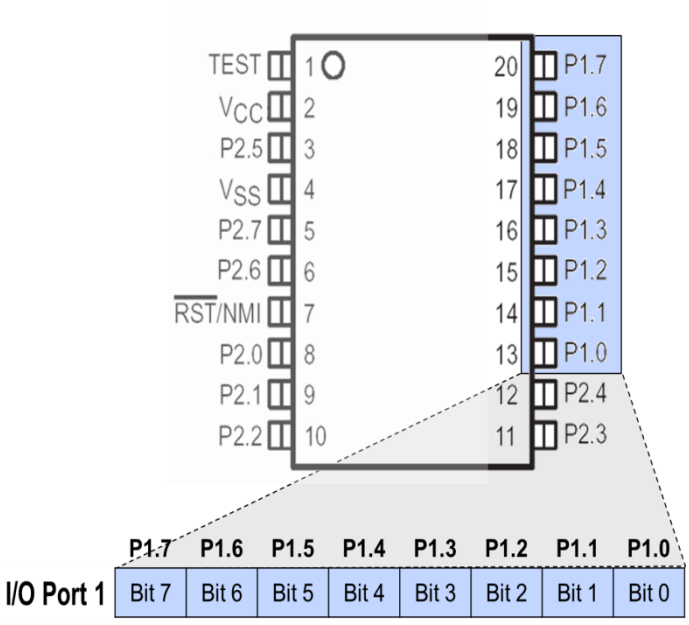
UNIT-IV

I/O Ports Pull Up/Down Registers Concepts:

Digital I/O Ports (GPIO):

Features:

- There are up to eight 8-I/O ports P1- P8 each Port is 8 bit wide
- All individual I/O bits are independently programmable.
- Any combination of input, output, and interrupt conditions is possible.
- Individually configurable pull-up or pull-down resistors
- Read and write access to port-control registers is supported by all instructions.
- Ports can be accessed byte-wise (P1 through P8) or word-wise in pairs (PA through PD).
- Independent input and output data registers
- GPIO = General Purpose Bit Input/Output



- 8-bit I/O ports
- 1 to 8 ports, depending on family and pin-count
- Each pin is individually controllable
- Input pins can generate interrupts
- Controlled by memory-mapped registers:
  - IN
  - OUT
  - DIR
  - REN
  - SEL

  …

I/O PORT REGISTERS:

PORTs are Controlled by memory-mapped registers:

PxIN → Input Register
PxOUT → Output Register
PxDIR → Direction Register
PxREN → Resistor Enable Register
PxSEL → Select Register

Byte Wise Access

PxIN.y
PxOUT.y
PxDIR.y
PxREN.y
PxSEL.y

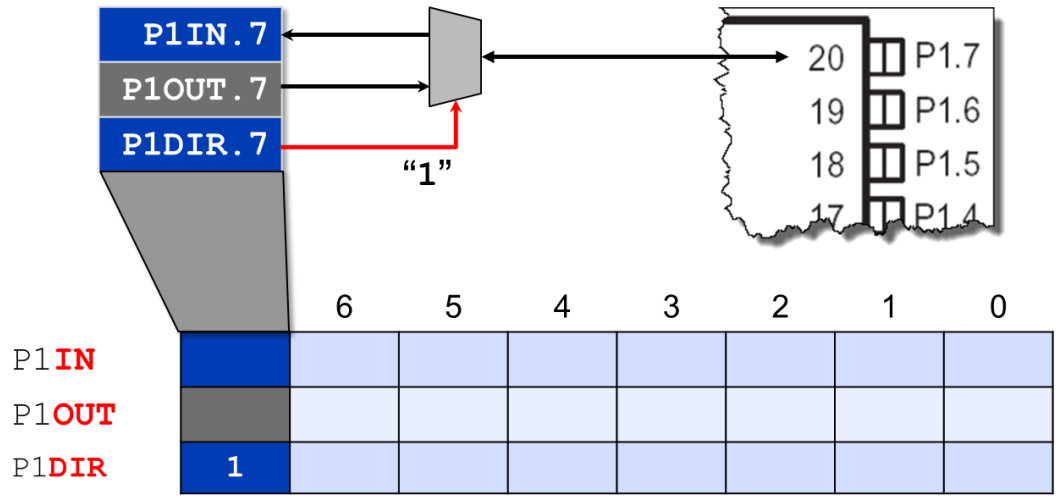
Bit Wise Access

x → Port No.
Y → Port Bit No.

In Short Description:

PxIN.y	Receives the data (1 or 0) from Outside
PxOUT.y	1) Sends the data (1 or 0) to Outside (If the Port Pin is O/P)
	2) Selects the pull-up or pull-down resister (If the Port Pin is I/P and REN Register enabled)
PxDIR.y	Sets the Direction of Port (I/P or O/P)
PxREN.y	Enable or Disable the pull-up/pull-down resister
PxSEL.y	Selects the Alternate function of Port

PxDIR (Pin Direction): Input or Output:



- PxDIR.y: 0 = input, 1 = output
- Register example: `P1DIR |= 0x80;`

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin.

Dual role of Output Registers (PxOUT):

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

PxOUT:

Bit = 0: Output is low

Bit = 1: Output is high

If the pin is configured as I/O function, input direction and the pull-up or pull-down resistor are enabled; the corresponding bit in the PxOUT register selects pull-up or pull-down.

PxOUT:

Bit = 0: Pin is pulled down

Bit = 1: Pin is pulled up

PxREN:

Each bit in each PxREN register enables or disables the pull-up or pull-down resistor of the corresponding I/O pin.

The corresponding bit in the PxOUT register selects if the pin contains a pull-up or pull-down.

Bit = 0: Pull-up or pull-down resistor disabled

Bit = 1: Pull-up or pull-down resistor enabled

I/O Ports pull up/down registers:

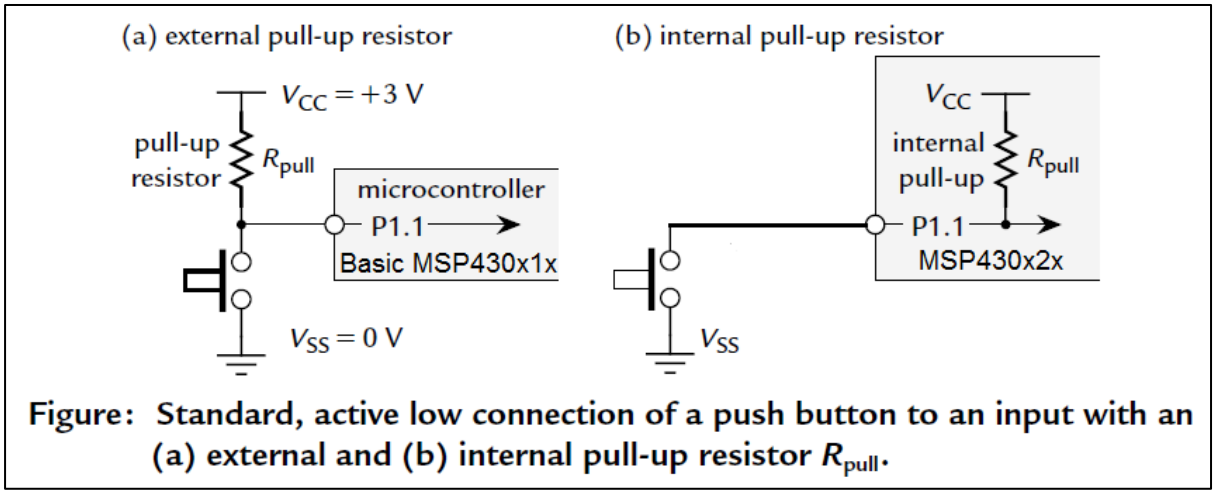
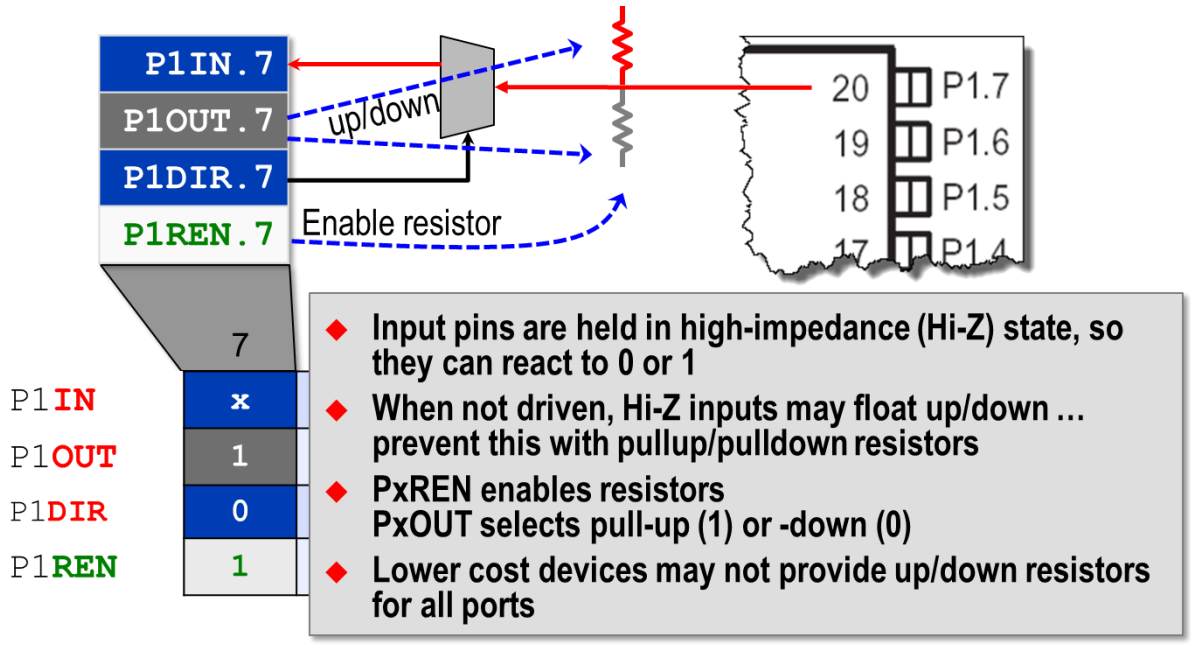
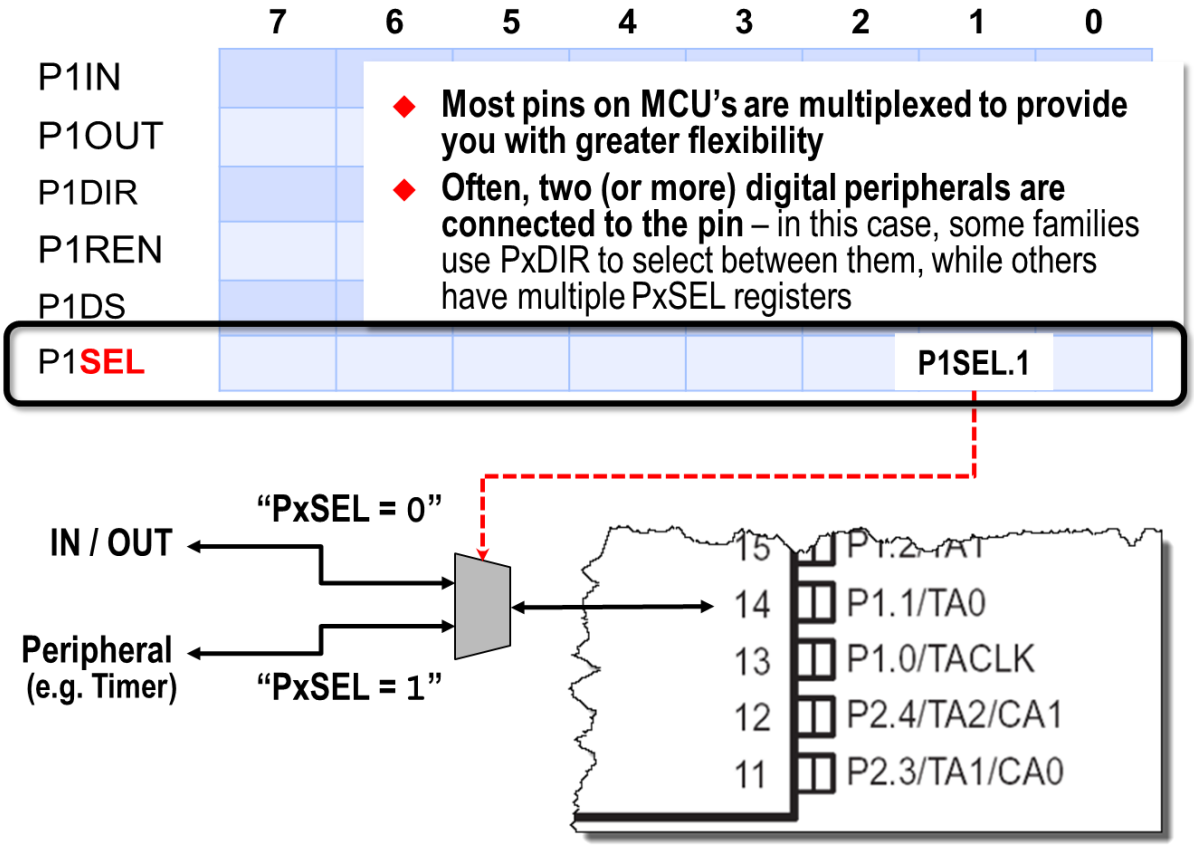


Figure: Standard, active low connection of a push button to an input with an (a) external and (b) internal pull-up resistor R_{pull}.



Pin Flexibility:



Interrupts and Interrupt Programming:

Definition:

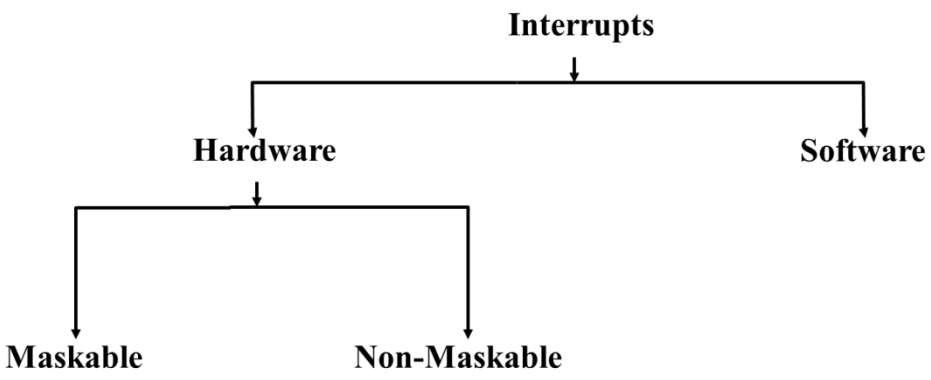
An interrupt is a mechanism by which an I/O device or an instruction can suspend the normal execution of processor and get itself serviced.

(Or)

An interrupt is a signal to the processor generated by hardware or software indicating an event/a task/a device that needs processor service

Generally, a particular task/service routine/small program is assigned to the interrupt signal and this program is called interrupt service routine (ISR).

CLASSIFICATION OF INTERRUPTS



Task Handling:

Event/Task/Service routine can handle by two methods.
They are:

1) Polling method
2) Interrupt method

1) Polling method is continuous checking of condition or status of device by program or device to see what state they are in. 100% CPU Load

2) In interrupt method continuous checking of condition or status is not required because in this if condition or status met the required condition/state then automatically a signal is send to processor. <0.1% CPU Load

Example for Polling Method & Interrupt Method:

Waiting for an Event: Family Vacation

Polling

Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?
Are we there yet?

Interrupts

Wake me up when we get there...

Waiting for an Event: Button Push

Polling

```
while(1)
{
if(P1IFG & BIT2) // P1.2 IFG cleared
{
P1IFG &= ~BIT2; // P1.2 IFG cleared
P1OUT ^= BIT0; // Toggle LED at P1.0
}
```

100% CPU Load

Interrupts

```
// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
P1OUT ^= BIT0; // Toggle LED at P1.0
P1IFG &= ~BIT2; // P1.2 IFG cleared
}
```

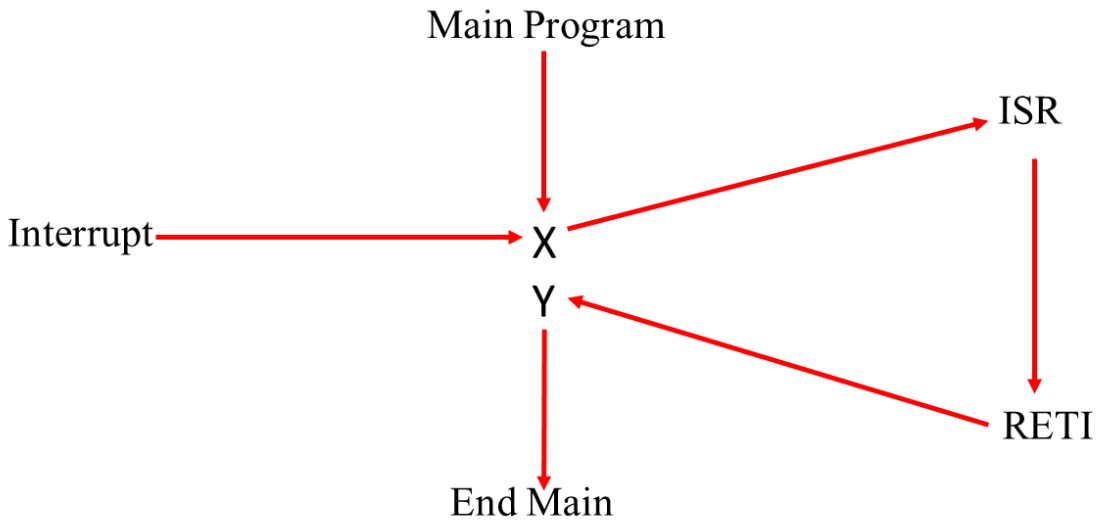
> 0.1% CPU Load

What Happens when an Interrupt Is Requested?

Hardware performs the following steps to launch the ISR:

- ❑ Any currently executing instruction is completed if the CPU was active when the interrupt was requested. MCLK is started if the CPU was off.
- ❑ The PC, which points to the next instruction, is pushed onto the stack.
- ❑ The SR is pushed onto the stack.
- ❑ The interrupt with the highest priority is selected if multiple interrupts are waiting for service.
- ❑ The interrupt request flag is cleared automatically for vectors that have a single source. Flags remain set for servicing by software if the vector has multiple sources
- ❑ The SR is cleared, which has two effects. First, further maskable interrupts are disabled because the GIE bit is cleared; non-maskable interrupts remain active. Second, it terminates any low-power mode
- ❑ The interrupt vector is loaded into the PC and the CPU starts to execute the interrupt service routine at that address.

What happens when interrupt occurs or requested



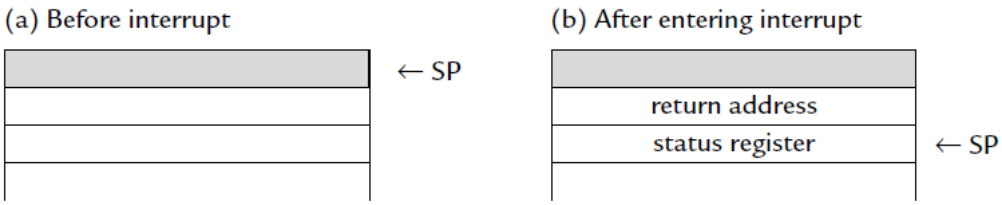
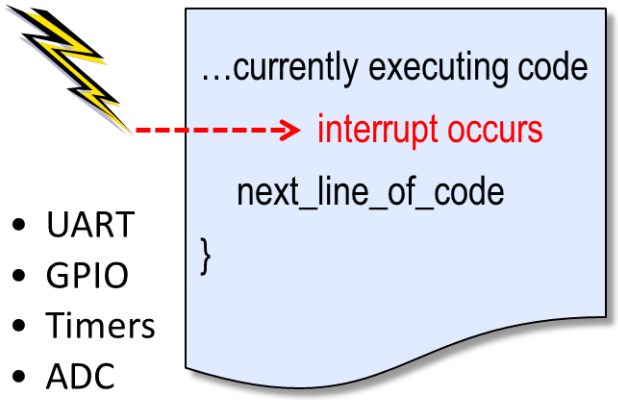


Figure: Stack before and after entering an interrupt service routine. The return address (PC) and status register (SR) have been saved, with SR on the top of the stack.

How do Interrupts Work?

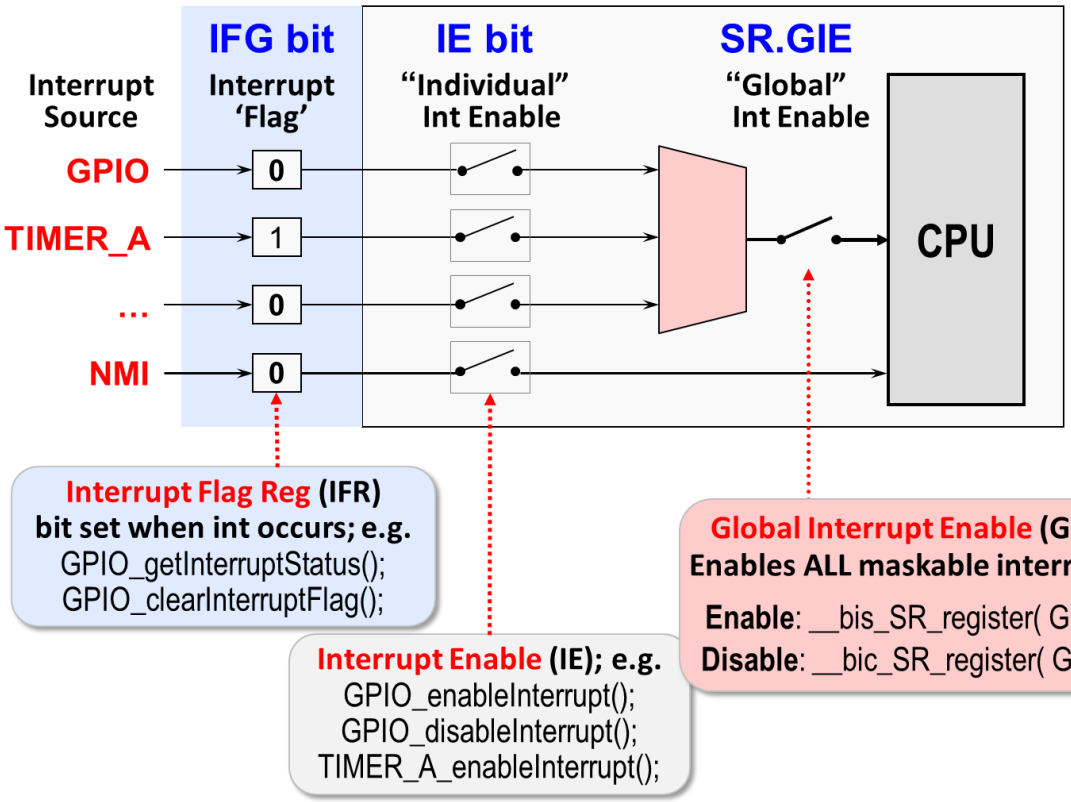
1. An interrupt occurs



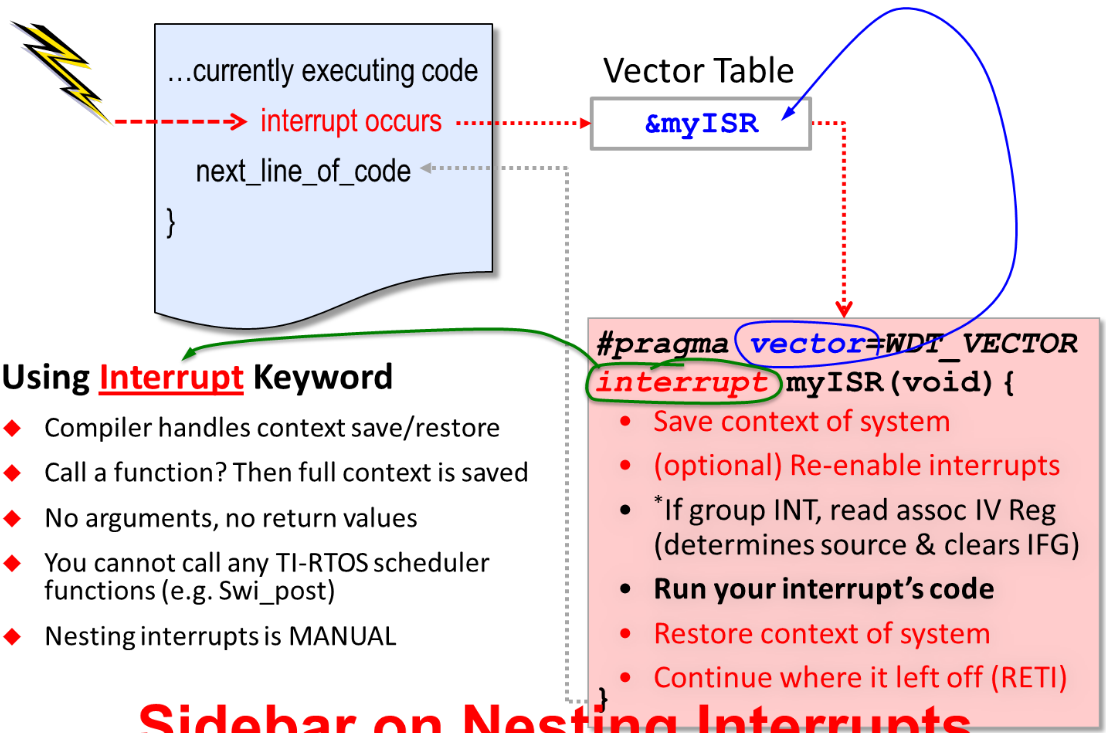
- UART
- GPIO
- Timers
- ADC
- Etc.

2. It sets a flag bit in a register

Interrupt Flow



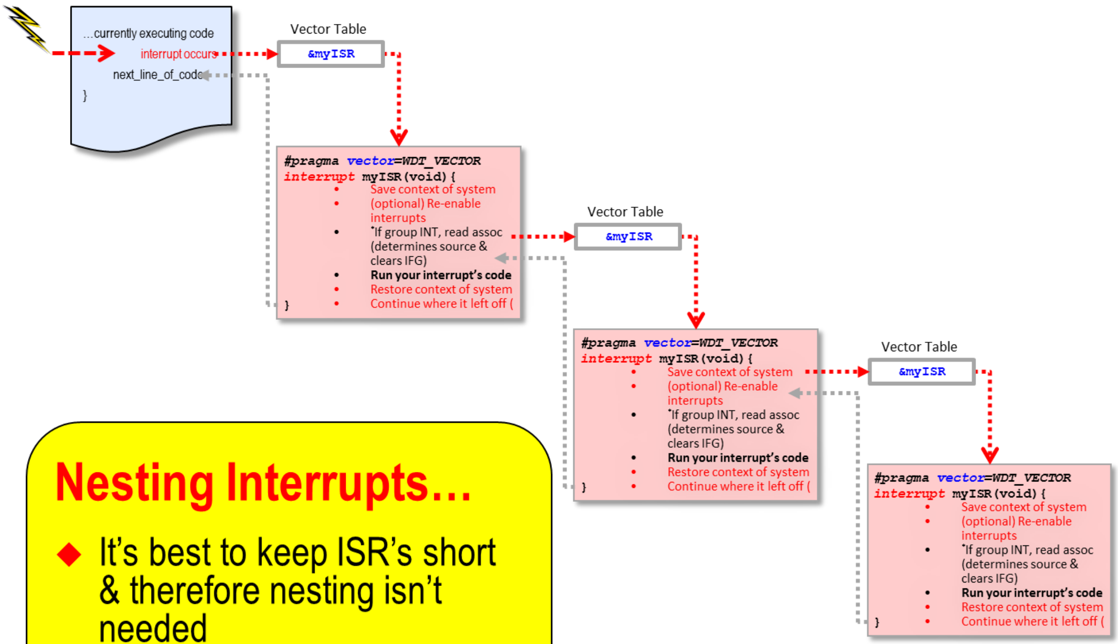
Interrupt Service Routine (ISR)



Using Interrupt Keyword

- Compiler handles context save/restore
- Call a function? Then full context is saved
- No arguments, no return values
- You cannot call any TI-RTOS scheduler functions (e.g. Swi_post)
- Nesting interrupts is MANUAL

Sidebar on Nesting Interrupts



Nesting Interrupts…

- It's best to keep ISR's short & therefore nesting isn't needed

Interrupt Priorities (F5529)

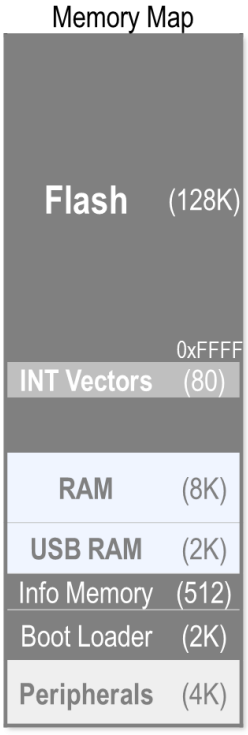
INT Source	Priority
System Reset	high
System NMI	
User NMI	
Comparator	
Timer B (CCIFG0)	
Timer B	
WDT Interval Timer	
Serial Port (A)	
Serial Port (B)	
A/D Convertor	
GPIO (Port 1)	
GPIO (Port 2)	
Real-Time Clock	low

- **There are 23 interrupts** (partially shown here)
- **If multiple interrupts (of the 23) are pending, the highest priority is responded to first**
- **By default, interrupts are not nested …**
  - That is, unless you re-enable INT's during your ISR, other interrupts will be held off until it completes
  - It doesn't matter if the new INT is a higher priority
  - As already recommended, you should keep your ISR's short
- **Most of these represent 'groups' of interrupt source flags**
  - 145 IFG's map into these 23 interrupts

Interrupt Vectors & Priorities (F5529)

INT Source	IV Register	Vector Address	Loc'n	Priority
System Reset	SYSRSTIV	RESET_VECTOR	63	high
System NMI	SYSSNIV	SYSNMI_VECTOR	62	
User NMI	SYSUNIV	UNMI_VECTOR	61	
Comparator	CBIV	COMP_B_VECTOR	60	
Timer B (CCIFG0)	CCIFG0	TIMER0_B0_VECTOR	59	
Timer B	TB0IV	TIMER0_B1_VECTOR	58	
WDT Interval Timer	WDTIFG	WDT_VECTOR	57	
Serial Port (A)	UCA0IV	USCI_A0_VECTOR	56	
Serial Port (B)	UCB0IV	USCI_B0_VECTOR	55	
A/D Convertor	ADC12IV	ADC12_VECTOR	54	
GPIO (Port 1)	P1IV	PORT1_VECTOR	47	
GPIO (Port 2)	P12V	PORT2_VECTOR	42	
Real-Time Clock	RTCIV	RTC_VECTOR	41	low

Legend:		
	Non-maskable	Grouped IFG bits
	Maskable	Dedicated IFG bits



Interrupt Vector Addresses

The interrupt vectors and the power-up start address are located in the address range 0FFFFh to 0FF80h. The vector contains the 16-bit address of the appropriate interrupt-handler instruction sequence.

Table: Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
System Reset Power-Up External Reset Watchdog Time-out, Password	WDTIFG, KEYV (SYSRSTIV)(1)(2)	Reset	0FFFEh	63, highest
System NMI PMM Vacant Memory Access	SVMLIFG, SVMHIFG, DLYLIFG, DLYHIFG, VLRLIFG, VLRHIFG, VMAIFG, JMBNIFG,	(Non)maskable	0FFFCh	62
User NMI NMI Oscillator Fault Flash Memory Access	NMIIFG, OFIFG, ACCVIFG, BUSIFG (SYSUNIV)(1)(2)	(Non)maskable	0FFFAh	61
Comp_B	Comparator B interrupt flags (CBIV)(1)	Maskable	0FFF8h	60
TB0	TB0CCR0 CCIFG0(3)	Maskable	0FFF6h	59
TB0	TB0CCR1 CCIFG1 to TB0CCR6 CCIFG6,	Maskable	0FFF4h	
Watchdog Timer_A Interval Timer	WDTIFG	Maskable	0FFF2h	57
USCI_A0 Receive or	UCA0RXIFG, UCA0TXIFG (UCA0IV)(1)(3)	Maskable	0FFF0h	56
USCI_B0 Receive or	UCB0RXIFG, UCB0TXIFG (UCB0IV)(1)(3)	Maskable	0FFEEh	55
ADC12_A	ADC12IFG0 to ADC12IFG15 (ADC12IV)(1)	Maskable	0FFECh	54
TA0	TA0CCR0 CCIFG0(3)	Maskable	0FFEAh	53
TA0	TA0CCR1 CCIFG1 to TA0CCR4 CCIFG4,	Maskable	0FFE8h	52
USB_UBM	USB interrupts (USBIV)(1)(3)	Maskable	0FFE6h	51
DMA	DMA0IFG, DMA1IFG, DMA2IFG (DMAIV)	Maskable	0FFE4h	50
TA1	TA1CCR0 CCIFG0(3)	Maskable	0FFE2h	49
TA1	TA1CCR1 CCIFG1 to TA1CCR2 CCIFG2,	Maskable	0FFE0h	48
I/O Port P1	P1IFG.0 to P1IFG.7 (P1IV)(1)(3)	Maskable	0FFDEh	47
USCI_A1 Receive or	UCA1RXIFG, UCA1TXIFG (UCA1IV)(1)(3)	Maskable	0FFDCh	46
USCI_B1 Receive or	UCB1RXIFG, UCB1TXIFG (UCB1IV)(1)(3)	Maskable	0FFDAh	45
TA2	TA2CCR0 CCIFG0(3)	Maskable	0FFD8h	44
TA2	TA2CCR1 CCIFG1 to TA2CCR2 CCIFG2,	Maskable	0FFD6h	43
I/O Port P2	P2IFG.0 to P2IFG.7 (P2IV)(1)(3)	Maskable	0FFD4h	42
RTC_A	RTCRDYIFG, RTCTEVIFG, RTCAIFG,	Maskable	0FFD2h	41
Reserved	Reserved(5)		0FFD0h	40
			⋮	⋮
			0FF80h	0, lowest

(1) Multiple source flags

(2) A reset is generated if the CPU tries to fetch instructions from within peripheral space or vacant memory space. (Non) maskable: the individual interrupt-enable bit can disable an interrupt event, but the general-interrupt enable cannot disable it.

(3) Interrupt flags are located in the module.

(4) Only on devices with ADC, otherwise reserved.

(5) Reserved interrupt vectors at addresses are not used in this device and can be used for regular program code if necessary. To maintain compatibility with other devices, TI recommends reserving these locations.

Issues Associated with Interrupts:

1. **Keep Interrupt Service Routines Short**

   Other interrupts, which may be urgent, are disabled during an ISR unless you choose to allow nested interrupts.
2. **Configure Interrupts Carefully**
   - ❑ Make sure that unwanted interrupts are disabled
   - ❑ Make sure that the module is fully configured before interrupts are enabled.
   - ❑ Do not set the GIE bit until you are ready to handle interrupts.
3. **Define All Interrupt Vectors**
   - ❑ Define a single ISR for unused interrupts that traps the processor in an infinite loop.
   - ❑ The address of this routine can then be stored in all the unused interrupt vectors. Use the debugger to trace back if one of these interrupts arises.
4. **The Shared Data Problem**
   - ❑ This is one of the classic issues with interrupts and multitasking systems
   - ❑ It arises when variables are used both in the main function and in ISRs.

WatchDog Timer:

- ❑ This is a safety feature, which resets the processor if the program becomes stuck in an infinite loop/hang.
- ❑ If the selected time interval expires, a system reset is generated.
- ❑ The operation of the watchdog is controlled by the 16-bit register **WDTCTL**
- ❑ *If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.*

Features of the watchdog timer module include:

- ❑ Eight software-selectable time intervals
- ❑ Watchdog mode
- ❑ Interval mode
- ❑ Access to WDT Control register is password protected
- ❑ Selectable clock source
- ❑ Can be stopped to conserve power

- Clock fail-safe feature

WDTCTL

- WDTCTL is a 16-bit, password-protected, read/write register
- The WDT module can be configured as either a watchdog or interval timer with the WDTCTL register.
- Any write accesses must include the write password 05Ah in the upper byte
- Any read of WDTCTL reads 069h in the upper byte.
- The watchdog timer counter (WDTCNT) is a 32-bit up-counter that is not directly accessible by software
- The WDTCNT can be sourced from SMCLK, ACLK, VLOCLK and X_CLK on some devices. The clock source is selected with the WDTSSEL bits

VLO: Very-Low-Power Low-Frequency Oscillator

- The WDTCNT is controlled and its time intervals are selected through the Watchdog Timer Control (WDTCTL) register.
- The clock source is selected with the WDTSSEL bits. The timer interval is selected with the WDTIS bits.

Note:

- Any write to WDTCTL with any value other than 05Ah in the upper byte is a password violation and triggers a PUC system reset, regardless of timer mode.
- Writing byte wide to upper or lower parts of WDTCTL results in a PUC.

Watchdog Mode:

- After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK.
- When the watchdog timer is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC.

Interval Timer Mode:

- Setting the WDTTMSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode

Clock Fail-Safe Feature:

- The WDT_A provides a fail-safe clocking feature, ensuring the clock to the WDT_A cannot be disabled while in watchdog mode. This means that the low-power modes may be affected by the choice for the WDT_A clock.
- If SMCLK or ACLK fails as the WDT_A clock source, VLOCLK is automatically selected as the WDT_A clock source.
- When the WDT_A module is used in interval timer mode, there is no fail-safe feature within WDT_A for the clock source.

Watchdog Timer Control (WDTCTL) Register:

WDTCTL Register

15	14	13	12	11	10	9	8
WDTPW							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

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

Watchdog Timer Block Diagram:

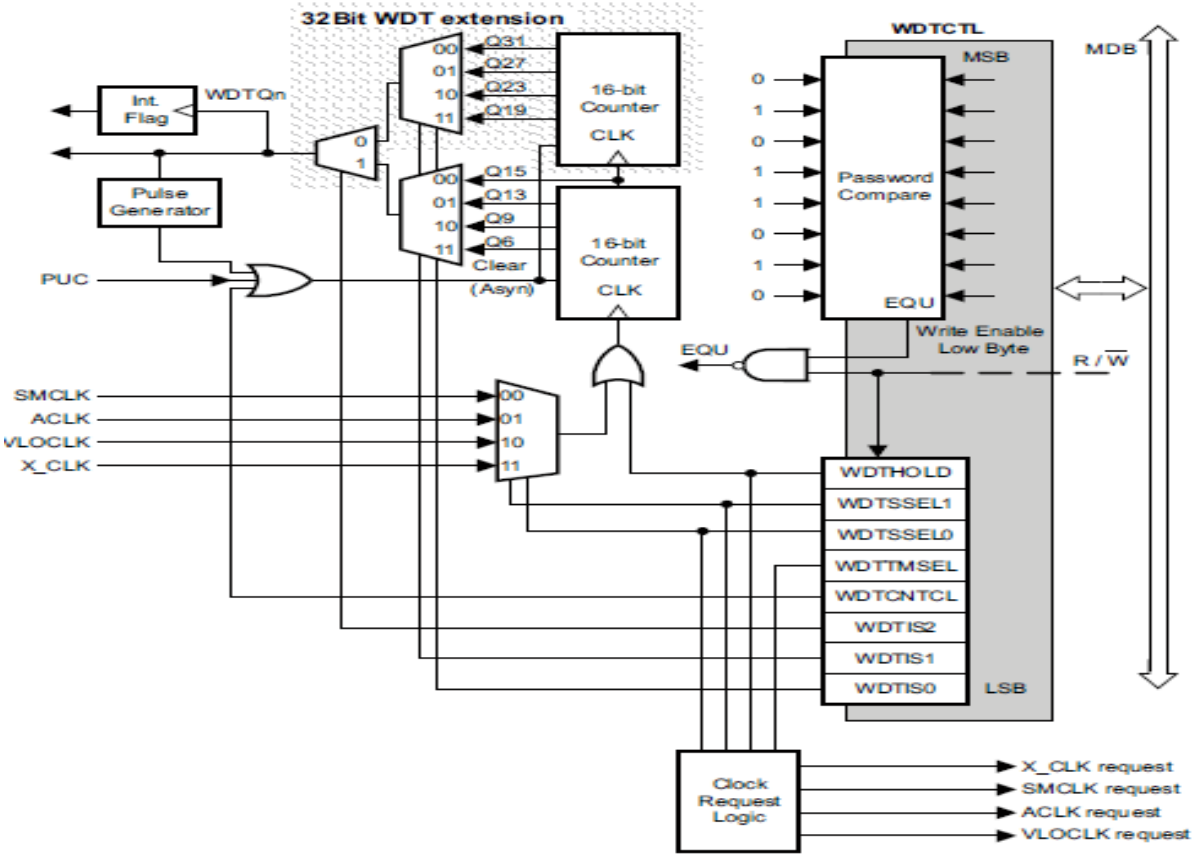


Figure: Watchdog Timer Block Diagram

SYSTEM CLOCKs (CLOCK SYSTEM):

- All microcontrollers contain a clock module to drive the CPU and peripherals
- Clock signal is a square wave whose edges trigger hardware throughout the device so that the changes in different components are synchronized.
- A crystal with a frequency of a few MHz would be connected to two pins. It would drive the CPU directly and was typically divided down by a factor of 2 or 4 for the main bus
- Modern Microcontrollers demands high performance and low power complicated clocks, often with two or more sources

❖ **Two clocks with quite different specifications are often needed:**

- ❑ A fast clock to drive the CPU, which can be started and stopped rapidly to conserve energy but usually need not be particularly accurate.
- ❑ A slow clock that runs continuously to monitor real time, which must therefore use little power and may need to be accurate.

❖ Several types of oscillator are used to generate the clock signal. In these **Crystal** & **RC** are most common

Crystal: Accurate and stable.

- Crystals for microcontrollers typically run at either a high frequency of a few MHz to drive the main bus or a low frequency of 32,768 Hz for a real-time clock.
- The disadvantages are that crystals are expensive and delicate; the oscillator draws a relatively large current, particularly at high frequency.
- The crystal is an extra component and may need two capacitors as well. Crystal oscillators also take a long time to start up and stabilize, often around 105 cycles

Resistor and capacitor *(RC):*

- Cheap and quick to start but used to have poor accuracy and stability.
- The components can be external but are now more likely to be integrated within the MCU.
- The quality of integrated *RC oscillators has improved dramatically in* recent years
- There are also ceramic resonators that lie between these extremes, being cheaper but less accurate and stable than crystals.

The MSP430 uses three internal clocks:

• **Master clock, MCLK**, is used by the CPU and a few faster peripherals.

• **Subsystem master clock, SMCLK**, is distributed to faster peripherals.

• **Auxiliary clock, ACLK,** is also distributed to slower peripherals.

- ❑ SMCLK & MCLK both in the **M**ega **H**ert**z** (MHz) range.
- ACLK is often derived from a watch crystal and therefore runs at a much lower frequency (32 KHz).
- Most peripherals can select their clock from either SMCLK or ACLK.

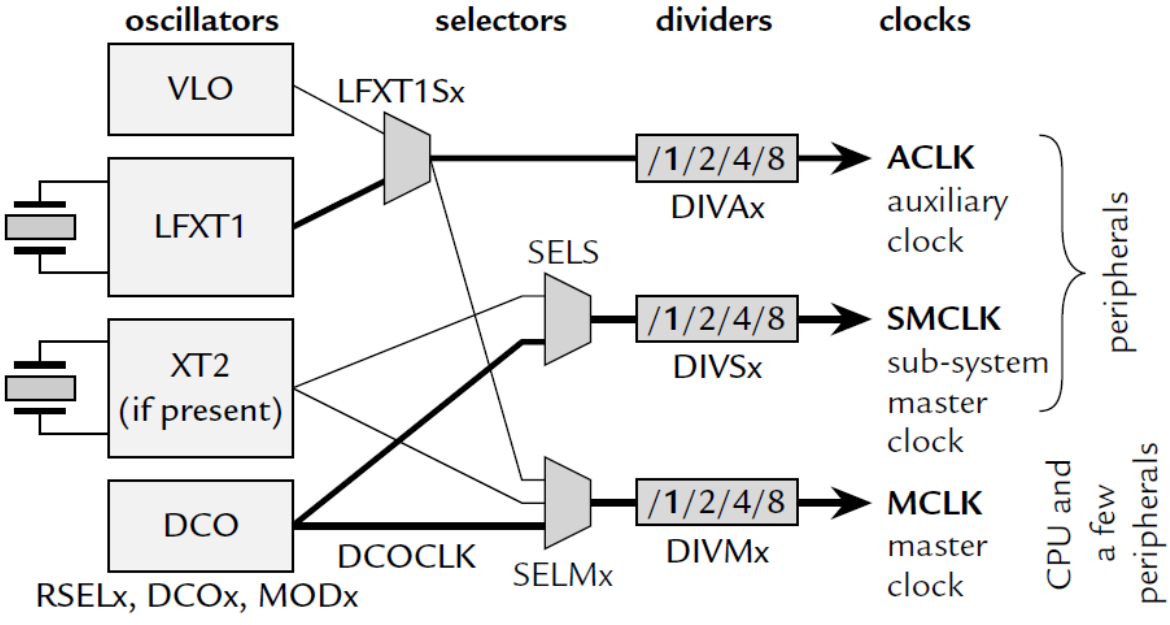


Fig: Simplified block diagram of the clock module of the MSP430F2xx family

Low- or high-frequency crystal oscillator, LFXT1: Available in all devices. It is usually used with a low-frequency watch crystal (32 KHz) but can also run with a high-frequency crystal (typically a few MHz) in most devices.

High-frequency crystal oscillator, XT2: Similar to LFXT1 except that it is restricted to high frequencies. It is available in only a few devices and LFXT1 (or VLO) is used instead if XT2 is missing.

Internal very low-power, low-frequency oscillator, VLO: Available in only the more recent MSP430F2xx devices. It provides an alternative to LFXT1 when the accuracy of a crystal is not needed.

Digitally controlled oscillator, DCO: Available in all devices and one of the highlights of the MSP430. It is basically a highly controllable RC oscillator that starts in less than **1µs** in newer devices.

Crystal Oscillators, LFXT1 and XT2:

Crystals are used when an accurate, stable frequency is needed:

- ***Accurate*** *means that the frequency is close to what it says on the package*
- ***Stable means*** *that does not change significantly with time or temperature.*
  - Oscillator circuits are integrated into the MSP430 and the crystal should be connected to pins XIN and XOUT.
  - The auxiliary clock ACLK can be derived only from LFXT1 in most devices so ACLK will not be available if there is no crystal.
  - The **disadvantage** of this is that the oscillator takes an equally long time to reach a stable state, typically around 10^5 cycles

Internal Low-Power, Low-Frequency Oscillator, VLO:

- The VLO is an internal *RC oscillator that runs at around 12KHz and can be used instead* of LFXT1 in some newer devices.
- It saves the cost and space required for a crystal and reduces the current drawn.
- LFXT1 draws about 0.8µA, which falls to 0.5µA with the VLO
- The purpose is often to wake the device periodically to check whether any inputs have changed, and accuracy is not important.
- ACLK is taken from LFXT1 by default even where the VLO is present.
- This means that current is wasted in LFXT1 and that pins P2.6 and P2.7 in the F20xx are configured for a crystal.
- It is usually a good idea to reconfigure the Clock Module to use the VLO and redirect port 2 for digital input/output.

Digitally Controlled Oscillator, DCO:

- One of the aims of MSP430 was that it should be able to start rapidly at full speed from a low-power mode, without waiting a long time for the clock to settle.
- DCO starts in less than **1µs** in newer devices.
- The frequency can be controlled through sets of bits in the module's registers at three levels
- **RSELx:** Selects one of 16 coarse ranges of frequency. The overall range is about 0.09–20 MHz's
- **DCOx:** Selects one of eight steps within each range. Each step increases the frequency by about 8%, giving a factor of 1.7 from bottom to top of the range.

Control of the Clock Module through the Status Register:

- **CPUOFF** disables MCLK, which stops the CPU and any peripherals that use MCLK.
- **SCG1** disables SMCLK and peripherals that use it.
- **SCG0** disables the DC generator for the DCO
- **OSCOFF** disables VLO and LFXT1.

Low Power aspects of MSP430: LOW POWER MODES

The MSP430 is designed for low-power applications and uses different operating modes.

The MSP430 has one active mode and five software selectable low-power modes of operation.

An interrupt event can wake up the device from any of the low-power modes, service the request, and restore back to the low-power mode on return from the interrupt program.

The following six operating modes can be configured by software:

- ❑ Active mode (AM)
- ❑ Low-power mode 0 (LPM0)
- ❑ Low-power mode 1 (LPM1)
- ❑ Low-power mode 2 (LPM2)
- ❑ Low-power mode 3 (LPM3)
- ❑ Low-power mode 4 (LPM4)

The Active mode and low-power modes 0 to 4 are configured with the CPUOFF, OSCOFF, SCG0, and SCG1 bits in the status register. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the status register is that the present operating mode is saved onto the stack during an interrupt service routine.

Active mode: CPU, all clocks, and enabled modules are active, I ≈ 300µA.

LPM0: CPU and MCLK are disabled, SMCLK and ACLK remain active, I ≈ 85µA. This is used when the CPU is not required but some modules require a fast clock from SMCLK and the DCO.

LPM1: CPU, MCLK, SMCLK, DCO and DC generator are disabled, SMCLK and ACLK remains active.

LPM2: CPU, MCLK, SMCLK, DCO are disabled, DC generator and ACLK remains active.

LPM3: CPU, MCLK, SMCLK, DCO and DC generator are disabled; only ACLK remains active; I ≈ 1µA. This is the standard low-power mode when the device must wake itself at regular intervals and therefore needs a (slow) clock. It is also required if the MSP430 must maintain a real-time clock.

LPM4: CPU and all clocks are disabled, I ≈ 0.1µA. The device can be wakened only by an external signal. This is also called *RAM retention mode*.

CPUOFF → Controls the MCLK
OSCOFF → Controls the VLO & LFXT1
SCG0 → Controls the DC generator
SCG1 → Controls the SMCLK

0 → No Action (or) Active
1 → Action (or) Disables the CLK

Table: Operating Modes of MSP430

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

Real Time Clock (RTC):

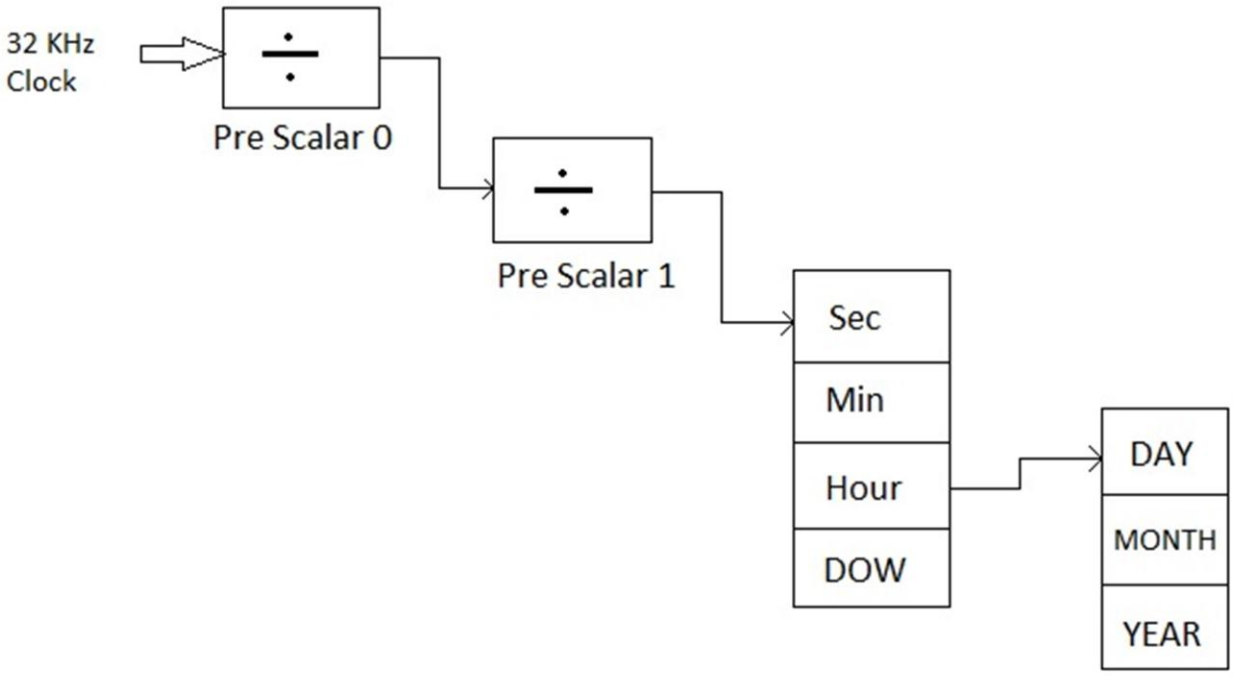
- The basic functions of a real time clock (RTC) are used to display the current time and an alarm clock with calendar functions.
- The alarms are basically a CPU interrupts.
- Most of the RTC are designed for ultra low power.
- The RTC is driven with 32 KHz clock.
- There are various RTC like
    I) RTC_A (uses ACLK or SMCLK).
    II) RTC_B (uses LFXT (32 KHz crystal)).

⇨ RTC_A is used on F5529 and provides to use external crystal or internal (on chip) REFO.

⇨ RTC_B is used on FR5969 which is driven with Low frequency external crystal.

⇨ The advantage of RTC_B can be operate LPM 3.5

RTC Block Diagram



- The input clock is divided by pre scalar 0 and pre scalar 1 to provide Sec, Min, Hour, and Day of Week (DOW), from there to Day, Month, and Year.
- These pre scalar count values are used to…
    - Create time-stamps
    - Set displays & etc.

RTC Interrupts:

Interrupts are of six types:

1) **Alarm** (RTCA): Generate interrupt if match between time register and alarm register for
    - Minutes
    - Hours
    - Day of the week
    - Day (of the month)

2) **Interval timer (**RTCTEV):
    - Generates when RTC count events occur.
    - Interrupt can happen for each minute, each hour, midnight, & noon.
3) **Pre scalar 0**(RT0PS) & 4) **Pre scalar 1**(RT1PS) : RTC has a capable of generating the time based interrupts from the pre scalar count value when clock rate is divided by 2, 4, 8, 16, 32, 64, 128, 256.
5) **Ready** (RTCRDY) : Used to tell or notifies the CPU to safe read or write the RTC registers.
6) **Oscillator fault:** Generates when CPU is running in LPM 3.5, which cannot detect oscillator. Then this interrupt wakes up the CPU and notify that the fault occurred at oscillator.

RTC Control Register (RTCCTL)**:**

7	6	5	4	3	2	1	0
RTCBCD	RTCHOLD	RTCMODEx		RTCTEVx		RTCIE	RTCFG

- If RTCHOLD is set then the clock does not run by default.
- The current time and date held in registers.
- The registers sometimes will be as word length for accessing the “year”.
- The RTCBCD is used to set the display.
- The RTC takes the control on Basic Timer 1, when BTDIV bit = 1.
- The interrupt flag bit RTCFG is enabled when RTCIE bit is set.
- The flag is set for every minute, hour, daily at noon and at midnight based on RTCTEVx bit.
- The bit RTCTEVx determines the intervals.
- The maskable interrupts can be accessed in two ways:
    1. Interrupts are generated when RTCIE bit is set. Both BTIFG & RTCFG are set when an interrupt occurs and cleared automatically when it served.

2. Interrupts come from Basic Timer1, if RTCIE is clear. The interval is determined by BTIPx. The Real-Time Clock sets its RTCFG flag according to RTCTEVx but this does *not* request an interrupt. A program can poll the flag to check whether an interval of time has elapsed and must clear RTCFG in software.

Additional features of RTC:

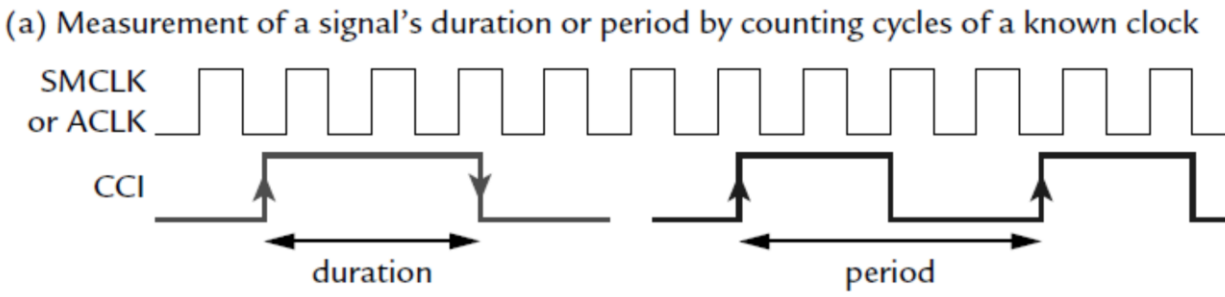
- All RTC's work in LPM 3
- The RTC_B and RTC_C can be operates in LPM 3.5 due to RTC_B & RTC_C are access the LF crystal directly.
- RTC_B / RTC_C provide BCD conversion
- RTC_A can be used as 32-bit counter (rather than calendar mode).
- Counter mode generates an overflow interrupts.

RTC Comparison

Feature	RTC_A	RTC_B	RCT_C
Highlights	32-bit Counter Mode	LPM3.5, Calendar Mode Only	Protection Plus Improved Calibration & Compensation
Calendar Mode with Programmable Alarms	Yes	Yes	Yes
Counter Mode	Yes	No	Device-dependent
Input Clocks	ALCK, SMCLK	32-kHz crystal	32-kHz crystal
LPM3.5 Support	No	Yes	Yes
Offset Calibration Register	Yes	Yes	Yes
Temperature Compensation Register	No	No	Yes
Temperature Compensation	With software, manipulating offset calibration value		With software using separate temperature compensation register
Calibration and Compensation Period	64 min	60 min	1 min
BCD to Binary Conversion	Integrated for Calendar Mode	Integrated for Calendar Mode plus separate conversion registers	
Event/Tamper Detect With Time Stamp	No	No	Device-dependent
Password Protected Calendar Registers	No	No	Yes

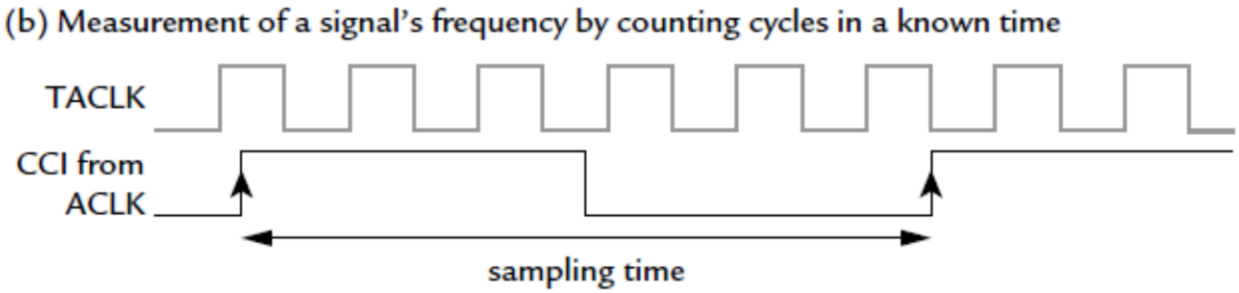
Measurement in Capture Mode:

- The capture mode is used to take a time stamp of an event. (Time at which it occurs).
- The timer can be used in two opposite ways as mentioned.
    1. In most cases the timer clock is either ACLK or SMCLK.
        - In the first method to measure the length of a single pulse the both edges are captured (start & end).
        - The duration of the pulse is measured with unit of the timer clock's period.
        - The rising edges of the pulse are captured for a periodic signal (falling edge if needed).
        - For a good resolution of a signal the period of the clock should be less than the duration.



2. The second method used to measure a signal with a higher frequency.
    - The signal is captured the events with clock edges whose frequency known.
    - The difference between the timer clock and captured value gives the frequency rather than the period.

⇨ **The first method is much more common.**



Example:

- Many speed sensors produce a given number of pulses per revolution of a shaft where capture mode is used to measure the period between pulses.
- In communication the capture mode is used to detect the time stamp of the start time of data received.
- The FLL can be emulated with capture mode to compare the frequencies of SMCLK and ACLK.

Measurement of Time: Press and Release of a Button

- For example the time between press and release of the button S2 is connected to port P1.1 active low with pull-up resistor.
- The Timer_A runs fast in continuous mode to detect the switch bounce.
- The period should be longer than the press of button is 0.5 s.
- The ACLK is accurate when it comes from crystal but SMCLK is not unless a FLL is used.
- Capture/compare channel 0 is configured to capture and interrupt request on both raising and falling edge to release and press the button.
- The capture compare channel copies the value of TAR into TACCRn when input is detected.
- Timer_A avoids the delay occurs if another interrupt is being served when it want to be computed.
- Timer_A can be programmed by setting CCIS 1 bit in TACCTL0, which set up port 1 to generate interrupt when S2 is pressed or released and copy the new state to TACCTL0.

Measurement of Time: Reaction Time

- The Timer_A is used to measure the “Reaction Time”.
- The time is displayed in milliseconds, so 1 KHz clock is sufficient.
- The ACLK generates 4 KHz with divider 8.
- The limitation is the slowest reaction time is just 2s, which is not a problem in all cases.
- In FG4618 microcontroller
    - The Interrupts are enabled for the start button S2, connected to P1.1.
    - The timing button S1, connected to P1.0, is routed to capture input CCI0A of Timer_A.
    - Capture/compare channel 0 is used for the captures because S1 provides the input CCI0A.
    - Timer_A runs from ACLK with no division, giving a period of 2s
- To measure the reaction time the formula is :

$$\text{time (ms)} = \frac{1000 \times \text{time (counts)}}{f_{\text{ACLK}}}$$

with $f_{\text{ACLK}} = 32\,\text{KHz} = 32{,}768$ Hz. It is tempting to “simplify” the numbers to give

$$\text{time (ms)} = \frac{\text{time (counts)}}{32{,}768}$$

Measurement of frequency: Comparison of SMCLK and ACLK

- The measure of frequency is to count the number cycles, “N” in a known interval of time T.

$$\boxed{f = \frac{\mathrm{N}}{T}}$$

- If T = 1 ms then N gives the frequency in “KHz”
- The count of number of cycles is requires an accurate reference provided by ACLK.
- The channel is configured to capture the value of TAR, which requires an interrupt at raising edge of ACLK.
- The difference between the new and old value of TAR (N1-N2) gives number of cycles of the signal in one cycle of ACLK
- Thus the frequency measured is

$$f = (N_2 - N_1)\, f_{\text{ACLK}}.$$

- Basic timer 1 is configured for a real-time interrupts every 0.5s for measurement.

- Capture/compare channel 2 is used for the measurements because ACLK is connected to CCI2B. It is configured for rising edges and synchronized captures.
- The ISR for basic Timer 1 enables the interrupt on TACCR2 to start a new measurement. First it clears the COV and CCIFG bits, which are set by previous capture events.
- The value of captures is determined automatically without TACCR2 value.
- The MSP430 returns to LPM0 until the next interrupt generated from Basic timer 1.

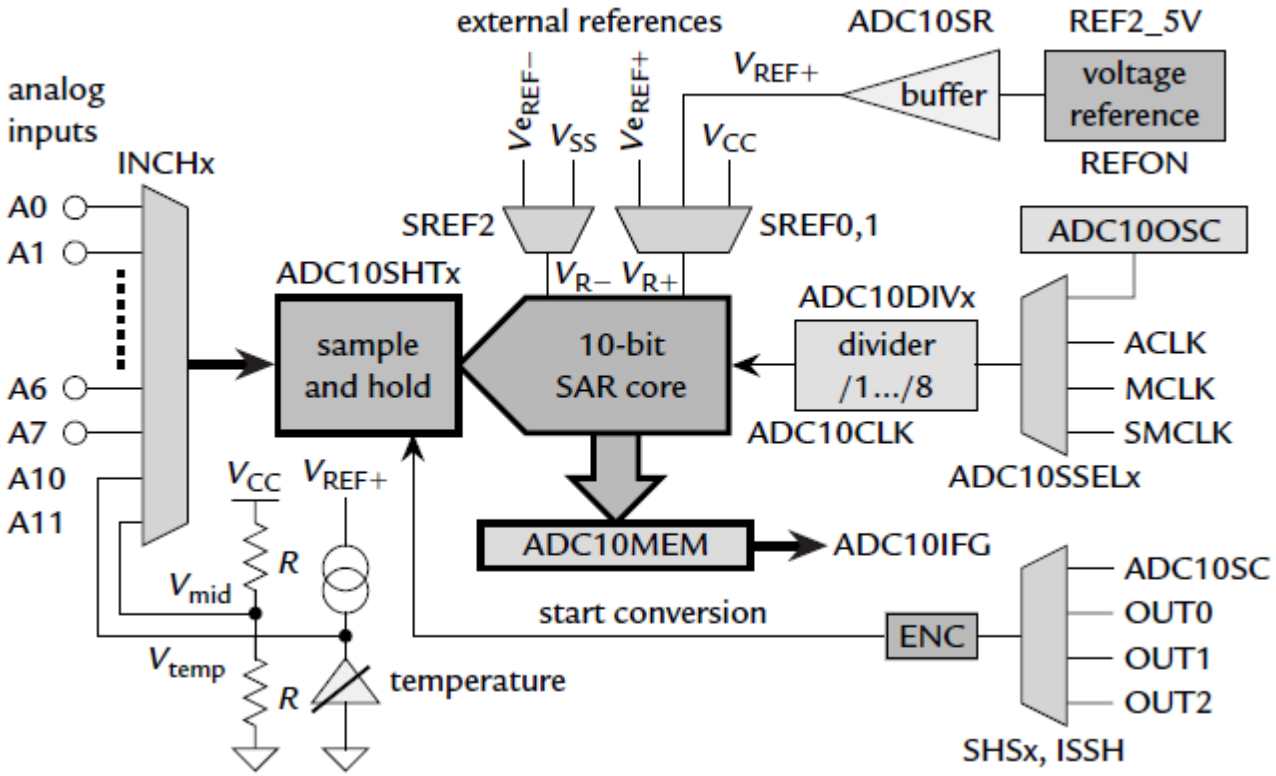
Analog to Digital Converter: ADC 10:

- The ADC 10 in MSP430 implements a 10-bit SAR code, sample select control reference generator and data transfer controller (DTC).
- The DTC allows the ADC 10 samples to be converted and stored in the memory without CPU intervention.

ADC 10 features:

- ADC 10 conversion rate is greater than 200 KSPS.
- It is a monotonic 10-bit converter without any missing code.
- Sampled and can be programmable.
- Software Timer_A can initiate the conversion
- Software selectable of internal or external reference voltage (1.5v or 2.5v)
- MSP430 supports up to 8 input channels. (12 on MSP430F22XX family)
- It supports single channel, repeated-single channel, sequence and repeated sequence conversion modes.
- ADC core and reference voltages sources can be power down automatically.

Simplified block diagram of ADC 10/12:



10-Bit ADC Core:

- The ADC core converts an analog input to its 10-bit digital equivalent and stores in memory (ADC10 MEM register).
- The upper and lower voltage levels are defined as V_{R+} and V_{R-} .
- The output digital value (N_{ADC}) arrives when full scale value (03FFH) reached when input signal equal or greater than V_{R+} and zero when V_{R-} .
- The input channel and the reference voltage levels (VR+ & VR-) are known as "conversion – control memory".
- The conversion formula using straight binary format (2's complement format) is

$$N_{ADC} = 1023\ X\ \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

- The ADC10 has two control word register ADC10CTL0 and ADC10CTL1.
- The core is enabled with ADC10 on bit.
- This ADC10 control bit can be modified using ENC = 0.
- ENC must be 1 before conversion.

Conversion clock selection:

- The ADC source clock is selected by ADC10SSELx bit (SMCLK, ACLK, MCLK and internal oscillator ADCOSC).
- ADC10 CLK remains active until the end of conversion

ADC10 inputs & multiplexer:

- There are 8-external and four internal signals are selected by multiplexer.

Analog port selection:

- ADC10AEx bits provide the disabling the ports.
    - P2.3 ON: MSP43022XX configures analog input.
    - B1S.3 #08, ADC10AE0; P2.3 ADC10 function enable.

Direct Memory Access (DMA) Introduction:

- The DMA controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC conversion memory to RAM.
- Devices that contain a DMA controller may have up to eight DMA channels available. Therefore, depending on the number of DMA channels available, some features described in this chapter are not applicable to all devices. See the device-specific data sheet for number of channels supported.
- Using the DMA controller can increase the throughput of peripheral modules. It can also reduce system power consumption by allowing the CPU to remain in a low-power mode, without having to awaken to move data to or from a peripheral.

DMA controller features include:

- Up to eight independent transfer channels
- Configurable DMA channel priorities
- Requires only two MCLK clock cycles per transfer
- Byte or word and mixed byte and word transfer capability
- Block sizes up to 65535 bytes or words
- Configurable transfer trigger selections
- Selectable-edge or level-triggered transfer
- Four addressing modes
- Single, block, or burst-block transfer modes

DMA Operation:

- The DMA controller is configured with user software. The setup and operation of the DMA is discussed in the following sections.

DMA Addressing Modes:

- The DMA controller has four addressing modes. The addressing mode for each DMA channel is
- independently configurable. For example, channel 0 may transfer between two fixed addresses, while
- channel 1 transfers between two blocks of addresses. The addressing modes are shown in Figure 1-2.
- The addressing modes are:
- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses
- The addressing modes are configured with the DMASRCINCR and DMADSTINCR control bits. The
- DMASRCINCR bits select if the source address is incremented, decremented, or unchanged after each
- transfer. The DMADSTINCR bits select if the destination address is incremented, decremented, or
- unchanged after each transfer.
- Transfers may be byte to byte, word to word, byte to word, or word to byte. When transferring word to
- byte, only the lower byte of the source-word transfers. When transferring byte to word, the upper byte of
- the destination-word is cleared when the transfer occurs.

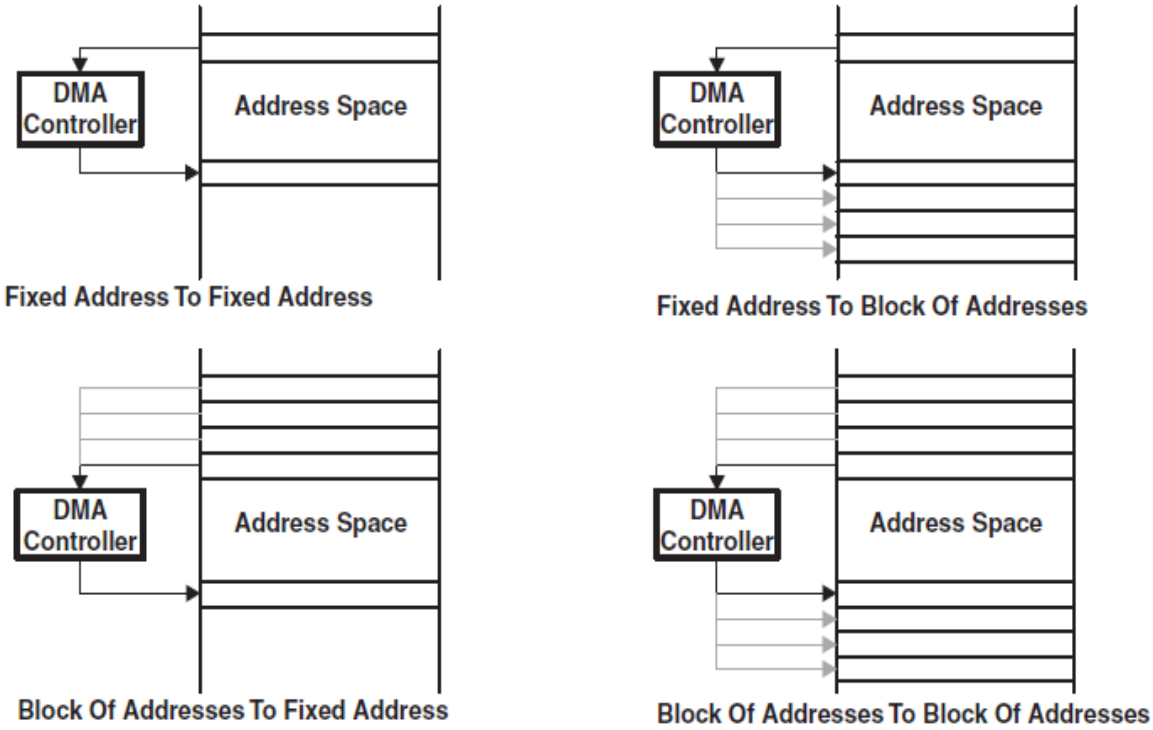


Figure 1-2. DMA Addressing Modes

DMA Transfer Modes:

- The DMA controller has six transfer modes selected by the DMADT bits as listed in Table 1-1. Each channel is individually configurable for its transfer mode. For example, channel 0 may be configured in single transfer mode, while channel 1 is configured for burst-block transfer mode, and channel 2 operates in repeated block mode.
- The transfer mode is configured independently from the addressing mode. Any addressing mode can be used with any transfer mode.
- Two types of data can be transferred selectable by the DMAxCTL DSTBYTE and SRCBYTE fields. The source and/or destination location can be either byte or word data. It is also possible to transfer byte to byte, word to word, or any combination.

Table 1-1. DMA Transfer Modes

DMADT	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

Single Transfer:

- In single transfer mode, each byte/word transfer requires a separate trigger. The single transfer state diagram is shown in Figure 1-3.
- The DMAxSZ register is used to define the number of transfers to be made. The DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer. If DMAxSZ = 0, no transfers occur.
- The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer.
- The DMAxSZ register is decremented after each transfer. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set. When DMADT = {0}, the DMAEN bit is cleared automatically when DMAxSZ decrements to zero and must be set again for another transfer to occur.
- In repeated single transfer mode, the DMA controller remains enabled with DMAEN = 1, and a transfer Occurs every time a trigger occurs.

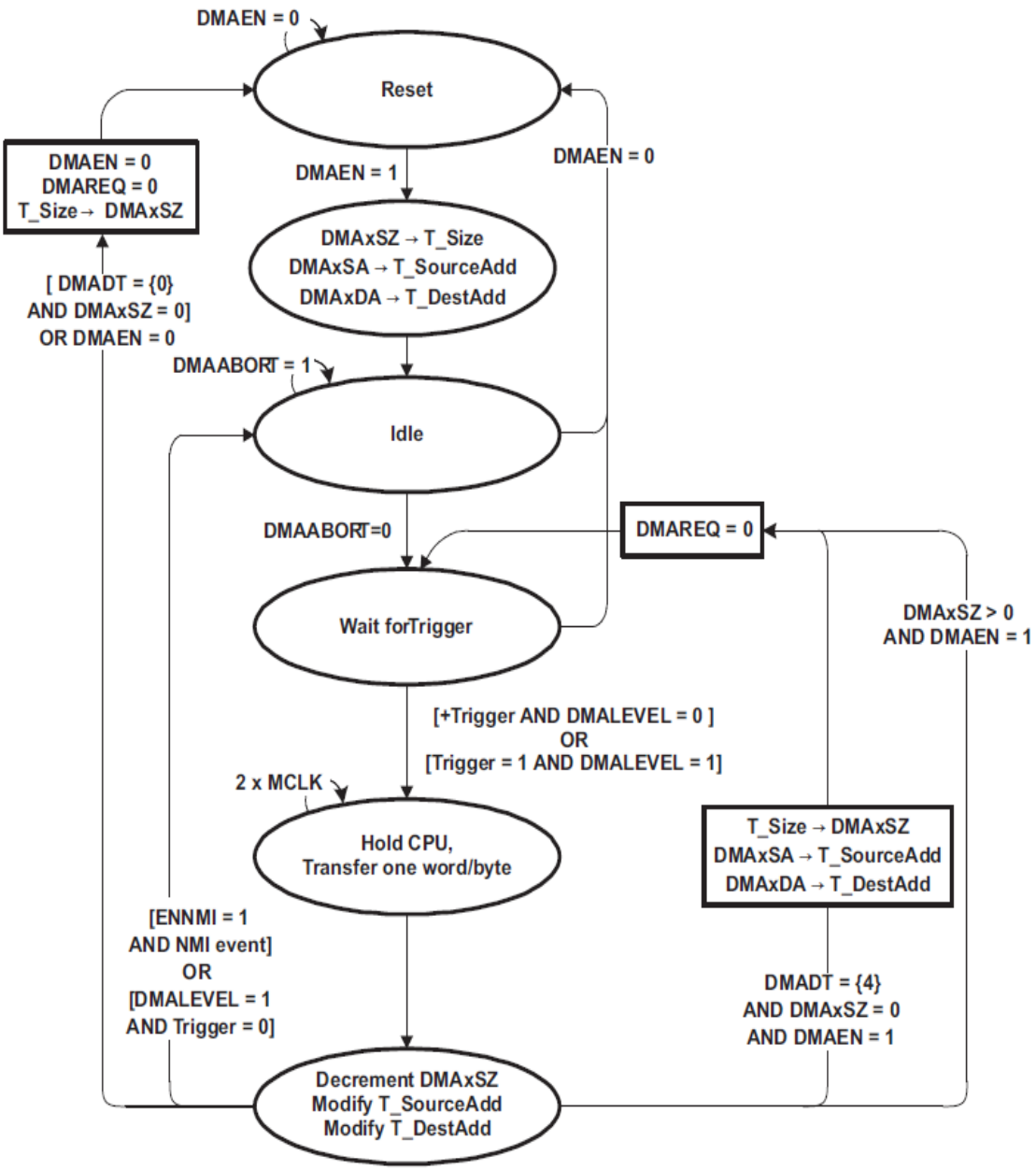


Figure 1-3. DMA Single Transfer State Diagram

Block Transfer:

- In block transfer mode, a transfer of a complete block of data occurs after one trigger. When DMADT = {1} ,the DMAEN bit is cleared after the completion of the block transfer and must be set again before another block transfer can be triggered. After a block transfer has been triggered, further trigger signals occurring during the block transfer are ignored. The block transfer state diagram is shown in Figure 1-4.
- The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

- The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set.
- During a block transfer, the CPU is halted until the complete block has been transferred. The block transfer takes 2 × MCLK × DMAxSZ clock cycles to complete. CPU execution resumes with its previous state after the block transfer is complete.
- In repeated block transfer mode, the DMAEN bit remains set after completion of the block transfer. The next trigger after the completion of a repeated block transfer triggers another block transfer.

Burst-Block Transfer:

- In burst-block mode, transfers are block transfers with CPU activity interleaved. The CPU executes two MCLK cycles after every four byte/word transfers of the block, resulting in 20% CPU execution capacity. After the burst-block, CPU execution resumes at 100% capacity and the DMAEN bit is cleared.
- DMAEN must be set again before another burst-block transfer can be triggered. After a burst-block transfer has been triggered, further trigger signals occurring during the burst-block transfer are ignored.
- The burst-block transfer state diagram is shown in Figure 1-5.
- The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.
- The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero, it is reloaded from its temporary register and the corresponding DMAIFG flag is set.
- In repeated burst-block mode, the DMAEN bit remains set after completion of the burst-block transfer and no further trigger signals are required to initiate another burst-block transfer.
- Another burst-block transfer begins immediately after completion of a burst-block transfer. In this case, the transfers must be stopped by clearing the DMAEN bit, or by an (non)maskable interrupt (NMI) when ENNMI is set. In repeated burst block mode the CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.

Stopping DMA Transfers:

- There are two ways to stop DMA transfers in progress:
- A single, block, or burst-block transfer may be stopped with an NMI, if the ENNMI bit is set in register
- DMACTL1.
- A burst-block transfer may be stopped by clearing the DMAEN bit.

DMA Channel Priorities:

- The default DMA channel priorities are DMA0 through DMA7. If two or three triggers happen simultaneously or are pending, the channel with the highest priority completes its transfer (single, block, or burst-block transfer) first, then the second priority channel, then the third priority channel.
- Transfers in progress are not halted if a higher-priority channel is triggered. The higher-priority channel waits until the transfer in progress completes before starting.
- The DMA channel priorities are configurable with the ROUNDROBIN bit. When the ROUNDROBIN bit is set, the channel that completes a transfer becomes the lowest priority.
- The *order* of the priority of the channels always stays the same, DMA0-DMA1-DMA2, for example, for three channels.
- When the ROUNDROBIN bit is cleared, the channel priority returns to the default priority.

DMA Priority	Transfer Occurs	New DMA Priority
DMA0-DMA1-DMA2	DMA1	DMA2-DMA0-DMA1
DMA2-DMA0-DMA1	DMA2	DMA0-DMA1-DMA2
DMA0-DMA1-DMA2	DMA0	DMA1-DMA2-DMA0

- **1.2.6.1 DMA Triggering:**

Table 1-2. DMA Trigger Operation

Module	Operation
DMA	A transfer is triggered when the DMAREQ bit is set. The DMAREQ bit is automatically reset when the transfer starts. A transfer is triggered when the DMAxIFG flag is set. DMA0IFG triggers channel 1, DMA1IFG triggers channel 2, and DMA2IFG triggers channel 0. None of the DMAxIFG flags are automatically reset when the transfer starts. A transfer is triggered by the external trigger DMAE0.
Timer_A	A transfer is triggered when the TAxCCR0 CCIFG flag is set. The TAxCCR0 CCIFG flag is automatically reset when the transfer starts. If the TAxCCR0 CCIE bit is set, the TAxCCR0 CCIFG flag dies not trigger a transfer. A transfer is triggered when the TAxCCR2 CCIFG flag is set. The TAxCCR2 CCIFG flag is automatically reset when the transfer starts. If the TAxCCR2 CCIE bit is set, the TAxCCR2 CCIFG flag does not trigger a transfer.
Timer_B	A transfer is triggered when the TBxCCR0 CCIFG flag is set. The TBxCCR0 CCIFG flag is automatically reset when the transfer starts. If the TBxCCR0 CCIE bit is set, the TBxCCR0 CCIFG flag does not trigger a transfer. A transfer is triggered when the TBxCCR2 CCIFG flag is set. The TBxCCR2 CCIFG flag is automatically reset when the transfer starts. If the TBxCCR2 CCIE bit is set, the TBxCCR2 CCIFG flag does not trigger a transfer.
USCI_Ax	A transfer is triggered when USCI_Ax receives new data. UCAxRXIFG is automatically reset when the transfer starts. If UCAxRXIE is set, the UCAxRXIFG does not trigger a transfer. A transfer is triggered when USCI_Ax is ready to transmit new data. UCAxTXIFG is automatically reset when the transfer starts. If UCAxTXIE is set, the UCAxTXIFG does not trigger a transfer.
USCI_Bx	A transfer is triggered when USCI_Bx receives new data. UCBxRXIFG is automatically reset when the transfer starts. If UCBxRXIE is set, the UCBxRXIFG does not trigger a transfer. A transfer is triggered when USCI_Bx is ready to transmit new data. UCBxTXIFG is automatically reset when the transfer starts. If UCBxTXIE is set, the UCBxTXIFG does not trigger a transfer.
DAC12_A	A transfer is triggered when the DAC12_xCTL0 DAC12IFG flag is set. The DAC12_xCTL0 DAC12IFG flag is automatically cleared when the transfer starts. If the DAC12_xCTL0 DAC12IE bit is set, the DAC12_xCTL0 DAC12IFG flag does not trigger a transfer.
ADC10_A	A transfer is triggered by an ADC10IFG0 flag. A transfer is triggered when the conversion is completed and the ADC10IFG0 is set. Setting the ADC10IFG0 with software does not trigger a transfer. The ADC10IFG0 flag is automatically reset when the ADC10MEM0 register is accessed by the DMA controller.
ADC12_A	A transfer is triggered by an ADC12IFG flag. When single-channel conversions are performed, the corresponding ADC12IFG is the trigger. When sequences are used, the ADC12IFG for the last conversion in the sequence is the trigger. A transfer is triggered when the conversion is completed and the ADC12IFG is set. Setting the ADC12IFG with software does not trigger a transfer. All ADC12IFG flags are automatically reset when the associated ADC12MEMx register is accessed by the DMA controller.
MPY	A transfer is triggered when the hardware multiplier is ready for a new operand.
Reserved	No transfer is triggered.

DMA Transfer Cycle Time:

- The DMA controller requires one or two MCLK clock cycles to synchronize before each single transfer or complete block or burst-block transfer. Each byte/word transfer requires two MCLK cycles after synchronization, and one cycle of wait time after the transfer. Because the DMA controller uses MCLK, the DMA cycle time is dependent on the MSP430 operating mode and clock system setup.
- If the MCLK source is active but the CPU is off, the DMA controller uses the MCLK source for each transfer, without reenabling the CPU.
- If the MCLK source is off, the DMA controller temporarily restarts MCLK, sourced with DCOCLK, for the single transfer or complete block or burst-block transfer.

- The CPU remains off and after the transfer completes, MCLK is turned off. The

Table 1-3. Maximum Single-Transfer DMA Cycle Time

CPU Operating Mode Clock Source	Maximum DMA Cycle Time
Active mode MCLK = DCOCLK	4 MCLK cycles
Active mode MCLK = LFXT1CLK	4 MCLK cycles
Low-power mode LPM0/1 MCLK = DCOCLK	5 MCLK cycles
Low-power mode LPM3/4 MCLK = DCOCLK	5 MCLK cycles + 5 µs(1)
Low-power mode LPM0/1 MCLK = LFXT1CLK	5 MCLK cycles
Low-power mode LPM3 MCLK = LFXT1CLK	5 MCLK cycles
Low-power mode LPM4 MCLK = LFXT1CLK	5 MCLK cycles + 5 µs(1)

(1) The additional 5 µs are needed to start the DCOCLK. It is the $t_{(LPMx)}$ parameter in the data sheet.

maximum DMA cycle time for all operating modes is shown in Table 1-3.

Using DMA With System Interrupts:

- DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the transfer. NMIs can interrupt the DMA controller if the ENNMI bit is set.
- System interrupt service routines are interrupted by DMA transfers. If an interrupt service routine or other routine must execute with no interruptions, the DMA controller should be disabled prior to executing the routine.

DMA Controller Interrupts:

- Each DMA channel has its own DMAIFG flag. Each DMAIFG flag is set in any mode when the corresponding DMAxSZ register counts to zero. If the corresponding DMAIE and GIE bits are set, an interrupt request is generated.
- All DMAIFG flags are prioritized, with DMA0IFG being the highest, and combined to source a single interrupt vector. The highest-priority enabled interrupt generates a number in the DMAIV register. This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled DMA interrupts do not affect the DMAIV value.
- Any access, read or write, of the DMAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.
- For example, assume that DMA0 has the highest priority. If the DMA0IFG and DMA2IFG flags are set when the interrupt service routine accesses the DMAIV register, DMA0IFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the DMA2IFG generates another interrupt.

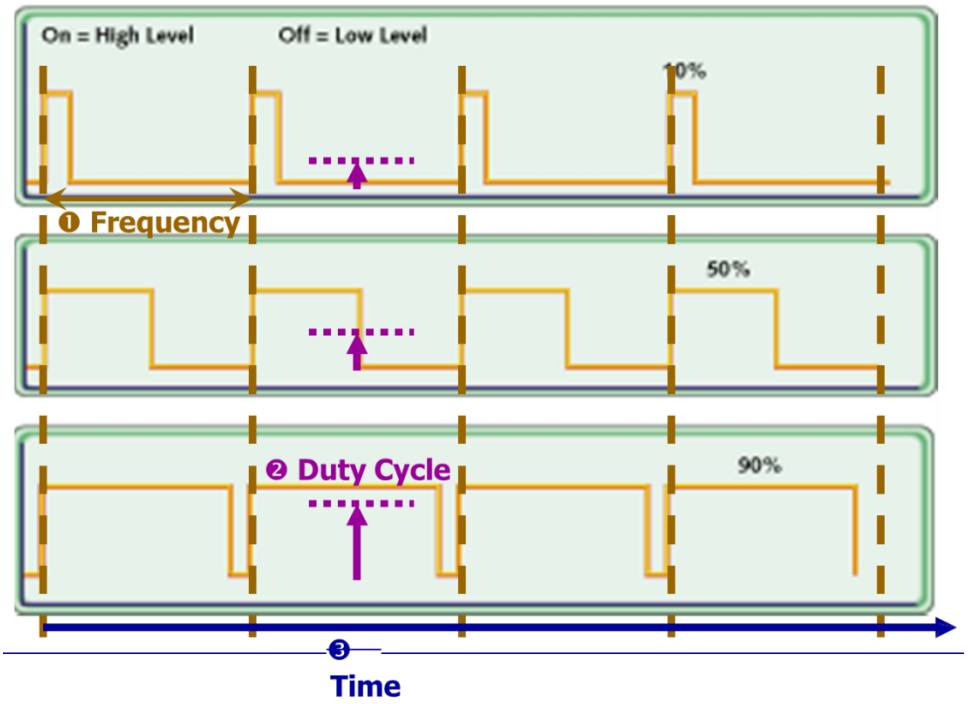
PWM Control:

A Pulse Width Modulation (PWM) Signal is a method for generating an analog signal using a digital source. A PWM signal consists of two main components that define its behavior: a duty cycle and a frequency. The duty cycle describes the amount of time, the signal is in a high (on) state as a percentage of the total time of it takes to complete one cycle. The frequency determines how fast the PWM completes a cycle (i.e. 1000 Hz would be 1000 cycles per second), and therefore how fast it switches between high and low states. By cycling a digital signal on and off at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices. The PWM is still digital because at any given time the full DC supply is either fully ON (or) OFF. The duration of the period where the signal is high is known as “Duty Cycle”.

Uses of PWM Control:

- Pulse Width Modulation is used to control analog circuits with digital outputs.
- PWM signals are used for a wide variety of control applications. Their main use is for controlling DC motors, Servo motors, LEDs and also to control valves, pumps, hydraulics, and other mechanical parts.

PWM Frequency/Duty Cycle:



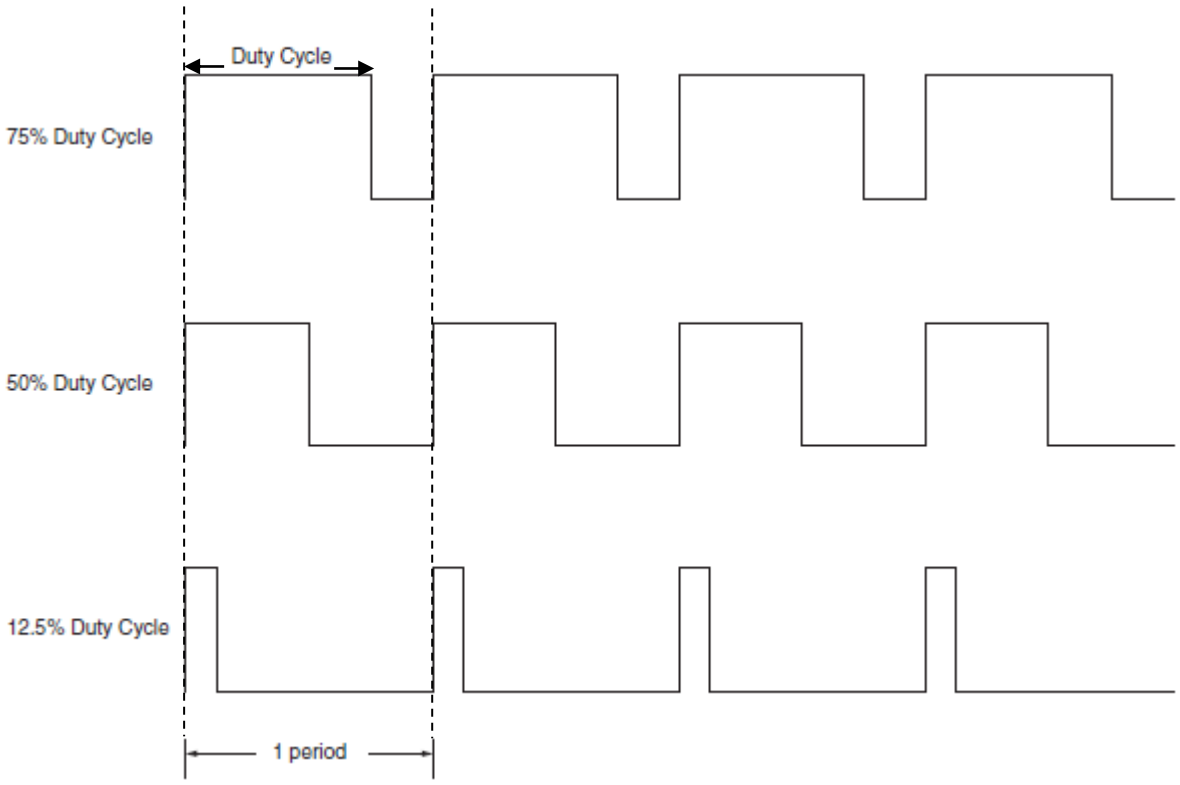
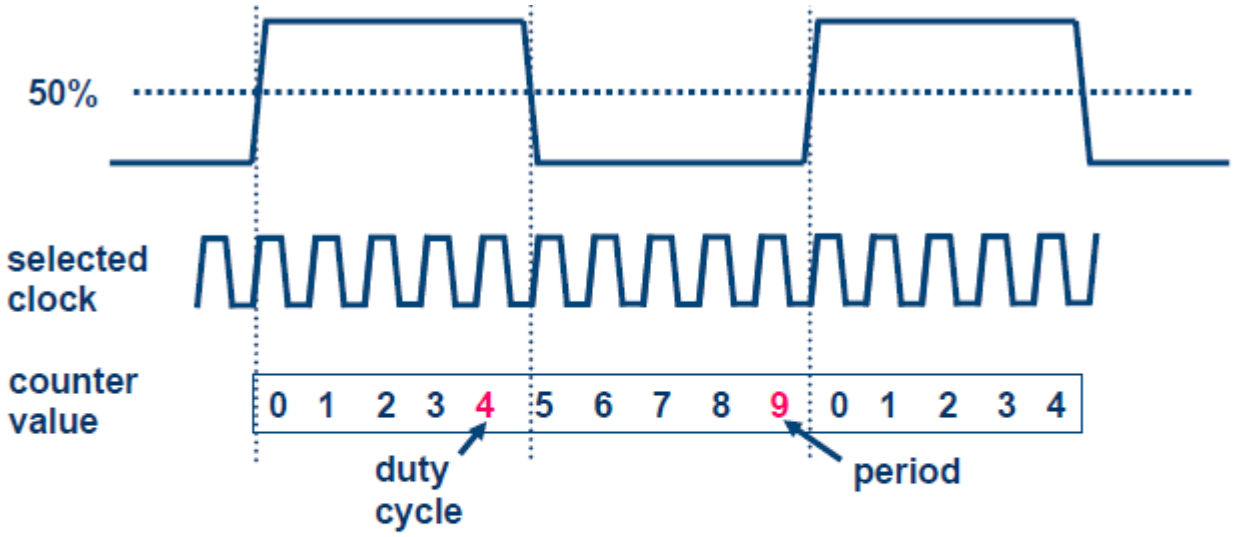


Figure: PWM Sample Outputs



PWM frequency = F_{sc} / (period)
= 100KHz / 10 = 10 KHz

PWM duty cycle = ((period – duty cycle) / (period)) * 100%
= ((10 – 5) / 10) * 100% = 50%

Figure: PWM Frequency and Duty Cycle

Measurement in the Capture Mode:

The Capture mode is used to take a time stamp of an event: to note the time at which it occurred. A measurement typically requires two or more captures. The timer can be used in two opposite ways shown in Figure.

(1) In most cases the timer clock is either ACLK or SMCLK, whose frequency is known, and the unknown signal is applied to the capture input. To measure the length of a single pulse, we should capture both edges and subtract the captured times. This gives the duration of the pulse in units of the timer clock's period.

(2) The opposite approach is used to measure a signal with a high frequency. The signal is used as the timer clock and the captured events are typically edges of ACLK, whose frequency is known. The difference between that and the captured value gives the number of cycles of the signal in one cycle of ACLK. This gives the frequency rather than the period.

The first method is much more common.

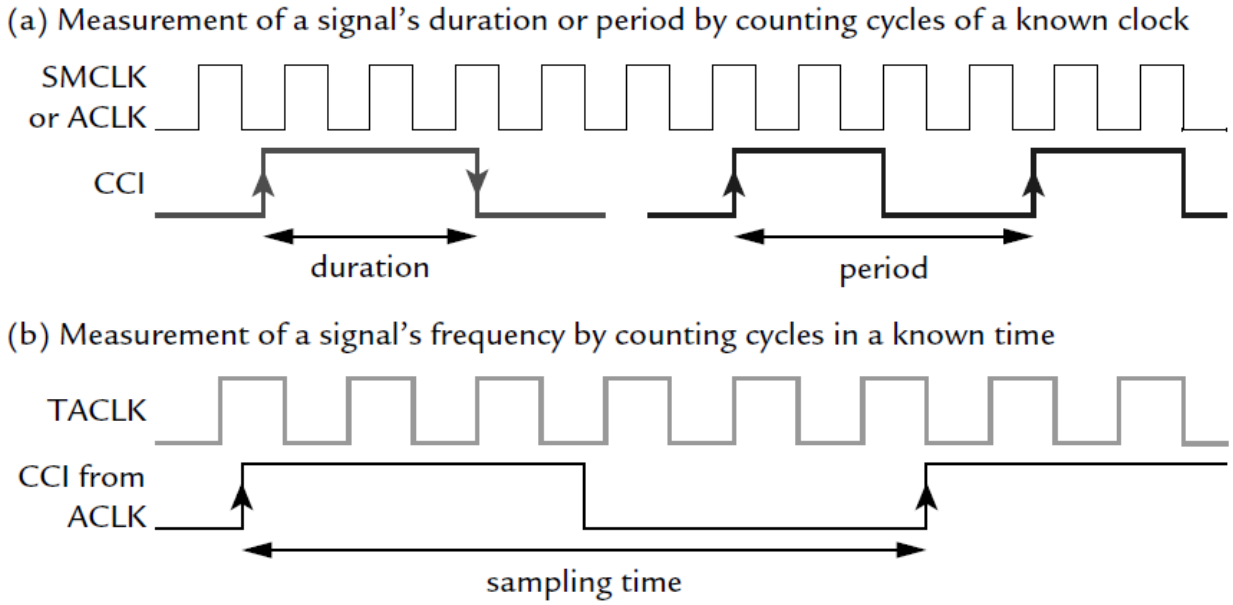


Figure: Two ways in which the Capture mode is used to time a signal. (a) The duration or period of the signal CCI is measured by counting the number of cycles of a known clock. (b) The frequency of the external signal TACLK is measured by using it as the clock and counting the number of cycles during a known interval.