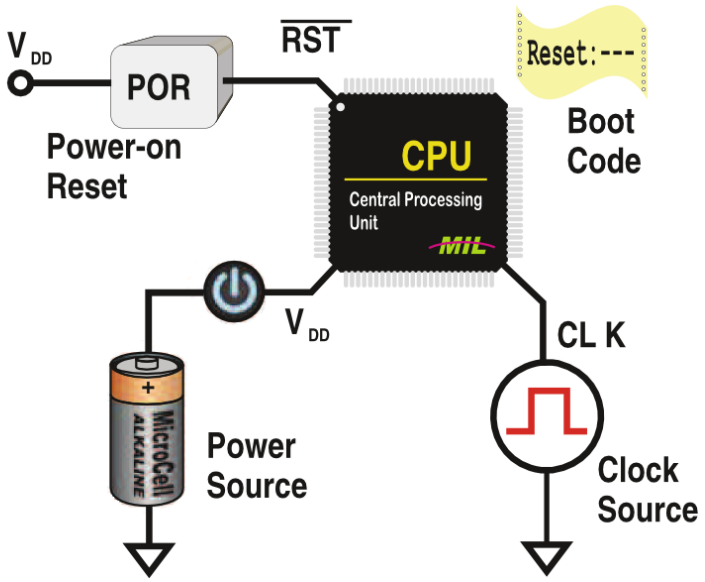
Week 5
FUNDAMENTALS OF INTERFACING AND TIMERS for MSP430

Hacettepe University

Elements in Basic MCU Interface

Fig. 6.1 Components in a basic CPU interface: power source, clock generator, power-on-reset, and boot sequence



- Power Source
  - Feeds CPU and peripherals
- Clock Oscillators
  - System synchronization
- Power-on Reset
  - Physical reset hardware
- Booting Function
  - System configuration and initialization

PROCESSORS' POWER SOURCES

Main Function

- Provide Power to CPU and Surrounding Electronics
- Establish reference levels for internal device operation

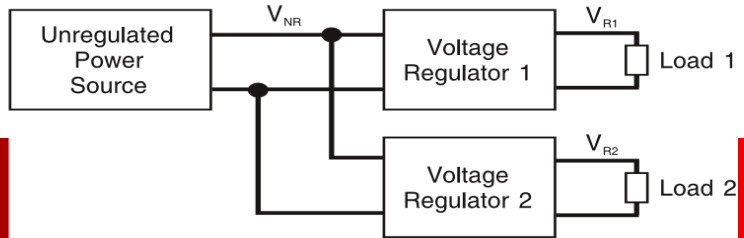
Basic Requirements

- Steady voltage source
- Sufficient current capability
- Load regulation
- Power quality

Implications

- System functionality and integrity
- Signal compatibility

Fig. 6.2 Structure of a typical MCU power supply



Absolute Maximum Ratings

- Levels of stress that if exceeded will cause permanent damage to the device
- If used for extended periods will affect device reliability
- DO NOT design for operating devices at these levels

Absolute Maximum Ratings* T_A = 25°C unless otherwise noted

Symbol	Parameter	Value							Units
		4001	4002	4003	4004	4005	4006	4007	
V_{RRM}	Peak Repetitive Reverse Voltage	50	100	200	400	600	800	1000	V
$I_{F(AV)}$	Average Rectified Forward Current, .375 " lead length @ T_A = 75°C	1.0							A
I_{FSM}	Non-repetitive Peak Forward Surge Current 8.3 ms Single Half-Sine-Wave	30							A
T_{stg}	Storage Temperature Range	-55 to +175							°C
T_J	Operating Junction Temperature	-55 to +175							°C

*These ratings are limiting values above which the serviceability of any semiconductor device may be impaired.

Absolute Maximum Ratings

T_A = 25°C		VALUE	UNIT
V_{DS}	Drain-to-Source Voltage	60	V
V_{GS}	Gate-to-Source Voltage	±20	V
I_D	Continuous Drain Current (Package limited)	200	A
	Continuous Drain Current (Silicon limited), T_C = 25°C	349	
	Continuous Drain Current (Silicon limited), T_C = 100°C	247	
I_{DM}	Pulsed Drain Current (1)	400	A
P_D	Power Dissipation	375	W
T_J, T_{stg}	Operating Junction and Storage Temperature Range	–55 to 175	°C
E_{AS}	Avalanche Energy, single pulse I_D = 128 A, L = 0.1 mH, R_G = 25 Ω	819	mJ

(1) Max $R_{\theta JC}$ = 0.4°C/W, pulse duration ≤100 µs, duty cycle ≤1%

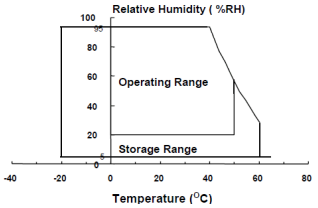
Absolute Maximum Ratings

Ta=25+/-2°C

Parameter	Symbol	Min.	Max.	Unit	Remarks
Power Supply Voltage	V_{DD}	-0.3	4.0	V	
Logic Supply Voltage	V_{IN}	-0.3	V_{DD}+0.3	V	
Lamp Current	I_L	3.0	7.0	mArms	(1)
Lamp frequency	F_L	45	80	kHz	
Operating Temperature	T_{OP}	0	+50	℃	(2)
Operating Humidity	RH_{OP}	-	80	%	
Storage Temperature	T_{SP}	-20	+60	℃	
Storage Humidity	RH_{SP}	-	90	%	

Note (1) Permanent damage to the device may occur if maximum values are exceeded Functional operation should be restricted to the condition described under normal operating conditions.

Note (2) Temperature and relative humidity range are shown in the figure below.
95 % RH Max. (40 °C ≥ Ta)
Maximum wet - bulb temperature at 39 °C or less. (Ta > 40 °C) No condensation.



Recommended Operating Conditions

- Manufacturer's recommended levels for reliable operation
- Conditions the application circuit should provide to device for it to function as intended
- Shall be used for design calculations

7.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

PARAMETER		MIN	MAX	UNIT
SUPPLIES				
RX_ANA_SUP	AFE analog supply	2.0	3.6	V
RX_DIG_SUP	AFE digital supply	2.0	3.6	V
TX_CTRL_SUP	Transmit controller supply	3.0	5.25	V
LED_DRV_SUP	Transmit LED driver supply			
	Difference between LED_DR TX_CTRL_SUP			
TEMPERATURE				
	Specified temperature range			
	Storage temperature range			

(1) V_{LED} refers to the maximum voltage drop ac (in H-bridge mode) and from the TXP and T
(2) V_{CABLE} refers to voltage drop across any cal

7.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

		MIN	MAX	UNIT
VM	Power supply voltage range(1)	4	18	V
VREF	Reference rms voltage range(2)	1	3.3	V
f_{PWM}	Applied STEP signal	0	250	kHz
I_{VINT}	VINT external load current			
I_{rms}	Motor rms current per H-bridge(3)			
T_A	Operating ambient temperature			

(1) Note that $R_{DS(ON)}$ increases and maximum outp
(2) Operational at VREF between 0 to 1 V, but accu
(3) Power dissipation and thermal limits must be ob

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

		MIN	MAX	UNIT
V_{CC}	Supply voltage	2.3	5.5	V
V_{NC} V_{NO} V_{COM}	Signal path voltage	V_{CC} – 5.5	V_{CC}	V
V_{IN}	Digital control	GND	V_{CC}	V

Sample Abs. Max. Ratings Specs

- Voltage Levels
- Maximum and minimum supply values
- Applied to any pin
- Diode Current
- ESD clamping diode current
- Storage Temperature

Table 6.1: Absolute maximum ratings for MSP430G2231

Parameter	Condition	Limits
Voltage applied at V_{CC} to V_{SS}		−0.3V to 4.1V
Voltage applied to any pin		0.3V to V_{CC} + 0.3V
Diode current at any device pin		±2mA
Storage temperature range, T_{stg}	Unprogrammed device	−55◦Cto 150◦C
	Programmed device	−40 to 85 ◦C

- **Recommendation**
  - **Stay below absolute maximum ratings**
  - **Ground level specified independent of the Vcc**

Table 6.2: Recommended operating conditions for MSP430G2231

Symb.	Parameter	Condition	Min.	Max.	Unit
V_{CC}	Supply voltage	Program execution	1.8	3.6	V
		Flash programming	2.2	3.6	
V_{SS}	Supply voltage		0	0	V
TA	Operating temp.	Free air	−40	85	◦C
f_{clk}	Clock frequency	V_{CC} = 1.8V,	dc	4.15	MHz
		Duty cycle = 50% ± 10%			
		V_{CC} = 2.7V,	dc	12	
		Duty cycle = 50% ± 10%			
		V_{CC} = 3.3V,	dc	16	
		Duty cycle = 50% ± 10%			

Power vs frequency

- Minimum Power Dissipation
  - Obtained at minimum V_{DD}
  - Power varies with V_{DD2}
- Maximum f_{clk} Limit
  - $f_{clk\ max}$ is usually limited by the chosen V_{DD} level
  - Direct relationship
- Choose Wisely
  - Use the minimum frequency and voltage necessary for proper functionality

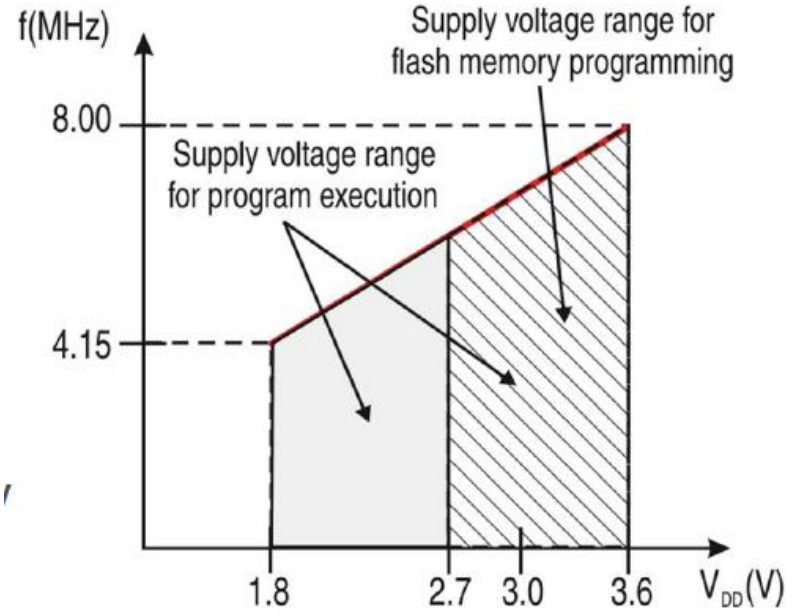


Fig. 6.3: Frequency versus supply voltage plot for an MSP430F149

Power Supply Capacity

- Regulator Capacity (I_R)

$$I_R \geq I_L = \sum_{i=1}^{n} I_{Li} \quad (1)$$

where I_{Li} = current from load i = 1,2,...,n

- Input voltage (V_{NR})

$$V_{NR} \geq (V_R + V_D) \quad (2)$$

where

V_R = regulated voltage and
V_D = dropout voltage

- Non-regulated capacity

$$I_{NR} \geq (I_R + I_{gnd}) \quad (3)$$

- Non-regulated power:

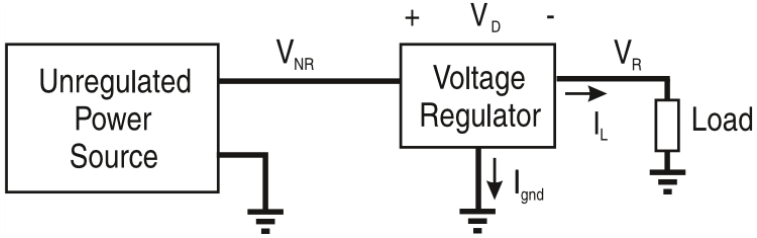
$$P_{NR} = V_R (I_R + I_{gnd}) \quad (4)$$

- Power to the load:

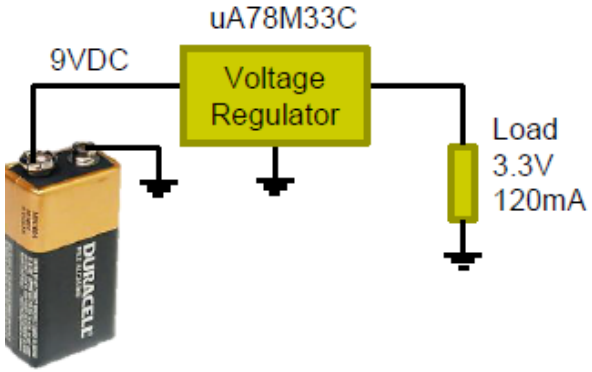
$$P_R = V_R \cdot I_L \quad (5)$$

- Power Efficiency:

$$Eff = (P_R / P_{NR}) \times 100\% \quad (6)$$



Example 6.1 : *Consider using a standard 9V alkaline battery for feeding a 3.3V, 120mA load via a linear voltage regulator uA78M33C. Estimate the approximated battery life and usage efficiency assuming a constant load current.*



Regulator data:

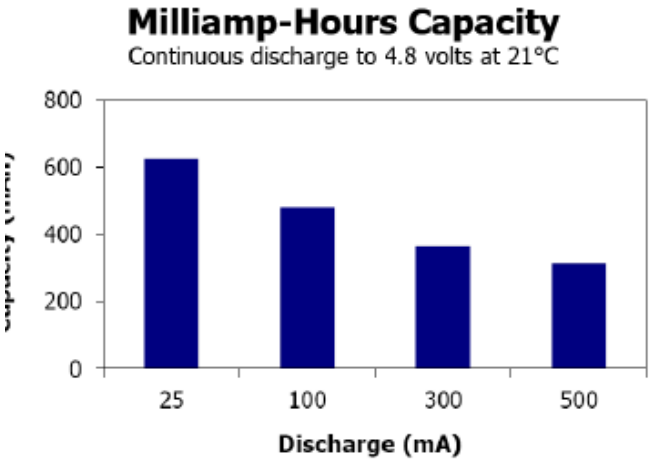
$V_o = 3.3V \quad V_D = 2.0V \quad I_{out} = 500mA$

$I_{bias} = I_{gnd} = 6mA$

Solution: *Total load seen by battery (I_{NR})*

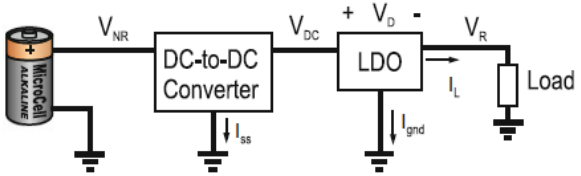
$I_{NR} = I_L + I_{gnd} = 126mA \leq I_{out}$

From battery datasheet: Q_{bat} = 480mAh at a 100mA discharge rate. At 300mA, Q_{bat} drops to 380mAh. Interpolating we get $Q_{bat}|_{126mA} \cong 447mAh$



$$t_{bat} = \frac{Q_{bat}}{I_{NR}} = \frac{447\,mAh}{126\,mA} = 3.55\,h$$

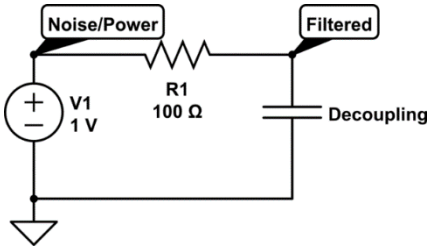
$$Eff = \frac{V_R \times I_R}{V_{NR} \times (I_R + I_{gnd})} \times 100\,\% = \frac{3.3\,V \times 120\,mA}{9\,V \times (120+6)mA} = 34.92\,\%$$



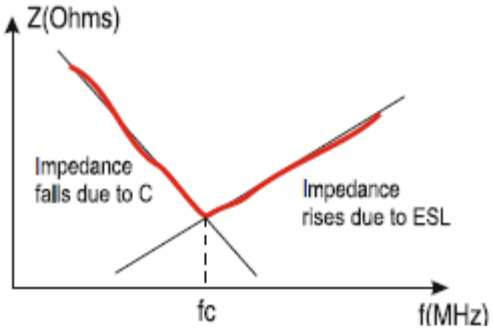
.5 Power supply integrating a DC-to-DC converter and LDO

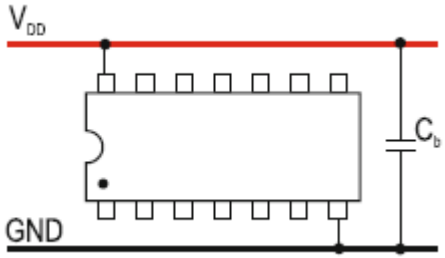
POWER SUPPLY NOISE CONTROL

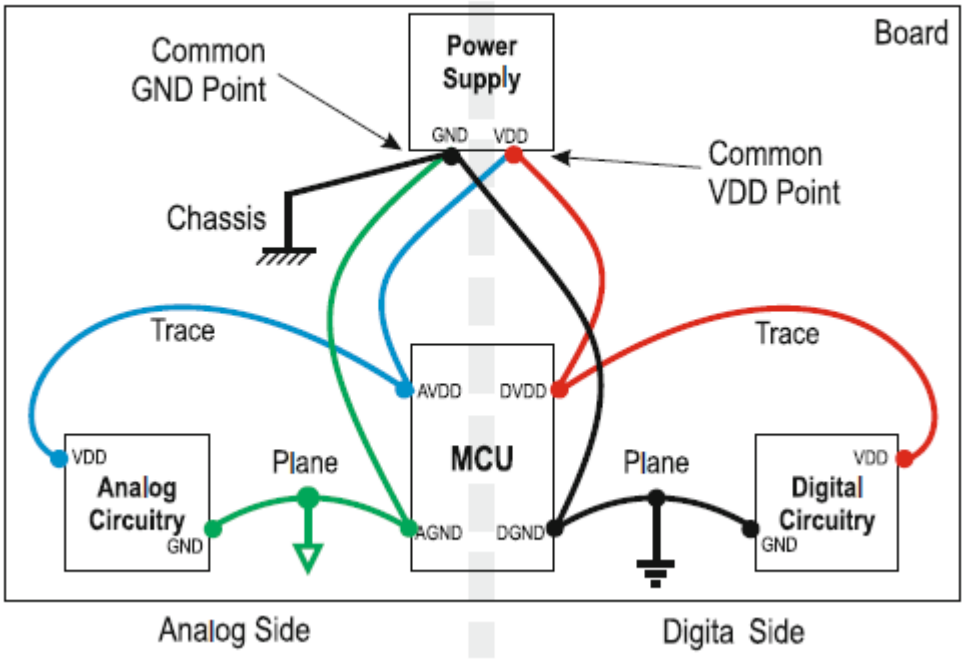
- Coping with Power Supply Noise
  - Reducing the effect of noise in the power distribution lines
- Common Methods
  - Bypassing Techniques
  - Source Decoupling
  - Wise Power Distribution
- Combine ALL of them
  - Plan the power distribution network
  - Power lines noise increases with:
    - Clock frequency
    - Power distribution loop length
    - Dirty power supplies



Bypassing Capacitors



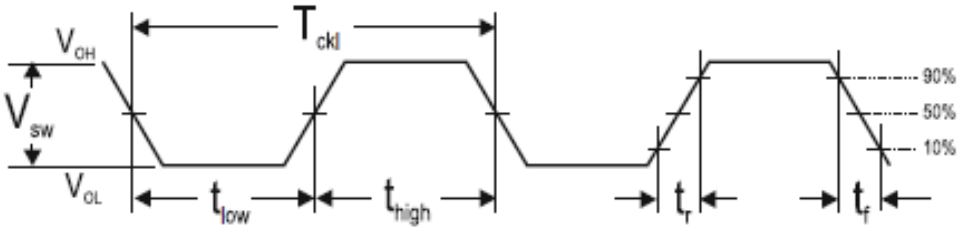




Bypassing refers to the act of reducing a high frequency current flow in a circuit path by adding a shunting component that reacts to the target frequency. The most commonly used shunting devices in microprocessor-based designs are bypassing capacitors.

A bypass capacitor reduces the rate of change of the current circulating in the power line by providing a high-frequency, low impedance path to the varying load current. Two factors determine the effectiveness of a bypassing capacitor: size and location.

Clock Issues



- **Clock Frequency**
- **Clock Duty Cycle**

The clock duty cycle defines the ratio of the high to the period of the clock signal

- **Clock Stability**

Example 6.2 *Consider a 12MHz clock signal that exhibits a* ±10% *deviation from its nominal value. Two applications employing such a clock signal are analyzed: a dynamic display that sets* ***a 60Hz refresh ratio from this clock*** *and a* ***real-time clock slated to run uninterruptedly for weeks****. Evaluate the impact of the clock accuracy on each system.*

Solution: *The impact of the clock deviation will be analyzed independently for each system.*

Impact on the display system*: A 10% frequency deviation would translate into an equally proportional deviation in the refresh rate, implying that the actual rate could be 6Hz off the target value. In the worst case, assuming a 10% frequency loss, the refresh ratio would be 54Hz. Considering that for persistence of vision, the human eye only requires 24Hz of refresh ratio to perceive motion, this 10% change of frequency can be deemed as negligible.*

Impact on the RTC*: A 10% deviation in frequency would cause the RTC to drift from the actual time at a rate of 6s each minute or 2h and 24min per day. At the end of only one week, assuming a negative δ fCLK, the error would accumulate to 16.8h, which would be totally unacceptable.*

Clock Issues

- **Clock Jitter** Clock Jitter refers to the uncertainty in the periodicity of a clock signal
  Example 6.3 *Consider a microprocessor with a total system clock jitter specified at 150ps under the JESD65B standard.*
  This specification establishes that the total time deviation in the signal period resulting from the sum of all deterministic and random sources in the clock frequency over a minimum of $10^{\wedge 4}$ *cycles cannot exceed 150ps. For some devices, the number of cycles specified in the JESD65B standard establishes a bare minimum. A commonly used number is* $10^{\wedge 5}$ *cycles, but for some devices it can reach as much as* $10^{\wedge 12}$ *cycles.*
- **Clock Drift** FrequencyDrift refers to the linear component of a systematic change in the frequency of an oscillator over time. *For example, a 4MHz oscillator with an age induced drift of 20PPM (parts per million) per year will deviate its frequency by 80Hz every year.*

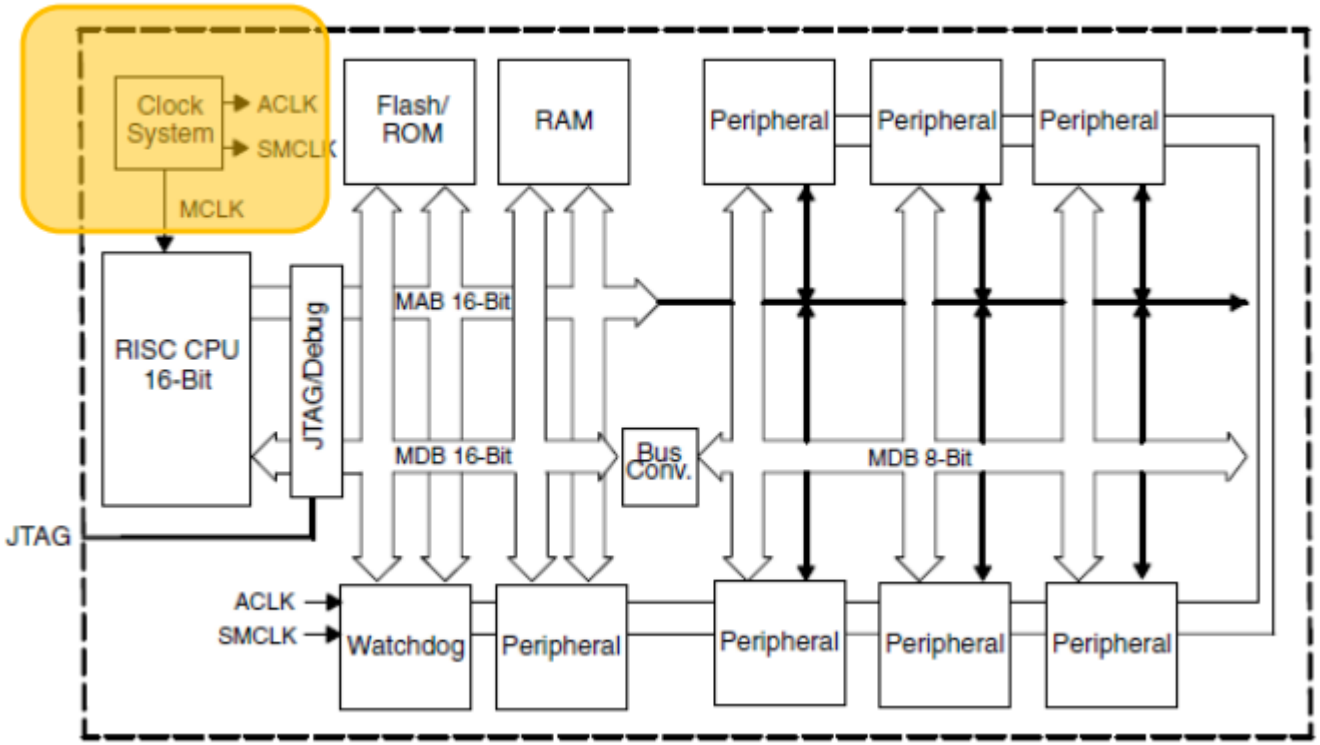
Choosing the Clock

- *What is the fastest event the system will need to handle?*
- *Has the value of VDD been assigned?*
- *What peripherals will share the same clock frequency?*
- *How precise does the clock need to be?*
- *What are the capabilities of the Clock System in my MCU?*

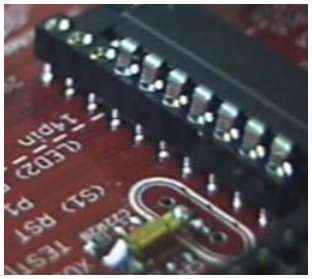
Table 6.3 Common frequencies for embedded systems applications

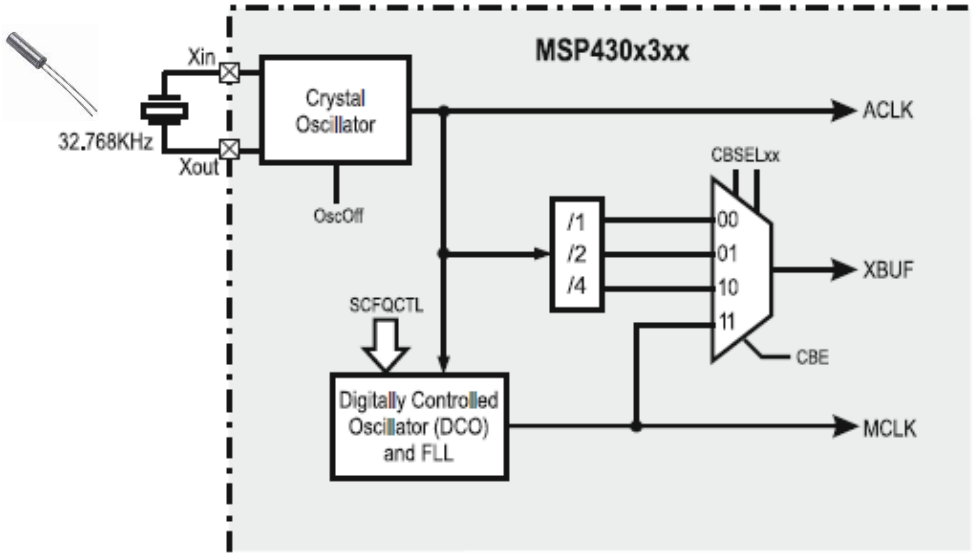
Freq. (MHz)	Typical application
0.032768	Real-time clocks. Allows binary division to 1.0000 Hz ($2^{15} \times 1$ Hz)
1.843200	UART clock. ($16 \times 115,200$ baud or $96 \times 16 \times 1,200$ baud)
2.457600	UART clock. ($64 \times 38,400$ baud or $2,048 \times 1,200$ baud)
3.276800	Allows binary division to 100 Hz ($32,768 \times 100$ Hz, or $2^{15} \times 100$ Hz)
3.579545	NTSC M color subcarrier and DTMF generators
3.686400	UART clock (2×1.8432 MHz)
4.096000	Allows binary division to 1 kHz ($2^{12} \times 1$ kHz)
4.194304	Real-time clocks, divides to 1 Hz signal ($2^{22} \times 1$ Hz)
6.144000	UART baud rates up to 38,400.
6.553600	Allows binary division to 100 Hz ($2^{16} \times 100$ Hz)
7.372800	UART clock (4×1.8432 MHz)
9.216000	Allows integer division to 1,024 kHz (2^{10})
11.059200	UART clock (6×1.8432 MHz)

CLOCK ON MSP430



Choosing Clock





The earliest generation of the MSP430 device family, the MSP430x3xx uses a *Frequency-Locked Loop (FLL)* clock module as system clock generator. This clocking system consists of two oscillators:
a crystal oscillator and a frequency stabilized & RC-based, Digitally Controlled Oscillator (DCO). The DCO is locked to a multiple of the crystal frequency, forming a frequency-locked loop (FLL). frequency stability and quick startup.

This clocking system fundamentally provides two clock signals:

- ***Main Clock* (MCLK),** taken from the DCO output
- ***Auxiliary Clock* signal (ACLK),** taken out from the crystal oscillator.

A buffered, software selectable output XBUF is also available, that provides as output either **MCLK, ACLK, ACLK/2, or ACLK/4.** The XBUF clock can also be turned off via software.
The MCLK signal is used to drive the CPU, while **ACLK and XBUF can be software selected to drive peripherals.**

Choosing Clock

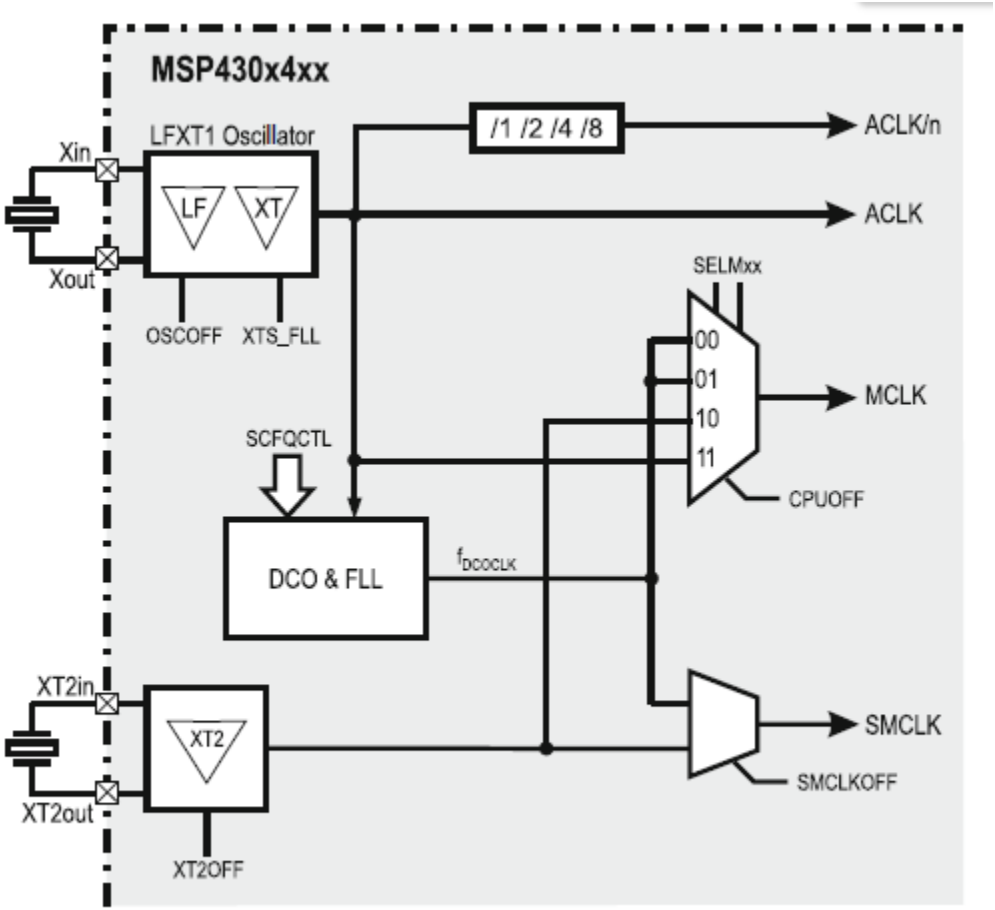


Fig. 6.17 Simplified block diagram of FLL+ clock module in MSP430x4xx devices

The second generation of MSP430, the MSP430x4xx, features an improved clock generator designated the *FLL+ Clock Module*.

LFXT1, DCO and now an XT2!

The FLL+ module can provide four clock signals to the system:

- ACLK,
- ACLK/*n* (*n* = 1, 2, 4, or 8),
- MCLK,
- SMCLK.

Choosing Clock

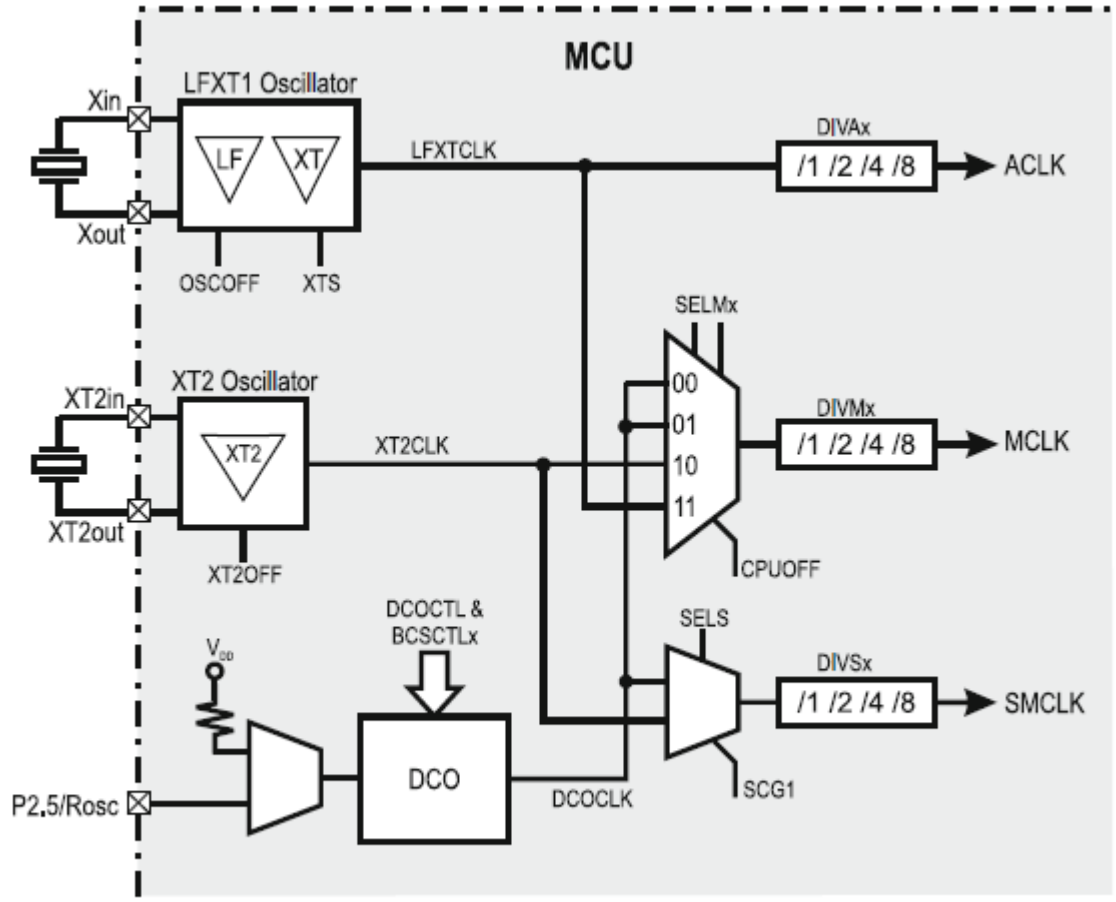


Fig. 6.18 Simplified block diagram of basic clock module in MSP430x1xx devices

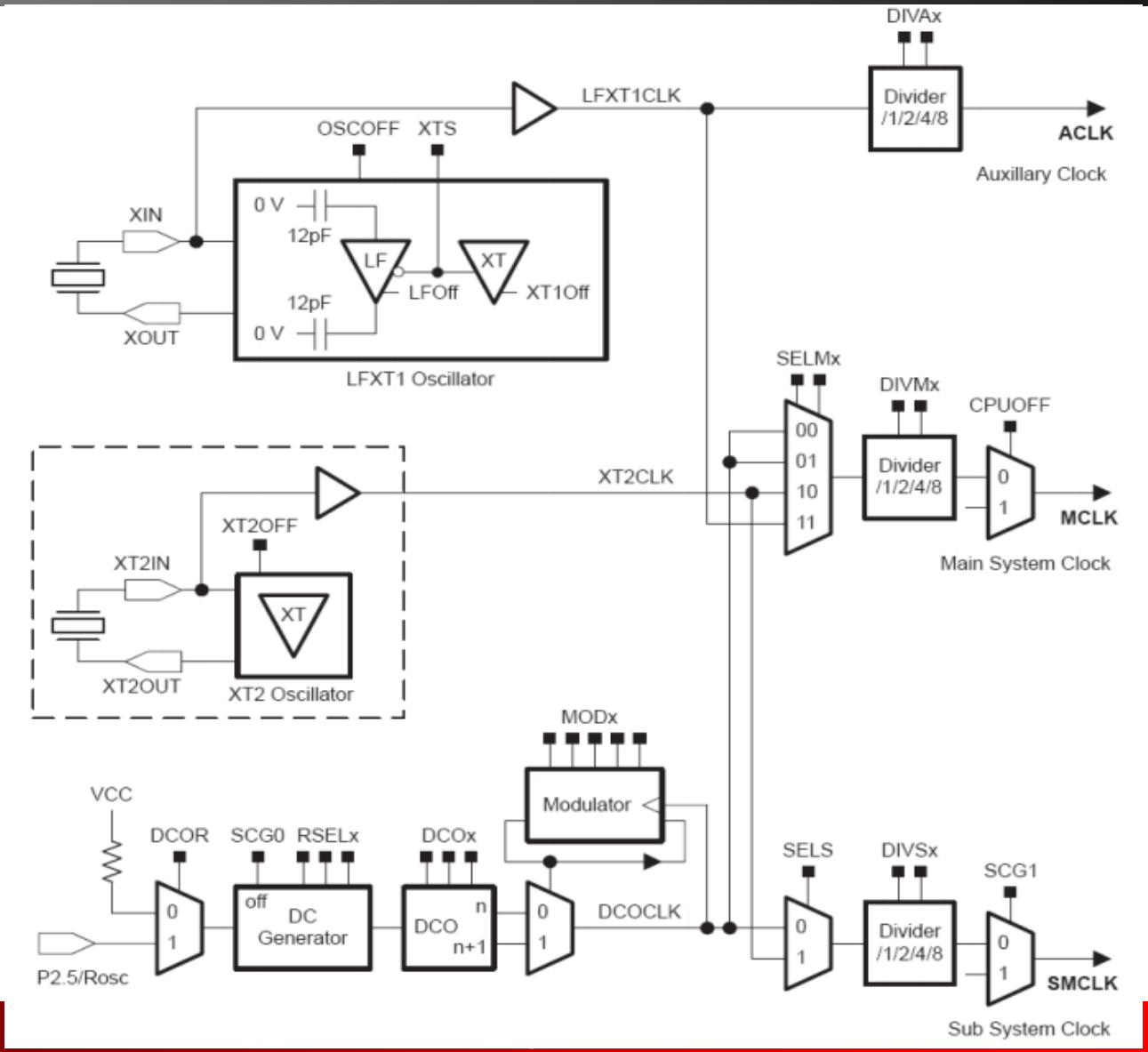
Generations x1xx and x2xx featured a modified clock module with respect to the design included in earlier generations. This new clock generator is designated as ***Basic Clock Module***.

DCO is now redesigned to operate in open loop (no FLL), while offering a better frequency stability without the need of an external crystal.

In addition, only three clock signals are made available:

- ACLK,
- MCLK, and
- SMCLK.

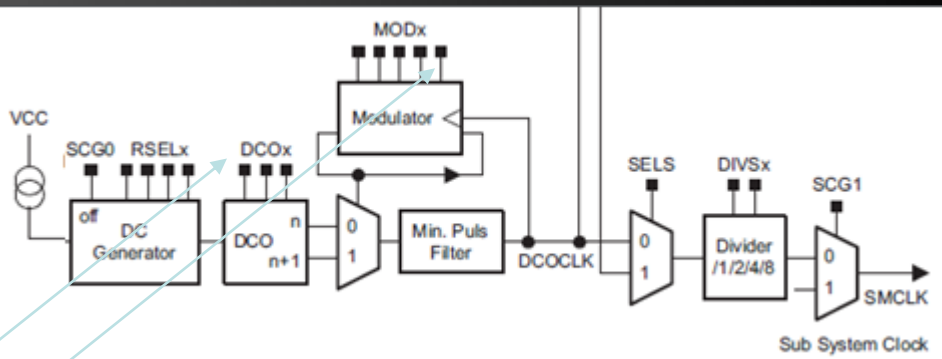
Basic Clock Module



MSP430 Basic Clock Module

- ACLK: Auxiliary clock. The signal is sourced from LFXT1CLK with a divider of 1, 2, 4, or 8. ACLK can be used as the clock signal for Timer A and Timer B.
- MCLK: Master clock. The signal can be sourced from LFXT1CLK, XT2CLK (if available), or DCOCLK with a divider of 1, 2, 4, or 8. MCLK is used by the CPU and system.
- SMCLK: Sub-main clock. The signal is sourced from either XT2CLK (if available), or DCOCLK with a divider of 1, 2, 4, or 8. SMCLK can be used as the clock signal for Timer A and Timer B.

DCO Control Register



DCOCTL, DCO Control Register

7	6	5	4	3	2	1	0
DCOx			MODx				
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

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−MOD) the f_{DCO} frequency is used. Not useable when DCOx=7.

Example: DCOCTL = 0xD3; // C code to set DCOCTL.

After a PUC, RSELx = 7 and DCOx = 3, allowing the DCO to start at a mid-range frequency. MCLK and SMCLK are sourced from DCOCLK. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution typically begins from PUC in less than 2 µs. The typical DCOx and RSELx ranges and steps are shown in Figure 5-6.

The frequency of DCOCLK is set by the following functions:

- The four RSELx bits select one of sixteen nominal frequency ranges for the DCO. These ranges are defined for an individual device in the device-specific data sheet.
- The three DCOx bits divide the DCO range selected by the RSELx bits into 8 frequency steps, separated by approximately 10%.
- The five MODx bits, switch between the frequency selected by the DCOx bits and the next higher frequency set by DCOx+1. When DCOx = 07h, the MODx bits have no effect because the DCO is already at the highest setting for the selected RSELx range.

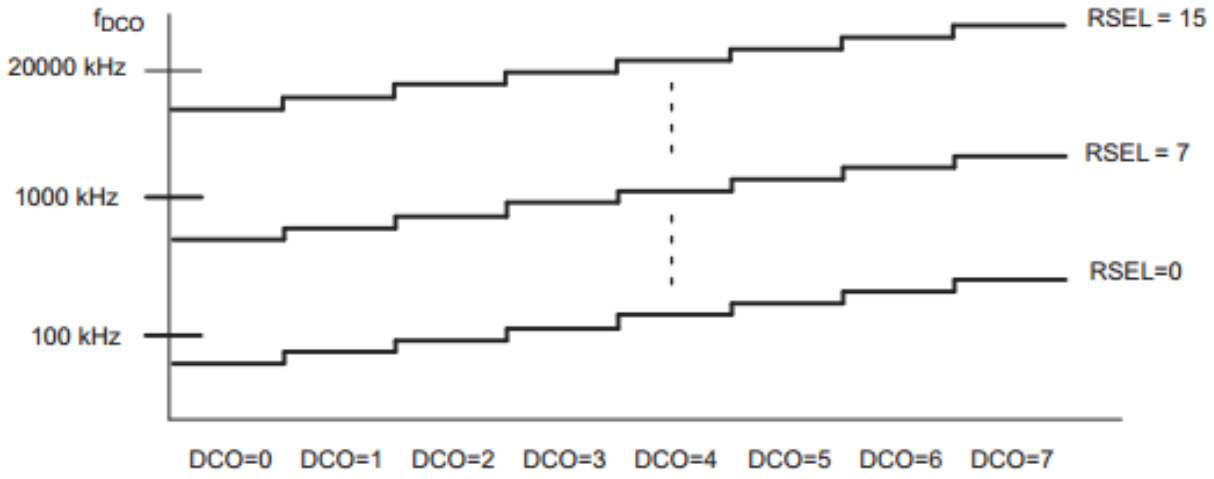


Figure 5-6. Typical DCOx Range and RSELx Steps

DCO Registers

Basic Clock Module+ Registers

The basic clock module+ registers are listed in Table 5-1.

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

BCSTL1 Basic Clock System Control Register 1

7	6	5	4	3	2	1	0
XT2OFF	XTS	DIVAx		XT5V	RSELx		
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

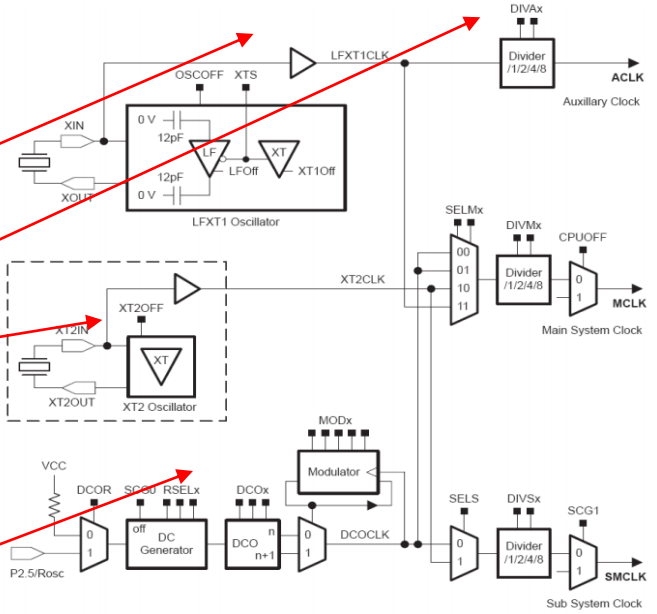
XT2OFF Bit 7 XT2 off. This bit turns off the XT2 oscillator
- 0 XT2 is on
- 1 XT2 is off if it is not used for MCLK or SMCLK.

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

XT5V Bit 3 Unused. XT5V should always be reset.

RSELx Bits 2-0 Resistor Select. The internal resistor is selected in eight different steps. The value of the resistor defines the nominal frequency. The lowest nominal frequency is selected by setting RSELx=0.



BCSTL2 Basic Clock System Control Register 2

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

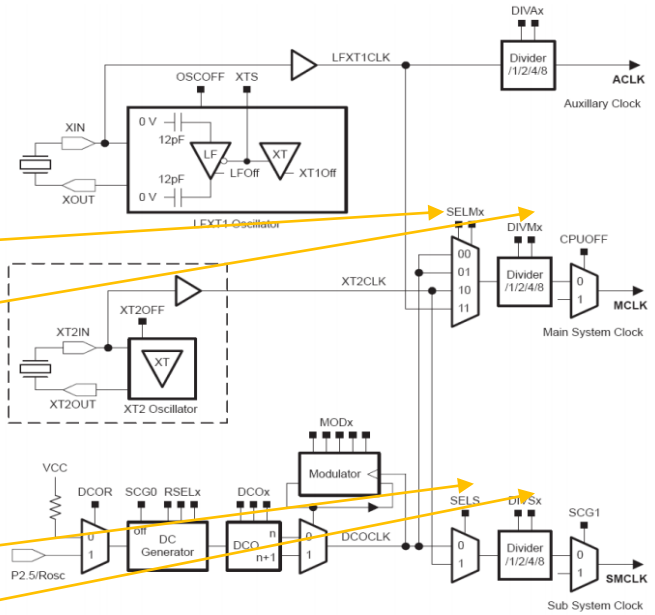
SELMx Bits 7-6 Select MCLK. These bits select the MCLK source.
- 00 DCOCLK
- 01 DCOCLK
- 10 XT2CLK when XT2 oscillator present on-chip. LFXT1CLK when XT2 oscillator not present on-chip.
- 11 LFXT1CLK

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 on-chip. LFXT1CLK when XT2 oscillator not present on-chip.

DIVSx BitS 2-1 Divider for SMCLK
- 00 /1
- 01 /2
- 10 /4
- 11 /8

DCOR Bit 0 DCO resistor select
- 0 Internal resistor
- 1 External resistor



Summary of Clocks in MSP

Table 6.4 Summary of clock configurations for MSP430 generations x1xx through x6xx

MSP430	Name	Oscillators	Clock signals	Max freq. (MHz)	DCO type
x3xx	FLL	XTAL, DCO	MCLK, ACLK,XBUF	0.032	FLL
x4xx	FLL+	LFXT1, DCO, XT2	MCLK, SMCLK, ACLK, ACLK/n	8.000	FLL
x1xx	BCM	LFXT1, DCO, XT2	MCLK, ACLK, SMCLK	8.000	DCO
x2xx	BCM+	LFXT1, DCO, XT2, VLO	MCLK, ACLK, SMCLK	16.000	DCO
x5xx x6xx	UCS	LFXT1, DCO, XT2, REFO, VLO, MODOSC	MCLK, ACLK SMCLK	32.000	FLL/DCO

Use Predefined Constants in code

```
#define SELM0 (0x40) /* MCLK Source Select 0 */
#define SELM1 (0x80) /* MCLK Source Select 1 */
#define DIVS_0 (0x00) /* SMCLK Divider 0: /1 */
#define DIVS_1 (0x02) /* SMCLK Divider 1: /2 */
#define DIVS_2 (0x04) /* SMCLK Divider 2: /4 */
#define DIVS_3 (0x06) /* SMCLK Divider 3: /8 */
#define DIVM_0 (0x00) /* MCLK Divider 0: /1 */
#define DIVM_1 (0x10) /* MCLK Divider 1: /2 */
#define DIVM_2 (0x20) /* MCLK Divider 2: /4 */
#define DIVM_3 (0x30) /* MCLK Divider 3: /8 */
#define SELM_0 (0x00) /* MCLK Source Select 0: DCOCLK */
#define SELM_1 (0x40) /* MCLK Source Select 1: DCOCLK */
#define SELM_2 (0x80) /* MCLK Source Select 2: XT2CLK/LFXTCLK */
#define SELM_3 (0xC0) /* MCLK Source Select 3: LFXTCLK */
```

```
/* Basic Clock System Control 2
 * SELM_0 -- DCOCLK
 * DIVM_0 -- Divide by 1
 * ~SELS -- DCOCLK
 * DIVS_0 -- Divide by 1
 * ~DCOR -- DCO uses internal resistor */
 BCSCTL2 = SELM_0 + DIVM_0 + DIVS_0;
```

```
/* Follow recommended flow. First, clear all DCOx and MODx bits. Then
 * apply new RSELx values. Finally, apply new DCOx and MODx bit values. */
 DCOCTL = 0x00;
 BCSCTL1 = CALBC1_16MHZ; /* Set DCO to 16MHz */
 DCOCTL = CALDCO_16MHZ;

 /* Basic Clock System Control 1
 * XT2OFF -- Disable XT2CLK
 * ~XTS -- Low Frequency
 * DIVA_0 -- Divide by 1 */
 BCSCTL1 |= XT2OFF + DIVA_0;
 /* Basic Clock System Control 3
 * XT2S_0 -- 0.4 - 1 MHz
 * LFXT1S_2 -- If XTS = 0, XT1 = VLOCLK ; If XTS = 1, XT1 = 3 - 16-MHz crystal or resonator
 * XCAP_1 -- ~6 pF */
 BCSCTL3 = XT2S_0 + LFXT1S_2 + XCAP_1;
```

Power-On Reset

• Loads the program counter register (PC) with the address of the first instruction to be executed. This causes execution of the first instruction in the booting sequence.
• Disables the reception of maskable interrupts by the CPU.
• Clears the status register (SR). The specific value loaded into the SR changes from one processor or MCU to another.
• Initializes some or all system peripherals (list changes for specific devices). For example many MCUs set all their I/O pins to input mode, timers are initialized to zero, and the default CPU operating mode is selected.
• Cancels any bus transaction in progress and returning control to the default bus master

The RESET signal in an embedded system is generated through a specialized circuit called a *power-on reset circuit* or POR for short.

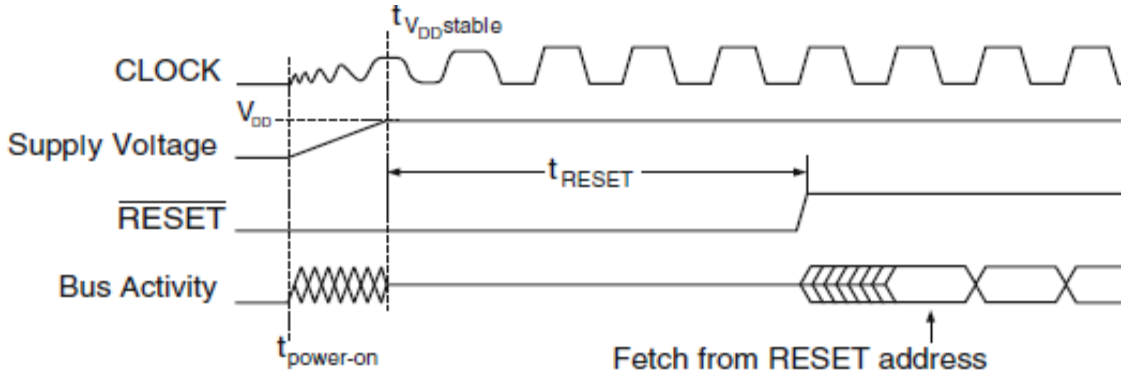


Fig. 6.24 Typical CPU RESET timing

Power-On Reset

$$t_{reset} = RC \cdot \ln\left(\frac{V_{DD}}{V_{DD} - V_T^+}\right)$$

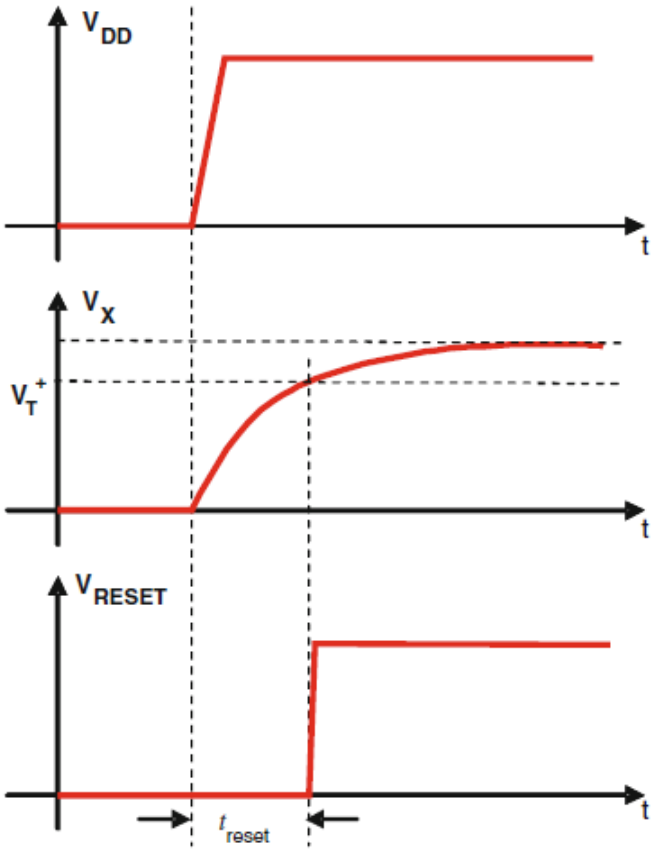
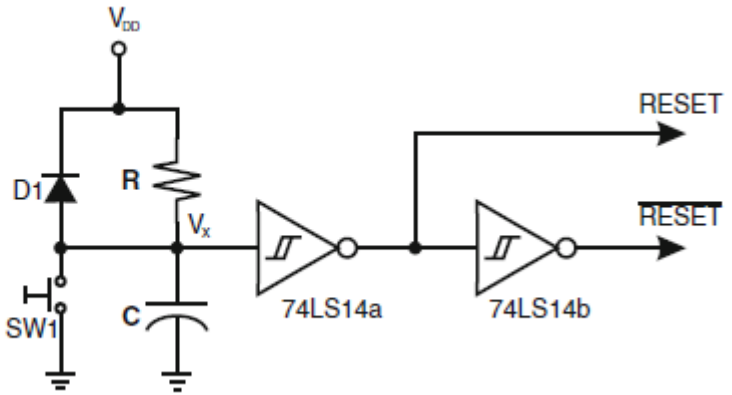
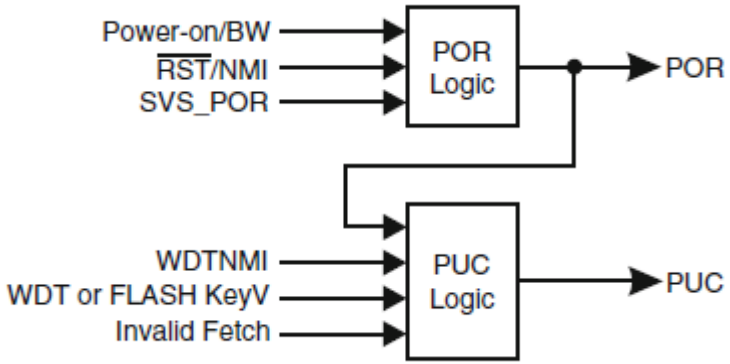


Fig. 6.25 A POR circuit based on a one-shot topology





After a POR

After a POR, the initial conditions in an MSP430 device are the following:

- The functionality of the *RST/NMI* pin is set to ***RST***.
- GPIO pins in all ports are configured as inputs.
- The processor status register (SR) is loaded with the reset value, which clears the V, N, Z, and C flags, disables all maskable interrupts (GIE = 0), and the CPU is set to active mode, cancelling any low-power mode previously set.
- The watchdog timer is activated in watchdog mode.
- The program counter (PC) is loaded with the address pointed by the reset vector (0FFFEh). Setting the reset vector contents to 0FFFFh, disables the device, entering a low power mode.
- Particular peripheral modules and registers are also affected, depending on the specific device being used.

The MSP430 System Clock

- Loops are OK up to a point but timers are more precise and leave the CPU free for more productive activities. Alternatively, the device can be put into a low-power mode if there is nothing else to be done
  - **Watchdog timer:** Included in all devices (newer ones have the enhanced watchdog timer+). Its main function is to protect the system against malfunctions but it can instead be used as an interval timer if this protection is not needed.
  - **Basic timer1:** Present in the MSP430x4xx family only. It provides the clock for the LCD and acts as an interval timer. Newer devices have the LCD_A controller, which contains its own clock generator and frees the basic timer from this task.
  - **Timer_A:** Provided in all devices. It typically has three channels and is much more versatile than the simpler timers just listed. Timer_A can handle external inputs and outputs directly to measure frequency, time-stamp inputs, and drive outputs at precisely specified times, either once or periodically. There are internal connections to other modules so that it can measure the duration of a signal from the comparator, for instance. **It can also generate interrupts.**
  - **Timer_B:** Included in larger devices of all families. It is similar to Timer_A with some extensions that make it more suitable for driving outputs such as **pulse-width modulation**. Against this, it lacks a feature of sampling inputs in Timer_A that is useful in communication.
  - **Real-time clock:** In which the basic timer has been extended to provide a real-time clock in the most recent MSP430x4xx devices.

Watchdog Timer

- The main purpose of the watchdog timer is to protect the system against failure of the software, such as the program becoming trapped in an unintended, infinite loop.
- Left to itself, the watchdog counts up and resets the MSP430 when it reaches its limit. The code must therefore keep clearing the counter before the limit is reached to prevent a reset.

Watchdog Timer

- The operation of the watchdog is controlled by the 16-bit register WDTCTL. It is guarded against accidental writes by requiring the password WDTPW = 0x5A in the upper byte.
- A reset will occur if a value with an incorrect password is written to WDTCTL. This can be done deliberately if you need to reset the chip from software.
- Reading WDTCTL returns 0x69 in the upper byte, so reading WDTCTL and writing the value back violates the password and causes a reset.

WDTCNT and WDTCTL Registers

- The watchdog counter is a 16-bit register WDTCNT, which is not visible to the user.
- It is clocked from either SMCLK (default) or ACLK, according to the *WDTSSEL* bit in the WDTCTL.
- The period is 64, 512, 8192, or 32,768 (default) times the period of the clock. This is controlled by the *WDTISx* bits in WDTCTL. The intervals are roughly 2ms, 16ms, 250ms, and 1000 ms if the watchdog runs from ACLK at 32 KHz.
- *WDTHOLD = 1 when the WDT+ is not in use conserves power.*

WDTCTL

- The lower byte of WDTCTL contains the bits that control the operation of the watchdog timer,

7	6	5	4	3	2	1	0
WDT-HOLD	WDT-NMIES	WDTNMI	WDT-TMSEL	WDT-CNTCL	WDTSSEL	WDTISx	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	r0(w)	rw–(0)	rw–(0)	rw–(0)

The watchdog is always active after the MSP430 has been reset. By default the clock is SMCLK, which is in turn derived from the DCO at about 1 MHz. The default period of the watchdog is the maximum value of 32,768 counts, which is therefore around 32 ms. You must clear, stop, or reconfigure the watchdog before this time has elapsed. Stopping the watchdog, means setting the WDTHOLD bit.

WDTCTL

WDTISx (Bits 1-0) Watchdog timer interval select. These bits select the watchdog

timer interval to set the WDTIFG flag and/or generate a PUC. The alternatives are

- WDTIS0 for 00: Watchdog clock source / 32768 (Default)
- WDTIS1 for 01: Watchdog clock source / 8192
- WDTIS2 for 10: Watchdog clock source / 512
- WDTIS3 for 11: Watchdog clock source / 64

Some useful examples of instructions associated to the WDT configuration are the following.

```
mov #WDTPW+WDTHOLD,&WDTCTL ; To stop WDT
mov #WDTPW+WDTCNTCL,&WDTCTL ;Reset WDT
mov #WDTPW+WDTCNTCL+WDTSSEL,&WDTCTL ; select ACLK clock
;WDT interval timer mode with ACLK, and interval clock/512
mov #WDTPW+WDTCNTCL+WDTMSEL+WDTIS2,&WDTCTL
```

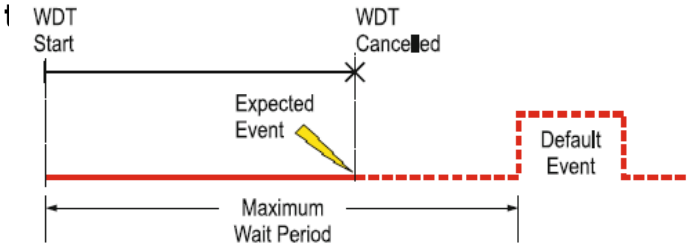
WDTCTL

- If the watchdog is left running, the counter must be repeatedly cleared to prevent it counting up as far as its limit. This is done by setting the *WDTCNTCL (count clear)* bit in WDTCTL.
- As a result of reaching the limit, the watchdog timer sets the *WDTIFG* flag in the special function register IFG1

Example watchdog application

- The clock is selected from ACLK (WDTSSEL = 1) and the longest period (WDTISx = 00), which gives 1s with a 32 KHz crystal for ACLK.
- It is wise to restart any timer whenever its configuration is changed so the counter is cleared by setting the WDTCNTCL bit. LED1 shows the state of button B1 and LED2 shows WDTIFG.
- The watchdog is serviced by rewriting the configuration value in a loop while button B1 is held down. If the button is left up for more than 1s the watchdog times out, raises the flag WDTIFG, and resets the device with a PUC.
- This is shown by LED2 lighting.

Listing 8.1: Program wdtest1.c to demonstrate the watchdog t



7.17 Watchdog timer cancellation

```
// wdtest1.c - trival program to demonstrate watchdog timer
// Olimex 1121 STK board , 32KHz ACLK
// -----------------------------------------------------------------------
#include <io430x11x1.h> // Specific device
// -----------------------------------------------------------------------
// Pins for LEDs and button
#define LED1 P2OUT_bit.P2OUT_3
#define LED2 P2OUT_bit.P2OUT_4
#define B1 P2IN_bit.P2IN_1
// Watchdog config: active , ACLK /32768 -> 1s interval; clear counter
#define WDTCONFIG (WDTCNTCL|WDTSSEL) // Include settings for _RST/NMI pin here as well
// Setting WDTCNTCL = 1 clears the count value to 0000h.
void main (void)
{ //WDTPW = 0x5A00 or 01011010 00000000
//WDTSSEL Bit 2 Watchdog timer+ clock source select
      WDTCTL = WDTPW | WDTCONFIG; // Configure and clear watchdog
      P2DIR = BIT3 | BIT4; // Set pins with LEDs to output
      P2OUT = BIT3 | BIT4; // LEDs off (active low)
      for (;;) { // Loop forever
            LED2 = ~IFG1_bit.WDTIFG; // LED2 shows state of WDTIFG
            if (B1 == 1) { // Button up
                  LED1 = 1; // LED1 off
            } else { // Button down
                  WDTCTL = WDTPW | WDTCONFIG; // Feed/pet/kick/clear watchdog
                  LED1 = 0; // LED1 on
            }
      }
}
```

Watchdog as an Interval Timer

- The watchdog can be used as an interval timer if its protective function is not desired.
- Set the WDTTMSEL bit in WDTCTL for interval timer mode. The periods are the same as before and again WDTIFG is set when the timer reaches its limit, but no reset occurs!
- The counter rolls over and restarts from 0. An interrupt is requested if the WDTIE bit in the special function register IE1 is set. This interrupt is maskable and as usual takes effect only if GIE is also set.
- The watchdog timer has its own interrupt vector, which is fairly high in priority but not at the top. It is not the same as the reset vector, which is taken if the counter times out in watchdog mode.
- The WDTIFG flag is automatically cleared when the interrupt is serviced.

TIMERS

The MSP430 family supports three timers.

- Timer_A, present in all models;
- Timer_B included in all but the legacy 3xx series; and
- Timer_D, appearing in the 5xx/6xx series.

Any timer can be used for applications such as real-time clock, pulse width modulation, or baud rate generation, among others

Timer_A

- This is the most versatile, general-purpose timer in the MSP430 and is included in all devices.
- There are two main parts to the hardware:
  - **Timer block:** The core, based on the 16-bit register TAR. There is a choice of sources for the clock, whose frequency can be divided down (prescaled). The timer block has no output but a flag TAIFG is raised when the counter returns to 0.
  - **Capture/compare channels:** In which most events occur, each of which is based on a register TACCRn. They all work in the same way with the important exception of TACCR0. Each channel can
    - **Record the "time"** (the value in TAR) at which the input changes in TACCRn; the input can be either external or internal from another peripheral or software.
    - **Compare** the current value of TAR with the value stored in TACCRn and update an output when they match; the output can again be either external or internal.
    - **Request an interrupt** by setting its flag TACCRn CCIFG on either of these events; this can be done even if no output signal is produced.
    - **Sample** an input at a compare event; this special feature is particularly useful if Timer_A is used for serial communication in a device that lacks a dedicated interface.

Configuring the Timer

- **TACTL – Timer A Control Register**

This register used to configure how the timer runs

- **TACCTL0 – Capture/Compare Control Register**

For enabling and disabling TimerA0 interrupt

- **TACCR0 – Capture/Compare Register**

This register holds the value YOU define to configure the timing

Timer A Block Diagram

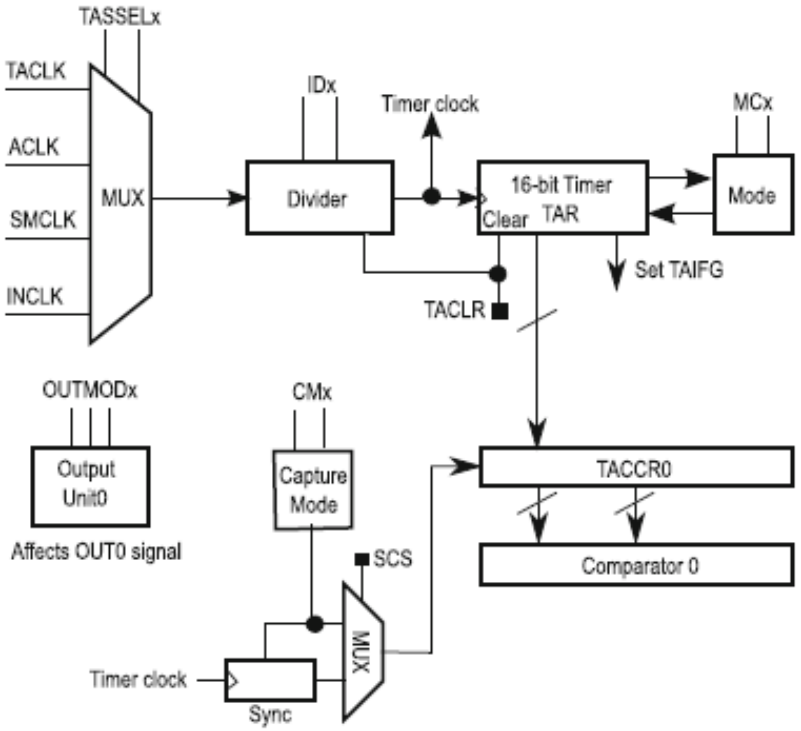
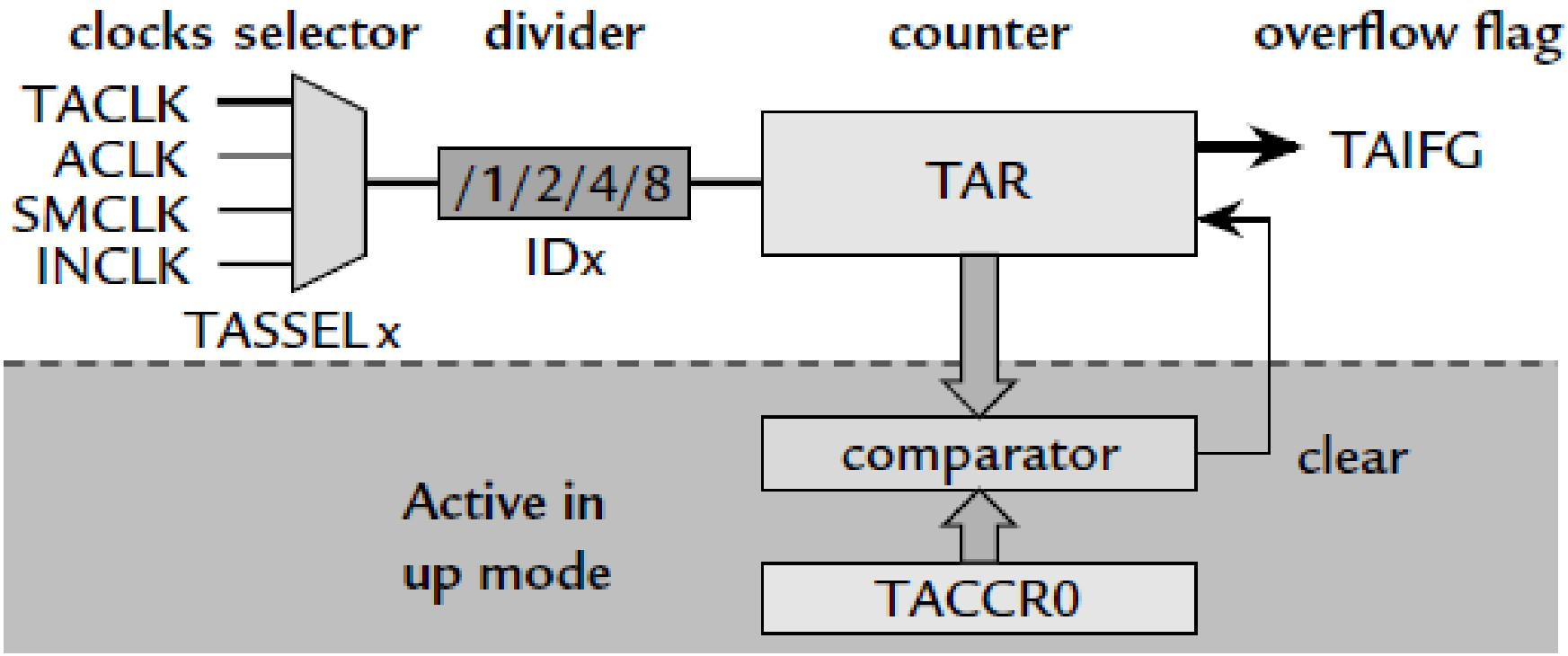


Fig. 7.22 Simplified MSP430 Timer_A block diagram

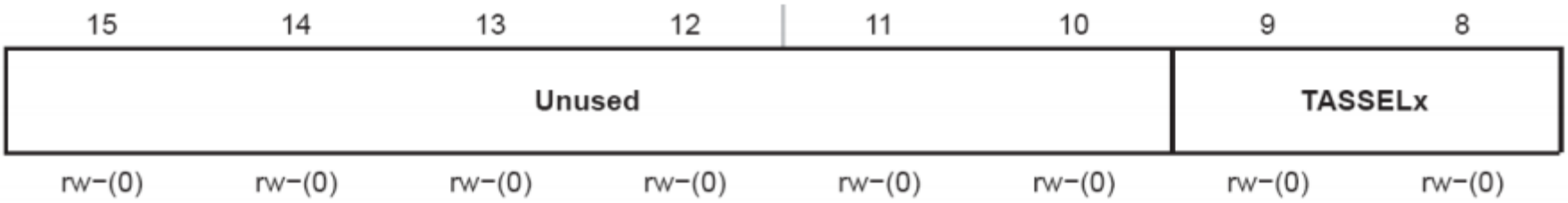
Table 7.4 Timer_A implementation features

TIMER_A	x3xx	x4xx	x1xx	x2xx	x5xx/x6xx
16-bit timer/counter w/4 operating modes	Yes	Yes	Yes	Yes	Yes
Asynchronous	No	Yes	Yes	Yes	Yes
Selectable and configurable clock source	Yes	Yes	Yes	Yes	Yes
Independently configurable CCRs	5	3 or 5	3	2 or 3	up to 7
PWM output capability	No	Yes	Yes	Yes	Yes
Asynchronous I/O latching	No	Yes	Yes	Yes	Yes
Interrupt vector register	No	Yes	Yes	Yes	Yes
Second Timer_A	No	Yes	No	No	No



Timer_A Registers

- **TACTL, Timer_A Control Register (PART 1)**



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

TACTL, Timer_A Control Register (PART 2)

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

- Three items are given for each bit:
  - Its position in the word, which should not be needed (use symbolic names instead).
  - Its name, which is defined in the header file and should be known to the debugger; some bits are not used, which we show by a gray fill.
  - The accessibility and initial condition of the bit; here they can all be read and written with the exception of TACLR, where the missing r indicates that there is no meaningful value to read. The (0) shows that each bit is cleared after a power-on reset (POR).

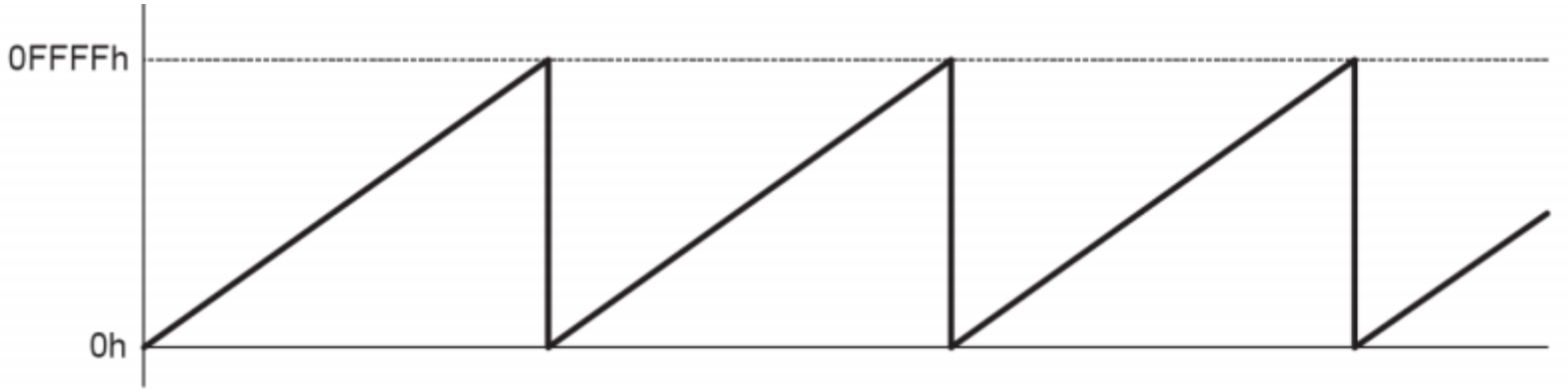
The user's guide goes on to describe the function of each bit or group of bits:

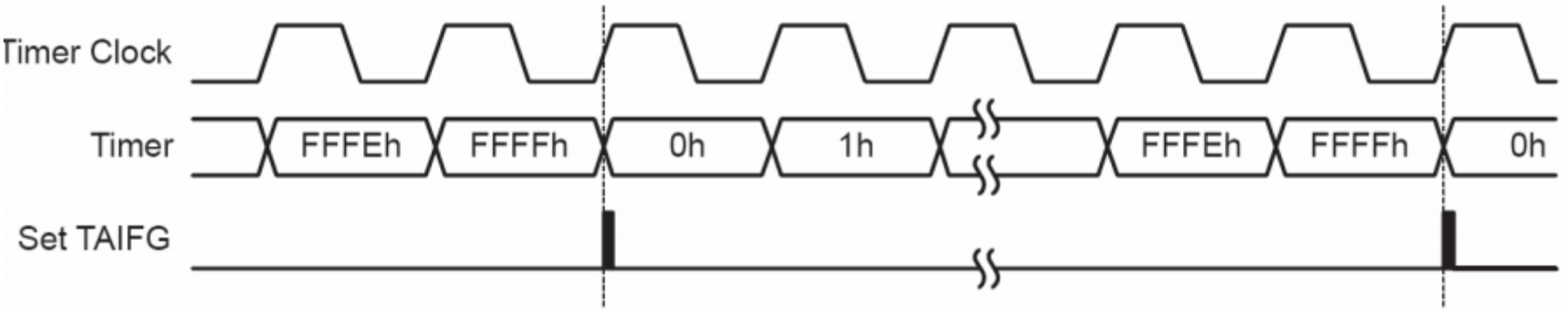
- **Timer_A clock source select, TASSELx:** There are four options for the clock: the internal SMCLK or ACLK or two external sources. TACLK (00), ACLK (01), SMCLK (10), or INCLK (11)
- **Input divider, IDx:** The frequency of the clock can be divided before it is applied to the timer, which extends the period of the counter. IDx bits determine the frequency division factor in the prescaler: 1 (00), 2 (01), 4 (10), and 8 (11)
- **Mode control, MCx:** The timer has four modes. By default it is off to save power. MCx bits set the operation mode: Halt (00), up mode (01), continuous mode (10), and up/down mode (11).
- **Timer_A clear, TACLR:** Setting this bit clears the counter, the divider, and the direction of the count (it can go both up and down in up/down mode). The bit is automatically cleared by the timer after use. It is usually a good idea to clear the counter whenever the timer is reconfigured to ensure that the first period has the expected duration.
- **Timer_A interrupt enable, TAIE:** Setting this bit enables interrupts when TAIFG becomes set.We do not use this here.
- **Timer_A interrupt flag, TAIFG:** This bit can be modified by the timer itself or by a program. It is raised (set) by the timer when the counter becomes 0. In continuous mode this happens when the value in TAR rolls over from 0xFFFF to 0x0000. An interrupt is also requested if TAIE has been set. The program must clear TAIFG so that the next overflow can be distinguished.

Continuous Mode

- Timer counts from 0 to 0xFFFF
- Fewer timing errors because timer never stops – keeps counting up until it reaches 0xFFFF and rolls over to 0 and keeps going.

Modes of Operation: Continuous Mode

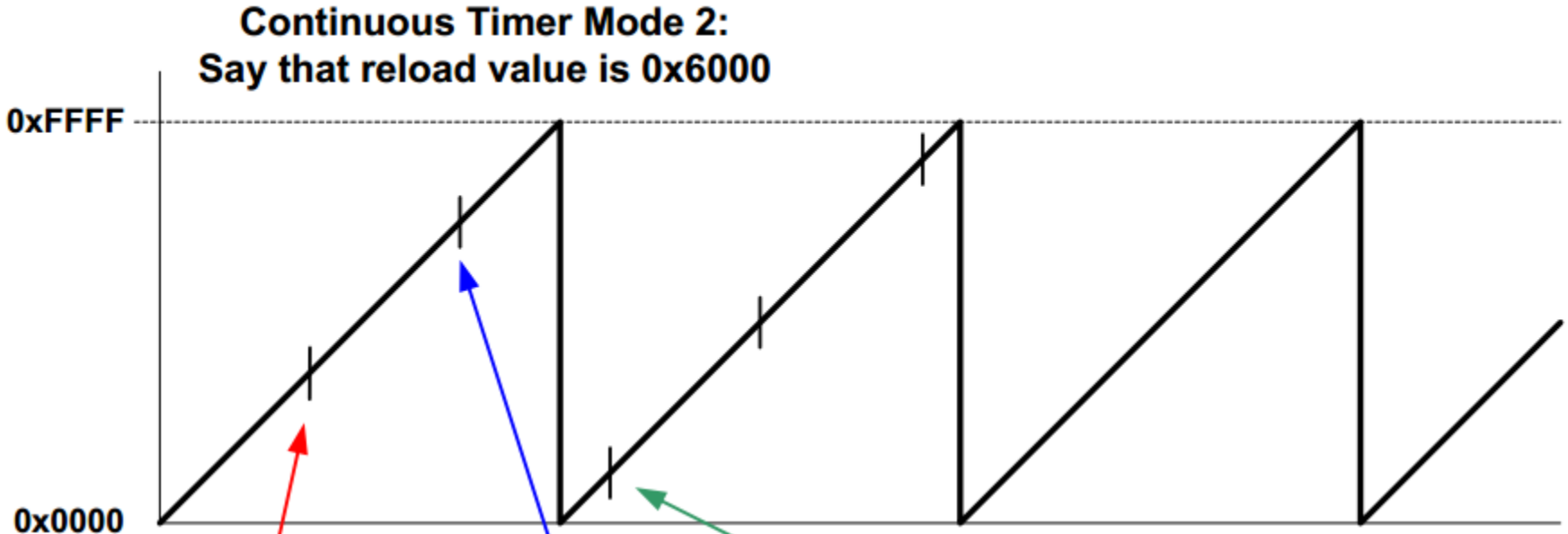




Continuous Mode

- **If we have a period value in TACCR0**
  - The ACTUAL VALUE of the timer does not matter – only the RELOAD VALUE matters – this controls the period of the interrupt.
  - Interrupt DOES NOT OCCUR AT 0 OR 0xFFFF!
  - Occurs when timer reaches current TACCR0 value!

Continuous Mode



1) Timer reaches 0x6000, fires interrupt, inside ISR we ADD the period to the CURRENT VALUE of the timer:
TACCR0 += 0x6000
Therefore TACCR0 = 0xC000, and next interrupt will fire when timer reaches this value.

2) Timer now reaches 0xC000, fires interrupt, inside interrupt
TACCR0 += 0x6000
New TACCR0 = 0x2000 (rolls over 0xFFFF)

3) Timer rolls over 0xFF and now reaches 0x2000. Cycle repeats.

Cont. Mode example

- The sub-main clock SMCLK runs at the same speed as MCLK by default, which is 800 KHz for example.
- If this were used to clock the timer directly, the period would be=2^16/800KHz ≈ 0.08 s.
- We want about 0.5 s and therefore divide the frequency of the clock by 8 using IDx. IDx = 8 (11) This gives a delay of about 0.64 s, close enough.
- We use the simplest Continuous mode, in which TAR simply counts up through its full range of 0x0000–0xFFFF and repeats. This needs MCx = 10.

Cont. Mode example

```
// timrled1.c - toggles LEDs with period of about 1.3s
// Poll free -running timer A with period of about 0.65s
// Timer clock is SMCLK divided by 8, continuous mode
// Olimex 1121STK , LED1 ,2 active low on P2.3,4
#include <io430x11x1.h> // Specific device
// Pins for LEDs
#define LED1 BIT3
#define LED2 BIT4
void main (void)
{
    WDTCTL = WDTPW|WDTHOLD; // Stop watchdog timer
    P2OUT = ~LED1; // Preload LED1 on , LED2 off
    P2DIR = LED1|LED2; // Set pins for LED1 ,2 to output
    TACTL = MC_2|ID_3|TASSEL_2|TACLR; // Set up and start Timer A
    // Continuous up mode , divide clock by 8, clock from SMCLK , clear timer
    for (;;) { // Loop forever
        while (TACTL_bit.TAIFG == 0) { // Wait for overflow
        } // doing nothing
        TACTL_bit.TAIFG = 0; // Clear overflow flag
        P2OUT ^= LED1|LED2; // Toggle LEDs
    } // Back around infinite loop
}
```

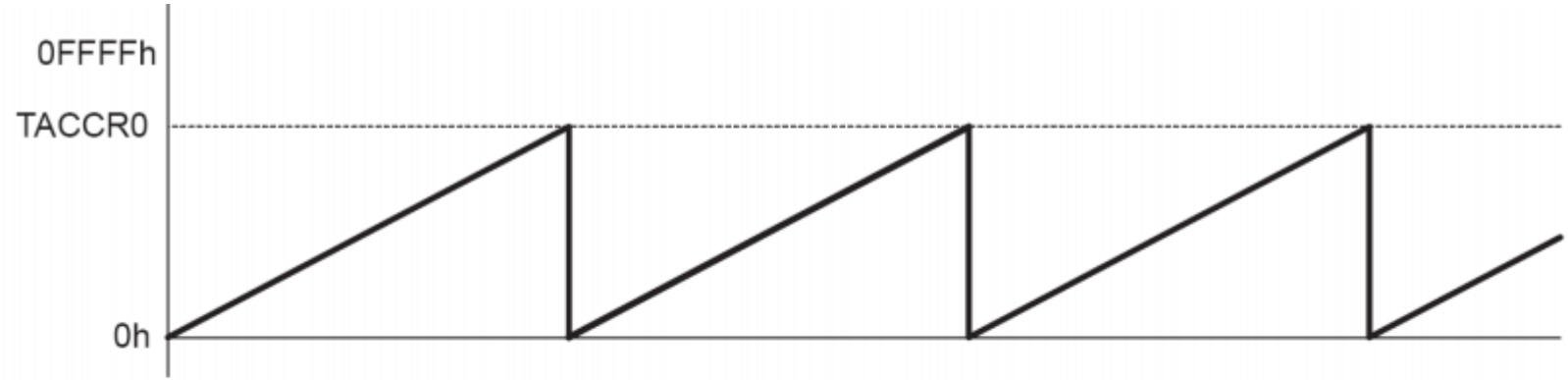
Cont. Mode example

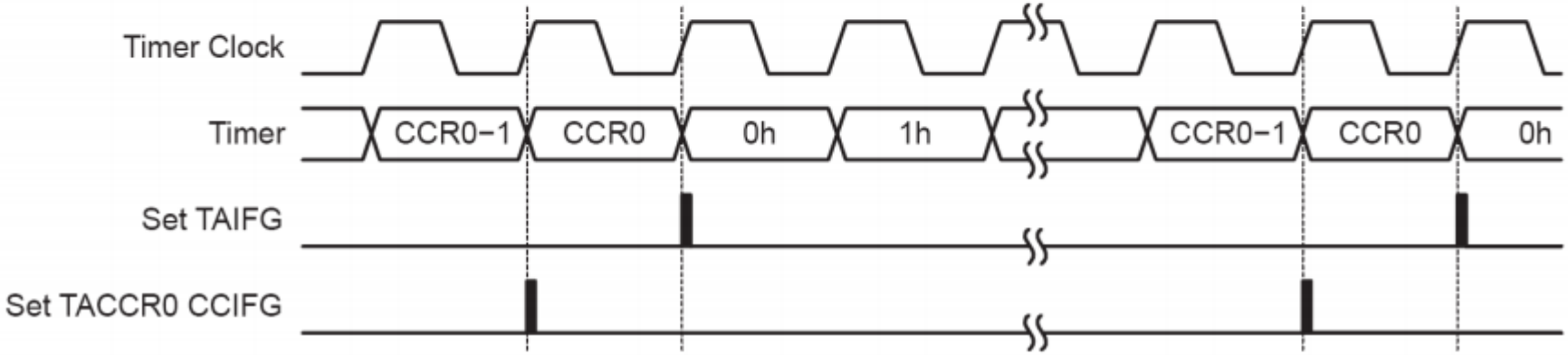
- More tasks could be added here, provided that they do not take longer than the period of the timer. The result is a *paced loop*, a straightforward structure for a program that carries out a sequence of tasks at regular intervals.
- Nowadays it would be unusual to pace the loop by polling the timer; instead the MCU would save energy by entering a low-power mode after it had completed the tasks and wait for the timer to wake it again.

Modes of Operation: Up Mode

- Timer counts UP from zero to TACCRO
- Interrupt occurs when timer goes back to zero
- Useful for periods other than 0xFFFF

Modes of Operation: Up Mode





Timer_A in Up Mode

- Finer control over the delay is obtained by using the timer in Up mode rather than continuous mode. The maximum desired value of the count is programmed into another register, TACCR0. In this mode 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
- Here the clock has been divided down to 100 KHz so we need 50,000 counts for a delay of 0.5 s and should therefore 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
```

Random Light Display

- A pretty application of the delay is a random light show on the LEDs. Of course this is rather limited with only two LEDs but the principle can be applied to bigger displays. This again uses a delay set by the timer but requires a calculation for the next pattern to display

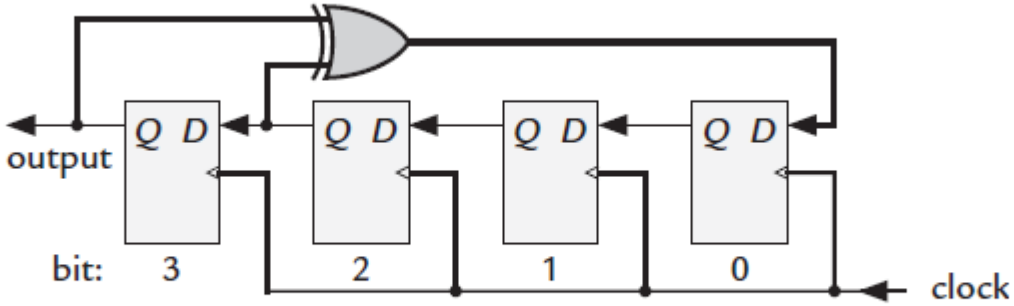


Figure 4.10: A shift register with feedback through an exclusive-OR gate from the last two stages, used to generate a pseudorandom stream of bits.

- The circuit without the exclusive-OR gate and its connections is a plain *shift register*. A *D* flip-flop simply reads the value on its *D* input at a clock transition and transfers it to its *Q* output. Thus the value in flip-flop 0 is transferred to flip-flop 1 after a clock transition.
- At the same time the value in flip-flop 1 is transferred to flip-flop 2 and so on. The pattern of bits simply shifts one place to the left in each clock cycle. An input is applied to the first flip-flop, 0.
- A pseudorandom sequence requires more complicated feedback. The simplest method, shown in the figure, is to take the feedback from an exclusive-OR gate connected to the outputs of the last two stages.
- The counter must therefore be "seeded" with a nonzero value. The counter in
- Figure with $N = 4$ gives the sequence 0001, 0010, 0100, 1001, 0011, 0110, 1101, 1010, 0101, 1011, 0111, 1111, 1110, 1100, 1000 and repeat

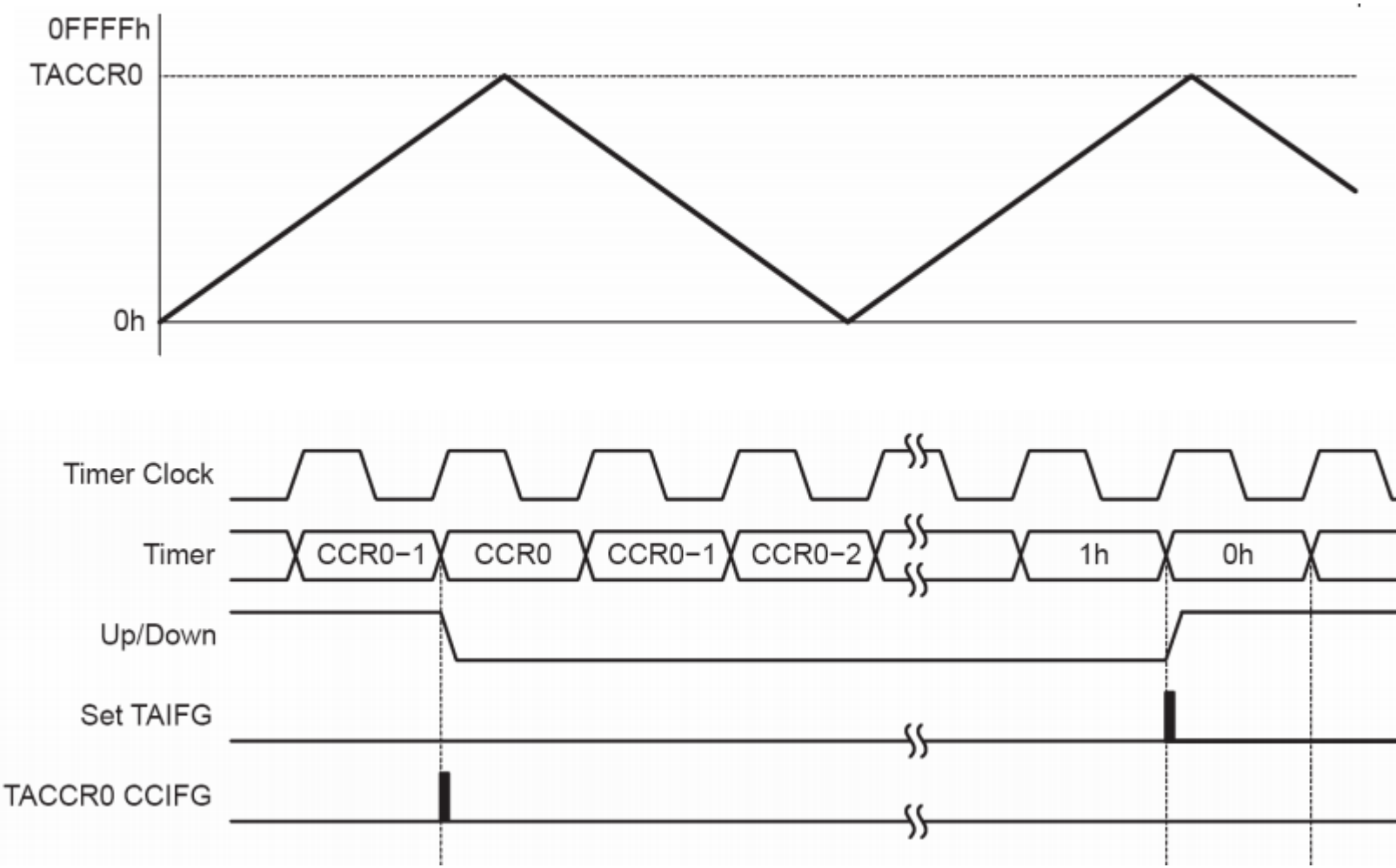
Program to produce a pseudorandom bit sequence by simulating a shift register with feedback.

```
// random1.c - pseudorandom sequence on LEDs  Poll timer A in Up mode with period of about 0.5s
// Timer clock is SMCLK divided by 8, up mode , p eriod 50000  Olimex 1121STK , LED1 ,2 active low on P2.3,4
// -----------------------------------------------------------------------
#include <io430x11x1.h> // Specific device
#include <stdint.h> // For uint16_t
#define LED1 BIT3// Pins for LEDs
#define LED2 BIT4
// Parameters for shift register; length <= 15 (4 is good for testing)
#define REGLENGTH 15
#define LASTMASK (( uint16_t) (BIT0 << REGLENGTH ))
#define NEXTMASK (( uint16_t) (BIT0 << (REGLENGTH -1)))
void main (void)
{
        WDTCTL = WDTPW|WDTHOLD; // Stop watchdog timer
        P2OUT = LED1|LED2; // Preload LEDs off
        P2DIR = LED1|LED2; // Set pins with LEDs to output
        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
        pattern = 1;
        for (;;) { // Loop forever
                while (TACTL_bit.TAIFG == 0) { // Wait for timer to overflow
                } // doing nothing
                TACTL_bit.TAIFG = 0; // Clear overflow flag
                P2OUT = pattern; // Update pattern (lower byte)
                pattern <<= 1; // Shift for next pattern
                // Mask two most significant bits , simulate XOR using switch , feed back
                switch (pattern & (LASTMASK|NEXTMASK )) {
                        case LASTMASK:
                        case NEXTMASK:
                        pattern |= BIT0; // XOR gives 1
                        break;
                        default:
                        pattern &= ~BIT0; // XOR gives 0
                        break;
                }
        } // Back around infinite loop
}
```

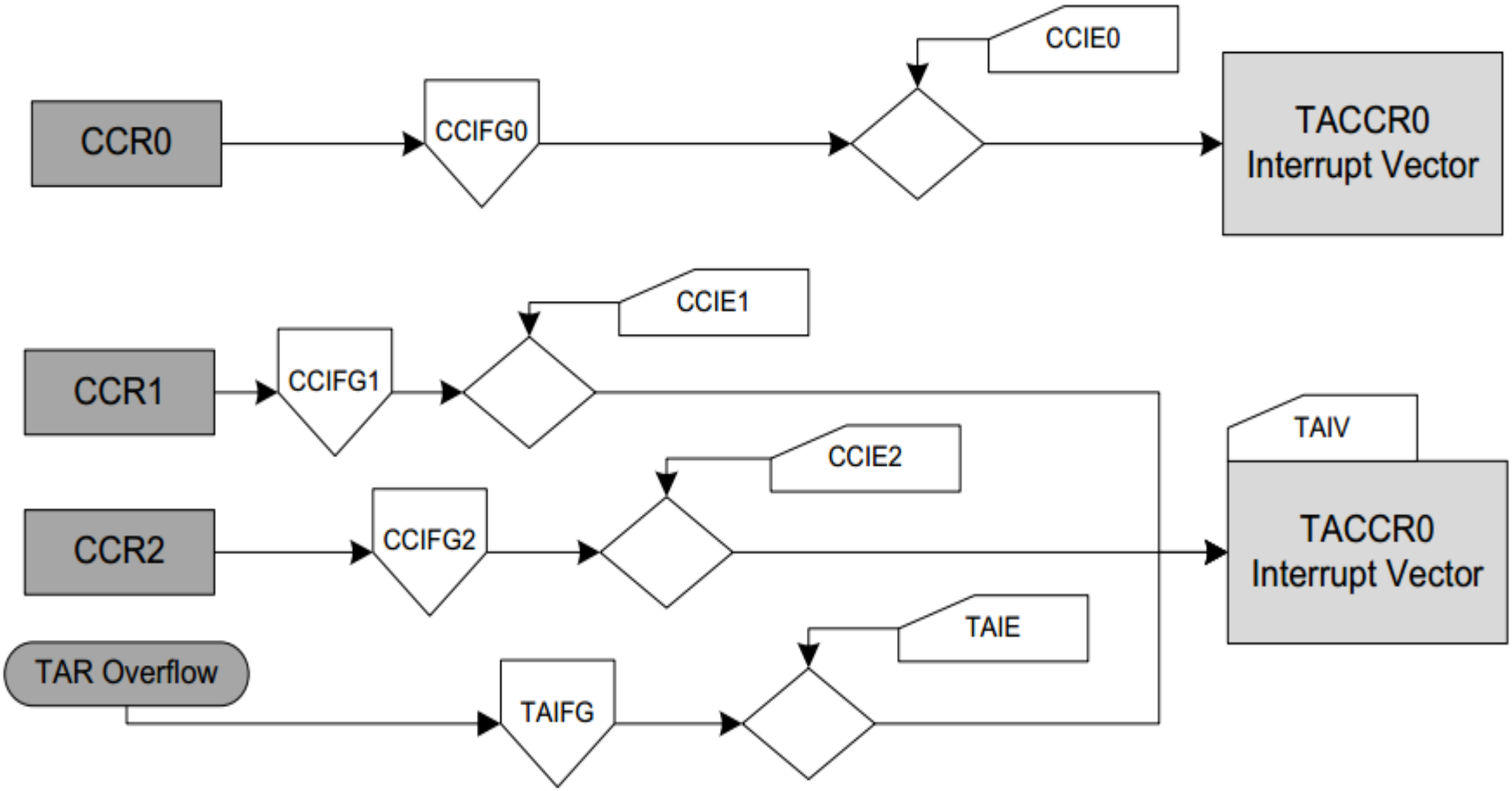
Modes of operation: Up Down mode

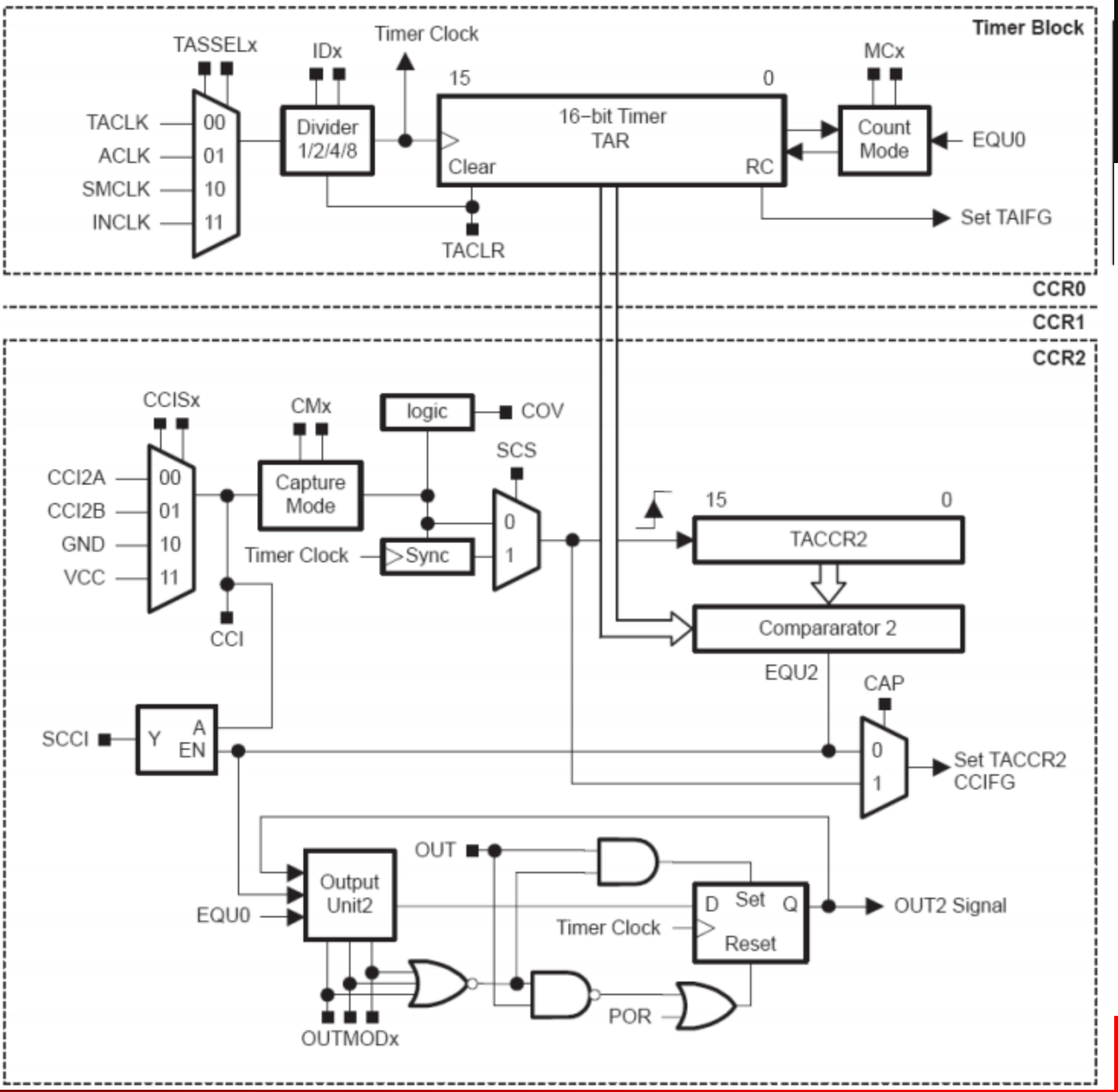
- Timer counts from 0 to TACCRO, then back down to 0
- Used when timer period must be different from 0xFFFF and when pulse needs to be symmetric
- Good for driving motors (ON pulse to control speed)

Modes of operation: Up Down mode

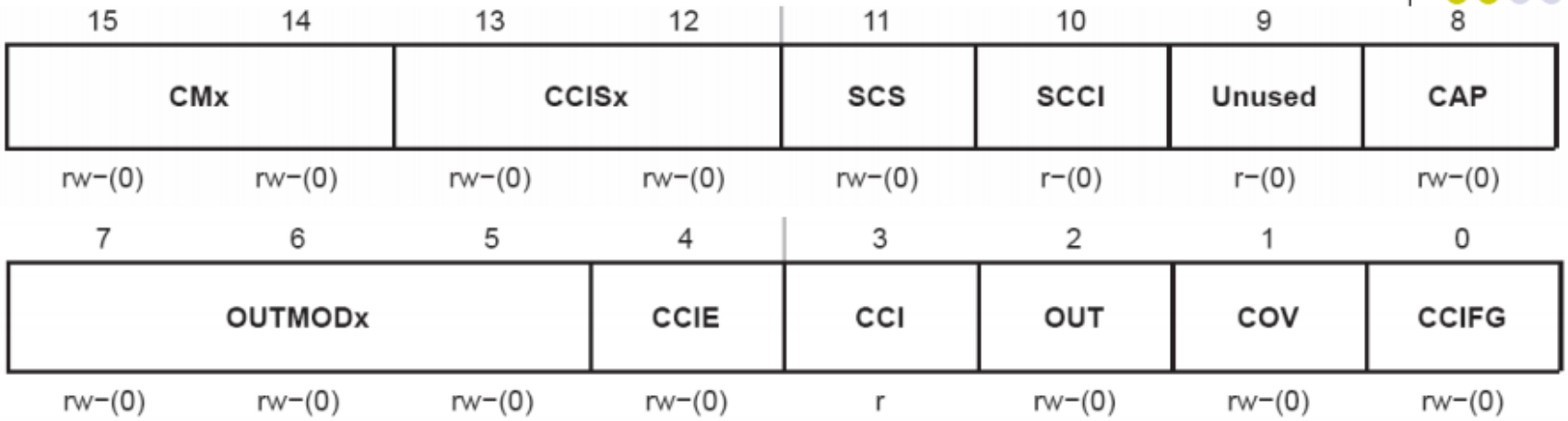


Timer_A Interrupt Vectors




Timer Block
TASSELx
IDx
Timer Clock
MCx
15
0
TACLK
ACLK
SMCLK
INCLK
00
01
10
11
Divider
1/2/4/8
16–bit Timer
TAR
Clear
RC
Count
Mode
EQU0
Set TAIFG
TACLR
CCR0
CCR1
CCR2
CCISx
CMx
logic
COV
SCS
CCI2A
CCI2B
GND
VCC
Capture
Mode
Timer Clock
Sync
TACCR2
Compararator 2
EQU2
CAP
CCI
SCCI
Y
A
EN
Set TACCR2
CCIFG
OUT
Output
Unit2
EQU0
D
Set
Q
Reset
OUT2 Signal
Timer Clock
POR
OUTMODx

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

Example 1

Continuous Mode

Output pin P6.0 with toggle rate = 32768/(2*50) = 328Hz

```
#include "include/include.h"
#include "include/hardware.h"
void main ( void )
{
      WDTCTL = WDTPW + WDTHOLD; // Stop WDT
      P6DIR |= 0x01; // P6.0 output
      CCTL0 = CCIE; // CCR0 interrupt enabled
      CCR0 = 50;
      TACTL = TASSEL_1 + MC_2; // ACLK, contmode
      eint(); // Enable the global interrupt
      //or _BIS_SR(LPM0_bits + GIE);
      LPM0; // Enter low power mode or wait in a loop
}
// Timer_A TACCR0 interrupt vector handler
interrupt (TIMERA0_VECTOR) TimerA_procedure(void){
          P6OUT ^= 0x01; // Toggle P6.0
          CCR0 += 50; // Add offset to CCR0
}
```

Example 2

- **Up Mode**
- **Output pin P6.0 with toggle rate = 32768/(2*50) = 328Hz**

```
#include "include/include.h"
#include "include/hardware.h"
void main ( void )
{
      WDTCTL = WDTPW + WDTHOLD; // Stop WDT
      P6DIR |= 0x01; // P6.0 output
      CCTL0 = CCIE; // CCR0 interrupt enabled
      CCR0 = 50-1;
      TACTL = TASSEL_1 + MC_1; // ACLK, upmode
      _BIS_SR(LPM0_bits + GIE); // Enable the global interrupt and enter LPM0
}
// Timer_A TACCR0 interrupt vector handler
interrupt (TIMERA0_VECTOR) TimerA_procedure ( void ){
            P6OUT ^= 0x01; // Toggle P6.0
}
```