TIMERS IN MSP430

A timer is a counting mechanism that is tied to some type of regular interval provided by a clock. The Timer_A peripheral is a 16-bit timer. This means that the timer counts from 0 in binary to 0b1111111111111111, or0xFFFF in hex, 65,535 in decimal. Simple enough, and very useful when we know how to manipulate the behavior of the timer.

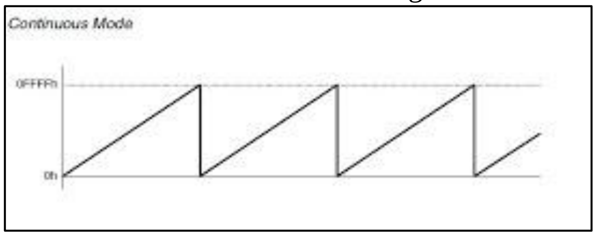
The main applications of timers are to:

1. generate events of fixed time-period
2. allow periodic wakeup from sleep of the device
3. count transitional signal edges
4. replace delay loops allowing the CPU to sleep
5. between operations, consuming less power
6. maintain synchronization clocks

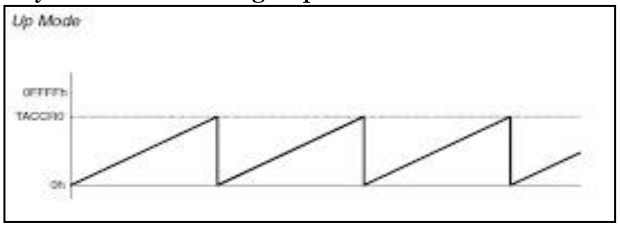
Timer Modes:

There are three modes to operate Timer_A, which one we use depends entirely on the application:

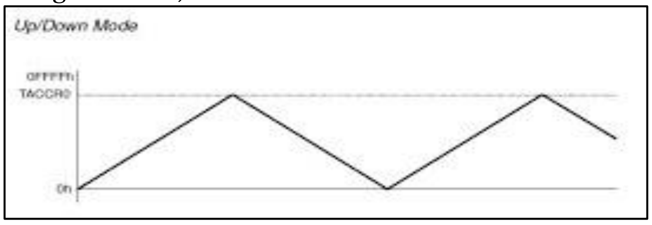
- The first mode is what we call the continuous mode: Timer_A acts just like a 16-digit, binary odometer; it counts from 0 to 0xFFFF and then "rolls over" to 0 again.



- The second mode is called up mode; here, just like in continuous mode, it counts up and rolls over to 0. This mode, however, lets you choose how high up the timer counts before rolling over.

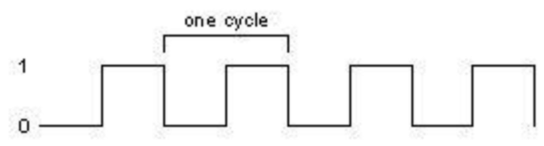


- The third mode, up/down mode, is similar to up mode in that you can program the upper limit. It differs in thatratherthan rolling over to 0, it turns around and counts down to 0.



Clocks:

A clock signal controls timing on a computer, a typical digital clock signal is below.



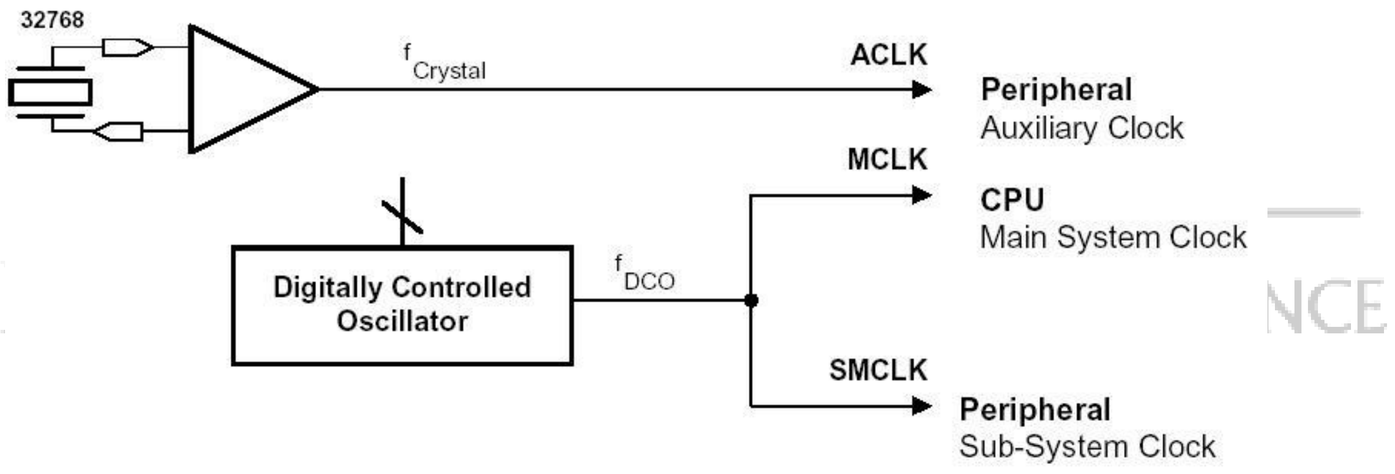
MSP430 CPU and other system devices use three internal clocks:

1. Master clock, MCLK, is used by the CPU and a few peripherals.
2. Subsystem master clock, SMCLK, is distributed to peripherals.
3. Auxiliary clock, ACLK, is also distributed to peripherals.

Typically SMCLK runs at the same frequency as MCLK, both in the megahertz range.

ACLK is often derived from a watch crystal and therefore runs at a much lower frequency. Most peripherals can select their clock from either SMCLK or ACLK.

For the MSP430 processor, both the MCLK and SMCLK clocks are supplied by an internal digitally controlled oscillator (DCO), which runs at about 1.1 MHz.



DCO - Digitally Controlled Oscillator internal runs at about 1Mhz.

The frequency of the DCO is controlled through sets of bits in the module’s registers at three levels. There calibrated frequencies of 1, 8, 12, and 16 MHz. To change frequency simply copy values into the clock module registers.

The following sets the DCO used by the CPU to 1MHz, which seems to be the default value:

```
BCSCTL1 = CALBC1_1MHZ;     // Set range

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

Clock Sources:

- **LFXT1CLK** : Low-frequency/high-frequency oscillator
- **XT2CLK** : Optional high-frequency oscillator
- **DCOCLK** : Internal digitally controlled oscillator (DCO)

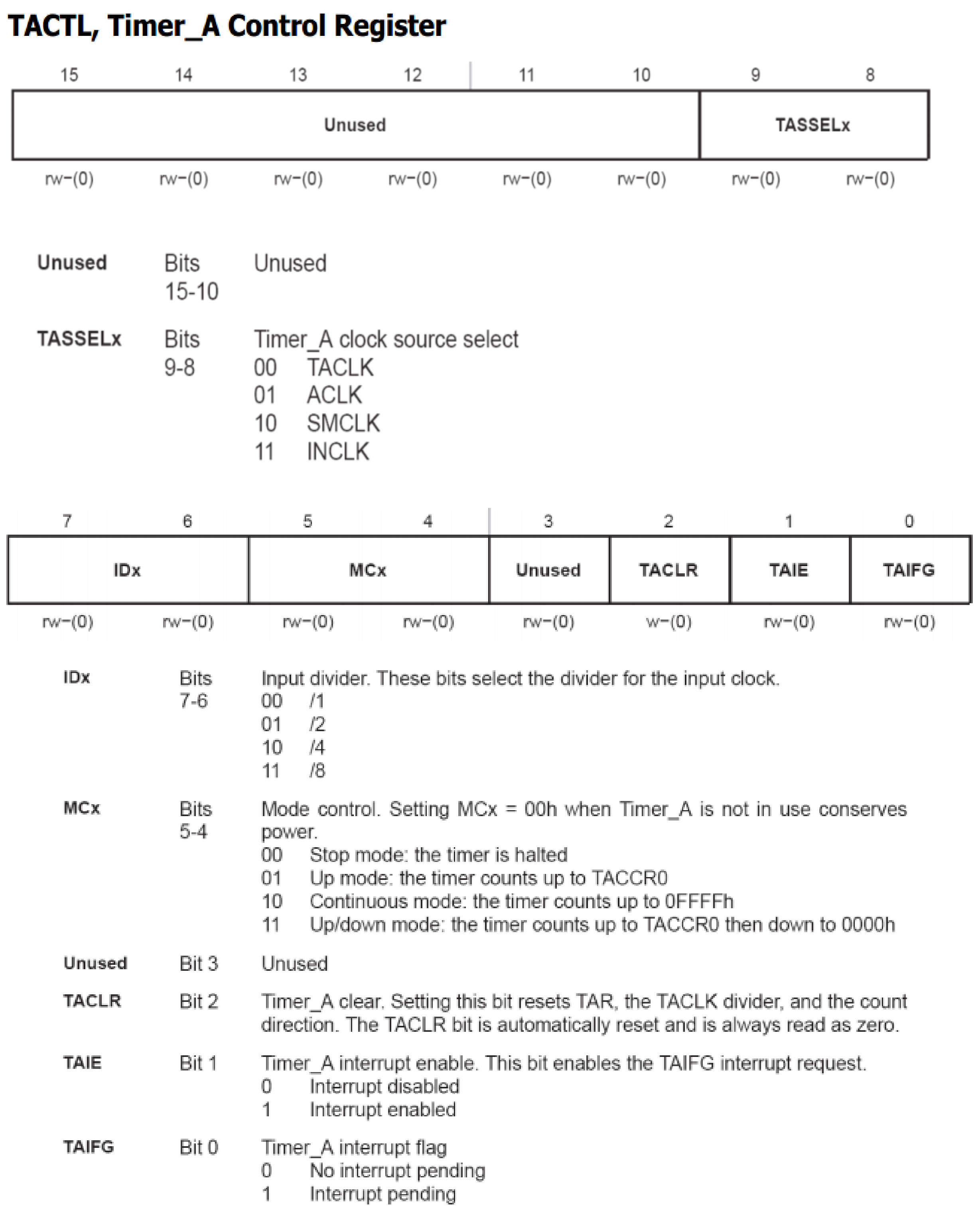
Registers

With so many ways to use Timer_A, you can probably imagine it corresponds to a lot of registers in the MSP430 to use and control the timer:

- TACTL -- The Timer_A Control Register is used to set up the link between the timer and a clock and select the mode used.
  - The TASSELx bits (8 and 9) tell the timer which clock to use as its source.

```
TASSEL_0  = TACLK
TASSEL_1  = ACLK @ 12KHz.
TASSEL_2  = SMCLK @ 1MHz
TASSEL_3  = INCLK
```

  - The clock frequency can be divided by a further factor of 2, 4, or 8 using the IDx bits (6 and 7). (Note that this is further division to any divisions done from the clock source for the clock itself; you could potentially have a total division of up to 64 from your clock source for this peripheral.)
  - The MCx bits (4 and 5) select the particular mode for the timer to use. Note particularly that setting these bits to 0 (the default setting on POR) halts the timer completely.
  - TACLR is bit 2. If you write a 1 to this bit, it resets the timer. The MSP430 will automatically reset this bit to zero after resetting the timer.
  - TAIE and TAIFG (bits 0 and 1) control the ability of the timer to trigger interrupts.



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)

Unused	Bits 15-10	Unused
TASSELx	Bits 9-8	Timer_A clock source select 00 TACLK 01 ACLK 10 SMCLK 11 INCLK

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

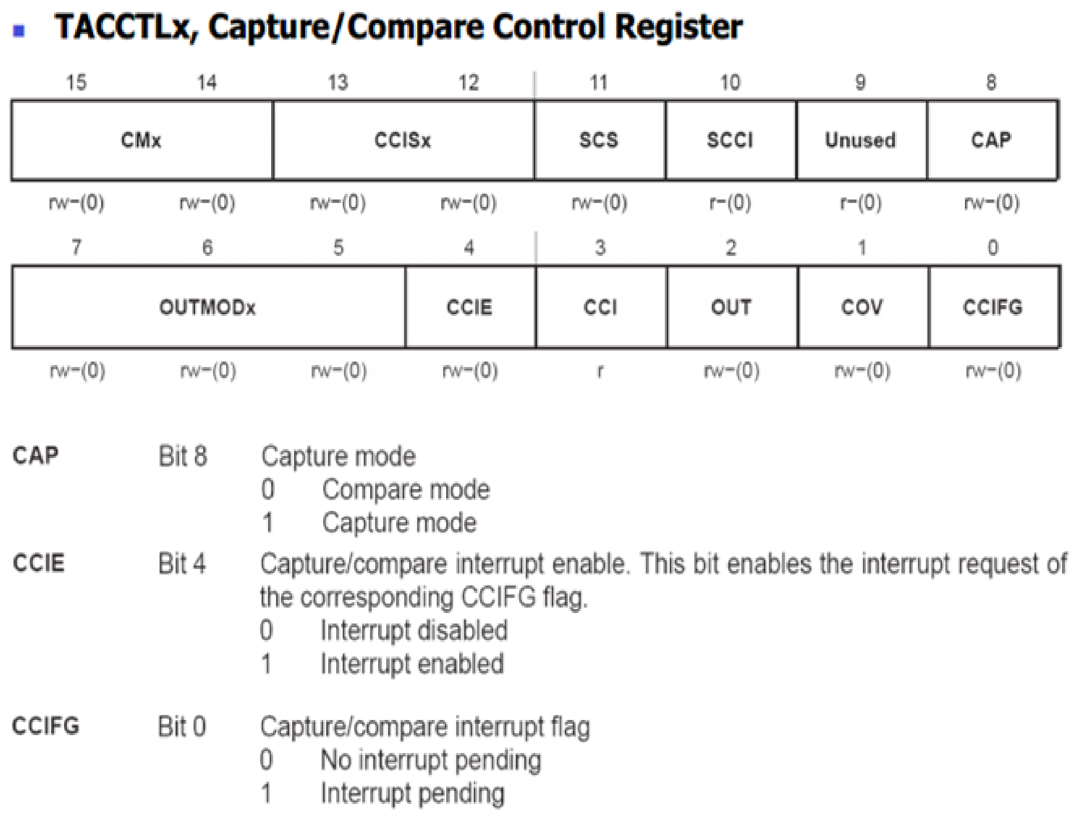
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: the timer is halted 01 Up mode: the timer counts up to TACCR0 10 Continuous mode: the timer counts 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 TACLK 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

- TAR -- The Timer_A Register is the actual counter; reading this register reports the current value of the counter.
- TACCRx -- The Timer_A Capture/Compare Registers, of which there are two in the value line devices (TACCR0 and TACCR1) are where particular values we want to use are stored. In compare mode, we write values here where we want the timer to signal an event. Particularly, TACCR0 is used to store the value to which we want Timer_A to count in up and up/down mode. In capture mode, the processor will record the value of TAR when the MSP430 is signaled to do so.
- TACCTLx -- The Timer_A Capture/Compare Control Registers correspond to the TACCRx registers. These set the behavior of how the CCR's are used.
    - CMx (bits 14 and 15) change what type(s) of signals tell the timer to perform a capture.

- CCISx (bits 12 and 13) select where the input signals are taken.
- SCS and SCCI (bits 11 and 10 respectively) change the synchronicity; the timer normally operates asynchronously to the input signals. (Yeah, I don't entirely understand this yet either)
- CAP (bit 8) changes whether capture mode (1) or compare mode (0) is used.
- OUTMODx (bits 5-7) select various output modes for the CCR's signal when the timer flags a capture or compare event.

```
#define OUTMOD_0            (0*0x20)  /* PWM output mode: 0 - output only */
#define OUTMOD_1            (1*0x20)  /* PWM output mode: 1 - set */
#define OUTMOD_2            (2*0x20)  /* PWM output mode: 2 - PWM toggle/reset */
#define OUTMOD_3            (3*0x20)  /* PWM output mode: 3 - PWM set/reset */
#define OUTMOD_4            (4*0x20)  /* PWM output mode: 4 - toggle */
#define OUTMOD_5            (5*0x20)  /* PWM output mode: 5 - Reset */
#define OUTMOD_6            (6*0x20)  /* PWM output mode: 6 - PWM toggle/set */
#define OUTMOD_7            (7*0x20)  /* PWM output mode: 7 - PWM reset/set */
```

- CCIE and CCIFG (bits 4 and 0) are more interrupts associated with the CCR's.
- CCI and OUT (bits 3 and 2) are the input and output for the CCR.
- COV (bit 1) is the capture overflow; this bit is set to 1 if two captures are signaled before the first capture value is able to be read.



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-(0)	r-(0)	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)

CAP	Bit 8	Capture mode 0 Compare mode 1 Capture mode
CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled
CCIFG	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

- TAIV -- The Timer_A Interrupt Vector Register; since there are multiple types of interrupts that can be flagged by Timer_A, this register holds details on what interrupts have been flagged.
  - The only bits used here are bits 1-3 (TAIVx), which show the type of interrupt that has happened, allowing us to take different actions to resolve the different types of interrupts.

TAIV, Timer_A Interrupt Vector Register

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

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	Reserved	−	
08h	Reserved	−	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	−	
0Eh	Reserved	−	Lowest

TOGGLE RED LED EVERY SECOND:

```
#include <msp430g2553.h>

void main(void) {
        WDTCTL = WDTPW + WDTHOLD;                            // Stop watchdog timer

        P1DIR |= BIT0;                                       // Set P1.0 to output direction
        P1OUT &= ~BIT0;                                      // Set the red LED on

        TA0CCR0 = 12000;                                             // Count limit (16 bit)

        TA0CCTL0 = 0x10;                                             // Enable counter interrupts, bit 4

        TA0CTL = TASSEL_1 + MC_1;                            // Timer A 0 with ACLK @ 12KHz, count UP

        _BIS_SR(LPM0_bits + GIE);                      // LPM0 (low power mode) with interrupts enabled
}

#pragma vector=TIMER0_A0_VECTOR
   __interrupt void Timer0_A0 (void) {               // Timer0 A0 interrupt service routine

        P1OUT ^= BIT0;                                               // Toggle red LED
}
```

Counters in the examples are off by 1. Since the count starts at 0, a count of 12000 would have a limit of 11999. For clarity and arithmetic ease, we'll live with the small error.

Counter

- TA0CCTL0 : Control register for Timer A0 counter 0. Interrupts are enabled by writing a 1 into bit 4 of this register.

- TA0CCR0: Holds the 16-bit count value.

  12000 is used because the ACLK clock operates at approximately 12KHz and we want to toggle the LEDs at approximately at one second intervals. With interrupts enabled, an interrupt is generated when the count reaches 12000.The counter, in UP mode, is automatically reset to 0 when count limit reached.

Interrupts

TA0CCTL0:Enables/disables timer A0 interrupts

TIMER0_A0_VECTOR:Vector for timer A0

One important point is that timers can run independently to the CPU clock, allowing the CPU to be turned off and then automatically turned on when an interrupt occurs.

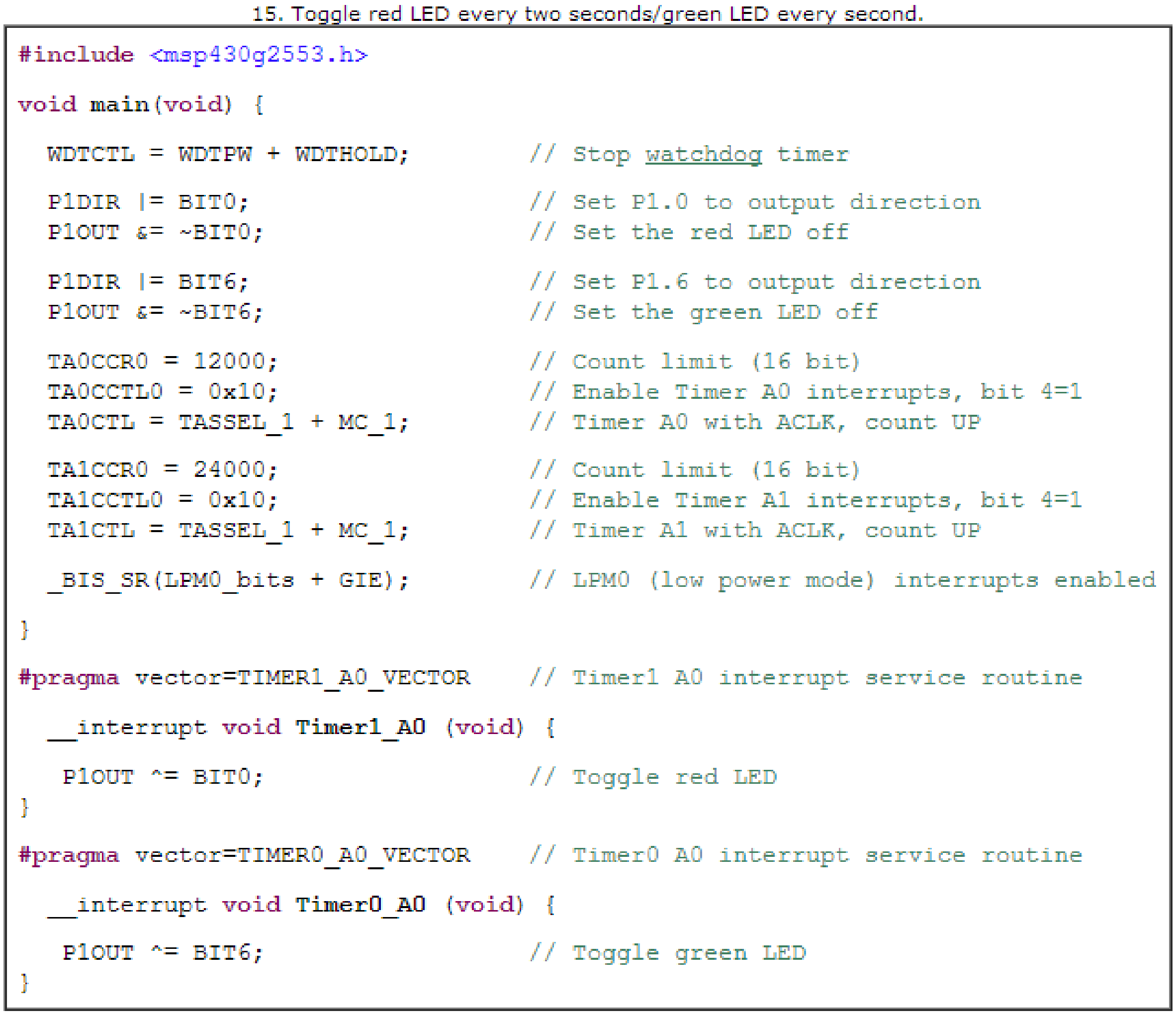
_BIS_SR(LPM0_bits + GIE);   places CPU in low power mode with interrupts enabled.

SR/R2::Status register.

CPUOFF = 1 in bit 4 turns off the CPU until interrupt occurs.

GIE = 1 in bit 3 enables global interrupts (same as EINT instruction).

15. Toggle red LED every two seconds/green LED every second.



```
#include <msp430g2553.h>

void main(void) {

  WDTCTL = WDTPW + WDTHOLD;         // Stop watchdog timer

  P1DIR |= BIT0;                    // Set P1.0 to output direction
  P1OUT &= ~BIT0;                   // Set the red LED off

  P1DIR |= BIT6;                    // Set P1.6 to output direction
  P1OUT &= ~BIT6;                   // Set the green LED off

  TA0CCR0 = 12000;                  // Count limit (16 bit)
  TA0CCTL0 = 0x10;                  // Enable Timer A0 interrupts, bit 4=1
  TA0CTL = TASSEL_1 + MC_1;         // Timer A0 with ACLK, count UP

  TA1CCR0 = 24000;                  // Count limit (16 bit)
  TA1CCTL0 = 0x10;                  // Enable Timer A1 interrupts, bit 4=1
  TA1CTL = TASSEL_1 + MC_1;         // Timer A1 with ACLK, count UP

  _BIS_SR(LPM0_bits + GIE);         // LPM0 (low power mode) interrupts enabled

}

#pragma vector=TIMER1_A0_VECTOR     // Timer1 A0 interrupt service routine

  __interrupt void Timer1_A0 (void) {

   P1OUT ^= BIT0;                   // Toggle red LED
}

#pragma vector=TIMER0_A0_VECTOR     // Timer0 A0 interrupt service routine

  __interrupt void Timer0_A0 (void) {

   P1OUT ^= BIT6;                   // Toggle green LED
}
```

Watchdog Timer

The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. In most examples, the timer is stopped during the first line of code.

The WDT counts down from a specified value and either resets or interrupts when count overflow is reached. A way to use this timer is to periodically service it by resetting its counter so that the system “knows” that everything is all right and there is no reset required. It can also be configured configured as an interval timer to generate interrupts at selected time intervals.

A computer watchdog is a hardware timer used to trigger a system reset if software neglects to regularly service the watchdog . In a watchdog mode, the watchdog timer can be used to protect the system against software failure, such as when a program becomes trapped in an unintended, infinite loop.

Left unattended in watchdog mode, the watchdog counts up and resets the MSP430 when its counter reaches its limit. Your code must therefore keep clearing the watchdog counter before it reaches its limit to prevent a non-maskable, reset interrupt. Be aware that watchdog mode is immediately activated after the MSP430 has been reset and therefore the watchdog must be cleared, stopped, or reconfigured before the default time expires. (Otherwise, your software enters an infinite reset loop.)

Applications needing a periodic "tick" may find the watchdog interval mode ideal for generating a regular interrupt. The disadvantage is the limited selection of interval times - only 4 intervals are available and they are dependent upon the clock assigned to the watchdog.

Assuming the MSP430 system clock has a frequency of 1Mhz and the watchdog timer is clocked by the sub-master clock (SMCLK), the following intervals are available:

Constant	Interval	Clocks/Interval (@1Mhz)	Intervals/Second (@1Mhz)
WDT_MDLY_32	32ms (default)	32000	1000000/32000 = 31.25
WDT_MDLY_8	8ms	8000	1000000/8000 = 125
WDT_MDLY_0_5	0.5ms	500	1000000/500 = 2000
WDT_MDLY_0_064	0.064ms	64	1000000/64 = 15625

Example

WDTCTL = WDT_MDLY_0_5

Sets the WDT to a 0.5ms interval or 2000 intervals/second.

Programming WDT

The watchdog timer is set to interval mode by setting its register to WDTCTL = WDT_MDLY_32.

Defined in the *msp430g2553.h* include file, this sets the watchdog to interrupt every ≈32ms when clocking from a 1Mhz SMCLK.

An interrupt counter is set up so that the device goes to low power after ≈ 8 seconds (32ms * 250 = 8 s). The watchdog interrupt counter is cleared every time P1 interrupts, as well as wakes up if the device is currently in low power mode.

Key things to remember:

Enable WDT interrupt
Enable P1 and global interrupts
Clear P1 interrupt flag before exiting
When setting LPM3, the GIE must also be set to allow interrupts

Watchdog Timer flashing red LED every 1/3 second

```
#include <msp430g2553.h>
unsigned int counter = 0;
void main(void){
    WDTCTL = WDT_MDLY_32;           // Watchdog Timer interval to ≈32ms
    IE1 |= WDTIE;                   // Enable WDT interrupt
    P1DIR |= BIT0;                  // Set red LED P1.0 to output direction
    _BIS_SR(GIE);                   // Enable interrupts
    while(1);
}
#pragma vector=WDT_VECTOR           // Watchdog Timer interrupt service routine
 __interrupt void watchdog_timer(void) {
    if(counter == 10){              // 10 * 32 ms = 320 ms, ≈.3 s
        P1OUT ^= BIT0;              // P1.0 toggle, red LED
        counter = 0;
    }
    else
        counter++;
}
```

PULSE WIDTH MODULATION

- Pulse-width modulation (PWM) is a commonly used technique for controlling power to inertial[ambiguous] electrical devices, made practical by modern electronic power switches.

- The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast pace. The longer the switch is on compared to the off periods, the higher the power supplied to the load is.

- The PWM switching frequency has to be much faster than what would affect the load, which is to say the device that uses the power. Typically switching's have to be done several times a minute in an electric stove, 120 Hz in a lamp dimmer, from few kilohertz (kHz) to tens of kHz for a motor drive and well into the tens or hundreds of kHz in audio amplifiers and computer power supplies.
- Period is the time of each pulse, both the On and Off times.

- Duty cycle describes the proportion of 'on' time to the regular interval or 'period' of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on.

- The main advantage of PWM is that power loss in the switching devices is very low. When a switch is off there is practically no current, and when it is on, there is almost no voltage drop across the switch. Power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle.

PWM signal consists of two parts:

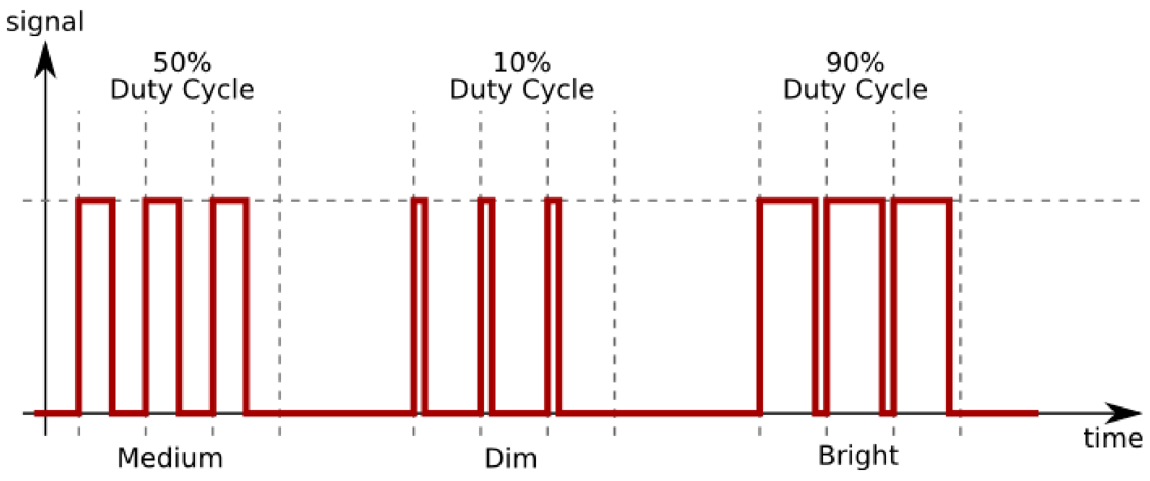
- Period
- Time of each pulse

5Hz signal has periods of 1/5 second = 0.2 second.

Duty cycle: The percentage of period the PWM signal is On or high.

A period of 0.2 second and 10% duty cycle = 0.10 * 0.2 second = 0.02 seconds.

If the signal has a low voltage of 0 and a high voltage of 10 volts, a 50% duty cycle produces an average of 5 volts, a 10% duty cycle produces an average of 1 volt.



PWM on the LaunchPad

PWM can be used to control the brightness of the green LED by varying the duty cycle, a longer duty cycle results in a brighter LED.

LED (green) direction is set and selected for PWM.

```
    P1DIR |= BIT6;              // Green LED as output
    P1SEL |= BIT6;              // Green LED controlled by Pulse width modulation
```

TA0CCR0/TA0CCR1

Timer registers determine when events occur and the clock count limit. Combined, the register values determine the PWM period (TA0CCR0) and duty cycle (TA0CCR1).

```
    TA0CCR0 = 1000;             // PWM period
                          // 12KHz clock gives 12000/1000 = 12Hz = 1/12s period

    TA0CCR1 = 100;              // PWM duty cycle = TA0CCR1/TA0CCR0
                          // time cycle on vs. off, on 10% initially
```

Achieves a 10% duty cycle, where PWM output is 1 for 10% of the time and 0 for 90%

"TA0CCR1 output is reset (0) when the timer counts to the TA0CCR1 value and is set (1) when the timer counts to the TA0CCR0 value" (From User Guide).

Means that when TA0CCR0 count is reached PWM output is 1 and counting starts over, and when TA0CCR1 is reached, PWM output is 0.

TA0CCTL1

Controls PWM output, whether high or low when TA0CCR1 is below the count.

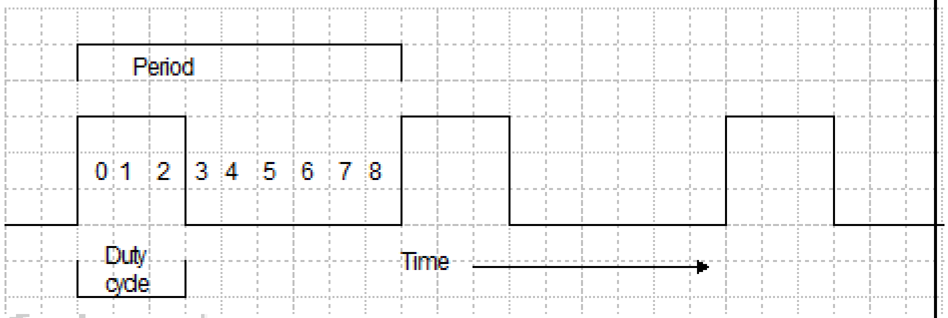
```
   TA0CCTL1 = OUTMOD_7;           // TA0CCR1 reset/set
                                  // high voltage below TA0CCR1 count and low voltage when past
```

TA0CCR0=8 (0-8) period 9 ticks

TA0CCR1=2 (0-2) duty cycle 3/9=33%

33% duty cycle is generated.

Clock	TA0CCR0 Period	TA0CCR1 Duty Cycle	PWM
0	0	0	**1**
1	1	1	**1**
2	2	2	**1**
3	3	0	0
4	4	1	0
5	5	2	0
6	6	0	0
7	7	1	0
8	8	2	0
9	0	0	**1**
10	1	1	**1**
11	2	2	**1**
12	3	0	0
13	4	1	0
14	5	2	0
15	6	0	0
16	7	1	0
17	8	2	0
18	0	0	**1**
19	1	1	**1**
20	2	2	**1**
21	3	0	0



TA0CTL

Timer A control sets SMCLK as clock source and count Up Mode.

```
TA0CCR0 = 8;              // 33% duty cycle
TA0CCR1 = 2;
TA0CTL = TASSEL_1 + MC_1;   // Timer A control set to submain clock TASSEL_1 and count up mode MC_1
```

12KHz clock input to Timer A, the PWM output:

12000/9 = 1333 periods per second

1s/1333 = 0.00075s period duration.

```
_BIS_SR(LPM0_bits);   // Enter Low power mode 0
```

Placing system in LPM0 keeps the SMCLK and timer A operating to control the PWM while shutting down the CPU.

The default DCO (digital controlled oscillator) which is the source of the MCLK and in this case the ACLK at approximately 12KHz, making the PWM period about 1 second and .10 second duty cycle or 10%.

To Make a Green LED Glow at 10% brightness

```
#include <msp430g2553.h>

void main(void){

    WDTCTL = WDTPW + WDTHOLD;            // Stop watchdog timer

    P1DIR |= BIT6;                       // Green LED

    P1SEL |= BIT6;                       // Green LED selected for Pulse Width Modulation

    TA0CCR0 = 12000;                     // PWM period, 12000 ACLK ticks or 1/second

    TA0CCR1 = 1200;                      // PWM duty cycle, time cycle on vs. off, on 10% initially

    TA0CCTL1 = OUTMOD_7;                 // TA0CCR1 reset/set -- high voltage below TA0CCR1 count
                                         // and low voltage when past

    TA0CTL = TASSEL_1 + MC_1;            // Timer A control set to submain clock TASSEL_1 ACLK
                                         // and count up to TA0CCR0 mode MC_1

    _BIS_SR(LPM0_bits);                  // Enter Low power mode 0
}
```

Which pins do PWM?

To know which pins can do pulse width modulation,we can modify the LED program to run PWM on all pins. Then hook up a multimeter to see which pins changed in time with the green LED.

It appears that on the msp430g2553, the only pins that do PWM are pin 1.2 and pin 1.6.

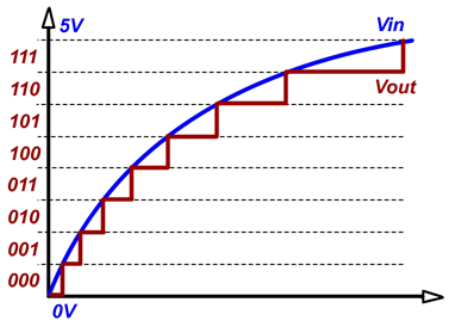
ANALOG TO DIGITAL CONVERSION:

ADC - Analog to Digital Conversion

Analog to digital conversion (ADC) senses the analog, continuous, world and converts it to a digital value for use on a computer.

For Eg:A 3-bit ADC produces 8 values.

Analog Volts	Digital Value
4.375 to 5	111
3.75 to 4.375	110
3.125 to 3.75	101
2.5 to 3.125	100
1.875 to 2.5	**011**
1.25 to 1.875	010
.625 to 1.25	001
0 to .625	000



Example: 2.0 volts = 011

(FOR ADC REPRESENTATION OF 5VOLTS)

ADC is often performed by successive approximation. The essentials of the process are:

1. A digital counter is driven by a clock, so each tick updates the counter.
2. The counter output determines an output voltage.
3. The analog voltage input is compared with the output voltage.
4. When the input and output voltage match, the counter holds the digital value.

Because of the time it takes time for the counter to "find" the proper value, an ADC is generally much slower than the CPU.

To prevent receiving stale or erroneous results, after starting an ADC conversion the CPU can poll the ADC to determine when the conversion is complete.

The LaunchPad MSP430G2553 chip contains a temperature sensor, a thermistor, connected to an ADC; the LaunchPad can take its own temperature.

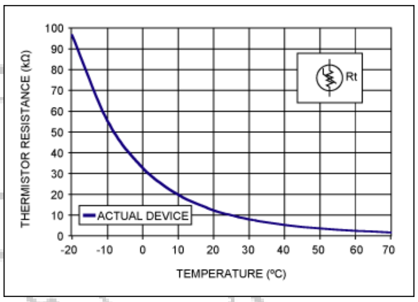
An ADC compares a variable input voltage to a reference voltage, returning the proportion between the two.

The 10-bit ADC on the G2553, returns a number between 0 and 1023, or 1024 = 2^{10} values.

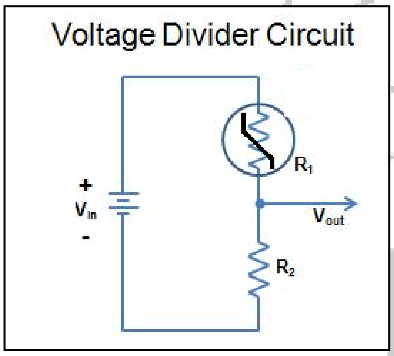
The ADC, with an analog input of 1 volt and the LaunchPad's reference voltage of 1.5 volts, returns 682 because its the same proportion with 1024:

$1/1.5 = .666 = 682/1024$

NTC thermistors change resistance with temperature, higher temperature results in lower resistance.



Digital temperature sensing often places a thermistor in a voltage dividing circuit such as the one below.



$$V_{out} = \frac{R_2}{R_2 + R_1} * V_{in}$$

Note that as temperature increases, R_1 decreases and V_{out} increases.
R_1 - thermistor, 10K Ohms at 25 Celsius is typical
R_2 - a fixed resistor, 10K Ohms
V_{in} - the fixed reference voltage, 1.5v
At 25 Celsius and V_{in} of 1.5v reference, a typical thermistor results:

V_{out} = 10,000/(10,000+10,000) * 1.5v = 1/2 * 1.5
= 0.75v

The ADC, with an analog input of 0.75 volt and the LaunchPad's reference voltage of 1.5 volts, returns 512 because proportional.

$.75/1.5 = 1/2 = 512/1024$

20.3 ADC10 Registers

The ADC10 registers are listed in Table 20−3.

Table 20−3. ADC10 Registers

Register	Short Form	Register Type	Address	Initial State
ADC10 input enable register 0	ADC10AE0	Read/write	04Ah	Reset with POR
ADC10 input enable register 1	ADC10AE1	Read/write	04Bh	Reset with POR
ADC10 control register 0	ADC10CTL0	Read/write	01B0h	Reset with POR
ADC10 control register 1	ADC10CTL1	Read/write	01B2h	Reset with POR
ADC10 memory	ADC10MEM	Read	01B4h	Unchanged
ADC10 data transfer control register 0	ADC10DTC0	Read/write	048h	Reset with POR
ADC10 data transfer control register 1	ADC10DTC1	Read/write	049h	Reset with POR
ADC10 data transfer start address	ADC10SA	Read/write	01BCh	0200h with POR

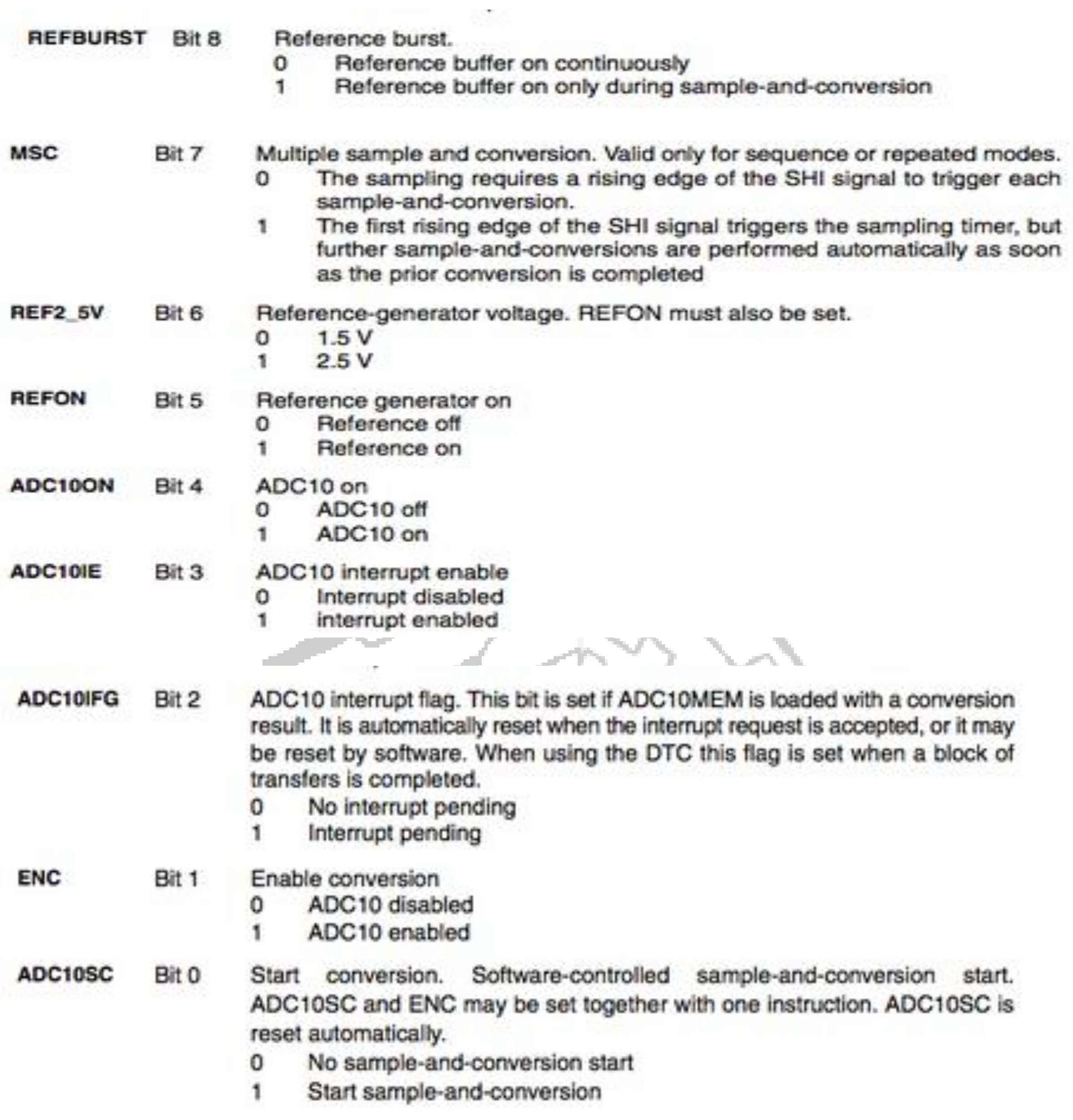
ADC10CTL0, ADC10 Control Register 0

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)

7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)	rw−(0)

▨ Modifiable only when ENC = 0

SREFx	Bits 15-13	Select reference 000 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$ 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$ 010 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{SS}$ 011 $V_{R+} =$ Buffered Ve_{REF+} and $V_{R-} = V_{SS}$ 100 $V_{R+} = V_{CC}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$ 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$ 110 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$ 111 $V_{R+} =$ Buffered Ve_{REF+} and $V_{R-} = V_{REF-}/ Ve_{REF-}$
ADC10 SHTx	Bits 12-11	ADC10 sample-and-hold time 00 4 x ADC10CLKs 01 8 x ADC10CLKs 10 16 x ADC10CLKs 11 64 x ADC10CLKs
ADC10SR	Bit 10	ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer. 0 Reference buffer supports up to ~200 ksps 1 Reference buffer supports up to ~50 ksps
REFOUT	Bit 9	Reference output 0 Reference output off 1 Reference output on



REFBURST	Bit 8	Reference burst. 0 Reference buffer on continuously 1 Reference buffer on only during sample-and-conversion
MSC	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes. 0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion. 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed
REF2_5V	Bit 6	Reference-generator voltage. REFON must also be set. 0 1.5 V 1 2.5 V
REFON	Bit 5	Reference generator on 0 Reference off 1 Reference on
ADC10ON	Bit 4	ADC10 on 0 ADC10 off 1 ADC10 on
ADC10IE	Bit 3	ADC10 interrupt enable 0 Interrupt disabled 1 interrupt enabled
ADC10IFG	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed. 0 No interrupt pending 1 Interrupt pending
ENC	Bit 1	Enable conversion 0 ADC10 disabled 1 ADC10 enabled
ADC10SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically. 0 No sample-and-conversion start 1 Start sample-and-conversion

Example settings of ADC10CTL0

```
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON;
```

SREF_1	Sets reference voltage
ADC10SHT_3	Sets sample-and-hold times to 64x ADC10CLK
REFON	Sets reference ON.
ADC10ON	ADC10 ON.

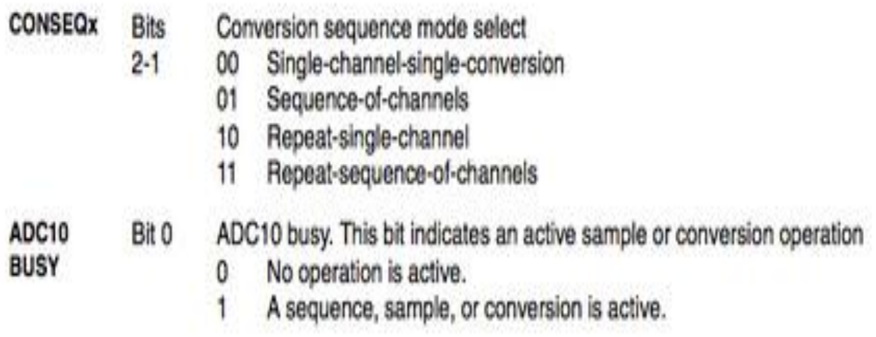
ADC10CTL1, ADC10 Control Register 1

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10 BUSY
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–0

▨ Modifiable only when ENC = 0

INCHx	Bits 15-12	Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions.
		0000 A0
		0001 A1
		0010 A2
		0011 A3
		0100 A4
		0101 A5
		0110 A6
		0111 A7
		1000 Ve_{REF+}
		1001 V_{REF-}/Ve_{REF-}
		1010 Temperature sensor
		1011 $(V_{CC} - V_{SS}) / 2$
		1100 $(V_{CC} - V_{SS}) / 2$, A12 on MSP430x22xx devices
		1101 $(V_{CC} - V_{SS}) / 2$, A13 on MSP430x22xx devices
		1110 $(V_{CC} - V_{SS}) / 2$, A14 on MSP430x22xx devices
		1111 $(V_{CC} - V_{SS}) / 2$, A15 on MSP430x22xx devices
SHSx	Bits 11-10	Sample-and-hold source select
		00 ADC10SC bit
		01 Timer_A.OUT1
		10 Timer_A.OUT0
		11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430x20x2 devices)
ADC10DF	Bit 9	ADC10 data format
		0 Straight binary
		1 2s complement
ISSH	Bit 8	Invert signal sample-and-hold
		0 The sample-input signal is not inverted.
		1 The sample-input signal is inverted.
ADC10DIVx	Bits 7-5	ADC10 clock divider
		000 /1
		001 /2
		010 /3
		011 /4
		100 /5
		101 /6
		110 /7
		111 /8
ADC10 SSELx	Bits 4-3	ADC10 clock source select
		00 ADC10OSC
		01 ACLK
		10 MCLK
		11 SMCLK



CONSEQx	Bits 2-1	Conversion sequence mode select 00 Single-channel-single-conversion 01 Sequence-of-channels 10 Repeat-single-channel 11 Repeat-sequence-of-channels
ADC10 BUSY	Bit 0	ADC10 busy. This bit indicates an active sample or conversion operation 0 No operation is active. 1 A sequence, sample, or conversion is active.

SAMPLE CODE FOR ADC-USING INBUILT TEMPERATURE SENSOR IN MSP430

```
#include <msp430g2553.h>
long sample;
long DegreeF;
void main(void) {
  WDTCTL = WDTPW + WDTHOLD;                    // Stop WDT

  ADC10CTL1 = INCH_10 + ADC10DIV_3;             // Temp Sensor ADC10CLK/4

  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON;      // Ref voltage/sample & hold time/
                                      // reference generator ON/ADC10 ON
  while(1) {
   ADC10CTL0 |= ENC + ADC10SC;                  // Sampling and conversion start

   while ( ADC10CTL1 & ADC10BUSY );              // Wait for ADC to complete

   sample = ADC10MEM;                          // Read ADC sample

   DegreeF = ((sample - 630) * 761) / 1024;

   __no_operation();                        // SET BREAKPOINT HERE
 }
}
```

TUTORIAL BY:
RAMITHA SUNDAR
EEE
107112069