CPE/EE 421 Microcomputers

Instructor: Dr Aleksandar Milenkovic
Lecture Note
S15

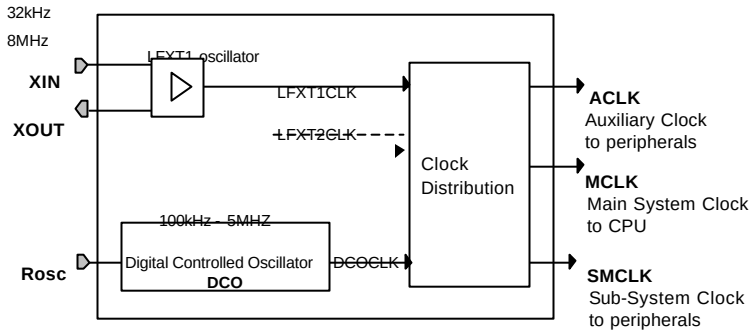
MSP430 Documentation

- MSP430 home page (TI)
  - www.ti.com/msp430
- User's manual
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/slau049c.pdf
- Datasheet
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/slas272c.pdf
- TI Workshop document
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/430_2002_atc_workshop.pdf
- IAR Workbench Tutorial
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/TUTOR.pdf

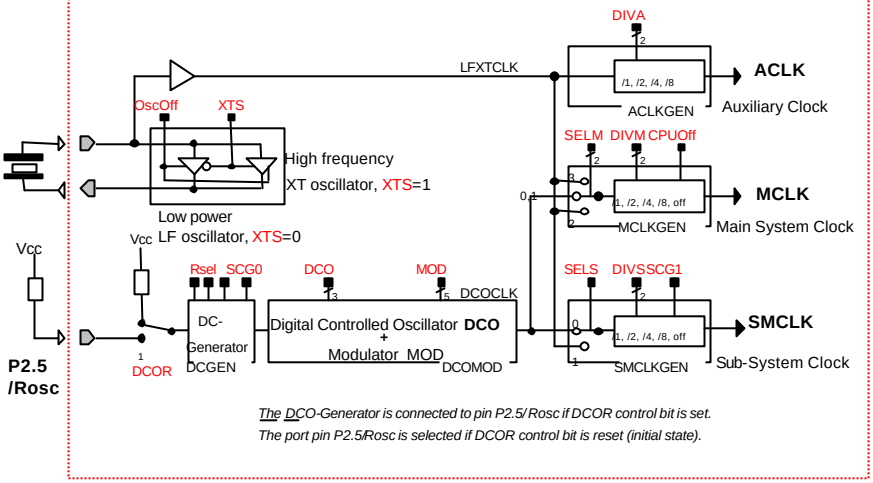
The MSP430
Clock Module

Basic Clock Systems

- DCOCLK Generated on-chip with 6µs start-up
- 32KHz Watch Crystal - or - High Speed Crystal / Resonator to 8MHz
  - (our system is 4MHz/8MHz high Speed Crystal)
- Flexible clock distribution tree for CPU and peripherals
- Programmable open-loop DCO Clock with internal and external current source



Basic Clock Systems-detail



Basic operation

- After POC (Power Up Clear) MCLK and SCLK are sourced by DCOCLK (approx. 800KHz) and ACLK is sourced by LFXT1 in LF mode
- Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the basic clock module. The DCOCTL, BCSCTL1, and BCSCTL2 registers configure the basic clock module
- The basic clock can be configured or reconfigured by software at any time during program execution

Low-power operation: An example

- ACLK can be configured to oscillate with a low-power 32,786-Hz watch crystal
- MCLK can be configured to operate from the on-chip DCO that can be only activated when requested by interrupt-driven events.
- SMCLK can be configured to operate from either the watch crystal or the DCO, depending on peripheral requirements.

Basic Clock Systems-control registers

- Direct SW Control
- DCOCLK can be Set - Stabilized
- Stable DCOCLK over Temp/Vcc.

BCSCTL2

058h

SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

BCSCTL1

057h

XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

Selection of DCO nominal frequency

DCOCTL

056h

DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

Which of eight discrete DCO frequencies is selected

Define how often frequency f_{DCO+1} within the period of 32 DCOCLK cycles is used. Remaining clock cycles (32-MOD) the frequency f_{DCO} is mixed

RSEL.x Select DCO nominal frequency

DCO.x and MOD.x set exact DCOCLK

… select other clock tree options

Basic Clock Systems-control registers(detail)

- **Basic Clock Module Control Registers**

The Basic Clock Module is configured using control registers DCOCTL, BCSCTL1, and BCSCTL2, and four bits from the CPU status register: SCG1, SCG0, OscOff, and CPUOFF.

User software can modify these control registers from their default condition at any time. The Basic Clock Module control registers are located in the byte-wide peripheral map and should be accessed with byte (.B) instructions.

Register State	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h
Basic clock system control 1	BCSCTL1	Read/write	057h	084h
Basic clock system control 2	BCSCTL2	Read/write	058h	reset

Basic Clock Systems-control registers(detail)

- **Digitally-Controlled Oscillator (DCO) Clock-Frequency Control**

DCOCTL is loaded with a value of 060h with a valid PUC condition.

	7							0
DCOCTL	DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
056H	0	1	1	0	0	0	0	0

MOD.0 .. MOD.4: The MOD constant defines how often the discrete frequency f_{DCO+1} is used within a period of 32 DCOCLK cycles.

During the remaining clock cycles (32–MOD) the discrete frequency f_{DCO} is used. When the DCO constant is set to seven, no modulation is possible since the highest feasible frequency has then been selected.

DCO.0 .. DCO.2: The DCO constant defines which one of the eight discrete frequencies is selected. The frequency is defined by the current injected into the dc generator.

Basic Clock Systems-control registers(detail)

➢ **Oscillator and Clock Control Register**

BCSCTL1 is affected by a valid PUC or POR condition.

	7							0
BCSCTL1	XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.0	Rsel.1	Rsel.2
057h	1	0	0	0	0	1	0	0

Bit0 to Bit2: The internal resistor is selected in eight different steps.

Rsel.0 to Rsel.2 The value of the resistor defines the nominal frequency. The lowest nominal frequency is selected by setting Rsel=0.

Bit3, XT5V: XT5V should always be reset.

Bit4 to Bit5: The selected source for ACLK is divided by:

DIVA = 0: 1
DIVA = 1: 2
DIVA = 2: 4
DIVA = 3: 8

Basic Clock Systems-control registers(detail)

Bit6, XTS: The LFXT1 oscillator operates with a low-frequency or with a high-frequency crystal:

XTS = 0: The low-frequency oscillator is selected.
XTS = 1: The high-frequency oscillator is selected.

The oscillator selection must meet the external crystal's operating condition.

Bit7, XT2Off: The XT2 oscillator is switched on or off:

XT2Off = 0: the oscillator is on
XT2Off = 1: the oscillator is off if it is not used for MCLK or SMCLK.

Basic Clock Systems-control registers(detail)

BCSCTL2 is affected by a valid PUC or POR condition.

	7							0
BCSCTL2	~~SELM.1~~	~~SELM.0~~	~~DIVM.1~~	~~DIVM.0~~	~~SELS~~	~~DIVS.1~~	~~DIVS.0~~	~~DCOR~~

058h

Bit0, DCOR: The DCOR bit selects the resistor for injecting current into the dc generator. Based on this current, the oscillator operates if activated.

DCOR = 0: Internal resistor on, the oscillator can operate. The fail-safe mode is on.

DCOR = 1: Internal resistor off, the current must be injected externally if the DCO output drives any clock using the DCOCLK.

Bit1, Bit2: The selected source for SMCLK is divided by:

DIVS.1 .. DIVS.0 DIVS = 0:1

DIVS = 1: 2

DIVS = 2: 4

DIVS = 3: 8

Basic Clock Systems-control registers(detail)

Bit3, SELS: Selects the source for generating SMCLK:

SELS = 0: Use the DCOCLK

SELS = 1: Use the XT2CLK signal (in three-oscillator systems)

or

LFXT1CLK signal (in two-oscillator systems)

Bit4, Bit5: The selected source for MCLK is divided by DIVM.0 .. DIVM.1

DIVM = 0: 1

DIVM = 1: 2

DIVM = 2: 4

DIVM = 3: 8

Bit6, Bit7: Selects the source for generating MCLK:

SELM.0 .. SELM.1

SELM = 0: Use the DCOCLK

SELM = 1: Use the DCOCLK

SELM = 2: Use the XT2CLK (x13x and x14x devices)

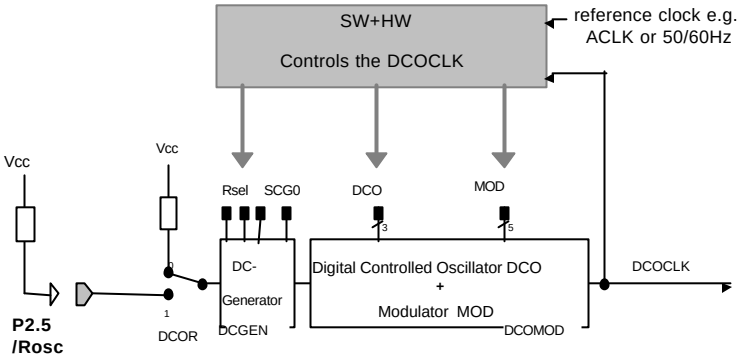
or

Use the LFXT1CLK (x11x(1) devices)

SELM = 3: Use the LFXT1CLK

Basic Clock Systems-software Fll idea

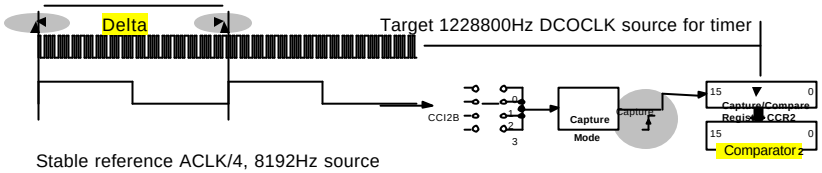
- **Basic Clock DCO is an open loop - close with SW+HW**
  - A reference frequency e.g. ACLK or 50/60Hz can be used to measure DCOCLK's
  - **Initialization or Periodic** software set and stabilizes DCOCLK over reference clock
  - DCOCLK is programmable 100kHz - 5Mhz and stable over voltage and temperature



Basic Clock Systems-software Fll implementation

- **Example: Set DCOCLK= 1228800, ACLK= 32768**
  - ACLK/4 captured on CCI2B, DCOCLK is clock source for Timer_A
  - Comparator2 HW captures SMCLK (1228800Hz) in one ACLK/4 (8192Hz) period
  - Target Delta = 1228800/8192= 150

```
CCI2BInt …                      ; Compute Delta
        cmp     #150,Delta      ; Delta= 1228800/8192
        jlo     IncDCO          ; JMP to IncDCO
DecDCO  dec     &DCOCTL         ; Decrease DCOCLK
        reti
IncDCO  inc     &DCOCTL         ; Increase DCOCLK
        reti
```



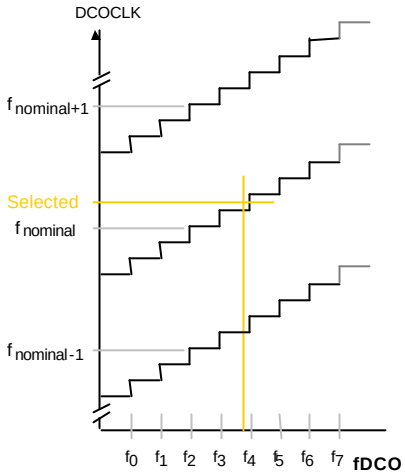
Stable reference ACLK/4, 8192Hz source

Basic Clock Systems-DCO TAPS

- **DCOCLK frequency control**
  - nominal - injected current into DC generator
    1) internal resistors Rsel2, Rsel1 and Rsel0
    2) an external resistor at Rosc (P2.5/11x)
  - Control bits DCO0 to DCO2 set fDCO tap
  - Modulation bits MOD0 to MOD4 allow mixing of fDCO and fDCO+1 for precise frequency generation

Example	Frequency	Cycle time
Selected:	1000kHz	1000 nsec
f3:	943kHz	1060 nsec
f4:	1042kHz	960 nsec

MOD=19

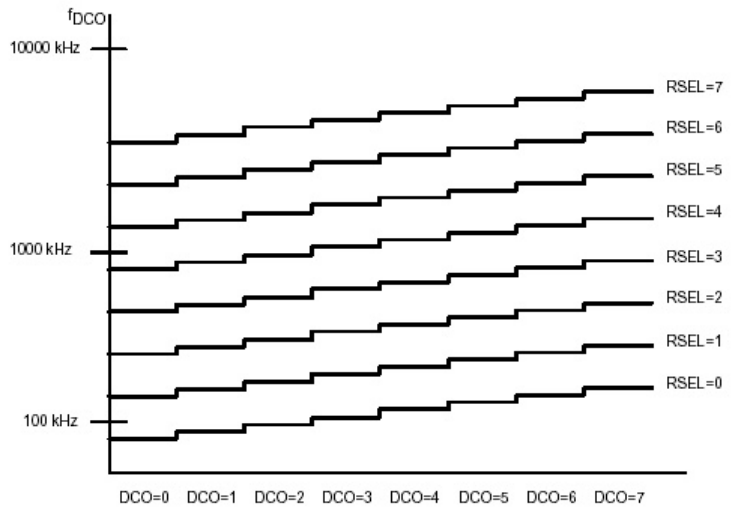


Cycle_time = ((32-MOD)*t1+MOD*t2)/32 = 1000.625 ns, selected frequency ≈ 1 MHz.

F149 default DCO clock setting

PARAMETER	TEST CONDITIONS		MIN	NOM	MAX	UNIT
$f_{(DCO03)}$	R_{sel} = 0, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.08	0.12	0.15	MHz
		V_{CC} = 3 V	0.08	0.13	0.16	
$f_{(DCO13)}$	R_{sel} = 1, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.14	0.19	0.23	MHz
		V_{CC} = 3 V	0.14	0.18	0.22	
$f_{(DCO23)}$	R_{sel} = 2, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.22	0.30	0.36	MHz
		V_{CC} = 3 V	0.22	0.28	0.34	
$f_{(DCO33)}$	R_{sel} = 3, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.37	0.49	0.59	MHz
		V_{CC} = 3 V	0.37	0.47	0.56	
$f_{(DCO43)}$	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.61	0.77	0.93	MHz
		V_{CC} = 3 V	0.61	0.75	0.90	
$f_{(DCO53)}$	R_{sel} = 5, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1	1.2	1.5	MHz
		V_{CC} = 3 V	1	1.3	1.5	
$f_{(DCO63)}$	R_{sel} = 6, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1.6	1.9	2.2	MHz
		V_{CC} = 3 V	1.69	2.0	2.29	
$f_{(DCO73)}$	R_{sel} = 7, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	2.4	2.9	3.4	MHz
		V_{CC} = 3 V	2.7	3.2	3.65	
$f_{(DCO47)}$	R_{sel} = 4, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V/3 V	f_{DCO40} × 1.7	f_{DCO40} × 2.1	f_{DCO40} × 2.5	MHz
$f_{(DCO77)}$	R_{sel} = 7, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	4	4.5	4.9	MHz
		V_{CC} = 3 V	4.4	4.9	5.4	
$S_{(Rsel)}$	$S_R = f_{Rsel+1} / f_{Rsel}$	V_{CC} = 2.2 V/3 V	1.35	1.65	2	
$S_{(DCO)}$	$S_{DCO} = f_{DCO+1} / f_{DCO}$	V_{CC} = 2.2 V/3 V	1.07	1.12	1.16	
D_t	Temperature drift, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 30)	V_{CC} = 2.2 V	–0.31	–0.36	–0.40	%/°C
		V_{CC} = 3 V	–0.33	–0.38	–0.43	
D_V	Drift with V_{CC} variation, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 30)	V_{CC} = 2.2 V/3 V	0	5	10	%/V

Range (RSELx) and Steps (DCOx)



Basic Clock Systems-Examples

- **How to select the Crystal Clock**

```
void selectclock(void)
{
  IFG2=0;          /* reset interrupt flag register 1 */
  IFG1=0;          /* reset interrupt flag register 2 */
  BCSCTL1|=XTS;    /*attach HF crystal (4MHz) to XIN/XOUT  */
  do {
                   /*wait in loop until crystal is stable*/
     IFG1&=~OFIFG;
  }while(OFIFG&IFG1);

  Delay();
  IFG1&=~OFIFG;          /*Reset osc. fault flag again*/
}
```

- **How to select a clock for MCLK**

```
BCSCTL2=SELM0+SELM1;       /*Then set MCLK same as LFXT1CLK*/
TACTL=TASSEL0+TACLR+ID1;   /*USE ACLK/4 AS TIMER_A INPUT CLOCK
                             (1MHz) */
```

Basic Clock Systems-Examples

- **Adjusting the Basic Clock**

The control registers of the Basic Clock are under full software control. If clock requirements other than those of the default from PUC are necessary, the Basic Clock can be configured or reconfigured by software at any time during program execution.

- ACLKGEN from LFXT1 crystal, resonator, or external-clock source and divided by 1, 2, 4, or 8. If no LFXTCLK clock signal is needed in the application, the OscOff bit should be set in the status register.
- SCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. The SCG1 bit in the status register enables or disables SMCLK.
- MCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. When set, the CPUOff bit in the status register enables or disables MCLK.
- DCOCLK frequency is adjusted using the RSEL, DCO, and MOD bits. The DCOCLK clock source is stopped when not used, and the dc generator can be disabled by the SCG0 bit in the status register (when set).
- The XT2 oscillator sources XT2CLK (x13x and x14x only) by clearing the XT2Off bit.

Interrupt Service Routines

- **Interrupt Service Routine declaration**

```
// Func. declaration
Interrupt[int_vector] void myISR (Void);

Interrupt[int_vector] void myISR (Void)
{
//  ISR code
}
```

- **EXAMPLE**

```
Interrupt[TIMERA0_VECTOR] void myISR (Void);

Interrupt[TIMERA0_VECTOR] void myISR (Void)
{
//  ISR code
}
```

Interrupt Service Routines

- **MSP430 interrupt vectors (int_vector)**

```
/************************************************************
* Interrupt Vectors (offset from 0xFFE0)
************************************************************/

#define PORT2_VECTOR        1 * 2  /* 0xFFE2 Port 2 */
#define UART1TX_VECTOR      2 * 2  /* 0xFFE4 UART 1 Transmit */
#define UART1RX_VECTOR      3 * 2  /* 0xFFE6 UART 1 Receive */
#define PORT1_VECTOR        4 * 2  /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR      5 * 2  /* 0xFFEA Timer A CC1-2, TA */
#define TIMERA0_VECTOR      6 * 2  /* 0xFFEC Timer A CC0 */
#define ADC_VECTOR          7 * 2  /* 0xFFEE ADC */
#define UART0TX_VECTOR      8 * 2  /* 0xFFF0 UART 0 Transmit */
#define UART0RX_VECTOR      9 * 2  /* 0xFFF2 UART 0 Receive */
#define WDT_VECTOR         10 * 2 /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR 11 * 2 /* 0xFFF6 Comparator A */
#define TIMERB1_VECTOR     12 * 2 /* 0xFFF8 Timer B 1-7 */
#define TIMERB0_VECTOR     13 * 2 /* 0xFFFA Timer B 0 */
#define NMI_VECTOR         14 * 2 /* 0xFFFC Non-maskable */
#define RESET_VECTOR       15 * 2 /* 0xFFFE Reset [Highest Pr.] */
```


Watchdog Timer-General

General

The primary function of the watchdog-timer module (WDT) is to perform a controlled-system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval.

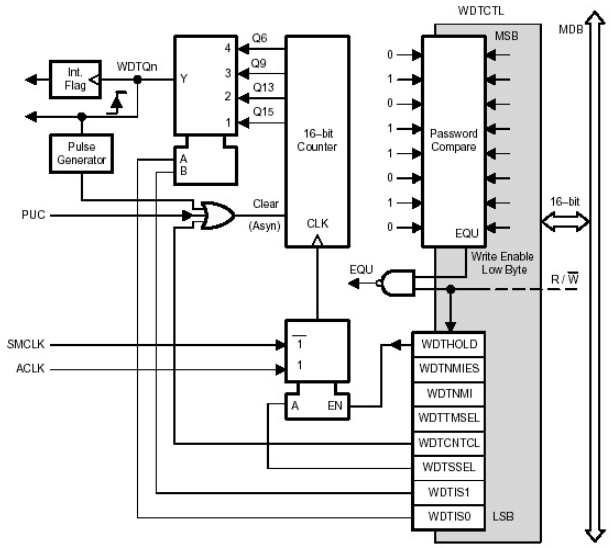
Features of the Watchdog Timer include:

- Eight software-selectable time intervals
- Two operating modes: as watchdog or interval timer
- Expiration of the time interval in watchdog mode, which generates a system reset; or in timer mode, which generates an interrupt request
- Safeguards which ensure that writing to the WDT control register is only possible using a password
- Support of ultralow-power using the hold mode

Watchdog/Timer two functions:

- SW Watchdog Mode
- Interval Timer Mode

Watchdog Timer-Diagram

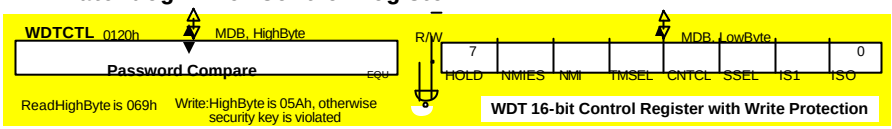


Watchdog Timer-Registers

- **Watchdog Timer Counter**

  The watchdog-timer counter (WDTCNT) is a 16-bit up-counter that is not directly accessible by software. The WDTCNT is controlled through the watchdog-timer control register (WDTCTL), which is a 16-bit read/write register located at the low byte of word address 0120h. Any read or write access must be done using word instructions with no suffix or .w suffix. In both operating modes (watchdog or timer), it is only possible to write to WDTCTL using the correct password.

- **Watchdog Timer Control Register**



Bits 0, 1: Bits IS0 and IS1 select one of four taps from the WDTCNT, as described in following table. Assuming f crystal = 32,768 Hz and f System = 1 MHz, the following intervals are possible:

Watchdog Timer-Registers

SSEL	IS1	IS0	Interval [ms]		
0	1	1	0.064	t SMCLK × 2^6	Table: WDTCNT Taps
0	1	0	0.5	t SMCLK × 2^9	
1	1	1	1.9	t ACLK × 2^6	
0	0	1	8	t SMCLK × 2^{13}	
1	1	0	16.0	t ACLK × 2^9	
0 (reset)	**0**	**0**	**32**	t SMCLK × 2^{15}	<– Value after PUC
1	0	1	250	t ACLK × 2^{13}	
1	0	0	1000	t ACLK × 2^{15}	

Bit 2: The SSEL bit selects the clock source for WDTCNT.

SSEL = 0: WDTCNT is clocked by SMCLK .

SSEL = 1: WDTCNT is clocked by ACLK.

Bit 3: Counter clear bit. In both operating modes, writing a 1 to this bit restarts the WDTCNT at 00000h. The value read is not defined.

Bit 4: The TMSEL bit selects the operating mode: watchdog or timer.

TMSEL = 0: Watchdog mode

TMSEL = 1: Interval-timer mode

Bit 5: The NMI bit selects the function of the RST/NMI input pin. It is

Watchdog Timer-Registers

NMI = 0: The RST/NMI input works as reset input.

As long as the RST/NMI pin is held low, the internal signal is active (level sensitive).

NMI = 1: The RST/NMI input works as an edge-sensitive non-maskable interrupt input.

Bit 6: If the NMI function is selected, this bit selects the activating edge of the RST/NMI input. It is cleared by the PUC signal.

NMIES = 0: A rising edge triggers an NMI interrupt.

NMIES = 1: A falling edge triggers an NMI interrupt.

CAUTION: Changing the NMIES bit with software can generate an NMI interrupt.

Bit 7: This bit stops the operation of the watchdog counter. The clock multiplexer is disabled and the counter stops incrementing. It holds the last value until the hold bit is reset and the operation continues. It is cleared by the PUC signal.

HOLD = 0: The WDT is fully active.

HOLD = 1: The clock multiplexer and counter are stopped.

Watchdog Timer-Interrupt Function

- **The Watchdog Timer (WDT) uses two bits in the SFRs for interrupt control.**

The WDT interrupt flag (WDTIFG) (located in IFG1.0, initial state is reset)

The WDT interrupt enable (WDTIE) (located in IE1.0, initial state is reset)

- When using the watchdog mode, the WDTIFG flag is used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, then the Watchdog Timer initiated the reset condition (either by timing out or by a security key violation). If the flag is cleared, then the PUC was caused by a different source. See chapter 3 for more details on the PUC and POR signals.
- When using the Watchdog Timer in interval-timer mode, the WDTIFG flag is set after the selected time interval and a watchdog interval-timer interrupt is requested. The interrupt vector address in interval-timer mode is different from that in watchdog mode. In interval-timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced.
- The WDTIE bit is used to enable or disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. Also, the GIE bit enables or disables the interrupt from the Watchdog Timer when it is being used in interval-timer mode.

Watchdog Timer-Timer Mode

- Setting WDTCTL register bit TMSEL to 1 selects the timer mode. This mode provides periodic interrupts at the selected time interval. A time interval can also be initiated by writing a 1 to bit CNTCL in the WDTCTL register.

- When the WDT is configured to operate in timer mode, the WDTIFG flag is set after the selected time interval, and it requests a standard interrupt service. The WDT interrupt flag is a single-source interrupt flag and is automatically reset when it is serviced. The enable bit remains unchanged. In interval-timer mode, the WDT interrupt-enable bit and the GIE bit must be set to allow the WDT to request an interrupt. The interrupt vector address in timer mode is different from that in watchdog mode.

Watchdog Timer-Examples

- **How to select timer mode**

```
/* WDT is clocked by fACLK (assumed 32Khz) */
WDTCL=WDT_ADLY_250; // WDT 250MS/4 INTERVAL TIMER
IE1 |=WDTIE;        // ENABLE WDT INTERRUPT
```

- **How to stop watchdog timer**

```
WDTCTL=WDTPW + WDTHOLD ;  // stop watchdog timer
```

- **Assembly programming**

```
WDT_key     .equ    05A00h                ; Key to access WDT
WDTStop     mov   #(WDT_Key+80h),&WDTCTL ; Hold Watchdog
WDT250      mov   #(WDT_Key+1Dh),&WDTCTL ; WDT, 250ms Interval
```


MSP430x1xx Microcontrollers
Low Power Modes

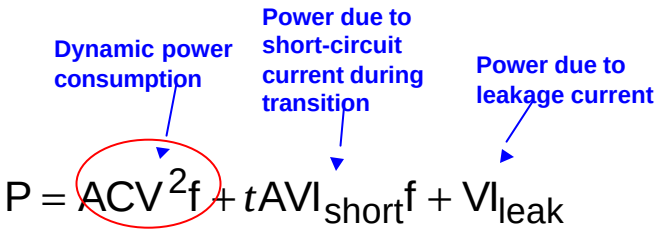
CPE/EE 421/521 Microcomputers

Power as a Design Constraint

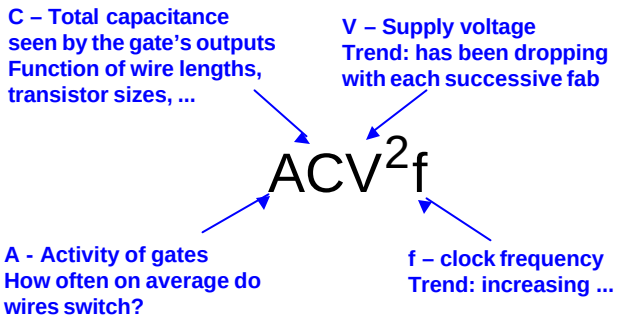
Power becomes a first class architectural design constraint

- Why worry about power?
  - Battery life in portable and mobile platforms
  - Power consumption in desktops, server farms
    - Cooling costs, packaging costs, reliability, timing
    - Power density: 30 W/cm2 in Alpha 21364 (3x of typical hot plate)
  - Environment?
    - IT consumes 10% of energy in the US

Where does power go in CMOS?



Dynamic Power Consumption

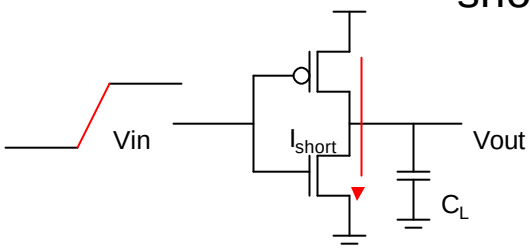


Reducing Dynamic Power

1) **Reducing V has quadratic effect; Limits?**
2) **Lower C - shrink structures, shorten wires**
3) **Reduce switching activity - Turn off unused parts or use design techniques to minimize number of transitions**

Short-circuit Power Consumption

$$tAVI_{short}f$$

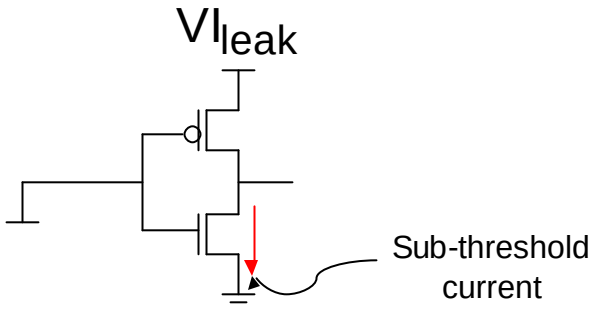


Finite slope of the input signal causes a direct current path between V_{DD} and GND for a short period of time during switching when both the NMOS and PMOS transistors are conducting

Reducing Short-circuit

1) **Lower the supply voltage V**
2) **Slope engineering – match the rise/fall time of the input and output signals**

Leakage Power



Sub-threshold current grows exponentially with increases in temperature and decreases in Vt

CMOS Power Equations

$$P = ACV^2f + tAVI_{short}f + VI_{leak}$$

Reduce the supply voltage, V

$$I_{leak} \propto \exp(-\frac{qV_t}{kT})$$

$$f_{max} \propto \frac{(V - V_t)^2}{V}$$

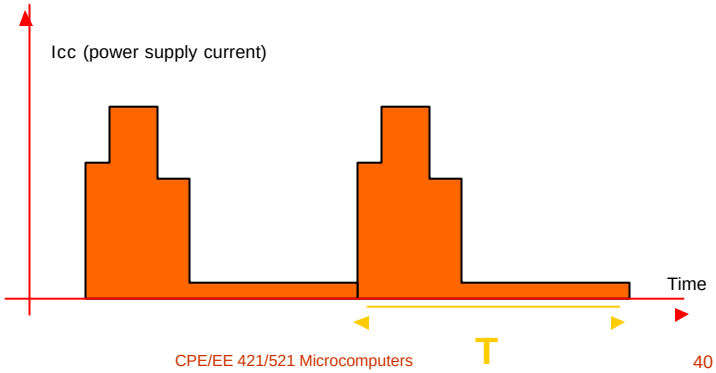
Reduce threshold V_t

How can we reduce power consumption?

- Dynamic power consumption
  - charge/discharge of the capacitive load on each gate's output
  - frequency
- Control activity
  - reduce power supply voltage
  - reduce working frequency
  - turn off unused parts (module enables)
  - use low power modes
  - interrupt driven system
- Minimize the number of transitions
  - instruction formats, coding?

Average power consumption

- Dynamic power supply current
  - Set of modules that are periodically active
  - Typical situation – real time cycle T
  - Iave = $\int$ Icc(t)dt /T
  - In most cases Iave = Σ Ii*ti/T



Low-Power Concept: Basic Conditions for Burst Mode

The example of the heat cost allocator shows that the current of the non-activity periode dominates the current consumption.

		Measure		Process data		Real-Time Clock		LCD Display
I_{AVG}	=	$I_{Measure}$	+	$I_{Calculate}$	+	I_{RTC}	+	$I_{Display}$
	=	$I_{ADC} * t_{Measure} / T$	+	$I_{active} * t_{calc} / T$	+	$I_{active} * t_{RTC} / T$	+	$I_{Display}$
	=	3mA *200µs/60s	+	0.5mA * 10ms/60s	+	0.5mA * 0.5ms/60s	+	2.1µA
	=	10nA	+	83nA	+	4nA	+	2.1µA
I_{AVG}	≅							2.1µA

The ***sleep current dominates*** the current consumption!

The currents are related to the sensor and µC system. Additional current consumption of other system parts should be added for the total system current

Battery Life

- Battery Capacity BC – [mAh]
- Battery Life
  - BL = BC / Iave
- In the previous example, standard 800 mAh batteries will allow battery life of:
  - BL = 750 mAh / 2.1 µA ≈ 44 years !!!
- Conclusion:
  - Power efficient modes
  - Interrupt driven system with processor in idle mode

Low power - features

- Peak power
  - Possible damage
- Dynamic power
  - Non-ideal battery characteristics
  - Ground bounce, di/dt noise
- Energy/operation ratio
  - MIPS/W
  - Energy x Delay

Reducing power consumption

- Logic
  - Clock tree (up to 30% of power)
  - Clock gating (turn off branches that are not used)
  - Half frequency clock (both edges)
  - Half swing clock (half of Vcc)
  - Asynchronous logic
    - completion signals
    - testing
- Architecture
  - Parallelism (increased area and wiring)
  - Speculation (branch prediction)
  - Memory systems
    - Memory access (dynamic)
    - Leakage
    - Memory banks (turn off unused)
  - Buses
    - 32-64 address/data, (15-20% of power)
    - Gray Code, Code compression

Reducing power consumption #2

- Operating System
  - Finish computation "when necessary"
  - Scale the voltage
    - Application driven
    - Automatic
- System Architecture
  - Power efficient and specialized processing cores
  - A "convergent" architecture
  - Trade-off
    - AMD K6 / 400MHz / 64KB cache – 12W
    - XScale with the same cache 450 mW @ 600 MHz (40mW@150MHz)
    - 24 processors? Parallelism?
- Other issues
  - Leakage current – Thermal runaway
  - Voltage clustering (low Vthreshold for high speed paths)

Operating Modes-General

The MSP430 family was developed for ultralow-power applications and uses different levels of operating modes. The MSP430 operating modes, give advanced support to various requirements for ultralow power and ultralow energy consumption. This support is combined with an intelligent management of operations during the different module and CPU states. An interrupt event wakes the system from each of the various operating modes and the RETI instruction returns operation to the mode that was selected before the interrupt event.

The ultra-low power system design which uses complementary metal-oxide semiconductor (CMOS) technology, takes into account three different needs:

- The desire for speed and data throughput despite conflicting needs for ultra-low power
- Minimization of individual current consumption
- Limitation of the activity state to the minimum required by the use of low power modes

Low power mode control

There are four bits that control the CPU and the main parts of the operation of the system clock generator:

- CPUOff,
- OscOff,
- SCG0, and
- SCG1.

These four bits support discontinuous active mode (AM) requests, to limit the time period of the full operating mode, and are located in the status register. The major advantage of including the operating mode bits in the status register is that the present state of the operating condition is saved onto the stack during an interrupt service request. As long as the stored status register information is not altered, the processor continues (after RETI) with the same operating mode as before the interrupt event.

Operating Modes-General

Another program flow may be selected by manipulating the data stored on the stack or the stack pointer. Being able to access the stack and stack pointer with the instruction set allows the program structures to be individually optimized, as illustrated in the following program flow:

- **Enter interrupt routine**

The interrupt routine is entered and processed if an enabled interrupt awakens

the MSP430:

- The SR and PC are stored on the stack, with the content present at the interrupt event.
- Subsequently, the operation mode control bits OscOff, SCG1, and CPUOff are cleared automatically in the status register.

- **Return from interrupt**

Two different modes are available to return from the interrupt service routine and continue the flow of operation:

- Return with low-power mode bits set. When returning from the interrupt, the program counter points to the next instruction. The instruction pointed to is not executed, since the restored low power mode stops CPU activity.
- Return with low-power mode bits reset. When returning from the interrupt, the program continues at the address following the instruction that set the OscOff or CPUOff-bit in the status register. To use this mode, the interrupt service routine must reset the OscOff, CPUOff, SCGO, and SCG1 bits on the stack. Then, when the SR contents are popped from the stack upon RETI, the operating mode will be active mode (AM).

Operating Modes - Software configurable

There are six operating modes that the software can configure:

- Active mode AM; SCG1=0, SCG0=0, OscOff=0, CPUOff=0: CPU clocks are active
- Low power mode 0 (LPM0); SCG1=0, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK and ACLK remain active
- Low power mode 1 (LPM1); SCG1=0, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - DCO's dc generator is disabled if the DCO is not used for MCLK or SMCLK when in active mode. Otherwise, it remains enabled.
  - SMCLK and ACLK remain active
- Low power mode 2 (LPM2); SCG1=1, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator automatically disabled because it is not needed for MCLK or SMCLK
  - DCO's dc-generator remains enabled
  - ACLK remains active

Operating Modes #2

- Low power mode 3 (LPM3); SCG1=1, SCG0=1,OscOff=0,CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO's dc-generator is disabled
  - ACLK remains active
- Low power mode 4 (LPM4); SCG1=X, SCG0=X, OscOff=1,CPUOff=1:
  - CPU is disabled
  - ACLK is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO's dc-generator is disabled
  - Crystal oscillator is stopped

Operating Modes-Low Power Mode in details

- **Low-Power Mode 0 and 1 (LPM0 and LPM1)**

Low power mode 0 or 1 is selected if bit CPUOff in the status register is set. Immediately after the bit is set the CPU stops operation, and the normal operation of the system core stops. The operation of the CPU halts and all internal bus activities stop until an interrupt request or reset occurs. The system clock generator continues operation, and the clock signals MCLK, SMCLK, and ACLK stay active depending on the state of the other three status register bits, SCG0, SCG1, and OscOff.

The peripherals are enabled or disabled with their individual control register settings, and with the module enable registers in the SFRs. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

- **Low-Power Modes 2 and 3 (LPM2 and LPM3)**

Low-power mode 2 or 3 is selected if bits CPUOff and SCG1 in the status register are set. Immediately after the bits are set, CPU, MCLK, and SMCLK operations halt and all internal bus activities stop until an interrupt request or reset occurs.

Peripherals that operate with the MCLK or SMCLK signal are inactive because the clock signals are inactive. Peripherals that operate with the ACLK signal are active or inactive according with the individual control registers and the module enable bits in the SFRs. All I/O port pins and the RAM/registers are unchanged. Wake up is possible by enabled interrupts coming from active peripherals or RST/NMI.

Operating Modes-Low Power Mode in details

- **Low-Power Mode 4 (LPM4)**

System Resets, Interrupts, and Operating Modes In low power mode 4 all activities cease; only the RAM contents, I/O ports, and registers are maintained. Wake up is only possible by enabled external interrupts.

Before activating LPM4, the software should consider the system conditions during the low power mode period . The two most important conditions are environmental (that is, temperature effect on the DCO), and the clocked operation conditions.

The environment defines whether the value of the frequency integrator should be held or corrected. A correction should be made when ambient conditions are anticipated to change drastically enough to increase or decrease the system frequency while the device is in LPM4.

Operating Modes-Examples

- **The following example describes entering into low-power mode 0.**

```
;===Main program flow with switch to CPUOff Mode==============
BIS #18h,SR ;Enter LPM0 + enable general interrupt GIE
            ;(CPUOff=1, GIE=1). The PC is incremented
            ;during execution of this instruction and
            ;points to the consecutive program step.
......      ;The program continues here if the CPUOff
            ;bit is reset during the interrupt service
            ;routine. Otherwise, the PC retains its
            ;value and the processor returns to LPM0.
```

- **The following example describes clearing low-power mode 0.**

```
;===Interrupt service routine=================================
......                ;CPU is active while handling interrupts
BIC #10h,0(SP)        ;Clears the CPUOff bit in the SR contents
                      ;that were stored on the stack.
RETI                  ;RETI restores the CPU to the active state
                      ;because the SR values that are stored on
                      ;the stack were manipulated. This occurs
                      ;because the SR is pushed onto the stack
                      ;upon an interrupt, then restored from the
                      ;stack after the RETI instruction.
```


Operating Modes C Examples

- **C – programming msp430x14x.h**

```
/************************
* STATUS REGISTER BITS
************************/

#define C       0x0001
#define Z       0x0002
#define N       0x0004
#define V       0x0100
#define GIE     0x0008
#define CPUOFF  0x0010
#define OSCOFF  0x0020
#define SCG0    0x0040
#define SCG1    0x0080

/* Low Power Modes coded with
     Bits 4-7 in SR */
/* Begin #defines for assembler */
#ifndef __IAR_SYSTEMS_ICC
#define LPM0    CPUOFF
#define LPM1    SCG0+CPUOFF
#define LPM2    SCG1+CPUOFF
#define LPM3    SCG1+SCG0+CPUOFF
#define LPM4    SCG1+SCG0+OSCOFF+CPUOFF
/* End #defines for assembler */

#else /* Begin #defines for C */
#define LPM0_bits   CPUOFF
#define LPM1_bits   SCG0+CPUOFF
#define LPM2_bits   SCG1+CPUOFF
#define LPM3_bits   SCG1+SCG0+CPUOFF
#define LPM4_bits   SCG1+SCG0+OSCOFF+CPUOFF
```

- **…**

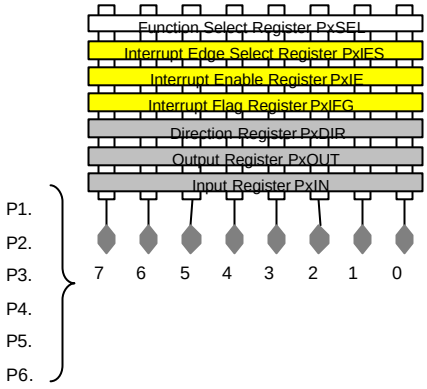
```
#include "In430.h"

#define LPM0      _BIS_SR(LPM0_bits) /* Enter LP Mode 0 */
#define LPM0_EXIT _BIC_SR(LPM0_bits) /* Exit LP Mode 0 */
#define LPM1      _BIS_SR(LPM1_bits) /* Enter LP Mode 1 */
#define LPM1_EXIT _BIC_SR(LPM1_bits) /* Exit LP Mode 1 */
#define LPM2      _BIS_SR(LPM2_bits) /* Enter LP Mode 2 */
#define LPM2_EXIT _BIC_SR(LPM2_bits) /* Exit LP Mode 2 */
#define LPM3      _BIS_SR(LPM3_bits) /* Enter LP Mode 3 */
#define LPM3_EXIT _BIC_SR(LPM3_bits) /* Exit LP Mode 3 */
#define LPM4      _BIS_SR(LPM4_bits) /* Enter LP Mode 4 */
#define LPM4_EXIT _BIC_SR(LPM4_bits) /* Exit LP Mode 4 */
#endif /* End #defines for C */
```

```
/* - in430.h -
    Intrinsic functions for the MSP430
*/

unsigned short _BIS_SR(unsigned short);
unsigned short _BIC_SR(unsigned short);
```

Digital I/O all MSP430



Port1 Port2	Port3 ... Port6
yes	yes
yes	no
yes	no
yes	no
yes	yes
yes	yes
yes	yes

Chapter 9, User's Manual pages 9-1 to 9-7

Digital I/O Introduction

- MSP430 family – up to 6 digital I/O ports implemented, P1-P6
- MSP430F14x – all 6 ports implemented

Ports P1 and P2 have interrupt capability.

Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising edge or falling edge of an input signal.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers

The digital I/O is configured with user software

Digital I/O Registers Operation

Input Register PnIN

Each bit in each PnIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.

Bit = 0: The input is low

Bit = 1: The input is high

Do not write to PxIN. It will result in increased current consumption

Output Registers PnOUT

Each bit in each PnOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function and output direction.

Bit = 0: The output is low

Bit = 1: The output is high

Digital I/O Operation

Direction Registers PnDIR

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

Function Select Registers PnSEL

Port pins are often multiplexed with other peripheral module functions.

Bit = 0: I/O Function is selected for the pin

Bit = 1: Peripheral module function is selected for the pin

Digital I/O Operation

Interrupt Flag Registers P1IFG, P2IFG

(only for P1 and P2)

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

(Only transitions, not static levels, cause interrupts)

Interrupt Edge Select Registers P1IES, P2IES

(only for P1 and P2)

Each PnIES bit selects the interrupt edge for the corresponding I/O pin.

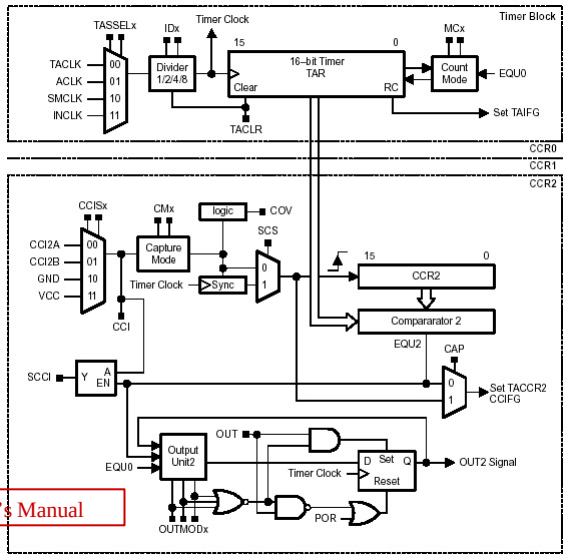
Bit = 0: The PnIFGx flag is set with a low-to-high transition

Bit = 1: The PnIFGx flag is set with a high-to-low transition

Timer_A MSP430x1xx

- 16-bit counter with 4 operating modes
- Selectable and configurable clock source
- Three (or five) independently configurable capture/compare registers with configurable inputs
- Three (or five) 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_A5 - MSP430x1xx Block Diagram



Page 11-3, User's Manual

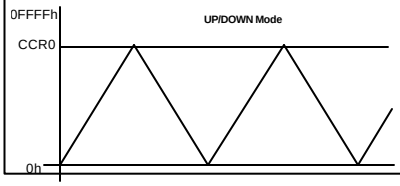
Timer_A Counting Modes

Stop/Halt Mode

Timer is halted with the next +CLK

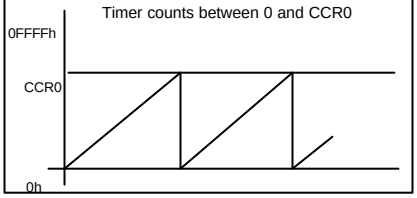
UP/DOWN Mode

Timer counts between 0 and CCR0 and 0



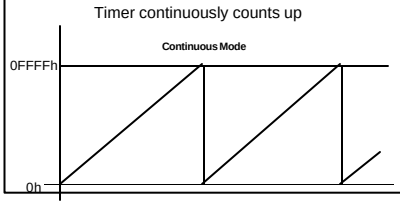
UP Mode

Timer counts between 0 and CCR0

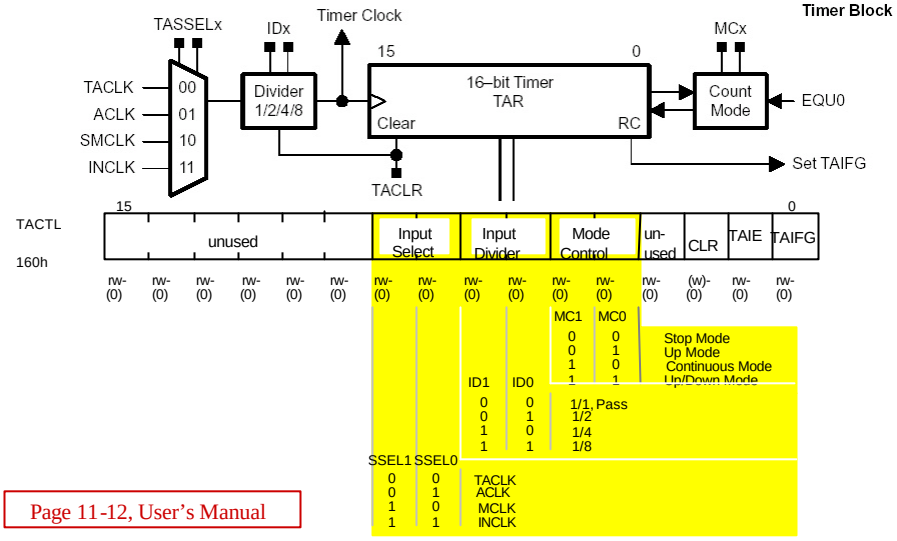


Continuous Mode

Timer continuously counts up

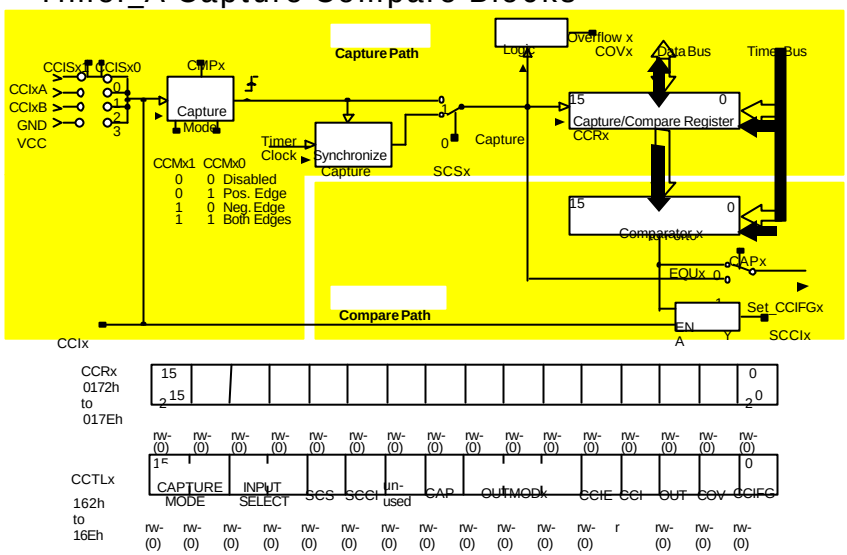


Timer_A 16-bit Counter

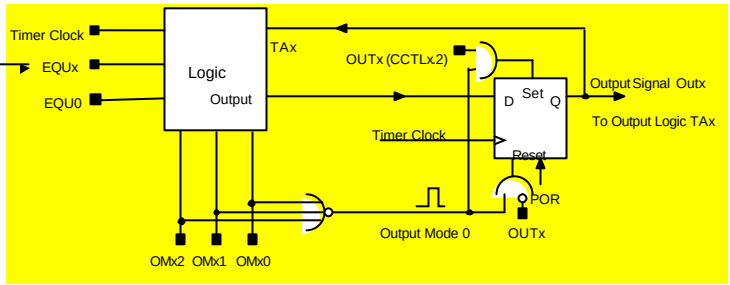


Page 11-12, User's Manual

Timer_A Capture Compare Blocks

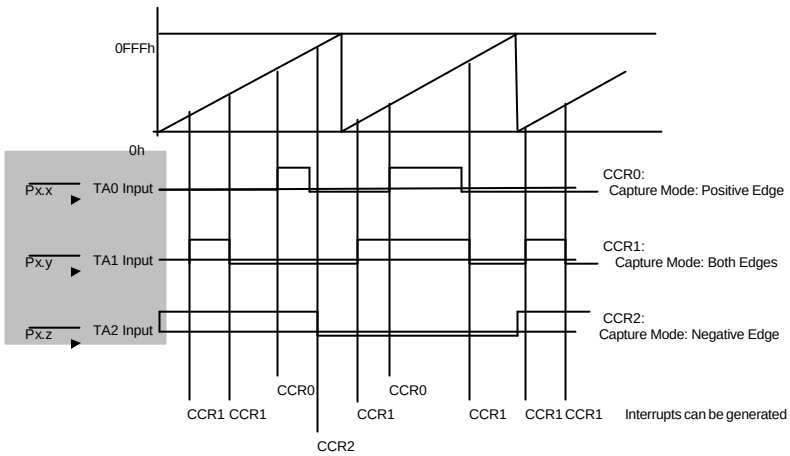


Timer_A Output Units



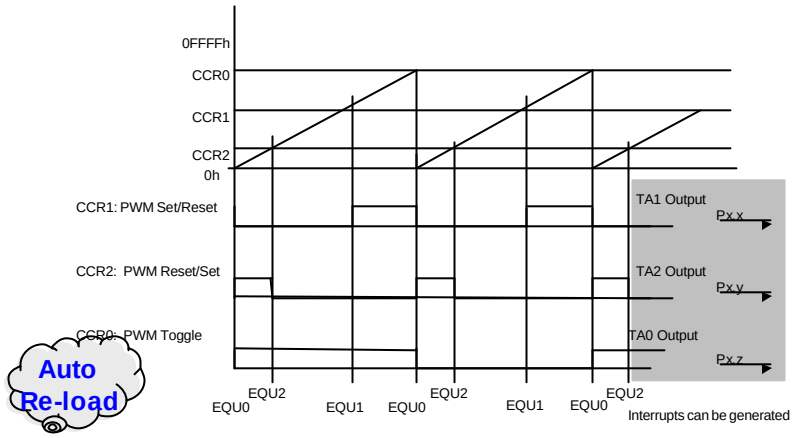
OMx2	OMx1	OMx0	Function	Operational Conditions
0	0	0	Output Mode	Outx signal is set according to Outx bit
0	0	1	Set	EQUx sets Outx signal clock synchronous with timer clock
0	1	0	PWM Toggle/Reset	EQUx toggles Outx signal, reset with EQU0, clock sync. with timer clock
0	1	1	PWM Set/Reset	EQUx sets Outx signal, reset with EQU0, clock synchronous with timer clock
1	0	0	Toggle	EQUx toggles Outx signal, clock synchronous with timer clock
1	0	1	Reset	EQUx resets Outx signal clock synchronous with timer clock
1	1	0	PWM Toggle/Reset	EQUx toggles Outx signal, set with EQU0, clock synchronous with timer clock
1	1	1	PWM Set/Reset	EQUx resets Outx signal, set with EQU0, clock synchronous with timer clock

Timer_A Continuous-Mode Example



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

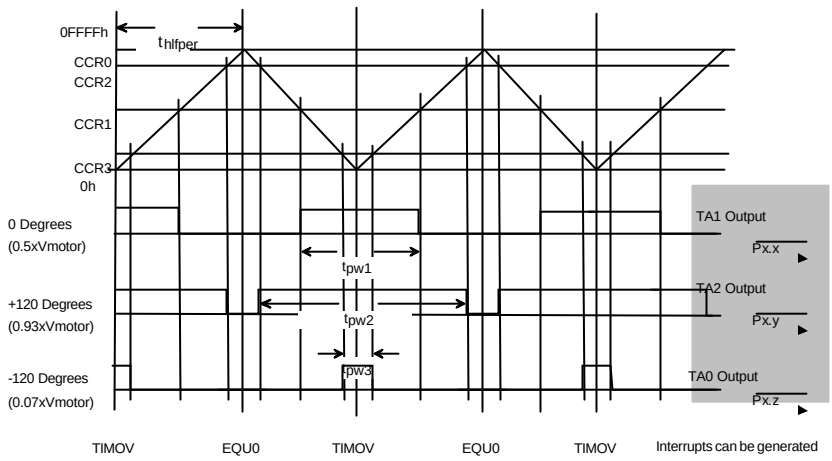
Timer_A PWM Up-Mode Example



Output Mode 4: PWM Toggle

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

Timer_A PWM Up/Down Mode Example



Example shows Symmetric PWM Generation - Digital Motor Control