CS4101 嵌入式系統概論

Timers and Clocks

Prof. Chung-Ta King

Department of Computer Science

National Tsing Hua University, Taiwan

Materials from *MSP430 Microcontroller Basics*, John H. Davies, Newnes, 2008

Recall the Container Thermometer

- Container thermometer: monitor the temperature of the interior of a container
  - Monitor the temperature every 5 minutes
  - Flash LED alarm at 1 Hz
  - If the temperature rises above a threshold, flash the LED alarm at 3 Hz and notify backend server
  - If the temperature drops below a threshold, return the LED alarm to normal and notify the server

Need to know exact time!

Time-based Control

Many embedded systems are used to control things based on time or that have time constraints

- Traffic light controller
- Power meter
- Pacemaker (心跳節律器)
- Subway collision avoidance system
- Airbag
- …

How to track real (wall clock) time?

Recall First MSP430 Program

```
#include <msp430.h>
void main(void) {
  WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
  P1DIR |= 0x41;  // set P1.0 & 6 to outputs
                  //(red & green LEDs)
  for(;;) {
     volatile unsigned int i;
     P1OUT ^= 0x41; // Toggle P1.0 & 6 using XOR
     i = 50000; // Delay
     do (i--);
     while (i != 0);
  }
}
```

How much time?

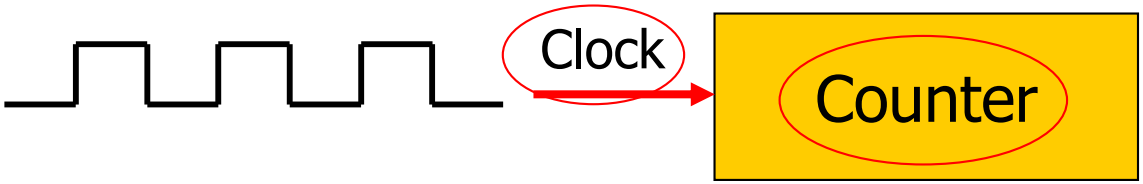
Problems Regarding Time

- Using software delay loops
  - Waste of processor because it is not available for other operations
  - Difficult to translate into actual time
  - Given a time for the delay, difficult to translate into number of iterations
  - The delays are unpredictable, e.g., compiler optimization, interrupts

We need an independent reference of time!

Reference of Time

- The simplest hardware to provide a reference of time is a counter that counts every fixed unit of time → *timer*

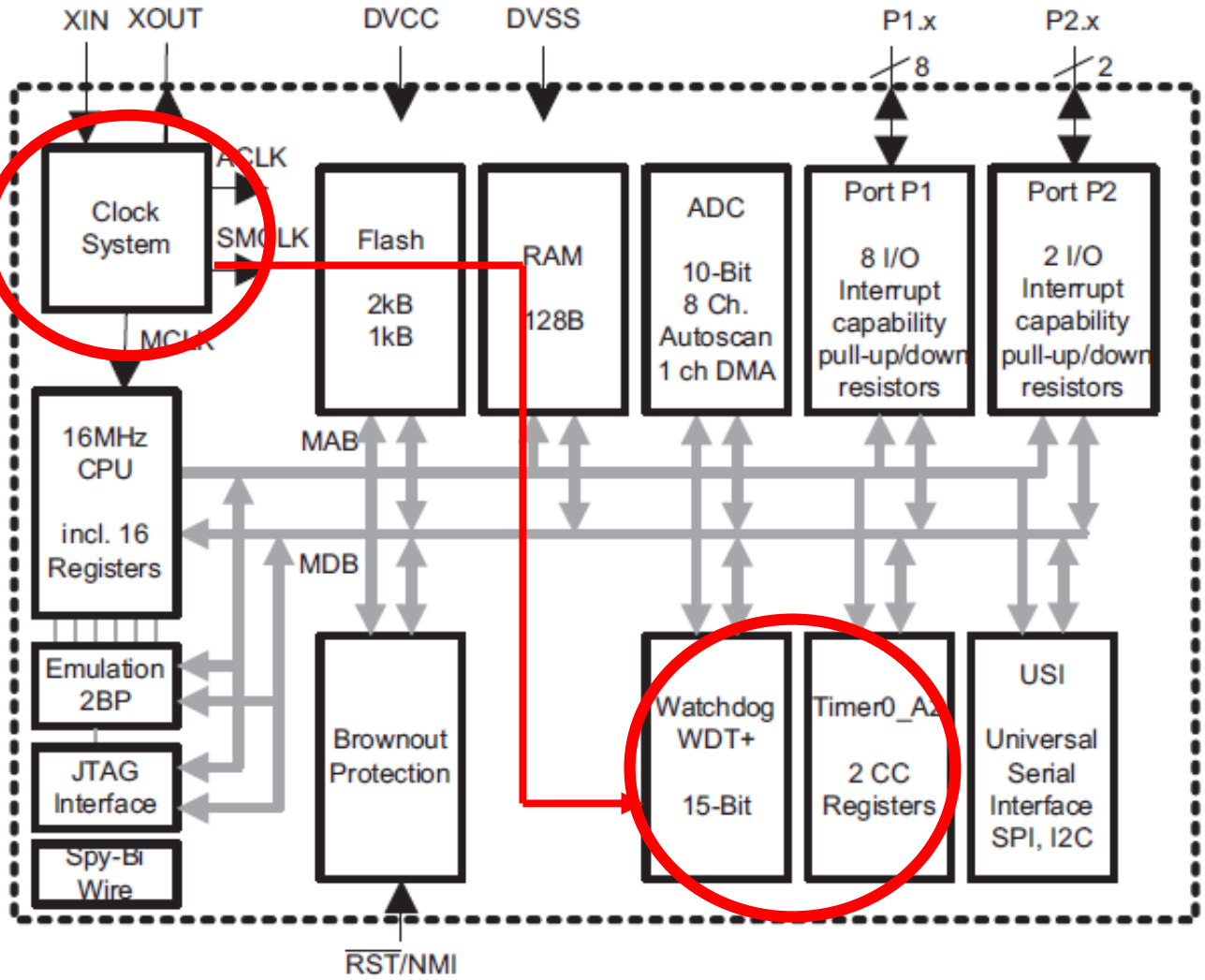


  - The actual time can be obtained by multiplying the counter with the clock interval time

    → The accuracy and stability of the clock is critical

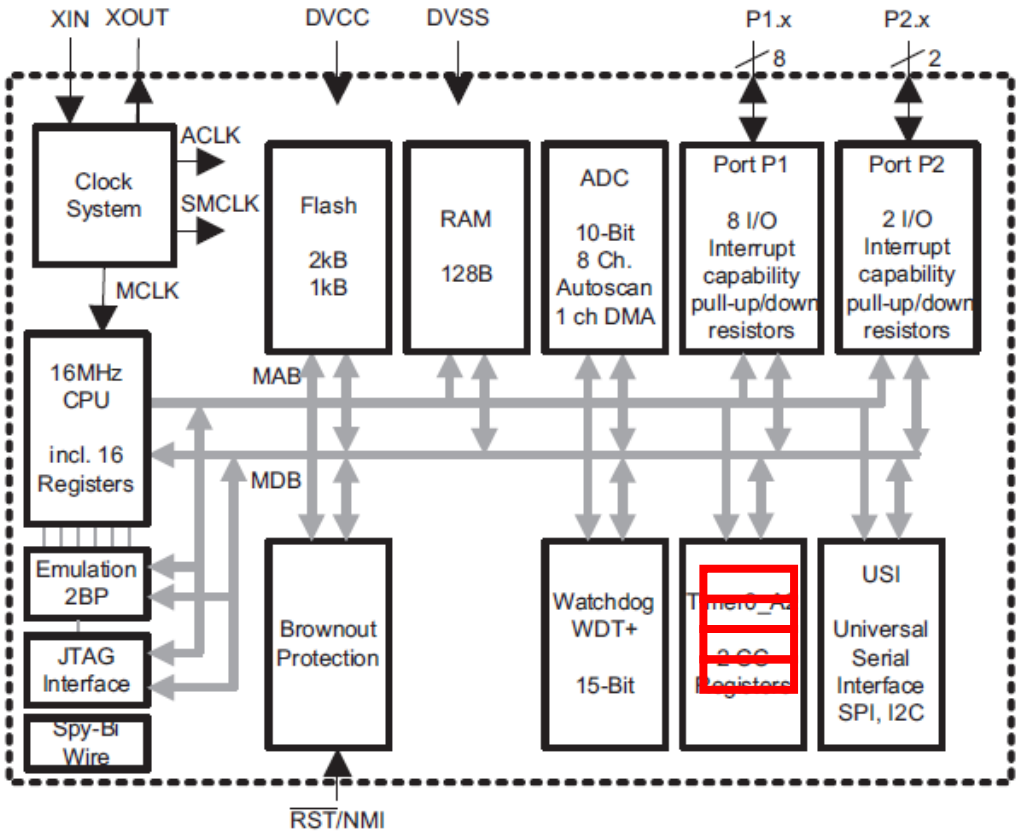
Where to put the timers?

Make Timer an IO Device!



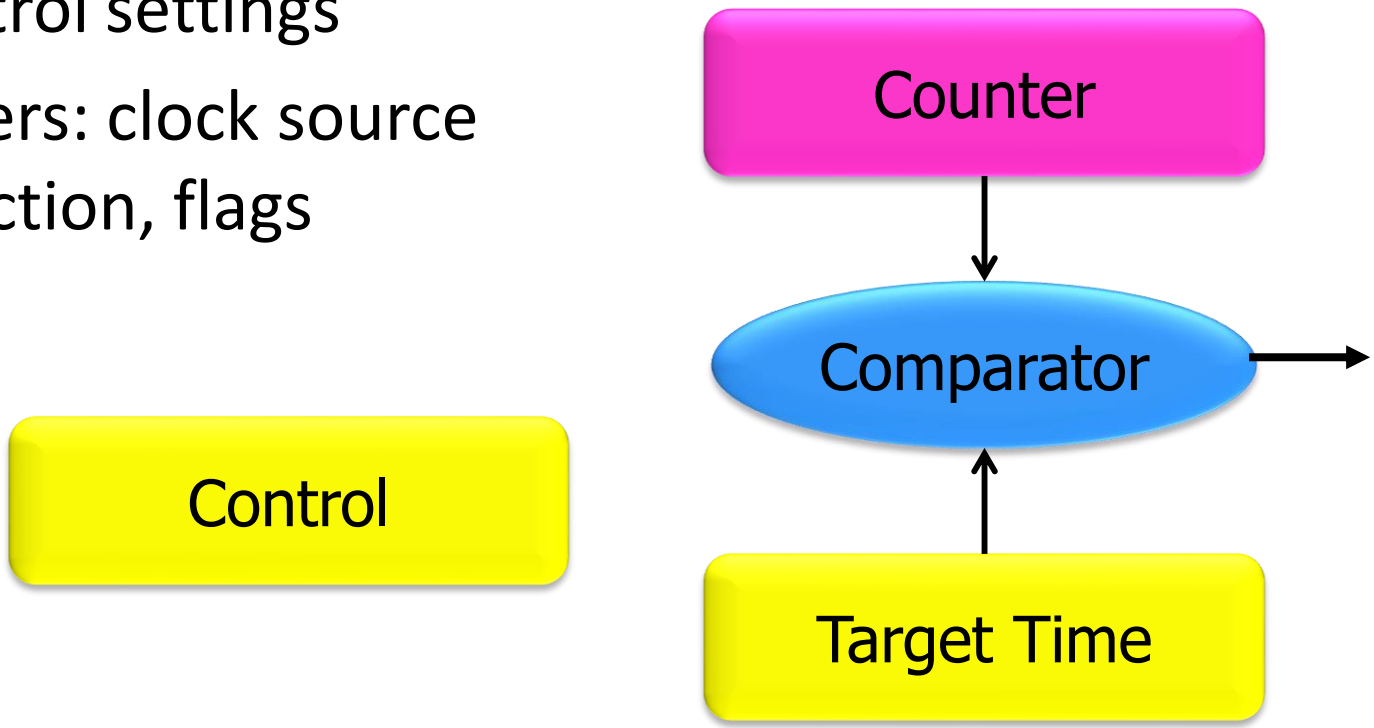
Timers Being IO Devices

- Have internal registers with addresses in the memory space for the CPU to access



Typical Registers in a Timer

- The counter itself
- Target for counting
- Control settings
- Others: clock source selection, flags



Outline

- Basic concepts of timers
- MSP430 timers
- An example of using MSP430 Timer_A
- Clocks in MSP430

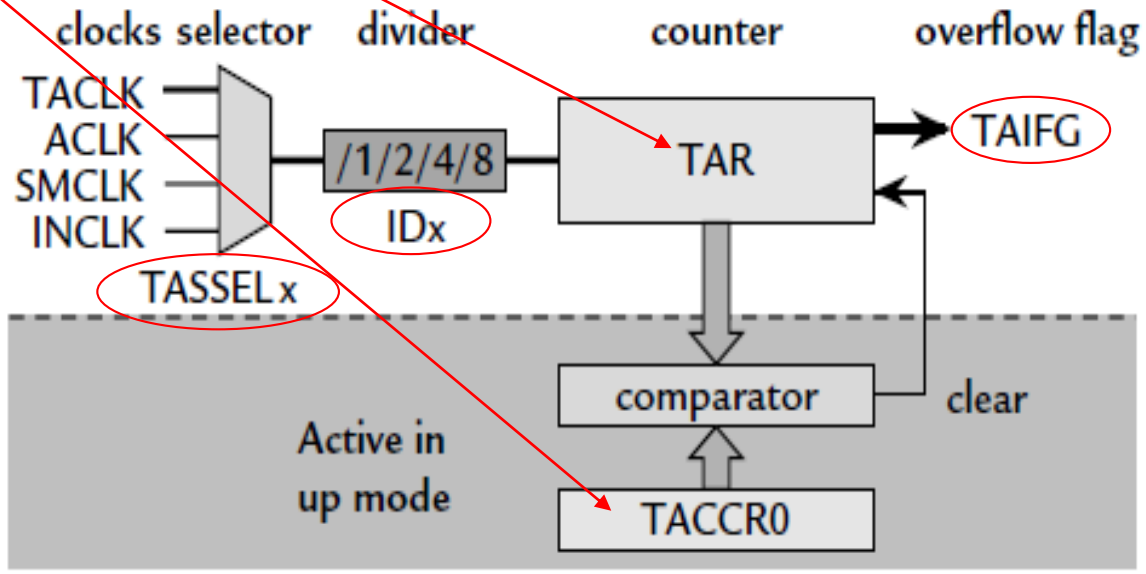
MSP430 Timers

Contain several timers, including:

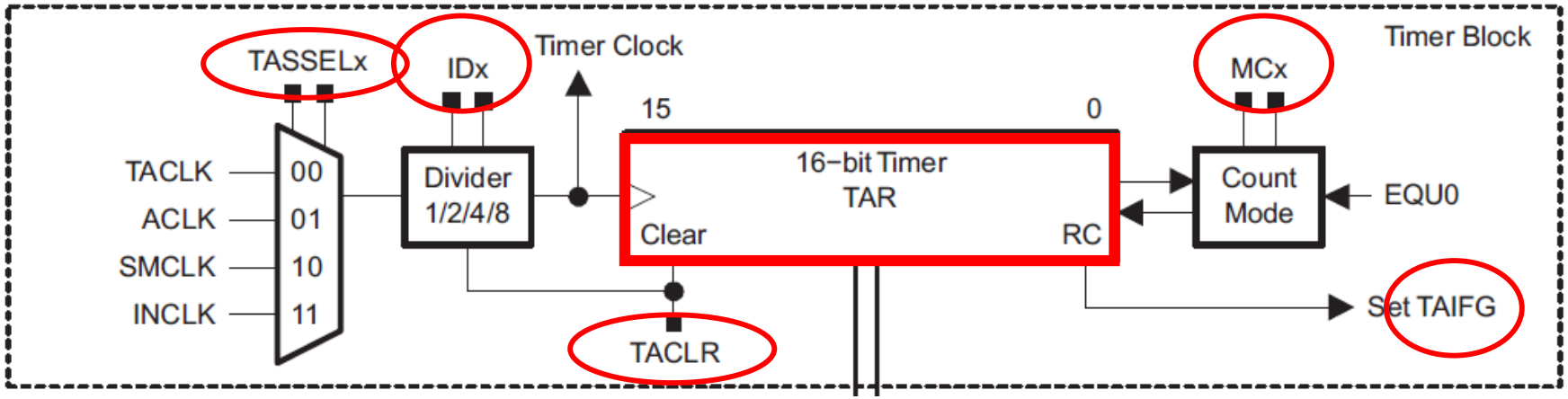
- Timer_A
  - A 16-bit counter, TAR, with three capture/compare registers.
- Watchdog timer
  - Count up and reset MSP430 when it reaches its limit
  - The code must keep clearing the counter before the limit is reached to prevent a reset
  - Protect system against failure of software, such as unintended, infinite loops

Registers in Timer_A

- TAR (0170h): the counter itself
- TACCR0 (0172h): target for counting
- TACTL (0160h): control settings
- Others: clock source selection, flags



Inside Timer_A

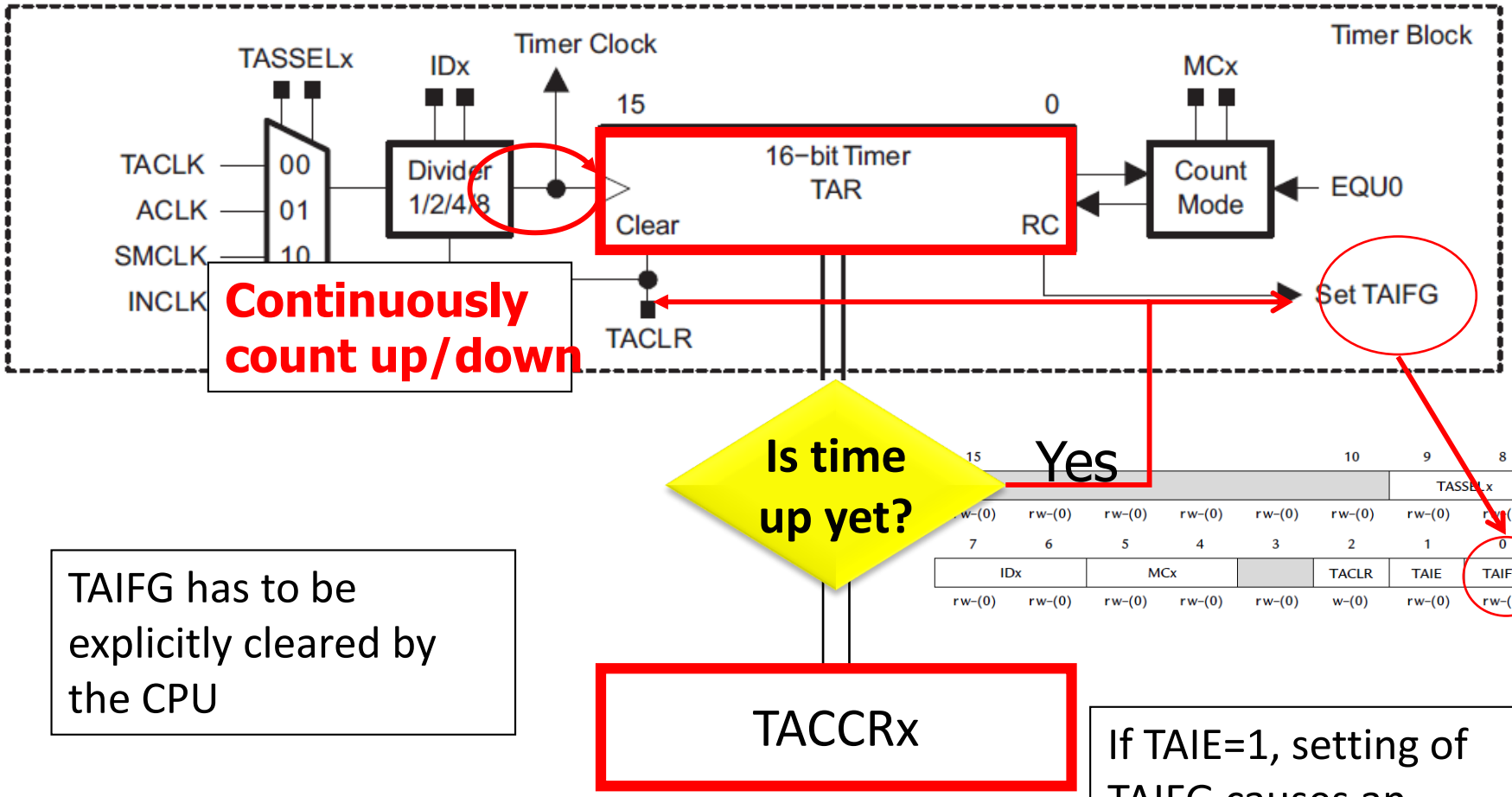


- Timer_A Control Register: TACTL

15	14	13	12	11	10	9	8
						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			TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Typical Operations of Timer_A

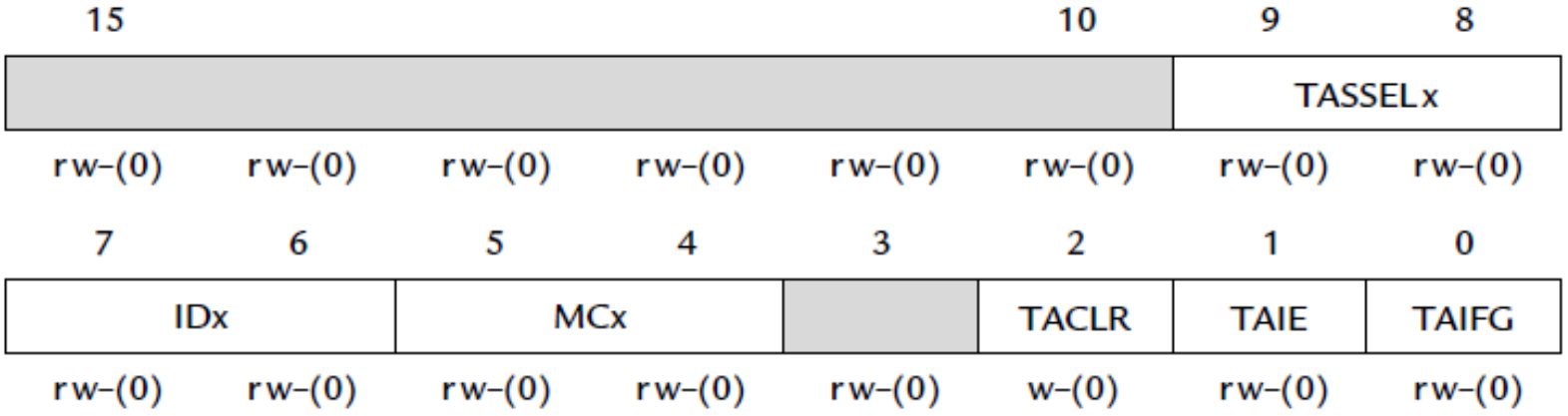


TAIFG has to be explicitly cleared by the CPU

If TAIE=1, setting of TAIFG causes an interrupt to the CPU

Timer_A Control Register (TACTL)

- TASSELx: Timer_A clock source select
- IDx: input divider
- MCx: mode control
- TACLR: Timer_A clear
- TAIE: Timer_A interrupt enable
- TAIFG: Timer_A interrupt flag



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

```
TACTL = TASSEL_2 + MC_1;                  // src from SMCLK, up mode
```

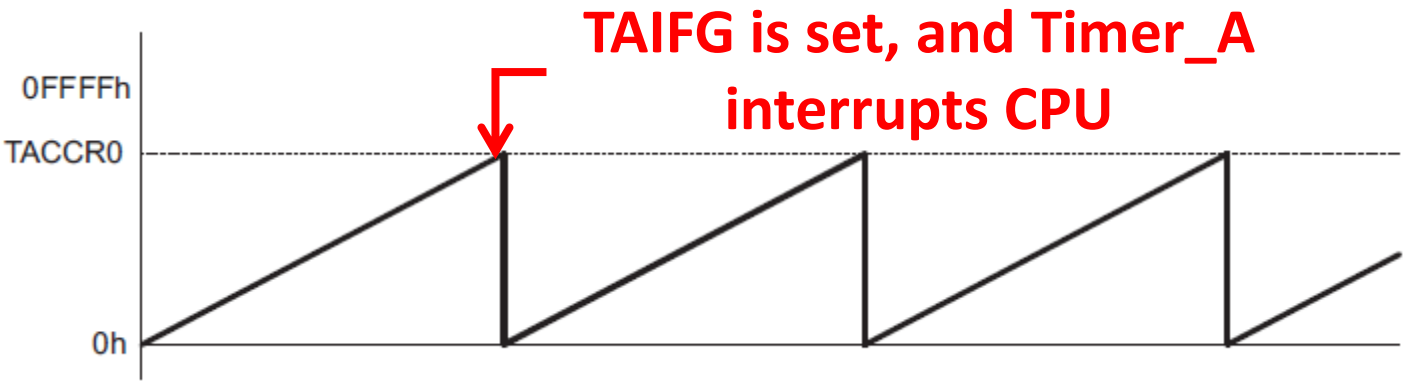
Timer Mode

- MCx=00: Stop mode
  - The timer is halted
- MCx=01: Up mode
  - The timer repeatedly counts from 0 to TACCR0
- MCx=10: Continuous mode
  - The timer repeatedly counts from 0 to 0FFFFh
- MCx=11: Up/down mode
  - The timer repeatedly counts from 0 to TACCR0 and back down to 0

Up Mode

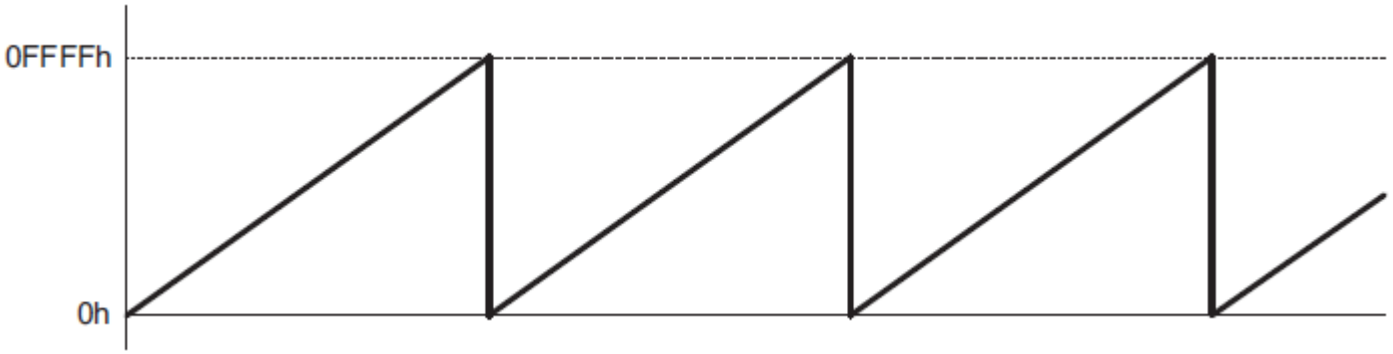
The up mode is used if the timer period must be **different** from **0FFFFh** counts.

1. Timer period 100 → store 99 to TACCR0
2. When TACCR0 == 99, set TACCR0 CCIFG interrupt flag
3. Reset timer to 0 and set TAIFG interrupt flag



Continuous Mode

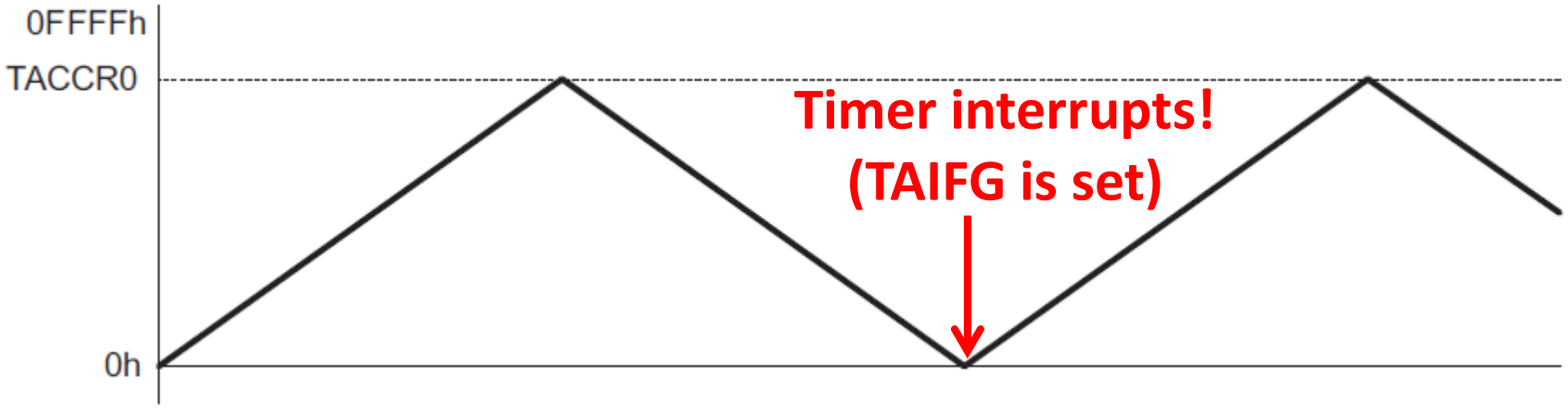
- In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero
- The TAIFG interrupt flag is set when the timer resets from 0FFFFh to zero



Up/Down Mode

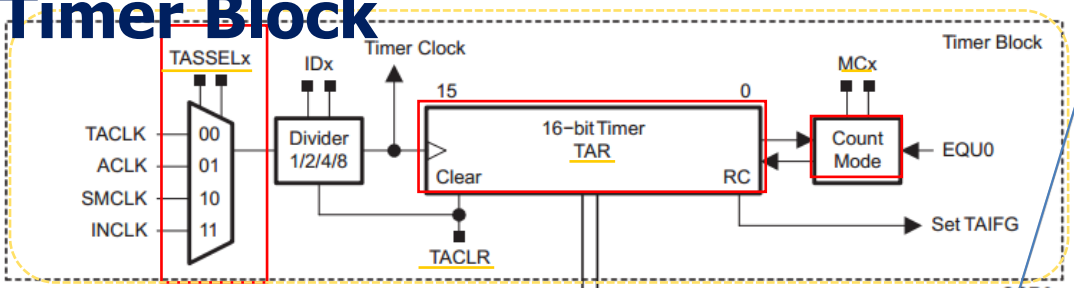
- The up/down mode is used if the timer period must be different from 0FFFFh counts, and if a *symmetrical pulse generation* is needed.

  → **The period is twice the value in TACCR0**

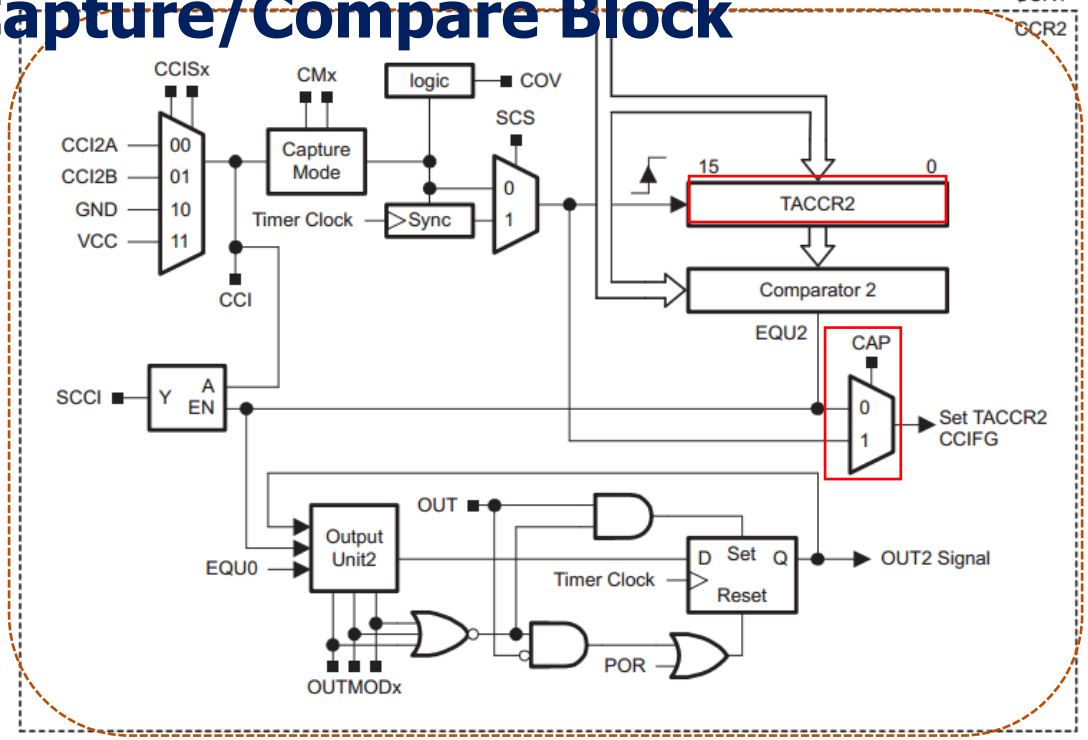


Timer_A Capture/Compare Block

Timer Block



Capture/Compare Block



- May contain several Capture/Compare Blocks
- Each Capture/Compare Block is controlled by a control register, **TACCTLx**
- Inside each Capture/Compare Block, the Capture/Compare Register, **TACCRx**, holds the count to configure the timer

Modes of Capture/Compare Block

- Compare mode:
  - Compare the value of TAR with the value stored in TACCRn and update an output when they match
- Capture mode: used to record time events
  - Records the “time” (value in TAR) at which the input changes in TACCRx
  - The input, usually CCIxA and CCIxB, can be either external or internal from another peripheral or software, depending on board connections

```
TACCR0 = 24000; // represent 2 sec with 12kHz clk src
```

TACCTL

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.

TACCTL cont'd

CAP	Bit 8	Capture mode
		0 Compare mode
		1 Capture mode
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

Sample Code 1 for Timer_A

- Goal: simplest way to flash an LED at 1 Hz
  - Need an event to trigger the flashing
    → counter (TAR) overflow
  - Need a way to detect the event
    → CPU polling
- How to make TAR overflow at 1 Hz?
  - Use SMCLK clock (discussed later) at 800 KHz
  - When TAR (16 bits) overflows, it has counted 2^{16}, equivalent to a period of 2^{16}/800KHz ≈ 0.08 sec
  - Divide the frequency of the clock by 8 to give a period of about 0.66 sec → close enough!
  - Continuously count up; on overflow return to 0

Sample Code 1 for Timer_A

```
#define LED1 BIT0
void main(void) {
  WDTCTL = WDTPW|WDTHOLD; // Stop watchdog timer
  P1OUT = ~LED1;
  P1DIR = LED1;
  TACTL = MC_2|ID_3|TASSEL_2|TACLR; //Setup Timer_A
  for (;;) { // Loop forever
    while (TACTL_bit.TAIFG == 0) { // Wait overflow
    } // CPU polling and doing nothing
    TACTL_bit.TAIFG = 0; // Clear overflow flag
    P1OUT ^= LED1;       // Toggle LEDs
  } // Back around infinite loop
}
```

Sample Code Settings Explained

The following symbols are defined in header file:

- MC_2: set MC of TACTL to 10 (continuous mode)
- ID_3: set ID of TACTL to 11 (divide freq. by 8)
- TASSEL_2: set TASSEL to 10 (use SMCLK)
- TACLR: clear the counter, the divider, and the direction of the count

Sample Code 2 for Timer_A

- Can have more accurate time if we can control the amount to count
  - The maximum desired value of the count is programmed into TACCR0
  - TAR starts from 0 and counts up to the value in TACCR0, after which it returns to 0 and sets TAIFG
  - Thus the period is TACCR0+1 counts
  - With SMCLK (800KHz) divided down to 100 KHz, we need 50,000 counts for a delay of 0.5 sec → store 49,999 in TACCR0

```
    TACCR0 = 49999;                          // Upper limit of count for TAR
    TACTL = MC_1|ID_3|TASSEL_2|TACLR;        // Set up and start Timer A
// "Up to CCR0" mode, divide clock by 8, clock from SMCLK, clear timer
```

Outline

- Basic concepts of timers
- MSP430 timers
- An example of using MSP430 Timer_A
- Clocks in MSP430

Theoretically, One Clock Is Enough

- A clock is a square wave signal whose edges trigger hardware
- A clock source, e.g. crystal, to drive CPU directly, which is divided down by a factor of 2 or 4 for the main bus and rest of circuit board
- But, systems have conflicting requirements
  - Low power, fast start/stop, accuracy

Different Requirements for Clocks

- Devices often in a low-power mode until some event occurs, then must wake up and handle event rapidly
  - Clock must get to be stabilized quickly
- Devices also need to keep track of real time: (1) can wake up periodically, or (2) time-stamp external events
- Therefore, two kinds of clocks often needed:
  - A fast clock to drive CPU, which can be started and stopped rapidly but need not be particularly accurate
  - A slow clock that runs continuously to monitor real time, which must use little power and be accurate

Different Requirements for Clocks

- Different clock sources also have different characteristics
  - Crystal: accurate and stable (w.r.t. temperature or time); expensive, delicate, drawing large current, external component, longer time to start up/stabilize
  - Resistor and capacitor (RC): cheap, quick to start, integrated within MCU and sleep with CPU; poor accuracy and stability
  - Ceramic resonator and MEMS clocks in between

Need multiple clocks

Clocks in MSP430

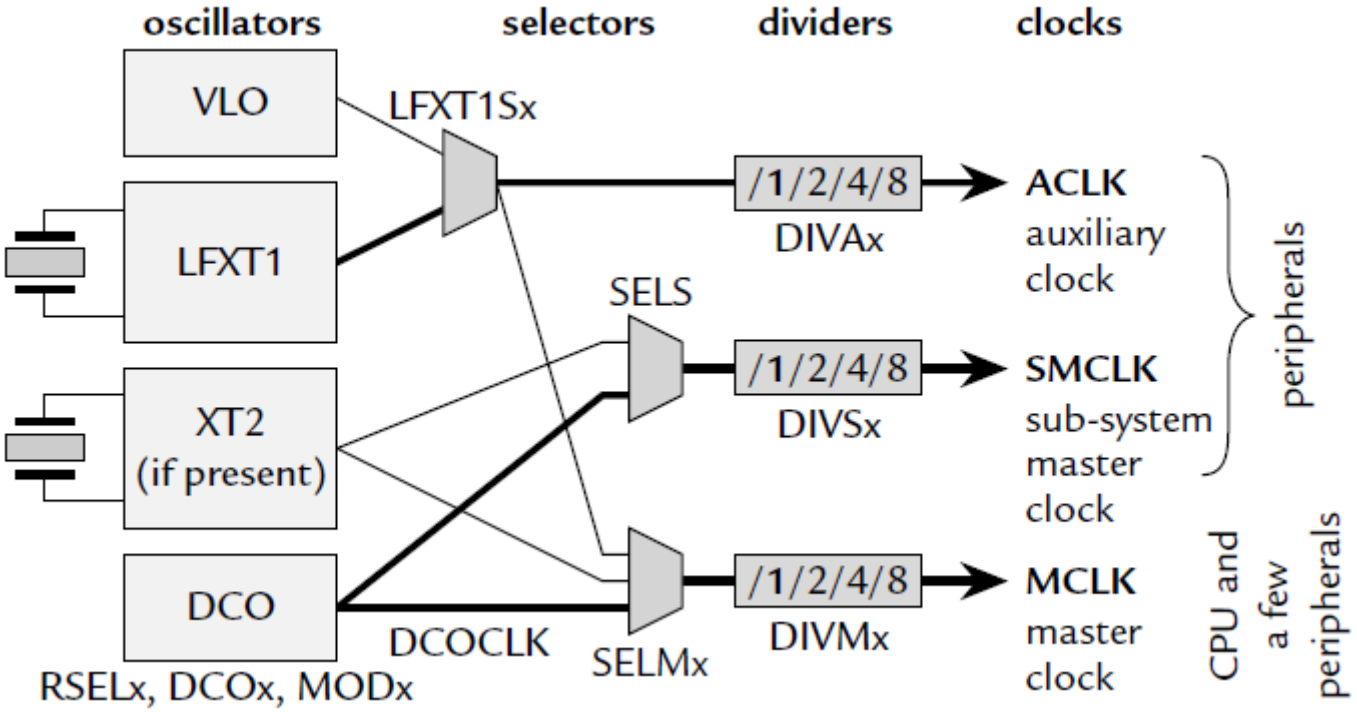
- MSP430 addresses the conflicting demands for high performance, low power, precise frequency by using 3 internal clocks, which can be derived from up to 4 sources
  - Master clock (MCLK): for CPU & some peripherals, normally driven by *digitally controlled oscillator* (DCO)
  - Subsystem master clock (SMCLK): distributed to peripherals, normally driven by DCO
  - Auxiliary clock (ACLK): distributed to peripherals, normally for real-time clocking, normally driven by a low-frequency crystal oscillator, typically at 32 KHz

Clock Sources

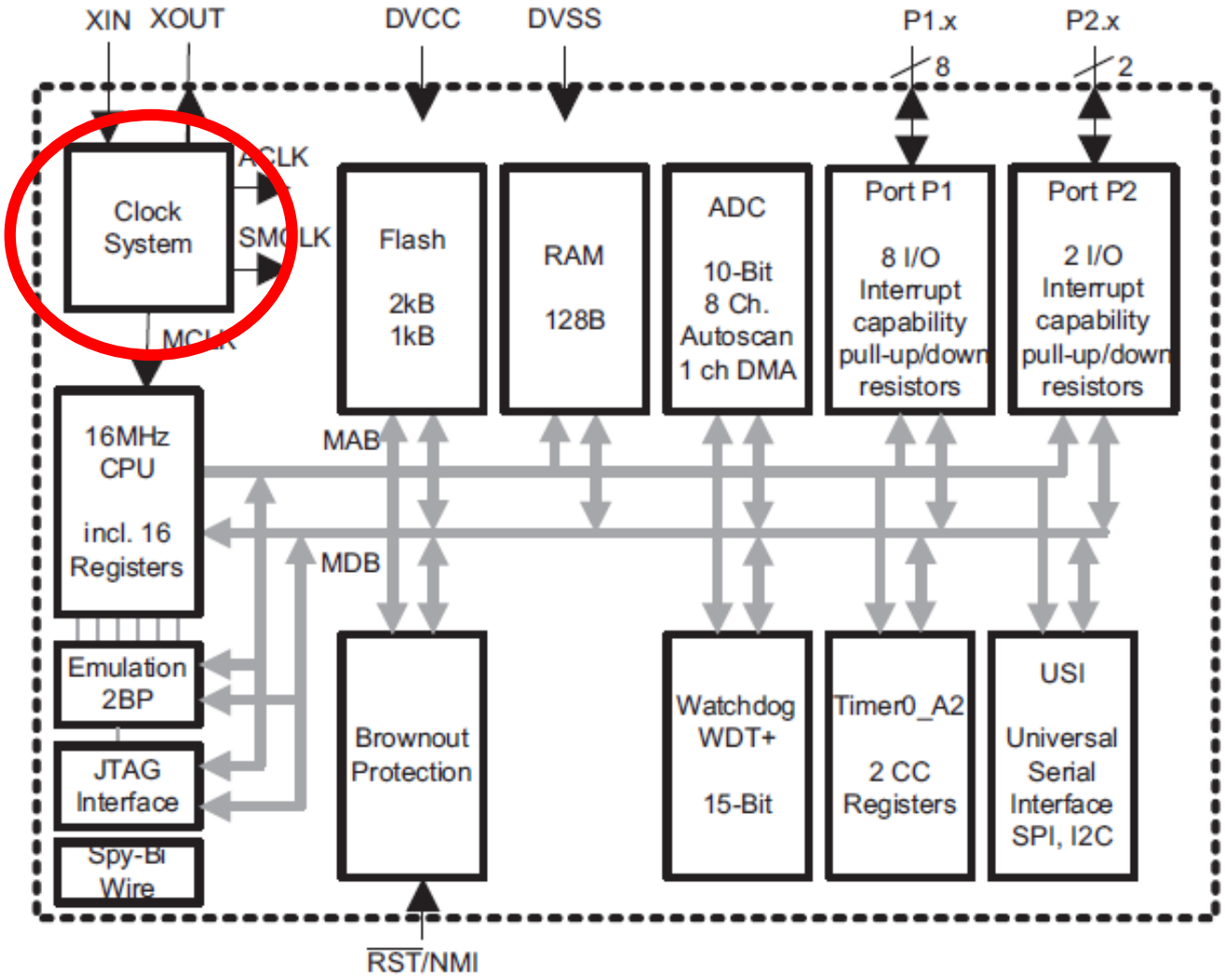
- Low- or high-frequency crystal oscillator, LFXT1:
  - External; used with a low- or high frequency crystal; an external clock signal can also be used; connected to MSP430 through XIN and XOUT pins
- High-frequency crystal oscillator, XT2:
  - External; similar to LFXT1 but at high frequencies
- Very low-power, low-frequency oscillator, VLO:
  - Internal at 12 KHz; alternative to LFXT1 when accuracy of a crystal is not needed; may not available in all devices
- Digitally controlled oscillator, DCO:
  - Internal; a highly controllable RC oscillator that starts fast

From Sources to Clocks

- Typical sources of clocks:
  - MCLK, SMCLK: DCO (typically at 1.1 MHz)
  - ACLK: LFXT 1 (typically at 32 KHz)



MSP430 Clock System



Controlling Clocks

- In MSP430, the Basic Clock Module is also an IO peripheral
- Being an IO peripheral, it can be controlled by registers, DCOCTL and BCSCTL1–3
  - DCOCTL (056h): configure DCO
  - BCSCTL1 (basic clock system control 1, 057h): configure ACLK
  - BCSCTL2 (basic clock system control 2, 058h): configure MCLK, SMCLK
  - BCSCTL3 (basic clock system control 3, 053h): control LFXT1/VLO

Control Registers for Clocks

Control Registers for Clock System

Table 5-1. Basic Clock Module+ Registers

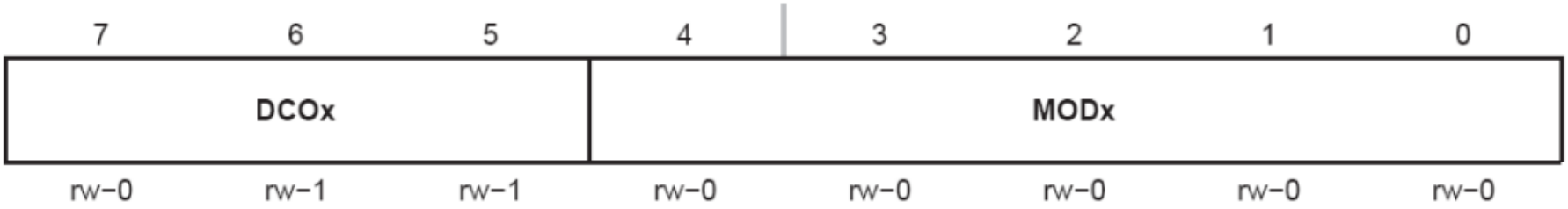
Register	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h with PUC
Basic clock system control 1	BCSCTL1	Read/write	057h	087h with POR(1)
Basic clock system control 2	BCSCTL2	Read/write	058h	Reset with PUC
Basic clock system control 3	BCSCTL3	Read/write	053h	005h with PUC(2)
SFR interrupt enable register 1	IE1	Read/write	000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	002h	Reset with PUC

(1) Some of the register bits are also PUC initialized (see Section 5.3.2).
(2) The initial state of BCSCTL3 is 000h in the MSP430AFE2xx devices.

- DCOCTL and BCSCTL1 combined define the frequency of DCO, among other settings

DCOCTL (at Memory Address 056h)

DCOCTL, DCO Control Register



DCOx	Bits 7-5	DCO frequency select. These bits select which of the eight discrete DCO frequencies of the RSELx setting is selected.
MODx	Bits 4-0	Modulator selection. These bits define how often the f_{DCO+1} frequency is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32−M used. Not useable when DCOx=7.

Tag-Length-Value

```
DCOCTL = CALDCO_1MHZ;          // Set DCO step + modulation
```

Tag-Length-Value

- Tag-Length-Value (TLV) stores device-specific information in the flash memory to set DCOCTL and BCSCTL1 for DCO frequency

DCO Calibration Data (Device Specific)

Label	Description	Offset
CALBC1_1MHZ	Value for the BCSCTL1 register for 1 MHz, T_A = 25°C	0x07
CALDCO_1MHZ	Value for the DCOCTL register for 1 MHz, T_A = 25°C	0x06
CALBC1_8MHZ	Value for the BCSCTL1 register for 8 MHz, T_A = 25°C	0x05
CALDCO_8MHZ	Value for the DCOCTL register for 8 MHz, T_A = 25°C	0x04
CALBC1_12MHZ	Value for the BCSCTL1 register for 12 MHz, T_A = 25°C	0x03
CALDCO_12MHZ	Value for the DCOCTL register for 12 MHz, T_A = 25°C	0x02
CALBC1_16MHZ	Value for the BCSCTL1 register for 16 MHz, T_A = 25°C	0x01
CALDCO_16MHZ	Value for the DCOCTL register for 16 MHz, T_A = 25°C	0x00

```
BCSCTL1 = CALBC1_1MHZ;                    // Set range
DCOCTL = CALDCO_1MHZ;
```

BCSCTL1

BCSCTL1, Basic Clock System Control Register 1

7	6	5	4	3	2	1	0
XT2OFF	**XTS**(1)(2)	**DIVAx**		**RSELx**			
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-1	rw-1

XT2OFF	Bit 7	XT2 off. This bit turns off the XT2 oscillator 0 XT2 is on 1 XT2 is off if it is not used for MCLK or SMCLK.
XTS	Bit 6	LFXT1 mode select. 0 Low-frequency mode 1 High-frequency mode
DIVAx	Bits 5-4	Divider for ACLK 00 /1 01 /2 10 /4 11 /8
RSELx	Bits 3-0	Range select. Sixteen different frequency ranges are available. The lowest frequency range is selected by setting RSELx=0. RSEL3 is ignored when DCOR = 1.

```
BCSCTL1 = CALBC1_1MHZ;                    // Set range
```

BCSCTL2

BCSCTL2, Basic Clock System Control Register 2

MCLK **SMCLK**

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR(1)(2)
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SELMx	Bits 7-6	Select MCLK. These bits select the MCLK source.	
		00	DCOCLK
		01	DCOCLK
		10	XT2CLK when XT2 oscillator present on-chip. LFXT1CLK or VLOCLK when XT2 oscillator not present on-chip.
		11	LFXT1CLK or VLOCLK
DIVMx	Bits 5-4	Divider for MCLK	
		00	/1
		01	/2
		10	/4
		11	/8
SELS	Bit 3	Select SMCLK. This bit selects the SMCLK source.	
		0	DCOCLK
		1	XT2CLK when XT2 oscillator present. LFXT1CLK or VLOCLK when XT2 oscillator not present
DIVSx	Bits 2-1	Divider for SMCLK	
		00	/1
		11	/8

```
BCSCTL2 |= SELM_3 + DIVM_3;          // MCLK = VLO/8
```

BCSCTL3

BCSCTL3, Basic Clock System Control Register 3

In MSP430G2231

7	6	5	4	3	2	1	0
XT2Sx		LFXT1Sx		XCAPx[1]		XT2OF[2]	LFXT1OF[1]
rw-0	rw-0	rw-0	rw-0	rw-0	rw-1	r0	r-(1)

XT2Sx Bits 7-6 XT2 range select. These bits select the frequency range for XT2.

00 0.4- to 1-MHz crystal or resonator
01 1- to 3-MHz crystal or resonator
10 3- to 16-MHz crystal or resonator
11 Digital external 0.4- to 16-MHz clock source

LFXT1Sx Bits 5-4 Low-frequency clock select and LFXT1 range select. These bits select between LFXT1 and VLO when XTS = 0, and select the frequency range for LFXT1 when XTS = 1.

When XTS = 0:
00 32768-Hz crystal on LFXT1
01 Reserved
10 VLOCLK (Reserved in MSP430x21x1 devices)
11 Digital external clock source

When XTS = 1 (Not applicable for MSP430x20xx devices)
00 0.4- to 1-MHz crystal or resonator
01 1- to 3-MHz crystal or resonator
10 3- to 16-MHz crystal or resonator

```
BCSCTL3 |= LFXT1S_2;     // Enable VLO as MCLK/ACLK src
```

01 Reserved

Interrupt Flag Register 1 (IFG1)

- OFIFG oscillator-fault flag is set when an oscillator fault (LFXT1OF) is detected.

IFG1, Interrupt Flag Register 1

7	6	5	4	3	2	1	0
						OFIFG	
						rw-1	

	Bits 7-2	These bits may be used by other modules. See device-specific data sheet.
OFIFG	Bit 1	Oscillator fault interrupt flag. Because other bits in IFG1 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 No interrupt pending
		1 Interrupt pending
	Bits 0	This bit may be used by other modules. See device-specific data sheet.

```
IFG1 &= ~OFIFG;                    // Clear OSCFault flag
```

Recall Sample Code for Timer_A

- Flash red LED at 1 Hz if SMCLK at 800 KHz

```
#include <msp430g2553.h>
#define LED1 BIT0
void main (void) {
  WDTCTL = WDTPW|WDTHOLD; // Stop watchdog timer
  P1OUT = ~LED1;
  P1DIR = LED1;
  TACCR0 = 49999;
  TACTL = MC_1|ID_3|TASSEL_2|TACLR; //Setup Timer_A
  //up mode, divide clk by 8, use SMCLK, clr timer
  for (;;) { // Loop forever
    while (!(TACTL&TAIFG)) { // Wait time up
    } // doing nothing
    TACTL &= ~TAIFG;  // Clear overflow flag
    P1OUT ^= LED1;    // Toggle LEDs
  } // Back around infinite loop
}
```

Sample Code for Setting Clocks

- Set DCO to 1MHz, enable crystal

```
#include <msp430g2231.h> (#include <msp430g2553.h> )
void main(void) {
  WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
  if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
     while(1);   // If TLV erased, TRAP!
  BCSCTL1 = CALBC1_1MHZ;     // Set range
  DCOCTL = CALDCO_1MHZ;
  P1DIR = 0x41;  // P1.0 & 6 outputs (red/green LEDs)
  P1OUT = 0x01;  // red LED on
  BCSCTL3 |= LFXT1S_0; // Enable 32768 crystal
  IFG1 &= ~OFIFG;// Clear OSCFault flag
  P1OUT = 0;     // red LED off
  BCSCTL2 |= SELS_0 + DIVS_3; // SMCLK = DCO/8
  // infinite loop to flash LEDs
}
```