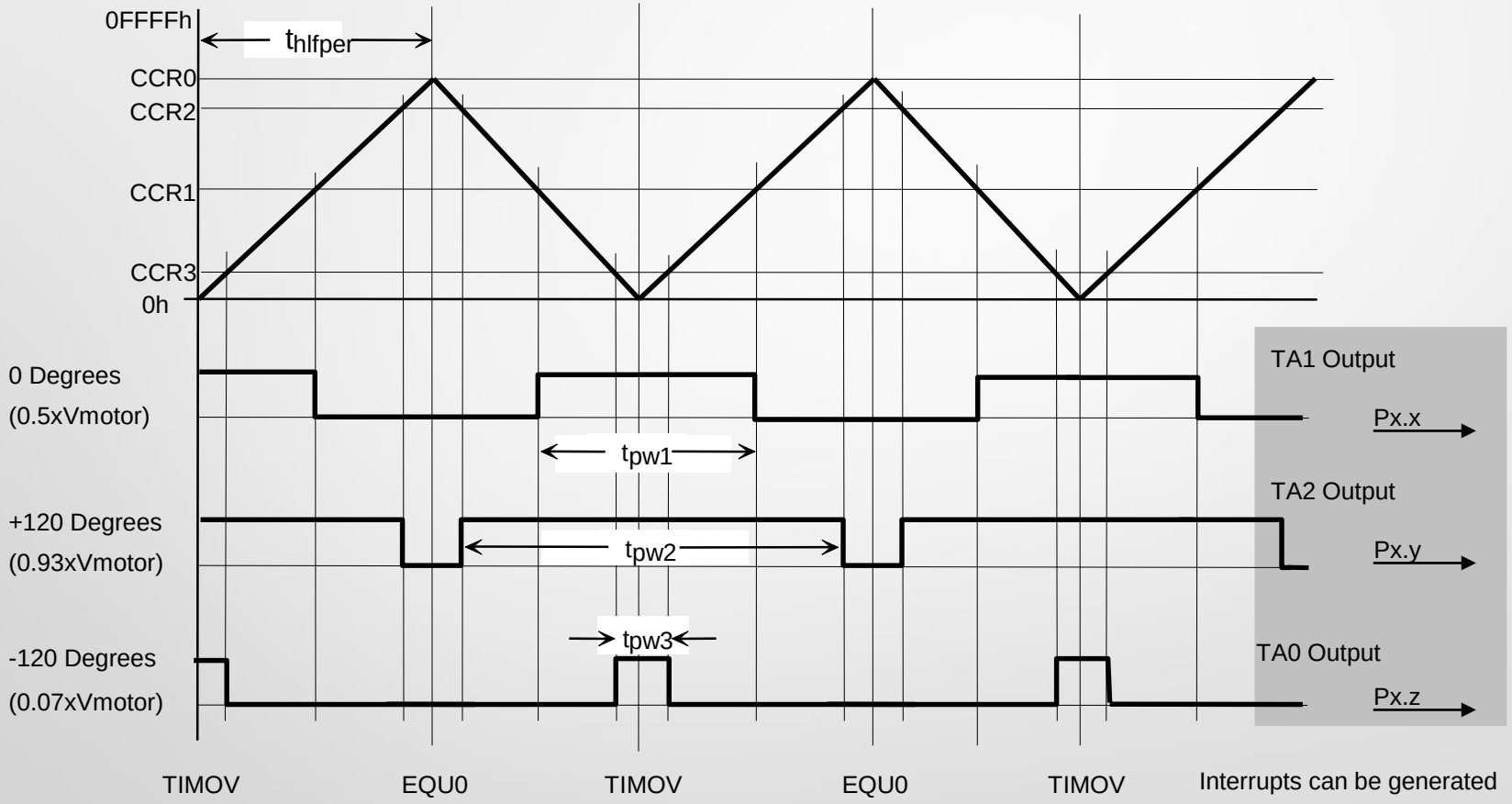
PWM (UP Mode)



Output Mode 4: PWM Toggle

Example shows three different asymmetric PWM-Timings generated with the Up-Mode

PWD (UP/DOWN Mode)



Example shows Symmetric PWM Generation - Digital Motor Control

Interrupts

- Sources: when TAIFG and CCIFG bit in each TACCTLn is set (CCIFGn for short)
- TACCR0 interrupt is privileged (has higher priority than others) and has its own vector TIMERA0_VECTOR (single source)
- TIMERA1_VECTOR is shared by the others (TAIFG + CCIFGx, x=1,2, ...) (multi source)
- Inspecting individual flags can take a lot of time in the ISR => Timer_A uses TAIV – interrupt vector register to identify the source of the interrupt rapidly
- When one or more of the shared and enabled interrupts is set, TAIV is loaded with the value that corresponds to the highest priority

ISRs

```
; Interrupt handler for TACCR0 CCIFG.                    Cycles
CCIFG_0_HND
;         ...       ; Start of handler Interrupt latency  6
          RETI                                             5

; Interrupt handler for TAIFG, TACCR1 and TACCR2 CCIFG.

TA_HND    ...                 ; Interrupt latency          6
          ADD    &TAIV,PC     ; Add offset to Jump table   3
          RETI                ; Vector 0: No interrupt     5
          JMP    CCIFG_1_HND  ; Vector 2: TACCR1           2
          JMP    CCIFG_2_HND  ; Vector 4: TACCR2           2
          RETI                ; Vector 6: Reserved         5
          RETI                ; Vector 8: Reserved         5

TAIFG_HND                     ; Vector 10: TAIFG Flag
          ...                 ; Task starts here
          RETI                                             5

CCIFG_2_HND                   ; Vector 4: TACCR2
          ...                 ; Task starts here
          RETI                ; Back to main program       5

CCIFG_1_HND                   ; Vector 2: TACCR1
          ...                 ; Task starts here
          RETI                ; Back to main program       5
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

TAIV

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	0	0	0	TAIVx			0
r0	r0	r0	r0	r–(0)	r–(0)	r–(0)	r0

TAIVx Bits 15-0 Timer_A interrupt vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2	TACCR2 CCIFG	
06h	Capture/compare 3†	TACCR3 CCIFG	
08h	Capture/compare 4†	TACCR4 CCIFG	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	–	
0Eh	Reserved	–	Lowest

† Timer1_A5 only

Timer_A Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control Timer0_A3 Control	TACTL/ TA0CTL	Read/write	0160h	Reset with POR
Timer_A counter Timer0_A3 counter	TAR/ TA0R	Read/write	0170h	Reset with POR
Timer_A capture/compare control 0 Timer0_A3 capture/compare control 0	TACCTL0/ TA0CCTL	Read/write	0162h	Reset with POR
Timer_A capture/compare 0 Timer0_A3 capture/compare 0	TACCR0/ TA0CCR0	Read/write	0172h	Reset with POR
Timer_A capture/compare control 1 Timer0_A3 capture/compare control 1	TACCTL1/ TA0CCTL1	Read/write	0164h	Reset with POR
Timer_A capture/compare 1 Timer0_A3 capture/compare 1	TACCR1/ TA0CCR1	Read/write	0174h	Reset with POR
Timer_A capture/compare control 2 Timer0_A3 capture/compare control 2	TACCTL2/ TA0CCTL2	Read/write	0166h	Reset with POR
Timer_A capture/compare 2 Timer0_A3 capture/compare 2	TACCR2/ TA0CCR2	Read/write	0176h	Reset with POR
Timer_A interrupt vector Timer0_A3 interrupt vector	TAIV/ TA0IV	Read only	012Eh	Reset with POR

Demo #1 (CCR0, CONT, TIMERA0)

```
//*******************************************************************************
//  MSP430xG46x Demo - Timer_A, Toggle P5.1, TACCR0 Cont. Mode ISR, DCO SMCLK
//
//  Description: Toggle P5.1 using software and TA_0 ISR. Toggles every
//  50000 SMCLK cycles. SMCLK provides clock source for TACLK. During the
//  TA_0 ISR, P5.1 is toggled and 50000 clock cycles are added to TACCR0.
//  TA_0 ISR is triggered every 50000 cycles. CPU is normally off and
//  used only during TA_ISR.
//  ACLK = 32.768kHz, MCLK = SMCLK = TACLK = Default DCO
//
//           MSP430xG461x
//         ---------------
//     /|\|              XIN|-
//      | |                 |   32kHz
//      --|RST          XOUT|-
//        |                 |
//        |             P5.1|-->LED
//
//  K. Quiring/ M. Mitchell
//  Texas Instruments Inc.
//  October 2006
//  Built with CCE Version: 3.2.0 and IAR Embedded Workbench Version: 3.41A
//*******************************************************************************
```

Demo #1 (CCR0, CONT, TIMERA0)

```
#include <msp430xG46x.h>

void main(void)
{
  volatile unsigned int i;

  WDTCTL = WDTPW +WDTHOLD;                  // Stop WDT
  FLL_CTL0 |= XCAP14PF;                     // Configure load caps

  // Wait for xtal to stabilize
  do
  {
  IFG1 &= ~OFIFG;                           // Clear OSCFault flag
  for (i = 0x47FF; i > 0; i--);             // Time for flag to set
  }
  while ((IFG1 & OFIFG));                   // OSCFault flag still set?

  P5DIR |= 0x02;                            // P5.1 output
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 50000;
  TACTL = TASSEL_2 + MC_2;                  // SMCLK, continuous mode
  _BIS_SR(LPM0_bits + GIE);                 // Enter LPM0 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P5OUT ^= 0x02;                            // Toggle P5.1
  TACCR0 += 50000;                          // Add Offset to TACCR0
}
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Demo #2 (CCR0, UP)

```
//*******************************************************************************
//  MSP430xG46x Demo - Timer_A, Toggle P5.1, TACCR0 Up Mode ISR, DCO SMCLK
//
//  Description: Toggle P5.1 using software and TA_0 ISR. Timer_A is
//  configured for up mode, thus the timer overflows when TAR counts
//  to TACCR0. In this example, TACCR0 is loaded with 20000.
//  ACLK = 32.768kHz, MCLK = SMCLK = TACLK = Default DCO
//
//           MSP430xG461x
//         ---------------
//     /|\|              XIN|-
//      | |                 |   32kHz
//      --|RST          XOUT|-
//        |                 |
//        |           P5.1|-->LED
//
//  K. Quiring/ M. Mitchell
//  Texas Instruments Inc.
//  October 2006
//  Built with CCE Version: 3.2.0 and IAR Embedded Workbench Version: 3.41A
//*******************************************************************************
```

Demo #2 (CCR0, UP)

```
#include <msp430xG46x.h>
void main(void) {
  volatile unsigned int i;
  WDTCTL = WDTPW +WDTHOLD;                  // Stop WDT
  FLL_CTL0 |= XCAP14PF;                     // Configure load caps
  // Wait for xtal to stabilize
  do {
    IFG1 &= ~OFIFG;                           // Clear OSCFault flag
    for (i = 0x47FF; i > 0; i--);             // Time for flag to set
  }
  while ((IFG1 & OFIFG));                   // OSCFault flag still set?
  P5DIR |= 0x02;                            // P5.1 output
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 20000;
  TACTL = TASSEL_2 + MC_1;                  // SMCLK, up mode
  _BIS_SR(LPM0_bits + GIE);                 // Enter LPM0 w/ interrupt
}
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void) {
  P5OUT ^= 0x02;                            // Toggle P5.1 using exclusive-OR
}
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Demo #3 (Overflow ISR)

```
//*******************************************************************************
//  MSP430xG46x Demo - Timer_A, Toggle P5.1, Overflow ISR, DCO SMCLK
//
//  Description: This program toggles P5.1 using software and the Timer_A
//  overflow ISR. In this example an ISR triggers when TA overflows.
//  Inside the ISR P5.1 is toggled. Toggle rate is 16Hz when using default
//  FLL+ register settings and an external 32kHz watch crystal.
//  ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = default DCO = 32 x ACLK = 1048576Hz
//  //* An external watch crystal between XIN & XOUT is required for ACLK *//
//
//           MSP430xG461x
//         ---------------
//     /|\|              XIN|-
//      | |                 | 32kHz
//      --|RST          XOUT|-
//        |                 |
//        |             P5.1|-->LED
//
//  K. Quiring/ M. Mitchell
//  Texas Instruments Inc.
//  October 2006
//  Built with CCE Version: 3.2.0 and IAR Embedded Workbench Version: 3.41A
//*******************************************************************************
```


LaCASA

Demo #3 (Overflow ISR)

```
#include <msp430xG46x.h>

void main(void)
{
  volatile unsigned int i;

  WDTCTL = WDTPW +WDTHOLD;                  // Stop WDT
  FLL_CTL0 |= XCAP14PF;                     // Configure load caps

  // Wait for xtal to stabilize
  do
  {
  IFG1 &= ~OFIFG;                           // Clear OSCFault flag
  for (i = 0x47FF; i > 0; i--);             // Time for flag to set
  }
  while ((IFG1 & OFIFG));                   // OSCFault flag still set?

  FLL_CTL0 |= XCAP14PF;                     // Configure load caps
  P5DIR |= 0x02;                            // Set P5.1 to output direction
  TACTL = TASSEL_2 + MC_2 + TAIE;           // SMCLK, cont. mode, interrupt

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

// Timer_A3 Interrupt Vector (TAIV) handler
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A (void)
{
  switch( TAIV )
  {
    case  2: break;                         // TACCR1 not used
    case  4: break;                         // TACCR2 not used
    case 10: P5OUT ^= 0x02;                 // overflow
             break;
  }
}
```