

Fundamentals of microcontrollers for Embedded Systems

The objective of this book is to serve as an introduction to Microcontrollers for Embedded Systems. In the first chapter of the book, the reader will gain a basic understanding of embedded systems and its applications.

The reader will be introduced to topics such as architecture consideration of an embedded system; difference between CISC and RISC; Von-Neumann and Harvard Architecture.

The chapter also includes an introduction to TI's MSP430x2xx and MSP430x5xx microcontrollers and their architectural features. Additionally, variants of MSP430 microcontroller family and their comparison along with application segments are also covered.

Topic	Page
1.1 Introduction	2
1.2 Embedded system overview	3
1.3 Embedded application development process.....	4
1.4 Applications	6
1.5 Features and architecture considerations	6
1.6 CISC Vs RISC design philosophy	12
1.7 Von-Neumann Vs Harvard architecture	12
1.8 Choosing a Microcontroller for an Embedded System.....	14
1.9 On-chip peripherals and Register sets of MSP430F5529	17
1.10 Addressing Modes.....	20
1.11 Instruction format.....	21
1.12 Instruction set	22
1.13 Variants of MSP430 family	24
1.14 Sample Applications of Embedded System on MSP430 Microcontroller.....	26
1.15 Summary	28
1.16 Review Questions.....	29
1.17 Exercises	30

1.1 Introduction

Today, embedded systems are used across a variety of devices right from toys to appliances such as automatic washing machines, TV remotes, handheld gadgets etc. These systems are built with the help of a control unit known as the CPU core, which consists of microcontrollers. Microcontrollers can be categorised based on the amount of data they can process. For example, TI's 16-bit microcontroller's process 16 bits of data and at the same time dissipate very less power as compared to other controllers of the same class.

Embedded systems also consist of high resolution ADCs, multipliers and high resolution PWMs; thus targeting low power real time applications like consumer electronics, tests and measurement devices etc. One such low power application is in portable medical devices such as pulse oximeter.

Pulse oximeter is a device that is used to measure the oxygen levels of a patient. TI's MSP430 16-bit low power controllers provide a single chip solution for the designers. The operation is described below:

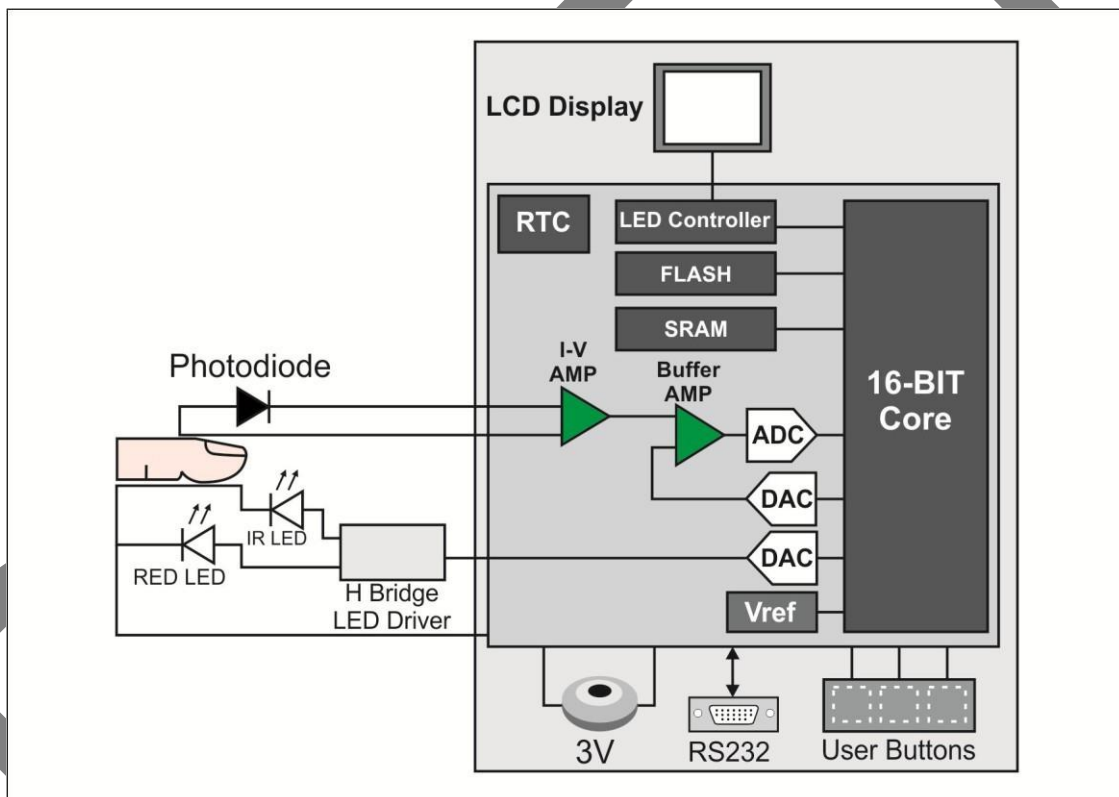


Fig 1.a: Pulse oximeter Block Diagram

TI has a portfolio of devices such as MSP430FR2633, MSP430F5528, MSP430FG439 and MSP430FG6626, which can be used to design a pulse oximeter. For e.g. MSP430FG6626 has a 16 bit inbuilt ADC and 12 bit DAC that suits high resolution data sampling requirements of ADC to design a pulse oximeter. Also, this type of portable devices (like pulse oximeters) require ultra-low power processors because they usually rely on an external battery source for their power requirements.

1.2 Embedded system overview

An embedded system is an electronic system designed using hardware and software to accomplish a dedicated application. Hardware may be either a microprocessor or microcontroller or an application specific chip. Software is an application specific program developed by the programmer and deployed inside the hardware by various methods to control over the application along with the hardware. The hardware used for the embedded applications is also called embedded hardware and the program developed by the programmer for embedded applications is called embedded software. Most of the time, the embedded system uses microcontrollers since it has a CPU to process the data, internal memory to accommodate the program, input/output path for data flow, internal buses to transport the data and information to various required places internally in addition to other peripheral devices essential for the applications. Hence the embedded system can also be called as a single chip computer for specific or dedicated applications. The microcontroller used for embedded applications can also be called as an embedded microcontroller or embedded computer. Like general purpose computers, the embedded system may also use an operating system called Real Time Operating System (RTOS) to plan, execute and control the series of tasks systematically. The basic architecture of an embedded system is shown in figure 1.b

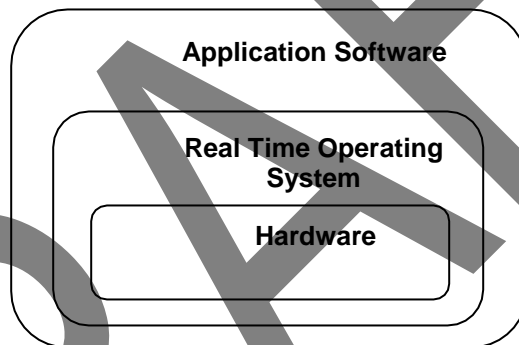


Figure 1.b: Basic architecture of an embedded system

The embedded system resembles activities of general purpose computers; it however differs in a few aspects. The general purpose computer or microcomputer is designed using a microprocessor, memories external to microprocessor, input/output devices and bus systems and can be used for a variety of applications. Even the manufacturer cannot predict the application, since it entirely depends on the user. The user may use it for applications such as data processing, gaming, listening to music or others. Whereas in the case of an embedded system, its application is known very well to the designer or manufacturer and it is used for dedicated applications by the users or customers. General purpose computers do not have any constraints to meet, but embedded systems have system specific constraints. The important constraints are reliable operation, low processing power, low density memory and size.

Embedded systems are also called as real time systems since they process real time events. Based on the type of real time processing, embedded systems are classified as soft real time systems and hard real time systems. A system that is not required to meet a specific deadline is called a soft real time system. On the other hand, a system that is required to meet a dynamic deadline is called a hard real time system. Missing the deadline of a hard real time system can cause severe damage to users. Hence, to meet this kind of requirement, more powerful microcontrollers are introduced by various manufacturers. One such type is a RISC (Reduced Instruction Set Computer or Code) machine which is designed to operate with a few simple instructions and provide greater computational capability. An earlier version of this is the CISC (Complex or Complicated Instruction Set Computer or Code) machine, designed to operate with large number of complex instructions and accomplish a task.

The market for embedded systems is growing steadily, with several billion embedded computers being sold every year. However, there exist certain challenges when it comes to the design of embedded systems:

- The application must be focused.
- The application must have a real time requirement.
- The system must have zero or very minimum user interaction.
- The system must co-operate with the environment.
- The system must be highly reactive.
- The system must be smaller in size, low weight and must use limited memory.
- The system must be low cost, safe and reliable

1.3 Embedded application development process

The development process varies according to the type of industry, but the traditional design cycle includes following steps shown in figure 2.

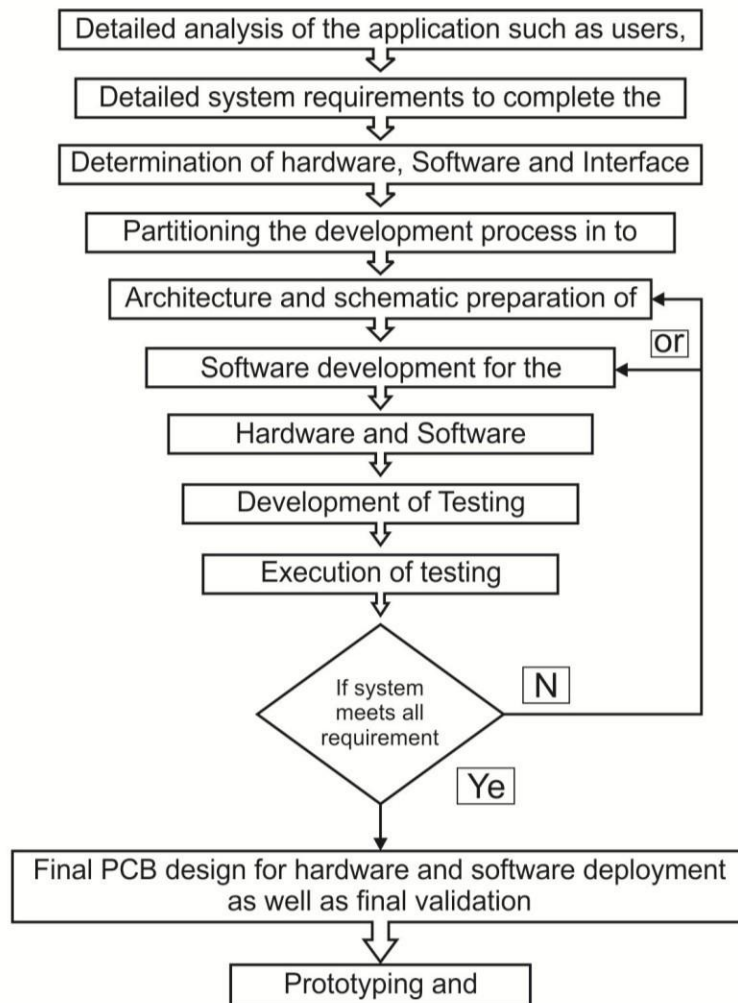


Figure 2: Embedded Application Design Cycle

Host and Target environment

A „Host“ is a personal computer used for the development of a software algorithm or software programme, which will be downloaded into the target to execute the application. „Target“ is as an actual embedded system machine (based on microprocessor/ microcontroller) that understands only the machine language composed of 0s and 1s. Programmers generally do not use machine language for programming due to complexities in the trouble shooting process. Instead, they use either assembly language or high level language. Assembly language instructions are developed by the microcontroller or microprocessor manufacturers; but programmers can use any high-level language that is supported by the target. The developed programme or code using any of the language (other than machine language) is finally converted in to machine language by any of the following tools installed in the host computer.

Assembler: A tool to convert an assembly language programme into machine language.

Compiler: A tool to convert a high level language programme into machine language.

Any instruction errors or syntax errors in the program can be identified by a tool called debugger.

The behaviour of the programme developed for the target machine can be visualised in a software tool called Emulator.

A combination of all the above tools is bundled as a package and is available for programming. It is called Integrated Development Environment (IDE) - One such tool for the MSP 430 programming is Code Composer Studio (CCS).

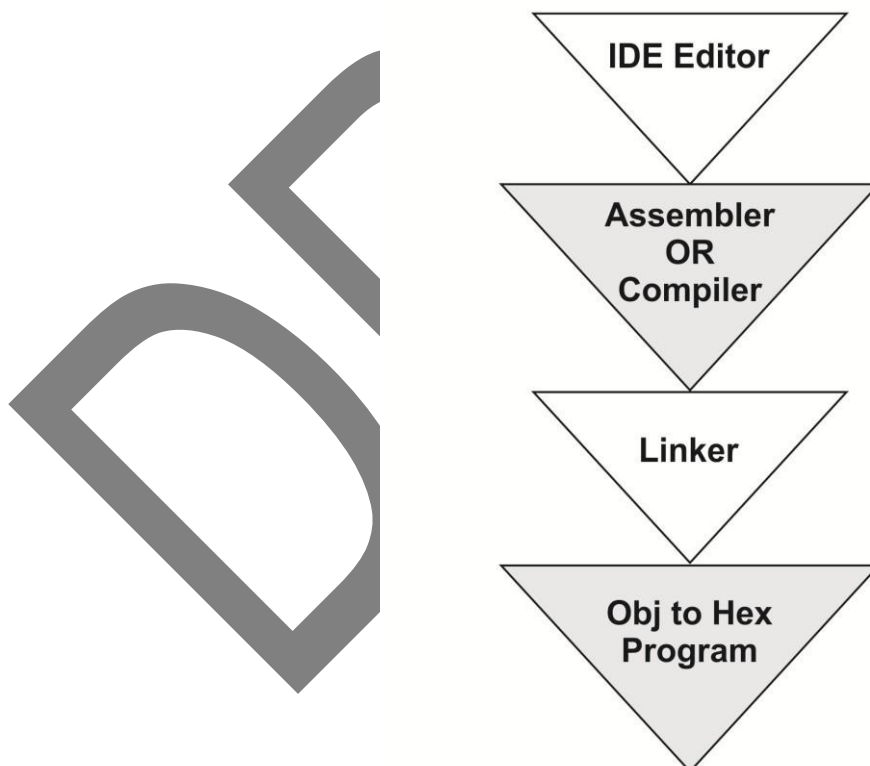


Figure 3: Program flow in IDE

The converted machine code will be in a hexadecimal format and is downloaded inside the target machine using a device called a Hardware Programmer for some of the microcontrollers. But the recent microcontrollers are provided with JTAG interface for debugging and programming and hence the use of separate programmer is also eliminated.

The hexadecimal code is also called as a machine code and is converted into 0s and 1s inside the target machine.

Downloaded code may not work perfectly inside the target machine. An In Circuit Emulator (ICE) is tool used to identify any issues.

The process of conversion of an Assembly language/High level language programme into hexadecimal machine language in the IDE is shown in the figure 3.

The programme typed in the editor can be assigned a file name with the extension of .asm for assembly language or .c for C language. During the assembling or compilation process, list and object files are generated with the same file name. The other object files required for the program are linked by the linker and the new object file is created with the same file name. Then the OH program (object file to Hex file) converts that file into hex file with the same file name and it can be downloaded in to the target machine.

1.4 Applications

An embedded system may be a standalone system or it may be a part of larger system. Hence it finds applications in:

- **Computer industry:** CD drives control, key board control, display control, mouse, printer etc.
- **Electronic industry:** Photo copier, scanner, digital diary, remote controller, data acquisition systems, testing and measuring instruments, games and toys, exercise equipments, washing machine, audio/video equipments, alert systems etc.
- **Communication industry:** Fax machine, telephones, mobile phones, networking devices, ATM etc.
- **Automotive industry:** Electronic ignition, braking systems, power windows, fuel indication, Engine Automation etc.
- **Military:** Smart weapons, missile guiding systems, pagers, global positioning systems etc.
- **Medical industry:** All type of medical instruments such as ECG machine, pacemaker, dialysis machine, CT scanner, drug delivery systems etc.

1.5 Features and architecture considerations

1.5.1 Features of Embedded Systems

- Since the size of the processors is reduced, the possibility for the portable products is increased.
- Embedded processors reduce the number of electronic components in the application design and prove to be efficient in performance and reliability.
- The cost of an embedded processor is reduced but its performance as well as the number of peripherals inside the chip has increased. Hence the cost and time required to design and develop the product is also reduced.
- The power consumption of the processor is also reduced and hence most of the embedded systems are battery operated.
- The increased number of efficient hardware and software development tools as well as testing and measurement tools facilitate complete bug free products.

1.5.2 Architecture Considerations

Embedded Architecture describes the process of planning and designing the components inside the embedded processor (Note: The word „processor“ is a common term used to represent both Microprocessor and Microcontroller). It is essential to know and understand the architecture of an embedded processor in order to design the system effectively. The internal components of the architecture may or may not be sufficient for system design. Hence architecture of the embedded chip provides a solution for following challenges.

- 1) Size and cost of the embedded system
- 2) Power and memory requirement
- 3) Safety and reliability
- 4) Operating environment and other resources required for the system.
- 5) Integration flexibility and interaction with external devices etc.

The number of components in the architecture, their capacity and efficiency depends on the development of technology, application requirement and the manufacturers.

The figure 6 shows the chief difference between the Microprocessor and Microcontroller due to the advancement of technology. The Microprocessor has a CPU, a few registers for data transfer and control units. On the other hand, a Microcontroller has a CPU, internal memory (program and data memory), input/output ports, Serial communication interface and peripherals required for the application design. The figure 7 shows the possibility of peripherals (I/O devices) present in the architecture of embedded processor chip irrespective of the manufacturers.

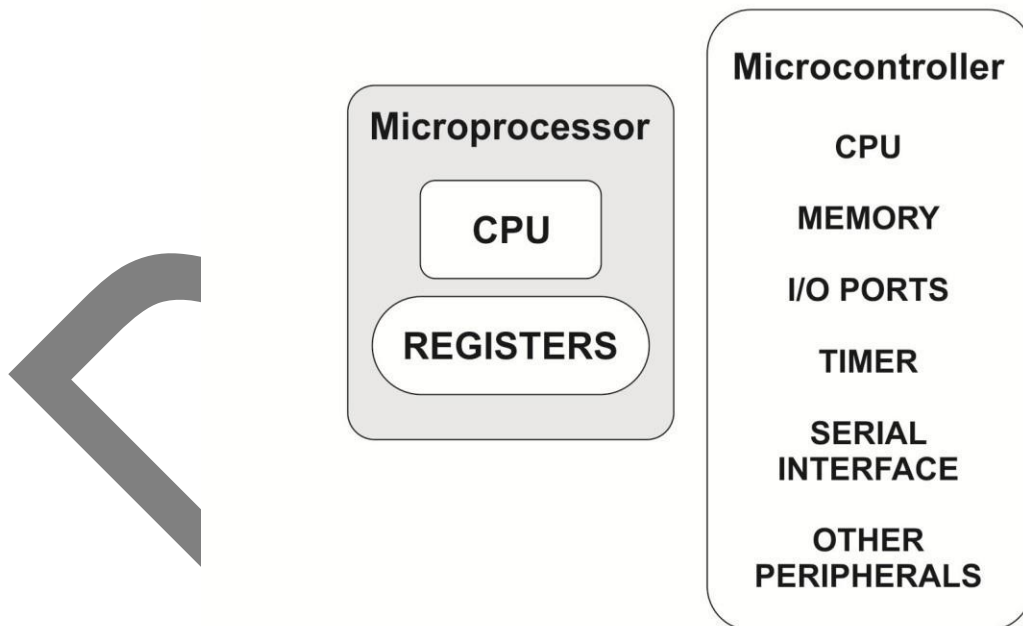


Figure 6: Block of microprocessor and microcontroller

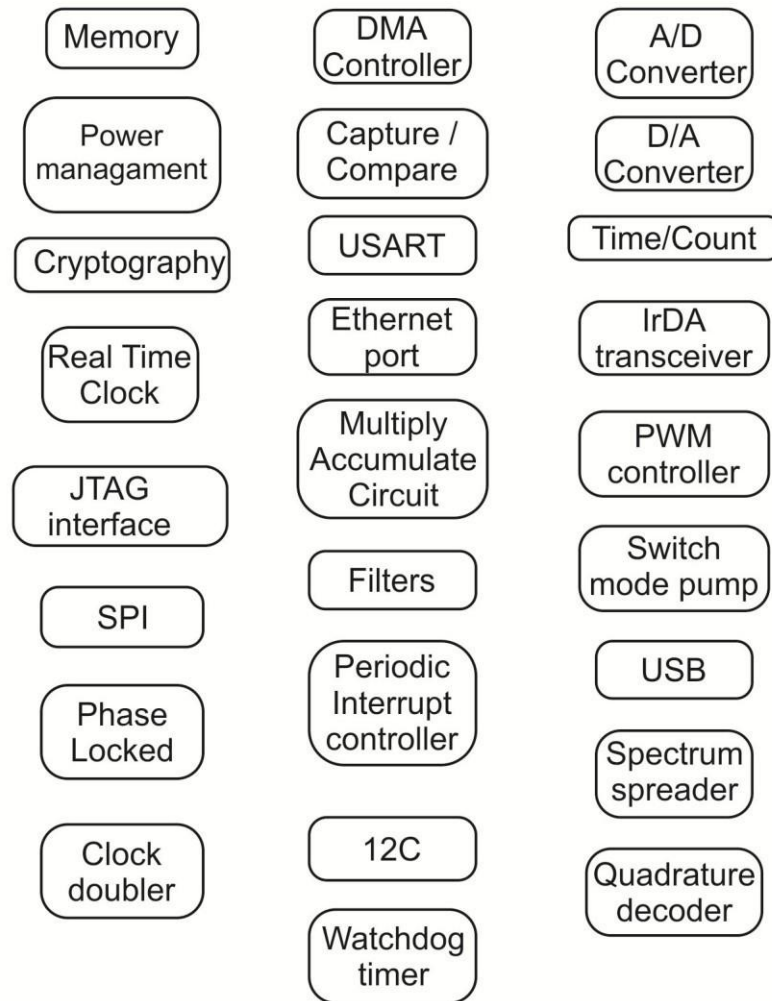


Figure 7: Possibility of peripherals in embedded processor

Below are short descriptions of the peripherals shown in figure 7.

DMA Controller helps peripheral devices to perform data transfer directly with memory without the involvement of the CPU, thereby increasing the data transfer speed.

A/D converter is the analog to digital converter, which converts analog signals into digital information for further processing. Some microcontrollers will have more than one A/D converter with different resolutions.

Power Management Controller does the power up or power down operations when necessary for the processor.

Capture/Compare unit records the timer values and compares it with the set values by the application. When both values are equal, it triggers an event.

D/A converter is the digital to analog converter that converts processed digital information into analog as required to drive the devices or display.

Cryptography accelerator is used to generate random numbers, hashing and supports for network security.

USART or universal synchronous/ asynchronous receiver/ transmitter reduces the external hardware interface and supports for serial data transmission to facilitate long distance communication.

Timer is a peripheral used to generate time delay and it can also act as a counter to count external events.

Ethernet port reduces hardware complexity and supports network based communications like internet and intranet oriented data transfer etc.

IrDA transceiver supports short distance communication.

Real Time Clock helps to monitor round the clock events to be controlled.

A Multiply accumulate circuit facilitates fast multiplication of data and also multiplication followed by summing.

PWM is the pulse width modulation channel whose pulse width can be varied to execute various applications such as motor control, audio generation etc.

Joint Test Action Group (JTAG) interface provides support for programming and debugging the processors.

Filters may be high pass or low pass and are used to produce high frequency or low frequency signals required for applications.

Switch Mode Pump is a boost converter used for voltage conversion i.e. from low voltage to high voltage.

SPI is a serial peripheral interface which supports for synchronous serial communication.

Periodic Interrupt Controller is an important peripheral to interrupt normal execution of an event in case the processor needs to attend to an emergency request.

Universal Serial Bus (USB) supports multiple device access.

Phase locked Loop (PLL) is the clock source for other required peripherals in the processor.

I2C bus is the two wire serial bus that supports the processor to communicate with a large number of external peripherals.

Spectrum Spreader is an important peripheral to reduce the Electro Magnetic Interference (EMI) problem that must be reduced for final product certification.

Clock doubler allows the user to use a crystal oscillator, which can produce half the value of frequency required for normal operation of the processor since it doubles it internally. This feature helps reduce power consumption and radiated emission.

Watchdog timer is another important peripheral used to monitor the operation of processor and reset it due to malfunction.

Quadrature decoder helps to receive the input from the encoders.

1.5.3 Memory

The embedded processor chip has two types of memory - program memory and data memory. The capacity and operating speed of the memory varies based on the manufacturer, version of the chip as well as the application requirement. Program memory is called non-volatile memory since the data is protected even after power shut down. But data memory is called Volatile memory and it loses its data during power shut down. For example, Read Only Memory (ROM) is non-volatile program memory while Random Access Memory (RAM) is data memory.

Read Only Memory (ROM)

ROM is used to store data permanently in an embedded system. Therefore, it is used to store the application program for which the embedded system is designed. The CPU of the embedded processor can access and read this memory to execute the application. The program stored in the ROM cannot be modified since it is protected. It can only be modified by the respective application developer. The ROM can be internal or external to the microcontroller. The external interfacing ROM capacity depends on the address bus width of the microcontrollers. Based on the technology age, manufacturer, and version of the chip, following are the types of ROM in the embedded processors used for program storage.

Mask ROM of an embedded processor chip is programmed during the fabrication process and cannot be reprogrammed. Since they are application specific processors, their application is decided prior to fabrication. The cost of the Mask ROM processor chip is also less for mass production.

PROM or Programmable ROM is known as One Time Programmable (OTP) memory since it can be programmed only once by PROM programmer or PROM burner. The application programme can be stored only once in the embedded processor chip with PROM and cannot be reprogrammed or modified. This type of processor chip can be used for specific applications.

EPROM or Erasable Programmable ROM was introduced in the market to overcome the disadvantages of PROM. The EPROM can also be programmed like PROM using the programmer. But the programme can be erased and reprogrammed more than once. The memory can be erased entirely or partially by exposing it to ultra violet (UV) rays. The UV rays can enter the die through the window provided for the memory. The application program in the EPROM for embedded processor chip can be modified by reprogramming it. Therefore, an processor chip with EPROM in the market is unprogrammed, allowing it to be programmed as per user applications.

EEPROM or Electrically Erasable Programmable ROM can be programmed and reprogrammed more than once just like an EPROM. However, since electrical input is used to erase the contents of the memory, individual bytes can be erased instead of entire memory. Once the information is written inside the memory, it remains there until it is erased and new information is written. EEPROM is used as an internal memory component in a huge number of embedded processors.

Flash memory was introduced for faster erasing and reprogramming. The information can be erased in fraction of seconds and hence the name flash. The memory can be written or erased as sectors or blocks instead of bytes. To program flash memory, flash programmers are used and like EEPROM, flash memory can also be erased and reprogrammed more than once. It is a popular memory used at present in all embedded processors.

Random-Access Memory (RAM)

RAM or Random Access Memory is used to store data in embedded processor chips. The data is temporary and variable for all applications. The capacity of the RAM is varied among microcontrollers and the data can be entered in a specific memory location. The data can be entered several times in the same location. The data is erased when power is lost since it is volatile memory. The accessibility of the number of memory locations depends on the address bus width of the processor chips. The RAM is classified as Static RAM and Dynamic RAM. The difference between the two memories lies in the data existence or data life in the memory cell.

Static RAM (SRAM) has the better life time for data than Dynamic RAM (DRAM). The static RAM uses flip-flop in its cell to store data but dynamic RAM uses capacitor. Since the capacitor leaks charge, it has to be refreshed periodically to hold the data. Hence the SRAM is faster than DRAM, but expensive.

RAM (random access memory), commonly referred to as *main memory*, is memory in which any location within it can be accessed directly and whose content can be changed more than once (the number depending on the hardware).

1.5.4 Timer

Timer is a peripheral which is used to create accurate time delay between two events. It is a register and counts the number of clock cycles. The number of registers and their size varies depending on the manufacturers. Commonly, the size of the registers is 8/10/16 bits. Therefore, the count starts from 0 and reaches the maximum value of 255/1023/65535 respectively. Then the counting again begins from 0. During this period, the timer overflows and generates an interrupt. The timer peripheral can also be used as a counter and it can count the external events. When the count reaches the desired value, interrupt can also be generated.

1.5.5 Data and Address bus

Data bus is used to carry the information between the CPU and memory or between CPU and I/O devices. Hence it is bidirectional and its width depends on the data length of the processor/controller. For example, in an 8 bit processor, the data length is 8 bit and the data bus width is also 8.

Address bus is used to identify the location of the memory since it carries the address bits where the data is stored. It is unidirectional and its width indicates the capability of addressing a number of locations. For example, if the address bus width 16, it means it can able to identify 2^{16} locations (i.e.) 65,536 locations.

In some of the processors/controllers, a single bus serves as both data and addresses bus and is called multiplexed bus.

There is another bus called Control bus, which is used to control the overall data transfer activity especially the time synchronisation between the events.

1.5.6 I/O interfacing concepts

The Input/output (I/O) interfacing connects peripheral devices with microcontroller using wireless techniques or wired media to perform data transfer between them. Generally, the I/O devices are interfaced with embedded processors using either memory address space called memory mapped I/O or I/O address space called peripheral mapped I/O or directly. Typical microcontrollers have parallel ports and serial ports through which the I/O devices can be interfaced. The I/O handling capability of a microcontroller is one of the important criteria to select that microcontroller for an application. Simple devices like LED, LCD, switches and keypad can be interfaced directly through the parallel port of a microcontroller and some of the I/O devices require a specific circuit to interface with it. Another

important criterion is the speed and time synchronization between the microcontroller and I/O devices. The data transfer rate of the I/O devices is usually several times lower than the microcontroller and hence the suitable mechanism has to be adopted to match the speed and time to avoid the data loss.

Memory mapped I/O

It is essential to interface the required I/O devices with embedded processors to complete the application design. One of the important devices to interface, especially with the microprocessor, is memory. Memory is a group of registers and data stored in the individual register and accessed by its address. The address is in the form of binary numbers and can be identified by the address bus of the microprocessor. The width of the address bus varies depending on type and manufacturers. As a result, the memory accessing capacity of the microprocessor is also varied. Similarly, the I/O devices can also be identified and accessed by its address from the memory space. This technique is called as a memory mapped I/O technique. But microcontrollers have parallel and serial I/O ports through which the I/O devices can be interfaced. The MSP430 also follows the memory mapped I/O technique since their ports acts as a register. The entire port or else the individual port pin can also be configured as input or output.

I/O mapped I/O

The peripheral devices or I/O devices interfaced with the processor can also be accessed by a separate device address. This technique is called I/O mapped I/O technique. Separate instructions are available in the instruction set of the processor to access the I/O devices that make use of this technique.

1.6 CISC Vs RISC design philosophy

Embedded system development can be based on either CISC or RISC philosophies. CISC is Complex Instruction Set Computer or Code and RISC is Reduced Instruction Set Computer or Code. CISC machines have a large number of instructions and in most cases, each instruction is designed for a single purpose. Hence it is time consuming operation. In RISC machines, the number of instructions is less and the instructions are also multipurpose. The instruction execution speed of RISC machine is greater than CISC machine. RISC machines are also designed using pipelined architecture since in a single operation cycle, one instruction is fetched, another instruction is decoded and yet another instruction is executed.

Basically embedded system design primarily falls in the following two categories.

- 1) Primary cost and secondary performance.
- 2) Primary performance and secondary cost.

In earlier days, designers used the CISC machines to fulfil the first category since the cost of RISC machines was higher in those days and was effectively used for soft real time systems. The RISC machine is used to fulfil the second category and falls into the hard real time systems. But the recent RISC machines are cost effective and used for a variety of applications.

1.7 Von-Neumann Vs Harvard architecture

The architecture of embedded processors is classified as Von-Neumann or Harvard. The accessibility of program memory and data memory of the embedded processors depends on their architecture design. Earlier microprocessors were designed using Von-Neumann architecture and most of the microcontroller families are designed using the Harvard architecture. But the MSP430 is designed using the Von-Neumann architecture.

Von – Neumann architecture is also referred to as Princeton architecture and its design is based on the single memory sharing system shown in figure 8. The same memory is shared between both program and data and a common bus exists for the address and data bus, which is used to carry information from both of the memory portions. This time consuming architecture is also highly complex in nature.

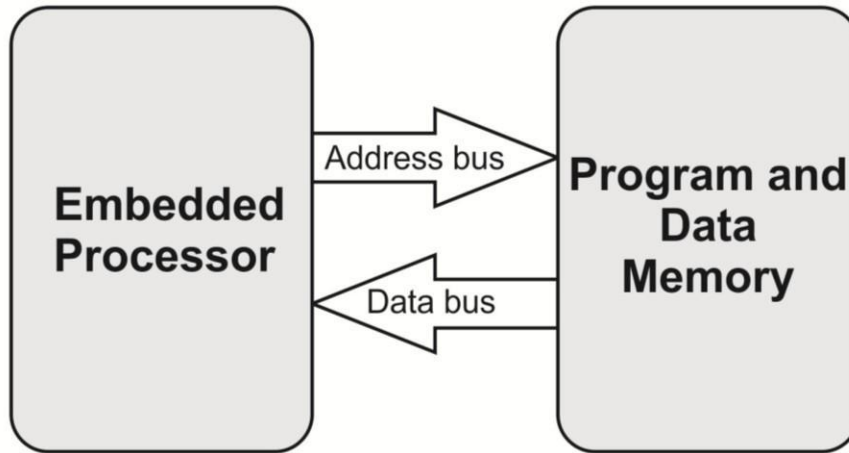


Figure 8: Von-Neumann architecture

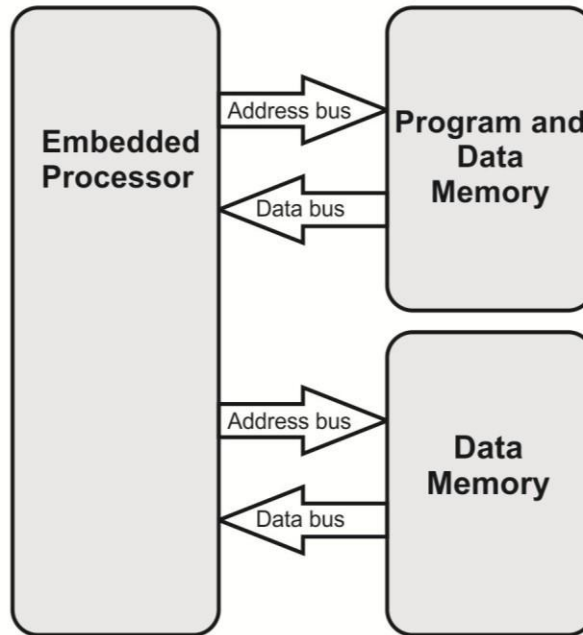


Figure 9: Harvard architecture

Instead of single memory sharing between program and data, the Harvard architecture design has separate memories for program and data as shown in figure 9. Both the memories are accessed using separate address and data busses. In this simple architecture, the data processing time is reduced since the program and data are fetched simultaneously from the memories.

1.8 Choosing a Microcontroller for an Embedded System

Microcontrollers can be selected for the applications based on the following criteria

- Complexity free architecture of the processor i.e user friendly
- Adequate number of internal peripherals required for the application
- Low power consumption and price
- Compliance with industrial standards
- Compatibility with the working environment
- Frequent availability in the market
- Availability of hardware, software development and debugging tools for the processor.
- Sufficient Human resource to support the development

1.8.1 MSP430X5xxx Series Microcontroller

As mentioned in the previous section, one significant criterion for embedded system application is low power consumption. This is because the requirement for battery operated portable products is effectively increasing in all engineering fields. Since the heart of the embedded system is a processor chip, its power consumption is an important consideration.

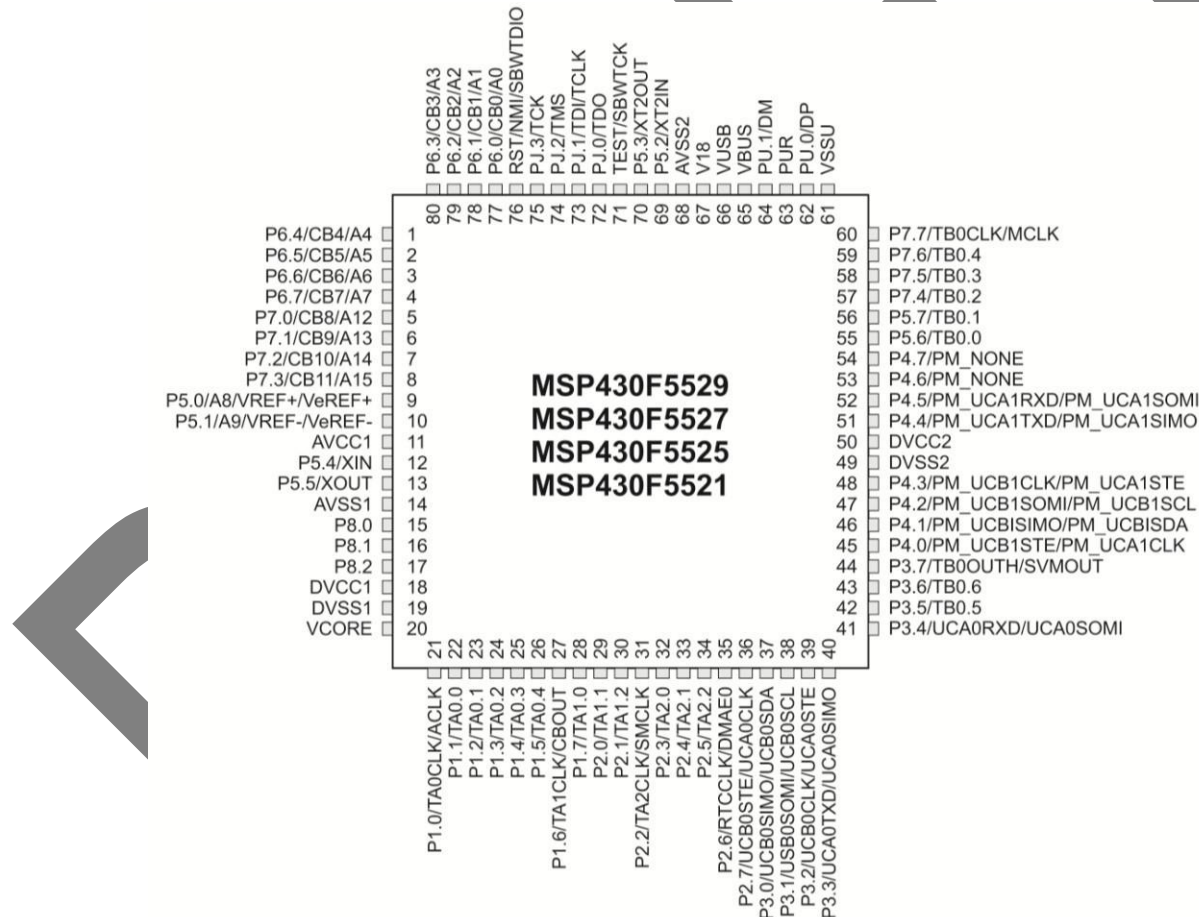


Figure 10: Pin configuration of MSP430X5XXX series microcontroller (Courtesy: TI MSP430F5529 data sheet)

Therefore, Texas Instruments MSP430 Microcontrollers are designed as low power microcontrollers since they are intended for use in various kinds of applications as well as product development. MSP stands for Mixed Signal Processor and its architectural low power modes support for the extended battery of portable products. The digital controlled oscillator (DCO) gives consent to the controller to wake up from the low power modes and enter in to the active mode within 3.5 μ s typically. Another significant criterion for the selection of processor chip is the support for program coding efficiency. Since the MSP 430 is a 16 bit RISC machine designed with 16 bit registers and constant generators, its coding efficiency is highly enhanced. The bootstrap loader (BSL) facility of the controller aids to program the memory using UART interface. The MSP430 memory can also be programmed using JTAG interface since MSP430 family supports the standard JTAG interface. Apart from these standard facilities, MSP430 memory can also be programmed using SPY – Bi – Wire interfacing technique. The figures 10 and 11 show the Pin configuration of MSP430x5xxx series such as MSP430F5529, MSP430F5527, MSP430F5525 etc and its block diagram respectively. The MSP430x5xxx series microcontrollers are available in ball grid array (BGA) as well as low profile quad flat (LQFP) packages for the application development. The Table 1 shows the nomenclature of MSP430F5529.

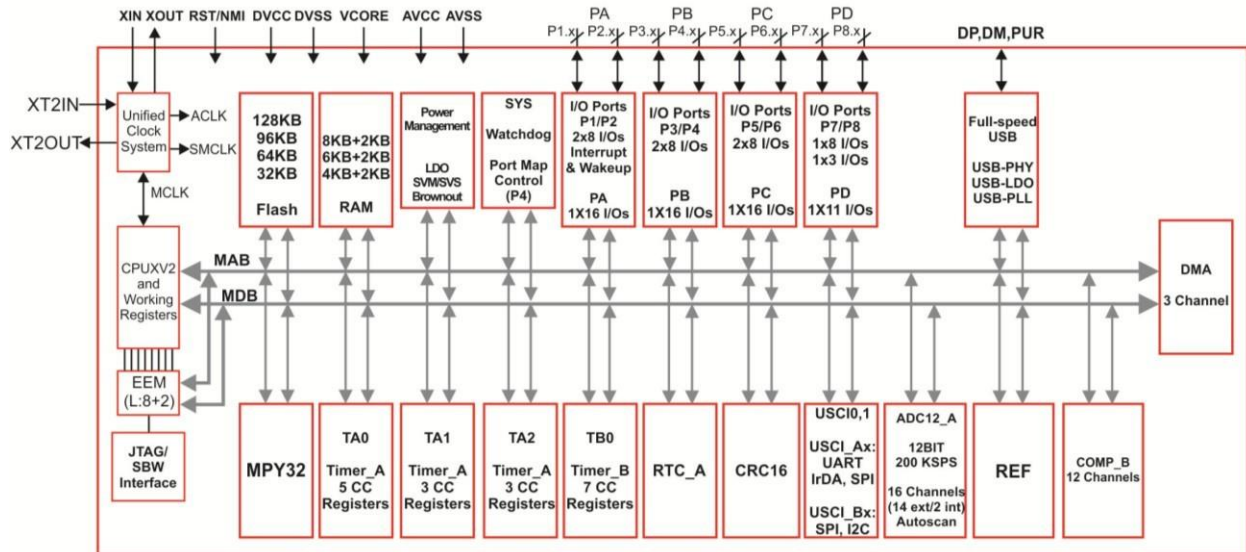


Figure 11: Block diagram of MSP430F5529 (Courtesy: TI MSP430F5529data sheet)

Device Name	Specification
MSP	Processor family
F	Device type
5	Series (2 series – up to 16 MHz,
529	Feature set

Table 1: Nomenclature of MSP430F5529

1.8.2 Address Space of MSP430F5529-

The figure 13 shows the memory management of MSP430F5529 microcontroller which is designed based on the Von – Neumann architecture. Hence the common memory is shared by flash, ROM, RAM, peripheral modules and special function registers

00FFFFh	Interrupt Vector
00FF80h	
0243FFh	Code Memory
004004	
0043FFh	RAM
002400h	
0023FFh	USB Ram
00IC00h	
0091FFh	Information Memory
001800H	
0017FFh	Boot Ladder Memory
00I000h	
000FFFh	Peripherals
0h	

Figure 13: Memory map of MSP430F4152 microcontroller

The memory mapping i.e. address space allotment varies based on the part number. The flash portion starts at the memory address 00000h and extends till 0FFFFh including the interrupt vector table. The boot ROM has memory address allocation between 1000h and 17FFh. The memory locations between 2400h and 43FFh are allotted for RAM. The address locations 0h to 0FFFh are shared by 8 and 16 bit peripheral modules. Based on the memory accessing capability of the address bus of the processor, the memory capacity of the processor varies. Therefore, the start and end addresses of the individual portions of the memory are also altered among the processors. The word or byte can be stored in the code memory as well as RAM. The 16 bit peripheral modules can be accessed by word (16 bit) instructions and the 8 bit peripheral modules can be accessed by byte (8 bit) instructions. When the 8 bit instruction is used for 16 bit peripheral module, only the even addresses are allowed to access and high byte is considered as 0.

1.9 On-chip peripherals and Register sets of MSP430F5529

1.9.1 CPU Registers

The CPU of the 16 bit RISC machine is designed with **16 registers** as shown in figure 14 to enhance the coding efficiency as mentioned in the earlier section. The registers are 16 bit registers and are named as R0, R1, R2..... to R15 respectively. The registers support to reduce the instruction execution time. The registers R0, R1, R2 and R3 have dedicated functions such as Program counter, Stack pointer, Status register /Constant generator 1 and Constant generator 2 respectively. The remaining registers R4 to R15 are used as general purpose working registers.

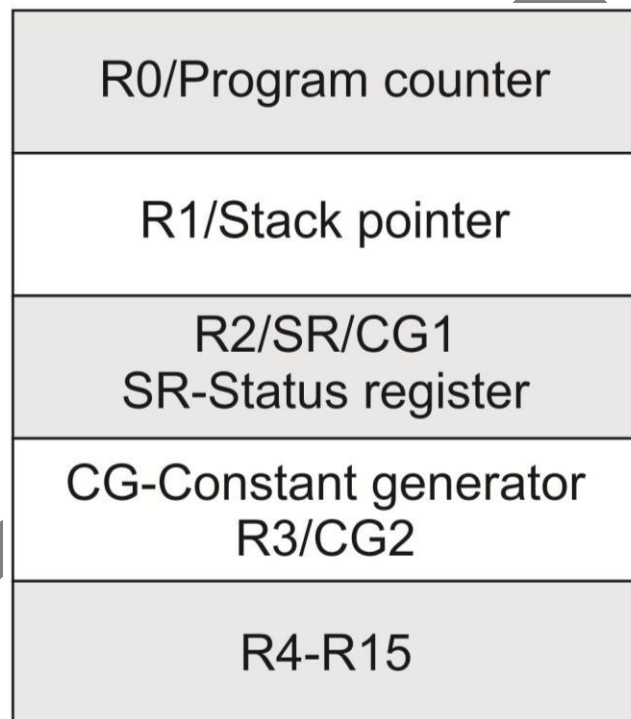


Figure 14: Memory map of MSP430F4152 microcontroller

The next instruction to be executed is pointed by the **16 bit program counter**. Since the even number of bytes in the memory is used by the instructions, program counter is pointed and incremented accordingly. The **stack pointer** holds the address of the next instruction to be executed when it is returned from the interrupt or subroutines. Except the program flow instructions, all the operations are carried out as register based operations in association with seven addressing modes for source operand and four addressing modes for destination operand. The **status register** is used as a source or destination register to register addressing mode operations and the constant generator is supported by other addressing modes. The figure 15 shows the bit pattern of the status register.

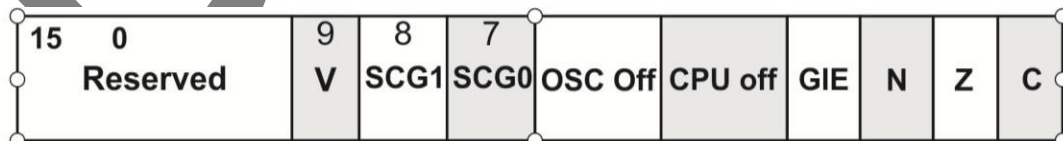


Figure 15: Status register of MSP430F5529 microcontroller

V: Overflow bit. When the result of arithmetic operation overflows, this bit is set.

SCG1: System clock generator 1. Turns off the SMCLK when this bit is set.

SCG0: System clock generator 0. Turns off the DCO when this bit is set.

Osc off: Oscillator off. Turns off the LFXT1 crystal oscillator when this bit is set.

CPU off: CPU off. Turns off the CPU when this bit is set.

GIE: General interrupt enable. Enables maskable interrupts when this bit is set and Vice versa.

N: Negative bit. When the result of arithmetic operation is negative, this bit is set.

Z: Zero bit. When the result of arithmetic operation is zero, this bit is set.

C: Carry bit. When the result of arithmetic operation produces carry, this bit is set.

The **constant generators CG1 and CG2** (Registers R2 & R3) are used to generate the six commonly used constants without the need of the additional program code. It also reduces the code memory requirement to retrieve the constants.

Register	Constant	Description
R2	-----	Register mode
R2	0	Absolute address mode
R2	00004h	+4,bit processing
R2	00008h	+8,bit processing
R3	00000h	0,word processing
R3	00001h	+1 bit
R3	00002h	+2,bit processing
R3	0FFFFh	-1,word processing

Table 2: Values of Constant Generators

1.9.2 Memory

MSP430F5529 microcontroller is provided with 16 bit address bus and hence it is capable of accessing $2^{16} = 65,536 = 64$ Kbytes memory. Therefore, the memory address starts at 0x0000h and ends at 0xFFFFh. The flash portion has a number of segments in the memory and can be programmed using any of the methods mentioned in section 1. The 8 bit data or 16 bit data can be written into the flash memory. The information memory has four segments (A to D) and each segment has 64 bytes each. Flash memory can be erased in one step or individual segments can be erased. The individual segment of the information memory can be erased or can be erased as a group.

1.9.3 Other peripherals

Commonly, the term peripheral refers to an input output device. The peripheral devices are physically connected to the CPU through the address, data and control buses. The instructions of the processor are used to access those devices.

1.9.4 Oscillator and system clock

The Frequency locked loop (FLL+) is dedicated to the clock system function of the MSP430F5529 microcontroller. This clock module has a 32kHz watch crystal oscillator, an internal very low power low frequency oscillator, an internal digitally-controlled oscillator (DCO), and an 8-MHz high-frequency crystal oscillator (XT1). Along with a digital modulator, the FLL+ module stabilizes the Digital controlled oscillator (DCO) frequency to the watch crystal frequency. The FLL+ module is used to provide the

Auxiliary clock (ACLK), Main clock (MCLK), Sub main clock (SMCLK), ACLK/n where n=2, 4, or 8. This module is designed to achieve low cost and low power consumable devices.

1.9.5 Brownout, supply voltage supervisor

Generally, normal program execution of the microcontroller is disturbed due to low power supply voltage. Hence, a mechanism is needed to hold the microcontroller in reset mode till the power supply returns back to the normal level. Such effort is carried out by the brown out reset. This supply voltage supervisor monitors the input supply voltage and if the level of the voltage is lower than the user selected level, it holds the microcontroller in reset mode. The CPU execution is started after the brown out circuit released the reset.

1.9.6 Digital I/O

The device has seven I/O ports named P0 to P7. Out of these, six ports i.e. P0 to P6 are 8 bit ports and P7 is 7 bit port. Individual programmable port pins can be configured as an input or output. The ports P1 and P2 are provided with the interrupt capability. All the bits of P1 and P2 are individually enabled for an interrupt for falling and rising edge input signals. All instructions are supported to independent read/write port control registers.

1.9.7 Watchdog timer

The watchdog timer is an important peripheral which resets the microcontroller during any problem of program execution such as program malfunctioning caused by the misinterpretation of data or supply voltage disturbance. It is a free run counter and generates an automatic reset after a particular period of time if not disabled. Hence the program instruction should reset this timer often to prevent from resetting the microcontroller.

1.9.8 Basic Timer1 and Real-Time Clock (RTC)

Three independent 8-bit timers are designed in basic timer1 and are cascaded to form a 16-bit timer/counter if needed. Using the software instructions, timers can be read and written. The Basic Timer1 is also extended to provide an integrated real-time clock (RTC). The Real-Time Clock (RTC) is used to count seconds, minutes, hours, day, date and year with leap-year compensation.

1.9.9 Timer_A3 and Timer_A5

Timer_A3 is a 16-bit timer with three capture/compare registers and Timer_A5 is a 16-bit timer with five capture/compare registers. Both of the timers can also be used as counters. Timers also support PWM outputs and interval timing. Usually the timers of the microcontroller have interrupt capabilities and the timer rolls over period is considered as an interrupt. Similarly, the interrupts can be generated from the counter on overflow conditions and from each of the capture/compare registers.

1.9.10 Comparator_A+

The comparator_A+ module is designed to monitor external analog signals as well as support for precision analog-to-digital conversions and battery voltage supervision.

1.9.11 ADC12_A

The ADC12_A module is a 12-bit analog-to-digital convertor that is implemented by a 12-bit successive approximation register (SAR), sample select control, reference generator, and a conversion and control buffer. The convertor also allows the converted sample to be stored without any CPU intervention.

1.9.12 Communication Protocols

The device also supports UART, Full speed USB, IrDA, SPI and I2C Logic communications.

1.10 Addressing Modes

Addressing mode refers to how the operation is performed on data. There are several addressing modes in MSP430F5529 microcontroller, out of which seven addressing modes belong to the source operand and four addressing modes belong to the destination operand. Following are the addressing modes with examples:

1) Register addressing mode

Ex. MOV R11, R12

The instruction copies the content of register R11 to R12.

2) Indexed addressing mode

Ex. MOV 3(R8), 4(R6)

Let, R8 = 01050h and R6 = 01080h before execution of an instruction.

Then $01050h+3 = 01053h$ and $01080h+4 = 01084h$ since 3 & 4 in the instruction are indexes. Therefore the content of address 01053h is moved to the address 01084h.

3) Symbolic addressing mode

EX. MOV X(PC), Y(PC)

The content of the source address (PC+X) is moved to the destination address (PC+Y). Hence the program counter is altered and continued with next instruction.

4) Absolute addressing mode

EX. MOV &ED, &TO

Let, ED and TO be user defined labels. The instruction copies the content of the source address ED to the destination address TO.

5) Indirect register addressing mode

Ex. MOV.B @R10, 0(R11)

Let, R10 = 0FA55h and R11 = 002A5h. The content of 0FA55h is considered as an address from which the content is copied to 002A5h.

6) Indirect auto increment addressing mode

Ex. MOV @R10+, 0(R11)

Let, R10 = 0FA55h and R11 = 010A5h. The content of 0FA55h is considered as an address from which the content is copied to 010A5h. Then R10 value 0FA55h is incremented by 1 for byte operation and incremented by two for word operation.

7) Immediate addressing mode

Ex. MOV #45h, TO

The instruction copies the immediate constant 45 to the destination address specified by TO + it's content.

Let TO = 0FF16h and its content is 01192h. Therefore the constant 45 is copied to the address 010A8h which is generated by 0FF16h + 01192h.

1.11 Instruction format

The MSP430x5xxx series instructions are designed using three core-instruction formats:

- Dual operand
- Single operand
- Jump

All the single operand and dual operand instructions are used with the extension .B or .C. The instructions may be byte or word instructions. The byte data or byte peripherals are accessed by byte instructions and word instructions are used to access word data or word peripherals. When no extension is used, the default instruction is a word instruction.

The following fields are used to define the source and destination of an instruction:

src The source operand defined by As and S-reg
 dst The destination operand defined by Ad and D-reg
 As The addressing bits responsible for the addressing mode used for the source (src)
 S-reg The working register used for the source (src)
 Ad The addressing bits responsible for the addressing mode used for the destination (dst)
 D-reg The working register used for the destination (dst)
 B/W Byte or word operation

1.11.1 Format 1 – Double operand instructions

The figure 16 shows the format of the double operand instruction

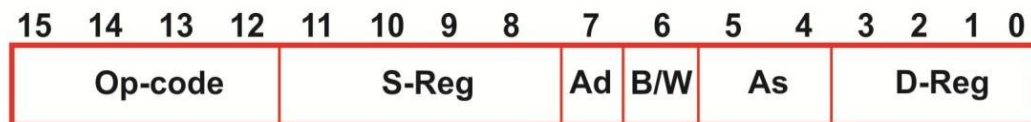


Figure 16: Format of double operand instruction

Ex. MOV(.B) src, dst

The source and destination are mentioned in the instruction and when this instruction is executed, the content of source is copied to the destination.

1.11.2 Format 2 – Single operand instructions

The figure 17 shows the format of the single operand instruction



Figure 17: Format of single operand instruction
(Courtesy: TI MSP430xx5xx user guide)

Ex. RRC(.B) dst

In this instruction format, only the destination is mentioned and when this instruction is executed,

content of the carry flag is copied to the most significant bit (MSB) of destination register, MSB to the next bit till the least significant bit (LSB).

1.11.3 Format 3 – Jump instructions

The figure 18 shows the format of the jump instruction



Figure 18: Format of jump instruction (Courtesy: TI MSP430xx5xx user guide)

Ex. JEQ/JZ Label

When this conditional jump instruction is executed, the program control is transferred to the label mentioned for the satisfied condition. Otherwise the instruction followed by this is executed.

1.12 Instruction set

The MSP430 series microcontrollers are designed with upward compatible 51 instructions. These instructions are divided in to two categories i.e. core instructions and emulated instructions. The core instructions are 27 and have op-codes that are decoded by the CPU. The emulated instructions are 24 and have no op-codes to decode, but are easier to develop codes. The emulated instructions are replaced as an equivalent core instruction by an assembler. The Table 3 shows the instruction set of MSP 430 series microcontroller.

Mnemonic		Description		V	N	Z	C
ADC (.B)†	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD (.B)	src, dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC (.B)	src, dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND (.B)	src, dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC (.B)	src, dst	Clear bits in destination	.not.src .and. dst → dst	-	-	-	-
BIS (.B)	src, dst	Set bits in destination	src .or. dst → dst	-	-	-	-
BIT (.B)	src, dst	Test bits in destination	src .and. dst	0	*	*	*
BR†	dst	Branch to destination	dst → PC	-	-	-	-
CALL	dst	Call destination	PC+2 → stack, dst → PC	-	-	-	-
CLR (.B)†	dst	Clear destination	0 → dst	-	-	-	-
CLRC†		Clear C	0 → C	-	-	-	0
CLRN†		Clear N	0 → N	-	0	-	-
CLRZ†		Clear Z	0 → Z	-	-	0	-
CMP (.B)	src, dst	Compare source and destination	dst - src	*	*	*	*
DADC (.B)†	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD (.B)	src, dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC (.B)†	dst	Decrement destination	dst - 1 → dst	*	*	*	*
DECD (.B)†	dst	Double-decrement destination	dst - 2 → dst	*	*	*	*
DINT†		Disable interrupts	0 → GIE	-	-	-	-
EINT†		Enable interrupts	1 → GIE	-	-	-	-
INC (.B)†	dst	Increment destination	dst + 1 → dst	*	*	*	*
INCD (.B)†	dst	Double-increment destination	dst+2 → dst	*	*	*	*
INV (.B)†	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
JEQ/JZ	label	Jump if equal/Jump if Z set		-	-	-	-
JGE	label	Jump if greater or equal		-	-	-	-

JL	label	Jump if less		-	-	-	-
JMP	label	Jump	PC + 2 x offset → PC	-	-	-	-
JN	label	Jump if N set		-	-	-	-
JNC/JLO	label	Jump if C not set/Jump if lower		-	-	-	-
JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
MOV (.B)	src, dst	Move source to destination	src → dst	-	-	-	-
NOP†		No operation		-	-	-	-
POP (.B)†	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	-	-	-	-
PUSH (.B)	src	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
RET†		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
RETI		Return from interrupt		*	*	*	*
RLA (.B)†	dst	Rotate left arithmetically		*	*	*	*
RLC (.B)†	dst	Rotate left through C		*	*	*	*
RRA (.B)	dst	Rotate right arithmetically		0	*	*	*
RRC (.B)	dst	Rotate right through C		*	*	*	*
SBC (.B)†	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC†		Set C	1 → C	-	-	-	1
SETN†		Set N	1 → N	-	1	-	-
SETZ†		Set Z	1 → C	-	-	1	-
SUB (.B)	src, dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC (.B)	src, dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		-	-	-	-
SXT	dst	Extend sign		0	*	*	*
TST (.B)†	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR (.B)	src, dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

† Emulated Instruction

**Table 3: Instruction set of MSP430 series microcontroller
(Courtesy: TI MSP430x2xx user guide)**

DRAFT

1.13 Variants of MSP430 family – Comparative Study of MSP430x2x, MSP430x4x and MSP430x5x:

MSP430 family of microcontrollers come with two more variants of microcontrollers other than MSP430x5xx i.e., MSP430x2xx and MSP430x4xx. These controllers carry almost the same features as MSP430x5xx with minor variations targeting some specific application areas. The comparison between all the three members is shown below:

Features	MSP430x2x	MSP430x4x	MSP430x5x
1) CPU Clock Speed	16 MHz	16MHz	25MHz
2) Memory	120KB Flash 8KB SRAM	120KB Flash 8KB SRAM	512KB Flash 66KB SRAM
3) Timers	2 16-bit timers	2 16-bit timers	4-16 bit timers
4) ADC	Slope to 24-bit sigma delta	Slope to 24-bit sigma delta	8ch 10-bit SAR to 16ch 12-bit SAR
5) Multiplier	16x16 multiplier	32x32 multiplier	32x32 multiplier
6) Temperature sensor	No	No	Yes
7) GPIOs	48	80	87
8) Application Areas	Metering, Transportation, Military, Space, Security and safety And consumer electronics	Metering and Mobile medical devices	Portable medical, Test and measurement and consumer electronics

The above comparison table highlights the important differences between various MSP series devices. TI has an MSP devices portfolio starting from basic Launchpad G2 to FRAM based devices.

MSP430x5xx comprises one 10 bit and one 12 bit SAR ADC giving higher resolution and more analog channels than MSP430x2x/4x controllers which use slope type or sigma delta ADC. MSP430x5xx also has a higher memory of 512KB flash than 128KB of MSP430x2xx/4xx controllers, thus increasing the code storing capacity of the controller. It also has 4 16 bit timers as compared to only 2 timers in 2xx and 4xx series controllers. The only feature where 4xx series can compete with 5xx devices is the multiplier as both of them carry 32 bit multiplier. All of the above features make MSP430x5xx controller a heavyweight in industrial application, portable medical devices and consumer electronics products.

Texas Instruments has come out with two Launchpad evaluation kits based on the MSP430x2xx and MSP430x5xx microcontrollers.

MSP-EXP430G2 evaluation kit consists of an MSP430G2553 device by default and also supports other MSP430G2xxx and MSP430F5xxx devices. The other Launchpad evaluation kit MSP430F529 succeeds the EXP430G2 in terms of advanced features and the use of flash memory as compared to flash value line as used by G2 series devices.

Below is the pin diagram for MSP-EXP430G2533

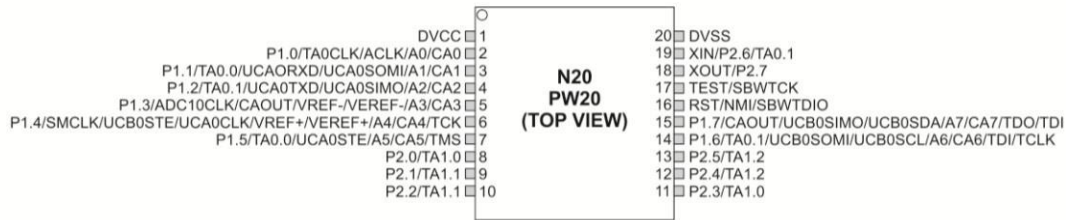


Figure 19: Pin Diagram of MSP430G2553 pin diagram
Source: TI's MSP430G2553 datasheet

MSP-EX430G2 kit has 14 easy accessible pins distributed in two ports. Port 1 has 8 pins and port 2 has 6 pins which acts as general purpose I/Os. It also supports a UART interface, push buttons, LEDs and a USB connection for programming the device.

The block diagram for MSP430F5529 is shown below:

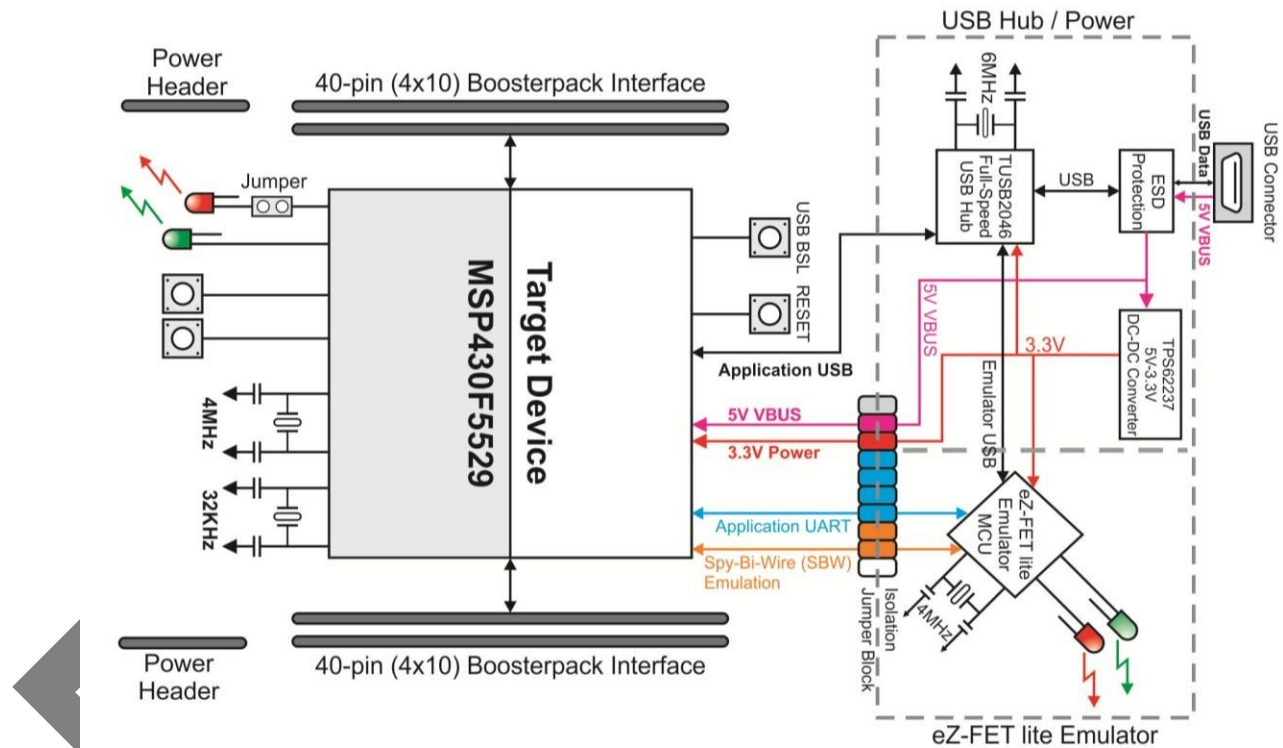


Figure 20: Block Diagram of MSP430F5529 launchpad evaluation kit
Source: MSP430F5529 Launchpad Evaluation kit user guide

The MSP430F5529 launchpad evaluation kit consists of a 40 pin booster pack interface , thus providing more general purpose I/O pins as compared to G2 development kit. It also consists of a power header , push buttons, LEDs and a USB connection for programming purpose. Except these, it also has an emulator which is connected with the device and is majorly used for debugging.

1.14 Sample Applications of Embedded System on MSP430 Microcontroller

Example: 1

Skin Care Teleconsultation Embedded System for Rural and Remote Areas:

People in rural and remote areas are found struggling to access timely medical treatment. Skin diseases are the most common problems that people face in rural and low-income areas situated in the outskirts of the city. The problems include infections, allergies, spots and rashes. Most of those skin problems are due to unhygienic and dusty environment in which these people live. Insects and dust are two main causes for skin problems found out in the analysis. People living in these areas might not want to travel long distances to get routine checkups and screenings. They might also have trouble getting to a hospital quickly in an emergency.

Therefore, there is a need for a cost effective, portable, intelligent Teleconsultation system for the dermatologist to provide the solutions for the above said problems to the people in rural and remote areas, using embedded technology with the support of internet services.

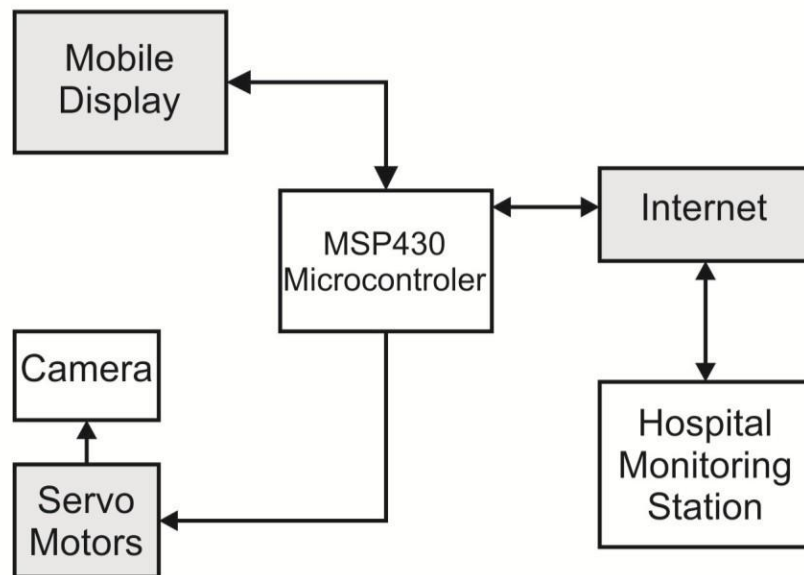


Figure 21: Basic Block of Skin Care Teleconsultation Embedded System

The block diagram of the skin care teleconsultation embedded system is shown in figure 21 and it has a Microcontroller, a Small camera, Servo Motors and Communication devices. This device can be installed in rural areas to provide consultation services to needy people with minimum cost and time. The patients can show their infections in front of the camera fixed on the servo motors and it can be transferred to the doctors in the cities through internet services with the support of a processing device, the Microcontroller. The doctors can also control the position (left, right, up and down movements) of the camera by accessing the servo motors on the patient side from their premises to examine the infected parts. Telephone/mobile phone connectivity can be established from the equipment side to communicate with the monitoring station. Hence, the doctor can explain the problem and prescribe medicines to the patient over this medium. Database can also be created to maintain records of the patients in the monitoring station to monitor every consultation for further progress.

Example: 2
Embedded System Based Automatic Irrigation System with Wireless Monitoring

Water is precious and every drop is turning to be a very valuable resource in the present scenario. Development of micro irrigation systems and better water resource management are essential for improving agricultural productivity in the country. In the past, canal irrigation system led to over irrigation the land. Instead, maintaining moisture at just adequate levels allows for more efficient utilization of water, enhancing agricultural productivity. Therefore, it requires a device that could automatically control water supply based on the soil condition. The device would be installed in the agricultural field where it would be subject to adverse conditions and may or may not be monitored by human personnel at regular intervals. Hence it has to be reliable, consume less power and also be cost efficient. In short, an embedded system is the only desirable solution and has to be developed and dedicated to the agricultural field.

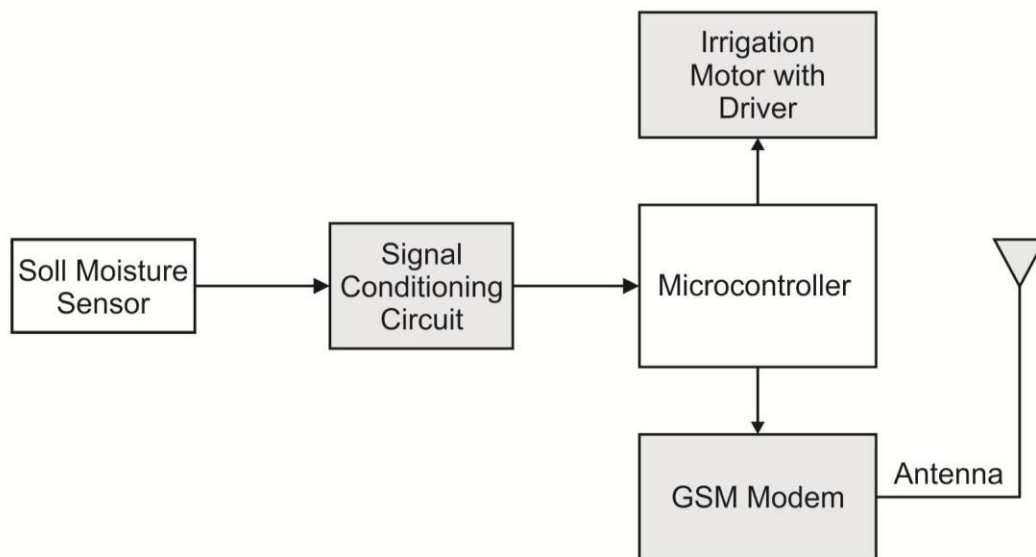


Figure 22: Basic Block of Embedded System Based Automatic Irrigation System

The figure 22 shows the basic block of the automatic irrigation system. This simple block consists of a soil moisture sensor, signal conditioning circuit, Microcontroller, Motor with driver and GSM modem. Soil moisture sensor produces the output signal based on the wet and dry conditions of the soil and the output is tuned by the signal conditioning circuit to the required level. Microcontroller processes the data from the signal conditioning circuit, takes a decision to drive the irrigation motor and transfers the status of the motor to the farmer's mobile phone with the support of interfaced GSM modem.

1.15 Summary

DRAFT

1.16 Review Questions

DRAFT

1.17 Exercises

DRAFT

Microcontroller Components and Peripherals

By the end of the first chapter of this book, readers were able to gain a basic understanding of embedded systems and its applications. Building on this, the second chapter will cover the configuration process for various components/peripherals of the microcontroller.

Readers will learn about mapping of Peripheral devices with the system as well as various Registers dedicated to the system. They will also understand how the system can handle the Interrupt.

The chapter will also cover low power aspects of MSP430 and help readers study the FRAM as nonvolatile memory and understand its contribution in low power devices.

Topic	Page
2.1. Introduction	32
2.2. Microcontroller Fundamentals for Basic Programming	32
2.3. Peripheral address and ranges	36
2.4. I/O pin multiplexing& its relevance	43
2.5. Pull up/down registers	44
2.6. GPIO control	45
2.7. Interrupts and interrupt programming.....	52
2.8. MSP430 Interrupts.....	54
2.9. Interrupt priority and Interrupt vector table	56
2.10. Interrupt programming	57
2.11 System clocks and Low power modes	66
2.12 Requirements of Low Power embedded system.....	72
2.13. Active vs Standby current consumption	79
2.14. Ferroelectric Random Access Memory (FRAM).....	81
2.15. Summary	87
2.16. Review questions	88
2.17. Exercises	89

2.1. Introduction

Demand for microcontrollers in the design of an effective embedded system is increasing every day. More peripherals with higher capability and low power operative CPU core in the microcontroller chip sets are being used in embedded systems.

A microcontroller interacts in many ways with the real world through its peripherals. Here are some great examples to highlight how peripherals play important role in embedded system: Microcontroller receives inputs from a human through switches. At the same time, it turns external devices on or off through a GPIO peripheral. A microcontroller generates accurate and precise time delay through a timer peripheral. A microcontroller reads the sensor data through ADC channel.

Peripheral devices can be either external or internal to the processor. For example, a printer is an external device to the computer that you connect using a cable, while an optical disc drive typically located inside the computer case but not the processor chip set is also an external peripheral.

Internal peripheral devices can be in-built or integrated with the CPU core of the microcontroller, and are preferred for optimistic design.

The chapter begins by discussing how peripherals are mapped in the memory space of the MSP430 and peripherals with the registers associated them explains how to use those peripherals by programming example. Next, various internal peripherals found in MSP430 such as GPIO, timer, ADC, PWM and other peripherals with low power operation are described. The chapter also covers the usage of peripherals in an embedded system with relevant examples.

2.2. Microcontroller Fundamentals for Basic Programming

2.2.1. Memory Mapped Peripherals

For handling the peripherals, it is necessary to have access techniques so that we can control them. There are two techniques available for the same:

- Memory mapped peripherals
- IO mapped peripherals

There are some basic differences between two.

Memory Mapped Peripherals	IO mapped peripherals
It uses the same available memory address for addressing & accessing IO.	It uses separate address space for addressing IO
Same CPU instruction set are used for accessing both IO and memory	CPU instructions are different for accessing memory and IO.

In memory mapped peripherals, the IO devices are treated as a part of memory. So, the same instructions which are used for memory based operations can be used for controlling the peripheral devices. The memory section needs to be divided for IO mapping and that space cannot be used for physical addressing. Peripheral devices need to monitor the address bus of the system to get instructions and perform accordingly.

In MSP430, memory mapped peripheral techniques are used to address and access IO devices. For the 16 bit peripheral module, the address range available is from 0100h to 01FFh. This memory range is used for accessing word instructions. If byte instructions are used with this section, only even addresses are permissible and the high byte of the provided result is zero. For the 8 bit peripheral module, the address range available is from 010h to 0FFh. This memory range is used with byte

instructions. If read access of word instruction is provided in byte module, then we get an unpredictable data in the high byte as a result. If word data is written to byte module, only low byte is written into peripheral register ignoring the higher byte.

Therefore, it is frequently more practical to take advantage of the benefits of memory-mapped I/O. The advantage of this method is that every instruction which can access memory can be used to manipulate an I/O device.

In this chapter, we provide a detailed description of the registers accessible in memory space. MSP430 has Von-Neumann architecture, which means that one address space is shared with all memory, including special function registers (SFRs), peripheral module registers, Random Access memory (RAM), Flash/ROM and information memory. They are mapped into a single, contiguous address space as shown in Figure below.

The address range and data size of the each memory is shown in the figure. Data size of the some of the memory is word, byte or both.

The memory map is slightly different for each device in the MSP430 series.

	OFFFh	
	Interrupt Vector Table	
Word/Byte	OFFE0h	32 Bytes
Word/Byte	OFFDFh	
	Flash ROM	
	↕	
	↕	
	RAM	
Word/Byte	O200h	
Word Access	O1FFh	
	16Bit peripheral Modules	128 Words
	O100h	
Byte Access	OFFh	
	8Bit peripheral Modules	240 Bytes
	O10h	
Byte Access	OFh	
	Special Function Register	16 Bytes
	O0h	

Figure – 2.1A Memory organization of MSP430x5xxx series

2.2.2. MSP430 x5xx Peripherals:

The Special Function Registers associated with controlling and configuring the peripherals in MSP430x5xxx series are located at memory addresses from 0000h to 0FFFh (1KB of memory space as shown in the table 2.1A). Also, the specific registers for Interrupt enables, Interrupt flags and Enable flags also mapped in this memory space. The register's data size for the peripherals can be 8 bit and 16 bit and are mapped in the memory space.

GPIO

There are up to 9 digital I/O ports (P1 to P9) in MSP430X5xx and up to 11 digital I/O ports (P1 to P11) in MSP430x6xx series. Different MSP430 series contain a different number of ports. Some contain eight I/O lines and some may contain fewer lines. This I/O line is individually used for input or output, and each can be individually read or written. Each I/O line is individually configurable for pullup or pulldown resistors. The Ports can be accessed as a byte wide or can be combined into another Port as word wide.

P1 and P2 referred in the programming to access byte wide and P1 and P2 are paired and referred to as PA in programming to access as word wide. Similarly, P3 and P4, P5 and P6, and so on, are referred as PB, PC, and so on, respectively

P1 and P2 ports have interrupt capability and each pin of the P1 and P2 I/O lines can be individually enabled and configured for rising or falling edge triggering type. Each pin of the P1 has a single interrupt vector address (P1IV) and similarly P2 also has a single vector address (P2IV) but P1 and P2 has different vector addresses.

Timer Peripherals:

MSP430 microcontroller can be used with a wide variety of timer peripherals such as Watch dog timer, Timer basic, Timer A, B and D, and Real Time Clock (RTC). Timer peripherals gets clock from various clock sources available in the controller itself.

A watchdog timer is used to detect programming stuck in a continuous loop of execution and to reset the processor. This is one of the powerful peripherals for protecting programming hang. The watchdog timer is a 32-bit timer that can be used as a watchdog or as an interval timer. The WDT can be used in an application and once it is used, it needs to be cleared so as to not expire the timer value.

MSP430x5xxx has three individual 16 bit timer peripherals namely Timers A, B and D. Each timer supports 4 different modes of operations such as interval timing generation, capture, comparison and pulse width modulation output generation. Additionally, there are multiple capture/compare modules in each timer peripheral. Interrupt may be generated from the counter on overflow conditions or from each of the capture/compare registers. Timer D operates in high resolution mode with a fine clock frequency input, which is up to 16 times the timer input clock frequency.

Real Time Clock (RTC)

Real Time Clock is also a timer based peripheral and it provides a real-time clock and calendar function that can also be configured as a general-purpose counter. It gets the clock from ACLK or SMCLK sources. It provides seconds, minutes, hours, day of week, day of month, month, and year in real-time clock with calendar function. It can be accessed along interrupt feature.

ADC

MSP430 microcontroller is facilitated with three different analog to digital converting (ADC) peripherals -ADC10, ADC12 and SD24. ADC10 and ADC12 have successive approximation type ADCs with the resolution of 10 bits and 12 bits respectively. Each ADC is facilitated with the 16 channel input and the conversion rate maximum of 200-ksps. The clock signal for A to D conversion and for sample and hold operation is given by clock source or by timer. Since an A to D Conversion is done with respect to reference voltage, the reference voltage for ADC10 and ADC12 can be given by internal reference generator module or through external pin and each of the ADC is having internal temperature sensor. Both ADCs are functioning with 4 different conversion modes of operations such as Single channel, repeat single channel, sequence (auto scan), and repeat sequence (repeated auto scan). The ADC10 and ADC12 are designed for low-power applications. When they are not actively converting, the peripheral core are automatically disabled and enabled individually when needed.

ADC10 stores the converted data into an ADC memory register (ADC10MEM) and sets the interrupt flag to generate interrupt. It stores the converted data of each channel among 16 channels into the same memory register for all the conversion modes of operation. So the converted data should be read by programming on each conversion complete. ADC10 comes with a window comparator circuit, which compares the ADC data with predefined low and high value stored in separate registers and sets the corresponding interrupt flags.

ADC12 stores the converted data of each channel into the separate memory register allocated for each channel. The overflow condition occurs when a conversion result is written to any one of the memory registers before its previous conversion result was read and it sets an overflow flag to generate interrupt.

The SD24 peripheral consists of up to eight independent sigma-delta analog-to-digital converters. The converters are based on second-order oversampling sigma-delta modulators and digital decimation filters. The decimation filters are comb type filters with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

DAC12

This peripheral is a 12-bit DAC and it can be configured in 8-bit or 12-bit mode and may be used in conjunction with the DMA controller. When multiple DAC12 modules are present, they may be grouped together for synchronous operation. Since, D to A conversion is done with respect to reference voltage and this can be given to the DAC internally or externally to the peripheral. It supports unsigned binary (uni polar) or twos-complement (Bipolar) data format with right or left justified.

Comparator

The comparator peripheral compares analog input voltages at the + and – input pins itself. If the voltage at + pin is more than the – pin, the comparator output (CBOUT) is high. The comparator can be switched on or off when not in use to reduce power consumption. It supports selectable reference voltage generator, voltage hysteresis generator and reference voltage input from shared reference.

USCI (UART, SPI and I2C)

The USCI module in MSP430x5xxx supports multiple serial synchronous and asynchronous communication modes supported peripherals such as UART, SPI, and I2C.

In asynchronous mode, the USCI module connects the device to an external system or device to device by two external pins TXD and RXD called universal asynchronous receiver transmitter (UART). In MSP430 series these pins are referred as UCxRXD and UCxTXD. It supports for programmable Baud rate (bits per second (bps)), 7 or 8 bit data with odd, even, or non-parity, LSB-first or MSB-first data transmit and receive configuration and multiprocessor communication. It has an independent interrupt capability on receive and transmit complete.

In synchronous mode, the USCI connects the device to an external system by three or four pins such as MOSI (master out slave in), MISO (master in slave out), SCK (serial clock) and SS (slave select) called SPI (serial peripheral interface). In MSP430 series, these pins are referred as UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. It supports for 7-bit or 8-bit data length, LSB-first or MSB-first data transmit and receive, and 3-pin and 4-pin SPI operation. Independent interrupts capability for receive and transmit complete.

The I2C architecture is in compliance to the Philips semiconductor I2C protocol. It supports for 7-bit and 10-bit device addressing modes, multi-master transmitter/receiver mode, and slave receiver/transmitter mode, standard mode up to 100 kbps and fast mode up to 400 kbps.

DMA

The direct memory access (DMA) controller peripheral transfers data from one memory space to another without CPU intervention across the entire address range. For DMA, MSP430 microcontroller has many features in transferring data. For example, it transfers data from the ADC conversion

memory to RAM without CPU intervention. It supports various modes of transfer such as single transfer, block transfer, and burst block transfer with the maximum block size of 64KB. It contains up to eight DMA channels and allows the CPU to remain in a low-power mode, without having to awaken to move data to or from a peripheral.

2.3. Peripheral address and ranges

Peripheral	Name	Memory mapped address space	Device
GPIO	P1 (Port 1)	20h to 26h, 27h and 41h	MSP430X11X
	P2	28h to 2Eh, 2Fh and 42h	
	P3	18h to 1Bh, 10h and 43h	
	P4	1Ch to 1Fh, 11h and 44h	
	P5	30h to 33h, 12h and 45h	
	P6	34h to 37h, 13h and 46h	
	P7	38h,3Ah,3Ch,3Eh, 14h and 47h	
	P8	39h,3Bh,3Dh,3Fh, 15h and 48H	
	P9	08h,0Ah,0Ch,0Eh, 16h	
	P10	09h,0Bh,0Dh,0Fh, 17h	
EEPROM	EEPROM Control	54h	
Comparator	Comparator-A	59h to 5Bh	
Serial Interface	USART0	50h to 52h, 70h to 77h, 00h,02h,04h	MSP430x12xx, MSP430x13xx, and MSP430x15x
	USART1	78h to 7Fh, 01h,03h,05h	Above with
	USI (SPI & I2C)	78h to 7Dh	MSP430x2xx

T a b l e	UART	60h to 67h, 5Dh to 5Fh, 1h and 3h D0h to D7h, CDh to CFh, 6h,7h 68h to 6Fh, 118h to 11Bh D8h to DFh, 17Ch to 17Fh	MSP430x2xx
2			
Flash Memory	Flash Memory	128h to 12Dh	
1			
Basic CLOCK	Basic clock control	56h to 58h	
OPe-Amp		C0h to C5h	MSP430x2xx
Timer	Watch Dog	00h,02h, 120h	
i p h e r a l	Basic Timer	01h, 03h, 40h, 46h, 47h	
	Timer_A	160h to 17Fh and 12Eh	
	Timer_B	180h to 19Fh and 11Eh	
	RTC	41h to 45h, 4Ch to 4Fh, 1h,3h	MSP430X4XX
a n d m e m o r y	ADC10	1B0h to 1B5h,48h to 49h & 4Ah 1BCh and 1BDh.	MSP430X11X
	ADC12	1A0h to 1A8h 140h to 15Eh 80h to 8Fh	
	SDADC16	100h to 103h, 110h to 113h, B0h, B7h	MSP430X2XX
	SDADC24	100h to 11Fh, B0h BEh	MSP430X2XX
DAC		1C0h to 1C3 1C8h, 1CBh	
m a t r i x b l e		122h to 127h 1D0h to 1D7h 1DAh to 1DFh 1E2h, 1E3h, 1E6h to 1Ebh 1EEh, 1EFh, 1F0h to 1F7h	MSP430X2XX

2.1: Peripheral and memory map

In the above Table 2.1, it is shown that some of the registers are common for many peripherals.

All of the listed peripherals in the table are not present in all the devices of MSP430. As an example, MSP430C1101 device in MSP430X1XX family is having P1, P2, Comparator_A, and Timer_A. Similarly, each device in a family has its own list of peripherals. The other capabilities like interrupt, Watchdog timers, clock circuit, and reset circuits are common for all families of the MSP430 devices.

When you develop a system with the MSP430 controller using certain peripherals, ensure that the device you have chosen has the required peripherals in the device by referring the data sheet from TI web site.

Also, users need to ensure that the registers associated with each peripheral are handled for configuration properly.

CPU Registers in MSP430:

The CPU incorporates sixteen 16-bit registers from R0 to R15. The registers R0, R1, R2 and R3 have dedicated functions. R4 to R15 are working registers for general use.

As shown in the figure below, four of the registers, R0 to R3, are dedicated as program counter, stack pointer, status register, and constant generator respectively. The remaining registers are general-purpose registers.

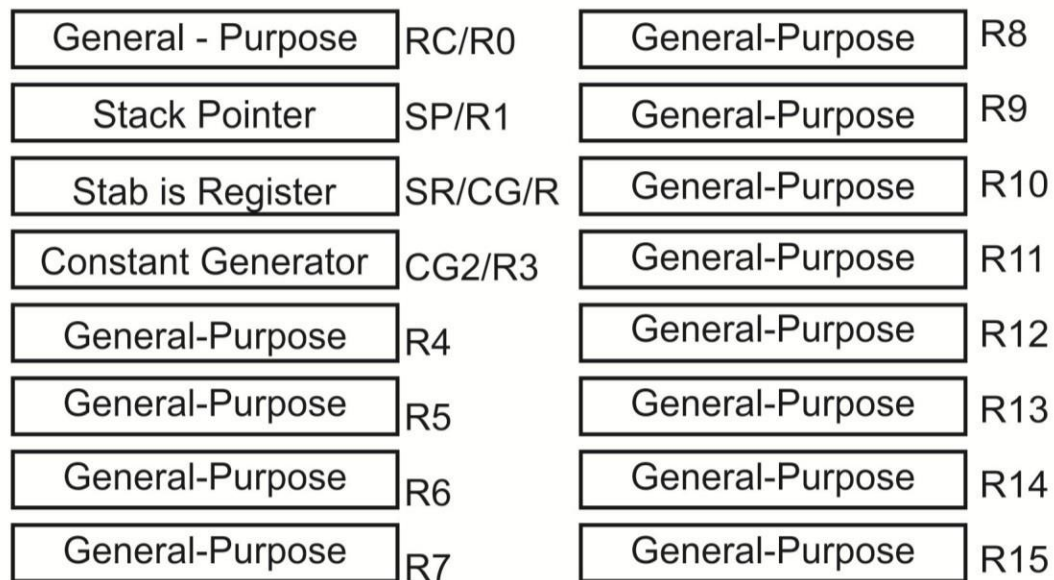


Figure 2.2.1 CPU registers in MSP430

The detailed description each CPU register in MSP430 is discussed below

Program Counter (PC)

The register R0 is a 16-bit program counter (PC), which points to the next instruction to be executed. The size of each instruction takes an even number of bytes (two, four, or six) and the PC is incremented according to the size of the previous instruction. The instruction in the memory space is accessed on a word by word basis and the PC is aligned to even addresses and LSB is hardwired to zero.

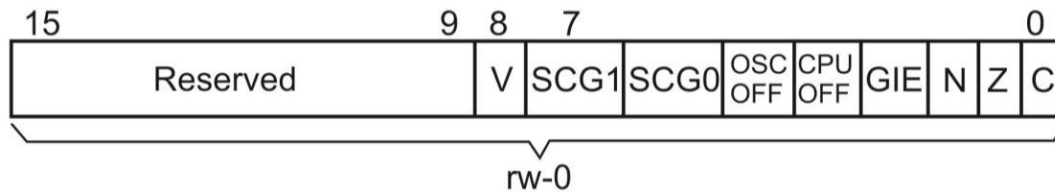
Stack Pointer (SP)

The register R1 is the stack pointer (SP) and is used by the CPU to store and read the return addresses of subroutine calls and interrupt handling. SP is adjusted by a pre-decrement, post-increment scheme when the stack is accessed by the CPU. In addition, the SP can be accessed manually by software with all instructions and addressing modes. The SP is initialized into RAM

locations of the memory space by the user, and is aligned to even addresses with LSB hardwired to zero.

Status Register (SR)

The register R2 is the status register (SR) and it is used as a source or a destination register. It can be used in the register mode only and addressed by word instructions. The remaining combinations of addressing modes are used to support the constant generator. Figure below shows the individual bit description of SR register.



Description of Status Register Bits

Bit Description

V:

(Overflow bit) This bit is set when the result of an arithmetic operation overflows the signed-variable range.

Set when: Positive + Positive = Negative or Negative + Negative = Positive, otherwise reset

Set when: Positive - Negative = Negative or Negative - Positive = Positive, otherwise reset

SCG1 (System clock generator 1):

This bit, when set, turns off the SMCLK.

SCG0 (System clock generator 0):

This bit, when set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.

OSCOFF (Oscillator Off):

This bit, when set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK

CPUOFF (CPU off):

This bit, when set, turns off the CPU.

GIE (General interrupt enable):

This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.

N (Negative bit):

This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative. Word operation: N is set to the value of bit 15 of the result Byte operation: N is set to the value of bit 7 of the result

Z (Zero bit):

This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.

C (Carry bit):

This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.

Constant Generator Registers CG1 and CG2:

Six commonly-used constants are generated with the constant generator registers R2 and R3, without requiring an additional 16-bit word of program code. The constants are selected with the source-register addressing modes (As), as described as below

Values of Constant Generators CG1, CG2

Register	As	Constant	Remarks
R2	00	-----	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

The advantage of having this structure is that there are no special instructions required, no additional code word for the six constants, no code memory access required to retrieve the constant.

The assembler uses the constant generator automatically if one of the six constants is used as an immediate source operand. Registers R2 and R3, used in the constant mode, cannot be used explicitly they act as constant registers.

Peripheral registers

In MSP430, the in-built peripherals such as GPIO, serial interface and timer etc. are mapped to the memory space of the CPU as memory mapped IO. The registers associated with the corresponding peripherals are discussed as follows.

Peripherals	Short Name	Register
GPIO registers	P1IN	Input
	P1OUT	Output
	P1DIR	Direction
	P1IFG	Interrupt Flag
	P1IES	Interrupt Edge Select
	P1IE	Interrupt Enable
	P1SEL	Port Select
Watchdog Timer Registers	WDTCTL	Watchdog timer control register
	IE1	SFR interrupt enable register 1
	IFG1	SFR interrupt flag register 1

Timer A Registers	TACTL TAR TACCTL0 TACCR0 TACCTL1 TACCR1 TACCTL2 TACCR2 TAIV	Timer_A control Timer_A counter Timer_A capture/compare control 0 Timer_A capture/compare 0 Timer_A capture/compare control 1 Timer_A capture/compare 1 Timer_A capture/compare control 2 Timer_A capture/compare 2 Timer_A interrupt vector
Timer B Registers	TBCTL TBR TBCCTL0 TBCCR0 TBCCTL1 TBCCR1 TBCCTL2 TBCCR2 TBCCTL3 TBCCR3 TBCCTL4 TBCCR4 TBCCTL5 TBCCR5 TBCCTL6 TBCCR6 TBIV	Timer_B control Timer_B counter Timer_B capture/compare control 0 Timer_B capture/compare 0 Timer_B capture/compare control 1 Timer_B capture/compare 1 Timer_B capture/compare control 2 Timer_B capture/compare 2 Timer_B capture/compare control 3 Timer_B capture/compare 3 Timer_B capture/compare control 4 Timer_B capture/compare 4 Timer_B capture/compare control 5 Timer_B capture/compare 5 Timer_B capture/compare control 6 Timer_B capture/compare 6 Timer_B Interrupt Vector
Peripherals	Short Name	Register
GPIO registers	P1IN P1OUT P1DIR P1IFG P1IES P1IE P1SEL	Input Output Direction Interrupt Flag Interrupt Edge Select Interrupt Enable Port Select
Watchdog Timer Registers	WDTCTL IE1	Watchdog timer control register SFR interrupt enable register 1

	IFG1	SFR interrupt flag register 1
Timer A Registers	TACTL	Timer_A control
	TAR	Timer_A counter
	TACCTL0	Timer_A capture/compare control 0
	TACCR0	Timer_A capture/compare 0
	TACCTL1	Timer_A capture/compare control 1
	TACCR1	Timer_A capture/compare 1
	TACCTL2	Timer_A capture/compare control 2
	TACCR2	Timer_A capture/compare 2
	TAIV	Timer_A interrupt vector
Timer B Registers	TBCTL	Timer_B control
	TBR	Timer_B counter
	TBCCTL0	Timer_B capture/compare control 0
	TBCCR0	Timer_B capture/compare 0
	TBCCTL1	Timer_B capture/compare control 1
	TBCCR1	Timer_B capture/compare 1
	TBCCTL2	Timer_B capture/compare control 2
	TBCCR2	Timer_B capture/compare 2
	TBCCTL3	Timer_B capture/compare control 3
	TBCCR3	Timer_B capture/compare 3
	TBCCTL4	Timer_B capture/compare control 4
	TBCCR4	Timer_B capture/compare 4
	TBCCTL5	Timer_B capture/compare control 5
	TBCCR5	Timer_B capture/compare 5
TBCCTL6	Timer_B capture/compare control 6	
TBCCR6	Timer_B capture/compare 6	
TBIV	Timer_B Interrupt Vector	

Table 2.2.1 Peripheral registers in MSP430

2.4. I/O pin multiplexing & its relevance

In a microcontroller, there should be higher performance and capability in terms of peripherals out of the hardware resources available with limited numbers of I/O pin. It is the responsibility of the chip design engineer to create a limited budget device by reducing I/O pins of a microcontroller, but having more peripherals integration. This is done through multiplexing the functions of the various peripherals with a single pin in the chip. Pins are expensive and most microcontroller applications won't use all of the features available in a chip for a single application. So, the cost of the chip is reduced by making many different functions available on the same pin. It is exemplified by pin-out detail of MSP430G2553 controller by figure shown below

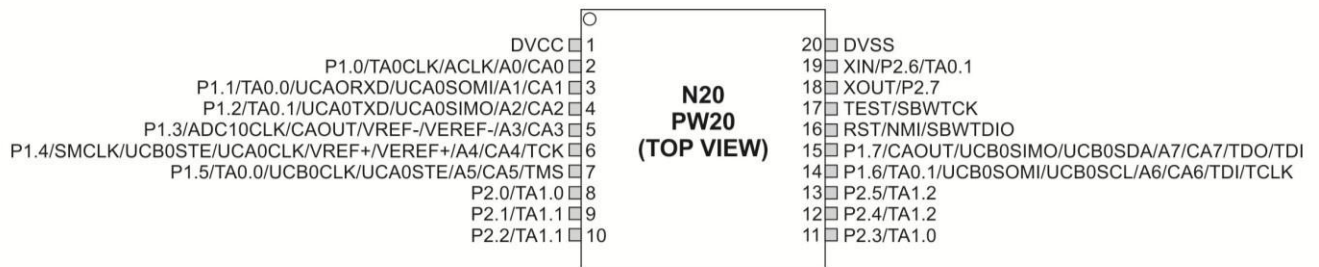


Figure 2.3.1A MSP430G2553 Pinout

It is noticed that each pin in the above chip in figure 2.3.1A has a long name, with several functionality separated by a slash. Because of limited number of pins in microcontroller and at the same time a large number of peripherals, the chip design is done to multiplex the pins among the internal modules. This means that each pin has a more number of functions that it can perform, but only one of them are active or functioning associated with that pin at a time. Most pins of the MSP430 operate as GPIO pins, with a possibility of functioning as a specialized pin in the right configuration. As GPIO pins, each pin is independently controlled and can be made an input and an output, high or low.

Embedded design engineer must carefully consider the use of each pin in the controller along with the peripherals associated with the pin. Once one of the peripheral functions is chosen for the pin, then other functions associated with that pin are compromised. The following Table 2.3.1A shows a few of the pin-out details of the MSP430F5638. In this, each pin of the has GPIO capability as well as multiplexed with other peripheral's pin.

NAME	PIN number	Description
P6.4/CB4/A4	1	General-purpose digital I/O Comparator_B input CB4 Analog input A4
P7.6/CB10/A14/DAC0	7	General-purpose digital I/O Comparator_B input CB10 Analog input A14 – ADC DAC12.0
P1.0/TA0CLK/ACLK	34	General-purpose digital I/O with port interrupt Timer TA0 clock signal TACLK input ACLK output

Table 2.3.1A -- List of few pin detail of MSP430F5358

As an example, the Pin 1 of MSP430F5638 is functioning as P6.4 of GPIO, comparator input 4 or analog input. Similarly some of the pins are multiplexed with other special functions as shown in the above table. Each device of MSP430 family has multiplexed with variants of peripherals functions.

2.5. Pull up/down registers

In general, the input of the microcontroller should not be left as float, which means input should be given either low or high. In this case, Pull-up and Pull-down resistors are often used when interfacing a switch or some other input device with the microcontroller.

The Figures 2.4.1 and 2.4.2 show the pull up and pull down resistor connection detail. In Figure 2.4.1, the switch S1 is connected to the IO pin configured as input to the controller and another end connected to logic 0. When the switch is pressed, the pin reads zero, when the switch is not pressed the pin reads logic 1 through a pull up resistor. If there is no pull up resistor in the circuit, when the switch is not pressed, the pin is in float case and it reads garbage input so the switch status cannot be distinguished. This is resolved by a pull up resistor. A pull-up resistor is a resistor which is used to ensure that a wire is pulled to a high level in the absence of an input signal.

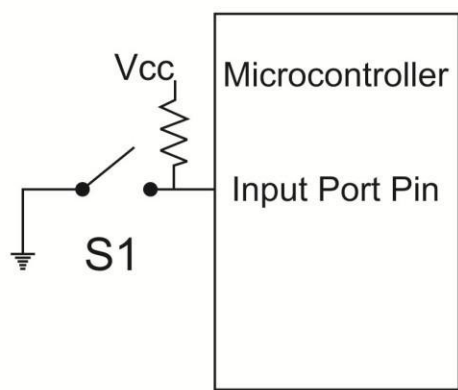


Figure 2.4.1 Pull up circuit

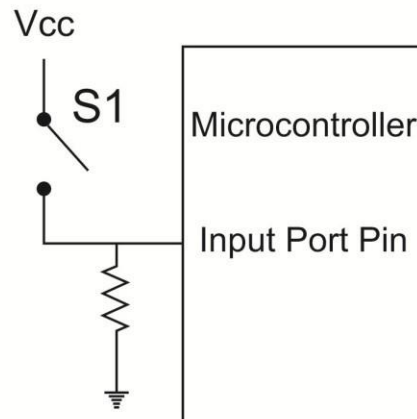


Figure 2.4.2 Pull down circuit

Similarly, as shown in figure 2.4.2, it is clear that the pull down resistor is used in the absence of an active low input signal when switch is not pressed. Pull-down resistors work in the same manner as pull-up resistors, except that they pull the pin to a logical low value. They are connected between ground and the appropriate pin on a device. An example of a pull-down resistor in a digital circuit can be seen in the figure. A pushbutton switch is connected between the supply voltage and a microcontroller pin. In such a circuit, when the switch is closed, the micro-controller input is at a logical high value, but when the switch is open, the pull-down resistor pulls the input voltage down to ground (logical zero value), preventing float (an undefined state) at the input. The pull-down resistor must have a larger resistance than the impedance of the logic circuit, or else it might be able to pull the voltage down by too much and the input voltage at the pin would remain at a constant logical low value regardless of the switch position.

Most microcontrollers like MSP430 have in-built programmable pull up or pull down resistors making it possible to interface a switch without pull up or pull down switches. Pull-up resistors are generally preferred to use for connecting switches more often than pull-down resistors, though the microcontroller is facilitated with internal pull-up and pull-down resistors. MSP430 supports enabling an internal Pull-up or Pull-down resistors via software when the GPIO is in input mode.

In MSP430 microcontroller, in order to enable the internal pull up or pull down resistor, the registers Port Direction Registers (PxDIR), Port Pull up or Pull down Resistor Enable Registers (PxREN) and Port Output Registers (PxOUT) are used. By referring to the Table 2.4.1, PxDIR register is used to

configure the port pin as an input. PxREN used to enable the Pull up/Pull down, but choosing Pull up or Pull down resistor to the port pin is done by the PxOUT register.

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

Table 2.4.1 I/O configuration

By referring to the Figure 2.4.3, it is clear that when the port pin is configured as an input by PxDIR and internal Pull up/down network for each pin is enabled by PxREN. PxOUT register controls whether the resistor is pull-up or pull-down.

```
P2DIR = 0x0F; //Higher 4 bits of P2 are configured as input
P2REN = 0xF0; // Higher 4 bit of P2 are enabled for pull up/down
P2OUT = 0xC0; // P2.7 and P2.6 are configured to pull up
           // P2.5 to P2.4 are configured to pull down
```

Figure 2.4.3 Pull up and Pull down configuration

2.6. GPIO control

General Purpose Input Output (GPIO) is the default necessary peripheral in any microcontroller as parallel ports. Input devices like switches and sensors, output devices like LEDs, Relays, buzzers, and display are connected to microcontroller through GPIO pins.

A GPIO Pin can be configured either as an input or an output pin. When a pin is in output mode configuration, this allows you to drive each bit / pin individually to logic high or low. This output can be seen by naked eye by connecting an LED and can be heard by connecting a buzzer. When a GPIO pin is configured as an input mode, it can read logic high or low from an external device. It's perfectly acceptable to have portion of pins be output and others input on the same port.

Port pins in MSP430 controller are capable of GPIO functionality and are referred in programming with the naming PX.Y, where X represents the port number to which the pin belongs and Y represents Pins belonging to port X. As an example, P2.1 is refers to pin 1 of port 2. Notice that some pins are GPIO only and some pins are multiplexed with special functions. The availability of the ports and special functions can be referred to in the corresponding data sheet.

Each device and series of MSP430 has a limited number of ports. As an example, MSP430x5xxx series has 9 GPIO ports and they are referred as P1 to P9 with 8 bit in size for each port. P1 and P2 are paired to form 16 bit port and it is referred as PA. The following Table 2.5.1 shows the list of ports and its pairs with how they are referred.

8 bit Port pairs	16 bit Port
P1-P2 P3-P4 P5-P6 P7-P8 P9	PA (PA_Low byte = P1 and PA_High byte = P2) PB (PB_Low byte = P3 and PB_High byte = P4) PC (PC_Low byte = P5 and PC_High byte = P6) PA (PD_Low byte = P7 and PD_High byte = P8) No pair

Table 2.5.1 Ports and its pairing

Additionally, ports P1 and P2 are facilitated with interrupt feature. Each interrupt associated with each pin of P1 and P2 can be individually enabled and configured to set to rising or falling edge triggered interrupt. All interrupt pins of P1 have a single interrupt vector (P1IV) and all interrupt pins of P2 have a single interrupt vector (P2IV). On some devices of MSP430, additional ports with interrupt capability are available.

2.6.1. Port registers

Each port in MSP430 is associated with some registers that control the function of the pins and provide information on their current status. The description of port registers are discussed as follows.

2.6.1.1. PxSEL (Port x function select)

As already discussed, the GPIOs in MSP430 are multiplexed pins so each Port pin of MSP430 is acting as a GPIO and a special function (Also used as the pin for other peripherals such as timer, serial, ADC, DAC and etc). The PxSEL register is used to configure the pin either as a GPIO pin or as a special function pin. Writing 0 into a bit of this register configures the pin corresponding to the bit as a GPIO and writing 1 configures the corresponding pin for an alternate specialized function of the other peripheral. The following table 2.5.2 describes the PxSEL configuration of the Port 2.

PxSEL register	Description
P2SEL = 0x80; (1000 0000 b)	P2.0 to P2.6 configured as GPIO. P2.7 is configured for special function

Table 2.5.2 PxSEL bit description

2.6.1.2. PxDIR (Port x Direction)

Once a port pin is configured as GPIO pin, it is then required to configure the direction of pin either for input or output. The PxDIR register is used to configure the particular port pin direction regardless of selected function in PxSEL. Writing 1 into a bit of the register configures the corresponding pin for an output and writing 0 configures the pin for an input. The following table 2.5.2 describes the PxDIR configuration of the Port 2.

PxDIR register	Description
P2DIR = 0x01; (0000 0001 b)	P2.0 is configured as output pin P2.1 to P2.7 configured as input pin

Table 2.5.3 PxDIR bit description

2.6.1.3. PxIN (Port x input)

Once a port pin is configured as an input, the pin is read for current input status by the CPU by using this register. Ensure that the input is configured for internal pull up and pull down configuration by PxREN register.

2.6.1.4. PxOUT (Port x Output)

Once a port pin is configured as an output, once the data written into the PxOUT register the corresponding data is coming out as a logic signal level to the corresponding port pin.

2.6.1.5. PxREN (Port x Resistor Enable)

Internal pull up/ pull down resistor circuit is enabled by this register. Each bit in each PxREN register is configured according to the corresponding I/O pin. The pull up or pull down resistor is chosen by the data sent to the PxOUT register. By referring to the Figure 2.5.1, some of the P2 pins are configured as inputs and some of them are enabled with Pull up/ down mode. Among these, a few of them chosen for pull up and others for pull down input.

```
#include < msp430x5xxx.h>
main()
{
P2DIR = 0xFF; // The entire P2 configured as input
P2REN = 0x80; // P2.7 is configured to pull up/down
           // P2.0 to P2.6 is float condition
P2OUT = 0x80; // P2.7 is configured to pull up
           // P2.4 to P2.6 are configured to pull down
           // P2.0 to P2.3 – No change
}
```

Figure 2.5.1 Pull up / Pull down configuration

2.6.1.6. PxIE (Port x interrupt enable)

Interrupt associated with each P1 and P2 pins is enabled by the PxIE register. The interrupt is occurred if only if the port x interrupt flag (PxIFG) and PxIE are set for the corresponding interrupt pin. The bits in PxIFG manually checked to identify the source of the interrupt pin in the particular port.

2.6.1.7. PxIFG (Port x interrupt flag)

As we discussed above, each bit in PxIFGx register is associated with an individual port pin and it is set when the selected input signal edge occurs at the pin. Once any one of an interrupt flag in PxIFGx is set it requests an interrupt when its corresponding bit and GIE (Global Interrupt Enable) bit in PxIE register are set. After servicing to the interrupt, the flag can be cleared only by software. If the bit is 0 there is no pending in interrupt, else if the bit is 1 there is a pending in the interrupt and CPU should respond to it.

2.6.1.8. PxIES (Port x interrupt edge select)

This register selects the interrupt triggering type either as rising edge or falling edge for the corresponding port pin, which will cause the interrupt and the bit in PxIFG register to be set.

Bit = 0: The rising edge triggered

Bit = 1: The falling edge triggered

2.6.1.9. PxIV (Port x interrupt vector)

It is a 16 bit register that stores the vector address of the port, which is configured as interrupt source. This 16-bit register is a priority encoder which can be used to handle pin-change interrupts. If n is the lowest-numbered interrupt bit which is pending in PxIFG and enabled in PxIE, this register reads as $2n+2$. If there is no such bit, it reads as 0. The scale factor of 2 allows direct use as an offset into a branch table. Reading this register also clears the reported PxIFG flag.

2.6.1.10. PxDS (Port x drive strength ('5xx only))

Bits set in this register enable high-current outputs. This increases output power, but may cause EMI or Electro Magnetic Interference. Ports 0–2 can produce interrupts when inputs change. Additional registers configure this ability:

Some pins have special purposes either as inputs or outputs. (For example, timer pins can be configured as capture inputs or PWM outputs.) In this case, the PxDIR bit controls which of the two functions the pin performs when the PxSEL bit is set. If there is only one special function, then PxDIR is generally ignored. The PxIN register is still readable if the PxSEL bit is set, but interrupt generation is disabled. If PxSEL is clear, the special function's input is frozen and disconnected from the external pin. Also, configuring a pin for general-purpose output does not disable interrupt generation.

2.6.1.11. Accessing GPIO-1 (Writing into the Port)

Writing data into a port pin is demonstrated by connecting an LED in P1.0 through a series resistor as shown in the Figure 2.5.2. The Port pin can makes logic 5v when logic 1 is output, so it is essential to limit the current to drive the LED. The objective of the example is to cause the LED to blink by software controlled delay. Use a launch pad as a target board which contains MSP430G2xxx, connect the LED in series with 1K ohms resistor as shown in figure. Write the C program as shown below in Code Composer studio and download the code into the target. Realize the code by changing the delay value.

2.6.2. Accessing GPIO-1 (Writing into the Port)

Writing data into a port pin is demonstrated by connecting an LED in P1.0 through a series resistor as shown in the Figure 2.5.2. The Port pin can makes logic 5v when logic 1 is output, so it is essential to limit the current to drive the LED. The objective of the example is to cause the LED to blink by software controlled delay. Use a launch pad as a target board which contains MSP430G2xxx, connect the LED in series with 1K ohms resistor as shown in figure. Write the C program as shown below in Code Composer studio and download the code into the target. Realize the code by changing the delay value.

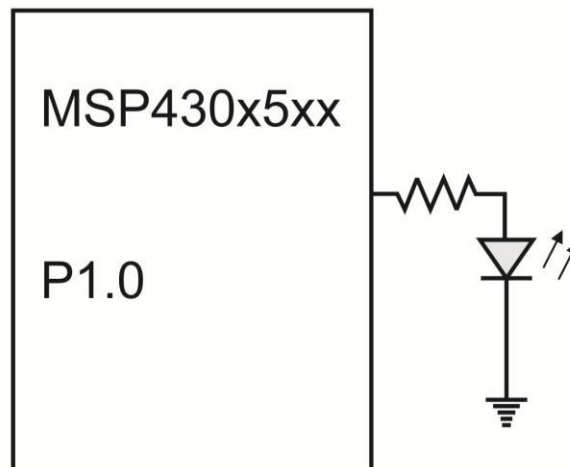


Figure 2.5.2 – Circuit for LED blinking

```

#include"msp430x5xxx.h"
void main (void)
{
  unsigned int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watch dog timer
  P1DIR = 0x01; // Set P1.0 to output direction
  While(1)
  {
    P1OUT = 0x01; // Set P1.0 Pin to Logic High
    for ( i=0; i<20000; i++); //delay
    P1OUT = 0x00; // Set P1.0 Pin to Logic Low
    for ( i=0; i<20000; i++); //delay
  }
}

```

Figure 2.5.3 – Program1 for LED blinking

In the above program in Figure 2.5.3, after the main there are 3 lines, and in this the second line stops the watch dog timer. It is mandatory and this is discussed in detail in the following chapters. P1DIR stored with 0x01 for the configuration of P1.0 as output and others are as an input. P1OUT is stored with 0x01 to set P1.0 pin to high. The objective is to toggle P1.0 but the program disturbs other pins in the Port1. So the program needs to be modified and optimized as in the following program in Figure 2.5.4.

```

#include"msp430x5xxx.h"

void main (void)
{
  unsigned int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watch dog timer
  P1DIR |= 0x01; // Set P1.0 to output direction
  While(1)
  {
    P1OUT ^= 0x01; // Toggle P1.0 Pin by EXOR operation
    for ( i=0; i>20000; i--); //delay
  }
}

```

Figure 2.5.4 – Program2 for LED blinking

2.6.3. Accessing GPIO-2 (Reading from the Port)

There are two LEDs connected in P1.6 and P1.0 and these LEDs are controlled by the switch SW1 connected in P1.3 as shown in the Figure 2.5.5. By referring to the program in figure 2.5.6, P1.3 is configured to input with internal pull up enabled. P1.0 and P1.6 are configured as output. When the switch is not pressed the P1.3 is read as high because of internal pull up

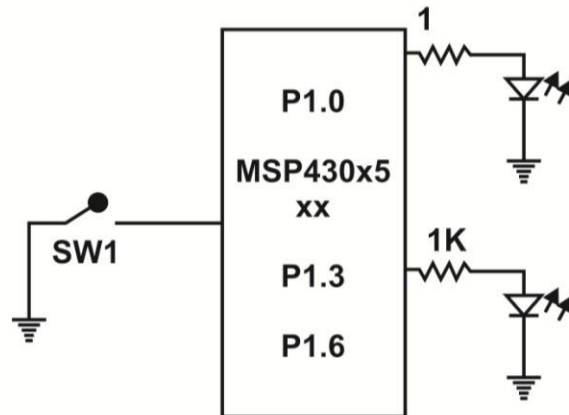


Figure 2.5.5 – Circuit for controlling LEDs by the switch

```
#include "msp430x5xxx.h"
void main (void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watch dog timer
    P1DIR |= 0x41; // Set P1.0 and P1.6 are to output
    P1REN = 0x04; // P1.3 is enabled for pull up/down circuit
    P1OUT |= 0x04; // P1.3 is chosen for pull up resistor
    P1OUT |= 0x40; // P1.0 = 0 and P1.6 = 1
    while(1)
    {
        if (!(P1IN & 0x04)) // It is true when P1.3 is zero and other bits are
            masked
            P1OUT ^= 0x41; // Toggle P1.0 and P1. 6 by EXOR operation
                            // Other bits are not disturbed
    }
}
```

Figure 2.5.6 – Program for controlling LEDs by polling

2.6.4. Accessing GPIO-3 (port input by interrupt)

Using port pin as an interrupt input is explained by referring the same above circuit in Figure 2.5.4. The port P1.3 is connected to the switch SW1 and it is configured as falling edge triggered interrupt. Upon falling edge interrupt by p1.3 the LEDs in P1.0 and P1.6 are toggled.

```
#pragma vector=PORT1_VECTOR

__interrupt void Port_1 (void)
{
    P1OUT ^= 0x41;          // Toggle P1.0 and P1. 6 by EXOR operation
                          // Other bits are not disturbed
    P1IFG &= ~0x04;        // clean P1.3 Interrupt Flag
}

void main(void)
{
    WDTCTL = WDTPW | WDTHOLD; //Stop Watchdog Timer
    P1DIR |= 0x41;           // Set P1.0 and P1.6 are to output
    P1REN = 0x04;           // P1.3 is enabled for pull up/down circuit
    P1OUT |= 0x04;          // P1.3 is chosen for pull up resistor
    P1OUT |= 0x40;          // P1.0 = 0 and P1.6 =1
    P1IE |= 0x04;           //Interrupt Enable in P1.3
    P1IES |= 0x04;          //P1.3 Interrupt flag high-to-low transition

    _BIS_SR (LPM3_bits + GIE); //Low Power Mode with interrupts enabled
}

```

Figure 2.5.7 – Program for controlling LEDs by interrupt

2.7. Interrupts and interrupt programming.

2.7.1. Introduction:

An interrupt is an event generated by hardware or software. When an event that is asynchronous to the current program execution occurs, then it automatically transfers CPU execution to a defined sub program called interrupt sub program.

The following Figure 2.6.1 shows an example of how a microcontroller responds to an input signal. The system is made for controlling the speed of the motor with proper feedback mechanism. The program is written for sensing the speed and controlling the motor for the required speed and this program is running forever. When the operator wants to stop the motor immediately and he presses the Panic OFF switch then the motor should be OFF immediately.

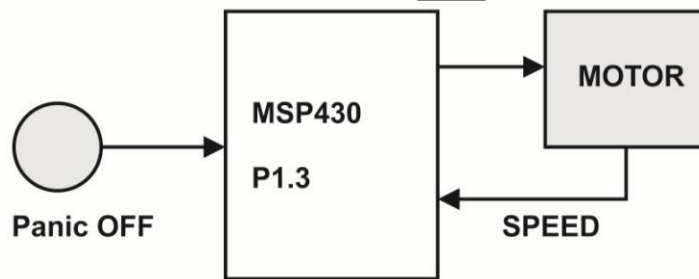


Figure 2.6.1 Microcontroller with interrupt

2.7.2. Polling method:

The flowchart given in Figure 2.6.2 explains the above example in polling method as given below. The input (Panic OFF) is read first, if it is not pressed then it senses the speed and runs a speed control algorithm to tune it to the required speed and this loop is continued till the panic switch is switched OFF.

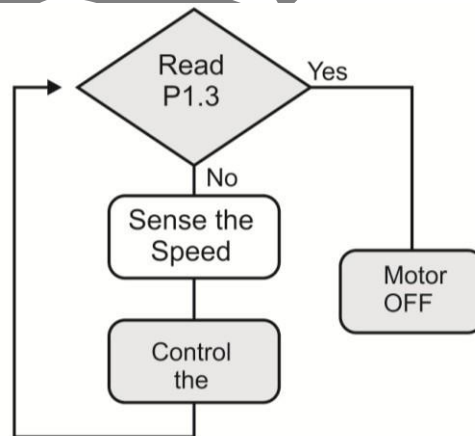


Figure 2.6.2. Flow chart for polling the input

The method of periodically checking the input pin status for every looping is called polling. The critical problem of this method is real time panic control. If the panic switch is pressed when the CPU is performing for sensing the speed of the motor then the switch is ignored or the switch should be still pressed as long as the CPU comes to read the pin. In polling, the microcontroller keeps checking the status of external pin and then it responds to it. While doing so, it does no other operation and

consumes all its processing time for only monitoring the pin. This problem is overcome by using interrupts.

2.7.3. Interrupt driven method:

In the interrupt driven method, the controller responds to only when an interruption occurs and the CPU is not required to regularly monitor the status of the input pin. The flowchart for the same example in interrupt driven method is given below. The input P1.3 is configured as an interrupt input.

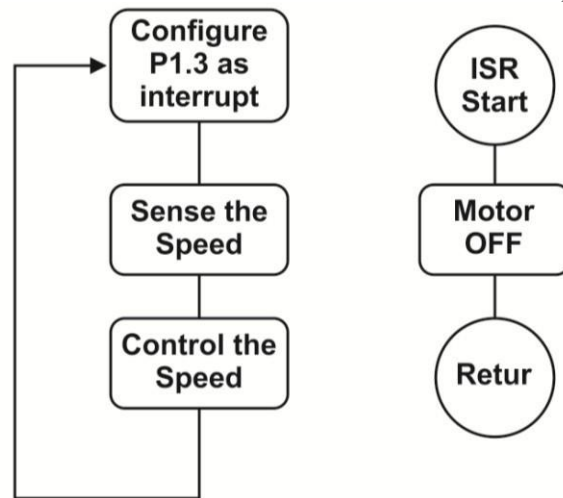


Figure 2.6.3.a

Figure 2.6.3.b

Figure 2.6.3.a, Main program of the interrupt routing, b. Interrupt service routine

2.7.4. Sources of interrupt:

The interrupts can be triggered either by hardware or software interrupt sources. Hardware sources can be external or internal to the microcontroller chip set. External sources are multiplexed with GPIO port pins such as NMI, INT0, and INT1. It is different for each microcontroller.

The internal interrupt sources are from the internal peripherals. It is different from device to device in the microcontroller families. They are, UART Serial transmit /receive, Portpin, ADC End of Conversion, Pulse-Accumulator overflow, Real Time Clock time-outs, Watchdog Timer Reset, Timer Overflow on time-out, Timer comparison with Output compare, Timer capture on inputs and etc. Software sources for interrupt are related to processor execution by detecting computational error.

2.7.5. Interrupt triggering mechanism:

It is important to consider that each individual interrupt request has a separate execution but local variables and registers used in the interrupt service routine are unique and separate from one interrupt event to the next interrupt. Each interrupt input has its corresponding interrupt flag. As an example, whenever watch dog timer generates an interrupt it sets watch dog timer interrupt flag (WDTIFG) and execution goes to its service routing. In the service routine, WDTIFG is cleared. There are the standard terms used in interrupt such as masking, pending, enabling, disabling, triggering type, interrupt service routine (ISR), interrupt flag, priority, vectored interrupt and non vectored interrupt.

2.8. MSP430 Interrupts

There are three types of interrupts in MSP430 architecture they are

- System reset
- Non-maskable interrupt (NMI)
- Maskable Interrupts

2.8.1. System Reset

The system reset circuitry such as Power-On Reset (POR), Power-Up Clear (PUC) signal and Brown-Out Reset (BOR) initiates interrupt upon its detection.

POR is can be generated by the following three events:

- Powering up the device
- A low signal on the RST/NMI pin when configured in the reset mode
- supply voltage supervisor (SVS) low condition when PORON = 1

SVS is used to monitor the AVCC supply voltage or an external voltage. The SVS can be configured to set a flag or generate a POR reset when the supply voltage or external voltage drops below a user-selected threshold.

PUC is can be generated by the following four events:

- Watchdog timer expiration when in watchdog mode only
- Watchdog timer security key violation
- A Flash memory security key violation
- POR signal also generate the PUC

Some devices have a brownout reset circuit that replaces the POR detect and POR delay circuits. The brownout reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the VCC terminal. The brownout reset circuit resets the device by triggering a POR signal when power is applied or removed

2.8.2. Non-maskable interrupt (NMI):

Non-Maskable (NMI) interrupts are not masked by the General Interrupt Enable bit (GIE). There are two levels of NMIs such as system NMI (SNMI) and user NMI (UNMI) and both NMIs are having individual vector address with individual interrupt enable bits. The number and types of NMI sources may vary from device to device. UNMI interrupt can be generated by three sources:

- An edge on the RST/NMI pin when configured in NMI mode
- An oscillator fault occurs
- An access violation to the flash memory

The SNMI interrupt can be generated by following sources:

- Power Management Module (PMM) , SVML/SVMH supply voltage fault
- PMM high/low side delay expiration
- Vacant memory access

At power-up, the RST/NMI pin is configured in the reset mode. The CPU is held in the reset state as long as the RST/NMI pin is held low. After changing to a high state, the CPU starts program execution at the word address stored in the reset vector, 0FFFFh. The flash access violation can be enabled to generate an NMI interrupt by setting the ACCVIE bit. The ACCVIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by a flash access violation. The oscillator fault signal warns of a possible error condition with the crystal oscillator. The oscillator fault can be enabled to generate an NMI interrupt by setting the OFIE bit. The OFIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by an oscillator fault.

2.8.3. NMI Interrupt Handler:

The NMI interrupt is a multiple-source interrupt. An NMI interrupt automatically resets the NMIIE, OFIE and ACCVIE interrupt-enable bits. The user NMI service routine resets the interrupt flags and re-enables the interrupt-enable bits according to the application needs as shown in Figure 2.6.4

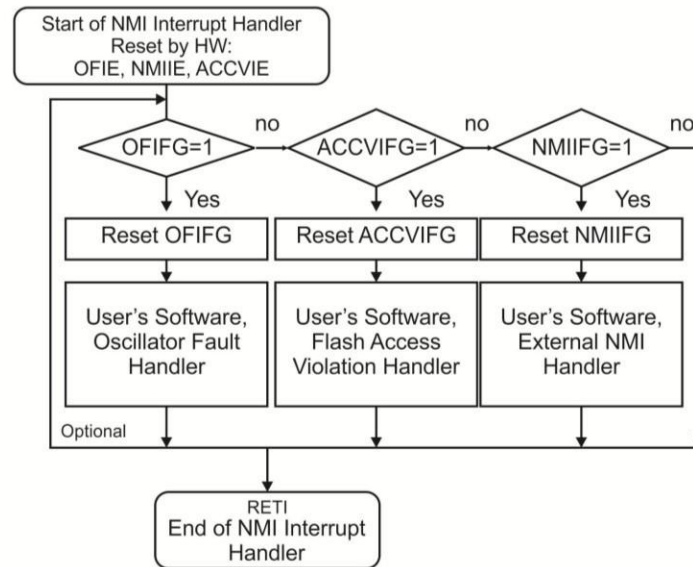


Figure 2.6.4 Flow chart for NMI Handler

2.8.4. Maskable Interrupts:

Maskable interrupts are caused by internal peripherals of MSP430 with interrupt capability including the watchdog timer. Each maskable interrupt source can be enabled or disabled individually by an interrupt enable bit and also by global interrupt enable (GIE) bit in the status register (SR).

The interrupt latency is 6 cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt-service routine.

The interrupt logic executes the following:

- 1) Current instruction execution is completed.
- 2) The Program Counter is pushed onto the stack and the SR is also pushed onto the stack.
- 4) The interrupt with the highest priority is serviced if multiple interrupts occurred.
- 5) The interrupt request flag resets automatically. Other flags remain unchanged.
- 6) The SR is cleared. This terminates any low-power mode. Because the GIE is cleared, further interrupts are disabled.
- 7) The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.
- 8) After servicing to the interrupt, the SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
- 9) The PC pops from the stack and begins execution at the point where it was interrupted

2.9. Interrupt priority and Interrupt vector table

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain in MSP430 series as shown in the Figure 2.6.5 and Table 2.6.1. When the highest priority is in service the other interrupt requests are in the Flag as interrupt pending. Once the service of the interrupt is finished then the interrupt flag associated with the interrupt source should be cleared by program.

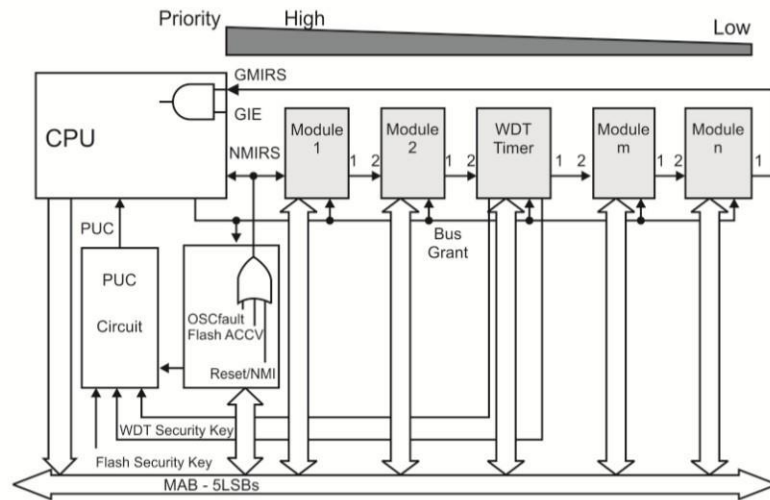


Figure 2.6.5 Interrupt prioritized modules

The interrupt vectors and the power-up starting address as well as the interrupt flags associated with the interrupts are described in Table 2.6.1. When an interrupt occurs, the flag is set and the CPU releases the current process and calls the interrupt service routing. When CPU is in any one of low power mode used, the interrupt wakes the CPU to service for interrupt.

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
System Reset Power-Up, External Reset Watchdog Timeout, Key Violation Flash Memory Key Violation	WDTIFG, KEYV (SYSRSTIV) ^{(1) (2)}	Reset	0FFFEh	63, highest
System NMI PMM Vacant Memory Access JTAG Mailbox	SVMLIFG, SVMHIFG, DLYLIFG, DLYHIFG, SVMLVLRIFG, SVMHVLRFIFG, VMAIFG, JMBNIFG, JMBOUTIFG (SYSSNIV) ⁽¹⁾	(Non)maskable	0FFFC h	62
User NMI NMI Oscillator Fault Flash Memory Access Violation	NMIIFG, OFIFG, ACCVIFG, BUSIFG (SYSUNIV) ^{(1) (2)}	(Non)maskable	0FFFAh	61
Comp_B	Comparator B interrupt flags (CBIV) ^{(1) (2)}	Maskable	0FFF8h	60
Timer TB0	TB0CCR0 CCIFG0 ⁽³⁾	Maskable	0FFF6h	59
Timer TB0	TB0CCR1 CCIFG1 to TB0CCR6 CCIFG6, TB0IFG (TB0IV) ^{(1) (3)}	Maskable	0FFF4h	58
Watchdog Interval Timer Mode	WDTIFG	Maskable	0FFF2h	57
USCI_AD Receive or Transmit	UCA0RXIFG, UCA0TXIFG (UCA0IV) ^{(1) (3)}	Maskable	0FFF0h	56
USCI_B0 Receive or Transmit	UCB0RXIFG, UCB0TXIFG (UCB0IV) ^{(1) (3)}	Maskable	0FFEeh	55
ADC12_A	ADC12IFG0 to ADC12IFG15 (ADC12IV) ^{(1) (3)}	Maskable	0FFECh	54
Timer TA0	TA0CCR0 CCIFG0 ⁽³⁾	Maskable	0FFEAh	53
Timer TA0	TA0CCR1 CCIFG1 to TA0CCR4 CCIFG4, TA0IFG (TA0IV) ^{(1) (3)}	Maskable	0FFE8h	52
USB_UBM ⁽⁴⁾	USB Interrupts (USBIV) ^{(1) (3)}	Maskable	0FFE6h	51
LDO-PWR ⁽⁵⁾	LDOOFFIFG, LDOONIFG, LDOOVIFG			

DMA	DMA0IFG, DMA1IFG, DMA2IFG, DMA3IFG, DMA4IFG, DMA5IFG (DMAIV) ^{(1) (3)}	Maskable	0FFE4h	50
Timer TA1	TA1CCR0 CCIFG0 ⁽³⁾	Maskable	0FFE2h	49
Timer TA1	TA1CCR1 CCIFG1 to TA1CCR2 CCIFG2, TA1IFG (TA1IV) ^{(1) (3)}	Maskable	0FFE0h	48
I/O Port P1	P1IFG.0 to P1IFG.7 (P1IV) ^{(1) (3)}	Maskable	0FFDEh	47
USCI_A1 Receive or Transmit	UCA1RXIFG, UCA1TXIFG (UCA1IV) ^{(1) (3)}	Maskable	0FFDC h	46
USCI_B1 Receive or Transmit	UCB1RXIFG, UCB1TXIFG (UCB1IV) ^{(1) (3)}	Maskable	0FFDAh	45
I/O Port P2	P2IFG.0 to P2IFG.7 (P2IV) ^{(1) (3)}	Maskable	0FFD8h	44
LCD_B ⁽⁸⁾	LCD_B Interrupt Flags (LCDBIV) ⁽¹⁾	Maskable	0FFD6h	43
RTC_B	RTC RDYIFG, RTC TEVIFG, RTC AIFG, RTC PSIFG, RTC PSIFG, RTC OFIFG (RTCIV) ^{(1) (3)}	Maskable	0FFD4h	42
DAC12_A	DAC12_0IFG, DAC12_1IFG ^{(1) (3)}	Maskable	0FFD2h	41
Timer TA2	TA2CCR0 CCIFG0 ⁽³⁾	Maskable	0FFD0h	40
Timer TA2	TA2CCR1 CCIFG1 to TA2CCR2 CCIFG2, TA2IFG (TA2IV) ^{(1) (3)}	Maskable	0FFCEh	39
I/O Port P3	P3IFG.0 to P3IFG.7 (P3IV) ^{(1) (3)}	Maskable	0FFCCh	38

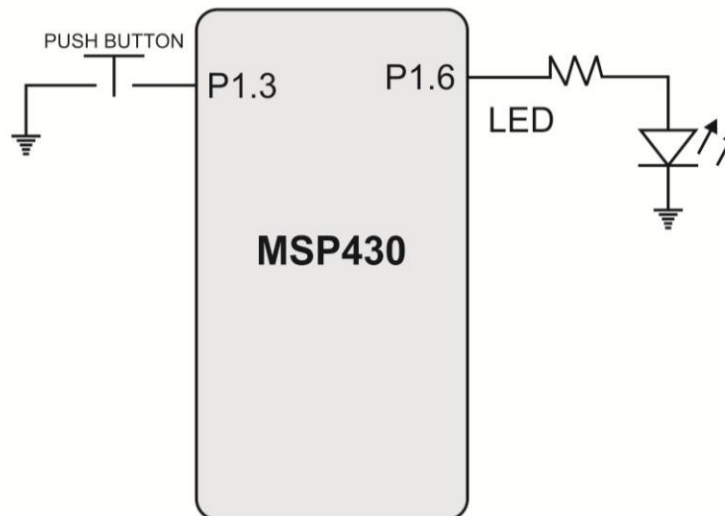
Table 2.6.1 Interrupt priority and vector address

The other registers associated with interrupt such as peripheral enable bit, interrupt enable bits, and interrupt flags are located in the SFRs.

2.10. Interrupt programming

2.10.1 Push Button Interrupt

By referring to the Figure 2.6.6 shown below, the push button is connected to GPIO P1.3 configured as interrupt pin. The interrupt by push button toggles the LED connected to another GPIO P1.6 configured as an output. Here it is assumed that there is no bounce in the push button. To start with programming, each device must be initialized to generate an interrupt and the CPU must be enabled to respond to a maskable interrupt.


Figure 2.6.6 Push button with interrupt

The P1.3 (connected with Push button) is configured as an input port pin with internal pull up by the following three statements.

```
P1SEL &= ~0x08; // Select Port 1 P1.3 as GPIO
P1DIR &= ~0x08; // P1.3 as input
P1REN |= 0x08; // Enable Port P1.3 with internal pull-up resistor
```

A mask setting determines which devices (e.g. push button) can generate an interrupt. In P1 Interrupt enable register, bit 3 (0x08) of P1.3 is connected with the push button is configured by 1 to enable (0 to disables or mask) device to trigger an interrupt and globally enable the interrupt. This is done with the following statements.

```
P1IE |= 0x08; // Enables P1.3 to generate interrupt
_BIS_SR(GIE); // Enable global interrupts enable bit
```

The interrupt handler mechanism in the C level program for MSP430 is done as follows. The push button is part of port 1, a PORT1_VECTOR interrupt handler is executed when a port 1 interrupt handler occurs.

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
```

When the interrupt in P1.3 occurs, the corresponding flag P1IFG bit is set to 1 by hardware. The interrupt handler should normally disable the interrupt by setting the same flag to 0, allowing another interrupt to occur. This is done by the following statement in the interrupt service routing.

```
P1IFG &= ~0x08; // Clears the P1.3 interrupt
```

```

#include <msp430.h>
#define LED 0x40 // BIT6
long i=0;
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    P1DIR |= LED; // Set P1.6 to output
    P1OUT &= ~LED; // Set LED to off
    P1SEL &= ~0x08; // Select Port 1 P1.3
    P1DIR &= ~0x08; // Port 1 P1.3 as input
    P1REN |= 0x08;
    P1IE |= 0x08;
    P1IFG &= ~0x08; // Clear interrupt flag
    _BIS_SR(GIE); // Enable interrupts
    _BIS_SR(LPM4_bits + GIE); // Enter LPM4
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    P1OUT ^= LED; // Toggle LED state
    P1IFG &= ~0x08; // P1.3 Interrupt Flag cleared
}

```

Figure 2.6.7 Push Button Interrupt program

Exercise:

Develop the c program for the following.

- 1) Three push buttons are connected in P1.0, P1.1 and P1.2 respectively and Three LEDs are connected in P2.0, P2.1 and P2.3 respectively. Each push button input is configured as interrupt input and assumed to be debounced. When any one of the push button is pressed, its corresponding LED is toggled.
- 2) Measure the bouncing frequency of switch button. A push button is connected in P1.0 with interrupt pin configured. Four LEDs are connected in P2.0, P2.1, P2.2 and P2.3. When the push button is pressed the bounce rate is computed and stored as no of 1000s, no of 100s, no of 10s and ones. The stored value is seen in the LED by blinks for no of 1000s, 100s, 10s and 1s by its LED connected in P2.0, P2., P2.2 and P2.3 respectively.

Questions:

- 1) What does this statement P1IFG &= ~0x08" do?
- 2) What statement is executed after: P1OUT ^= LED;
- 3) Suppose the LED is OFF. What results if the ISR is executed 2 times?
- 4) State the execution of the above program when the button is not debounced
- 5) What are the steps that occur when an interrupt is processed?

- 6) How do you enable interrupts?
- 7) What all the conditions must be true for an interrupt to occur?

Case Study: MSP430 based embedded system application bringing up the salient features of GPIO, Watchdog timer, low power, FRAM etc.

2.10.2 Watchdog Timer

Introduction

A watchdog timer is one of the powerful peripherals in a microcontroller that can be used to automatically detect software anomalies such as being stuck in a continuous loop on execution and to reset the processor if such a problem occurs. So, the primary function of the watchdog timer (WDT) module is to perform a controlled system restart when a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not required in an application, the module can be disabled or configured as an interval timer and can generate interrupts at selected time intervals as a conventional timer.

In general, a watchdog timer runs in a controller based on a counter as shown in Figure 2.7.1, which counts the clocks from some initial value to maximum, when the timer expires (i.e. rolls over from zero), the CPU is reset and CPU starts running from the beginning of the program. The user program selects the counter's initial value and periodically clears the WDT value so as to not allow the WDT to cause the CPU to reset. If the counter ever reaches zero before the software clears it, the software is presumed to be malfunctioning and the processor's reset signal is asserted.

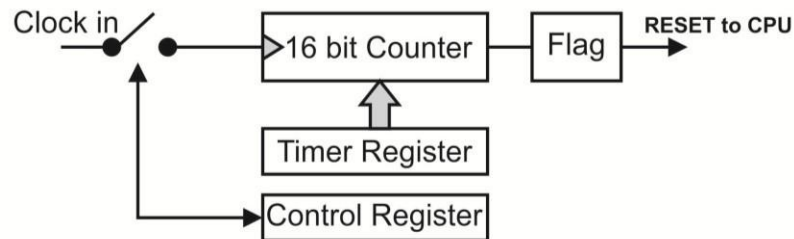


Figure 2.7.1- General Watch dog timer block diagram

Watch dog timer hardware in MSP430:

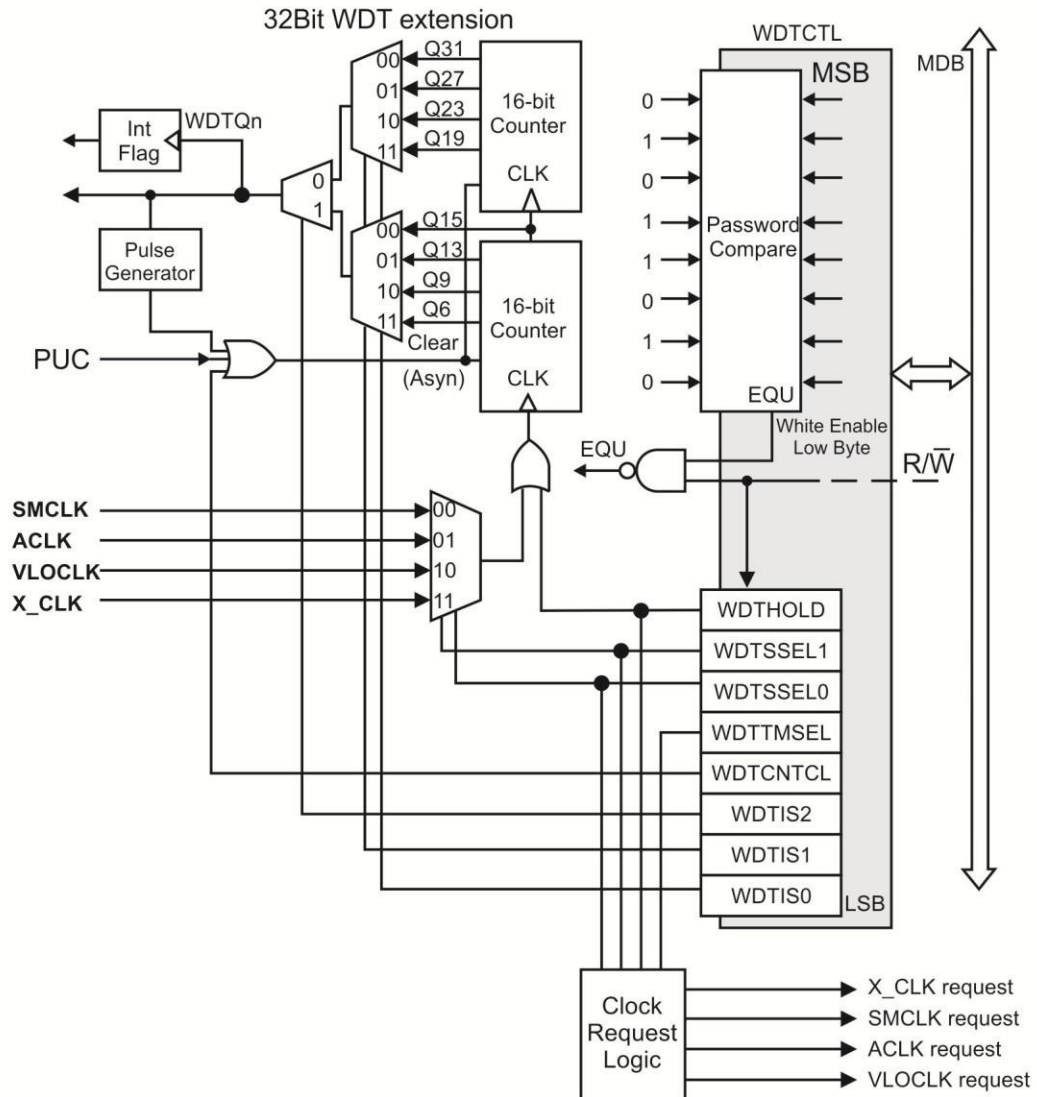


Figure 2.7.2 WDT hardware

Watch Dog Timer Registers

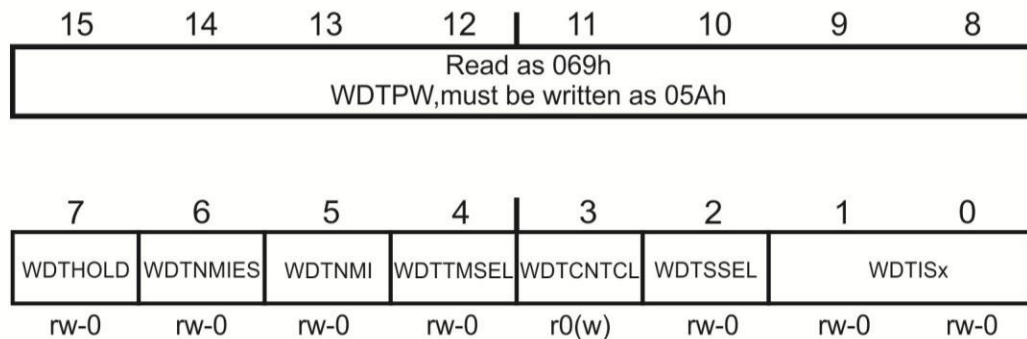
There are three registers for configuring and controlling the WDT

WDTCTL – Watch Dog Control register

IE1 – Interrupt Enable Register

IFG1 – Interrupt Flag register

1) WDCTL



WDTPW (15-8): Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC will be generated.

WDTHOLD (7): Watchdog timer hold. This bit stops the watchdog timer.

- 0 -- Watchdog timer is not stopped
- 1 -- Watchdog timer is stopped (WDT is not in use conserves power)

WDTNMIES (6): Watchdog timer NMI edge select. Selects the triggering edge for the NMI interrupt

- 0 -- NMI on rising edge
- 1 -- NMI on falling edge --- Modifying this bit can trigger an NMI

WDTNMI (5): Watchdog timer NMI select. Selects the function for the RST/NMI pin

- 0 -- Reset function
- 1 -- NMI function

WDTTMSSEL (4): Mode select.

- 0 -- Watchdog mode
- 1 -- Interval timer mode

WDCNTCL (3): Watchdog timer counter clear. Setting WDCNTCL = 1 clears the count value to 0000h. WDCNTCL is automatically reset.

- 0 -- No action
- 1 -- WDCNT = 0000h

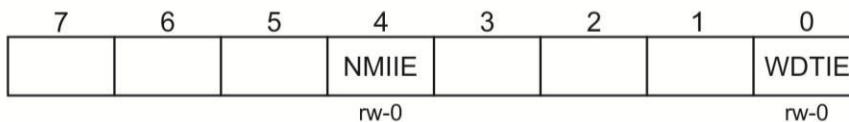
WDTSSSEL (2): Watchdog timer clock source select 0 – SMCLK.1 – ACLK

WDTISx (1-0): Watchdog timer interval select. Select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC.

- 00 -- Watchdog clock source /32768
- 01 -- Watchdog clock source /8192
- 10 -- Watchdog clock source /512
- 11 -- Watchdog clock source /64

Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte. Any write to WDTCTL with any value other than 05Ah in the upper byte is a security key violation and triggers a PUC system reset regardless of timer mode. Any read of WDTCTL reads 069h in the upper byte.

2) IE1



NMIIE (4): NMI interrupt enable. Enables the NMI interrupt. Set or clear this bit done by using BIS.B or BIC.B

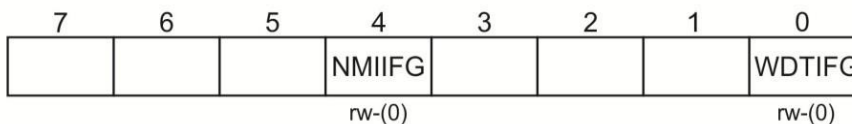
- 0 -- Interrupt not enabled
- 1 -- Interrupt enabled

WDTIE (0): Watchdog timer interrupt enable. Enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode.

- 0 -- Interrupt not enabled
- 1 -- Interrupt enabled

Bits 3-1 and 7-5 may be used by other MSP430 series.

3) IFG1



NMIIFG (4): NMI interrupt flag. NMIIFG must be reset by software.

- 0 -- No interrupt pending
- 1 -- Interrupt pending

WDTIFG (0): Watchdog timers interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software.

- 0 -- No interrupt pending
- 1 -- Interrupt pending

WDT Mode of Operation

WDT mode:

After a Power Up clear (PUC) condition, the WDT module is configured in the watchdog mode with an initial 32-ms reset interval using the DCOCLK. The user must setup, halt, or clear the WDT prior to the expiration of the initial reset interval or another PUC will be generated as in Figure 2.7.3. When the WDT 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. A PUC resets the WDT to its default condition and configures the RST/NMI pin to reset mode.

When the WDT used in an application, the WDT should be periodically cleared otherwise it resets the system.

Enabling WDT

As in following example program, one iteration of the while loop of the program execution takes maximum 10 milli seconds. It is assumed that a system stuck within the while loop for more than 50 milliseconds cannot be tolerated. Here is what is to be done: Configure the watchdog in such a way that it will reset your system when the count reaches 32768 - if the watchdog is running on a 1Mhz clock, this will take about 32 milli seconds. Now, simply adding a statement to the beginning of the program's main loop which will reset the watchdog count to 0 such a way that watchdog timer resets the system.

```
main( )
{
    while(1)
    {
        WDTCNT =0; // reset watchdog count to 0
        ....
        ....
        ....
    }
}
```

Figure 2.7.3 Program for clearing WDT

If the program is working perfectly, the watchdog count will never become a full scale value (32768) because one iteration of the loop will take only a maximum of 10 milli seconds, and there is a statement to reset the count to zero at the beginning of each iteration. If the system stuck(hangs) because of any reason, the reset watchdog-count-to-zero statement will not get executed within the next 10 milli seconds - the watchdog count will become 32768 (in 32 milli seconds) and this will result in a system reset. This ensures that the system will stay in the frozen state for only a time period well below the danger level.

Disabling WDT

When the watchdog functionality is not used in any of the application programs, it is essential that the watchdog timer is disabled otherwise, the processor will reset every 32 milliseconds and the program will seem to behave in mysterious ways. One way to disable the WDT is to write the below statement as the first line of main program.

WDTCTL = 0x5A00 | (1 << 7)

(OR)

WDTCTL = WDTPW | WDTHOLD

Interval Timer Mode

This mode is available in MSP430x2xxx series onwards. This mode can be used to provide periodic interrupts like conventional timer. The WDTIFG flag is set at the expiration of the selected time interval. A power on clear (PUC) is not generated in interval timer mode at expiration of the selected timer interval and the WDTIFG enable bit WDTIE remains unchanged. When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode

Accessing WDT-1

Blinking a LED using WDT in interval timer mode

The program shows how to use and configure

```
#include "msp430g2553.h"
unsigned int i=0;
void main(void)
{
    WDTCTL = WDT_MDLY_0_5; // WDT as interval timer (period 0,5 ms)
                        // WDTPW + WDTTMSEL + WDTCNTCL + WDTIS1
    IE1 |= WDTIE; // Enable WDT interrupt
    P1DIR |= BIT0+BIT6; // Set P1.0 & P1.6 as output
    P1OUT |= BIT0+BIT6; // P1.0 & P1.6 are ON at start
    _bis_SR_register(GIE); // Enable global interrupt
    while(1); // looping and doing nothing
}

// Watchdog Interval Timer interrupt service
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    if(i==1000)
    {
        P1OUT ^= BIT0; // Toggle LED1 (P1.0)
        i=0;
    }
    i++;
}
```

Figure 2.7.4 using WDT as interval Timer

The main function configures WDT as a watch dog mode (WDTTMSEL=1), clearing the initial value (WDTCNTCL=1), the watch dog timer interval of 512 (WDTIS1 = 1) and watch dog password of 5Ah (WDTPW) into the WDT control register (WDTCTL). The WDT interrupt is enabled in a separate statement with IE1 register. The global enable interrupt (GIE) bit in status register is enabled by the function as

“_bis_SR_register(GIE)”.

At last in the main function the CPU is in continues loop by executing while(1). When time is expired in WDT, the interrupt is generated and it call the interrupt service routine declared and defined next to the main function as

“ interrupt **void watchdog_timer(void)**”

The statements declaration

```
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
```

sets up the interrupt service routine for WDT. The definition of the function is to simply toggle the LED in P1.0. And the observations of the above program, The watchdog function manipulates two registers WDTCTL and IE1.

2.11 System clocks and Low power modes

MSP430 microcontroller has a powerful unified clock system (UCS) module which supports low system cost and ultra-low power consumption. The UCS module generates three types of clock signals and these clock signals are used to meet the major design targets of low system cost and low power consumption. This is done by optimizing system performance by choosing the proper clock signal for the required condition of operations. The UCS module can be configured to operate without any external components, with one external resistor, with one or two external crystals, or with resonators.

The UCS module in MSP430x5xxx series is having five clock sources (i.e. oscillator circuits) such as:

XT1CLK: Low-frequency or high-frequency oscillator

VLOCLK: Very low power, low-frequency oscillator

REFOCLK: low-frequency oscillator

DCOCLK: Digitally controlled oscillator

XT2CLK: High-frequency oscillator

Each of the clock sources generates the intermediate clock signals in the name of its own. In combination or individual operation of these oscillators, and proper clock scaling of the intermediate clock signals using clock divider, three clock signals are generated. They are:

ACLK: Auxiliary clock.

MCLK: Master clock. MCLK

SMCLK: Subsystem master clock

Among these clock signals, any one of the clock signals can be chosen for CPU operation and peripherals by proper clock configuration and selection by program. The internal block diagram of each clock source and clock distribution mechanism is discussed below.

2.11.1 Clock sources:

UCS modules have a variety of oscillators such as low frequency, high frequency crystal oscillators, and digitally controlled oscillators. Their descriptions are discussed one by one in the following.

2.11.1.1 XT1CLK, VLOCLK, REFOCLK sources:

The oscillator circuit as shown in Figure 2.8.1 consists of XT1CLK, VLOCLK and REFOCLK sources. The XT1 oscillator generates an ultra-low-current consumption low-frequency XT1CLK clock signal by using externally connected 32.768-KHz watch crystal and generates high-frequency XT1CLK signal by using standard crystals, resonators. The XT1 oscillator supports for external clock sources in the 4-MHz to 32-MHz range using bypass mode of operation.

The VLO oscillator generates low power 10-kHz frequency in the name of VLOCLK signal and the REFO oscillator generates internally trimmed 32.768-KHz frequency in the name of REFOCLK.

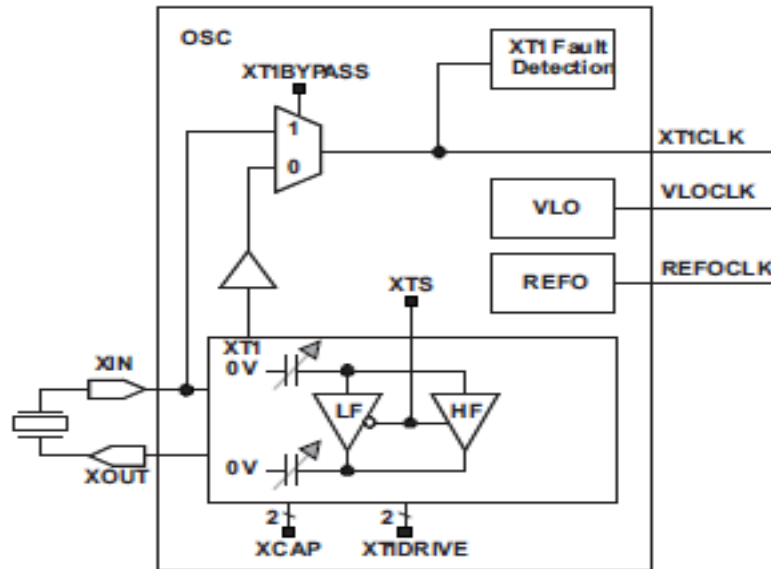


Figure 2.8.1 XT1CLK, VLOCLK, REFOCLK oscillator

2.11.1.2 DCOCLK (Digitally-Controlled Oscillator Clock) source:

The DCO is an integrated RC-type oscillator which generates DCOCLK signal. The frequency of DCOCLK varies with temperature, voltage, and from device to device. The DCO frequency can be adjusted by user program using the DCOx, MODx, and RSELx bits.

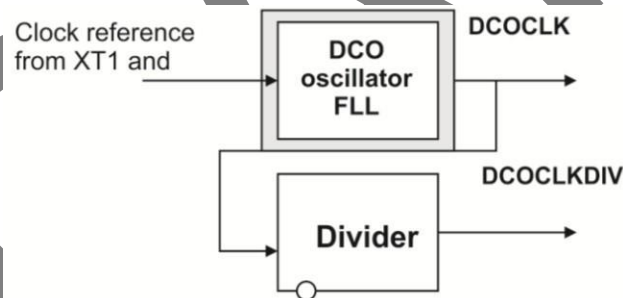


Figure 2.8.2 DCOCLK oscillator

The digital control of the oscillator allows frequency stabilization despite its RC-type characteristics using FLL hardware as shown in Figure 2.8.2. The XT1CLK can be used as a clock reference into the FLL which stabilizes the clock reference and generates DCOCLK. The DCOCLK signal is further divided by 1,2,4,8, 6 or 32 using clock divider and generates the DCOCLKDIV signal.

When power is switched on, DCO is configured with internal resistor and is start with mid-range frequency as reset default. Also MCLK and SMCLK are generated from DCOCLK in power up because of the CPU executes code from MCLK. The code execution begins from PUC in less than 6 us with MCLK.

The internal or external resistor can be selected with DCOR bit in BCSCT2 register by user program. The three DOCX bits in DCOCTL register select one of eight nominal frequency ranges. RSELx bits in BCSCTL1 into 8 frequency steps, separated by approximately 10% within the frequency range

selected by DOCX bits in DCOCTL registers. These frequency ranges are defined for an individual device in the device-specific data sheet.

The five modulators selection 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. The typical DCOx and RSELx ranges and steps are shown in Figure below and DCO frequency is different from MSP430 device to device.

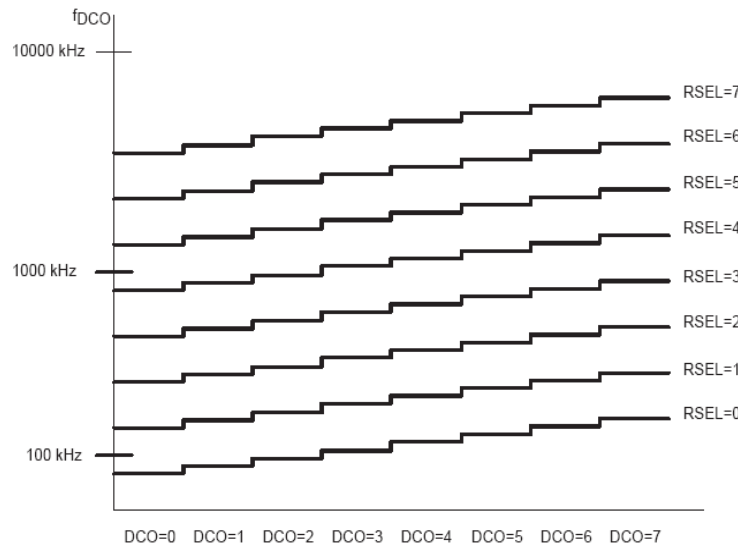


Figure 2.8.3 DCO range and steps

2.11.1.3 XT2CLK source:

XT2CLK is a 4 MHz to 32 MHz range high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources. XT2CLK can also be used as a clock reference into the FLL to generate DCOCLK.

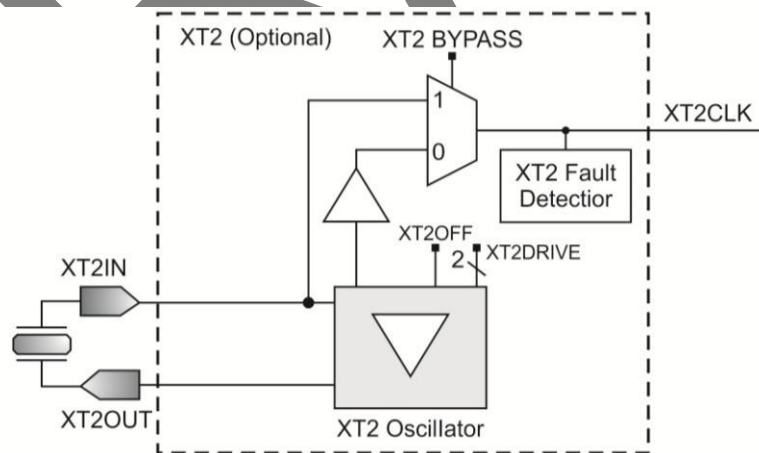


Figure 2.8.3 XT2CLK oscillator

2.11.2 Clock signals generation:

The UCS module at last generates three clock signals namely ACLK, MCLK and SMCLK from the individual or combination operation clock sources of the five oscillators discussed before. These three clock signals generated by the individual clock distributor hardware contain clock selector and clock divider as shown in Figure 2.8.4. The clock selector in each clock distributor selects any one the clock sources from the clock oscillators and the selected clock signal is further divided by 1,2,4,8,16 or 32 using clock divider hardware then it outputs the individual clock signal as in the name of ACLK, MCLK and SMCLK.

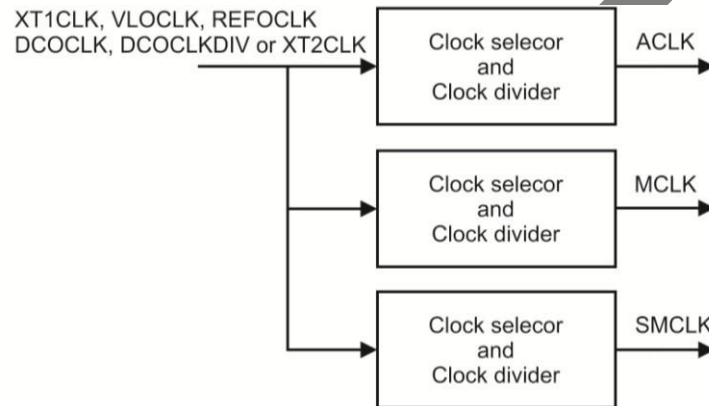


Figure 2.8.4 Clock distributor

The ACLK output signal is further divided by 1, 2, 4, 8, 16 or 32 using additional hardware and new clock signal in the name of ACLK/n is generated. ACLK is software selectable by individual peripheral modules. MCLK is used by the CPU and system. SMCLK is software selectable by individual peripheral modules.

2.11.3 UCS Operation

After a PUC, the UCS module default configuration is:

- XT1 in LF mode is selected as the oscillator source for XT1CLK. XT1CLK is selected for ACLK
- DCOCLKDIV is selected for MCLK.
- DCOCLKDIV is selected for SMCLK.

On power up, MCLK and SMCLK signals are derived from DCOCLK module at 800 kHz and ACLK signal is derived from LFXT1 in LF mode. Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the basic clock module. DCO control register (DCOCTL), Basic clock system control 1 (BCSCTL1), and Basic clock system control 2 (BCSCTL2) registers configure the basic clock module.

Features in Low-Power operation

ACLK can be configured to oscillate with a low-power 32,768-Hz watch crystal, providing a stable time base for the system and low power stand-by operation. The MCLK can be configured to operate from the on-chip DCO that can be only activated when requested by interrupt-driven events. The SMCLK can be configured to operate from a crystal or the DCO, depending on peripheral requirements.

Fail safe operation and Oscillator Fault Detection

The basic clock module is featured with an oscillator-fault detection fail-safe operation. The oscillator fault detector is designed with an analog circuit that regularly monitors the XT1CLK (in HF mode) and the XT2CLK modules. If there is no presence of clock signals from those clock modules, circuit detects

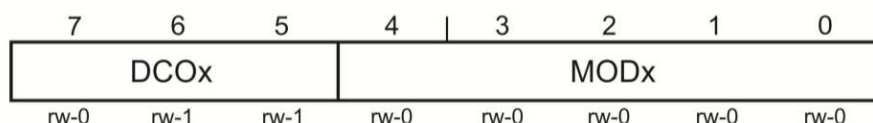
as fault. During the fault in those clocks, MCLK signal generation is automatically switched to DCO module to allow code execution to continue by CPU even though the crystal oscillator has stopped from generation of clock signal.

The Basic clock module is facilitated with two interrupt registers namely Interrupt Enable1 register (IE1) and Interrupt Flag Register 1 (IFG1) to allow the CPU to be interrupted when oscillator fault occurs. When oscillator fault occurs, the Oscillator fault interrupt flag (OFIFG) in IFG1 register is set there by NMI interrupt generated, provided interrupt enable bit (OFIE) in IE1 register is set. The NMI interrupt service routine can test the OFIFG flag to determine if an oscillator fault has occurred. The OFIFG flag must be cleared by software.

UCS Module Registers

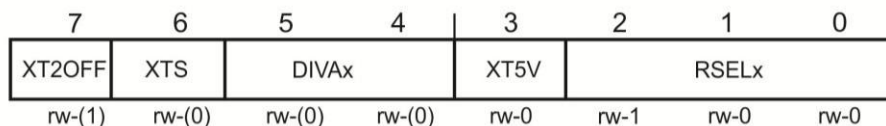
There are five registers associated with the basic clock module such as DCOCTL1, BCSCCTL1, BCSCCTL2, IE1 and IFG1. The description of each bit in the registers is discussed below.

DCO Control Register (DCOCTL)



- 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 fDCO+1 frequency is Modulator selection. These bits define how often the fDCO+1 frequency is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32-MOD) the fDCO frequency is used. Not useable when DCOx=7

Basic Clock System Control Register 1 (BCSCCTL1)



- 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 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.

BCSCTL2, Basic Clock System Control Register 2



SELMx Bits Select MCLK. These bits select the MCLK source.

7-6 00 DCOCLK

01 DCOCLK

10 XT2CLK when XT2 oscillator presents on-chip. LFXT1CLK when XT2 oscillator not presents on-chip.

11 LFXT1CLK

DIVMx BitS Divider for MCLK

5-4 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 presents on-chip. LFXT1CLK when XT2 oscillator not presents on-chip.

DIVSx BitS Divider for SMCLK

2-1 00 /1

01 /2

10 /4

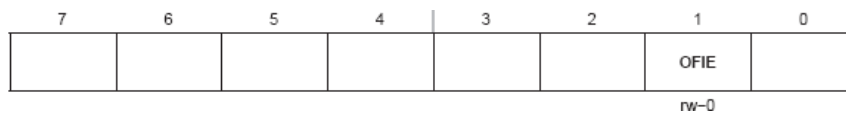
11 /8

DCOR Bit 0 DCO resistor select

0 Internal resistors

1 External resistor

IE1, Interrupt Enable Register 1



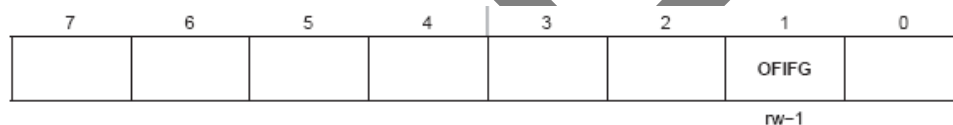
Bits 7-2 These bits may be used by other modules. See device-specific datasheet.

OFIE Bit 1 Oscillator fault interrupt enable. This bit enables the OFIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions.

0 Interrupt not enabled
1 Interrupt enabled

Bits 0 This bit may be used by other modules. See device-specific datasheet.

IFG1, Interrupt Flag Register 1



Bits 7-2 These bits may be used by other modules. See device-specific datasheet.

OFIFG Bit 1 Oscillator fault interrupt flag. Because other bits in IFG1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions.

0 No interrupt pending
1 Interrupt pending

Bits 0 This bit may be used by other modules. See device-specific datasheet.

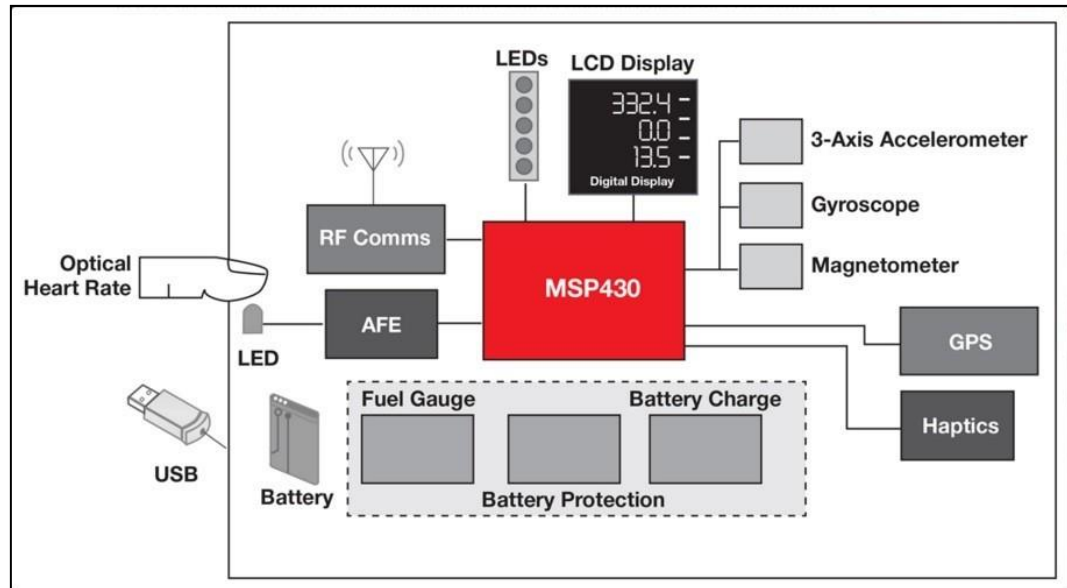
2.12 Requirements of Low Power embedded system

The various embedded applications and products like access control & security systems, electronic locks, electric meters, fitness and activity trackers, pulse oximeter etc. have requirements of ultralow power MCUs. These devices are usually powered by battery source. So there is necessity of minimum power consumption. The battery capacity is limited; so to make device run for a long time, it is required to have a low power microcontroller to design similar embedded systems. The requirement of low power embedded system is justified by considering the following benefits:

- Longer battery life
- Less electronic waste
- Less power dissipation in device
- Less heat dissipation, therefore heat sink requirement can be minimized

One of the ultralow power applications is Activity/Fitness Trackers. This is one of the wearable devices, which is used for fitness monitoring. It keeps track of fitness related parameters such as distance walked or run, heart rate, blood pressure etc. The block diagram Activity/Fitness Tracker is shown in fig.

In this device, heart rate monitor is used to provide the reading for heart rate and blood pressure. For the speed and motion reading we have an accelerometer, gyroscope and magnetometer. There is a provision of LCD displays, which displays real time measured parameter values. Here the USB peripheral allows the device to download the data and update the device software. As this device is operated by battery, it requires an ultralow power MCU to keep it running for longer hours.



Broadly, there are different kinds of methods to minimize the power consumption of microcontroller by its operating modes. Basically MSP430 works on two types of mode:

- 1) Active Mode
- 2) Low Power Mode

Low power modes are classified into different categories depending upon the different clock condition that is explained in next section of this chapter.

The operating modes take into account three different needs:

- Ultralow-power
- Speed and data throughput
- Minimization of individual peripheral current consumption

MSP430 microcontroller is a mixed signal processor having power efficient digital and analog signal processing capability by having various digital and analog signal supported peripherals with different kind of low power modes of operation supported.

2.12.1 Low power modes in MSP430

The MSP430 microcontroller is designed for ultralow-power applications. It offers the lowest power consumption and the perfect mix of integrated peripherals for a variety of battery operated applications. MSP430 microcontroller has various power operating modes such as Active Mode (AM), Low power modes (LPM) 0, 1, 2, 3, 4, and 3.5 and 4.5. AM is the normal running mode and all LPM for low power consideration modes. The low power operating modes consider the needs of ultralow-power consumption, speed, data throughput and minimization of individual peripheral current consumption.

Status Register (SR) of MSP430 has some of the control bits used for selecting a particular low power operating modes and the description of the bits are shown in Table 2.9.1.

Bit in Status Register	Description
CPUOFF (CPU off)	Turns off the CPU
OSCOFF (Oscillator Off)	Turns off the XT1 (LFXT1) oscillator
SCG0 (System Clock Generator 0)	Turns off the DCO dc generator
SCG1 (System Clock Generator 1)	turns off the SMCLK

Table 2.9.1 Description of control bits in SR for Low power modes

By properly setting and clearing the above bits in status register, the MSP430 gets into a desired low power mode. The advantage of including the CPUOFF, OSCOFF, SCG0 and SCG1 mode control bits in the SR is that the present operating mode is saved onto the stack during an interrupt service routine. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine.

When setting any of the mode control bits, the selected operating mode takes effect immediately. The configuration for the Low power modes, status of CPU and clock generator in each low power modes are described in the Table 2.9.2.

SCG1	SCG0	OSCOFF	CPUOFF	Modes	Status of CPU and Clock generator
0	0	0	0	AM	CPU is active. All enabled clocks sources are active. CPU and MCLK are disabled.
0	0	0	1	LPM0	All other enabled clock sources 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 or ACLK.
1	0	0	1	LPM2	FLL is disabled if DCO is disabled SMCLK and ACLK are active.
1	0	0	1	LPM3	CPU, MCLK, SMCLK are disabled. DCO and DC generator are enabled if it is used for ACLK
1	1	1	1	LPM4	FLL is disabled if DCO is disabled ACLK is active.
					CPU, MCLK, SMCLK and DCO are

	1				disabled.
	1				ACLK is active.
					CPU and all clocks disabled

Table 2.9.2 Configuration of Low power modes

Peripherals are enabled and disabled by its clock source or controlled with their individual control register settings. In Low power modes, peripherals operating with any disabled clock are disabled until the clock becomes active. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

The flow chart as shown in Figure 2.9.1 depicts the flow of execution from a running active mode to various available lower modes and vice versa. After the power up clear (PUC), the execution gets into Active mode in which all clock sources and CPU are active and consume power. When the microcontroller gets into any one of the low power modes by programming the corresponding LPM configuration as discussed in Table 2.9.2 is made automatically. When the microcontroller is entering into any one of the LPM then the CPU is intentionally disabled as in the configuration of LPM. Wakeup the CPU by getting into active mode (AM) from LPM0 to LPM4 is possible through all enabled interrupts.

As an example, the microcontroller gets into LPM3 from its active mode and no execution is carried out by CPU, when an enabled interrupt from a peripheral or by an external pin is occurred then execution goes to active mode by pushing the current LPM3 configuration into stack and servicing to the interrupt. After servicing to the interrupt again microcontroller gets into LPM3, there by the power consumption by CPU during no interrupt is reduced and this claims low power modes of operation.

DRAFT

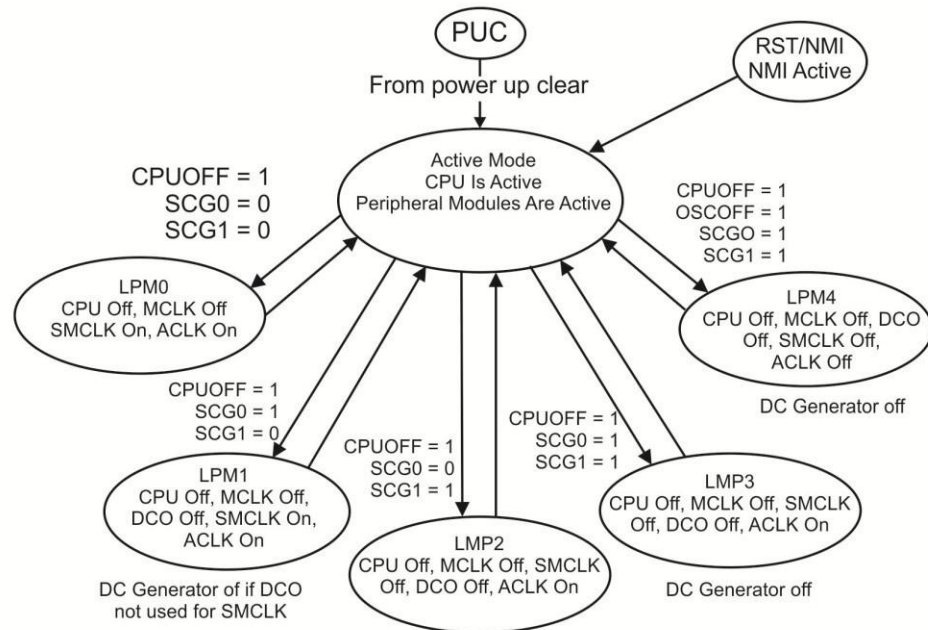


Figure 2.9.1 Flow chart of transition Active mode to LPM and vice versa

MSP430x5xxx and higher order series have LPM3.5 and LPM 4.5. When the controller is entered into any one of those modes, the voltage regulator of the Power Management Module (PMM) is disabled. All the RAM, register and IO contents are lost, the I/O pin states are locked upon LPMx.5 entry. Wakeup from LPM4.5 is possible by the power sequence, a RST event, RTC or from specific I/O. The more description about LPM3.5 and 4.5 are discussed in its user manual and datasheet.

2.12.2 Entering and Exiting Low-Power Modes

The Figure 2.9.2 illustrates the programming flow that when the controller is in main program, it gets into interrupt service routing ISR and returning to the same LPM. Flow Diagram is with reference to the application Activity Tracker discussed in introduction to Low Power. The working of whole system can be understood by this flow diagram. From this flow we can easily understand which programming part can be put in ISR and main program.

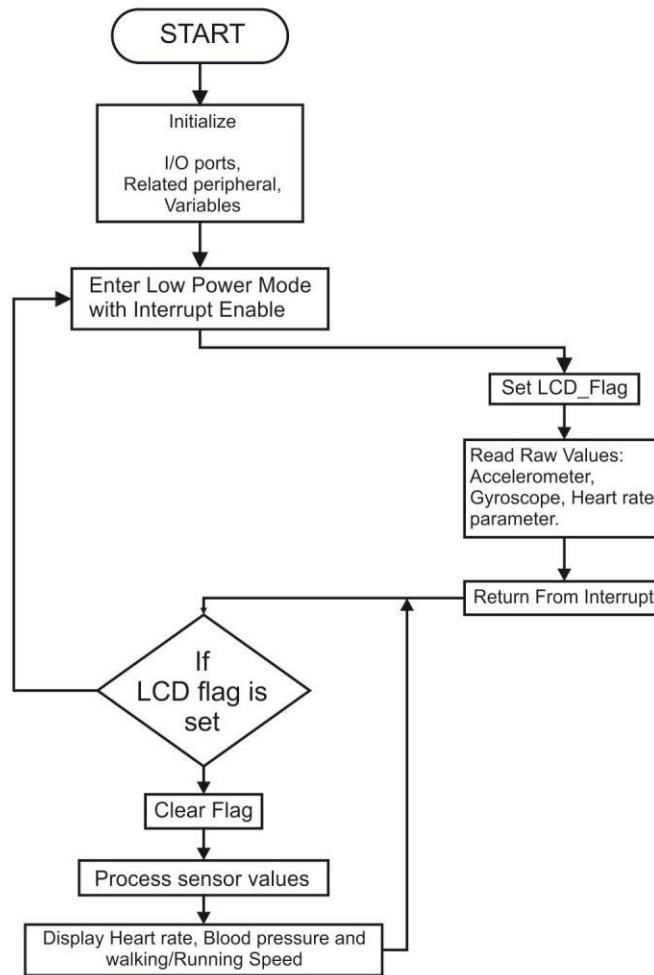


Figure 2.9.2 Flow chart for handling LPM by MSP430

But it is possible to use a single power supply to obtain V+ and V- too as shown in Fig 1.9. In this circuit, R must be greater than 10KΩ and capacitors provide decoupling of power supply and may range from 0.01μF to 10μF.

Upon interrupt by the enabled interrupt source, the Program counter PC (return address is stored in this register) and Status register SR (LPM configuration is stored in this register) are pushed to the stack. The CPUOFF, SCG1, SCG0 and OSCOFF bits are automatically reset; then execution goes to the corresponding interrupt service routine. After servicing to the interrupt and returning to main, the original PC and SR value is popped from the stack for restoring the previous operating mode. The SR bits stored in the stack can be modified within the interrupt service routine and it can take a different low power operating mode when the execution is returned from interrupt.

The following statements are used in the CCS compiler to configure the low power modes in MSP430. By invoking the first statement in a program, the bits CPUOFF, SCG1, SCG0 and OSCOFF in SR register are set to logic 1 and the microcontroller is getting into low power mode 4. The corresponding configuration is discussed in Table 2.9.2.

```
_BIS_SR(LPM4_bits);
_BIC_SR(LPM4_bits);
```

By invoking the second statement, all the bits CPUOFF, SCG1, SCG0 and OSCOFF in SR register are cleared and the microcontroller is exiting from low power mode 4. The following program illustrates that how the microcontroller is entering into the LPM as shown in Figure 2.9.3.

```
#include < MSP430xxx.h>
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
    P1DIR = 0x01; // P1.0 output
    P1REN |= 0x10; // P1.4 enabled for pullup/down
    P1OUT = 0x10; // P1.4 selected for pullup

    P1IE |= 0x10; // P1.4 Interrupt is enabled
    P1IES |= 0x10; // P1.4 Falling edge triggered interrupt
    P1IFG &= ~0x10; // P1.4 interrupt flag IFG is cleared
    _BIS_SR(LPM4_bits + GIE); // Enter LPM4 and enable global interrupt
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    P1OUT ^= 0x01; // P1.0 = toggle
    P1IFG &= ~0x10; // P1.4 IFG cleared
}
```

Figure 2.9.4 Program for entering and exiting from a LPM

In the above program shown in Figure 2.9.4, the processor stays in LPM4. Once interrupt on P1.4 occurs the interrupt service routine is called and after returning from interrupt service it stays in LPM3. In the above program, If the statement “_BIC_SR(LPM4_bits);” is invoked instead of the statement “_BIS_SR(LPM3_bits);” in the ISR, then the processor is exit from LPM4.

2.12.3 Principles for Low-Power Applications

The most important method for minimizing power consumption is maximizing the clock system time in LPM3 or LPM4 whenever possible. In LPM3, CPU is active by ACLK and the power consumption is very low in the order of few μA with the both a real-time clock function and all interrupts active. A 32-kHz watch crystal is used for the ACLK and the CPU is clocked from the DCO. In LPM 4, the CPU and all clock sources are disabled and these are enabled and starts consume power in interrupt service routine. The following ideas will direct the programmer to have better power reduction.

- Turn off clock for CPU when no work to do.
- Provide scaled clock frequencies for CPU and peripherals for the required cases.
- Use interrupts to wake the processor and then control the flow of program.
- Peripherals should be enabled only when needed.

- Use low power inbuilt peripheral modules and configure it to run autonomously. As an example Timer_A and Timer_B automatically generate PWM output and capture timing without CPU resources.
- Polling the pins and flags has to be avoided strictly. Prefer interrupt driven method for all peripherals function like TIMER, ADC, UART, SPI and I2C as well as other modules as much as possible. Build the application to operate asynchronously as much as possible.
- Polling the flag can be avoided by invoking calculated branching.
- Instead of doing long software calculations by CPU, the faster look-up table can be used.
- Avoid frequent subroutine and function calls.
- For longer software routines, single-cycle CPU registers should be used.

2.12.4. Summary and conclusion

The chapter is discussed with various battery operated systems and the static and dynamic power consumption of the system. The flow chart for operation of MSP430 in active mode to LPM and vice versa is shown. How the power is reduced by means of various low power modes present in MSP430 with sample programs is illustrated. Also it discussed for methods to follow in programming to design the system for power reduction.

Low power modes are predominantly used in MSP430 to support us to conserve energy in a battery operated application. In some critical application, upon interrupt in LPM, the delay in getting into active mode to handle the interrupt is too long. Slower clock such as 32.768 kHz oscillator takes much longer to stabilize approximately in milliseconds but the faster clock from crystal oscillator takes very few microseconds. For this reason, the programmer prefers to wake up the CPU from LPM the faster clock source is used in active mode and we do pay a penalty for this. This penalty is not only the time it takes to go from a low power mode to Active Mode but it is that how long stays in active mode. The deeper the time we go into the low power modes, the longer it takes to go to Active Mode makes the system in power optimistic.

In another aspect, understanding the variants of peripherals with various modes of its operation towards low power and the methods of handling those peripherals with interrupt driven for an application will make us to develop the program for power optimized application.

2.13. Active vs Standby current consumption

We already discussed the operating modes of the microcontroller. So according to that, current can be classified as:

- 1) Active current
- 2) Standby current

Active Current: It is the maximum current that a system can draw, when all the clocks are activated. This is a maximum current because all the modules of the system are active.

Standby current: It is the minimum current that a system can draw when clocks are deactivated depending upon the Low power modes. This is the minimum current that a system can draw with respect to low power mode configured.

There are many configurations for the MSP430 in terms of operating frequency, supply voltage, operating mode, and active peripherals. Because of this, only some combinations current consumption characteristics are enumerated in the device datasheet. Current consumption for the MSP430 is found from the datasheet of the device. It lists numbers for current consumption against each operating mode with certain frequencies and operating voltages. Additionally each peripheral's current consumption of the particular device is listed separately in its corresponding section of the datasheet. Adding these values can give a good estimate of total current consumption for the system and you can choose the power supply and battery for the system. Experimental testing is needed to determine the actual consumption for your system by connecting ammeter in series with power source. The average system power consumption is the absolute lowest, without compromise in performance. The system

enters and remains in an ultra-low power standby mode as long as possible, and is awakened only to service interrupts as fast as possible. Multiple oscillators are utilized to provide both an ultra-low power standby mode and high-performance processing.

The clock system is very flexible and allows the MSP430 to operate optimally from a single 32 KHz crystal to the internal digitally controlled oscillator (DCO) used for the CPU and high-speed peripherals. A low-frequency Auxiliary Clock (ACLK) is driven directly from a common 32KHz watch crystal with no additional external components. The ACLK enables the MSP430's ultra-low power standby mode (LPM3) and an internal real-time clock function. In LPM3, the MSP430 typically consumes current in the 1µA range. The integrated high-speed DCO can source the master clock (MCLK) used by the CPU and high-speed peripherals. In design, the DCO is active and fully stable in less than 6µs with no intermediate steps. This enables the CPU to conduct instant high-performance processing without waiting for a long start-up for a second crystal or second speed start-up. Because the DCO is digitally adjustable with software and hardware, stability over time and temperature is assured.

The operating modes of MSP430 series microcontrollers are for three different needs such as

- Ultra-low power
- Speed and data throughput
- Minimization of individual peripheral current consumption

As in the data sheet of MSP430G2553, in Active mode the current consumption varies from 225µA to 340µA against the operating voltage that varies from 2.2V to 3V. Similarly current consumption by other low power modes are given in the Figure 10.1

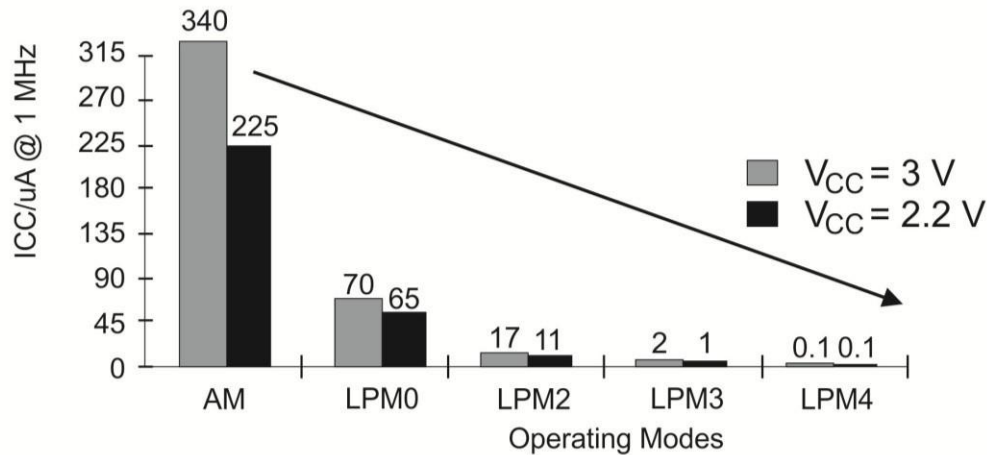


Figure 2.10.1 Current consumption over different operating modes

From the above Figure 2.10.1, it is shown that MSP430 can operate in several low power modes such LPM 0 to 4. The mode operation in LP is 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. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine. The mode-control bits and the stack can be accessed with any instruction.

When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. The peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

Active mode current consumption:

The current consumption by the active mode is explained in the following table

Test condition	Operating voltage	Current in μA
Active mode	2.2V	225 - 250
$f_{\text{DCO}}=f_{\text{MCL}}=f_{\text{SMCLK}}=1\text{MHz}$ $f_{\text{ACLK}}=0$	3V	340 - 420
Program excution in Flash		

Table 10.1 current against to modes and frequency of operation

The test is made by connecting all inputs to logic 0 and output to be floated. The current consumption by varying supply voltage against to change in DCO frequency is shown in the following Figure 10.2. The current reading is plotted at the temperature of 25 degree Celsius.

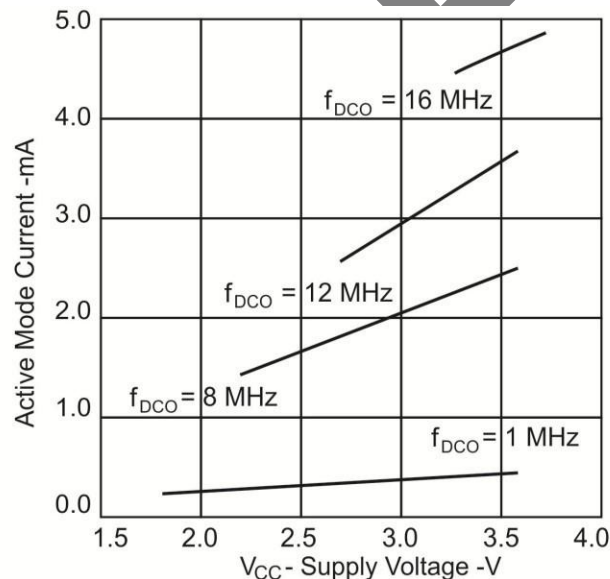


Figure 10.2 Active mode current against to operating voltage and Frequency.

2.14. Ferroelectric Random Access Memory (FRAM)

FRAM is a non-volatile embedded memory technology and is known for its ability to be ultra-low power and also for being the most flexible and easy-to-use universal memory solution available today. This application report is intended to give new FRAM users and those migrating from flash-based applications knowledge on how FRAM meets key quality and reliability requirements such as data retention and endurance.

Introduction to FRAM:

Ferroelectric Random Access Memory (FRAM) is an ultra-low power nonvolatile memory technology with write speeds akin to static RAM (SRAM). The technology has been in the industry for over a decade, implemented as stand-alone memory. FRAM's first introduction as an embedded memory in a general purpose ultra-low-power MCU was on Texas Instruments 16-bit MSP430™ product line, the MSP430FR57xx family.

Some of the important attributes of FRAM are:

- 1) FRAM is non-volatile; that is, it retains its contents on loss of power.
- 2) FRAM is a true random-access memory. The memory is not segmented; addressing of data for read or write at word or byte level happens directly in the same way as SRAM.
- 3) The embedded FRAM on MSP430 devices can be accessed (read or write) at a maximum speed of 8 MHz. Above 8 MHz, wait states are used when accessing FRAM. Typical write speeds can exceed 2 MBps with FRAM compared to approximately 14 kBps on flash devices
- 4) Writing to FRAM and reading from FRAM require no setup or preparation such as pre-erase before write or unlocking of control registers.
- 5) FRAM write accesses are extremely low power, because writing to FRAM does not require a charge pump.
- 6) FRAM writes can be performed across the full voltage range of the device.
- 7) FRAM meets and exceeds reliability requirements on data integrity. It provides practically unlimited endurance for read and write operations on the order of 10^{15} write or erase cycles.

FRAM Characteristics:

Data Retention:

The basic goal of performing data retention tests is to ensure that FRAM can meet the retention specification for nonvolatile memory characterized by a ten-year lifetime at 85°C.

FRAM Data Retention Test Flow

To clarify the test procedure for data retention, the following two terms are defined:

- *Same-state* refers to the logic state of FRAM; that is, the state of polarization of the ferroelectric crystal prior to the high-temperature bake when testing for imprint.
- *Opposite-state* refers to the polarization of the crystal in a direction opposite to that in which it was imprinted.

Figure 1 shows the test procedure for data retention. To test for imprint, a data pattern with a set logic state is written onto FRAM, and then the device is exposed to a high-temperature bake at 125°C. This bakes the bit-cells in one logic state referred to as same-state. This bake is followed by a read-restore to further strengthen the same-state data. Following this, opposite-state data is written to FRAM. This is followed by a thermal depolarization bake to weaken/stress the opposite-state data. This bake is performed at the maximum operating temperature for the device (85°C for the MSP430). After the depolarization bake is completed, the data is read out to ensure the integrity of the opposite-state data. The test is then repeated until the testing time is reached.

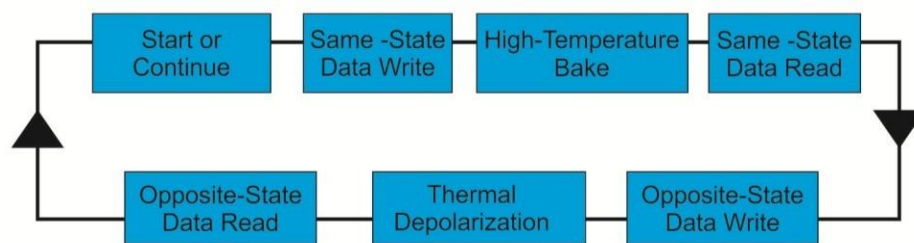


Figure .11.1. Test flow FRAM data Retention

The opposite-state bake identifies potential imprint issues if any while the read ensures that the FRAM cells have retained the ability to be polarized with opposite-state data as well preserved enough polarization to be read without data loss. MSP430 FRAM data retention is tested for a cumulative bake time of 1000 hours at 125°C. See the device-specific data sheet for data retention (retention).

Data retention duration T	25° C	100	Years
	70° C	40	
	85° C	10	

Table 11.1 Data retention table

Endurance

Flash memory is specified in terms of write endurance, which is defined as the measure of the number of erase and write cycles that a flash array can achieve while retaining data integrity. Per IEEE Standard Definitions, endurance is defined as "The measure of the ability of a nonvolatile memory device to meet its data-sheet specification as a function of accumulated nonvolatile data changes."

Flash memory on MSP430 flash-based devices has a typical write endurance of 10⁵ write or erase cycles. In comparison, the endurance of a single FRAM cell in the MSP430FRxx family is 10¹⁵ write or erase cycles. However, FRAM is inherently different from flash in that the total endurance is impacted by read cycles as well. As explained before, every read of an FRAM byte also necessitates a write-back and, therefore, there is no difference between read and write endurance on FRAM.

Any access from FRAM, either a read or a write, has an impact on the endurance. On the other hand, the endurance of flash-based devices is affected only by a write access. The following section describes the impact of read and writes cycles on lifetime endurance.

To comprehend the endurance specification on FRAM, it is important to know that while the flash limit is derived from experimental results that show a failure of flash memory beyond 10⁵ write or erase cycles, there is no known fail limit in the case of FRAM. Therefore, it is possible to state in clear terms that FRAM is capable of exceeding 10¹⁵ write or erase cycles. The exact minimum endurance is not yet quantifiable, because testing to the fail limit on FRAM requires several years. As of this writing, no failure has been detected on FRAM units that have executed continuous read and write cycling. Therefore, based on test data, Texas Instruments is confident in specifying the *minimum* endurance for MSP430 FRAM to be 10¹⁵ write or erase cycles.

Calculating the Impact of Read and Write Cycles on Endurance

To understand the impact of read/write cycles on FRAM it is important to know that:

- Every read of FRAM includes a write-back, so FRAM is being written to every time it is accessed, irrespective of whether the access is a read or a write.
- The number of instructions executed cannot be directly related to the number of FRAM accesses on the MSP430FRxx device. This is because the FRAM controller uses a four-word two-way associative cache to buffer instructions before execution. Therefore, instructions are fetched 64 bits at a time and executed from cache memory until a cache miss occurs. When there are no more instructions to execute in the cache and a FRAM access is required to execute the next instruction, it is termed as a cache miss. The access of the cache is transparent to the application and the filling of the cache occurs within the time taken for one FRAM access cycle. Hence, we can see that in a typical use case, for every 4-word (64-bit) instruction that executed, only one access needs to be performed to FRAM.

However, it is possible to stress the device by using code that causes a cache miss on every access. While this implementation is not practical in a real-world application, it is done by reaching outside the 64-bit boundary every time an instruction is executed.

For example, to calculate the worst-case endurance for location A:

Location A is accessed once every 1.125 μs or (0.888 × 10⁶) times per second.

The total number of accesses allowed at location A (as specified in the data sheet) is 10^{15} .

Time to reach the minimum endurance limit = $10^{15} / (0.88 \times 10^6)$ seconds, which is approximately 36 years.

Flash Memory

Flash memory is a memory storage device for computers and electronics. It is most frequently used in devices like digital cameras, USB flash drives, and video games. It is quite similar to EEPROM. It is a widely used, reliable, and flexible non-volatile memory to store software code and data in a microcontroller.

Flash memory is different from RAM because RAM is volatile (not permanent). When power is turned off, RAM loses all its data. Flash can keep its data intact even without any power. A hard drive is also permanent (non-volatile) storage, but it is bulky and fragile. Flash memory is slower than RAM, but faster than hard drives. It is much used in small electronics because it is small and has no moving parts.

The main weakness of flash memory is that it is more expensive than hard drives for the same amount of storage. Another weakness of flash memory is the number of times that data can be written to it. Data can be read from flash as many times as desired, but after a certain number of "write" operations, it will stop working. Most flash devices are designed for about 100,000 - 1,000,000 write operations (or "write cycles").

EEPROM has the same limitation that flash does: it can only survive about 100,000 write cycles. But it is more expensive than flash, so it is rarely used for storage greater than 128kB. The main difference between EEPROM and flash memory is that EEPROM can "write" to any byte of memory, at any time. Flash memory can only write to an entire chunk, or "sector", of memory at a time. That means that if the user wants to change only one byte, flash must also re-write all the bytes in that sector. This means that flash memory can wear out faster than EEPROM. Flash Memory can store as many as hundreds of GBs. Many are in the form of a pen drive.

Flash memory is used in USB Drives, SSD Drives, computer RAM (occasionally), hybrid drives (small SSD + Hard Drive), graphics cards, and memory cards.

The smallest unit that can be erased in a flash memory is a complete flash segment. Erasing flash means generating logical 1s in the memory. The MSP430 main flash memory has a segment size of 512 bytes.

All MSP430 devices also have some smaller 64-byte or 128-byte flash segments. This area of the flash is called the information memory. In most applications, the main memory stores program code and constant data values, and the information memory is used for calibration data, serial numbers, etc. Except for the segment size, there is no difference between the main and the information memory. Hence, both can store program code or constant data values or both. Programming flash memory generates logical 0 values in the flash memory cell. Programming can be performed bit, byte, or word wise. Flash segment size has no influence on programming of flash cells.

Only logical 0 can be written as single bits, bytes, or words. Because writing logical 1 in flash memory is only possible with erasing the memory, 1s can only be written segment-wise. A high voltage is needed to program and erase flash memory. All MSP430 devices have an integrated charge pump and automatically generate this high voltage, when mandated. Also, a flash timing generator for automatic generation of erase and programming sequences is integrated in every MSP430. Hence, MSP430 flash programming is done without any external components. For programming and erasure, it only is important to keep the supply voltage VCC within the data sheet limits and to program the input clock frequency of the flash timing generator in accordance with data sheet requirements.

Flash Endurance

Like any other erasable memory, flash devices have a limited number of erase/write cycles they can withstand without failure. The reason for the limitation depends on either charge trapping characteristics or the dielectric breakdown characteristics of the tunnel oxide. This introduces a term called endurance.

Endurance is a measure of the number of erase/write cycles that a flash array can achieve while retaining data integrity.

PARAMETER	MIN	NOM	MAX	UNIT
Program/erase endurance	10 ⁴ 10 ⁵			Cycles

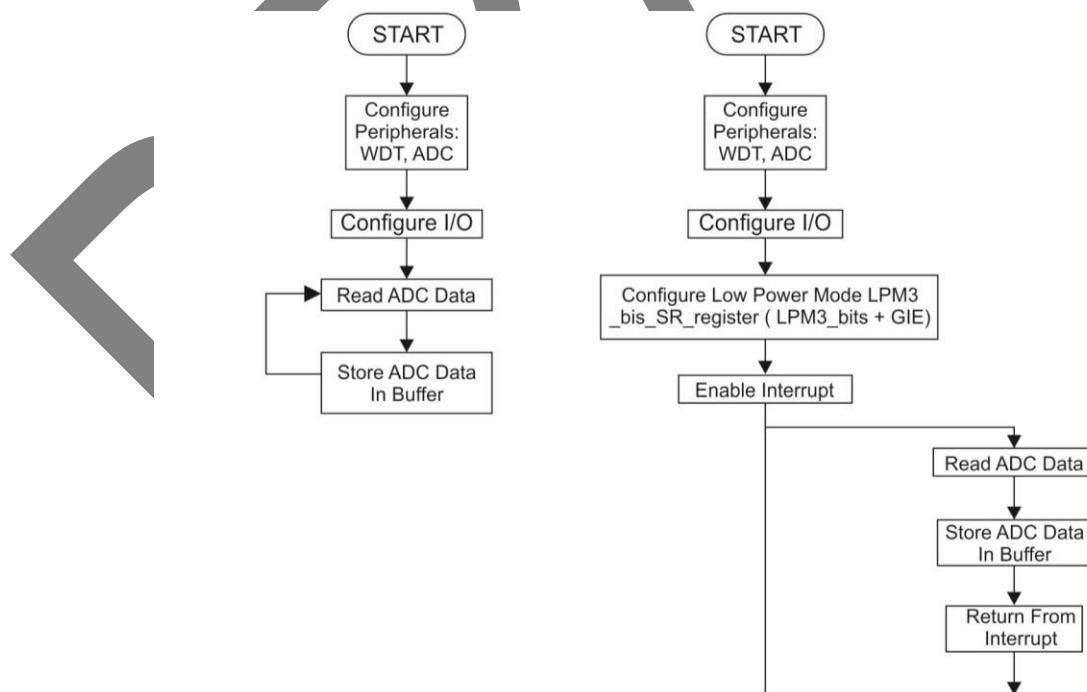
Table 11.2. Flash Endurance in the Data Sheets

The data sheet specifies a minimum of 10 000 read/write cycles, while the devices typically fulfill 100 000 read/write cycles.

MSP430 endurance testing shows that most devices easily achieve more than 100 000 cycles at ambient or high temperature, while the number of cycles at low temperature is in the range of a few ten thousands. Failures are always a single bit failing erase.

Measurements show improvement in the number of cycles if a quiescent period after erase operations is added. The most common problem is stuck cells caused by charge trapping. Charge trapping occurs in the insulating tunnel oxide during erase operations, causing the cell to read a logic 0 although erased. This is a self-healing effect and usually detraps automatically in the quiescent period after an erase cycle. During endurance testing at Texas Instruments, the flash cells are continuously erased and rewritten. With a delay of at least one or two seconds between two erase/write cycles, the flash endurance increases significantly during the tests.

Energy and power consumption estimation of MSP430 Launchpad:



Performance of any device depends upon the energy and power consumption. We know that power is dependant upon the operating voltage and current. Operating voltage of the whole system is fixed, so the power depends upon the current drawn by the system. For the calculation of power, we need to estimate the current. MSP430 microcontrollers have Active mode and Low power modes. So, we will calculate the Active Current and Standby current.

For understanding the power saving of the system in Low Power modes we should perform the same experiment in both the modes. For example, we are reading and storing ADC data. Operation in both the modes can be understood by the flow diagram shown in fig.

To make system configure in Low Power mode we need to add “_bis_SR_register(LPM3_bits + GIE);” within the while loop and will automatically work in Low power mode. In both the modes we have to observe current, in case of MSP430G2553 the estimated current is given in table below:

Platform	Active current consumption	Stand by current consumption
MSP430G2553	123 μ A	1.2 μ A

Energy trace technology is provided to calculate the power and energy of the device. Energy Trace technology is included in Code Composer Studio version 6.0 and newer. It requires specialized debugger circuitry, which is supported with the second-generation on-board eZ-FET flash emulation tool and second-generation standalone MSP-FET JTAG emulator. Target power must be supplied through the emulator when Energy Trace is in use.

2.15. Summary

DRAFT

2.16. Review questions

DRAFT

2.17. Exercises

DRAFT

Advanced features of MSP430 microcontroller

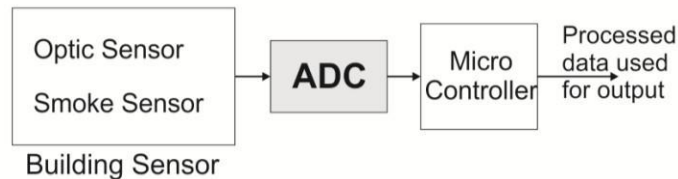
The previous chapter provided a deep understanding of peripherals and the programming concepts of the MSP430 microcontroller. It also covered topics such as memory mapped peripherals followed by the system registers used to control GPIOs, interrupts and watchdog timer as well as system clocks present in the device.

In this chapter, readers will learn about some more advanced features of the MSP430 microcontroller. At the end of the chapter, the reader will be able to have an understanding of timers and Real time Clock; pulse width modulation control and its applications; ADC and comparator as well as data transfer using the DMA module. Additionally, the chapter will also cover the programming methodology of peripherals for optimal power management.

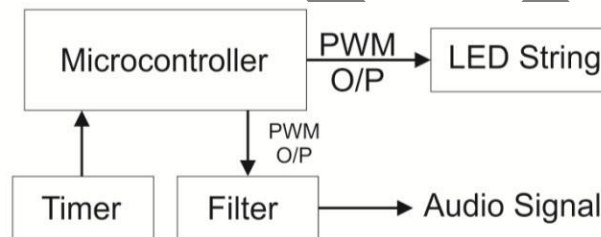
Topic	Page
3.1. Introduction.....	91
3.2. Timer and RTC	91
3.3. Timer_A	95
3.4. Real Time Clock (RTC)	106
3.5. Pulse Width Modulation.....	113
3.6. Timing generation and measurement.....	117
3.7. Analog interfacing and acquisition.....	118
3.8. Direct Memory Access (DMA)	147

3.1. Introduction

In today's world, the increased use of real time systems requires a large number of high accuracy peripherals. For example, in building automation applications such as smoke detectors or occupancy & motion detectors, analog sensors such as optic, smoke need to be used. The received signal requires some external peripherals such as ADC to convert the data to the digital form to be processed by the processor as shown in the figure below.



In other applications such as audio playback modules and multi string LED drivers, timers are used to produce PWM signals which are required to control the intensity of light and sound as shown below.



Microcontrollers, being a digital device, need peripherals like ADC, comparators and timers (to produce PWM control) to communicate with the sensor data. The data needs to be understood by the device and to supply variable outputs. These external peripherals incur heavy cost when used separately, which results in an inefficient system being developed in terms of cost and size. With the decreasing chip size, microcontrollers in the current market are being loaded with a large number of advanced peripherals in addition to regular attributes such as GPIOs, system clocks and memory. Microcontrollers now comprise most of the necessary peripherals, which have made the development of an embedded system more effective and area & cost efficient.

The main focus of this chapter is to understand the technical intricacies of the on-chip peripherals present in TI's MSP430 low power microcontrollers and the programming methodologies for low power consumption.

3.2. Timer and RTC

3.2.1. Timer Basic

Timer hardware in a microcontroller is fundamentally a counter driven by an internal or external clock signal. This counter is commonly incremented (up counter) or decremented (down counter) on each clock tick of the clock source. When the counter reaches the predefined set value, the timer can generate an interrupt to the CPU. In the interrupt service routine, the CPU does the required job associated with the timer. The timer is used for generating delay and counting external events. The delay is used for performing some execution on a regular basis (blink a LED every second).

The timer is an indigenous hardware component separately without intervention of CPU's normal execution. Timers generally have counter registers with the size of 8 or 16 bits, 8 bit timer is capable of holding value within 0-255 similarly 0-65535 for 16 bits timer as in figure below. This register can be initialized to any value by the program. The value initialized in the register is increased (if up counter) or decreased (if down counter) automatically at a predefined rate given by the clock source. Once the

register gets roll over (reaching to zero value and starts new counting), it sets a flag called timer overflow flag (T1OFG) and there by the processor is interrupted from its current program to execute the service program (ISR) associated with timer.

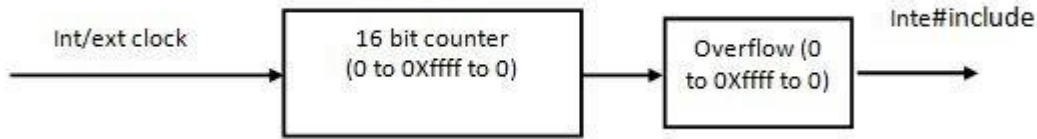


Figure 3.1 Timer and its buffer overflow mechanism

The program for responding to overflow on timer can be executed by just polling the overflow flag or by the interrupt generated from enabling the corresponding Timer overflow interrupt. In the service routing the timer register is reinitialized for the required timer value.

As an example,

A 16 bit timer generates overflow for every 100ms, it gets clock for every 1 ms (1KHz). What is the value to be initialized in timer register? FF9B

The timer value = $0xFFFF - \text{hex equivalent of initial timer value} + 1$

Initial timer value = required time / clock time period

Required time = 100ms and clock time period = 1ms (1 KHz)

Initial timer value = $100\text{ms} / 1\text{ms} = 100 = 0x64$

The timer value = $0xFFFF - 0x64 + 1$ (1 count added because of timer overflow happens when it Reaches) = 0xFF9C

The timer hardware contains counter circuitry. When the counter counts for a fixed clock, it is called the time peripheral while the internal clock is always configured to fixed clock. If the circuit counts for an external non periodic clock signal (event), then it is called counter. The timer hardware is configured as a counter for counting external event which is non-periodic.

MSP430 controller has timers named Basic Timer1, Timer A and Timer B.

3.2.2. Basic Timer 1

The Basic Timer1 generates LCD control signal and low frequency time intervals. The Basic Timer1 has two independent 8-bit timers that can also be cascaded to form one 16-bit timer function. Basic Timer1 can be used as a real-time clock (RTC) function with software time increments.

Basic Timer1 features include a selectable clock source, two independent cascadable 8-bit timers, interrupt capability and LCD control signal generation. The block diagram of the basic timer1 is given below in Figure 3.2.

3.2.3. Basic Timer1 Operating modes

The Basic Timer1 module can be configured as two 8-bit counter/timers as counter 1 and counter2 by BTCNT1 and 2 registers respectively or one 16-bit timer by BTCTL register. The Basic Timer1 controls the LCD frame frequency with basic timer counter.

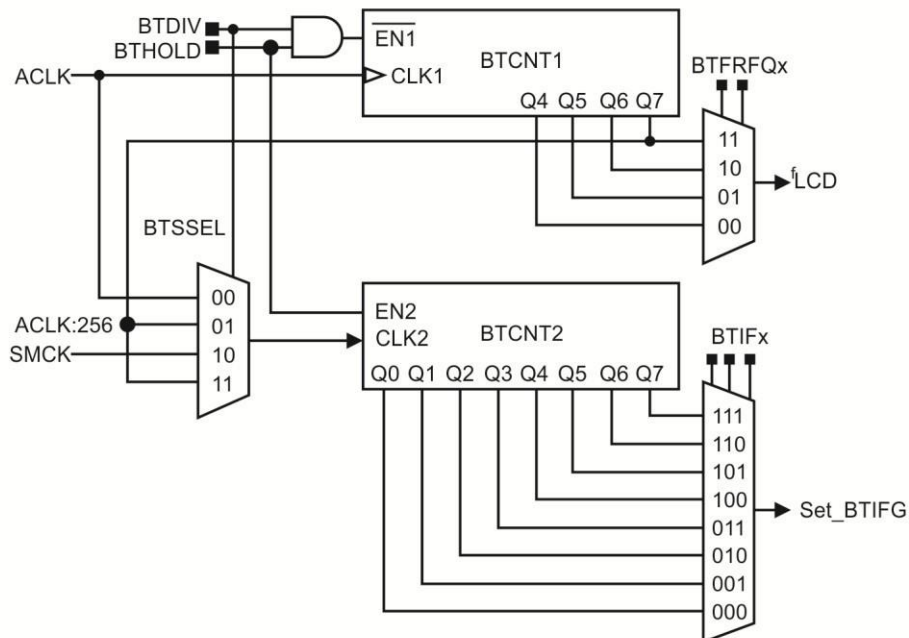


Figure 3.2 Block Diagram of Basic Timer1

3.2.3.1. Basic Timer1 Counter One (BTCNT1)

BTCNT1 is an 8-bit timer/counter directly accessible by software. It is incremented with ACLK clock source and provides the frame frequency for the LCD controller. It can be stopped by setting the BTHOLD and BTDIV bits in BTCNT1 register.

3.2.3.2. Basic Timer1 Counter Two (BTCNT2)

BTCNT2 is an 8-bit timer/counter directly accessible by software. It can be sourced by clock from ACLK or SMCLK, or from ACLK/256 when cascaded with BTCNT1. The BTCNT2 clock source is selected with the BTSSSEL and BTDIV bits. BTCNT2 can be stopped to reduce power consumption by setting the HOLD bit. BTCNT2 uses the Basic Timer1 interrupt, BTIFG. The interrupt interval is selected with the BTIPx bit.

3.2.3.3. 16-Bit Counter Mode

The 16-bit timer/counter mode is selected when control the BTDIV bit is set. In this mode, BTCNT1 is cascaded with BTCNT2. The clock source of BTCNT1 is ACLK, and the clock source of BTCNT2 is ACLK/256.

3.2.4. Basic Timer1 Interrupts

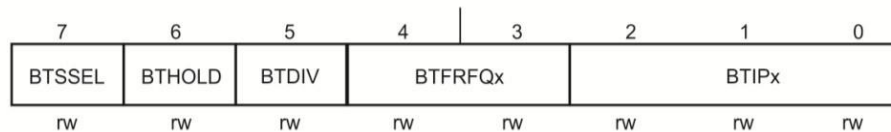
The **Basic Timer1 uses two bits in the SFRs for interrupt control. These are Basic Timer1 interrupt flag, (BTIFG), located in interrupt flag register (IFG2.7) and Basic Timer1 interrupt enable (BTIE), located in interrupt enable register (IE2.7).** The BTIFG flag is set after the selected time interval and requests a Basic Timer1 interrupt if the BTIE and the GIE bits are set. The BTIFG flag is reset automatically when the interrupt is serviced, or it can be reset with software.

3.2.5. Basic Timer1 Registers

Register	Short form	Register type	Address	Initial State
Basic timer1 Control	BTCTL	Read/write	040h	Unchanged
Basic Timer1 Counter 1	BTCNT1	Read/write	046h	Unchanged
Basic Timer1 Counter 2	BTCNT2	Read/write	047h	Unchanged
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	Reset with PUC

3.2.5.1. Basic Timer1 Control Register (BTCTL)

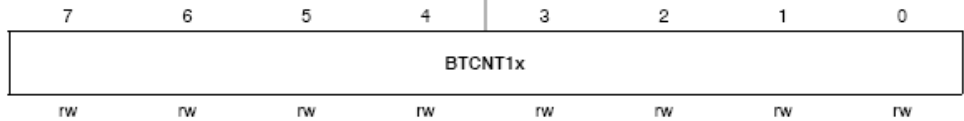
BTCTL, Basic Timer1 Control Register



NAME	BITS	DESCRIPTION															
BTSSSEL	7	BTCNT2 clock select. This bit, together with the BTDIV bit, selects the clock source for BTCNT2. See the description for BTDIV.															
BTHOLD	6	Basic Timer1 hold 0 -BTCNT1 and BTCNT2 are operational, 1-BTCNT1 is held if BTDIV=1 & BTCNT2 is held															
BDIV	5	Basic Timer1 clock divide. This bit together with the BTSSSEL bit, Selects the clock source for BTCNT2 <table border="1" style="margin-left: 20px; margin-top: 10px;"> <thead> <tr> <th>BTSSSEL</th> <th>BDIV</th> <th>BTCNT2 Clock Source</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">ACLK</td> </tr> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">ACLK/256</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">SMCLK</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">ACLK/256</td> </tr> </tbody> </table>	BTSSSEL	BDIV	BTCNT2 Clock Source	0	0	ACLK	0	1	ACLK/256	1	0	SMCLK	1	1	ACLK/256
BTSSSEL	BDIV	BTCNT2 Clock Source															
0	0	ACLK															
0	1	ACLK/256															
1	0	SMCLK															
1	1	ACLK/256															
BTFRFQx	4-3	fLCD frequency. These bits control the LCD update frequency. 00 -fACLK/32, 01 -fACLK/64, 10 -fACLK/128, 11 -fACLK/256															
BTIPx	2-0	Basic Timer1 interrupt interval 000 -fCLK2/2 001 -fCLK2/4 010 -fCLK2/8 011 -fCLK2/16															

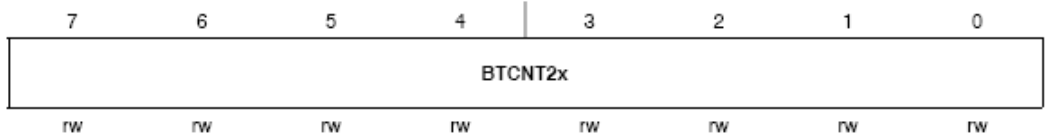
		100 -fCLK2/32	101 -fCLK2/64	110 -fCLK2/128	111 -fCLK2/256
--	--	---------------	---------------	----------------	----------------

3.2.6. RTCNT1, RTC Counter 1, Counter Mode



BTCN1X	Bits 7-0	BTCNT1 register. The BTCNT1 register is the count of BTCNT1.
---------------	-----------------	--

3.2.7. BTCNT2, Basic Timer1 Counter 2



BTCN2X	Bits 7-0	BTCNT2 register. The BTCNT2 register is the count of BTCNT2.
---------------	-----------------	--

3.3. Timer_A

Timer_A exists with a 16-bit timer/counter and three capture/compare registers. Timer_A can support multiple capture/comparers, PWM outputs, and interval timing. Timer_A also has extensive interrupt capabilities. Interrupts may be generated from the timer on overflow and from each of the capture/compare registers conditions.

Timer_A features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Three configurable capture/compare registers
- Configurable outputs with PWM capability
- Asynchronous input and output latching
- Interrupt mechanism for fast decoding of all Timer_A interrupts

The block diagram of Timer A is shown in Figure 3.3.

The timer register, TAR is incremented or decremented with each rising edge of the clock signal. TAR can be read or written with software. TAR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode. The Timer should be in OFF condition before modifying its configuration and its operation.

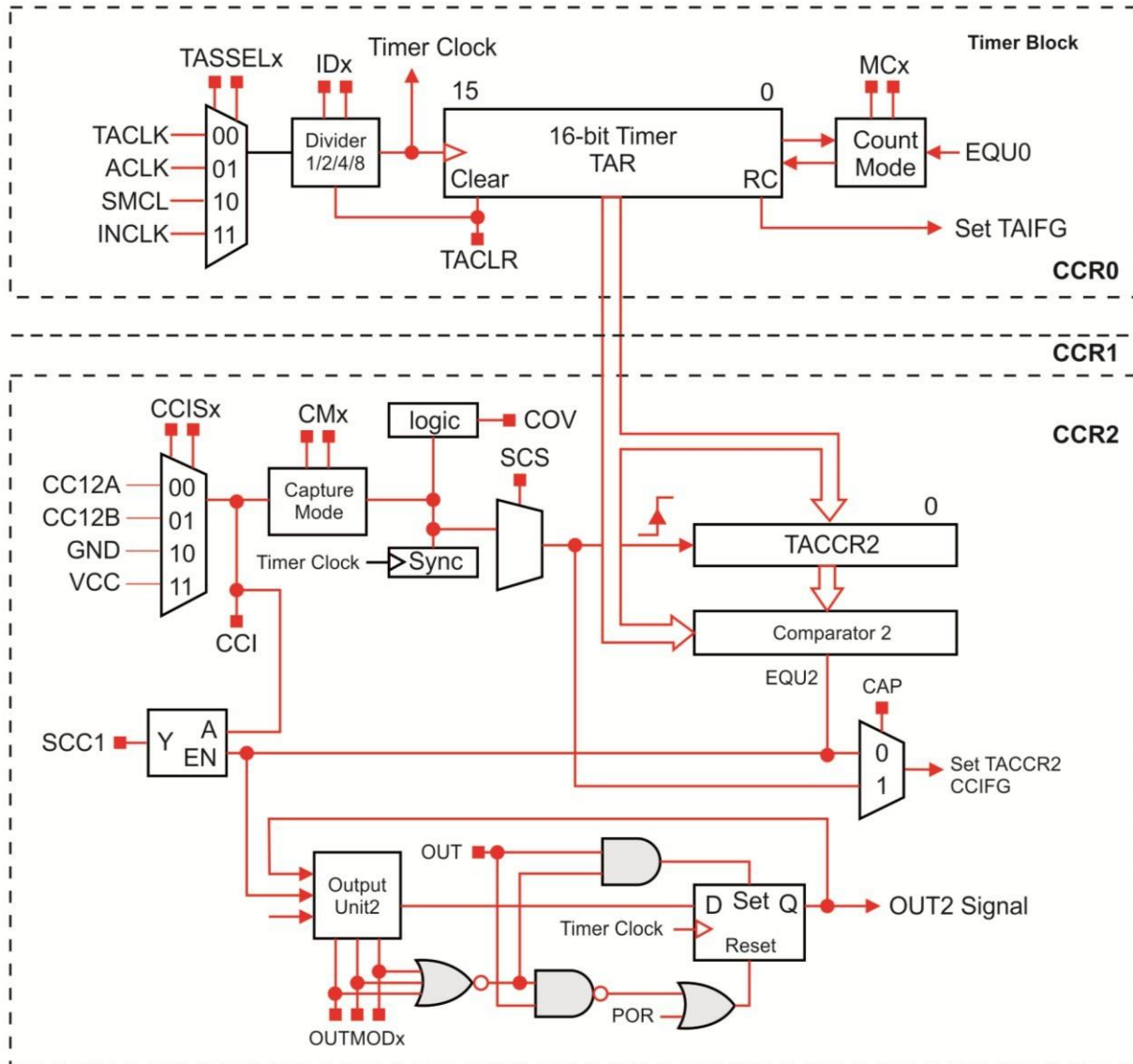


Figure 3.3 Block diagram of Timer A

3.3.1. Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TACLK or INCLK and it is selected with the TASSELx bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits.

3.3.2. Timer operating Modes

The timer has four modes of operation as described in the table. The mode is selected with the MCx bits in Timer A Control Register (TACTL).

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0.
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and back down to zero.

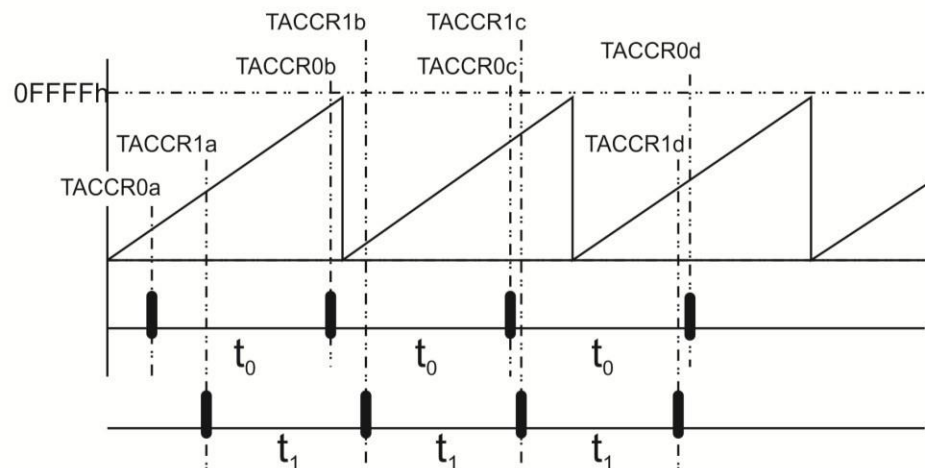
Table 3.2 Timer A - Modes of operation

3.3.2.1. Up Mode

The timer repeatedly counts up to the value of compare register TACCR0, which defines the period. The number of timer counts in the period is TACCR0+1. When the timer value equals TACCR0 the timer restarts counting from zero. The TACCR0 CCIFG interrupt flag is set when the timer counts to the TACCR0 value. The TAIFG interrupt flag is set when the timer counts from TACCR0 to zero. When TACCR0 is modified while running, the timer counts up to the new period and if the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

3.3.2.2. Continuous Mode

The timer repeatedly counts up to 0FFFFh and restarts from zero. The capture/compare register TACCR0 works the same way as the other capture/compare registers. The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. This mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TACCRx register in the interrupt service routine. Figure below shows two separate time intervals t_0 and t_1 being added to the capture/compare registers. In this, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to three independent time intervals or output frequencies can be generated using all three capture/compare registers.


Figure 3.4 Continuous Mode Time Intervals

3.3.2.3. Up/Down Mode

The timer repeatedly counts up to the value of compare register TACCR0 and back down to zero. The period is twice the value in TACCR0. The TACCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by 1/2 the timer period. The TACCR0 CCIFG interrupt flag is set when the timer *counts* from TACCR0-1 to TACCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h.

3.3.3. Capture/Compare Blocks

Three identical capture/compare blocks, TACCRx, are present in Timer_A. Any of the blocks may be used to capture the timer data, or to generate time intervals.

3.3.3.1. Capture Mode

Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CClxA and CClxB are connected to external pins or internal signals and are selected with the CCISx bits. The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture occurs:

- 1) The timer value is copied into the TACCRx register
- 2) The interrupt flag CCIFG is set

The input signal level can be read at any time via the CCI bit. MSP430x1xx family devices may have different signals connected to CClxA and CClxB. The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. Capture over flow (COV) bit is set if a second capture was performed before the value from the first capture was read. COV must be cleared by software.

3.3.3.2. Compare Mode

The compare mode is selected when CAP = 0. The compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TAR *counts* to the value in a TACCRx:

- 1) Interrupt flag CCIFG is set
- 2) Internal signal EQUx = 1
- 3) EQUx affects the output according to the output mode
- 4) The input signal CCI is latched into SCCI

3.3.3.3. Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals.

3.3.3.4. Output Modes

The output modes are defined by the OUTMODx bits and are described in the table below. The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0 because EQUx = EQU0.

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer <i>counts</i> to the TACCRx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer <i>counts</i> to the TACCRx value. It is reset when the timer <i>counts</i> to the TACCR0 value.
011	Set/Reset	The output is set when the timer <i>counts</i> to the TACCRx value. It is reset when the timer <i>counts</i> to the TACCR0 value.
100	Toggle	The output is toggled when the timer <i>counts</i> to the TACCRx value. The output period is double the timer period.
101	Reset	The output is reset when the timer <i>counts</i> to the TACCRx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer <i>counts</i> to the TACCRx value. It is set when the timer <i>counts</i> to the TACCR0 value.
111	Reset/Set	The output is reset when the timer <i>counts</i> to the TACCRx value. It is set when the timer <i>counts</i> to the TACCR0 value.

Output in Timer in Up Mode

The OUTx signal is changed when the timer *counts* up to the TACCRx value, and rolls from TACCR0 to zero, depending on the output mode. An example is shown in Figure below using TACCR0 and TACCR1.

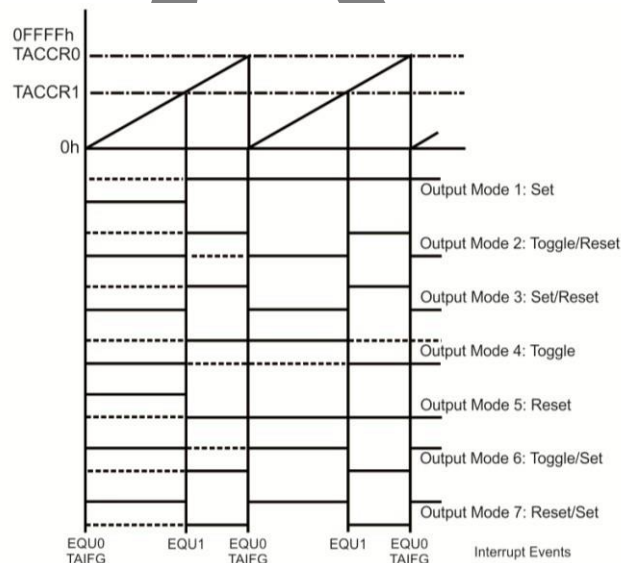


Figure 3.5. Output Example—Timer in Up Mode

Output when Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the TACCRx and TACCR0 values, depending on the output mode. An example is shown in Figure below using TACCR0 and TACCR1.

Output Example—Timer in Up/Down Mode

The OUTx signal changes when the timer equals TACCRx in either count direction or when the timer equals TACCR0, depending on the output mode. An example is shown in Figure below using TACCR0 and TACCR2.

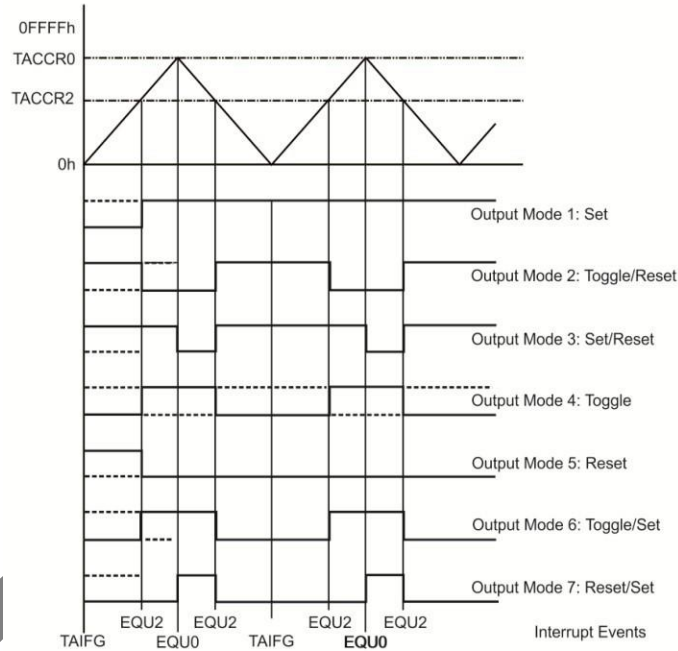


Figure 3.6 Output Example—Timer in Up/Down Mode

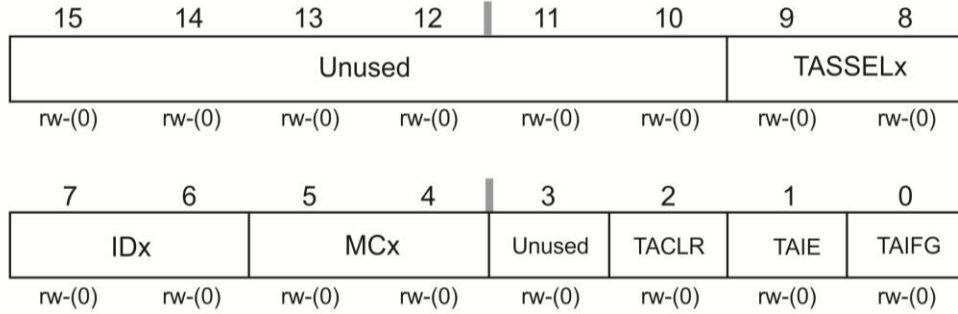
3.3.4. Timer_A Interrupts

Two interrupt vectors are associated with the 16-bit Timer_A module; they are TACCR0 interrupt vector for TACCR0 CCIFG and TAIV interrupt vector for all other CCIFG flags and TAIFG. In capture mode, any CCIFG flag is set when a timer value is captured in the associated TACCRx register. In compare mode, any CCIFG flag is set if TAR counts to the associated TACCRx value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set. The TACCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector. The TACCR0 CCIFG flag is automatically reset when the TACCR0 interrupt request is serviced. The TACCR1 CCIFG, TACCR2 CCIFG, and TAIFG flags are prioritized and combined to source a single interrupt vector TAIV. Any access, read or write, of the TAIV registers 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, if the TACCR1 and TACCR2 CCIFG flags are set when the interrupt service routine accesses the TAIV register, TACCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TACCR2 CCIFG flag will generate another interrupt.

3.3.5. Timer_A Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control	TACTL	Read/write	0160h	Reset with POR
Timer_A counter	TAR	Read/write	0170h	Reset with POR
Timer_A capture/compare control 0	TACCTL0	Read/write	0162h	Reset with POR
Timer_A capture/compare 0	TACCR0	Read/write	0172h	Reset with POR
Timer_A capture/compare control 1	TACCTL1	Read/write	0164h	Reset with POR
Timer_A capture/compare 1	TACCR1	Read/write	0174h	Reset with POR
Timer_A capture/compare control 2	TACCTL2	Read/write	0166h	Reset with POR
Timer_A capture/compare 2	TACCR2	Read/write	0176h	Reset with POR
Timer_A interrupt vector	TAIV	Read only	012Eh	Reset with POR

3.3.5.1. Timer_A Control Register (TACTL)

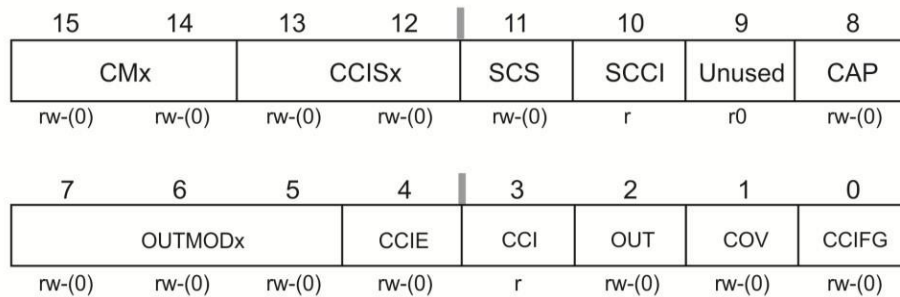


Name	Bits	Description
Unused	15-10	Unused
TASSELx	9-8	Timer_A clock source select 00-TACLK, 01-ACLK, 10-SMCLK and 11-INCLK
IDx	7-6	Input divider. These bits select the divider for the input clock 00 - /1, 01 - /2, 10 - /4, 11 - /8
MCx	5-4	Mode control 0- Stop mode: the timer is halted 1- Upmode: the timer counts up to TACCR0 10-Continuous mode: the timer counts upto 0FFFFh 11-Up/down mode: the timer counts up to TACCR0 then down to 0000h
Unused	3	Unused

TACLR	2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero
TAIE	1	Timer A interrupt enable. It enables the TAIFG interrupt request 0-interrupt disabled, 1-interrupt enabled
TAIFG	0	Timer_A interrupt flag 0-no interrupt pending,1-interrupt pending

3.3.5.2. Timer_A Register (TAR)

TARx Bits 15-0 Timer_A register. The TAR register is the count of Timer_A.

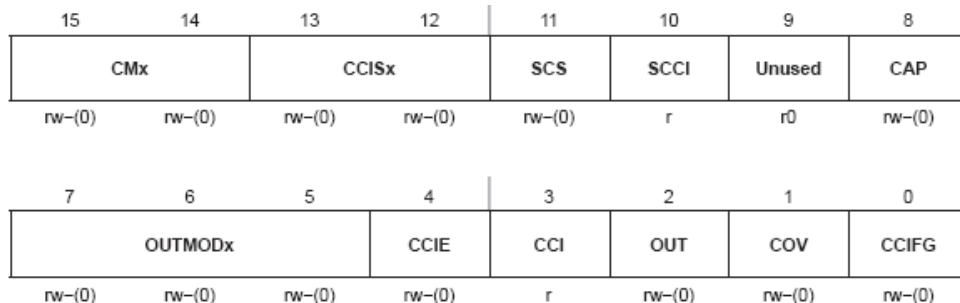


Name	Bits	Description
Unused	15-10	Unused
TASSELx	9-8	Timer_A clock source select 00-TACLK, 01-ACLK, 10-SMCLK and 11-INCLK
IDx	7-6	Input divider. These bits select the divider for the input clock 00 - /1, 01 - /2, 10 - /4, 11 - /8
MCx	5-4	Mode control 0- Stop mode: the timer is halted 1- Upmode: the timer counts up to TACCR0 10-Continuous mode: the timer counts upto 0FFFFh 11-Up/down mode: the timer counts up to TACCR0 then down to 0000h
Unused	3	Unused
TACLR	2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero
TAIE	1	Timer A interrupt enable. It enables the TAIFG interrupt request 0-interrupt disabled, 1-interrupt enabled
TAIFG	0	Timer_A interrupt flag 0-no interrupt pending,1-interrupt pending

3.3.5.3. Timer_A Register (TAR)

TARx Bits Timer_A register. The TAR register is the count of Timer_A.
15-0

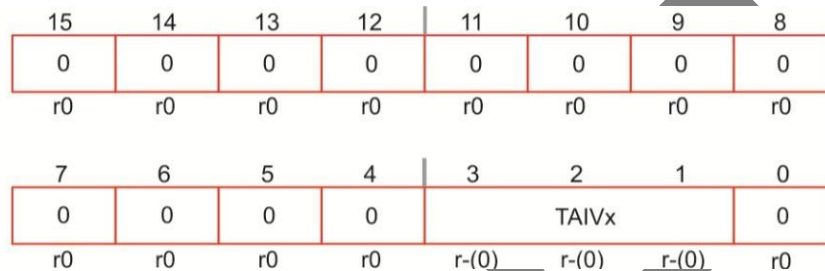
3.3.5.4. Capture/Compare Control Register (TACCTLx)



Name	Bits	Description
CMx	15-14	Capture mode 00-No capture, 01-Capture on rising edge, 10-Capture on falling edge 11-Capture on both rising and falling edges
CCISx	13-12	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific datasheet for specific signal connections. 00-CCIxA, 01-CCIx B, 10-GND and 11-VCC
SCS	11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0-Asynchronous capture, 1-Synchronous capture
SCCI	10	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.
Unused	9	
CAP	8	Capture mode 0-Compare mode, 1-Capture mode
OUTMODx	7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0 because 7-5 EQUx = EQU0. 000-OUT bit value, 001-Set, 010-Toggle/reset, 011-Set/reset, 100-Toggle, 101-Reset, 110-Toggle/set and 111- Reset/set
CCIE	4	Capture/compare interrupt enable. This bit enables the Interrupt request of the corresponding CCIFG flag. 0-Interrupt disabled, 1-Interrupt enabled
CCI	3	Capture/compare input. Selected input signal can be read by this bit.

OUT	2	Output. In output mode0, this bit controls the state of the output. 0-Output low, 1-Output high
COV	1	Capture overflow. Indicates a capture overflow occurred. Reset by sw. 0-No capture overflow occurred, 1-Capture overflow occurred
CCIFG	0	Capture/compare interrupt flag 0-No interrupt pending, 1-Interrupt pending

3.3.5.5. Timer_A Interrupt Vector Register (TAIV)



3.3.5.6. TAIVx Timer_A Interrupt Vector value

TAIV Contents	Interrupt source	Interrupt flag	Interrupt priority
00h	No interrupt pending	-	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2	TACCR2 CCIFG	
06h	Reserved	-	
08h	Reserved	-	
0Ah	Timer Overflow	TAIFG	
0Ch	Reserved	-	
0Eh	Reserved	-	Lowest

Example Application: Timers

A pedometer is a device that counts the number of steps taken by a person and thus provides the total distance covered. It is becoming hugely popular in the fitness equipment industry. TI's MSP430x5xxx device microcontrollers are used in designing pedometers as shown in fig below.

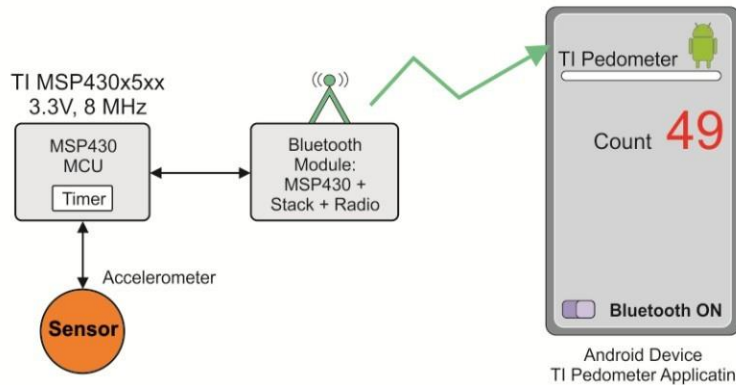


Fig3.7. Pedometer block diagram

The pedometer takes in sensor data from the 3-axis accelerometer for detecting stepping motions in all axes. The input sensor value is sampled at 20 ms or 50Hz. To make sure every sample is captured at 20 ms, the microcontroller's timer module is used. Timer interrupts are generated in microcontrollers invoke an interrupt service routine to capture sensor data for every 20 ms. The interrupt service routine role is to read and process the sensor values. The processed data is sent and displayed on a mobile application using Bluetooth module.

A flowchart illustrating the complete application of timer is shown below:

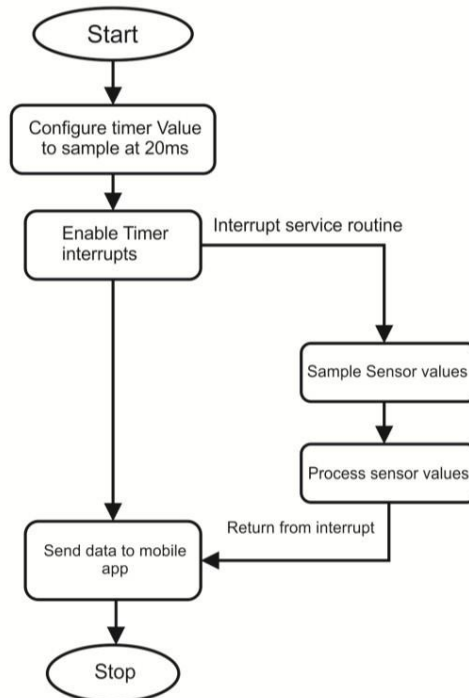


Fig3.8. Flowchart of timer application in creating an ISR

3.4. Real Time Clock (RTC)

Real time clocks (RTC) are used in a variety of applications from watches and clocks to time stamping events and generating events. The general implementation of the RTC is simple. It consists of a timer/counter giving 1-second interrupts and a small CPU routine to count the interrupts. The CPU can sleep or perform other functions between interrupts. The clock setup for the RTC implementation uses the LFXT1 oscillator in LF mode with a 32768-Hz crystal. The output of the LFXT1 oscillator is selected to source ACLK. ACLK is then selected as the clock source for the timer/counter that serves as the time base for the RTC.

The Real-Time Clock (RTC) module can be used as a general-purpose 32-bit counter or as an RTC with calendar function with selectable clock sources. It counts for seconds, minutes, hours, day of week, day of month, month and year in calendar mode. It has Interrupt capability. Data is selectable in BCD format.

3.4.1. RTC block diagram

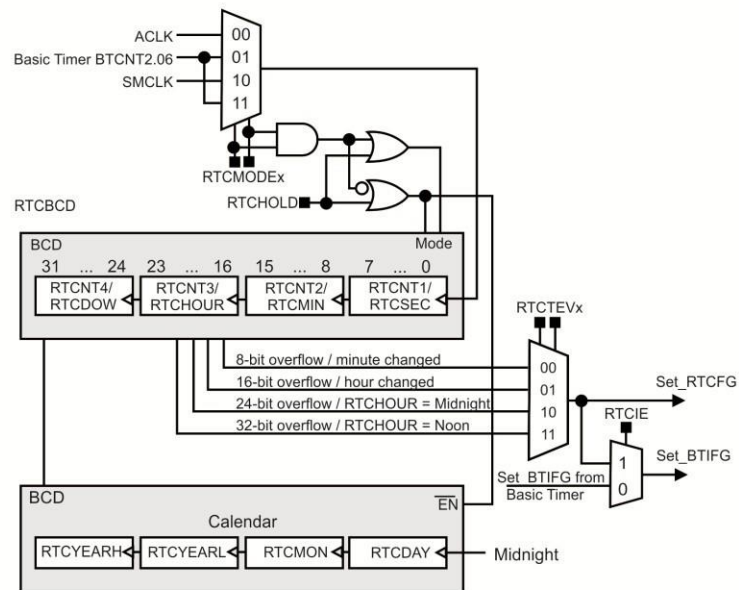


Figure 3.7 Block Diagram of RTC

3.4.2. Counter Mode

Four individual 8-bit counters are cascaded to provide the 32-bit counter. This provides interrupt triggers at 8-bit, 16-bit, 24-bit, and 32-bit overflows. Each counter register RTCNT1 to RTCNT4 is individually accessible and may be read or written.

The clock to increment the counter can be sourced from ACLK, SMCLK, or from the BTCNT2 input clock divided by 128 from the Basic Timer1 module, selected by the RTCMODEx bits. The counter can be stopped by setting the RTCHOLD bit. Switching from calendar to counter mode resets the count value.

3.4.3. Calendar Mode

Calendar mode is selected when RTCMODEx = 11. In this mode, the RTC provides seconds, minutes, hours, day of week, day of month, month, and year in selectable BCD or hexadecimal format.

Switching from counter to calendar mode clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1.

When RTCBCD = 1, BCD format is selected for the calendar registers. The format must be selected before the time is set. Changing the state of RTCBCD clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. The calendar includes a leap year algorithm that considers all years evenly divisible by 4 as leap years. This algorithm is accurate from the year 1901 through 2099.

Note: Any write to any counting register takes effect immediately. However the clock is stopped during the write. This could result in losing up to one second during a write.

RTC Mode Interrupt Interval	RTCDEVx	Interrupt Interval
Counter Mode	00	8-bit overflow
	01	16-bit overflow
	10	24-bit overflow
	11	32-bit overflow
Calendar Mode	00	Minute changed
	01	Hour changed
	10	Every day at midnight (00:00)
	11	Every day at noon (12:00)

3.4.4. Real-Time Clock Interrupts

The Real-Time Clock uses two bits for interrupt control. These include:

- Basic Timer1 interrupt flag, BTIFG, in IFG2 register and
- Real-Time Clock interrupt enable, RTCIE.

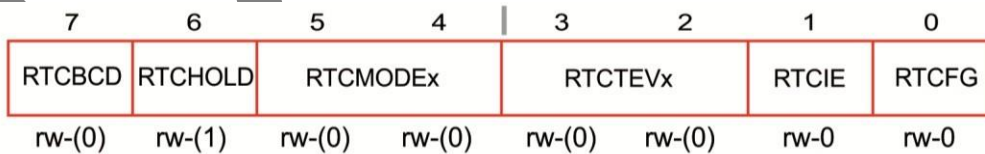
The Real-Time Clock module shares the Basic Timer1 interrupt flag and vector. When RTCIE = 0, the Basic Timer1 controls interrupt generation, otherwise RTC controls the interrupt generation. The RTCFG and BTIFG are used for configuring the interrupt for RTC and Basic timer respectively.

3.4.5. RTC Registers:

Register	Short Form	Register Type	Address	Initial State
Real-Time Clock control register	RTCCTL	Read/write	041h	040h with POR
Real-Time Clock second Real-Time Counter register 1	RTCSEC/ RTCNT1	Read/write	042h	None, not reset
Real-Time Clock minute Real-Time Counter register 2	RTCMIN/ RTCNT2	Read/write	043h	None, not reset
Real-Time Clock hour Real-Time Counter register 3	RTCHOUR/ RTCNT3	Read/write	044h	None, not reset
Real-Time Clock day-of Week Real-Time Counter register 4	RTCDOW/ RTCNT4	Read/write	045h	None, not reset
Real-Time Clock day-of-month	RTCDAV	Read/write	04Ch	None, not

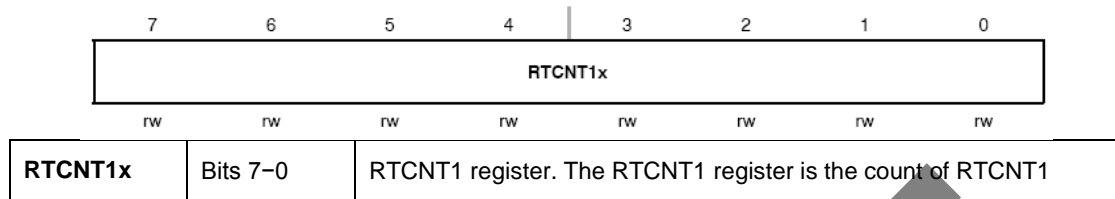
				reset
Real-Time Clock month	RTCMON	Read/write	04Dh	None, not reset
Real-Time Clock year (low byte)	RTCYEARL	Read/write	04Eh	None, not reset
Real-Time Clock year (high byte)	RTCYEARH	Read/write	04Fh	None, not reset
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC

Word Register	Short Form	High-Byte Register	Low-Byte Register	Address
Real-Time control Register	RTCTL	RTCCTL	BTCTL	040h
Real-Time Clocktime 0 Real-Time Counter registers 1,2	RTCTIM0/ RTCNT12	RTCMIN/RTCNT2	RTCSEC/RTCNT1	042h
Real-Time Clocktime 1 Real-Time Counter registers 3,4	RTCTIM1/ RTCNT34	RTCDOW/RTCNT4	RTCHOUR/RTCNT3	044h
Real-Time Clock date	RTCDATE	RTCMON	RTCDAY	04Ch
Real-Time Clock year	RTCYEAR	RTCYEARH	RTCYEARL	04Eh

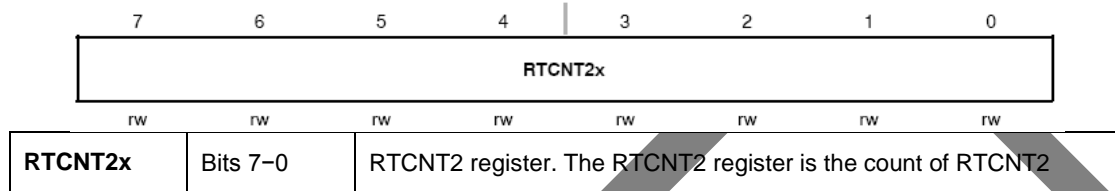
RTCCTL, Real-Time Clock Control Register


Name	Bits	Description																					
RTCBCD	7	BCD format select. This bit selects BCD format for the calendar registers when RTCMODEx = 11. 0 -Hexadecimal format,1 -BCD format																					
RTCHOLD	6	Real-Time Clock hold 0 -Real-Time Clock is operational 1 -RTCMODEx < 11: The RTC module is stopped RTCMODEx = 11: The RTC and the Basic Timer1 are stopped																					
RTCMODEx	5-4	Real-Time Clock mode and clock source select <table border="1"> <thead> <tr> <th>RTCMODEx</th> <th>Counter Mode</th> <th>Clock Source</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>32-bit counter</td> <td>ACLK</td> </tr> <tr> <td>1</td> <td>32-bit counter</td> <td>BTCNT2.Q6</td> </tr> <tr> <td>10</td> <td>32-bit counter</td> <td>SMCLK</td> </tr> <tr> <td>11</td> <td>Calendar mode</td> <td>BTCNT2.Q6</td> </tr> </tbody> </table>	RTCMODEx	Counter Mode	Clock Source	0	32-bit counter	ACLK	1	32-bit counter	BTCNT2.Q6	10	32-bit counter	SMCLK	11	Calendar mode	BTCNT2.Q6						
RTCMODEx	Counter Mode	Clock Source																					
0	32-bit counter	ACLK																					
1	32-bit counter	BTCNT2.Q6																					
10	32-bit counter	SMCLK																					
11	Calendar mode	BTCNT2.Q6																					
RTCDEVX	3-2	Real-Time Clock interrupt event. These bits select the event for Setting RTCFG. <table border="1"> <thead> <tr> <th>RTCMODEx</th> <th>RTCDEVx</th> <th>Interrupt Interval</th> </tr> </thead> <tbody> <tr> <td rowspan="4">Counter Mode</td> <td>00</td> <td>8-bit overflow</td> </tr> <tr> <td>01</td> <td>16-bit overflow</td> </tr> <tr> <td>10</td> <td>24-bit overflow</td> </tr> <tr> <td>11</td> <td>32-bit overflow</td> </tr> <tr> <td rowspan="4">Calendar Mode</td> <td>00</td> <td>Minute changed</td> </tr> <tr> <td>01</td> <td>Hour changed</td> </tr> <tr> <td>10</td> <td>Every day at midnight (00:00)</td> </tr> <tr> <td>11</td> <td>Every day at noon (12:00)</td> </tr> </tbody> </table>	RTCMODEx	RTCDEVx	Interrupt Interval	Counter Mode	00	8-bit overflow	01	16-bit overflow	10	24-bit overflow	11	32-bit overflow	Calendar Mode	00	Minute changed	01	Hour changed	10	Every day at midnight (00:00)	11	Every day at noon (12:00)
RTCMODEx	RTCDEVx	Interrupt Interval																					
Counter Mode	00	8-bit overflow																					
	01	16-bit overflow																					
	10	24-bit overflow																					
	11	32-bit overflow																					
Calendar Mode	00	Minute changed																					
	01	Hour changed																					
	10	Every day at midnight (00:00)																					
	11	Every day at noon (12:00)																					
RTCIE	1	Real-Time Clock interrupt enable 0 -Interrupt not enabled 1 -Interrupt enabled																					
RTCIFG	0	Real-Time Clock interrupt flag 0 -No time event occurred 1 -Time event occurred																					

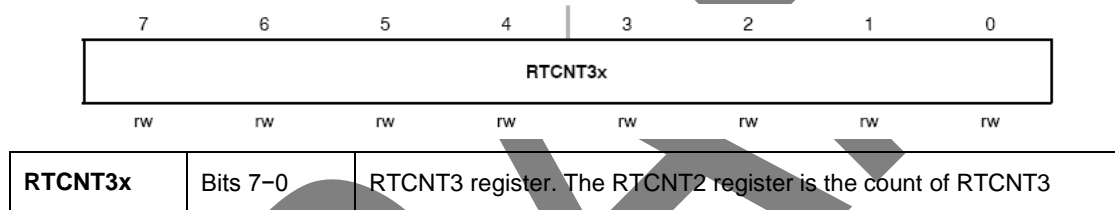
RTCNT1, RTC Counter 1, Counter Mode



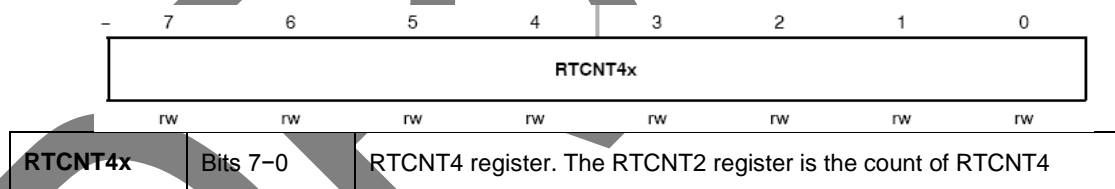
RTCNT2, RTC Counter 2, Counter Mode



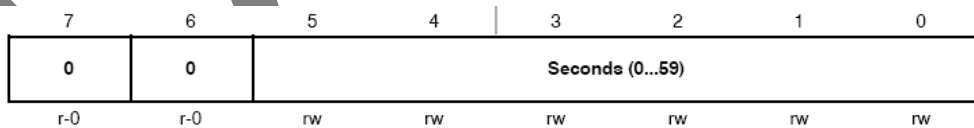
RTCNT3, RTC Counter 3, Counter Mode



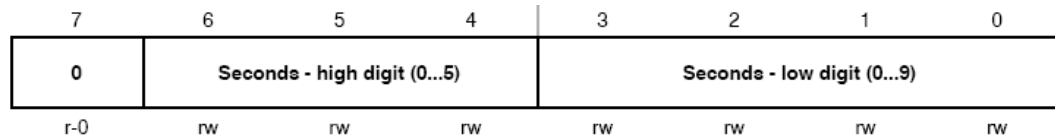
RTCNT4, RTC Counter 4, Counter Mode



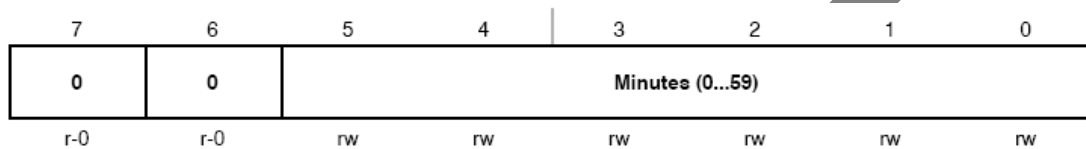
RTCSEC, RTC Seconds Register, Calendar Mode with Hexadecimal Format



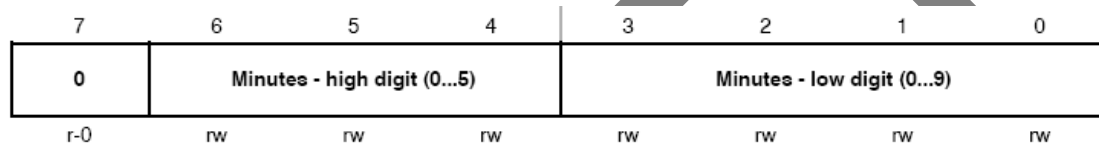
RTCSEC, RTC Seconds Register, Calendar Mode with BCD Format



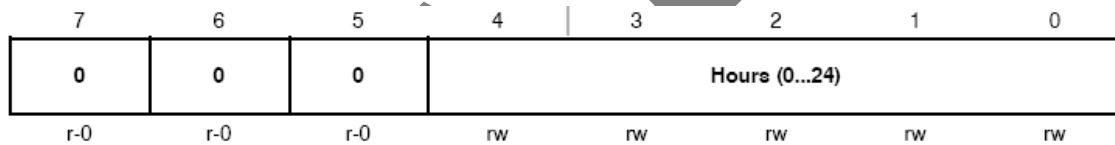
RTCMIN, RTC Minutes Register, Calendar Mode with Hexadecimal Format



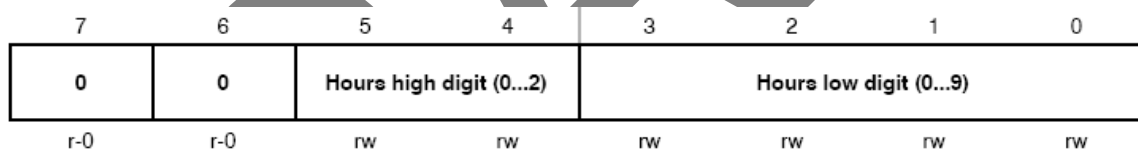
RTCMIN, RTC Minutes Register, Calendar Mode with BCD Format



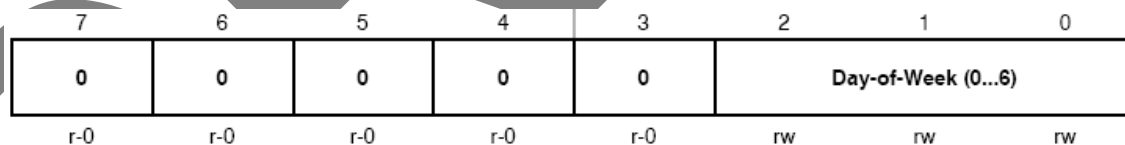
RTCHOUR, RTC Hours Register, Calendar Mode with Hexadecimal Format



RTCHOUR, RTC Hours Register, Calendar Mode with BCD Format



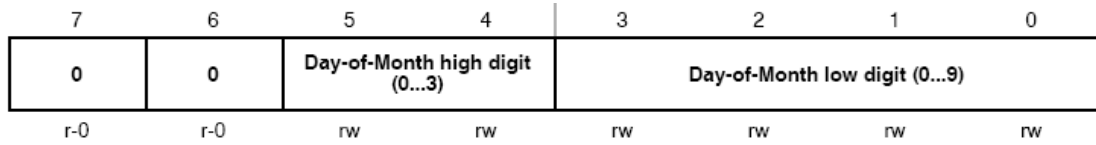
RTCDOW, RTC Day-of-Week Register, Calendar Mode



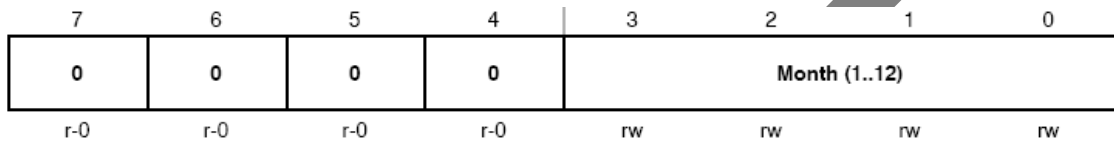
RTCDAY, RTC Day-of-Month Register, Calendar Mode with Hexadecimal Format



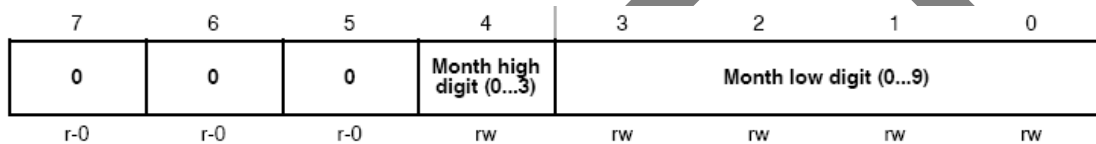
RTCDAY, RTC Day-of-Month Register, Calendar Mode with BCD Format



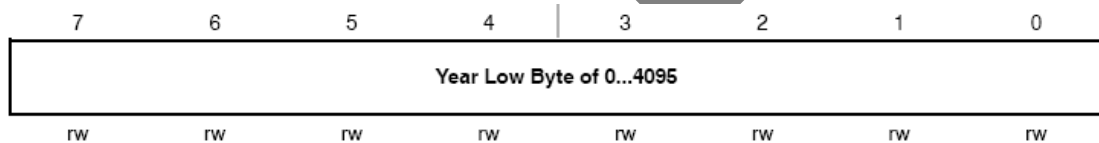
RTCMON, RTC Month Register, Calendar Mode with Hexadecimal Format



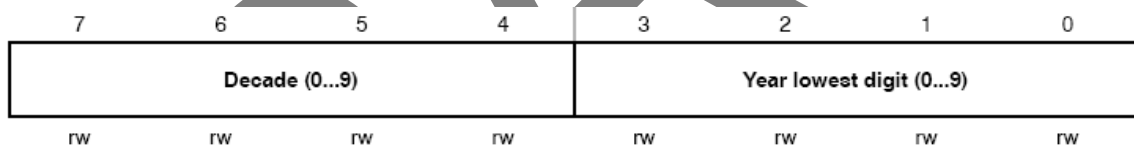
RTCMON, RTC Month Register, Calendar Mode with BCD Format



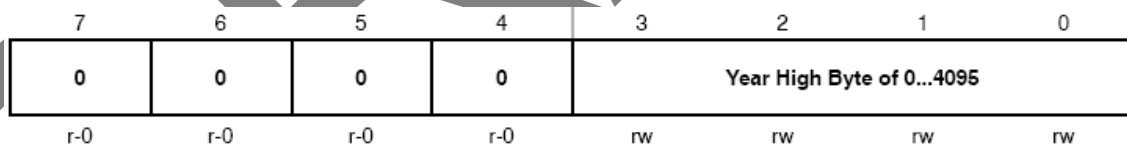
RTCYEARL, RTC Year Low-Byte Register, Calendar Mode with Hexadecimal Format



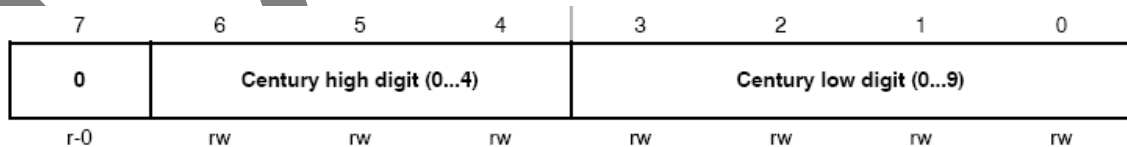
RTCYEARL, RTC Year Low-Byte Register, Calendar Mode with BCD Format



RTCYEARH, RTC Year High-Byte Register, Calendar Mode with Hexadecimal Format



RTCYEARH, RTC Year High-Byte Register, Calendar Mode with BCD Format



NAME	Bits	Description
BTIE	7	Basic Timer1 interrupt enable. This bit enables the BTIFG interrupt. Because other bits in IE2 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0 -Interrupt not enabled 1 -Interrupt enabled
-	6-1	These bits may be used by other modules. See device-specific data sheet

IE2, Interrupt Enable Register 2

7	6	5	4	3	2	1	0
BTIE							

rw-0

NAME	Bits	Description
BTIFG	7	Basic Timer1 interrupt flag. Because other bits in IFG2 may be used for other modules, it is recommended to clear BTIFG automatically by servicing the interrupt, or by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0-No interrupt pending 1 -Interrupt pending
-	6-1	These bits may be used by other modules. See device-specific data sheet

3.5. Pulse Width Modulation

Pulse Width Modulation, or **PWM**, is a technique for getting analog output by a digital means without DAC hardware. It is a method by which digital circuit can output analog values by switching the digital between high and low voltage signals. Assigning proper timing intervals for ON and OFF produces equivalent DC output voltage to the desired analog value. The fraction of the period in which the signal is high is known as the duty cycle.

$$\text{Duty cycle} = \text{ON time} / \text{Total time period}$$

The analog output of the PWM circuit is proportional to the product of duty cycle and DC level for ON of the digital wave.

$$V_{\text{output (PWM)}} = \text{Duty cycle} \times \text{ON Voltage level}$$

The PWM technique is used in communication and predominantly on the analog by digital control side. Let us understand how the microcontroller generates PWM signal. The PWM circuit requires the counter to count the time ticks and two registers to manipulate the duty cycle and total time period. If the PWM output starts with high, the counter counts the ticks by every clock and the counter value is compared with both period register and duty cycle register.

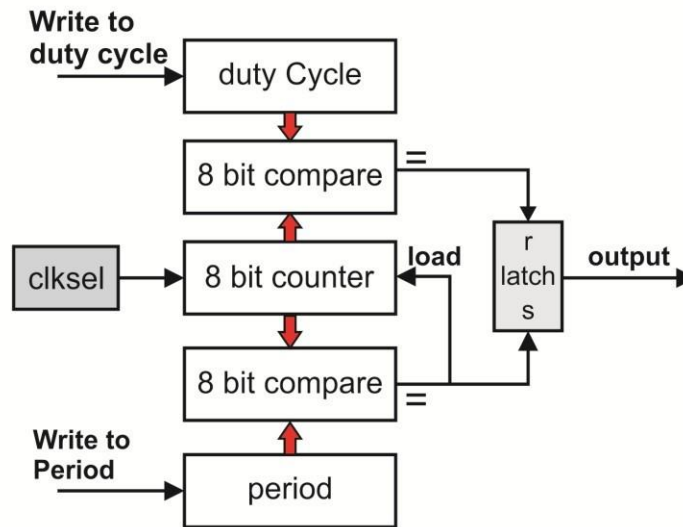


Figure 3.8 Block Diagram of PWM Circuit

Since the duty cycle value is less than the period value, the duty cycle value is matched by compare and output is set to low by resetting the output latch. Counter continues to count the time tick, when it reaches the period value to which the output latch is set, the counter is reset to beginning value. The process is continuing for generating PWM. The change in value of the duty cycle register is altered in the PWM wave

3.5.1. PWM generation in MSP430:

There are three to five capture/compare blocks inside Timer_A of MSP430 controller. Any one of the blocks may be used to capture timer data or generate intervals. The compare mode is used to generate PWM output signals. Each capture/compare block has an output unit, which is used to generate output PWM signals. In Timer_A peripheral circuitry of MSP430X4XX series, there are five compare/capture hardware integrated, the compare mode is enabled for generating PWM.

Timer_A is used here for generating PWM signal. In Timer_A, the TACCR0 is stored for total period value of PWM and TACCR1 is stored for ON period of PWM (Duty cycle). The Timer_A count register (TAR) is in upmode and gets incremented for every clock given by the selected clock source. OUT1 signal of compare module of Timer_A is configured to an output pin and the pin value is started with Logic 1. Compare circuitry compares TAR and TACCR1, when TAR reaches TACCR1 the OUT1 signal is changed to logic 0. The TAR is still counting up and compared with TACCR0. When it matches, then the OUT1 signal goes back to Logic 1 and TAR is reset to zero. The module also sets the compare capture interrupt flag 1 (CCIFG1) upon TAR matching with TACCR1. This flag can be used for a typical timer purpose and is mostly not used in PWM generation. In a similar way, Timer_B can also be configured to PWM generation.

Example 1:

The following program is used for realizing PWM signal by connecting an LED in P1.2 and this LED's intensity can be adjusted by changing ON and Total time. The PWM can be realized by changing Period and duty cycle in TARCC0 and TARCC1 respectively. The output mode can be changed to set/rest mode and output can be realized by changing the TACCR0 and 1.

```
#include msp430g2553.h
void main(void)
{
WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
P1DIR |= BIT2; // Initialize port for Output signal OUT1
P1SEL |= BIT2;
TACCR0 = 0xFF;
TACCR1 = 0x20; // initial duty cycle
TACCTL1 = OUTMOD_7; // Reset/set output mode
// Start timer from ACLK /2, Up mode , clear , no interrupts
TACTL |= TASSEL_1|ID_1|MC_1;
__low_power_mode_3 ();
for(;;);
}
```

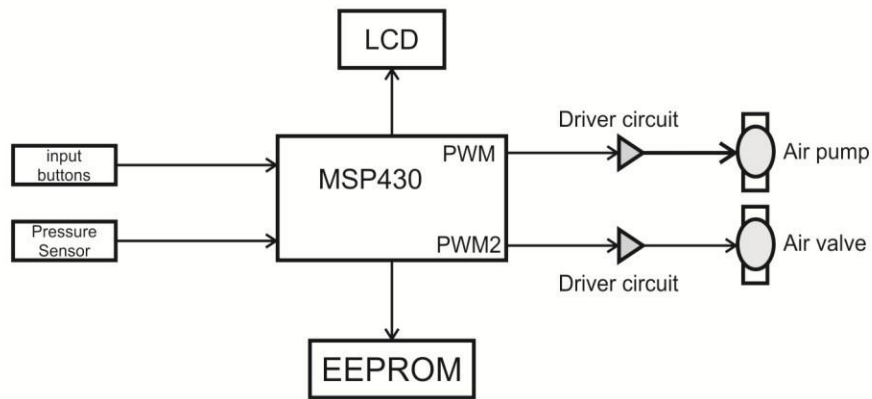
Example 2:

The following program is used for realizing PWM signal by connecting a LED in P1.2 and this LED's intensity is slowly brightened and reaches full intensity. Again the LED comes to OFF and starts brightening continuously. The PWM can be realized by changing Period and duty cycle in TARCC0 and TARCC1 respectively. The output mode can be changed to set/rest mode for creating dimming effect using PWM.

```
#include msp430g2553.h
void main(void)
{
WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
P1DIR |= BIT2; // Initialize port for Output signal OUT1
P1SEL |= BIT2;
TACCTL0 = CCIE;
TACCR0 = 0x00FF;
TACCR1 = 0x0; // initial duty cycle
TACCTL1 = OUTMOD_7; // Reset/set output mode
// Start timer from ACLK /2, Up mode , clear , no interrupts
TACTL |= TASSEL_1|ID_1|MC_1;
while(1);
}

# pragma vector = TIMERA0_VECTOR
__interrupt void TIMERA0_ISR ( void )
{
delay_ms(50); // delay for 50ms
TACCR1++;
If(TACCR1==0x00FF) TACCR1=0;
}
```

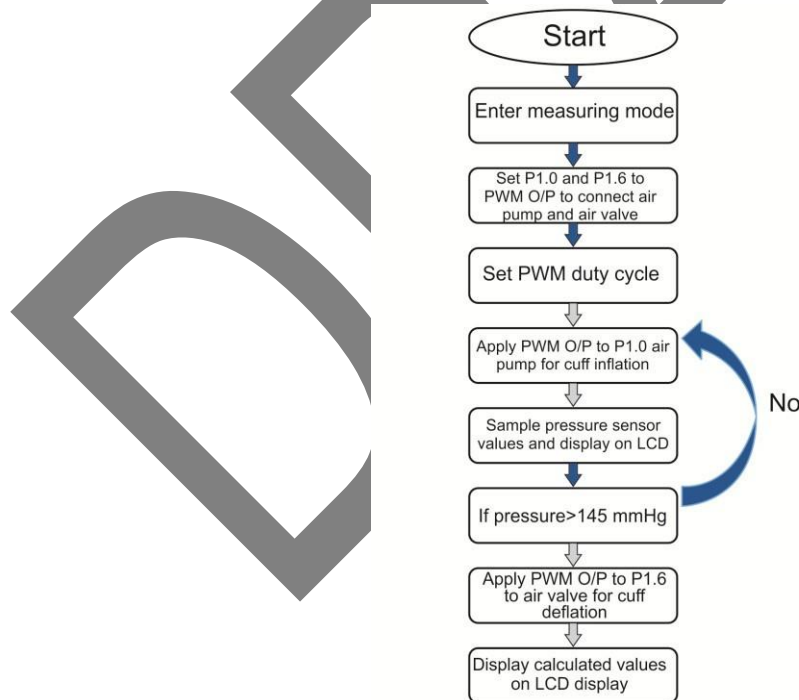
Example Application: PWM:



Block diagram of blood pressure monitor using MSP430

The controller takes input from input buttons and can be used to either put the device in measuring mode or view the stored readings. The air pump is controlled and operated by the PWM module in the controller after the device is set to measuring mode to inflate the arm cuff for measuring systolic pressure. The average voltage applied to the pump can also be varied by only changing the duty cycle to ensure correct readings. The air valve is also controlled by PWM module to deflate the arm cuff automatically to measure the diastolic pressure. The pressure sensor senses the pressure of the cuff and displays it on the LCD. The final readings are displayed on the LCD and stored in the EEPROM.

A flowchart is illustrated to understand the application of PWM in blood pressure monitor.



Flowchart for setting up PWM values

3.6. Timing generation and measurement

To generate a timing event, MSP430 has up to five types of timers. It is important to understand which timer module in MSP430 to prefer for a particular application. The next important criterion is choosing clock source for the timer modules. The following guidelines can be used for choosing the clock source and timer module. As we discussed already, that there are three clock sources - ACLK, MCLK and SMCLK from various clock generators such as LFXT1 oscillator, XT2 oscillator, and DCO in MSP430. An external crystal device is required for LFXT1 and XT2; the programmer should refer to the MSP403 target board for identifying the crystal frequency. The programmer must understand and find the time tick duration for generating a minimum event triggering timing. Accordingly, the program is made for configuring the clock source frequency to the timer module input.

3.6.1. Basic Timer:

It takes clock source from ACLK and SMCLK and supplies LCD timing and low frequency time intervals. It can be used for regular interval generation and the timer value can be adjusted at any time by stopping the timer. The output is controlled in this timer event is done in its interrupt service routine (ISR).

3.6.2. Timer_A and B

Timer_A and B modules take clock source from ACLK, SMCLK and TACLK (an external direct crystal clock source). They can be used for regular interval generation and the timer value can be adjusted at any time by stopping the timer. Timer_A and B output can be connected directly to the Port Pin so that it can be driven directly by timer hardware. Also the output can be controlled in ISR. These timers are in compare/capture provisions and they can be used for generating PWM. The Timers A and B can be connected directly to an input to trigger capture and compare while timer is running and it is used for sampling mode. Slow inputs can directly be sent to a Capture input of Timer_A or B where as the fast signals should be connected to one of the timer clock inputs, such as TACLK or INCLK. Using the internal connections to other peripherals wherever possible, both for capture and compare events are carried out. This provides proper timing for the peripheral to start or stop without CPU intervention.

3.6.3. Watchdog timer

If this timer is not used as a watchdog and if the timer interval is appropriate, it produces four discrete intervals for a given clock frequency. They are roughly 2, 16, 250, and 1000 ms from ACLK at 32 KHz, slower if VLO is used instead. Shorter intervals can be obtained by using SMCLK instead of ACLK.

3.6.4. Software

When all the timers in the MSP430 device are busy, then software loop is used. It is best to avoid these whenever possible, except for trivial cases such as delays while a clock stabilizes.

```
#include <msp430g2253.h>
int main(void) {
    /** Watchdog timer and clock Set-Up ***/

    WDTCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
    DCOCTL = 0;                  // Select lowest DCOx and MODx
    BCCTL1 = CALBC1_1MHZ;       // Set range
    DCOCTL = CALDCO_1MHZ;       // Set DCO step + modulation
```

```

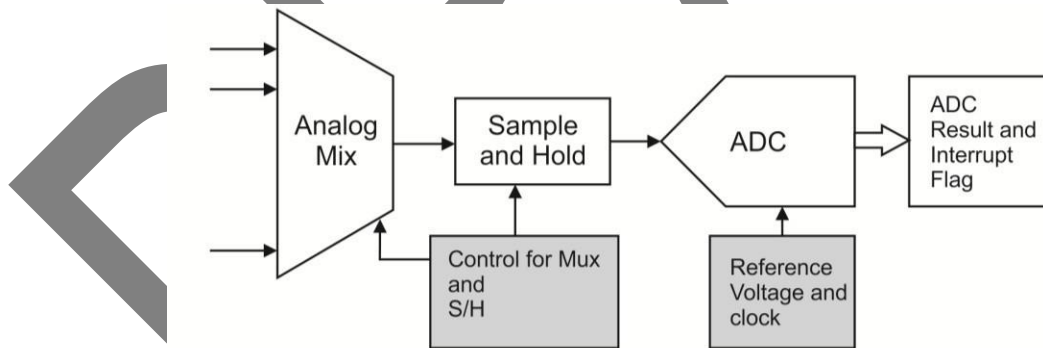
/** GPIO Set-Up */
P1DIR |= BIT2;           // P1.2 set as output
P1SEL |= BIT2;           // P1.2 selected Timer_A Out1
P2DIR |= BIT1;           // P2.1 set as output
P2SEL |= BIT1;           // P2.1 selected Timer_A Out1
P2DIR |= BIT4;           // P2.4 set as output
P2SEL |= BIT4;           // P2.4 selected Timer_A Out2

/** Timer_A Set-Up */
TACCR0 |= 200 - 1;       // PWM Period
TACCTL1 |= OUTMOD_7;     // TACCR1 output mode = reset/set
TACCR1 |= 100;           // TACCR1 PWM duty cycle
TACTL |= TASSEL_2 + MC_1; // SMCLK, Up Mode (Counts to TACCR0)
        _BIS_SR(LPM0_bits); // Enter Low power mode 0
    }
    
```

3.7. Analog interfacing and acquisition

The letters MSP stand for *mixed signal processor*, which means that many practical applications require analog inputs and analog output. There is a section of analog-to-digital converters with a resolution of up to 10, 12bits and digital to analog converter. An analog-to-digital converter (ADC, or A/D) is an electronic circuit that converts an analog input voltage into its digital representation. The output of the ADC is an integer that is proportional to the magnitude of the sampled analog voltage and relative to an internal or external reference voltage.

$$\text{Digital output} = (\text{Analog input voltage} / \text{Reference voltage}) \times 2^{\text{(ADC bit size-1)}}$$



3.9 Analog to digital Conversion Mechanism

The analog input circuitry in a microcontroller starts with analog multiplexer to support a multiple of analog input and any one of the input is allowed to further block to process the analog signal. The sample and hold circuitry holds the analog voltage and hold for ADC to complete its conversion. The selection of the channel in the multiplexer and controlling sample and hold circuit are done by the control registers. The output of the sample and hold circuit is converted into a corresponding digital output and is updated to ADC result memory as well as setting the flag. The CPU in which the ADC is integrated controls the ADC modules and recognizes the conversion complete flag by programming.

3.7.1. ADC

Many of the practical real time signals around us are analog in nature. In order to feed this signal quantity into the digital world (such as to a microcontroller), it needs to be converted into digital by means of a special hardware called Analog to Digital Converter (ADC). The ADC gets the signal from an analog source like pressure, temperature, lights intensity and then converts it into a proportional equivalent electric signal. That means a physical parameter is converted into an electrical signal. MSP430 microcontroller has two separate ADCs such as 10 bit and 12 bit namely ADC10 and ADC12 respectively.

3.7.2. ADC 10

The ADC10 module is a high-performance 10-bit analog-to-digital converter. This chapter describes the ADC10.

3.7.2.1. ADC10 features:

It is an 8-channel, 10-bit SAR converter with conversion rates of more than 200 ksp/s and in built sample-and-hold hardware where the sample period is programmable. Conversion is initiated by software or by Timer_A. It has a software selectable on-chip reference voltage generation (1.5 V or 2.5 V), which can be external or internal. It operates in single-channel, repeated single-channel, sequence, and repeated sequence conversion modes. Most importantly, there is a data transfer controller for automatic storage of ADC conversion results.

3.7.2.2. ADC10 Block Diagram and function

The ADC core converts an analog input to its 10-bit digital representation and stores the result in the ADC10MEM register. The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The input channel and the reference voltage levels (V_{R+} and V_{R-}) are defined in the conversion-control memory. Conversion results may be in straight binary format or 2s-complement format.

The conversion formula for the ADC result when using straight binary format is:

$$N_{ADC} = 1023 * \frac{V_n - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC10 core is configured by two control registers, ADC10CTL0 and ADC10CTL1. The core is enabled with the ADC10ON bit. With a few exceptions, the ADC10 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place.

3.7.2.3. ADC Clock Selection

The clock for ADC10 is used both as the conversion clock and to generate the sampling period. The ADC10 source clock is selected using the ADC10SSELx bits and can be divided from 1-8 using the ADC10DIVx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK and an internal oscillator ADC10OSC. The ADC10OSC is generated internally.

3.7.2.4. Analog input and Voltage Reference

Eight external and four internal analog signals are selected as the Channel for conversion by the analog input multiplexer. The ADC10 external inputs A0 - A4, V_{eREF+} and V_{REF} share the terminals with I/O port P2. Optional inputs A5 to A7 are shared on port P3 on selected devices. The ADC10AEx bits provide the ability to disable the port pin input and output buffers. As an example, the following instruction configures P2.3 as an analog input.

BIS.B #08h,&ADC10AE ; P2.3 ADC10 function and enable

The ADC10 module contains a built-in voltage reference with two selectable voltage levels. The Internal voltage reference levels are 1.5V and 2.5V. External references may be supplied for VR+ and VR- through pins A4 and A3 respectively.

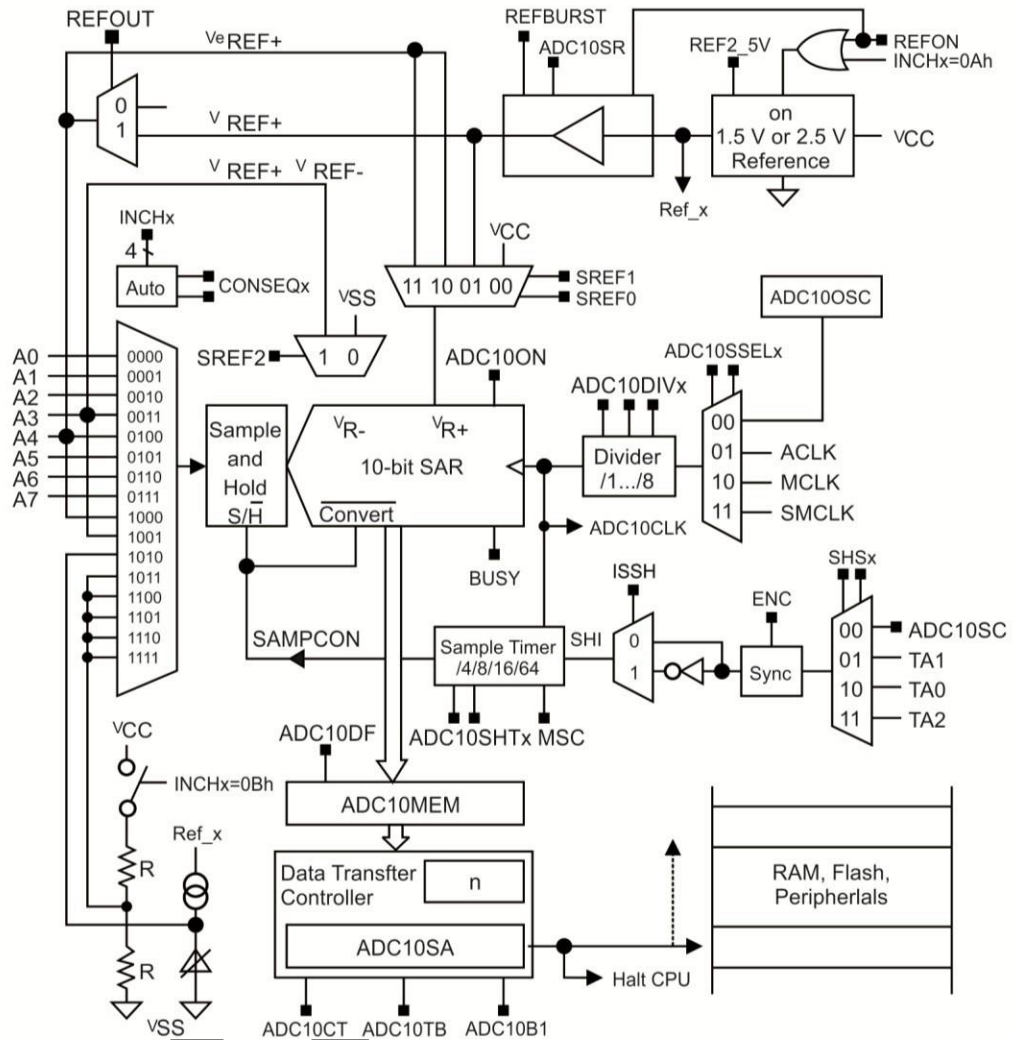


Figure 3.10 ADC10 Block Diagram

3.7.2.5. Auto Power-Down

The ADC10 is designed for low power applications. When the ADC10 is not actively converting, the core is automatically disabled and re-enabled when needed. The ADC10OSC is also automatically enabled when needed and disabled when not needed. When the core or oscillator are disabled, they consume no current.

3.7.2.6. Sample and Conversion Timing

Conversion of ADC is initiated with a rising edge of sample input Signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- The ADC10SC bit

- The Timer_A Output Unit 1
- The Timer_A Output Unit 0
- The Timer_A Output Unit 2

The polarity of the SHI signal source can be inverted with the ISSH bit. The SHTx bits select the sample period t_{sample} to be 4, 8, 16, or 64 ADC10CLK cycles. The sampling timer sets SAMPCON high for the selected sample period after synchronization with ADC10CLK. Total sampling time is t_{sample} plus t_{sync} . The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC10CLK cycles as shown in Figure below.

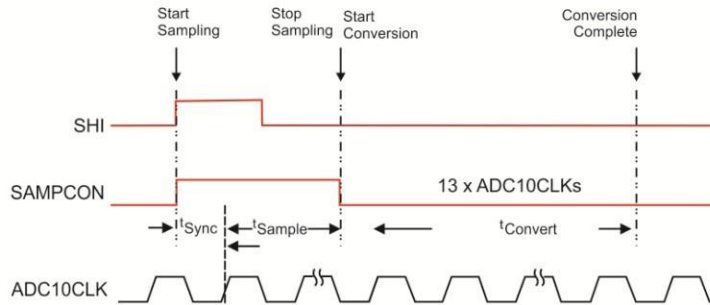


Figure 3.11 ADC10 Sample Timing

Sample Timing calculation is discussed in the datasheet of the corresponding series of MSP430.

3.7.2.7. Conversion Modes

The ADC10 has four operating modes selected by the CONSEQx bits as discussed in Table below.

CONSEQx	Mode	Operation
00	Single channel single-conversion	A single channel is converted once.
01	Sequence of channels	A sequence of channels is converted once.
10	Repeat single channel	A single channel is converted repeatedly.
11	Repeat sequence of channels	A sequence of channels is converted repeatedly.

Single-Channel Single-Conversion Mode

A single channel selected by INCHx is sampled and converted once. The ADC result is written to ADC10MEM register. When ADC10SC triggers a conversion, successive conversions can be triggered by the ADC10SC bit. When any other trigger source is used, Enable conversion bit ENC must be toggled between each conversion. The flow chart for the Single-Channel Single-Conversion Mode is explained in the datasheet of the MSP430 series.

Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The sequence begins with the channel selected by INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM register. The sequence stops after conversion of channel A0. When ADC10SC triggers a sequence, successive sequences can be triggered by the ADC10SC bit. When any other trigger source is used, ENC must be toggled between each sequence. The flow chart for the Sequence-of-Channels Mode explained in the datasheet of the MSP430 series.

Repeat-Single-Channel Mode

A single channel selected by INCHx register is sampled and converted continuously. Each ADC result is written to ADC10MEM register.

Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The sequence begins with the channel selected by INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM register. The sequence ends after conversion of channel A0, and the next trigger signal re-starts the sequence.

ADC 10 can be configured to perform successive conversions automatically as multiple sample and convert function using MSC bit in ADC10CTL register. This is function is valid only for sequence or repeated modes and successive conversions are triggered automatically as soon as the prior conversion is completed. The function of the ENC bit is unchanged when using the MSC bit.

Resetting the ENC bit in any time the conversion is stopped when it is in any mode of operation. Stopping ADC10 activity depends on the mode of operation.

3.7.2.8. ADC10 Data Transfer Controller

The ADC10 includes a data transfer controller (DTC) to automatically transfer conversion results from ADC10MEM to other on-chip memory locations without CPU intervention. When the DTC is enabled, each time the ADC10 completes a conversion and loads the result to ADC10MEM register, a data transfer is triggered and data is transferred to selected location with defined number of transfers.

ADC10 data transfer operation is performed in two modes. They are:

- 1) One-Block Transfer Mode
- 2) Two-Block Transfer Mode

No software intervention is required to manage the ADC10 until the predefined amount of conversion data has been transferred. Each DTC transfer requires one CPU MCLK. To avoid any bus contention during the DTC transfer, the CPU is halted, if active, for the one MCLK required for the transfer. A DTC transfer must not be initiated while the ADC10 is busy. Software must ensure that no active conversion or sequence is in progress when the DTC is configured.

One-Block Transfer Mode

The one-block mode is selected if the ADC10TB is reset. The value n in ADC10DTC1 defines the total number of transfers for a block. The block start address is defined anywhere in the MSP430 address range using the 16-bit register ADC10SA. The block ends at $ADC10SA+2n-2$. The one-block transfer mode is shown in Figure below.

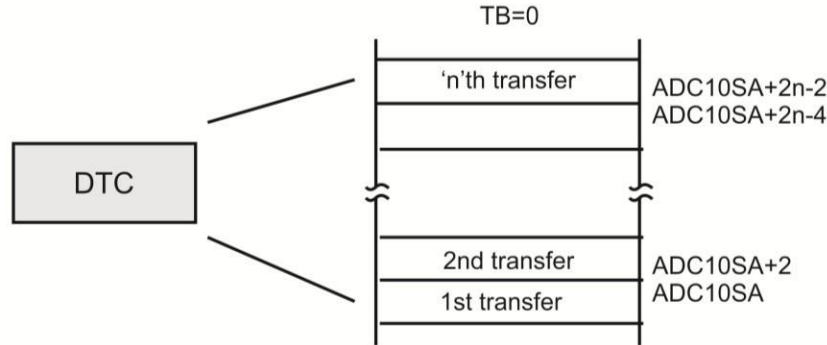


Figure 3.12 One-Block Transfer

The internal address pointer is initially equal to ADC10SA and the internal transfer counter is initially equal to „n“. The internal pointer and counter are not visible to software. The DTC transfers the word-value of ADC10MEM to the address pointer ADC10SA. After each DTC transfer, the internal address pointer is incremented by two and the internal transfer counter is decremented by one. The DTC transfers continue with each loading of ADC10MEM, until the internal transfer counter becomes equal to zero.

Two-Block Transfer Mode

The two-block mode is selected if the ADC10TB bit is set. The value n in ADC10DTC1 defines the number of transfers for one block. The address range of the first block is defined anywhere in the MSP430 address range with the 16-bit register ADC10SA. The first block ends at ADC10SA+2n-2. The address range for the second block is defined as SA+2n to SA+4n-2.

Continuous Transfer

A continuous transfer is selected if ADC10CT bit is set. The DTC will not stop after block one in (one-block mode) or block two (two-block mode) has been transferred. The internal address pointer and transfer counter are set equal to ADC10SA and n respectively. Transfers continue starting in block one

For each ADC10MEM transfer, the DTC requires one or two MCLK clock cycles to synchronize, one for the actual transfer (while the CPU is halted), and one cycle of wait time. Because the DTC uses MCLK, the DTC cycle time is dependent on the MSP430 operating mode and clock system setup. The maximum DTC cycle time for all operating mode is discussed in the MSP430 series data sheet.

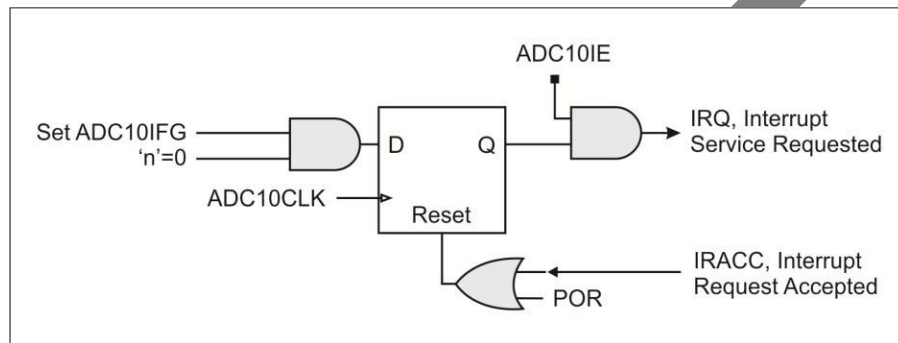
3.7.2.9 ADC10 with Integrated Temperature Sensor

ADC10 module is facilitated with internal temperature sensor. To use this on-chip temperature sensor, the user selects the analog input channel INCHx = 1010. The typical temperature sensor transfer function is $V_{TEMP} = 0.00355(TEMP_C) + 0.986$. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See the device-specific datasheet for the parameters. Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the VREF+ output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

The grounding and noise issues are discussed in the MSP430 series data sheet.

3.7.2.10 ADC10 Interrupts

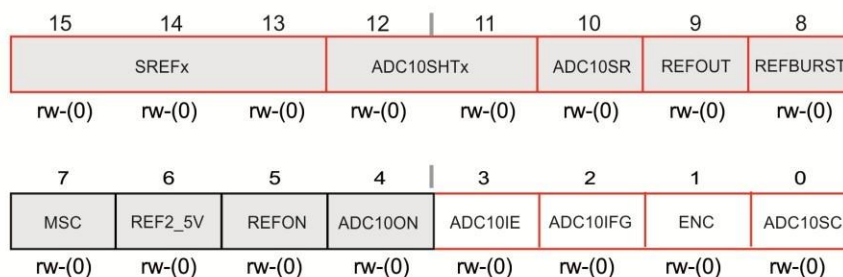
One interrupt and one interrupt vector are associated with the ADC10 as shown in Figure below. When the DTC is not used ($ADC10DTC1 = 0$) $ADC10IFG$ is set when conversion results are loaded into $ADC10MEM$. When DTC is used ($ADC10DTC1 > 0$) $ADC10IFG$ is set when a block transfer completes and the internal transfer counter "n" = 0. If both the $ADC10IE$ and the GIE bits are set, then the $ADC10IFG$ flag generates an interrupt request. The $ADC10IFG$ flag is automatically reset when the interrupt request is serviced or may be reset by software.



3.7.2.11 ADC10 Registers

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

ADC10CTL0, ADC10 Control Register 0

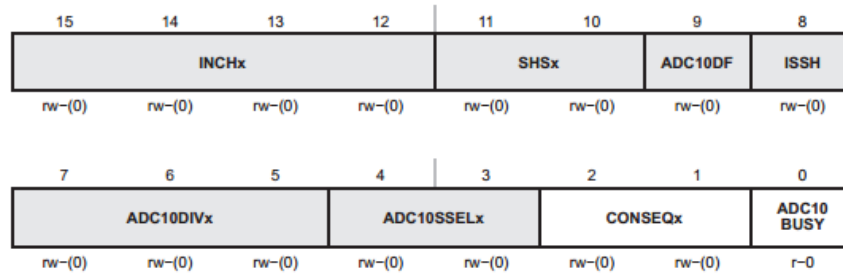


 Modifiable only when ENC=0

Name	Bits	Description
SREFx	15-13	Select reference 0 VR+= VCC and VR-= VSS 1 VR+= VREF+ and VR-= VSS 10 VR+= VeREF+ and VR-= VSS 11 VR+= VeREF+ and VR-= VSS 100 VR+= VCC and VR-= VREF-/ VeREF- 101 VR+= VREF+ and VR-= VREF-/ VeREF- 110 VR+= VeREF+ and VR-= VREF-/ VeREF- 111 VR+= VeREF+ and VR-= VREF-/ VeREF-
ADC10SHTX	12-11	ADC10 sample-and-hold time 00-4 x ADC10CLKs, 01-8 x ADC10CLKs, 10-16 ADC10CLKs, 11-64 x ADC10CLKs.
ADC10SR	10	ADC10 sampling rate. 0-Reference buffer supports up to ~200 ksps 1-Reference buffer supports up to ~50 ksps
REFOUT	9	Reference output 0-Reference output off, 1-Reference output on
REFBURST	8	Reference burst. REFOUT must also be set. 0-Reference buffer on continuously 1-Reference buffer on only during sample-and-conversion
MSC	7	Multiple sample and conversion. Valid only for sequence or repeated modes. 0-The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion. 1-The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed
REF2_5V	6	Reference-generator voltage. REFON must Also be set. 0 - 1.5 V, 1 - 2.5 V
REFON	5	Reference generator on 0-Reference off, 1-Reference on
ADC10ON	4	ADC10 on 0-ADC10 off, 1-ADC10 on
ADC10IE	3	ADC10 interrupt enable 0- Interrupt disabled, 1-interrupt enabled
ADC10IFG	2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a Conversion result. It is automatically reset when the interrupt request is accepted. When using the DTC this flag is set when a block

		of transfers is completed. 0- No interrupt pending,1- Interrupt pending
ENC	1	Enable conversion 0-ADC10 disabled,1-ADC10 enabled
ADC10SC	0	Start conversion. Software-controlled sample-and-conversion Start.ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically. 0-No sample-and-conversion start, 1-Start sample-and-conversion

ADC10CTL1, ADC10 Control Register 1



Modifiable only when ENC = 0

Name	Bits	Description
INCHx	15-12	Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. 0000-A0, 0001-A1, 0010-A2, 0011-A3, 0100-A4, 0101-A5, 0110-A6, 0111-A7, 1000 VeREF+, 1001 VREF-/VeREF-, 1010 Temperature sensor, 1011 (VCC- VSS) / 2, 1100 (VCC- VSS) / 2, 1101 (VCC- VSS) / 2, 1110 (VCC- VSS) / 2, 1111 (VCC- VSS) / 2
SHSx	11-10	Sample-and-hold source select 00-ADC10SC bit, 01-Timer_A.OUT1, 10-Timer_A.OUT, 11-Timer_A.OUT2
ADC10DF	9	ADC10 data format 0-Straight binary,1-2"s complement
ISSH	8	Invert signal sample-and-hold 0 The sample-input signal is not inverted. 1 The sample-input signal is inverted
ADC10DIVx	7-5	ADC10 clock divider 000-/1, 001-/2, 010-/3, 011-/4, 100-/5, 101-/6, 110-/7, 111-/8
ADCSSSELx	4-3	ADC10 clock source select

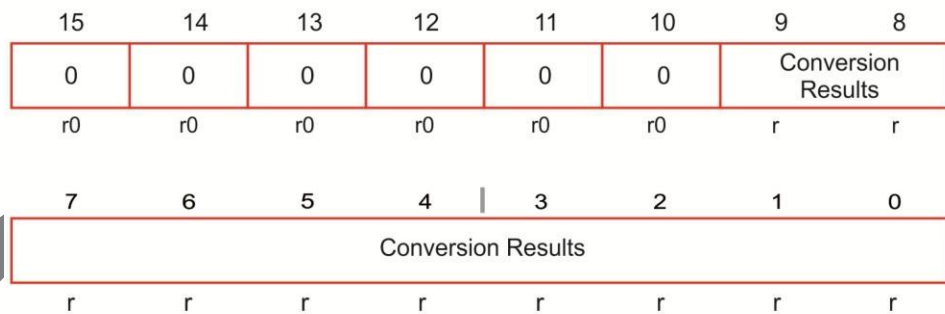
		00-ADC10OSC, 01-ACLK, 10-MCLK, 11-SMCLK
CONSEQx	2-1	Conversion sequence mode select 00-Single-channel-single-conversion 01-Sequence-of-channels 11-Repeat-sequence-of-channels 10-Repeat-single-channel
ADC10	0	ADC10 busy. This bit indicates an active sample or BUSY Conversion Operation.

ADC10AE, Analog (Input) Enable Control Register



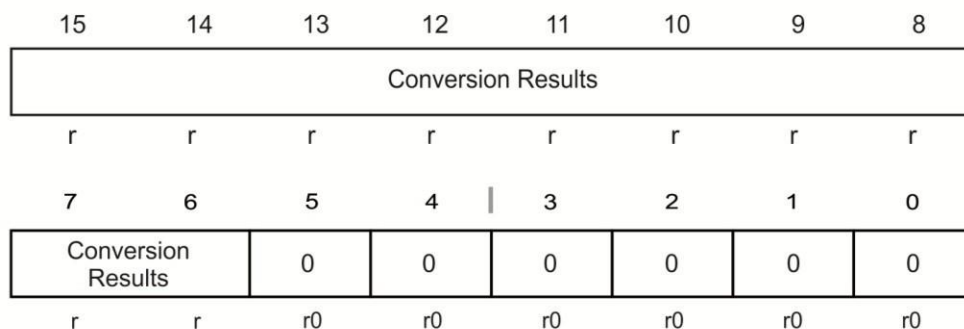
ADC10AEx	Bits 7-0	ADC10 analog enable 0 -Analog input disabled, 1-Analog input enabled
-----------------	-----------------	---

ADC10MEM, Conversion-Memory Register, Binary Format



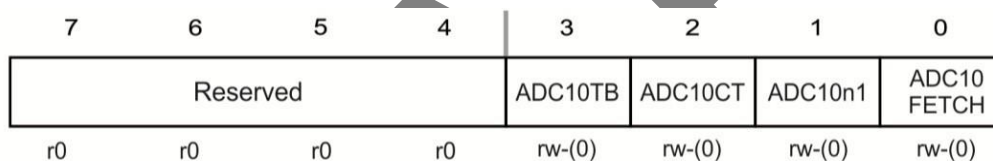
Conversion results	Bits 15-0	The 10-bit conversion results are right justified, Straight-binary format. Bit 9 is the MSB. Bits 15-10 are always 0.
---------------------------	------------------	---

ADC10MEM, Conversion-Memory Register, 2's Complement Format



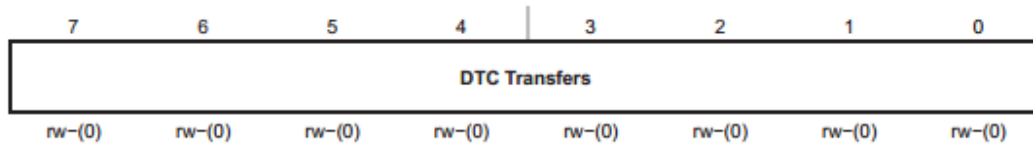
Conversion results	Bits 15-0	The 10-bit conversion results are left-justified, 2's complement format. Bit 15 is the MSB. Bits 5-0 are always 0.
---------------------------	------------------	--

ADC10DTC0, Data Transfer Control Register 0



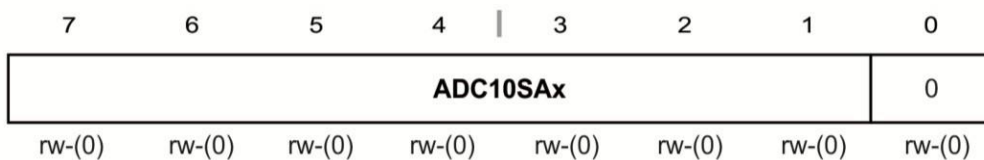
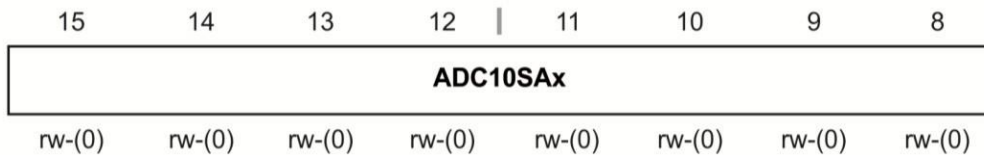
Name	Bits	Description
Reserved	7-4	Reserved. Always read as 0.
ADC10TB	3	ADC10 two-block mode. 0-One-block transfer mode 1-Two-block transfer mode
ADC10CT	2	ADC10 continuous transfer. 0-Data transfer stops when one block (one-block mode) or two blocks (two-block mode) have completed. 1-Data is transferred continuously. DTC operation is stopped only if ADC10CT cleared, or ADC10SA is written to.
ADC10B1	1	ADC10 block one. This bit indicates for two-block mode which block is filled with ADC10 conversion results. ADC10B1 is valid only after ADC10IFG has been set the first time during DTC operation. ADC10TB must also be set 0-Block 2 is filled, 1-Block 1 is filled
ADC10FETCH	0	This bit should normally be reset.

ADC10DTC1, Data Transfer Control Register 1



DTC transfers	7-0	DTC transfers. These bits define the number of transfers in each block. 0 DTC is disabled, 01h-0FFh Number of transfers per block
----------------------	-----	--

ADC10SA, Start Address Register for Data Transfer



ADC10SAx	15-1	ADC10 start address. These bits are the start address for the DTC. A Write to register ADC10SA is required to initiate DTC transfers
Unused	0	Unused, Read only. Always read as 0.

3.7.3. ADC 12

The ADC12 module is a high-performance 12-bit analog-to-digital converter. This chapter describes the ADC12. The ADC12 is implemented in the MSP430x13x, MSP430x14x, MSP430x15x, and MSP430x16x devices. The module supports fast (greater than 250ksps), 12-bit analog-to-digital conversions, SAR core, sample select control, reference generator and a 16 word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention.

3.7.3.1. ADC12 architecture

The ADC12 core converts an analog input to a 12-bit digital representation and stores the result in its conversion memory. The core uses two programmable/selectable voltage levels (VR+ and VR-) to define the upper and lower limits of the conversion.

The conversion formula for the ADC result NADC is:

$$N_{ADC} = 4095 \times \frac{V_i V_R}{V_{R+} - V_{R-}}$$

The ADC12 core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The configuration of the ADC12 in the control register can only be modified when ENC = 0.

3.7.3.2. Conversion Clock Selection

The ADC12CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. ADC12CLK sources are SMCLK, MCLK, ACLK, and an internal oscillator ADC12OSC (5-MHz range). The ADC12 source clock is selected using the ADC12SSELx bits and can be divided using the ADC12DIVx bits. The clock chosen for ADC12CLK remains active until the end of a conversion.

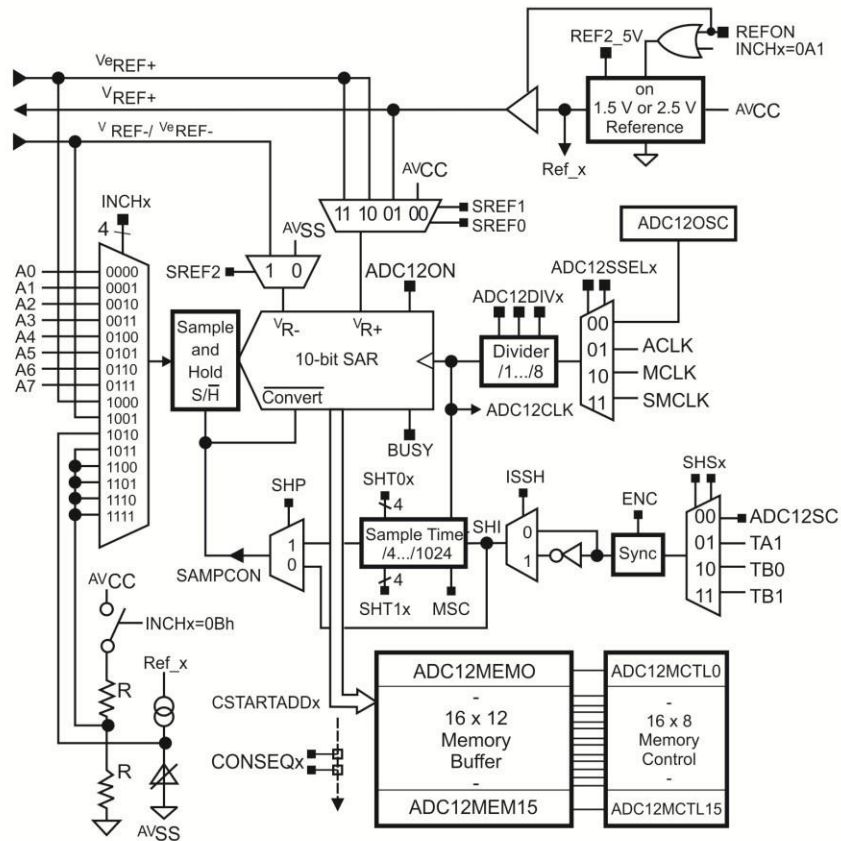


Figure 3.14 ADC12 Block Diagram

3.7.3.3. Analog input and Voltage Reference

The eight external and four internal analog signals are selected as the channel for conversion by the analog input multiplexer. The ADC12 inputs are multiplexed with the port P6 pins, which are digital. The ADC12 module contains a built-in voltage reference with two selectable voltage levels, 1.5 V and 2.5 V. Either of these reference voltages may be used internally and externally on pin VREF+. Setting REFON=1 enables the internal reference. When REF2_5V = 1, the internal reference is 2.5 V, the reference is 1.5 V when REF2_5V = 0. The reference can be turned off to save power when not in use. External references may be supplied for VR+ and VR- through pins VeREF+ and VREF-/VeREF- respectively.

3.7.3.4. Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- 1) The ADC12SC bit
- 2) The Timer_A Output Unit 1
- 3) The Timer_B Output Unit 0
- 4) The Timer_B Output Unit 1

The polarity of the SHI signal source can be inverted with the ISSH bit. The SAMPCON signal controls the sample period and start of conversion. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC12CLK cycles. Two different sample-timing methods are defined by control bit SHP, they are extended sample mode and pulse mode.

Extended Sample Mode

The extended sample mode is selected when SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period t_{sample} . When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK.

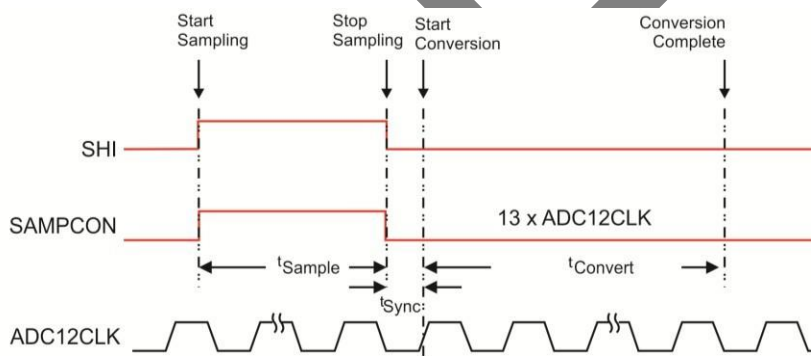


Figure 3.15 Extended Sample Mode

Pulse Sample Mode

The pulse sample mode is selected when SHP = 1. The SHI signal is used to trigger the sampling timer. The SHT0x and SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPCON sample period t_{sample} . The sampling timer keeps SAMPCON high after synchronization with AD12CLK for a programmed interval t_{sample} .

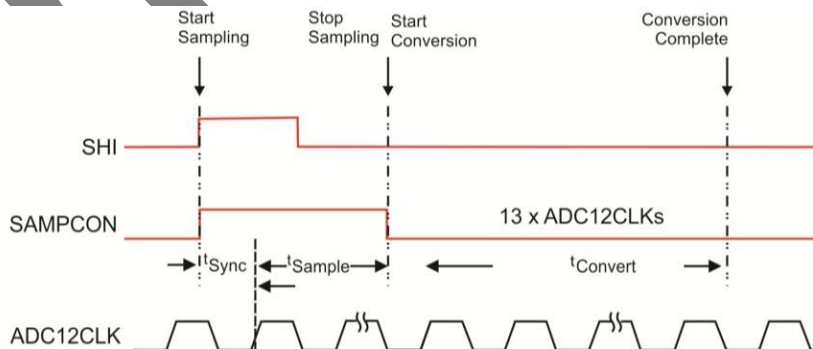


Figure 3.16 Pulse Sample Mode

The total sampling time is t_{sample} plus t_{sync} . The SHTx bits select the sampling time in 4x multiples of ADC12CLK. SHT0x selects the sampling time for ADC12MCTL0 to 7 and SHT1x selects the sampling time for ADC12MCTL8 to 15.

The sampling timing is discussed in the MSP430 series datasheet.

Conversion Memory

There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits define the voltage reference and the INCHx bits select the input channel. The EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the EOS bit in ADC12MCTL15 is not set.

The CSTARTADDx bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel the CSTARTADDx points to the single ADC12MCTLx to be used. If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. The sequence continues until an EOS bit in ADC12MCTLx is processed - this is the last control byte processed.

When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

3.7.3.5. ADC12 Conversion Modes

The ADC12 has four operating modes by the CONSEQx bits as discussed in the following Table

CONSEQx	Mode	Operation
00	Single channel Single-conversion	A single channel is converted once.
01	Sequence-of channels	A sequence of channels is converted once
10	Repeat single channel	A single channel is converted Repeatedly
11	Repeat sequence of-channels	A sequence of channels is Converted repeatedly

Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. When any other trigger source is used, ENC must be toggled between each conversion.

Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADCMEMx defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set EOS bit. When ADC12SC triggers a sequence, successive sequences can be triggered by the ADC12SC bit. When any other trigger source is used, ENC must be toggled between each sequence.

Repeat-Single-Channel Mode

A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion.

Multiple Sample and Conversion

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is used. When $MSC = 1$, $CONSEQx > 0$, and the sample timer is used, the first rising edge of the SH1 signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. The function of the ENC bit is unchanged when using the MSC bit.

Resetting the ENC bit in any time the conversion is stopped when it is in any mode of operation. Stopping ADC12 activity depends on the mode of operation.

Note: No EOS Bit Set For Sequence

If no EOS bit is set and a sequence mode is selected, resetting the ENC bit does not stop the sequence. To stop the sequence, first select a single-channel mode and then reset ENC.

3.7.3.6. Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel $INCHx = 1010$. The typical temperature sensor transfer function is $V_{TEMP} = 0.00355(TEMP_C) + 0.986$. When using the temperature sensor, the sample period must be greater than 30 microseconds. The temperature sensor offset error can be large, and may need to be calibrated for most applications. Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the VREF+ output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

ADC12 Grounding and Noise issues are discussed the MSP430 series datasheet.

3.7.3.7. ADC12 Interrupts

The ADC12 has 18 interrupt sources, they are

1. ADC12IFG0 to ADC12IFG15
2. ADC12OV (ADC12MEMx overflow)
3. ADC12TOV (ADC12 conversion time overflow)

The ADC12IFGx bits are set when their corresponding ADC12MEMx memory register is loaded with a conversion result. An interrupt request is generated if the corresponding interrupt is enabled by ADC12IEx bit and the GIE bit. The ADC12OV condition occurs when a conversion result is written to any ADC12MEMx before its previous conversion result was read. The ADC12TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed.

3.7.3.8. ADC12IV, Interrupt Vector Generator

All ADC12 interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which enabled ADC12 interrupt source requested an interrupt. The highest priority enabled ADC12 interrupt generates a number in the ADC12IV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled ADC12 interrupts do not affect the ADC12IV value.

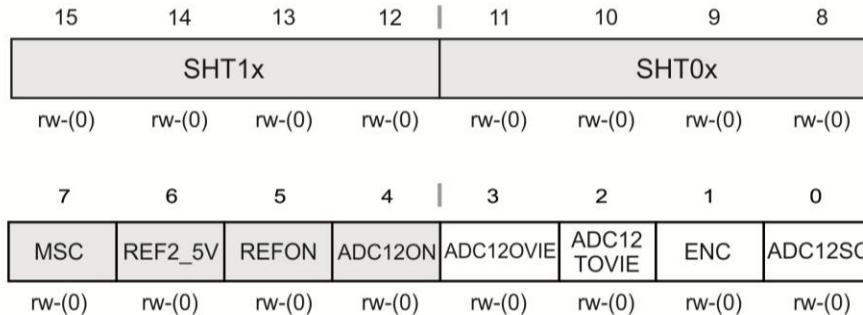
Any access read or writes, of the ADC12IV register automatically resets the ADC12OV condition or the ADC12TOV condition if either was the highest pending interrupt. Neither interrupt condition has an accessible interrupt flag. The ADC12IFGx flags are not reset by an ADC12IV access. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.


3.7.3.9. ADC12 Registers

Register	Short Form	Register Type	Address	Initial State
ADC12 control register 0	ADC12CTL0	Read/write	01A0h	Reset with POR
ADC12 control register 1	ADC12CTL1	Read/write	01A2h	Reset with POR
ADC12 interrupt flag register	ADC12IFG	Read/write	01A4h	Reset with POR
ADC12 interrupt enable register	ADC12IE	Read/write	01A6h	Reset with POR
ADC12 interrupt vector word	ADC12IV	Read	01A8h	Reset with POR
ADC12 memory 0	ADC12MEM0	Read/write	0140h	Unchanged
ADC12 memory 1	ADC12MEM1	Read/write	0142h	Unchanged
ADC12 memory 2	ADC12MEM2	Read/write	0144h	Unchanged
ADC12 memory 3	ADC12MEM3	Read/write	0146h	Unchanged
ADC12 memory 4	ADC12MEM4	Read/write	0148h	Unchanged
ADC12 memory 5	ADC12MEM5	Read/write	014Ah	Unchanged
ADC12 memory 6	ADC12MEM6	Read/write	014Ch	Unchanged
ADC12 memory 7	ADC12MEM7	Read/write	014Eh	Unchanged
ADC12 memory 8	ADC12MEM8	Read/write	0150h	Unchanged
ADC12 memory 9	ADC12MEM9	Read/write	0152h	Unchanged
ADC12 memory 10	ADC12MEM10	Read/write	0154h	Unchanged
ADC12 memory 11	ADC12MEM11	Read/write	0156h	Unchanged
ADC12 memory 12	ADC12MEM12	Read/write	0158h	Unchanged
ADC12 memory 13	ADC12MEM13	Read/write	015Ah	Unchanged
ADC12 memory 14	ADC12MEM14	Read/write	015Ch	Unchanged
ADC12 memory 15	ADC12MEM15	Read/write	015Eh	Unchanged
ADC12 memory control 0	ADC12MCTL0	Read/write	080h	Reset with POR
ADC12 memory control 1	ADC12MCTL1	Read/write	081h	Reset with POR
ADC12 memory control 2	ADC12MCTL2	Read/write	082h	Reset with POR
ADC12 memory control 3	ADC12MCTL3	Read/write	083h	Reset with POR
ADC12 memory control 4	ADC12MCTL4	Read/write	084h	Reset with POR
ADC12 memory control 5	ADC12MCTL5	Read/write	085h	Reset with POR
ADC12 memory control 6	ADC12MCTL6	Read/write	086h	Reset with POR
ADC12 memory control 7	ADC12MCTL7	Read/write	087h	Reset with POR
ADC12 memory control 8	ADC12MCTL8	Read/write	088h	Reset with POR
ADC12 memory control 9	ADC12MCTL9	Read/write	089h	Reset with POR
ADC12 memory control 10	ADC12MCTL10	Read/write	08Ah	Reset with POR
ADC12 memory control 11	ADC12MCTL11	Read/write	08Bh	Reset with POR

ADC12 memory control 12	ADC12MCTL12	Read/write	08Ch	Reset with POR
ADC12 memory control 13	ADC12MCTL13	Read/write	08Dh	Reset with POR
ADC12 memory control 14	ADC12MCTL14	Read/write	08Eh	Reset with POR
ADC12 memory control 15	ADC12MCTL15	Read/write	08Fh	Reset with POR

ADC12CTL0, ADC12 Control Register 0

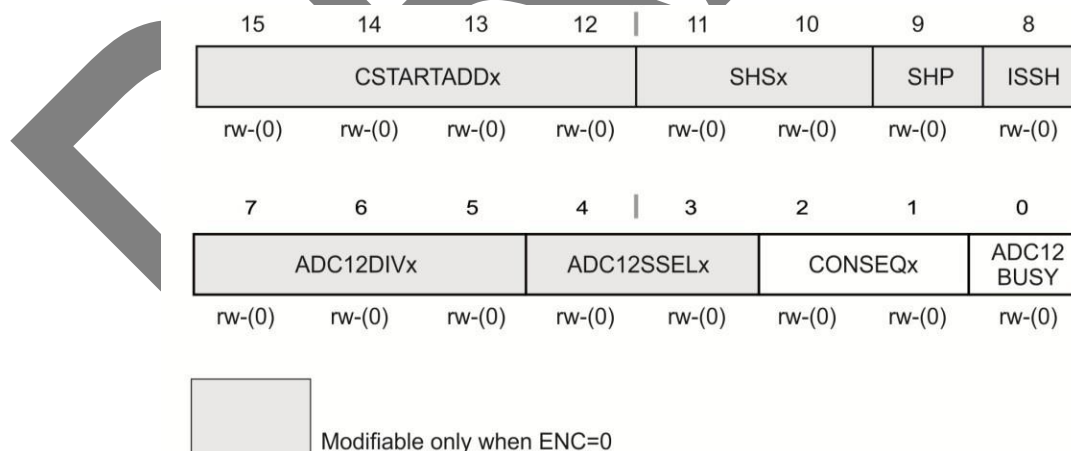


 Modifiable only when ENC=0

Name	Bits	Description																														
SHT1x	15-12	Sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.																														
SHT0x	11-8	Sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7.																														
		<table border="1"> <thead> <tr> <th>SHTx Bits</th> <th>ADC12CLK cycles</th> <th>SHTx Bits</th> <th>ADC12CLK cycles</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>4</td> <td>0111</td> <td>192</td> </tr> <tr> <td>0001</td> <td>8</td> <td>1000</td> <td>256</td> </tr> <tr> <td>0010</td> <td>16</td> <td>1001</td> <td>384</td> </tr> <tr> <td>0011</td> <td>32</td> <td>1010</td> <td>512</td> </tr> <tr> <td>0100</td> <td>64</td> <td>1011</td> <td>768</td> </tr> <tr> <td>0101</td> <td>96</td> <td rowspan="2">1100-1111</td> <td rowspan="2">1024</td> </tr> <tr> <td>0110</td> <td>128</td> </tr> </tbody> </table>	SHTx Bits	ADC12CLK cycles	SHTx Bits	ADC12CLK cycles	0000	4	0111	192	0001	8	1000	256	0010	16	1001	384	0011	32	1010	512	0100	64	1011	768	0101	96	1100-1111	1024	0110	128
SHTx Bits	ADC12CLK cycles	SHTx Bits	ADC12CLK cycles																													
0000	4	0111	192																													
0001	8	1000	256																													
0010	16	1001	384																													
0011	32	1010	512																													
0100	64	1011	768																													
0101	96	1100-1111	1024																													
0110	128																															
MSC	7	<p>Multiple sample and conversion. Valid only for sequence or Repeated modes.</p> <p>0 -The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-conversion.</p> <p>1 -The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior</p>																														

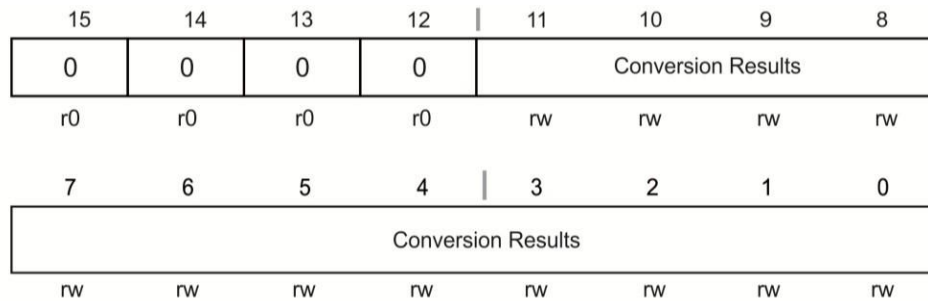
		conversion is completed.
REF2_5V	6	Reference generator voltage. REFON must also be set. 0 -1.5 V, 1 -2.5 V
REFON	5	Reference generator on 0 -Reference off 1 -Reference on
ADC12ON	4	ADC12 on 0 ADC12 off, 1 ADC12 on
ADC12OVIE	3	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0 -Overflow interrupt disabled, 1 -Overflow interrupt enabled
ADC12TOVIE	2	ADC12 conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 -Conversion time overflow interrupt disabled 1 -Conversion time overflow interrupt enabled
ENC	1	Enable conversion
ADC12SC	0	Start conversion. Software-controlled sample-and-conversion start. ADC12SC and ENC may be set together with one Instruction.ADC12SC is reset automatically. 0 - No sample-and-conversion-start 1 - Start sample-and-conversion

ADC12CTL1, ADC12 Control Register 1



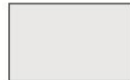
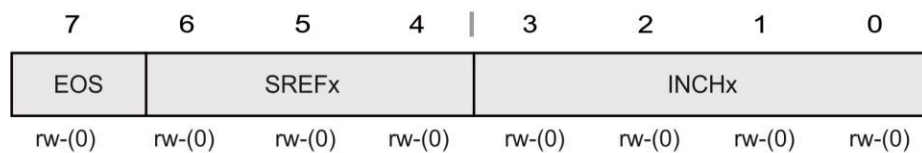
Name	Bits	Description
CSTARTADDx	15-12	Conversion start address. These bits select which ADC1 conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.
SHSx	11-10	Sample-and-hold source select 00 -ADC12SC bit 01 -Timer_A.OUT1 10 -Timer_B.OUT0 11 -Timer_B.OUT1
SHP	9	Sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. 0 - SAMPCON signal is sourced from the sample-input signal. 1 -SAMPCON signal is sourced from the sampling timer.
ISSH	8	Invert signal sample-and-hold 0 The sample-input signal is not inverted. 1 The sample-input signal is inverted.
ADC12DIVx	7-5	ADC12 clock divider 000-/1, 001-/2, 010-/3, 011-/4 100-/5, 101-/6, 110-/7, 111-/8
ADC12SSELx	4-3	ADC12 clock source select 00-ADC12OSC ,01-ACLK,10-MCLK,11-SMCLK
CONSEQx	2-1	Conversion sequence mode select 00 -Single-channel, single-conversion 01 -Sequence-of-channels 10 -Repeat-single-channel 11 -Repeat-sequence-of-channels
ADC12BUSY	0	Bit 0 ADC12 busy. This bit indicates an active sample or conversion operation. 0-No operation is active., 1-A sequence, sample, or conversion is active.

ADC12MEMx, ADC12 Conversion Memory Registers



CONVERSION RESULTS	15-0	12-bit conversion results are right-justified. Bit 11 is the MSB. Bits 15-12 are always 0. Writing to the conversion memory registers will corrupt the results always
---------------------------	------	---

ADC12MCTLx, ADC12 Conversion Memory Control Registers

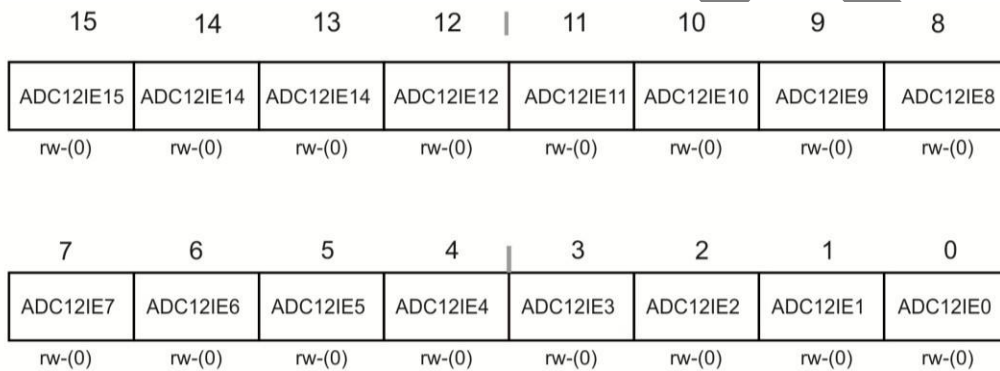


Modifiable only when ENC=0

Name	Bits	Description
EOS	7	End of sequence. Indicates the last conversion in a sequence. 0 - Not end of sequence, 1 - End of sequence
SREFx	6-4	Select reference 0 -VR+ = AVCC and VR- = AVSS 1 -VR+ = VREF+ and VR- = AVSS 10 -VR+ = VeREF+ and VR- = AVSS 11 -VR+ = VeREF+ and VR- = AVSS 100 -VR+ = AVCC and VR- = VREF-/ VeREF- 101 -VR+ = VREF+ and VR- = VREF-/ VeREF- 110 -VR+ = VeREF+ and VR- = VREF-/ VeREF- 111 -VR+ = VeREF+ and VR- = VREF-/ VeREF-
INCHx	3-0	Input channel select 0000 -A0,001-A1,010-A2,011-A3 0100 -A4,101-A5,110-A6,11-A7 1000 -VeREF+,001-VREF-/VeREF-

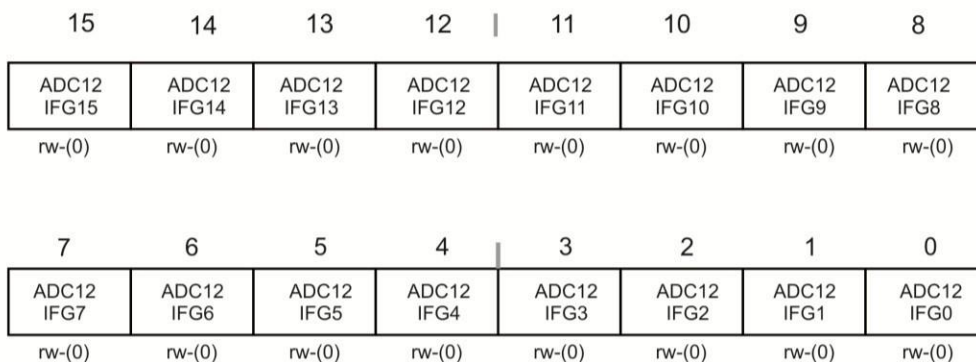
		1010-Temperature sensor 1011 (AVCC – AVSS) / 2 1100 (AVCC – AVSS) / 2, A12 on "FG43x and "FG461x devices 1101 (AVCC – AVSS) / 2, A13on "FG43x and "FG461x devices 1110 (AVCC – AVSS) / 2, A14on "FG43x and "FG461x devices 1111 (AVCC – AVSS) / 2, A15on "FG43x and "FG461x devices
--	--	--

ADC12IE, ADC12 Interrupt Enable Register



ADC12IE_x	15-0	Interrupt enable. These bits enable or disable the interrupt request For the ADC12IFG _x bits. 0-Interrupt disabled,1-Interrupt enabled
----------------------------	------	--

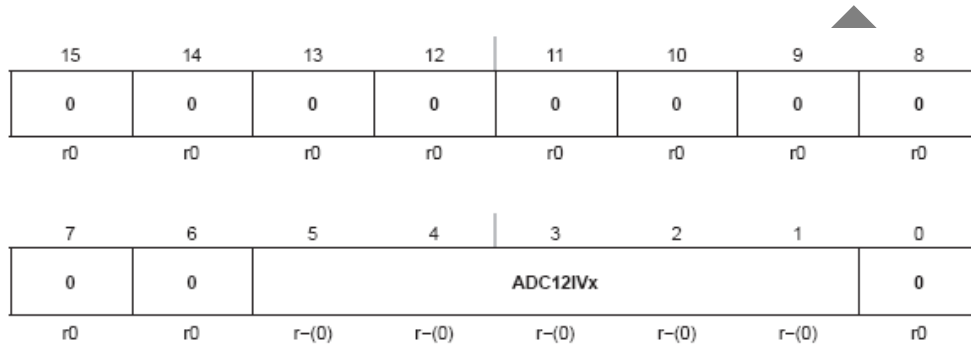
ADC12IFG, ADC12 Interrupt Flag Register



ADC12IFG_x	15-0	ADC12MEM _x Interrupt flag. These bits are set when corresponding ADC12MEM _x is loaded with a conversion result. The ADC12IFG _x bits are reset
-----------------------------	------	---

		if the corresponding ADC12MEMx is accessed, or may be reset with software. 0 -No interrupt pending, 1-Interrupt pending
--	--	--

ADC12IV, ADC12 Interrupt Vector Register



ADC12IVx Bits ADC12 interrupt vector value

ADC12IV	Interrupt source	Interrupt flag	Interrupt priority
000h	No interrupt pending	-	
002h	ADC12MEMx OVERFLOW	-	Highest
004h	Conversion time overflow	-	
006h	ADC12MEM0 interrupt flag	ADC12IFG0	
008h	ADC12MEM1 interrupt flag	ADC12IFG1	
00Ah	ADC12MEM2 interrupt flag	ADC12IFG2	
00Ch	ADC12MEM3 interrupt flag	ADC12IFG3	
00Eh	ADC12MEM4 interrupt flag	ADC12IFG4	
012h	ADC12MEM5 interrupt flag	ADC12IFG5	
014h	ADC12MEM6 interrupt flag	ADC12IFG6	
016h	ADC12MEM7 interrupt flag	ADC12IFG7	
018h	ADC12MEM8 interrupt flag	ADC12IFG8	
01Ah	ADC12MEM9 interrupt flag	ADC12IFG9	
01Ch	ADC12MEM10 interrupt flag	ADC12IFG10	
01Eh	ADC12MEM12 interrupt flag	ADC12IFG11	
020h	ADC12MEM13 interrupt flag	ADC12IFG12	
022h	ADC12MEM14 interrupt flag	ADC12IFG13	
024h	ADC12MEM15 interrupt flag	ADC12IFG14	Lowest

Configuring ADC12 in MSP430x5xxx microcontroller:

Before reading the values from the analog sensor, it is required to configure the ADC present inside the MSP430 microcontroller. As explained above, there are dedicated registers present for setting up the ADC. The MSP430x5xxx controller consists of a 12 bit ADC called as ADC12. This segment discusses the steps followed in configuring the inbuilt ADC12.

The following are the registers related to ADC12 that we use in our program:

Register	Short Form	Address	Function
ADC12 control register 0	ADC12CTL0	01A0h	Control and configure ADC12
ADC12 control register 1	ADC12CTL1	01A2h	Control and configure ADC12
ADC12 Interrupt enable register	ADC12IE	01A6h	Enabling the interrupts
ADC12 memory 0	ADC12MEM0	0140h	Stores 12 bit converted value

Now consider the various configuration parameters related to the various registers that we use in setting up the ADC12:

Parameter	Denoted by	Location	Comments
Turn on ADC12	ADC12ON	ADC12CTL0, bit 4	Overall on/off for ADC
Enable Conversion	ADC12ENC	ADC12CTL0, bit 1	Enables conversion
Start Conversion	ADC12SC	ADC12CTL0, bit 0	Start conversion (writing a 1 starts sampling and conversion), returns to 0 automatically.
Port x function selection		PxSEL	A set bit selects the peripheral module function
Interrupt enable		ADC12IE	Used to service an interrupt when a value is received on the ADC. The corresponding flag that is set is ADC10IFG.
Reference Generator	ADC12REFON, ADC12REF2_5V	ADC12CTL0 bits 5,6	Enable optional internal reference voltage of either 1.5 or 2.5V
Sample and hold time	ADC12SHT0x	ADC12CTL0 bits	Time used for sample and hold. Depends on circuit time constants (see data sheet). Default is 4 ADC clock cycles

Sample Code for configuring ADC12:

```
#include <msp430.h>
unsigned int value_adc;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Stop WDT
    ADC12CTL0 = ADC12SHT02 + ADC12ON;   // Sampling time, ADC12 on
    ADC12CTL1 = ADC12SHP;               // Use sampling timer
    ADC12CTL0 |= ADC12ENC;
    P6SEL |= 0x01;                      // P6.0 ADC option select for
    choosing channel A0
    while (1)
    {
        ADC12CTL0 |= ADC12SC;           // Start sampling/conversion
        value_adc=ADC12MEM0;
    }
}
```

Flowchart :

The flowchart given in the figure illustrates the procedure in the configuration of inbuilt ADC in MSP430x5xxx microcontrollers.

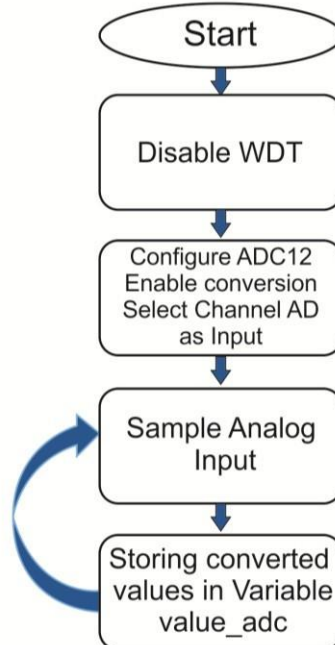


Fig 3.17. Flowchart for configuring and storing ADC values

Example Application for ADC: Pulse- Oximeter:

Various analog sensors that are present in the market take in physical values and provide a voltage reading for it. One such application using LED and photodiode optical sensor is the pulse-oximeter. Pulse-oximeter is a device used to measure oxygen levels of a person.

The necessary components in a pulse-oximeter are shown below:

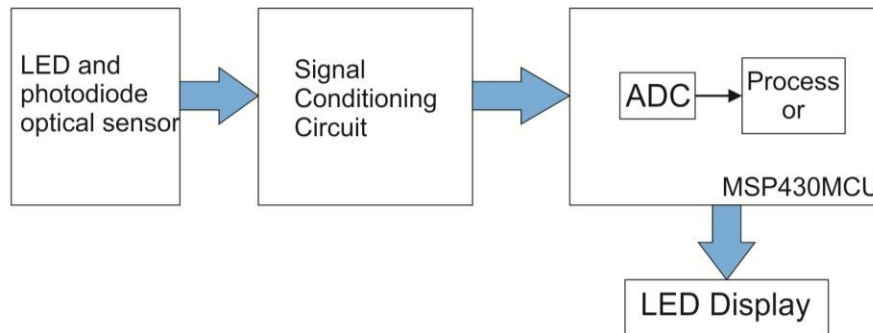


Fig 3.18 Block Diagram for Pulse-Oximeter

The LED and photodiode optical sensor senses the oxygen content present in the blood. The sensor produces voltage for the corresponding oxygen content. The voltage is passed to the MSP430 microcontroller through a signal conditioning circuit.

The MSP430x5xxx series microcontroller uses a 12 bit SAR ADC giving high resolution for better accuracy. The ADC converts the analog voltage values to digital signals to make it usable for processing. The data is processed by the processor and the corresponding oxygen readings are displayed on LCD screen of the device.

A flowchart showing the configuration of ADC in MSP430 microcontroller is shown below:

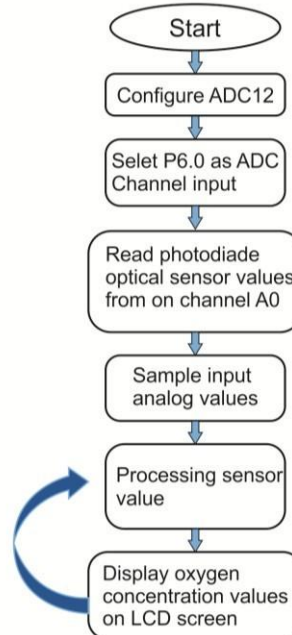


Fig 3.19 Flowchart for configuring ADC in MSP430 in pulse-oximeter

3.7.4. Comparator

MSP430 has inbuilt analog comparator based on open loop operational amplifier, compares inbuilt reference voltage with unknown analog input. This is implemented in MSP430x11x1, MSP430x12x, MSP430x13x, MSP430x14x, MSP430x15x and MSP430x16x devices. The comparator_A module supports for selecting the analog input to inverting or non-inverting of the comparator. The comparator output is RC-filtered and provided to CAOUT bit, Timer_A capture input and to generate interrupt.

This module is an analog precision slope for analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals. Comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, the CAOUT is always low.

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the P2CAx bits. It does the selection of the external signal to the + and – terminals of the comparator and routing of an internal reference voltage to an associated output port pin

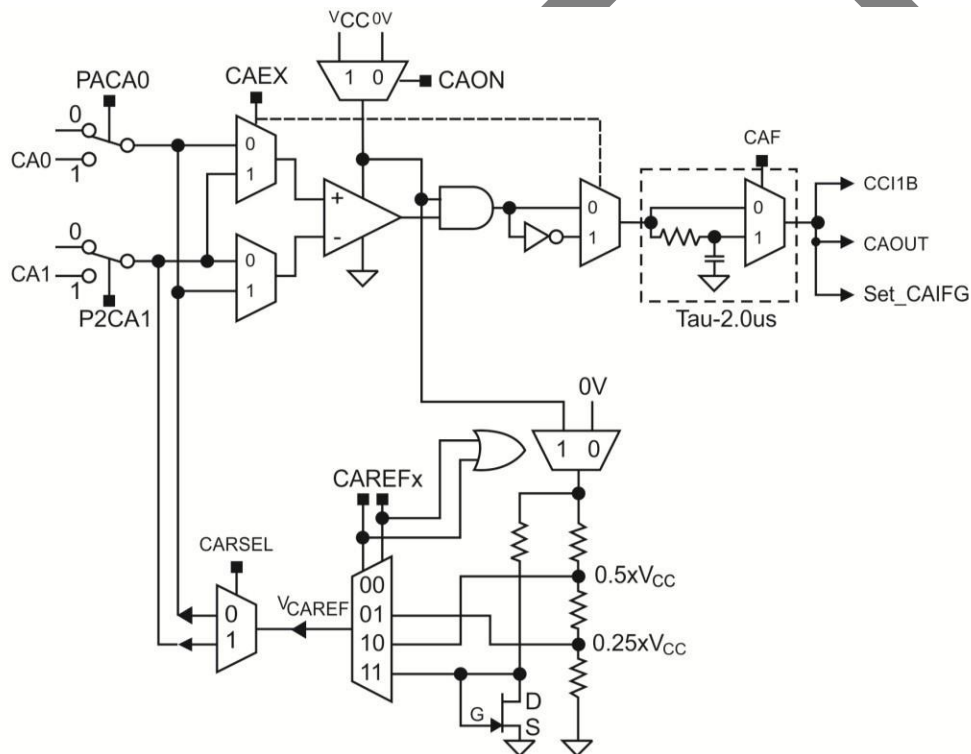


Figure 3.17 Block Diagram of Comparator_A

Note: Comparator Input Connection

“When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption”.

Any comparator output oscillates if the voltage difference across the input terminals is small. Internal and external parasitic effects and cross coupling on and between signal lines, power supply lines, and other parts of the system are responsible for this behavior as shown in Figure below. The comparator output oscillation reduces accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

3.7.4.1. Voltage Reference Generator

The voltage reference generator is used to generate VCAREF, which can be applied to either comparator input terminal. The CAREFx bits control the output of the voltage generator. The CARSEL bit selects the comparator terminal to which VCAREF is applied. If external signals are applied to both comparator input terminals, the internal reference generator should be turned off to reduce current consumption. The voltage reference generator can generate a fraction of the device's VCC or a fixed transistor threshold voltage of ~ 0.55 V.

3.7.4.2. Comparator and Port Disabling

The comparator input and output functions are multiplexed with the associated I/O port pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from VCC to GND. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption. The CAPDx bits, when set, disable the corresponding P2 input buffer as shown in Figure below. When current consumption is critical, any P2 pin connected to analog signals should be disabled with their associated CAPDx bit.

3.7.4.3. Comparator_A Interrupts

One interrupt flag and one interrupt vector are associated with the Comparator_A as shown in Figure below. The interrupt flag CAIFG is set on either the rising or falling edge of the comparator output, selected by the CAIES bit. If both the CAIE and the GIE bits are set, then the CAIFG flag generates an interrupt request. The CAIFG flag is automatically reset when the interrupt request is serviced or may be reset with software.

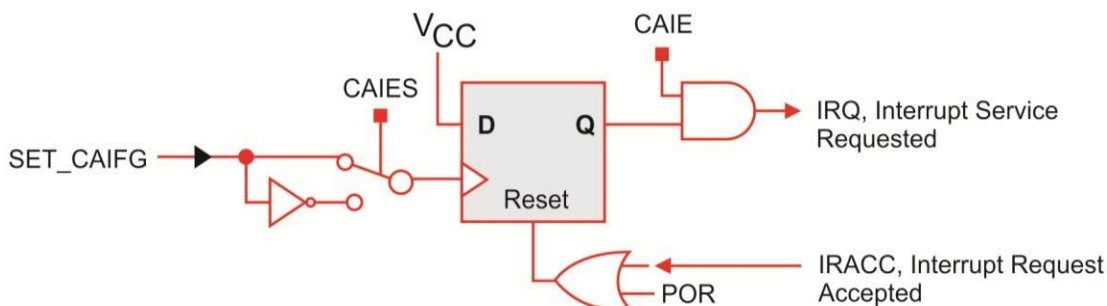


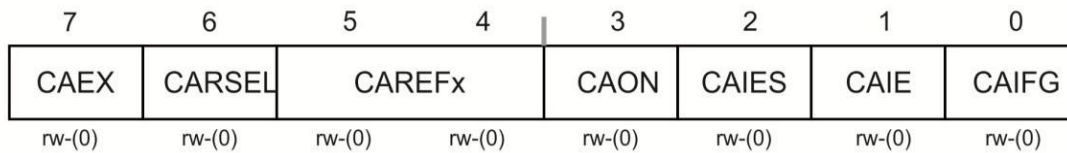
Figure 3.18 Comparator_A Interrupt System

3.7.4.4. Comparator_A Registers

Register	Short Form	Register type	Address	Initial state
Comparator_A control register 1	CACTL1	Read/write	059h	Reset with POR
Comparator_A control register 2	CACTL2	Read/write	05Ah	Reset with POR
Comparator_A control register 1	CAPD	Read/write	05Bh	Reset with POR

Comparator_A Registers

CACTL1, Comparator_A Control Register 1



Name	Bits	Description
CAEX	7	Comparator_A exchange This bit exchanges the comparator inputs and inverts the comparator output.
CARSL	6	Comparator_A reference select. This bit selects which terminal the VCAREF is applied to. When CAEX = 0:
CAREF_x	5-4	Comparator_A reference. These bits select the reference voltage VCAREF. 00 -Internal reference off. An external reference can be applied.
CAON	3	Comparator_A on. This bit turns on the comparator. When the Comparator is off it consumes no current. The reference circuitry is enabled or disabled independently. 0-Off , 1-On
CAIES	2	Comparator_A interrupt edge select 0-Rising edge, 1-Falling edge
CAIE	1	Comparator_A interrupt enable 0-Disabled, 1-Enabled
CAIFG	0	The Comparator_A interrupt flag 0-No interrupt pending, 1-Interrupt pending

CACTL2, Comparator_A, Control Register



Name	Bits	Description
Unused	7-4	Unused
P2CA1	3	Pin to CA1. This bit selects the CA1 pin function. 0-The pin is not connected to CA1, 1-The pin is connected to CA1
P2CA0	2	Pin to CA0. Pin to CA0. This bit selects the CA0 pin function. 0-The pin is not connected to CA0,1-The pin is connected to CA0
CAF	1	Comparator_A output filter 0-Comparator_A output is not filtered, 1-Comparator_A output is filtered
CAOUT	0	Comparator_A output. This bit reflects the value of the Comparator output. Writing this bit has no effect.

CAPD, Comparator_A, Port Disable Register

	7	6	5	4	3	2	1	0
	CAPD7	CAPD6	CAPD5	CAPD4	CAPD3	CAPD2	CAPD1	CAPD0
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
CAPDx	7-0							
	Comparator_A port disable. These bits individually disable the Input buffer for the pins of the port associated with Comparator_A. For example, if CA0 is on pin P2.3, the CAPDx bits can be used to individually enable or disable each P2.x pin buffer. CAPD0 disables P2.0, CAPD1 disables P2.1, etc. 0-The input buffer is enabled, 1- The input buffer is disabled.							

3.8. Direct Memory Access (DMA)

The DMA controller module transfers data from one address to another i.e. from peripherals or memory to memory or vice versa without CPU intervention i.e. by passing the data through the CPU. In an embedded system, DMA capability offers enhanced opportunities for implementing low-power solutions for data transfer intensive applications. DMA controller can operate as an I/O device when is configured by the CPU or as a bus master device. When operating as a bus master, the DMA replaces the CPU in regulating data transfers in the system buses in such a way that DMA transfers the data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can transfer data from the ADC12 conversion memory to RAM. Using the DMA controller one 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.

3.8.1. DMA controller in MSP430:

The DMA controller in MSP430 has three independent transfer channels and it is configurable for its DMA channel priorities. It requires only two MCLK clock cycles to transfer a Byte or word and mixed byte/word. DMA support the transfer block sizes up to 65535 bytes or words.

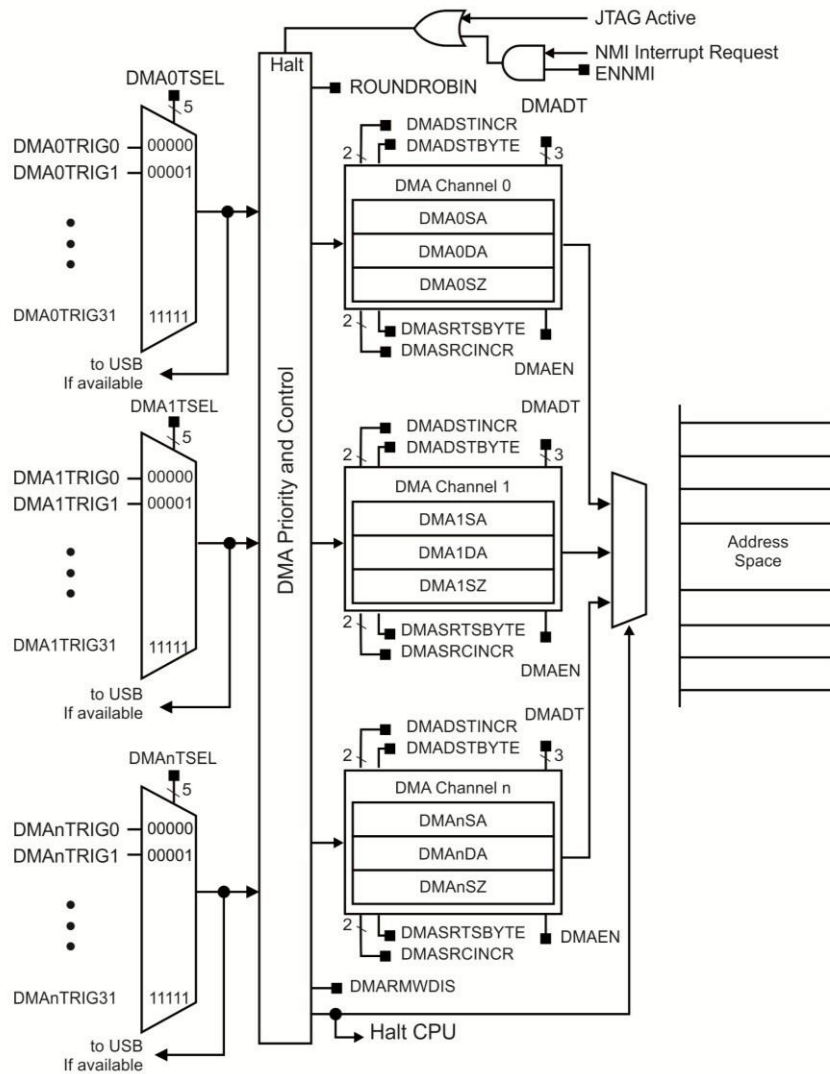
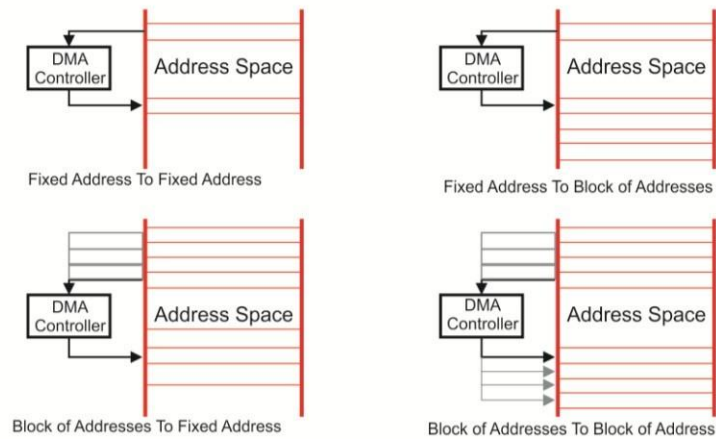


Figure 3.20 Block Diagram of DMA controller

3.8.2. DMA Addressing Modes

The DMA controller has four addressing modes they are 1. Fixed address to fixed address, 2.Fixed address to block of addresses, 3.Block of addresses to fixed address, 4.Block of addresses to block of addresses


Figure 3.20 DMA 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 using the DMA control (DMACTL0) register. The source and destination address of the DMA transfer mechanism is incremented, decremented, or unchanged after each transfer by DMA_SRCINCRx bits and DMA_DSTINCRx bits. The DMA channel x transfer size register DMAxSZ is used to define the number of transfers to be made. 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.

3.8.3. DMA Transfer Modes

The DMA controller has six transfer modes selected by the DMADTx bits as listed in Table below. Each channel is individually configurable for its transfer mode. The transfer mode is configured independently from the addressing mode. Any addressing mode can be used with any transfer mode.

DMADTx	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.

DMA Transfer Modes

3.8.3.1. Single Transfer

In this mode, each byte/word transfer requires a separate DMA trigger. The single transfer state diagram is shown in data sheet of the MSP430 series. 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 reaches zero, it is reloaded from its temporary register and the corresponding DMA interrupt flag DMAIFG flag is set. When DMADTx = 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.

3.8.3.2. Block Transfers

During a block transfer, the CPU is halted until the complete block has been transferred. The block transfer takes $2 \times \text{MCLK} \times \text{DMAxSZ}$ clock cycles to complete. CPU execution resumes with its previous state after the block transfer is complete.

In this mode, a transfer of a complete block of data occurs after one trigger. The DMAxSZ register is used to define the size of the block and the DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer of 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. When DMADTx = 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.

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. The block transfer state diagram is discussed in the respective datasheet.

3.8.3.3. Burst-Block Transfers

In this mode, transfers are block transfers with CPU activity interleaved. The CPU executes 2 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 below.

The DMAxSZ register is used to define the size of the block and the DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer of the block. When the DMAxSZ register decrements to zero it is reloaded with initialized value which is already in 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 NMI interrupt when ENNMI is set. The CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.

3.8.4. Initiating DMA Transfers

Each DMA channel is independently configured for its trigger source with the DMAxTSELx bits. The DMAxTSELx bits should be modified only when DMA is not enabled. When selecting the trigger, the trigger must not have already occurred, or the transfer will not take place. For example, if an interrupt

flag bit is selected as a trigger, and it is already set, no transfer will occur until the next time the same interrupt flag is set.

3.8.4.1. Edge-Sensitive Triggers

When DMALEVEL = 0, edge-sensitive triggers are used and the rising edge of the trigger signal initiates the transfer. In single-transfer mode, each transfer requires its own trigger. When using block or burst-block modes, only one trigger is required to initiate the block or burst-block transfer.

3.8.4.2. Level-Sensitive Triggers

When DMALEVEL = 1, level-sensitive triggers are used. Level-sensitive triggers can only be used when external trigger DMAE0 is selected as the trigger. DMA transfers are triggered as long as the trigger signal is high and the DMAEN bit remains set. The trigger signal must remain high for a block or burst-block transfer to complete. If the trigger signal goes low during a block or burst-block transfer the DMA controller is held in its current state until the trigger goes back high or until the DMA registers are modified by software. When the trigger signal goes high again, the transfer resumes from where it was when the trigger signal went low. When DMALEVEL = 1, transfer modes selected when DMADTx = {0, 1, 2, 3} are recommended because the DMAEN bit is automatically reset after the configured transfer.

3.8.4.3. Stopping CPU for DMA Transfers

The DMAONFETCH bit controls when the CPU is halted for a DMA transfer. When DMAONFETCH = 0, the CPU is halted immediately and the transfer begins when a trigger is received. When DMAONFETCH = 1, the CPU finishes the currently executing instruction before the DMA controller halts the CPU and the transfer begins. If the DMA controller is used to write flash memory (program memory space) the DMAONFETCH bit must be set. Otherwise, unpredictable operation can result.

3.8.4.4. Stopping DMA Transfers

A single, block, or burst-block transfer may be stopped with an NMI interrupt, if the ENNMI bit is set in register DMACTL1. A burst-block transfer may be stopped by clearing the DMAEN bit.

3.8.5. DMA Channel Priorities

The DMA channel's default priorities are DMA0–DMA1–DMA2 by ROUNDROBIN. If two or three triggers are in pending, the channel with the highest priority completes its transfer first. Transfers in progress are not halted if a higher priority channel is triggered. The higher priority channel waits until the transfer of last one. DMA transfer cycle is discussed in the respective device datasheet.

3.8.6. DMA with System Interrupts

DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the transfer. NMI interrupts can interrupt the DMA controller if the ENNMI bit is set. DMA transfer can interrupt the CPU when it is in system interrupt service routines.

3.8.7. DMA Controller Interrupts

The interrupt vector for DMA module and DAC12 module are common. The software should verify DMAIFG and DAC12IFG for the source of interrupt. Upon detection of interrupt by DMAIFG, the software should check for DMAxIFG in the DMA interrupt vector value register to identify the source of interrupt by DMA channels. If the corresponding DMAIE and GIE bits are set, DMA interrupt request is generated.

3.8.8. DMA Controller and I2C Module

The I2C module provides two trigger sources for the DMA controller. The I2C module can trigger a transfer when new I2C data is received and the when the transmit data is needed. The TXDMAEN and

RXDMAEN bits enable or disable the use of the DMA controller with the I2C module. When RXDMAEN = 1, the DMA controller can be used to transfer data from the I2C module after the I2C module receives data. When RXDMAEN = 0, RXRDYIE is ignored and RXRDYIFG will not generate an interrupt. When TXDMAEN = 1, the DMA controller can be used to transfer data to the I2C module for transmission. When TXDMAEN = 0, TXRDYIE is ignored and TXRDYIFG will not generate an interrupt.

3.8.9. DMA Controller and ADC12

MSP430 devices with an integrated DMA controller can automatically move data from any ADC12MEMx register to another location. DMA transfers are done without CPU intervention and independently of any low-power modes.

The DMA controller increases throughput of the ADC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

DMA transfers can be triggered from any ADC12IFGx flag. When CONSEQx = {0,2} the ADC12IFGx flag for the ADC12MEMx used for the conversion can trigger a DMA transfer. When CONSEQx = {1,3}, the ADC12IFGx flag for the last ADC12MEMx in the sequence can trigger a DMA transfer. Any ADC12IFGx flag is automatically cleared when the DMA controller accesses the corresponding ADC12MEMx.

3.8.10. DMA Controller and DAC12

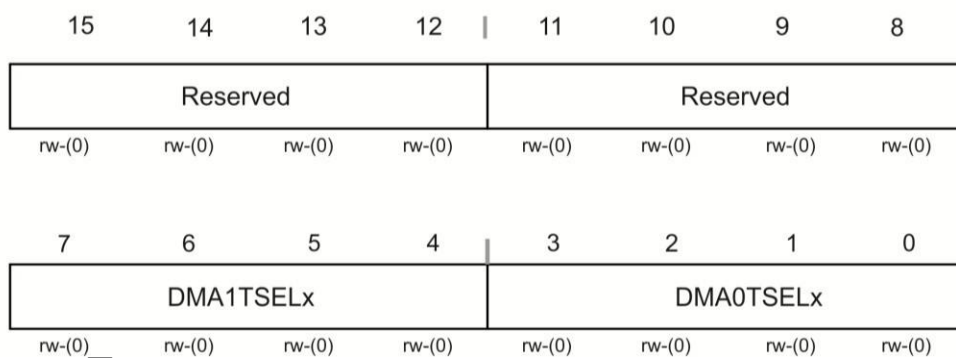
MSP430 devices with an integrated DMA controller can automatically move data to the DAC12_xDAT register. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput to the DAC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

Applications requiring periodic waveform generation can benefit from using the DMA controller with the DAC12. For example, an application that produces a sinusoidal waveform may store the sinusoid values in a table. The DMA controller can continuously and automatically transfer the values to the DAC12 at specific intervals creating the sinusoid with zero CPU execution. The DAC12_xCTL DAC12IFG flag is automatically cleared when the DMA controller accesses the DAC12_xDAT register.

3.8.11. DMA Registers

Register	Short Form	Register Type	Address	Initial State
DMA control 0	DMACTL0	Read/write	0122h	Reset with POR
DMA control 1	DMACTL1	Read/write	0124h	Reset with POR
DMA channel 0 control	DMA0CTL	Read/write	01E0h	Reset with POR
DMA channel 0 source address	DMA0SA	Read/write	01E2h	Unchanged
DMA channel 0 destination address	DMA0DA	Read/write	01E4h	Unchanged
DMA channel 0 transfer size	DMA0SZ	Read/write	01E6h	Unchanged
DMA channel 1 control	DMA1CTL	Read/write	01E8h	Reset with POR
DMA channel 1 source address	DMA1SA	Read/write	01EAh	Unchanged
DMA channel 1 destination address	DMA1DA	Read/write	01ECh	Unchanged
DMA channel 1 transfer size	DMA1SZ	Read/write	01EEh	Unchanged
DMA channel 2 control	DMA2CTL	Read/write	01F0h	Reset with POR
DMA channel 2 source address	DMA2SA	Read/write	01F2h	Unchanged
DMA channel 2 destination address	DMA2DA	Read/write	01F4h	Unchanged
DMA channel 2 transfer size	DMA2SZ	Read/write	01F6h	Unchanged

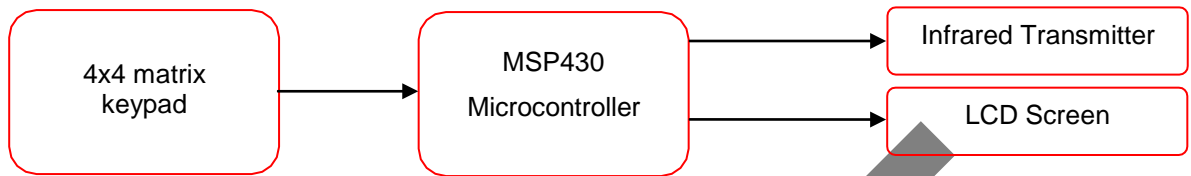
DMACTL0.DMA Control Register 0



Name	Bit	Description
Not in Use	15-12	Not in use
DMA2 TSELx	11-8	DMA trigger select. These bits select the DMA transfer trigger 0000 DMAREQ bit (software trigger) 0001 TACCR2 CCIFG bit 0010 TBCCR2 CCIFG bit 0011 URXIFG0 (UART/SPI mode), USART0 data received 0100 UTXIFG0 (UART/SPI mode), USART0 transmit ready (I2Cmode) 0101 DAC12_OCTL DAC12IFG bit 0110 ADC12 ADC12IFGx bit 0111 TACCR0 CCIFG bit 1000 TBCCR0 CCIFG bit 1001 URXIFG1 bit 1010 UTXIFG1 bit 1011 Multiplier ready 1100 No action 1101 No action 1110 DMA0IFG bit triggers DMA channel 1 DMA1IFG bit triggers DMA channel 2 DMA2IFG bit triggers DMA channel 0 1111 External trigger DMAE0
DMA1TSELx	7-4	Same as DMA2TSELx
DMA0TSEL	3-0	DMA2TSELx

Modern air conditioning systems available in the current market are automated systems. These air conditioners have a remote control system available for user to control various operations in the AC like turning on and off, sleep mode, setting a timer, shutting off the compressor, setting the temperature, controlling the panel light and entering the energy saving mode. The remote controller uses pulse width modulation to generate infrared signals for communication with the air condition system.

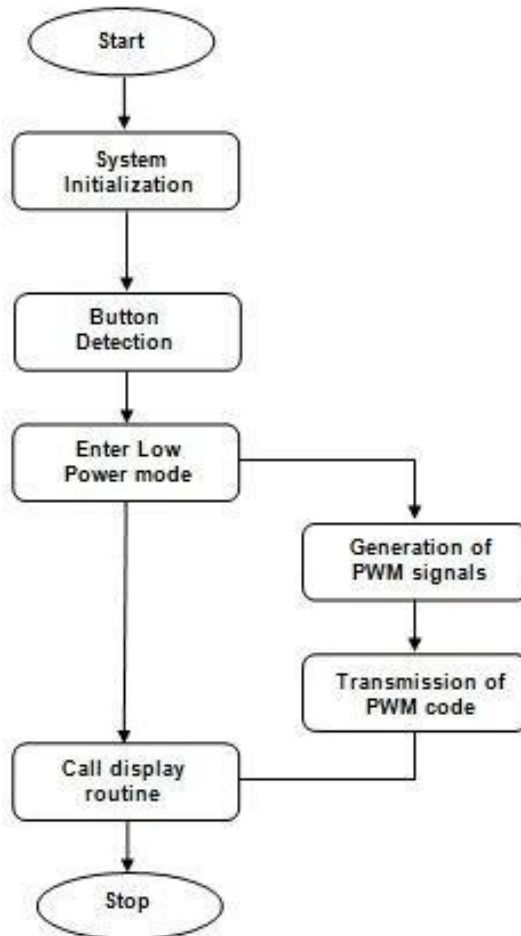
- A generic remote controller requires the necessary components:
- Keypad: Generally 4x4 matrix keypad is used to provide 15 different inputs.
- Microcontroller (MSP430): A microcontroller to control the complete operation
- Infrared Transmitter: An infrared LED to transmit the signal
- LCD Screen: To display the set values



Remote Control Block Diagram

Work Flow: The operation begins with the system initialization. The system initialization includes setting up the inputs and outputs pins and enabling the interrupts. Interrupts are enabled because the buttons of the keypad are connected to the interrupt-driven GPIOs. After a button is pressed, an interrupt is generated and the device enters into low power mode. PWM signals are generated in the interrupt service routine. After the generation, the PWM code is transmitted from the infrared LED transmitter. The ISR is completed and a display routine is called to display the set readings on the LCD screen connected to the microcontroller.

A flowchart is shown below to understand the remote control system:



3.9. Summary:

The chapter began with listing the importance of the presence of on chip peripherals in microcontrollers.

Important peripherals like timers, comparators, ADC were discussed in context of an MSP430 controller. Associated system registers along with configuration and application were also explained. PWM control, an important technique used to produce varying analog voltages was also described and its configuration using timers was explained with a simple application. The chapter was closed with a case study of an air conditioner remote controller developed using MSP430 microcontroller.

DRAFT

3.10. Review Questions

DRAFT

3.11. Exercises

DRAFT

Communication Protocols and Interfacing with External Devices

The previous chapter covered the process of configuring various interfaces such as PWM, ADC and comparator with theMSP430 microcontroller.

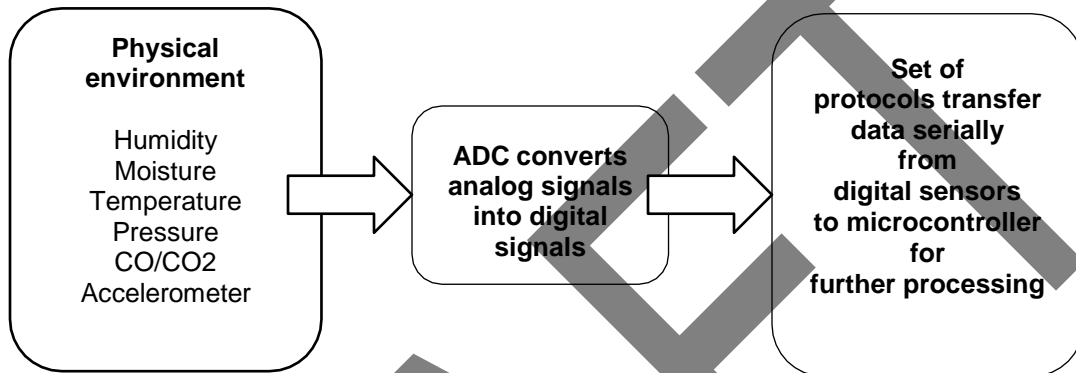
Most OEMs today design sensors that come with digital data output capability, thereby making it easier and more efficient (fewer I/O pins)to interface various sensors with the microcontroller. However, a microcontroller must have the capability to process digital data from sensors. It must provide set of interfaces to gather sensor data in order to process digital data.

With this in mind, this chapter will help readers understand Serial Communication protocols and their significance. It will cover various Serial interfaces and protocols as well as the operation of various serial protocols. Finally, it will cover the process of configuring various serial interfaces of MSP430 microcontroller and discuss digital Sensor interfacing techniques for real world applications.

Topic	Page
4.1. Introduction	160
4.2. Basics of embedded serial communication	160
4.3. Various serial communication interfaces and their comparison	163
4.4 Universal Asynchronous Receiver-Transmitter	164
4.5. I2C Serial Interface	167
4.6. SPI Interface	169
4.7. Overview of MSP430x5xx Serial communication Interfaces	170
4.8. Programming MSP430 for SPI protocol	176
4.9. Programming MSP430 for I2C protocol	182
4.10 Case Study: MSP430	192
4.11. Summary	195
4.12. Review Questions	196
4.13. Exercises	197

4.1. Introduction

In today's world of digital devices, there are several instances where data communication is required between various digital modules of the larger system. For example, in an automated gardening system, the various sensors deployed in the gardening area sense parameters such as temperature, pressure, moisture content, and precipitation level. These sensor parameters are converted into analog signals and thereafter digitized by all digital sensors.



A microcontroller provides the mechanism to process digital data received from digital sensors by means of a set of protocols and interfaces. A solid understanding of serial communication helps engineers to understand mechanisms by which data from digital sensors can be processed by microcontroller.

Digital data communication between various modules of embedded systems needs standards, interfaces and protocols that facilitate easy interconnection microcontrollers and other digital sensors. In a typical embedded system or device, microcontrollers need to communicate with external sensors and modules like humidity sensors, pressure sensors, GPS modules, GSM/GPRS modules etc. To support data communication between a microcontroller and digital sensor, microcontrollers support GPIOs as well as other communication interfaces and protocols like UART, SPI, I2C and CAN. There are two types of data communication techniques - serial communication and parallel communication. This chapter helps readers understand the intricacies of embedded serial communication protocols and its related concepts. The main aim is to introduce audiences to embedded communication concepts, real time interfacing of digital modules with microcontroller MSP430 using serial interfaces and to know the supporting communication protocols and its programming. This chapter will cover basic protocols, terminologies and help readers understand how digital devices transmit and receive data. Additionally, various digital interfaces available and factors affecting the right choice of interface; specific protocols and their operation are also covered. Finally, readers will also learn about configuring and programming various interfaces of MSP430 series microcontroller.

4.2. Basics of embedded serial communication

4.2.1. Data framing in parallel and serial communication

In digital communication, data is transferred between two digital modules in terms of data byte or a digital data frame. To transfer a digital data frame, for example 63H, between two digital devices, we need to have a communication channel between them. In serial communication protocol, data is transferred bit wise bit serially between two supporting devices, whereas parallel communication protocol transfers a complete byte (63H) at a time. Fig 4.1 illustrates the transfer of a 63H data frame in both parallel and serial communication. In a basic communication protocol, the device that transfers data is known as transmitter (Tx), whereas the device that receives the data is known as receiver (Rx).

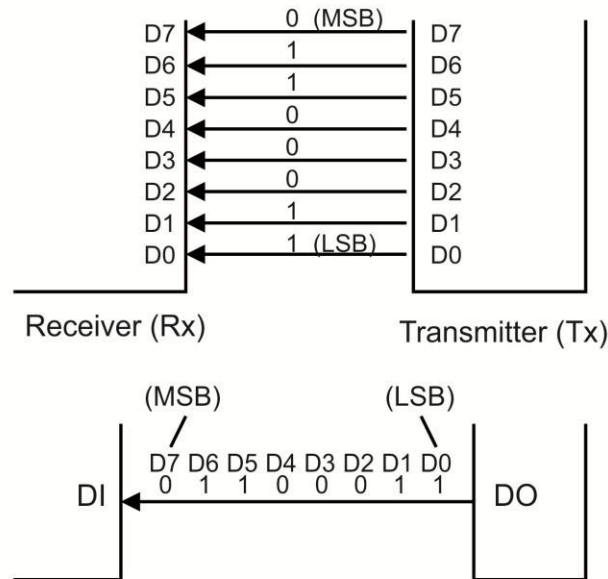


Fig 4.1 Data transfer in serial and parallel communication

4.2.2. Type of serial interfaces – Asynchronous Vs Synchronous

Serial Interface is a communication interface that transfers data as a series of voltage pulses from one device known as transmitter to other device known as receiver. Every serial interface requires proper data retrieval and hence requires a communication protocol. Every communication protocol should address following key issues:

- 1) How the receiver assumes that transmitter has sent the data.
- 2) What is the bit order (whether LSB sent or MSB sent first)?
- 3) When to look at the channel, for the availability of data?
- 4) Mechanism to know end of the transmission.

The above issues are addressed by various serial interfaces in different ways. In general, two types of serial interfaces are available- one is Asynchronous Serial Interface and other is Synchronous Serial Interface.

In Asynchronous Serial Interface, the receiver checks for the start bit and begins retrieving the data at regular intervals with prior agreement on the data transfer rate between the receiver and the transmitter. The data transfer rate between transmitter and receiver is known as baud rate. Typical data frame for RS232 asynchronous serial interface is depicted in Fig 4.2. The complete detail of the protocol and its operation is available in section 4.4.

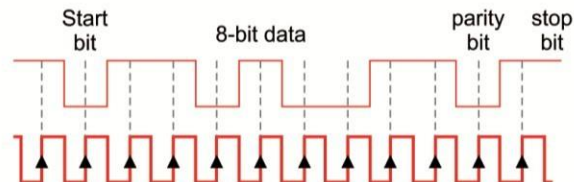


Fig 4.2 RS-232 data

synchronize their access to data line with the help of shared clock for data transfer to become successful. SPI, I2C are examples of Synchronous Serial Interfaces. Both these protocols are explained in forthcoming sections.

4.2.3. Various serial communication terminologies for data transfer

Asynchronous bus

An asynchronous bus transfers data without using an external clock.

Synchronous bus

A synchronous bus transfer data with the help of an external timing clock, which is shared between transmitter and receiver.

Full duplex/Half duplex

In full duplex communication, the data can be sent or received simultaneously, whereas in half-duplex, data can either be received or transmitted one at a time.

Master/Slave

In synchronous communication, the master is a device that provides timing signals using a clock for data to be sent or received; whereas slave is a device that responds to master.

Multi master bus

Multi-master bus is a master/slave bus that has more than one masters waiting to start communication.

Point to Point

Point-to-point or peer-to-peer communication is relevant in asynchronous serial interfaces, where two devices have a peer relation to each other i.e. no master/slave philosophy exists.

DRAFT

4.3. Various serial communication interfaces and their comparison

For embedded engineers, there are many serial interfaces available in the market. The choice of the right interface for your system depends upon several key factors such as the number of signal wires, data transfer length, data transfer rate etc. Table 4.1 shows the complete comparison of various serial communication interfaces in terms of synchronous/ asynchronous nature, type and maximum data transfer rate supported by these interfaces and protocols.

Name of interface	Synchronous /Asynchronous	Type	Duplex	Max data transfer
RS232	Asynchronous	peer-to-peer	Full-duplex	20
I2C	Synchronous	multi-master	Half-duplex	3400
SPI	Synchronous	multi-master	Full-duplex	>1,000
Microwire	Synchronous	master/slave	Full-duplex	> 625
1-wire	Asynchronous	master/slave	Half-duplex	16

Table 4.1 Comparison of various serial communication interfaces

Serial communication interfaces are preferred over parallel, one of the direct benefits of serial interfaces being a lower pin count. Choosing a particular communication interface depends upon the requirement with which microcontroller communicates with external sensors and modules. For example, if the controller is required to process data from multiple sensors, then an I2C interface with a pin count of 2 is advisable. Some applications such as atmospheric data gathering missions and environment monitoring require multiple sensors that can monitor CO, CO₂, NO₂, humidity, temperature, altitude and ambient light. These use I2C based interfaces. SPI is the choice when data rate of more than 1MBPS is required. However, it has a limitation, as it requires a dedicated slave select (SS) line to select a particular slave. For controllers that have fewer I/O pins, this interface has drawbacks as compared to I2C. RS232 interface is a choice when the controller needs to communicate with a PC or host environment. This interface is used for debugging, programming and data logging applications.

Most industry standard modules or sensors come with at least one of the above-mentioned interfaces. Hence, to transfer data from these modules or sensors, the microcontroller needs to support this set of serial communication interfaces. The most widely used serial communication interfaces are RS232, I2C and SPI. For example, the MSP430 microcontroller series supports universal, synchronous, asynchronous receive/transmit communication interface (USART) as an RS232 interface. It is also referred to as a serial-controller interface (SCI) in older versions of MSP430 series platforms, now replaced by USCI (Universal serial communication interface) module. The USCI module supports UART, SPI and I2C based communication. MSP430x5xx and MSP430x6xx series devices support USCI. Some of the MSP430 devices like MSP430x20xx support USI module support for SPI and I2C communication.

4.4. Universal Asynchronous Receiver-Transmitter

Universal Asynchronous Receiver-Transmitter (UART) is commonly known as Serial Communication Interface (SCI) and in MSP, it is known as Universal Serial Communication Interface (USCI). It is an asynchronous protocol and used mainly in interfacing microcontrollers with personal computers. This protocol is also known as RS-232 protocol. UART can be used in various applications such as GPS unit, modems etc.

4.4.1. Half and Full duplex communication

In data transmission, a duplex transmission is one in which the data can be transmitted and received. In a simplex transmission, a device is configured as transmitter or receiver. Ex: PC communicating with printers, in which the computer only sends data. Duplex transmissions are categorized as half or full duplex. When data is transmitted one way at a time; it is called as half duplex. When data transfer occurs both ways at the same time; it is referred as full duplex. Full duplex requires two data lines, one for transmission and one for reception.

4.4.2. Data Frame in UART

The receiver and transmitter should agree on a set of rules called protocol, to understand the data transmitted or received. It is difficult to sense the continuous stream of 1s and 0s. Generally protocols, define how data is packed, how many bits make a character, how data begins and ends.

Asynchronous serial data communication is used for character transmissions. Between start and stop bits, characters are packed. The start bit is always one bit but the stop bit can be one or two bits. The start bit is always logic „0“ (low) and the stop bit is logic „1“ (high).

Example:

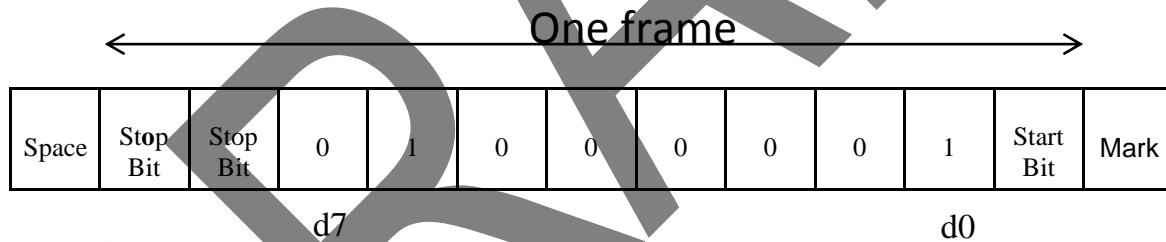


Figure 4.1: Frame for ASCII „A“ (01000001)

Figure shows the ASCII character "A", binary 0100 0001, is framed between the start bit and two stop bits. LSB will go out first. When there is no data transfer, the logic „1“ is transmitted. This is known as a „mark.“ The logic „0“ is referred to as space. Peripheral chips can be programmed to receive or transmit 5, 6, 7 or 8 bits of data. In the initial days, two stop bits were used to give sufficient time to organize the received data or to make suitable arrangements to transmit the next chunk of data. The modern day microcontrollers and personal computers use 1 stop bit. In modern UARTs, a total of 10-bits are used. 8-bits are for ASCII code, one for start bit and one for stop bit.

4.4.3. Parity Bit

A parity bit can be added into the frame in order to check integrity of data. The parity bit may be even or odd. Odd parity means odd number of 1"s in the data (parity = logic „1“). Even parity means an even number of 1"s in the data (parity = logic „0“). Ex: The binary of ASCII „A“ is 0100 0001. Parity bit is logic „0“ (Even parity). UART implementations in microcontrollers allow the programmer to decide whether to include parity or not. The parity bit is transmitted as MSB followed by a stop bit.

4.4.4. Data transfer rate

Data transfer rate in serial communication is measured in terms of bits per second (bps). This is also called as baud rate. Baud rate and bps can be used interchangeably with respect to UART.

Ex: The total number of bits that gets transferred during 10 pages of text, each with 100 × 25 characters with 8 bits per character and 1 stop bit is:

For each character, the total number of bits is 10. The total number of bits is:

$100 \times 25 \times 10 = 25,000$ bits per page. For 10 pages of data, it is required to transmit 2, 50,000 bits.

Generally baud rates of SCI are 1200, 2400, 4800, 9600, 19,200 etc. To transfer 2 50,000 bits at a baud rate of 9600, we need: $250000/9600 = 26.04$ seconds (27 seconds).

4.4.5. Serial I/O Standards

RS-232 is used widely as a serial communication standard. UART also follows RS-232 standards. RS-232 is an interfacing standard adopted by Electronics Industries Association. RS represents „Recommended Standard“ in RS-232 and this standard was adopted in 1960, before the advent of CMOS and TTL logic.

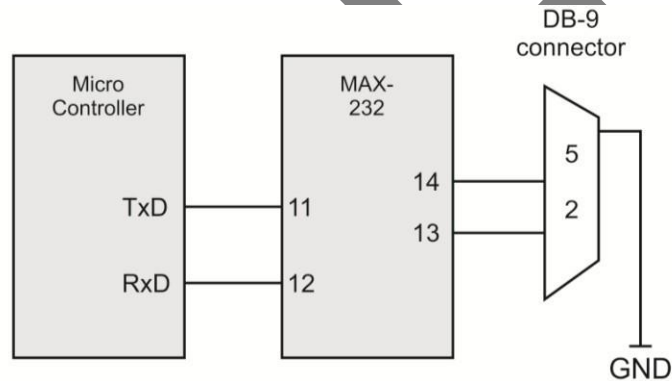


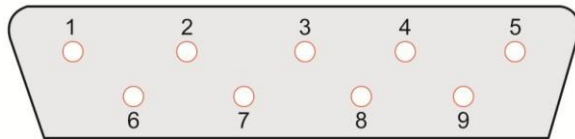
Figure 4.2: MAX 232 and microcontroller connection

So the input and output voltage levels are not compatible to TTL or CMOS. In RS-232, at the receiver, logic „1“ is represented by -3 to -25V and logic „0“ between +3 to +25V. The voltage range between -3 to +3 is undefined. To connect RS-232 to other devices which are in compliance with TTL, we need voltage converter circuits. The most commonly used voltage converter devices are MAX232 or MAX233.

It is required to use voltage converters such as MAX232 or MAX233 to convert the TTL logic levels to the RS232 voltage level and vice versa. The connections are shown in Figure 4.2.

4.4.6. Pin Map for RS232 (DB-9 connector)

The pins and their labels for the RS232 cable which is commonly known as DB-9 connector. The personal computer serial port is shown in Figure 4.3.



Pin	Description
1	Data Carrier Detect (DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal Ground (GND)
6	Data set ready (DSR)
7	Request to Send (RTS)
8	Clear to send (CTS)
9	Ring Indicator (RI)

Table 4.1: RS-232 pin description

The handshaking mechanism is important to ensure fast and reliable data transfer between the two devices. Few signals of RS-232 are used for handshaking.

DCD (Data carrier Detect)

This signal is also known as CD (Carrier Detect). The microcontroller asserts DCD to inform PC that a valid carrier has been detected and connection is successfully established. DCD is an output from microcontroller and input to PC.

DTR (Data Terminal Ready)

After self-test operation, DTR signal will inform microcontroller that it is ready for communication. DTR is output of PC COM port. The DTR is an active low signal and won't get activated when there is an error in COM port.

DSR (Data Set Ready)

When microcontroller is turned on, it will undergo self-test for UART and asserts DSR to signal PC COM port, that it is ready. This is an active low signal.

RTS (Request To Send)

When a PC COM port wants to transmit the byte of data to microcontroller, it will signal the microcontroller by asserting the RTS signal. RTS is an active low signal.

CTS (Clear To Send)

Microcontroller will respond to RTS signal by asserting the CTS line, indicating that it is ready to receive data. This signal is an output of the microcontroller and an input to the PC COM port. PC will start transmission once it receives CTS.

RI (Ring Indicator)

This signal is a handshaking signal, which is not often used. Microcontroller can send RI to PC, so that the telephone will ring and PC will receive it.

Note on Serial Communication in embedded systems:

There are many ways to encode the binary information during serial communication. The main aim of using different techniques is to maximize the bandwidth and minimize errors.

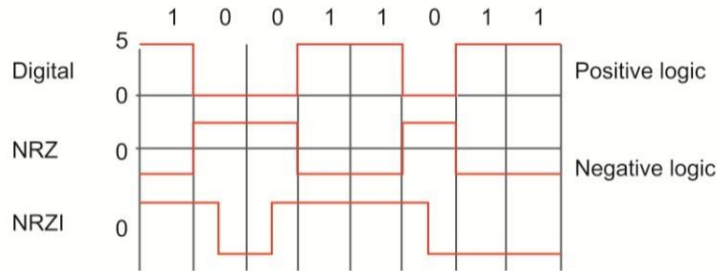


Figure 4.4: Different encoding schemes for serial communication

NRZ (Non-Return to Zero) is one type of encoding in which the signal never goes to zero voltage. To maintain high energy on wires, this coding is used to enable data to be transmitted for long distances. Voltages are measured with respect to ground in single wire (Ex: RS-232) and in twisted pairs, voltages are differential (Ex: RS-422, USB). Binary values are encoded as positive or negative voltages. Logic „1“ will have positive voltage and logic „0“ will have negative voltage in positive logic. In negative logic, it is vice-versa.

Non-return to zero inverted (NRZI) is encoding method, in which binary signal is represented as transitions in the signal. Binary information is encoded as the presence or absence of transition at the clock boundary.

The interface logic chips like MAX232 will convert voltages between TTL/MOS/CMOS logic levels.

4.5. I2C Serial Interface

I2C or inter-IC bus interface is a two wire (SCL, SDA) interface used by various IC manufacturers for transmitting data asynchronously between microcontroller and external sensors, modules using serial communication. The typical application of I2C is shown in Fig 4.5. Each I2C slave device has a unique address and is connected to master, which provides clock signal (SCL). Data is transmitted serially using SDA signal line. Each slave device is recognized by a 7-bit or 10-bit address. Hence multiple slave devices can be connected through an I2C bus. The SCL and SDA lines have to be pulled-up using registers R_p , the value of which depends upon the voltage V_{dd} .

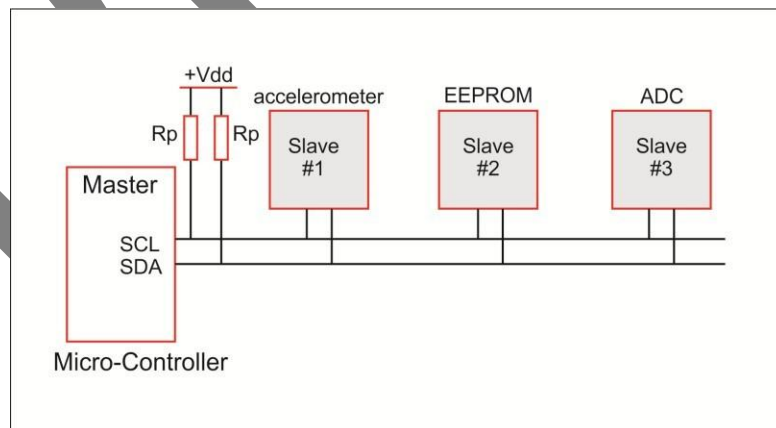


Fig 4.5 I2C interfacing with master/slave philosophy

I2C interface supports three data transfer modes: standard mode, fast mode and high speed mode with data rates of 100kbps, 400kbps and 3.4Mbps respectively.

4.5.1. Typical operation of I2C bus

In I2C protocol, I2C master starts command or operation with a START condition and ends with STOP condition as shown in Fig 4.5. START and STOP condition is differentiated by SDA line as for both conditions SCL must be high. For START condition

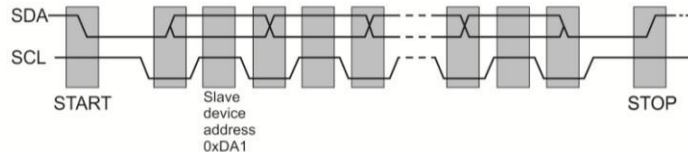
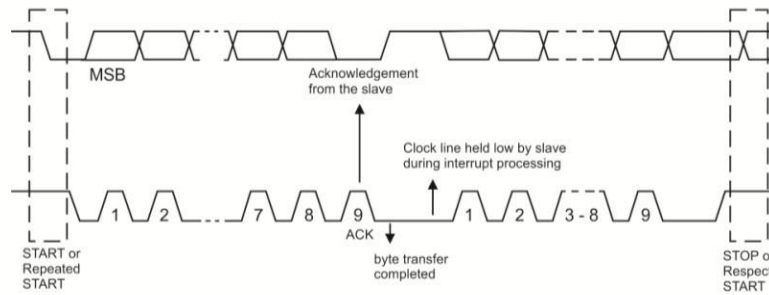


Fig 4.6. START and STOP conditions in I2C bus

SDA takes the transition from high to low, whereas for STOP condition, SDA takes transition from low to high. Before the start of operation both SDA and SCL lines are initially high. After the START condition by a particular master, the bus becomes busy and hence no other master can use the bus before the STOP condition. START and STOP conditions are generated by the master device.

4.5.2. Data transfer in I2C



START	Slave Address	R/W bit	ACK	DATA	ACK	DATA	ACK	STOP
1-bit	7-bits	1-bit	1-bit	8-bit	1-bit	8-bit	1-bit	1-bit

Fig 4.7 Data frame format in I2C communication

In the I2C protocol, master is a device that initiates data transfer on the bus and generates the clock signal. One clock bus is generated for each data bit transferred. HIGH to LOW transition on SDA line is allowed only when the clock line (SCL) is low. 8-bit data is transferred on the data bus through SDA line. Data is transferred with the most significant bit (MSB) first. Master sends the START condition followed by a 7-bit device address and data direction bit or R/W bit as shown in Fig 4.6. If R/W bit is 0, master will write to the slave device and if R/W bit is 1, master will read from the slave device. After setting R/W bit master will continue reading or writing. After read or write operation, communication is ended with the STOP condition. If master wants to communicate with other slaves, it continues with repeated START condition followed by slave address without generating STOP condition. Important points to be noted in I2C communication:

- Master can initiate communication with another slave device after reading or writing first slave device, with the help of repeated START condition.
- Master can read the slave device by setting R/W bit to 1 after sending device address.
- If a slave cannot receive or transmit data, it can hold the clock line (SCL) LOW to force the master into a wait state. After successful data transfer, the slave can release the clock line.

START	Slave Address	R/W bit	ACK	DATA	ACK	DATA	ACK	STOP
1-bit	7-bits	1-bit	1-bit	8-bits	1-bit	8-bits	1-bit	1-bit

master writes data to slave if R/W bit is '0'

START	Slave Address	R/W bit	ACK	DATA	ACK	DATA	ACK	STOP
1-bit	7-bits	1-bit	1-bit	8-bits	1-bit	8-bits	1-bit	1-bit

master reads data from slave if R/W bit is '1'



Fig 4.8 writing and reading data in I2C bus transaction

4.6. SPI Interface

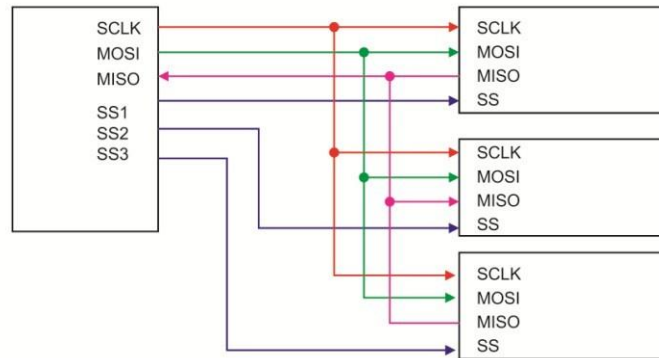


Fig 4.9 Interconnection between SPI devices

SPI stands for serial peripheral interface. It is a type of synchronous serial interface developed by Motorola. It uses separate clock and data lines along with device/slave select line. In SPI communication, there is at least one slave select (SS) line. It requires more slave select lines to connect to multiple slaves. SPI bus can be configured for two bus topologies.

- **Point-to-point topology** where master is connected to single slave device.
- **Multi-point topology** where master is connected to multiple slave devices.

In order to transfer data between master and slave, SPI protocol consists of at least four signal lines: SCL (clock signal), MOSI (Master out slave in), MISO (master in slave out) and SS (slave select).

4.6.1. Operation of SPI bus

SPI is a master-slave, full duplex protocol. Master and Slave devices consist of at least one Shift register to transfer data serially. MOSI pin is used to transfer data out from the master device or to receive data when the device is configured as a slave device. MISO pin is used to transmit data out of

the SPI device when it is configured as slave and to receive data when configured as master. Both master and slave devices are connected together by means of shift registers as shown in Fig 4.10

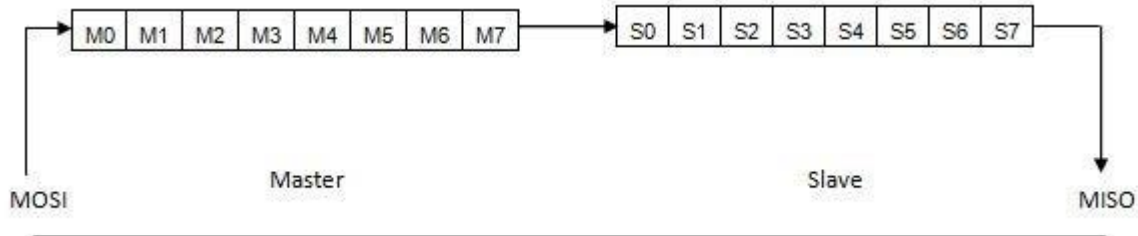


Fig 4.10 Master slave operation

As soon as the master provides the clock, data in the shift registers starts shifting towards the right. For the first clock pulse, M7 is shifted to S0, M6 is shifted to M7 and so on. Right shift also happens in the slave device register. In this process last bit, S7 shifted to M0. After 8 clock pulses, the entire data is transferred from master register to slave register and slave data is transferred to master register.

4.7. Overview of MSP430x5xx Serial communication Interfaces

In general, MSP430 series controller supports three serial communication modules viz. USART, USCI and USI. The MSP430x5xx device supports Universal Serial Communication Interface (USCI) module. The number of USCI modules depends upon the device series. In general, USCI_Ax and USCI_Bx modules are supported by device where x represents a decimal number starting from 0. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. USCI_Ax is an asynchronous channel that supports UART and SPI, whereas USCI_Bx is a synchronous channel that supports I2C and SPI. For SPI communication, both USCI_Ax as well as USCI_Bx is used. The USCI module has various modes as shown below:

USCI_Ax modules support:

UART mode

Pulse shaping for IrDA communications

Automatic baud-rate detection for LIN communications

SPI mode

USCI_Bx modules support:

I2C mode

SPI mode

4.7.1. Programming MSP430 UART for serial communication

MSP430 Serial USCI_A module

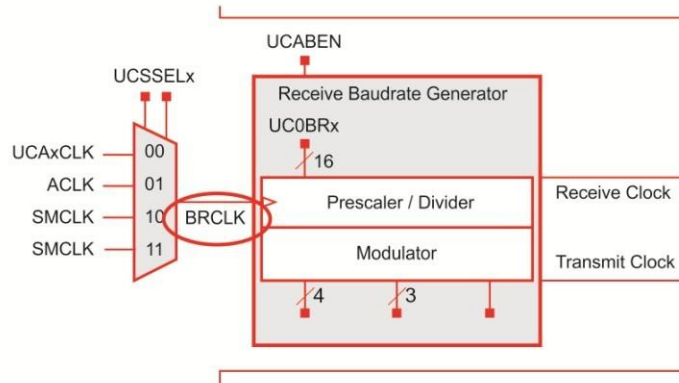


Fig 4.11 USCI module block diagram

For UART implementation, MSP430 uses USCI_Ax module. When configured in UART mode, USCI module transmits and receives bits as per the specific baud rate that depends upon configuration options. In UART mode, in order to interface external device / sensor UCxAxRx and UCxAxTx can be used. In an MSP430x5xx device, UCA0Rx is pin P3.4 whereas UCA0Tx pin is P3.3. The USCI_A module works in asynchronous mode or UART mode only when the UCSYNC bit is clear.

4.7.2. Configuring MSP430x5xx for UART operation

Configuration of MSP430 for UART operation means to write 0 or 1 bits to configuration registers of USCI module. USCI module has various 8-bit configuration registers to accomplish initialization and configuring USCI in UART mode. The steps for configuration are:

- Set UCSWRST; disable UART module by setting the BIT**
- Initialize all USCI registers with UCSWRST = 1 (including UCxAxCTL1).**
- Configure ports.**
- Clear UCSWRST; enable UART module for operation by clearing the BIT**
- Enable interrupts (optional) via UCRXIE and/or UCTXIE.**

The MSP430x5xx series device USCI_Ax module consists of two control registers for UART configuration as shown below:

- UCxAxCTL0 Register abbreviated as USCI_AxControlRegister0**
- UCxAxCTL1 Register abbreviated as USCI_AxControlRegister1**

		7	6	5	4	3	2	1	0						
		UCPEN		UCPAR		UCMSB		UC7BIT		UCSPB		UCMODEx		UCSYNC	
Bit	Field	Type	Reset	Description											
7	UCPEN	RW	0h	Parity enable 0b = Parity disabled 1b = Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the											
6	UCPAR	RW	0h	Parity select. UCPAR is not used when parity is disabled. 0b = Odd parity											
5	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first; 1b = MSB first											
4	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data; 1b = 7-bit data;											
3	UCSPB	RW	0h	Stop bit select. Number of stop bits. 0b = One stop bit; 1b = Two stop bits											
2-1	UCMODEx	RW	0h	USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0. 00b = UART mode 01b = Idle-line multiprocessor mode 10b = Address-bit multiprocessor mode 11b = UART mode with automatic baud-rate detection											
0	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode											

Table 4.1 UCAxCTL0 Register

The UART mode of the USCIAx module can be enabled by setting the UCSYNC bit of the UCAxCTL0 Register. Prior to the setting, we need to enable the UCSWRST bit of UCAxCTL1 Register.

UCA0CTL1 |= UCSWRST;

UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Bit	Field	Type	Reset	Description			
7-6	UCSSELx	RW	0h	USCI clock source select. These bits select the BRCLK source clock. 00b = UCAXCLK (external USCI clock) 01b = ACLK 10b = SMCLK 11b = SMCLK			
5	UCRXEIE	RW	0h	Receive erroneous-character interrupt enable 0b = Erroneous characters rejected and UCRXIFG is not set. 1b = Erroneous characters received set UCRXIFG.			
4	UCBRKIE	RW	0h	Receive break character interrupt enable 0b = Received break characters do not set UCRXIFG. 1b = Received break characters set UCRXIFG.			
3	UCDORM	RW	0h	Dormant. Puts USCI into sleep mode. 0b = Not dormant. All received characters set UCRXIFG. 1b = Dormant.			
2	UCTXADDR	RW	0h	Transmit address. Next frame to be transmitted is marked as address, depending on the selected multiprocessor mode. 0b = Next frame transmitted is data. 1b = Next frame transmitted is an address.			
1	UCTXBRK	RW	0h	Transmit break. Transmits a break with the next write to the transmit buffer. 0b = Next frame transmitted is not a break. 1b = Next frame transmitted is a break or a break/synch.			
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.			

Table 4.2 UCAXCTL1 Register

The USCI module registers need to be configured as:

UCA0CTL0: USCI_A0 control register 0, you need to configure the register in this manner:

No parity, LSB first, 8-bit data, 1 stop bit, UART, Asynchronous.

UCA0CTL1: USCI_A0 control register 1, use SMCLK as USCI clock source and enable software interrupts.

UCA0CTL1 |= UCSSEL_2

UCA0BR0 & UCA0BR1: USCI_A0 Baud rate control register 0 and 1, configure these 16-bit register to have 1MHz frequency and desired baud-rate (see section 4.7.3)

UCA0MCTL: USCI_A0 modulation control register, configure this register on this manner: second Stage modulation = 1, Oversampling off.

IFG2: Interrupt Flag Register 2, clear the flags.

IE2: Interrupt Enable Register 2, Enable it.

The next step is to configure relevant pins of port P3 of the MSP430 to work as Rx and Tx. As mentioned earlier, pin P3.4 works as UCA0RxD whereas pin P3.3 works as UCA0TxD. Each port of MSP430 has 8-bit configuration registers that control the functionality of the port pins. P3.4 and P3.3 work as Rx and Tx pins if P3SEL register is configured such that BIT3 and BIT4 of the P3SEL register are set to HIGH. This can be done in programming using below code:

```
P3SEL |= BIT3 | BIT4
```

```
P3SEL2 |= BIT3 | BIT4
```

4.7.3. Baud rate configuration

BRCLK is the main clock source for generating baud rate for UART module. The external clock UCAxCLK, or the internal clocks ACLK or SMCLK depending on the UCSSELx settings can source the clock frequency for BRCLK (Refer UCAxCTL1 Register). For a given BRCLK, the baud rate can be calculated if division factor N is known.

$$N = \text{BRCLK} / \text{Baudrate}$$

The UART module supports two baud-rate generation mechanisms:

- Low frequency baud-rate generation
- Oversampling baud-rate generation

Low frequency baud-rate generation is achieved when UCOS16=0. Generating baud rates at low frequency reduces the power consumption of the USCI module. To generate a specific baud rate, the values of baud rate prescaler register UCBRx and the fractional portion second stage modulator UCBRSx need to be calculated as shown below:

$$\text{UCBRx} = \text{Integer}(N)$$

$$\text{UCBRSx} = \text{Round}((N - \text{Integer}(N)) * 8)$$

For example, for 1MHz BRCLK frequency, to set 9600-baud rate the value of N is 104.16666667.

$$\text{UCBRx} = \text{Integer}(N) = 104$$

$$\text{UCBRSx} = \text{round}((104.16666667 - 104) * 8) = \text{round}(1.33333333) = 1$$

Oversampling baud-rate generation is achieved when UCOS16=1. For specific baud rate generation at higher frequencies the prescaler register and first stage modulator UCBRFx can be calculated as:

$$\text{UCBRx} = \text{Integer}(N/16)$$

$$\text{UCBRFx} = \text{round}(((N/16) - \text{Integer}(N/16)) * 16)$$

For example, for higher frequency of 4MHz to set baud rate of 9600 the values are:

$$\text{UCBRx} = 26 \text{ and } \text{UCBRFx} = 1$$

For UCAx, module values of UCBRx can be represented by two 8-bit registers - UCA0BR0 which contains LSBs and UCA0BR1 which contains MSBs. In case of 1MHz BRCLK frequency, to set 9600-baud rate the value of N is 104. This value is less than decimal 255. The code to set baud rate can be written as:

```
UCA0BR0 = 104; // 1MHz/9600 = 104.166
UCA0BR1 = 0x00;
```

UCBRx				
Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Lowbyte (LSB) of clock prescaler setting of the baud-rate generator.

UCAxBR0 Register

UCBRx				
Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	High byte (MSB) of clock prescaler setting of the baud-rate generator.

UCAxBR1 Register

Table 4.3 UCBRx Register

USCI_Ax Modulation Control Register UCA0MCTL sets the modulation for first and second stage modulation depending upon the modulation mode. To set the first stage modulation following line of code can be used:

4.7.4. Pseudo program to transfer data serially using UART module

In the previous sections, we have discussed the various configuration registers of USCI module to support serial communication through UART. MSP430 is a low power device that supports software programmable low power modes. Interrupt mechanism can be used to wake up the MSP430 from low power mode. To fully utilize the low power capability of the device, as interrupt related to USCI module is generated, then ISR related to USCI module interrupt can process a data transmission request. In order to achieve this, transmission of the buffered data can be carried out by ISR related to USCI0TX interrupt. Suppose the data to be transferred is in the buffer char *data_Tx., then MSP430 is configured in Low power mode LPM1. The algorithm to transfer data serial through interrupt can be represented as:

Main program code:

Step 1: Configure watchdog timer

Step 2: Configure port P3 specific pins to work as UART

Step 3: Set clock source for BRCLK

Step 4: Configure baud rate of USCI module for UART

Step 5: Set modulation by configuring modulation control register UCAxMCTL depending upon UCS016 bit

Step 6: Enable Tx interrupt of USCI_Ax module

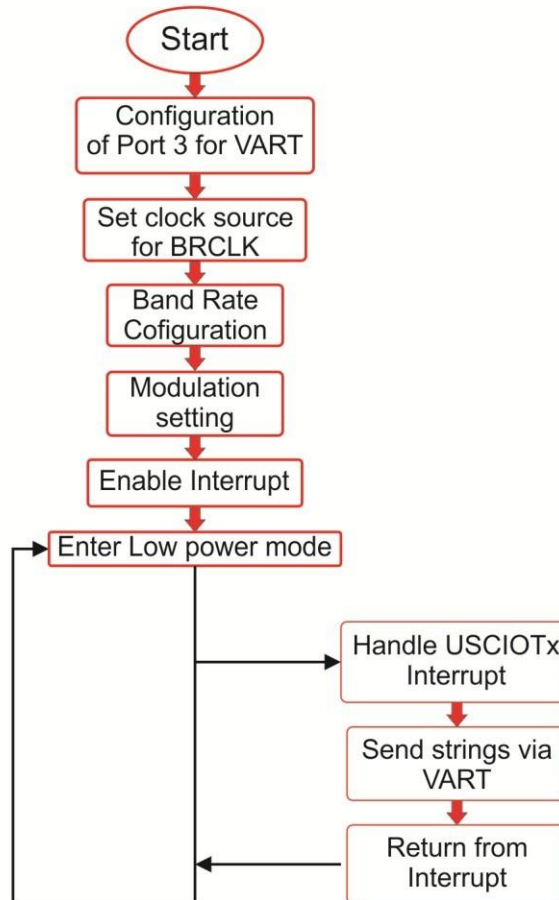
Step 7: Enable Low power mode LPM1 with interrupt enable.

Whenever data is ready for transmission in the UART buffer UCA0TXBUF the data is transmitted serially by ISR related to USCI0TX interrupt. The algorithm related to ISR can be represented as:

ISR_routine:

Step 1: Initialize the interrupt address

Step 2: Write ISR related to UART interrupt. The role is to monitor the interrupt flag IFG2 and transmit buffer flag UCA0TXIFG and put the data into buffer UCA0TXBUF for transmission, if buffer is empty.



4.8. Programming MSP430 for SPI protocol

4.8.1. Overview of SPI communication using MSP430

In synchronous mode, the US module connects the device to an external system via three or four pins:

UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. SPI mode is selected when the UCSYNC bit is set, and SPI mode (3-pin or 4-pin) is selected with the UCMODEx bits. For MSP430 series devices the four pins are mapped as per the table given below:

SPI pins	MSP430x55xx SPI pin	Description
MOSI	UCxSIMO	Master mode: UCxSIMO is the data output line. Slave mode: UCxSIMO is the data input line.
MISO	UCxSOMI	Master mode: UCxSOMI is the data input line. Slave mode: UCxSOMI is the data output line.
SCLK	UCxCLK	Master mode: UCxCLK is an output. Slave mode: UCxCLK is an input.
SS	UCxSTE	Used in 4-pin mode to allow multiple masters on a single bus.

Table 4.5 Pin mapping between SPI and USCI module of MSP430

In MSP430x55xx series devices, P3.4 (UCA0SOMI) and P3.3 (UCA0SIMO) are multiplexed and can be configured to be used in SPI communication using UCSI_A0 module; whereas P3.0 (UCB0SIMO) and P3.1 (UCB0SOMI) can be configured to be used for SPI communication using USCI_B0 module. For USCI_A0 module P2.7 port pin represents UCA0CLK pin and for USCI_B0 module P3.2 represents UCB0CLK pin.

4.8.2. Configuring SPI interface of MSP430

Configuration of MSP430 for SPI operation means to write 0 or 1 bits to configuration registers of USCI module. USCI module has various 8-bit configuration registers to accomplish initialization and configuring USCI in SPI mode.

The USCI is reset by a PU Cor by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in are set condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCOE, and UCFE bits, and sets the UCTXIFG flag. Clearing UCSWRST releases the USCI for operation.

The steps for configuration are:

Set UCSWRST bit

Initialize all USCI registers with UCSWRST = 1 (including UCAXCTL1).

Configure ports.

Clear UCSWRST; enable SPI module for operation by clearing the BIT

Enable interrupts (optional) via UCRXIE and/or UCTXIE.

SPI is a synchronous interface; USCI can be configured in SPI mode, only when UCSYNC bit of UCAXCTL0Register is set. The complete description of other bits of UCAXCTL0 register is shown below:

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMOD		UCSYNC
Bit	Field	Reset	Description				
7	UCCKPH	0h	Clock phase select 0b=Data is changed on the first UCLK edge and capture don the following edge. 1b=Data is captured on the first UCLK edge and changed on the following edge.				
6	UCCKPL	0h	Clock polarity select 0b=The inactive state is low; 1b=The inactive state is high				
5	UCMSB	0h	MSBfirstselect. Controls the direction of the receive and transmit shift register. 0b=LSBfirst; 1b=MSBfirst				
4	UC7BIT	0h	Character length. Selects 7-bit or 8-bit character length. 0b=8-bit data; 1b=7-bit data				
3	UCMST	0h	Master mode select 0b=Slave mode; 1b=Master mode				
2-1	UCMODEx	0h	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC=1. 00b=3-pin SPI 01b=4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE=1 10b=4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE=0 11b=I2C mode				
0	UCSYNC	0h	Synchronous mode enable 0b=Asynchronous mode; 1b=Synchronous mode				

Table 4.6 UCxCTL0 register

The pin UCSWRST is a part of UCxCTL1 register whose description is shown below:

Bit	Field	Reset	Description
7-6	UCSSELx	0h	USCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b=Reserved 01b=ACLK 10b=SMCLK 11b=SMCLK
5-1	Reserved	0h	Reserved. Always write as 0.
0	UCSWRST	1h	Software reset enable 0b=Disabled. USCI reset released for operation. 1b=Enabled. USCI logic held in reset state.

Table 4.7 USCAxCTL1 register

The first step of setting UCSWRST pin that can be achieved in programming as:

UCA0CLT1= UCSWRST;

Next step is to multiplex the relevant port pins. In case of USCI_A0 module, it can be achieved as (see section 4.8.1 for pin details):

P3SEL=BIT3 | BIT4 | BIT7;

P3SEL2=BIT3 | BIT4 | BIT7;

The master on the SPI bus provides UCA0CLK. When UCMST=1, the bit clock is provided by the USCI bit clock generator on the UCA0CLK pin. The clock used to generate the bit clock is selected with the UCSSELx bits. In order to generate bit clock using system clock (SMCLK) for USCI module:

UCA0CTL0= UCSSEL_2;

The bit clock frequency is given by

$f_{\text{bitclock}} = f_{\text{BRCLK}} / \text{UCBRx}$, where BRCLK is equal to SMCLK.

UCBRx register can be configured for even and odd decimal values depending upon the required bitclock frequency. The modulation field of modulation register UCA0MCTL remains empty. This can be achieved as:

UCA0MCTL = 0;

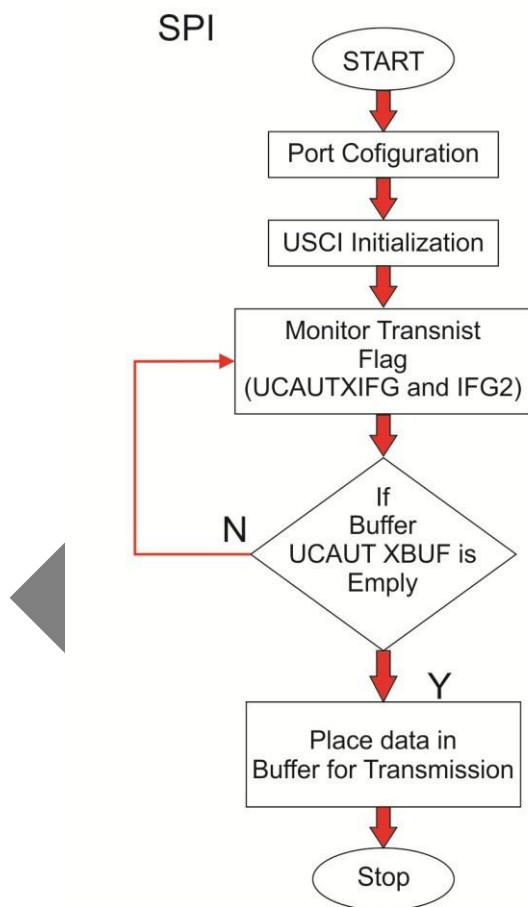
The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the USCI.

DRAFT

4.8.3. Receiving and transmitting data using SPI protocols

UCA0CLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCA0TXBUF and moved to the TXshift register before the start of UCA0CLK is transmitted on UCA0SOMI. Data on UCA0SIMO is shifted into the receive shift register on the opposite edge of UCA0CLK and moved to UCA0RXBUF when the set number of bits are received. When data is moved from the RX shift register to UCA0RXBUF, the UCRXIFG interrupt flag is set, indicating that data has been received. The over run error bit UCOE is set when the previously received data is not read from UCA0RXBUF before new data is moved to UCA0RXBUF. In general, the UCTXIFG interrupt flag is set by the transmitter to indicate that UCxTXBUF is ready to accept another character. UCTXIFG is automatically reset if a character is written to UCxTXBUF. UCTXIFG is set after a PUC or when UCSWRST=1. In order to transmit data the procedure is:

- Step 1: configure ports and initialize USCI module (as explained in section 4.6.2)**
- Step 2: Continuously monitor transmit flag UCA0TXIFG and IFG2.**
- Step 3: If buffer is empty, place data to be transmitted in the buffer UCA0TXBUF.**

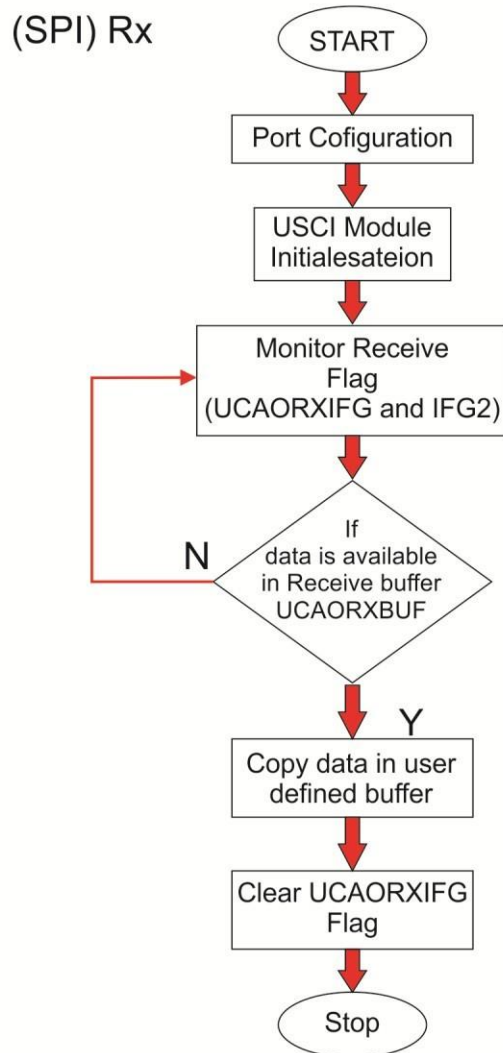


The UCRXIFG interrupt flag is set each time a character is received and loaded into UCxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG and UCRXIE are reset by a system reset PUC signal or when UCSWRST=1. UCRXIFG is automatically reset when UCxRXBUF is read. In order to receive data from slave device the procedure is:

Step 1: configure ports and initialize USCI module.

Step 2: continuously monitor receive flag UCA0RXIFG and IFG2.

Step 3: If data is available in receive buffer UCA0RXBUF, remove the data from the buffer by storing it in a user-defined buffer.



4.9. Programming MSP430 for I2C protocol

4.9.1. Overview of I2C communication using MSP430

In I²C communication, a device connected to the I2C bus can be considered as the master or the slave when performing data transfers. A master initiates a data transfer and generates the clock signal SCL.

Any device addressed by a master is considered a slave. I²C data is communicated using the serial data (SDA) pin and the serial clock (SCL) pin. Both SDA and SCL are bidirectional and must be connected to a positive supply voltage using a pullup resistor as shown in Fig 4.12. In MSP430 series devices USCI_B0 and USCI_B1 modules are used for I2C device communication. The SDA and SCL lines are mapped with port P3 and P4 as shown below:

UCB1SDA -> P4.2

UCB1SCL -> P4.1

UCB0SDA -> P3.0

UCB0SCL -> P3.1

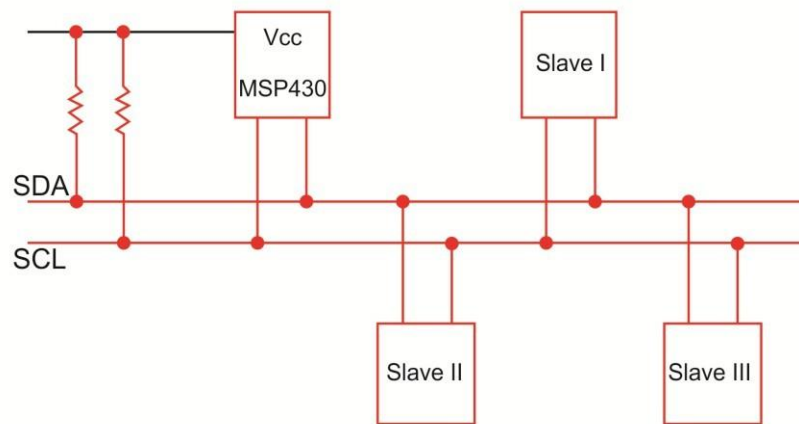


Fig 4.12 I2C bus connection diagram

4.9.2. Configuring I2C Interface of MSP430:

To start I2C communication, UCMODEx bits must be set to 11. After module initialization by setting UCSWRST bit or after PUC, USCI module is ready for transmit or receive operation. Setting UCSWRST in I²C mode has the following effects:

- I²C communication stops.
- SDA and SCL are high impedance.
- UCBxI2CSTAT, bits 6–0 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.

In order to initialize USCI module the steps are:

Set UCSWRST

Initialize USCI registers

Configure ports

Clear UCSWRST**Enable interrupts (optional)**

In most of the applications, the MSP430 is configured as a master and external sensor module acts as a slave device. In order for MSP430 to work as master, USCI_Bx module need to be configured as a master and provide clock to the slave device. The USCI_Bx module is configured as an I²C master by selecting the I²C mode with UCMODEx = 11 and UCSYNC = 1 and setting the UCMST bit. By setting UCSNYC=1 we ensure that USCI_Bx module is in synchronous mode. The role of the UCMST bit is to select whether USCI module will act as a slave or master. When UCMST bit is clear, the USCI module will act as slave, whereas when UCMST bit is set, it acts as a master. This can be done in programming as:

```
UCB1CTL1 = UCSWRST;
```

```
UCB1CTL0 = UCMST + UCMODE_3 + UCSYNC;
```

Prior to this we need to set UCSWRST bit to enable configuration of registers. Next we need to assign SDA and SCL pins to UCSI_Bx (in this example USCI_B1 module).

```
P4SEL |= BIT2 + BIT1;
```

```
P4SEL2 |= BIT2 + BIT1;
```

Before clearing the UCSWRST pin, we need to select clock source and pre-scale divider register value. This can be done in coding using the following lines:

```
UCB1CTL1 = USSEL_2; // Selecting SMCLK for BRCLK
```

```
UCB1BR0= 0x02; // pre-scaler divider value
```

```
UCB1BR1= 0;
```

After initialization is done, master transmitter mode is initiated by writing the desired slave address to the UCB1I2CSA register, selecting the size of the slave address with the UCCLA10 bit, setting UCTR for transmitter mode, and setting UCTXSTT to generate a START condition. Before sending START condition, we need to make sure the module has done with previous transmission by checking STOP condition. The various bit fields are part of Control registers:

USCI_Bx Control Register 0 (UCBxCTL0)

USCI_Bx control Register 1 (UCBxCTL1)

7	6	5	4	3	2	1	0
UCA10	UCSLA10	UCMM	Unused	UCMST	UCMODEx=11		UCSYNC=1
Bit	Field	Reset	Description				
7	UCA10	0h	Own addressing mode select 0b = Own address is a 7-bit address.				
6	UCSLA10	0h	Slave addressing mode select 0b = Address slave with 7-bit address				
5	UCMM	0h	Multi-master environment select 0b = Single master environment. There is no other master in the system. The address compare unit is disabled.				
4	Unused	0h					
3	UCMST	0h	Master mode select. When a master loses arbitration in a multi-master environment (UCMM = 1), the UCMST bit is automatically cleared and the module acts as slave.				
2-1	UCMODEx	0h	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I2C mode				
0	UCSYNC	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode				

Table 4.6 USCI Bx Control Register 0 (UCBxCTL0)

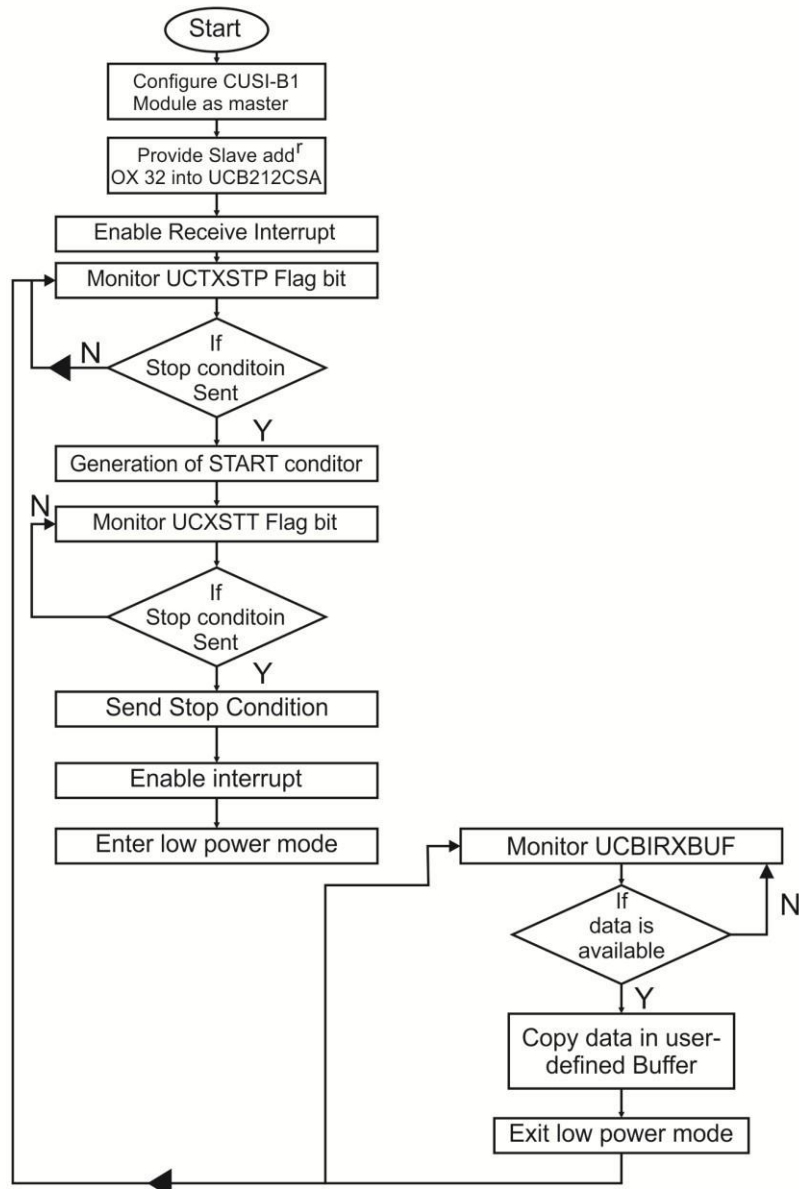
UCSSELx		Unused	UCTR	UCTXNACK	UCTXSTP	UCTXSTT	UCSWRST
Bit	Field	Reset	Description				
7-6	UCSSEL	0h	USCI clock source select. These bits select the BRCLK source clock. 00 = UCLKI 01 = ACLK 10 = SMCLK				
5	Unused	0h	Unused				
4	UCMST	0h	Transmitter/receiver 0b = Receiver; 1b = Transmitter				
3	UCTXNACK	0h	Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. 0b = Acknowledge normally 1b = Generate NACK				
2	UCTXSTP		Transmit STOP condition in master mode. Ignored in slave mode. In master receiver mode, the STOP condition is preceded by a NACK. UCTXSTP is automatically cleared after STOP is generated. 0b = No STOP generated 1b = Generate STOP				
1	UCTXSTT		Transmit START condition in master mode. Ignored in slave mode. In master receiver mode, a repeated START condition is preceded by a NACK. UCTXSTT is automatically cleared after START condition and address information is transmitted. Ignored in slave mode. 0b = Do not generate START condition 1b = Generate START condition				
0	UCSWRST		Software reset enable 0b = Disabled. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.				

Table 4.7 USCI_Bx control Register 1 (UCBxCTL1)

4.9.3. Receiving single byte from slave having address 0x32H

USCI_Bx has two buffers USCI_Bx Receive Buffer Register (UCBxRXBUF) and USCI_Bx transmit buffer register (UCBxTXBUF), one for receiving data and other one for transmitting data respectively. After initialization, master transmitter mode is initiated by writing the desired slave address to the UCBxI2CSA register, selecting the size of the slave address with the UCCLA10 bit, setting UCTR for transmitter mode, and setting UCTXSTT to generate a START condition.

In order to receive a single byte from the slave device 0x32H the steps are:

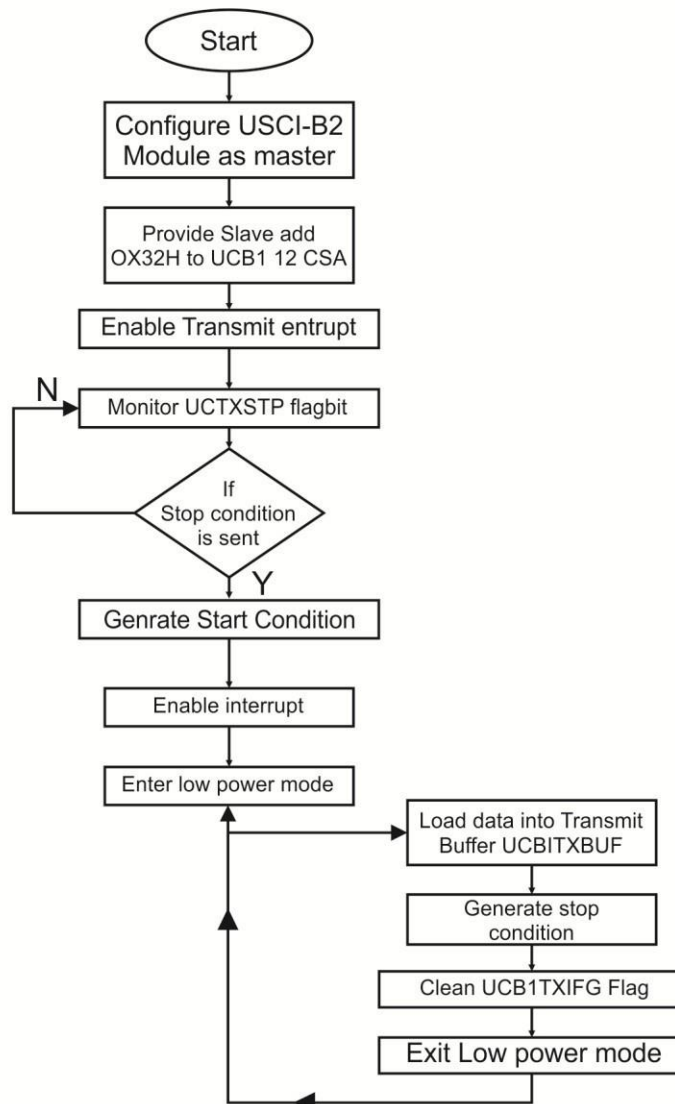

Main program:

- Step 1: Configure USCI_B1 module as a master;
- Step 2: Put slave address 0x32H into UCB1I2CSA;
- Step 3: Enable receive interrupt;
- Step 4: Check for the STOP condition, whether it is sent or not by monitoring UCTXSTP flag bit;
- Step 5: Generate START condition;
- Step 6: Wait until START condition sent by monitoring UCXSTT flag bit;
- Step 7: Send STOP condition;
- Step 8: Enter Low power mode with interrupt enable and repeat step 4 upto step 8 repeatedly.

ISR_routine:

- Copy data from UCB1RXBUF to user defined buffer, if data is available
- Exit Low Power mode;

4.9.4. Transmitting single byte to slave having address 0x32H



Main program:

Step 1: Configure USCI_B1 module as a master;

Step 2: Put slave address 0x32H into UCB1I2CSA;

Step 3: Enable transmit interrupt;

Step 4: Check for the STOP condition, whether it is sent or not by monitoring UCTXSTP flag bit ;

Step 5: Generate START condition;

Step 6: Enter Low power mode with interrupt enable and remains there.

ISR_routine:

Load data into transmit buffer UCB1TXBUF;

Generate stop condition;

Clear transmit flag UCB1TXIFG;

Exit from low power mode;

4.9.5. Real world interfacing of accelerometer using I2C Interface

In the previous section, we have discussed general techniques to configure the various registers and flags of USCI module for I2C communication. We have also discussed basic algorithms of transferring or receiving data bytes from hypothetical slave devices. In I2C based transaction, every slave has a device address. The external device or module must support I2C module to facilitate communication with USCI module of the MSP430 device. For example, MPU6050 is a sensor module that supports accelerometer, Gyro meter and in-built temperature sensor and communicates data through I2C bus. The MPU6050 has a device address register, some configuration registers and 14 data registers. The analog voltage corresponding to particular sensors is digitized by 16-bit ADC and stored in data registers. The device address of MPU6050 is 68H. The main features of MPU6050 are

- Three-axis MEMS rate gyroscope sensor with 16-bit ADCs and signal conditioning
- Three-axis MEMS accelerometer sensor with 16-bit ADCs and signal conditioning
- Digital Motion Processor (DMP) engine
- Primary I2C and SPI (MPU-6000 only) serial communications interfaces
- Auxiliary I2C serial interface for 3rd party magnetometer & other sensors
- Clocking
- Sensor Data Registers
- FIFO
- Interrupts
- Digital-Output Temperature Sensor

The pin-to-pin mapping of MSP430 and MPU6050 is given in the table 4.8

MPU6050 pin	MSP430 pin
VDD	3.3 V
GND	GND
INT	PIN 2
SYNC	X
SCL	P4.1 for USCI_B1 SCL line
SDA	P4.2 for USCI_B1 SDA line
VIO	3.3 V
CLK	X
ASCL	X
ASDA	X

Table 4.8 Pin interconnection details of MPU6050 with MSP430

In order to process data, MSP430 must be configured as master. In order to explain the complete algorithm to process sensor data, the overall software application must be divided into smaller modules. There is a need of functions or modules that configures USCI module for I2C communication. We can categorize these as:

- USCI initialization module
- Set slave address or data register address into the buffer UCBxI2CSA.
- Read single byte operation on I2C bus
- Write operation on I2C bus

Let us discuss these functions one by one for better understanding.

USCI initialization module:

MSP430x5xx series devices support the USCI module that first needs to be configured for I2C communication. USCI initialization routine module has to perform following tasks:

Set UCSWRST

Initialize USCI registers

Configure ports for I2C master operation

Clear UCSWRST

Once the USCI module is configured as a master then the various operations can be performed.

As per I2C protocol specifications, I2C protocol specifies the method of transaction as per read or writes operation. Read/write sequence follows the following order:

Start sequence; send start bit

Send slave address of the device, in our case the address is 68H

Send R/W bit; 0 for read from the bus and 1 for write to the bus

Wait for acknowledgement ACK bit or send ACK to slave

Read or Write the data.

Wait or send acknowledgement

Send the stop bit

Set slave address or data register address into the buffer UCBxI2CSA.

The address of slave device or data register can be written with following settings:

Set UCSWRST bit

Place address into buffer UCBxI2CSA

Clear UCSWRST bit

Read single byte operation on I2C bus

It is a good programming practice to check USCI module for stop condition before beginning read or write operation transaction on I2C bus. This can be achieved by checking UCTxSTP flag; **while**

(UCBxCLT1 & UCTxSTP);

If it is the first read operation, then with START bit, UCTR bit is set.

UCBxCLT1 |= UCTR + UCTxSTT;

Check that buffer UCBxTxBUF is empty by monitoring IFG2 and UCB0TXIFG flag simultaneously, then place slave address for transmission. UCBxTXIFG interrupt flag is set by the transmitter to indicate that UCBxTxBUF is ready to accept another character. UCBxTXIFG is automatically reset if a character is written to UCxTxBUF or a NACK is received. Whereas UCxRXIFG interrupt flag is set each time a character is received and loaded into UCBxRXBUF. UCxRXIFG flag is automatically reset if a character is written to UCBxRXBUF or a NACK is received.

while (!(IFG2&UCB1TXIFG)); // character is transmitted means transmit flag is set, then put SA to transmit buffer

UCB1TXBUF =SA; // SA is slave address of the device whose data need to be read.

The read operation is always followed by write operation. Therefore, once slave address is transmitted by setting UCTR bit, we need to wait for the transmission and then perform write operation. For write operation, UCTR bit must be clear. This can be achieved as:

while (!(IFG2&UCB1TXIFG));

UCB1CTL1 &= ~UCTR; // clear UCTR bit for receive operation

UCB1CTL1 |= UCTxSTT; // generate repeated start

After this operation there is a need to clear the transmit flag to indicate that buffer is not ready to receive any further data byte. Monitor the UCTXSTT flag to check START condition, generate STOP condition, and try to read the data from receive buffer UCBxRXBUF.

```
while (UCB1CTL1 & UCTXSTT);  
UCB1CTL1 |= UCTXSTP;
```

Write operation on I2C bus

In the USCI module, first monitor the UCTXSTP flag to become 0. Then set the UCTR bit to enable master transmission mode and generate start condition;

```
while (UCB1CTL1 & UCTXSTP);  
UCB1CTL1 |= UCTR + UCTXSTT; // START condition is generated with UCTR set for master transmitter mode
```

Before putting slave address or data, one must check the Transmit buffer and then copy the data into UCBxTXBUF.

```
while (!(IFG2&UCB0TXIFG));  
UCB1TXBUF = slave address or data register address;
```

After putting data, wait for the transmission and acknowledgement from slave and then place data into transmit buffer.

```
while (!(IFG2&UCB1TXIFG));  
if(UCB1STAT & UCNACKIFG) return UCB1STAT;  
UCB1TXBUF = data;
```

The UCB1STAT needs to be known to know whether or not the acknowledgement received is from slave.

If the slave does not acknowledge the transmitted data, the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition. If this data should be transmitted after a repeated START, it must be written into UCBxTXBUF again and to trigger a repeated START, UCTXSTT must be set again.

After this, we need to wait for the transmission to be completed, generate STOP bit and clear the transmit flag.

```
while (!(IFG2&UCB0TXIFG));  
UCB0CTL1 |= UCTXSTP;  
IFG2 &= ~UCB0TXIFG;
```

Till now, we have discussed the various operations that need to be included in the complete I2C program. Let us discuss about the internal register details of MPU6050 that need to be configured.

WHO_AM_I register

This register is used to verify the identity of the device. The contents of WHO_AM_I are the upper 6 bits of the MPU-60X0's 7-bit I2C address. The default value of the register is 0x68.

PWR_MGMT_1 Register

Register	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
6B H	DEVICE_RESET	SLEEP_MODE			TEMP_DIS			

Table 4.9 PWR_MGMT_1 register

The PWR_MGMT_1 register has an address of 6BH. *DEVICE_RESET*: When set to 1, this bit resets all internal registers to their default values. The bit automatically clears to 0 once the reset is done. BIT 6: When set to 1, this bit puts the MPU-60X0 into sleep mode. *TEMP_DIS*: When set to 1, this bit disables the temperature sensor.

Data Registers

There are three types of data registers available in MPU6050.

Accelerometer measurement registers:

ADDRESS (HEX)	ADDRESS (Decimal)	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
3B	59	ACCEL_XOUT_H [15:8]							
3C	60	ACCEL_XOUT_L [7:0]							
3D	61	ACCEL_YOUT_H [15:8]							
3E	62	ACCEL_YOUT_L [7:0]							
3F	63	ACCEL_ZOUT_H [15:8]							
40	64	ACCEL_ZOUT_L [7:0]							

Table 4.10 Internal data registers

Temperature and Gyro measurement registers:

ADDRESS (HEX)	ADDRESS (Decimal)	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
41	65	TEMP_OUT_H [15:8]							
42	66	TEMP_OUT_L [7:0]							
43	67	GYRO_XOUT_H [15:8]							
44	68	GYRO_XOUT_L [7:0]							
45	69	GYRO_YOUT_H [15:8]							
46	70	GYRO_YOUT_L [7:0]							
47	71	GYRO_ZOUT_H [15:8]							
48	72	GYRO_ZOUT_L [7:0]							

Table 4.11 Internal data registers

4.9.6. Algorithm to read sensor values from MPU6050

Init_MSP430 routine:

Step 1: Declare and initialize fourteen 8-bit variable to place sensor values

AcXL, AcYL, AcZL, TmpL, GyXL, GyYL, GyZL, AcXM, AcYM, AcZM, TmpH, GyXM, GyYM, GyZM

Step 2: Configure USCI module of MSP430 as master

Init_MPU6050 routine:

Step 3: Place the slave address 68H value into buffer UCBxI2CSA

Step 4: Start transmission to slave device with START bit and put MSP430 in transmit mode

Step 5: place the register address 6BH into transmit buffer and wait for transmission.
 Step 6: Now place data value 00H into the transmit buffer to reset the device registers of MPU6050.

Step 7: Stop transmission using STOP bit

Read_AC_data routine:

Step 8: Start transmission with START bit and put MSP430 in transmit mode.

Step 9: place the slave address 68H into transmit buffer and wait for transmission.

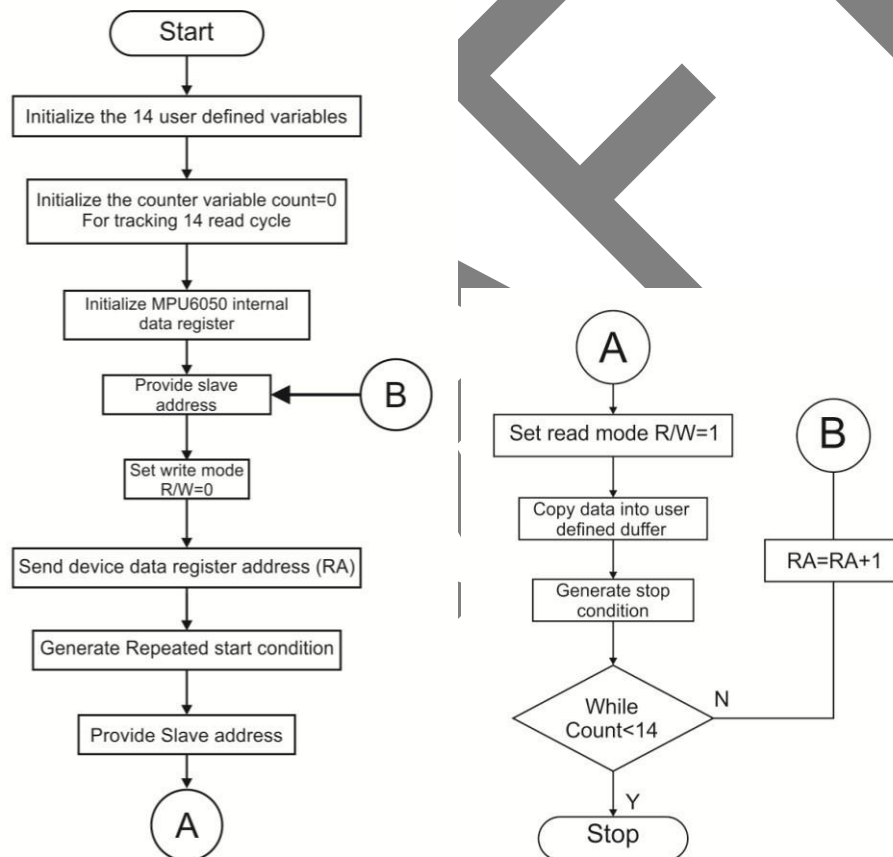
Step 10: Send internal register address 3BH.

Step 11: Send START sequence again

Step 12: Send slave device address with R/W bit HIGH.

Step 13: Read data byte from the bus and place it into MSB of AcX

Step 14: Send STOP bit and Repeat Step 8 upto 14 by incrementing the internal register address and placing the value into corresponding 8-bit buffer.



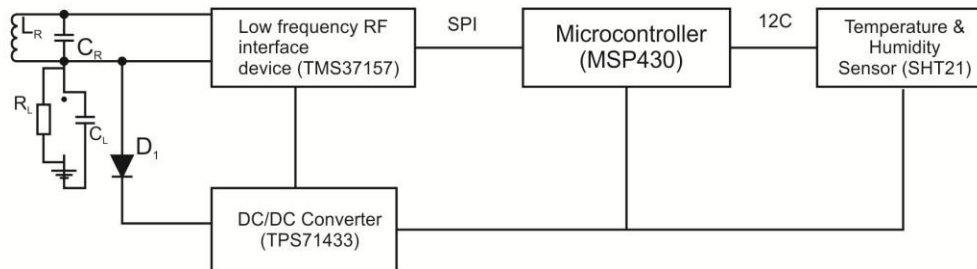
With the help of above algorithm, we can get the digitized acceleration values of various sensor modules for further processing depending upon application requirement.

4.10. Case Study: MSP430 based embedded system application using the interface protocols for communication with external devices: —A Low-Power Battery less Wireless Temperature and Humidity Sensor with Passive Low Frequency RFIDII

The smart weather monitoring systems today are quite sophisticated. They require wireless measurement of several environmental parameters such as humidity, pressure, temperature etc. For

this wireless measurement system, we need battery-less operations. In this type of applications we require wireless data and power transfer. We need a set of modules (shown in fig) combined together to perform this application. Blocks of this system are as follows:

- **Low frequency RF Interface Device(TMS37157):**To establish communication link
- **Microcontroller(MSP430):** To control the whole system
- **Humidity & Temperature Sensor(SHT21):**To sense the environmental parameter
- **DC/DC Converter(TPS71433):**To stabilize the power level

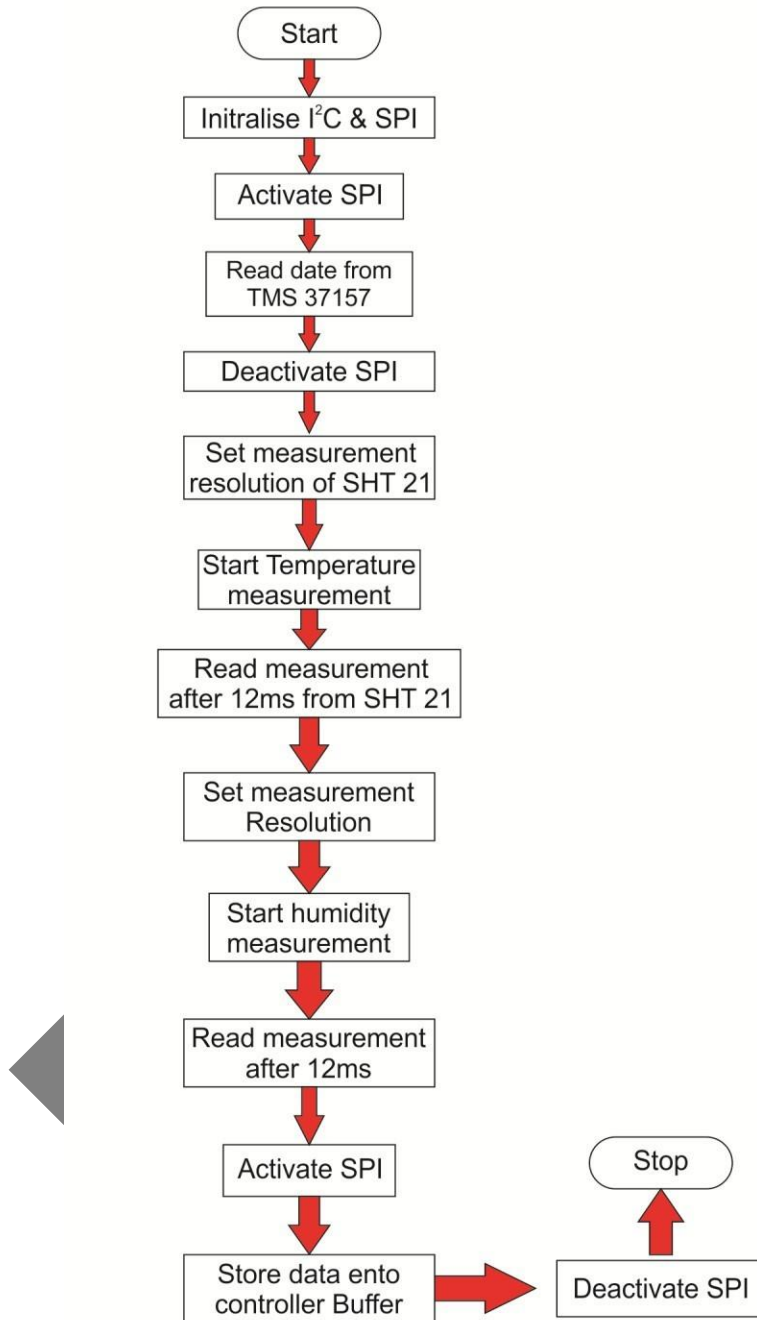


There are some phases to complete this functionality:

Charge phase: Generation of RF field from base station to the wireless sensor module to charge the power capacitor.

Downlink phase: Send command & instruction to wireless sensor to sense the atmospheric parameters.

Measurement & Recharge Phase: This phase contains three steps, first the measurement of temperature, second charging of power capacitor and third humidity measurement **Uplink Phase:** Send measurement result through Low frequency Interface Device.

Work Flow:


The operation of whole system starts with the initialization of SPI and I2C. This is because communication protocols are the backbone of this system either wired or wireless. After getting instruction from base station, the measurement starts. The power capacitor should be charged before measurement of each parameter. As the name suggests, the whole system is low power because the power capacitor is the only source. The overall flow of system can be understood by the flow chart below:

4.11. Summary

(Inputs required)

DRAFT

4.12. Review Questions

DRAFT

4.13. Exercises

DRAFT

Embedded Networking and Internet of Things

The previous chapter covered the process of interfacing the MSP430 with other devices using various serial communication protocols such as UART, I²C, SPI for sensor based and data logging applications. This learning is essential to integrate the MSP430 as part of a larger system of interconnected devices, which have the ability to communicate to each other over standard protocols.

With this background, the current chapter covers the study of IoT architecture and various wireless communication protocols to build the IoT applications with MSP430. Readers will also learn about the building blocks of IoT applications using MSP430 such as smart electric meter created by integrating it with a Wi-Fi module.

By the end of this chapter, readers can expect to understand wireless communication protocols and its integration with controllers; acquire knowledge about CC3100 SimpleLink Wi-Fi module and its architecture and know how to interface CC3100 module with MSP430.

Topic	Page
5.1. Introduction	199
5.2. Applications of IOT	201
5.3. Architecture of IOT	202
5.4. Challenges of IOT.....	203
5.5. Overview of Wireless Sensor Networks and Design Examples	204
5.6. Relation between WSN & IOT	205
5.7. Various Wireless Protocols and its Applications	208
5.8. Adding Wi-Fi to a Microcontroller-based system using CC3100 Simplelink Wi-Fi module	222
5.9. Building IoT Applications using CC3100 user API	226
5.10. Implementing Wi-Fi Connectivity in a Smart Electric Meter	228
5.11. Summary	229

5.1. Introduction

In this chapter, readers will be introduced to embedded networking technologies such as ZigBee, NFC, Bluetooth, Wi-Fi etc. These are key elements in designing internet enabled applications such as smart homes, low power wireless sensor networks, commercial building automation, industrial automation and in location tracking like asset tracking.

Let's take the example of a Wireless Sensor Network, which utilizes the harvested energy that comes from the energy sources such as wind, heat, light as the main power source. TI MSP430 series devices are ultra low power MCUs and are a better choice for designing various WSN networks.

This chapter covers wireless sensor network and different wireless protocols that provide connectivity between smart devices and gateway solutions and related case studies.

5.1.1. IOT Overview and Architecture

Today's embedded systems designers and developers are increasingly asked to incorporate Ethernet connectivity into their systems. Ethernet's ubiquity and longevity in connecting to a network makes it an attractive networking choice for embedded systems. Ethernet is a local area network (LAN) technology that is widely used to connect computers using wires or cables. Ethernet is similar to Wi-Fi technology, but with a different medium.

To put it differently, Ethernet is wired, and Wi-Fi is wireless. Ethernet is based on standards (IEEE802.3) that ensure reliability of network connections and data transmission, it ensures interoperability of Ethernet. Ethernet networks are scalable from the simplest to most complex networks or up to 2^{48} network nodes. Once equipment is connected to a Ethernet network, it can be monitored or controlled through the Internet removing any distance barrier that may have inhibited remote communication previously.

Based on its ease of use, low cost, high bandwidth, stability, security, and compatibility across devices, Ethernet has become the de facto standard of network access for 32-, 16- and even 8-bit microcontrollers. As we see from the below block diagram that MCU and Ethernet controller can connect any device to the world wide web. Thus it helps to monitor, control or access devices over internet.

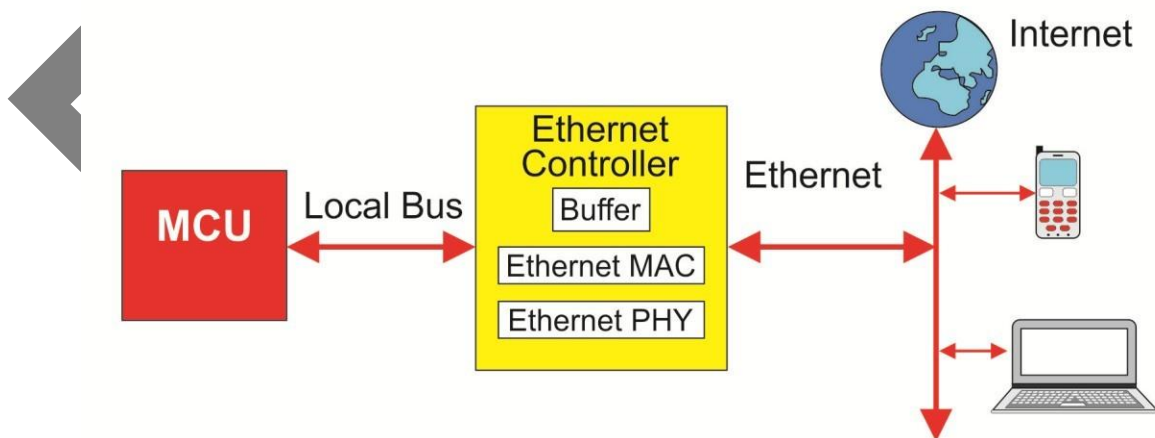
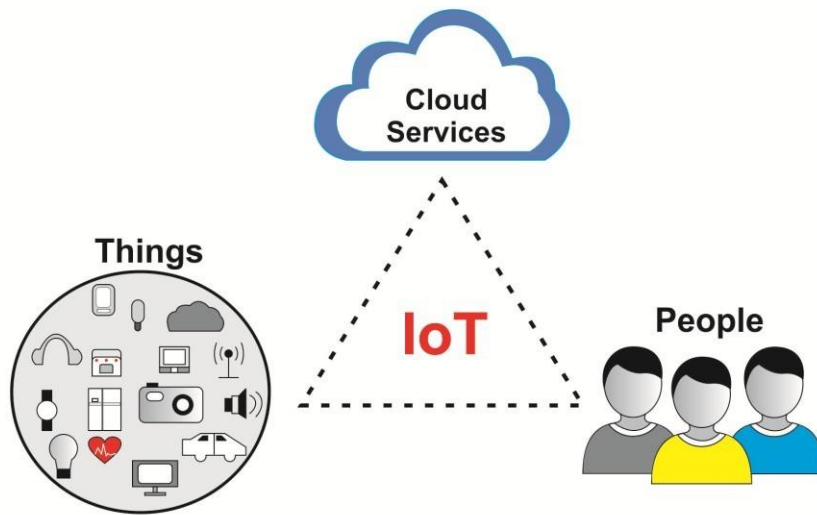


Fig 2. Embedded networking with Ethernet

Klevin Ashton introduced the term "Internet of Things" (IoT), to the world of technology in 1999. Since then, IoT has generated a lot of interest. It is expected that the number of „things“ connected to IoT will

grow from 20 billion in 2015 to an estimated 200 billion by 2020. The Internet has spread across various technological platforms and to billions of people in the world. With this spread getting wider, new technologies and devices have evolved. Examples include smart phones, smart watches and smart homes equipped with sensing technologies for smart energy, water management etc. The internet lies as the backbone for all of these devices and technologies. IoT refers to a scenario in which all the real-life things (including objects, people and animals) are connected to the Internet, and can transfer data over it, preferably to a cloud. This data can then be used by businesses and people to create a world of new possibilities and subsequently benefit from it. Fig. 1 shows the three main components of IoT i.e. things, data (cloud) and the people. For e.g. a smart refrigerator can sense the quantity of items inside it, and then automatically generate a shopping list to be ordered on-line. This list is then sent by the smart refrigerator on the cloud, where the best deals are offered for online purchase.



IoT can be generically thought of as connecting things to the Internet and using that connection to provide some kind of useful remote monitoring or control of these things. It is about harmonizing the way humans interact with devices and the environment using some common public services.

IoT in its culmination can be defined as the one that creates an intelligent, invisible network environment that can be sensed, controlled and programmed. IoT-enabled products employ embedded technology that allows them to communicate, directly or indirectly, with each other or the Internet.



Figure 1 IOT Enabled Home with Connected Devices and Appliances working invisibly for Consumers

Source: www.ti.com/lit/ml/slyb214/slyb214.pdf

IOT is considered as a scenario of accessing any information from anywhere and accessible to everyone. This is described as follows:

Anything: Eventually, any device, appliance or entity will be seamlessly connected to the Internet. Connectivity will not be the main feature of the device, but will extend the device's capabilities.

Anywhere: Any conceived wireless connectivity framework should be abstract enough to run from any location – both geographically and from a network topology perspective. The former refers to Internet-based ubiquity; the latter, refers to the ability to clone the framework into intranet environments where Internet access is restricted or undesired. Acknowledging the structure of the Internet beyond the public domain is important to enable the expansion of the IoT paradigm.

Anyone: Currently, not all things are connected to the IoT. But an IoT ecosystem that is easy to use and secure is not that far away. This will make the IoT accessible to anyone. Anyone will be able to connect their product to the Internet and also customize it to their personal preferences.

5.2. Applications of IOT

With the industry's broadest IoT-ready portfolio of wired and wireless connectivity technologies, microcontrollers, processors, sensors, analog signal chain and power solutions, Texas Instruments offers several cloud-ready system solutions. From high-performance home, industrial and automotive applications to battery-powered wearable and portable electronics or energy-harvested wireless sensor nodes, Texas Instruments makes developing applications easier with hardware, software, tools and support to get anything connected as an IoT device.

In automotive appliances, IoT is mainly used for infotainment purposes such as connecting between the phones and the speakers of the car, activating the engine through voice control etc.

The IoT paradigm discussed may be encountered in a wide variety of venues that span across various activity circles throughout the day utilizing different kinds of devices. In the personal area network we encounter wearable devices for entertainment and location tracking. As an example, it can be a Blue*tooth headset or GPS tracker. These devices enable users to enhance their health and wellness and gather information around the user. At home, we are surrounded with an ever-growing number of appliances, multimedia devices and other consumer gadgets. In home automation systems, IoT applications include monitoring and controlling the devices inside a home in an intelligent way. They include lighting and temperature control among the connected appliances for effective use of energy.

While on-the-go, we use private or public transportation vehicles and infrastructure to improve our mobility time utilization. In industries, sensors might be introduced for production efficiency, maintenance and failure management. And at a metropolitan level, smart building management systems include smart cities equipped with smart city lights, residential e-meters, surveillance cameras for traffic control, pipeline leak detection etc.

Healthcare IoT applications include remote monitoring of patients for example heart rate, blood pressure level etc.

5.3. Architecture of IOT:

The IoT players: We need to get a wider view of the IoT playground. To do that, the key players should necessarily be identified first. We classify the players into three clusters: users, things and services (Figure 3).

- *Users* are human participants that use services and their own end equipments. They mostly consume information and may inspire actions through profile settings and other decision-making processes.
- *Things* are physical or virtual endpoints representing either a data source, data sink or both. They feed or consume information to and from the Internet.
- *Services* are information aggregators and may provide tools for data analysis of different kinds. In some cases, they can be used to carry out actions requested by clients, either users or things.

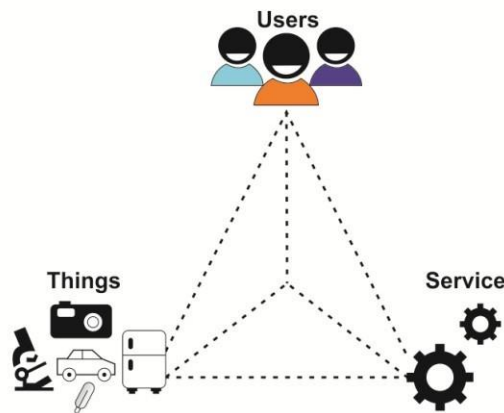


Figure 2 The IOT Players

The different devices and environments needed in IoT can be layered as shown in the Figure 3. The sensors and devices needed in the IoT environment are the bottom layer. The different types of sensors can be temperature, pressure, moisture etc. The data captured by the sensors needs to be processed using processors and enabling technologies. The technologies include RFID detection, motion sensing etc. Some of the technologies that enable these devices are discussed further in the Wireless Sensor networks section. Examples include Bluetooth, Wi-Fi etc. The processed data can be stored using cloud infrastructures and thus in turn provide different IoT services. The different types of IoT services include Home automation, healthcare services, energy management, emergency services among others.

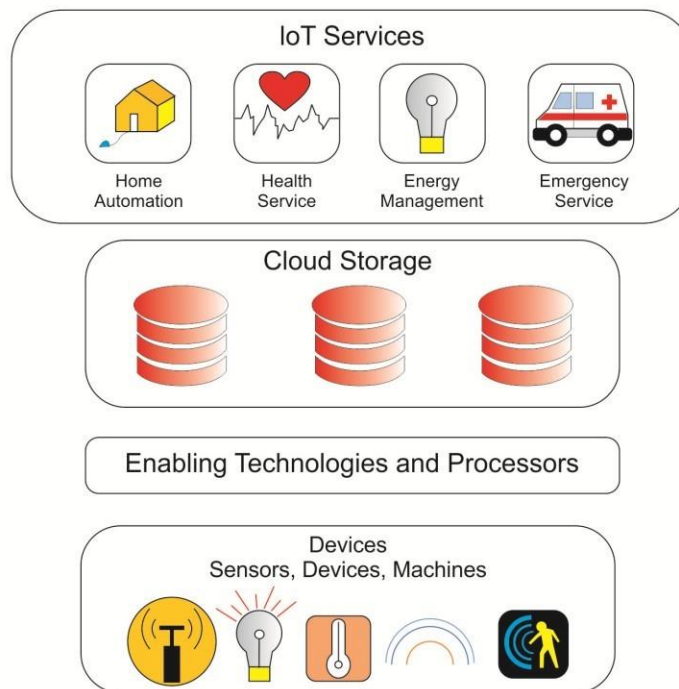


Figure 3 Architecture of IoT

5.4. Challenges of IOT

Preparing the lowest layers of technology for the horizontal nature of the IoT requires manufacturers to deliver on the most fundamental challenges, including:

- **Connectivity:** There is not one connectivity standard that ‘wins’ over the others. There is a wide variety of wired and wireless standards as well as proprietary implementations used to connect things in the IoT. The challenge is getting the connectivity standards to talk to one another with one common worldwide data currency.
- **Power management:** More things within the IoT require battery power, or need to use energy harvesting to be more portable and self-sustaining. Line-powered equipment needs to be more energy efficient. The challenge is making it easy to add power management to these devices and equipment. Wireless charging incorporates connectivity with charge management.

- **Complexity:** Manufacturers are looking to add connectivity to devices and equipment that have never been connected before to become part of the IoT. Ease of design and development is essential to get more things connected, especially when typical RF programming is complex. Additionally, the average consumers need to be able to set-up and use their devices without a technical knowledge.
- **Rapid evolution:** The IoT is constantly changing and evolving. More devices are being added everyday and the industry is still in its naissance. The challenge facing the industry is around unknown devices, unknown applications and unknown use cases. Given this, there needs to be flexibility in all facets of development. For example, it requires processors and microcontrollers⁴ that range from 16–1500 MHz to address the full spectrum of applications; right from a microcontroller (MCU) in a small, energy-harvested wireless sensor node to a high-performance, multi-core processors for IoT infrastructure. A wide variety of wired and wireless connectivity technologies are needed to meet the various needs of the market. Lastly, a wide selection of sensors, mixed-signal and power-management technologies are required to provide the user interface to the IoT and energy-friendly designs.

There are several fundamental features that a ‘thing’ has to encompass to be a good IoT solution. Among these, the most important features are energy efficiency, security, data handling and simplicity.

Energy Efficiency: As the number of devices grows, even small amounts of excessive power cause considerable waste. When it comes to power, the challenge is to ensure that adding Internet connectivity does not increase power requirement. It should ideally fit within the existing power budget headroom. The MSP430, being an ultra-low power MCU ensures that the IoT application takes minimal power.

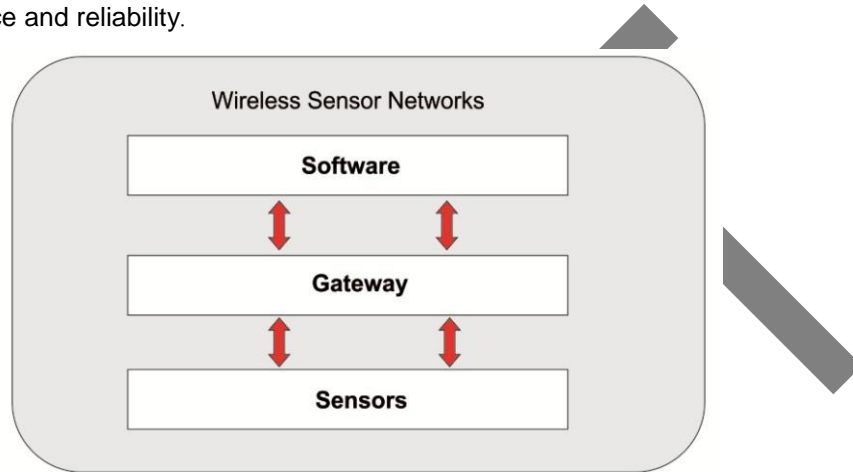
Security: Security is always a challenge in data networks. This challenge intensifies in the case of IoT simply because there are more entry points, thereby increasing vulnerability. The magnitude of the threat is also higher since it is not just data that is at risk. With IoT, the damage potential is much higher (e.g. opening a door remotely by taking a burglar-alarm system offline). Therefore, the fight towards better security can seem unending. The MSP430 provides inbuilt security features to address major security requirements.

Data handling: Massive deployment of endpoints results in higher node density. This requires demand for higher capacity. Furthermore, large quantity of data generated creates a need for accessible storage. In addition, real network latency introduces a challenge to limited resource systems. The TI wireless modules provide easy interfacing with the MSP430 to provide connectivity that suits the need of the IoT application.

5.5. Overview of Wireless Sensor Networks and Design Examples

Wireless Sensor Networks are infrastructures that contain sensing, computing and communication elements that aim to give its controllers the ability to measure, collect and react to occurrences in the monitored environment. It acts as an interface between the virtual and the physical worlds. Wireless Sensor Networks are networks consisting of dozens to thousands of small sensors that measure, calculate and communicate wirelessly among themselves and with central systems. The individual sensors or nodes function on extremely little power, harvesting the ambient energy around them to operate, so that they can be installed without a costly wired infrastructure and can keep going for years without maintenance. Small and inexpensive, these systems have been dubbed ‘stick-on’ sensors to emphasize how easy they are to install. Multipoint sensing has been prohibitively expensive since it consists of dozens, hundreds or even thousands of nodes. WSNs not only make it affordable, but also

enable dynamic control that reduces costs and extends capabilities. The main components of WSN include nodes, gateway and software (refer figure below). The spatially distributed nodes interface with sensors to monitor assets or their environment. The acquired data wirelessly transmits to the gateway, which can operate independently or connect to a host system where you can collect, process, analyze, and present your measurement data using software. Routers are a special type of node that can be used to extend WSN distance and reliability.



5.6. Relation between WSN & IOT

WSNs consist of smart sensing nodes with embedded CPUs, low power radios and sensors, which are used to monitor environmental conditions such as temperature, pressure, humidity, vibration, energy consumption etc. Wireless sensor networks behave like a digital „skin,“ providing a virtual layer where information about the physical world can be accessed by any computational system. As a result, they are an invaluable resource for realizing Internet of Things.

The benefits of connecting WSN and IoT elements go beyond remote access. It enables heterogeneous information systems that are able to collaborate by providing common services. Having IP connectivity does not mean that every sensor node needs to be directly connected to the Internet. There are many challenges to consider. The proliferation of wireless sensor networks and IoT solutions will drive enormous amounts of data across the Internet, requiring bigger pipes and the expansion of cloud computing. The potential of wireless sensor networks will be fully unlocked once it is connected to the Internet, thereby becoming part of the Internet of Things. However, the details and intricacy of integration at the network level (i.e. using direct TCP/IP connections) must be considered.

5.6.1. Applications of WSN's

Consider the example of a large building such as a food warehouse. It can have scores of wireless thermostats monitoring temperature and humidity, all reporting back to a central system that dynamically adjusts air-flow and cooling to keep conditions uniform and ensure minimize spoilage. As this example illustrates, WSNs find wide applications in areas such as industrial control and heating, ventilation and air conditioning (HVAC), which have traditionally relied on sensors. But as the variety of sensing nodes increases, it can potentially enable new types of applications such as body-area networks (BANs) that monitor health, agricultural networks that aid in crop cultivation, and environmental networks that help reduce pollution, warn of earthquakes or prevent widespread forest fires.

5.6.2. Design example of WSN

The movement towards an intelligent power network, frequently referred to as the ‘Smart Grid,’ promises not only improved delivery but also lower costs. These are achieved through improved infrastructure maintenance and a better understanding of patterns of consumption. Virtually every stage of power production and delivery can benefit from such remote sensors, from generators to high-power transmissions and substations, from street-level power lines and transformers to points of access in homes, factories and office buildings.

Energy Harvesting

While creating remote wireless sensors, it is important to achieve the right balance between system requirements (such as size, cost and reliable supply of components) and design support tools and software libraries. The availability of ultra-low-power (ULP) components is essential for the functioning of remote sensor nodes. Memories, transceivers, remote wireless sensors etc. are not feasible for low cost applications without ULP microcontrollers. Technologies that can gather and store ambient energy from light sources, vibrations or heat are also critical to the success of these devices and the WSNs they comprise (Figure 5).

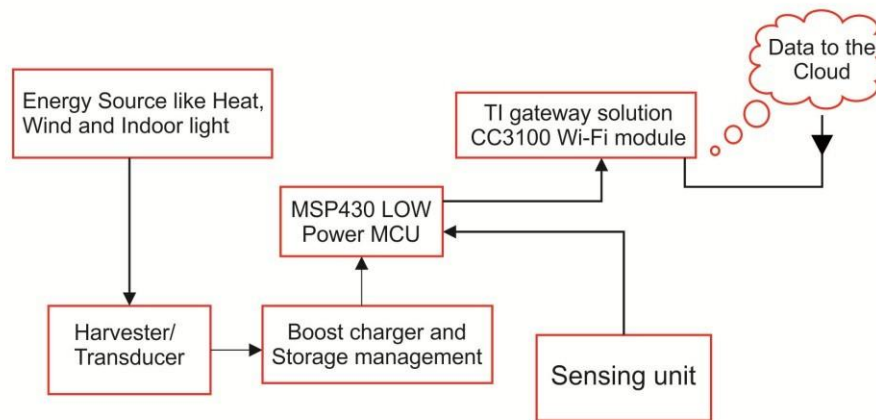


Figure 4 Energy Harvesting requires ultra-low-power components for all stages of the signal chain and power management

the heart of any measurement system is the microcontroller unit (MCU) that performs calculations and controls the system. Wireless sensor nodes require ULP microcontrollers that sleep for long periods—usually for more than 99 percent of the time—in order to conserve power. An active cycle refers to the process of waking quickly, performing measurement, running communication and control functions efficiently and returning to sleep. Because of the large number of potential applications for sensors, the MCU must include enough peripheral functions to make it flexible. But there needs to be a provision to turned off the ones that are not in use, to save power.

Smart World

As the Smart Grid is joined by intelligent communication among powered equipments everywhere: in homes, offices, factories, even outdoors (Figure 6). Such equipment and their networks are themselves extended by wireless sensor networks that monitor conditions and report to central systems, thus enabling control over extremely wide areas.

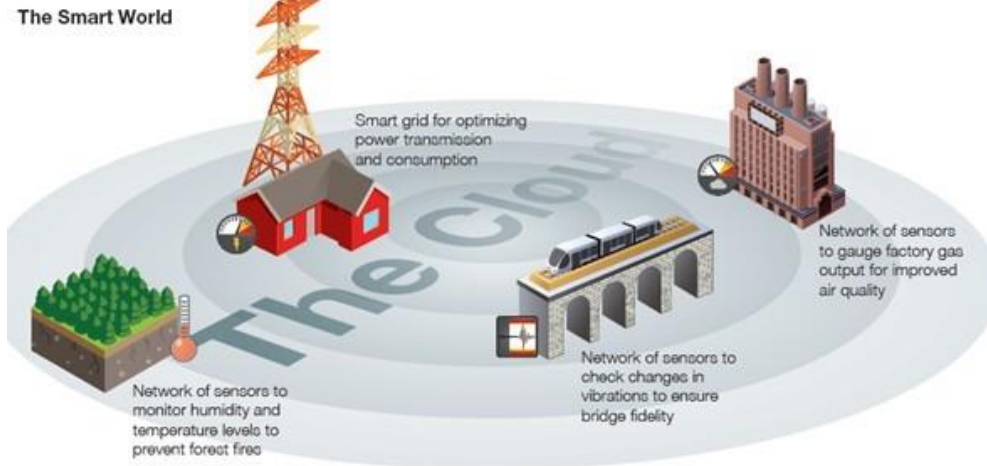


Figure 5 The Smart Grid complemented by many power and sensing networks

Source: <http://www.ti.com/lit/ml/sszy008/sszy008.pdf>

The design of the power supply subsystem is extremely important in wireless sensor nodes. The energy source, whether a solar panel, thermal or piezoelectric transducer, or other device, must be supported by circuitry designed to maximize energy harvesting despite the irregularities of ambient conditions. Energy storage in a rechargeable battery or super-capacitor, or both, requires careful management in order to optimize power so that it can be available whenever required. Elements that sense heat, current, chemicals or other environmental conditions must be sufficiently sensitive for accurate readings, yet not power-hungry. When all these components are selected to meet the right tradeoffs of functionality and low power consumption, the sensor node should be capable of functioning autonomously for years.

The WSNs depend on the availability of inexpensive, ultra-low-power components that can harvest ambient energy, sense local conditions, perform necessary measurement functions, and periodically communicate information via wireless transmissions. Based on its work with wireless utility meters and other applications, Texas Instruments has already developed this technology and today offers products that enable the development of "stick-on" sensor nodes for a variety of applications.

5.7. Various Wireless Protocols and its Applications

	Low Energy Bluetooth	ZigBee	NFC	Low Power WIFI
Frequency (MHz)	2402 – 2482	868 - 868.8, 902 - 928, 2402 – 2482	13.56	2400 - 2500
Channels	3	16	1	3
Modulation	GFSK	BPSK & QPSK	ASK	64QAM
Max potential data rate	1 Mbps	250 Kbps	424 Kbps	54 Mbps
Range	10m	100+m	10cm	30m
Power Profile	Days	Months/Years	Months/Years	Hours
Complexity	Complex	Simple	Simple	Complex
Nodes/Master	7	65,000	1+1	
Extendibility	No	Yes	No	Yes ³⁷

Figure 6 Comparison of different wireless technologies

5.7.1. NFC (Near Field Communication)

The MSP430 can interfaced with an NFC module to realize innovative close-contact battery-less, RF-field powered applications. Before we delve into using an NFC transponder with the MSP430, we must understand what makes NFC an ideal communication channel.

- ❑ NFC is a short-range high frequency wireless communication technology that enables the exchange of data between devices within a 10 cm distance from each other. NFC is an upgrade to the existing proximity card standard (RFID) that combines the interface of a smartcard and a reader into a single device.
- ❑ It allows users to seamlessly share content between digital devices, pay bills wirelessly or even use their cell phone as an electronic travelling ticket using existing contactless infrastructure already in use for public transportation.
- ❑ The significant advantage of NFC over Bluetooth is the shorter set-up time. Instead of performing manual configurations to identify Bluetooth devices, the connection between two NFC devices is established at once (under 1/10th of a second).
- ❑ Due to its shorter range, NFC provides a higher degree of security than Bluetooth and makes NFC suitable for crowded areas where correlating a signal with its transmitting physical device (and by extension, its user) might otherwise prove impossible. NFC can also work when one of the devices is not powered by a battery (e.g. on a phone that may be turned off, a contactless smart credit card, etc.).

Different types of modes: The different modes of NFC devices are as shown in the Figure 7.

An NFC device can operate in two modes: active, which is battery powered, and passive, which is radio-energy powered. NFC always involves both an initiator and a target. The initiator generates an RF field that can power a passive target. This enables NFC targets to take on non-battery operated, simple-form factors such as tags, stickers, key fobs and cards.

- *Active communication mode*: Both initiator and target device communicate by alternately generating their own fields. A device deactivates its RF field while it is waiting for data. In this mode, both devices typically have power supplies.
- *Passive communication mode*: The initiator device provides a carrier field and the target device answers by modulating the existing field. In this mode, the target device may draw its operating power from the initiator-provided electromagnetic field, thus making the target device a transponder.

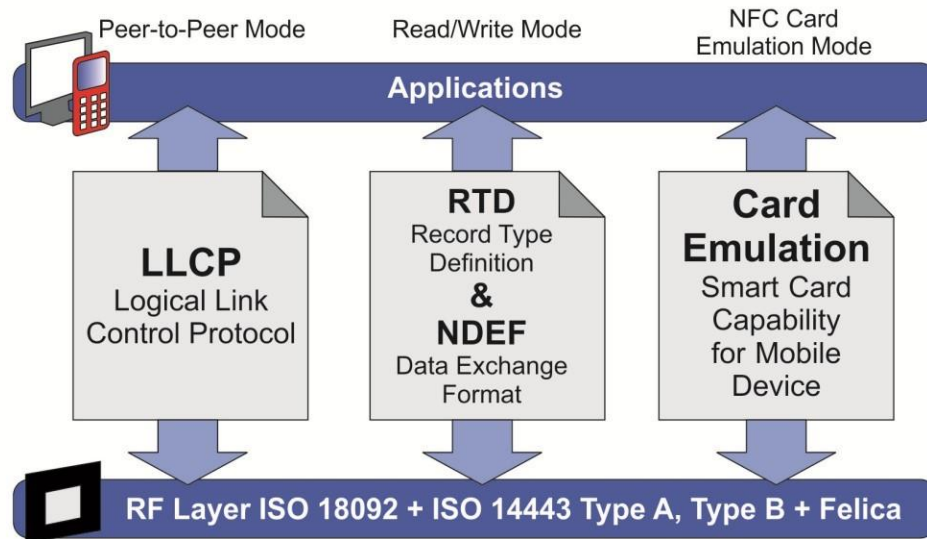


Figure 7: Different Types of Modes in NFC

Source: <http://www.oracle.com/technetwork/articles/javame/nfc-140183.html>

There are four tags-

- *Peer-to-Peer Mode* is defined for device-to-device, link-level communication.
- *Card Emulation Mode* allows the NFC handset to behave like a standard smartcard. This mode is secure and supported by the contactless communication API. The NFC device behaves exactly like a contactless card and can be used in transport fare payment systems based on MiFare, Calypso or FeliCa as well as open banking payment systems based on Visa® payWave, MasterCard® PayPass or American Express® ExpressPay.
- *Read/Write Mode* allows applications involving transmission of NFC forum-defined messages. Note that this mode is not secure. It is supported by the contactless communication API. The NFC device is active and reads a passive RFID tag; for example, reading and storing a web address or coupon from a poster for interactive advertising.
- *Person-to-Person (P2P) Mode* allows two NFC devices to communicate with each other while exchanging information.

Applications of NFC –

- Access control, provides additional security over traditional RFID tags
- Consumer electronics, implementing pay-for-use or smart debit functionality
- Healthcare, for secure identification of medical equipment
- Information collection and exchange from -smart posters
- Loyalty cards and coupons
- Mobile payments
- Transportation toll collection
- Electronic passport

Limitations of NFC -

- Equipment Required – NFC enabled equipment is required, NFC enabled cards and scanners
- Cost – Like any advance in technology, there are costs attributed to the development and implementation of NFC.
- Understanding – NFC (at the time of writing) is still a relatively unknown technology even-though it has been around for a long time. Many people do not know what NFC is, hence do not understand how to use it.
- Technical Limitations – NFC has limitations related to speed, volume of data it can transfer etc.
- Security - NFC has built-in security ready applications; but there is still a high risk of fraud; whereby people who fail to protect their passwords with any encryption or security setting could easily be targeted by hackers and criminals.

Case Study I: Use of NFC in integration with MSP430

This case study uses RF430CL330H Dynamic NFC Interface Transponder by Texas Instruments, which adds NFC functionality to the MSP430. In this case study, basic interfacing of the NFC module with the MSP430 is demonstrated, particularly, on writing data onto the NFC tag and reading newly written data from it.

The RF430CL330H is a Dynamic NFC Interface Transponder that combines a wireless NFC interface and a wired SPI or I2C interface to connect the device to a host. The NDEF message in the SRAM can be written and read from the integrated SPI or I2C serial communication interface. This operation allows NFC connection handover for an alternative carrier like Bluetooth, Bluetooth Low Energy (BLE), and Wi-Fi as an easy and intuitive pairing process or authentication process with only a tap.

As a general NFC interface, the RF430CL330H enables end equipments to communicate with the fast-growing infrastructure of NFC-enabled smart phones, tablets, and notebooks.

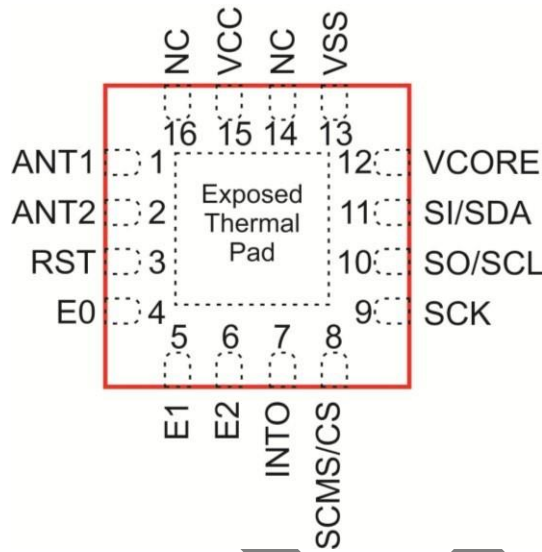
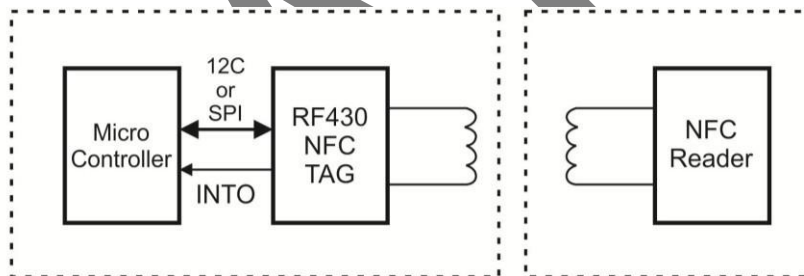


Figure 8a: Pin Diagram of NFC

Source: <http://www.ti.com/lit/ds/symlink/rf430cl330h.pdf>



Typical application diagram of NFC

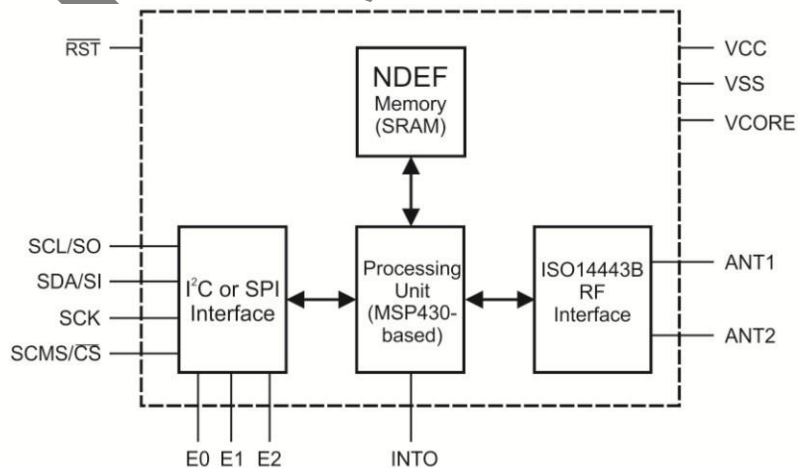
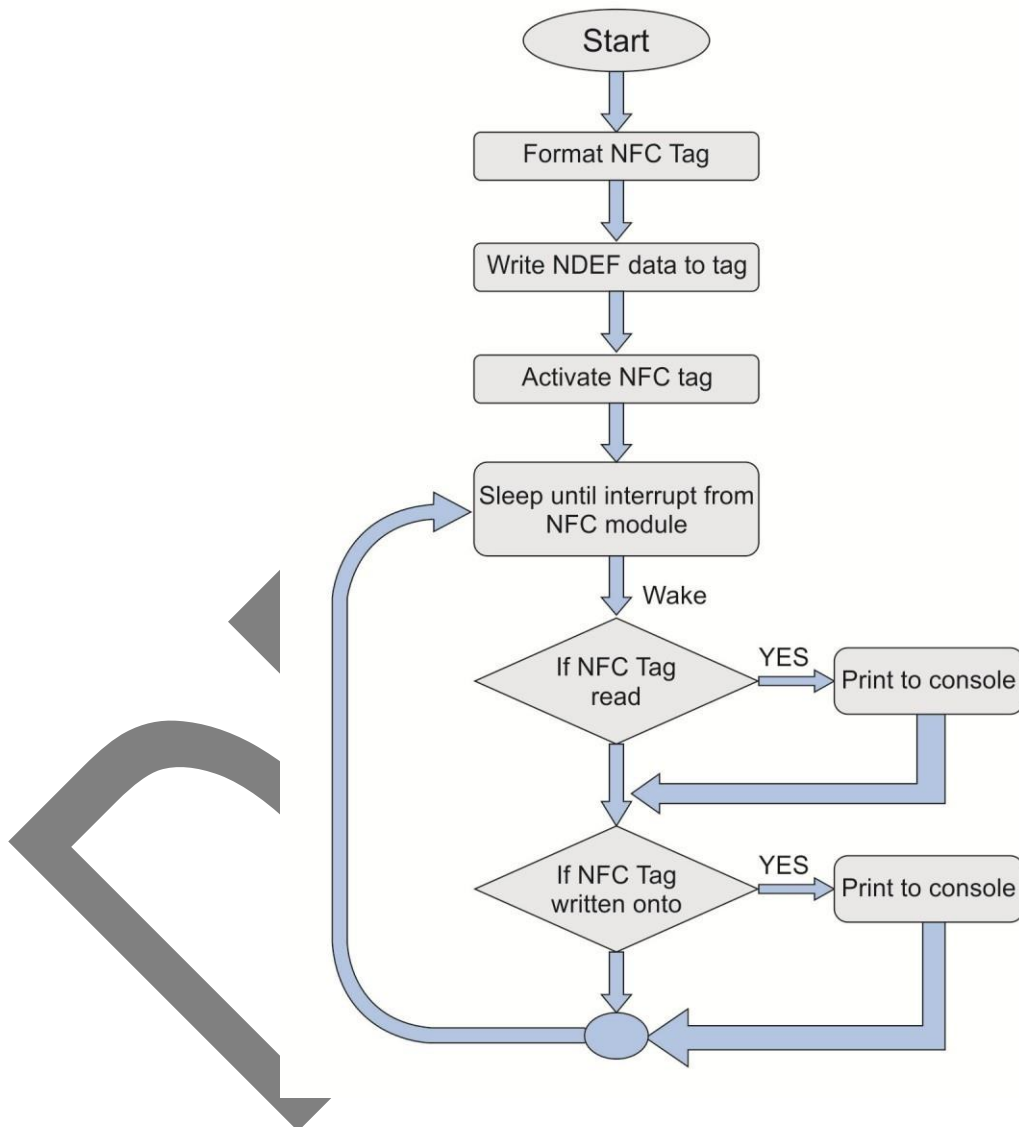


Figure 8b: Connection Diagram of NFC with MSP430

Source: <http://www.ti.com/lit/ds/symlink/rf430cl330h.pdf>

Key advantages of RF430CL330H Transponder

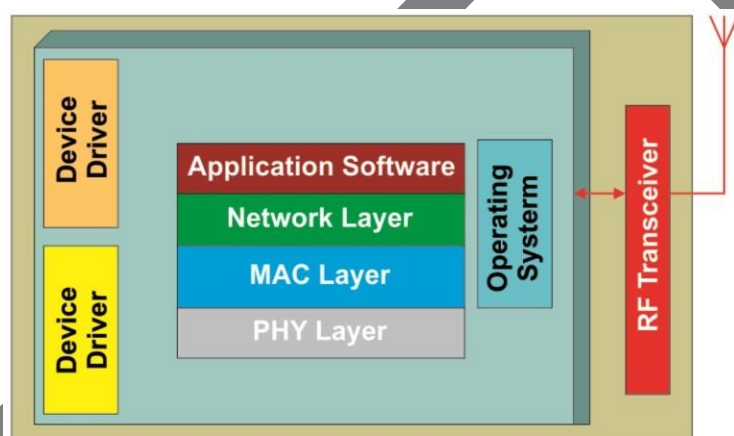
- ❑ It is a combination of both wireless NFC interface and wired SPI/I2C interface
- ❑ Dynamic update of data content supports update of pairing parameters
- ❑ The NFC uses alternative carriers like BT, BLE, Wi-Fi and RF4CE for pairing process.
- Wake up on RF – measurement of current consumption only when the device is in active state.
- ❑ The firmware requirements for microcontroller are very minimal
- ❑ NFC also acts as a service interface for firmware update and for data diagnostics



5.7.2. ZigBee

The MSP430 can be interfaced with a ZigBee wireless module to enable communication with other ZigBee compatible devices such as home automation systems.

ZigBee is a specification for a suite of high-level communication protocols used to create personal area networks built from small, low-power digital radios. ZigBee is based on an IEEE 802.15.4 standard. Though its low power consumption limits transmission distances to 10–100 meters line-of-sight, depending on power output and environmental characteristics, ZigBee devices can transmit data over long distances by passing data through a mesh network of intermediate devices to reach more distant ones. ZigBee is typically used in low data rate applications that require long battery life and secure networking. ZigBee has a defined rate of 250kbit/s, best suited for intermittent data transmissions from a sensor or input device.



Zigbee device block diagram

Source: <http://www.ti.com/lit/ml/slap129/slap129.pdf>

Components

The network model of ZigBee is as shown in Figure 8. The ZDO (ZigBee Device Object), a protocol in the ZigBee protocol stack, is responsible for overall device management, security keys and policies. It is responsible for defining the role of a device as either coordinator or end device but also for the discovery of new (one-hop) devices on the network and the identification of their offered services. It may then go on to establish secure links with external devices and reply to binding requests accordingly.

The application support sublayer (APS) is the other main standard component of the layer, and it offers a well-defined interface and control services. It works as a bridge between the network layer and the other components of the application layer: it keeps up-to-date binding tables in the form of a database, which can be used to find appropriate devices depending on the services that are needed and those the different devices offer. As the union between both specified layers, it also routes messages across the layers of the protocol stack.

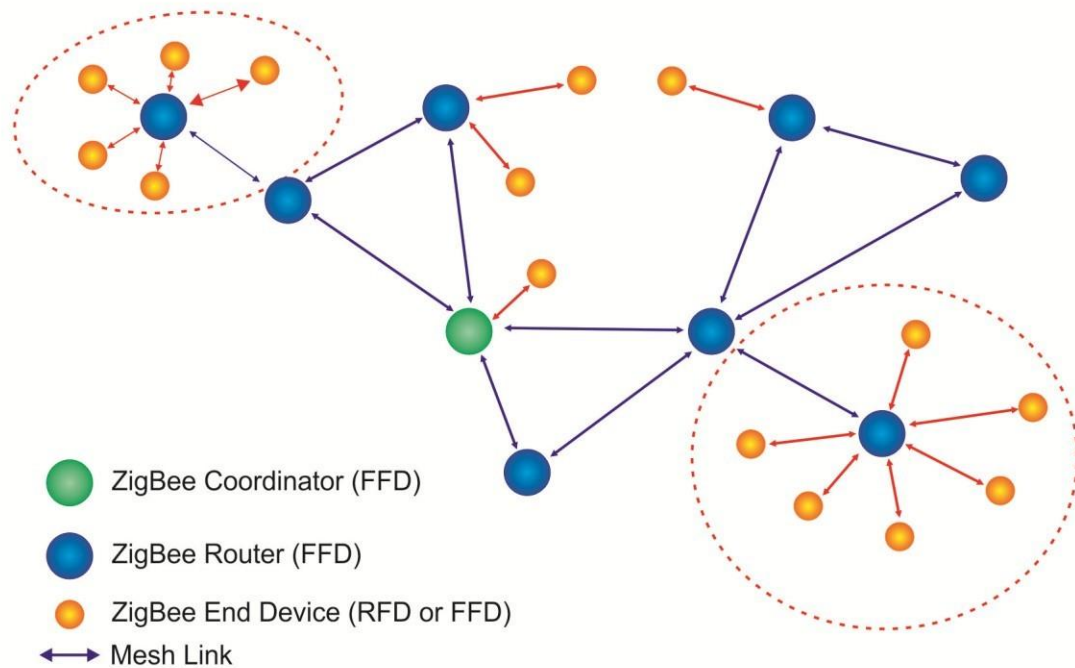


Figure 7 ZigBee Network Model

Source: <http://rfandwireless.ebv.com/glossary/overview/wlan-wireless-local-area-networks-wireless-lan/>

Applications –

- Home Entertainment and Control
- Automatic Meter Reading (AMR)
- Lighting, Heating, Alarm, Security
- White Goods health status monitoring
- Low power wireless sensor networks
- Industrial control like energy management
- Location technology like Asset tracking or Active RFID.
- Embedded sensing
- Hospital and patient care, medical data collection
- Smoke and intruder warning
- Building automation

Limitations –

Zigbee has limitations in the area of energy use restrictions for certification, memory size, processing speed of data, and size of bandwidth (Radmand, unk). Most importantly, Zigbee is vulnerable to various Zigbee Wireless Mesh Protocol network attacks and penetration.

Case Study II: Use of Zigbee in integration with MSP430 (The detailed pin diagram is shown in Figure 9).

This case study uses the Anaren A2530x24AZ1 AIR Module BoosterPack which is a combination of CC2530 low-power-RF chip and CC2591 range extender designed for the MSP430 value line development kit. AIR Module BoosterPack is intended to communicate wirelessly in compliance with Zigbee standard.

This case study discusses about the commissioning of ZigBee module as a ZigBee coordinator.

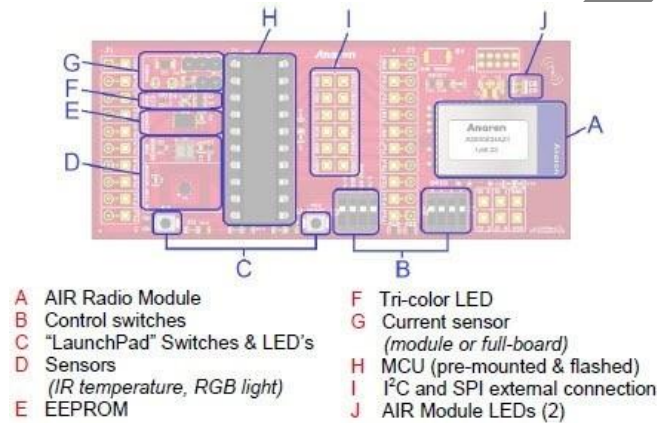
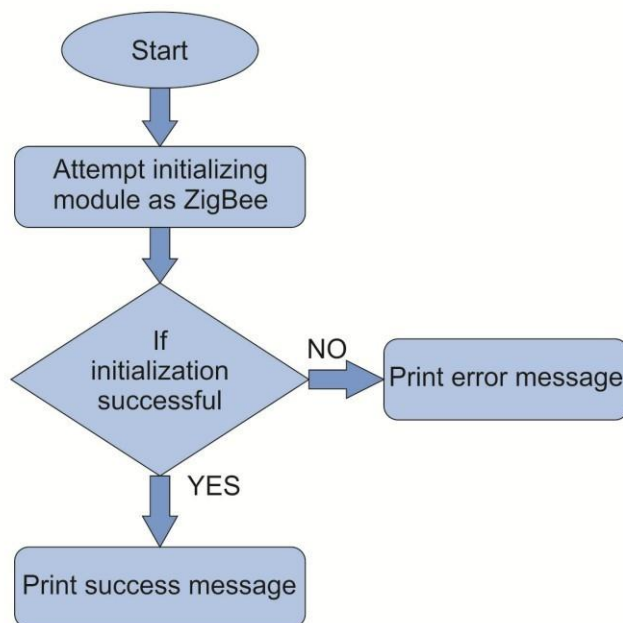


Figure 8 Pin Diagram of ZigBee Module

Source:

<http://www.mouser.com/newproducts/newproductsmanufacturers.aspx?mfg=anaren&virtualdir=anaren-A2530x24AZ1AIR/>



5.7.3. Bluetooth

Bluetooth functionality can be easily added to the MSP430 by interfacing a low-cost Bluetooth module. This enables the MCU to communicate with several devices that support Bluetooth connectivity and remotely control such devices by sending commands over Bluetooth. We may also receive data from a wide range of Bluetooth enabled sensors, which cannot be physically connected to the MSP430.

Bluetooth is a wireless protocol for exchanging data over short distances from fixed and mobile devices, creating personal area networks. Bluetooth devices have two important parameters - class and supported profiles. "Class" signifies the distance at which a Bluetooth connection is possible. Most mobile devices are Class 2, which means they have a range of up to 10 m. Class 1 devices are rare and have a range of up to 100 feet. A "profile" is a type of Bluetooth connection. The most common are the Headset (HSP) and Handsfree (HFP) profiles that enable the device to connect to a wireless headset or handsfree. Some other profiles are OBEX (*OBject EXchange*), which allows transfer of files, contacts and events; A2DP, which adds support for streaming of stereo sound and AVRC, which allows remote control of playback.

Components

Any Bluetooth solution consists of four major components (Figure 10): antenna/RF component, Bluetooth hardware and firmware (baseband and Link Controller), Bluetooth software protocol stack, and the application itself. Each of these components is a product in itself, and companies exist that have entire business models based around solving only one of these four areas.

Antenna/RF: The antenna and RF design portion is interesting in that it requires a unique solution for each device. When purchasing a Bluetooth module for Ericsson, for instance, the antenna is not provided. Bluetooth silicon manufacturers cannot effectively provide an antenna with the hardware. Even single chip solutions require specialized antenna design, depending on the device. Antenna design requires specialized skills to ensure that the Bluetooth radio will operate within its specification.

Bluetooth Radio and Baseband: The Bluetooth radio is the hardware transceiver unit that implements the Bluetooth radio specification. The purpose of the specification is to provide compatibility between Bluetooth devices that operate in the 2.4 GHz ISM band, and to define the quality of the system. Further information on the Bluetooth radio specifications may be found in the Bluetooth core specification document.

The Bluetooth baseband consists mainly of a Link Controller (LC) that carries out baseband protocols and low-layer link routines. Protocols defined within the scope of the baseband specification include physical channels and links, data packet definitions, error correction and detection, logical channels, channel control, and hop selection among others. For more information about the Bluetooth baseband specification, see the Bluetooth core specification document.

An example implementation of the Bluetooth radio and baseband is the Ericsson Bluetooth Module. In addition to the hardware, this module contains the firmware that implements the baseband specifications. As one would expect, there are a number of other manufacturers developing Bluetooth modules too.

Bluetooth Software Protocol Stack: The Bluetooth software protocol stack can be thought of as driver code. This code allows the application software to send and receive information from the Bluetooth module. Several implementations of this currently exist, and vary from GNU licensed code to commercial products targeted at various operating systems.

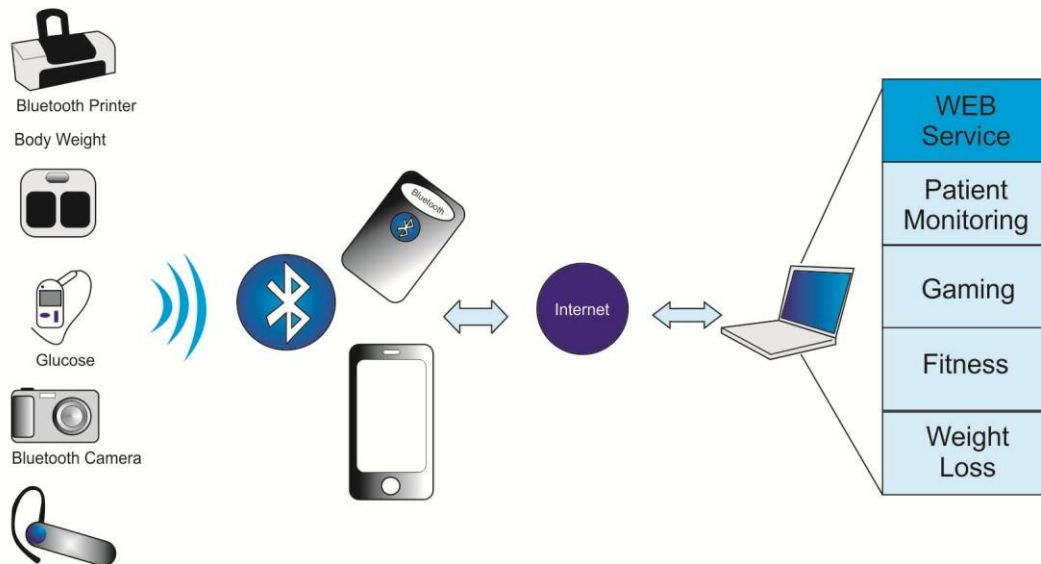


Figure 9 Applications of Bluetooth

Source: <http://www.arrownac.com/solutions-applications/machine-to-machine-technologies/bluetooth.php>

Applications

Bluetooth provides support for three general application areas using short-range wireless connectivity:

- Data and voice access points: Bluetooth facilitates real-time voice and data transmissions by providing effortless wireless connection of portable and stationary communications devices.
- Cable replacement: Bluetooth eliminates the need for numerous, often proprietary cable attachments for connection of practically any kind of communications device. Connections are instant and are maintained even when devices are not within line of sight. The range of each radio is approximately 10 m, but can be extended to 100 m with an optional amplifier.
- Ad hoc networking: A device equipped with a Bluetooth radio can establish instant connection to another Bluetooth radio as soon as it comes into range.

Limitations

- Data Transfer Rate
- Range
- Security
- Battery Use

Bluetooth Low Energy

Bluetooth low energy (Bluetooth LE, BLE, marketed as Bluetooth Smart) is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in healthcare, fitness, beacons, security and home entertainment industries. Compared to Classic, Bluetooth Smart is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range.

Components/Architecture –

Bluetooth Low Energy is a connectionless protocol that appreciably reduces the amount of time the radio is on and thus significantly decreases power consumption (Figure 11). Single mode BLE devices are designed for the smallest footprint, lowest power and lowest cost price point. Dual mode BLE devices support Bluetooth low energy protocols and classical Bluetooth technology. Bluetooth low energy consumes 10-20 times less power and is able to transmit data 50 times quicker than classical Bluetooth solutions.

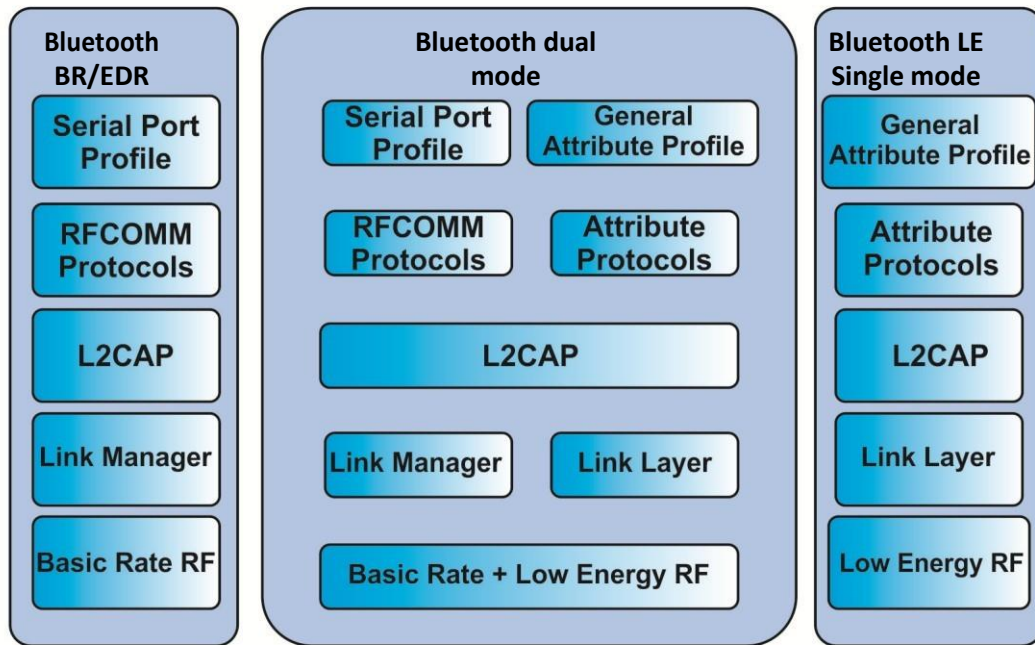


Figure 10 Bluetooth Low Energy

Applications

- Cable replacement
- Wireless sensors
- Industrial automation
- Sports and fitness
- Medical and healthcare devices
- Measurement
- Bluetooth access point
- Audio applications

Limitations

Data transfer rates with Classic Bluetooth technology using Enhanced Data Rate (Bluetooth v2.1 + EDR) can exceed 2 Mbps (actual payload), but practical transfer rates for Bluetooth low energy technology are below 100 kbps (1/20 of the actual payload). Streaming Bluetooth low energy connections lose a great deal of the huge potential power savings as the utilization approaches continuous transmission. Some

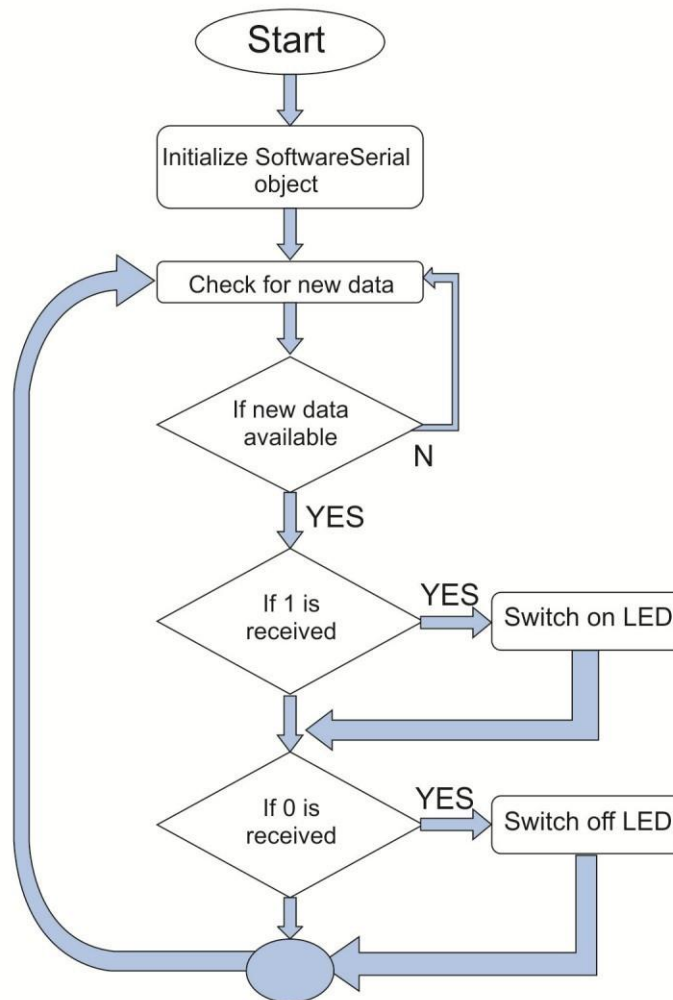
applications are better served with a Bluetooth low energy connection than others. The use varies depending on the type of device a sensor is connected to.

Case Study III: Use of Bluetooth in integration with MSP430

This code connects the PC to MSP430 via Bluetooth. When if 1 is pressed on the keyboard, LED is turned on and when if 0 is pressed, LED is turned off. The PC is first paired with the SimpleLink™ Bluetooth® Smart ultra-low power wireless module connected to the MSP430. A terminal emulator (Tera Term for instance) is used on the PC to send and receive data from the SimpleLink CC2640 using the Bluetooth Serial Interface by connecting to the respective COM port of the Bluetooth module on your PC.



Figure 11: Simple Link Bluetooth module



5.7.4. Embedded Wi-Fi

Embedded systems are not limited to simple 8-bit controllers; many are sophisticated and self-contained computer systems. These advanced embedded systems can support color LED displays, vast data storage, significant computing power and myriad interfaces. One growing trend in these systems is to add wireless connectivity. The use of proprietary or standards-based radios to add wireless requires RF and protocol expertise. Most companies do not have the RF expertise or tools to develop their own wireless interface.

While many wireless options exist, Wi-Fi has quickly become the de facto standard for wireless embedded systems because of its global acceptance and interoperability. Additionally, with the availability of low-power, self-contained, industry-certified solutions, Wi-Fi fits into many markets; including those with mobile and battery-powered requirements.

Components:

A typical WLAN hardware implementation uses several voltages and often includes power management logic to reduce power consumption. Ease-of-integration demands that the module be supplied with a single, standard voltage. A power management unit internal to the module generates the required voltages efficiently--often employing one or more DC-DC switching converters followed by low dropout linear regulators. In addition, the power management unit handles the fine-grained control of individual, internal supply paths used to power sections of the device, enabling maximum power savings through the selective switching on or off of portions of the device based on the needs of the operation profile over time. The WLAN device also requires a number of clocks and reference frequency signals for operation. The RF transceiver contains a frequency synthesizer that uses a stable frequency reference that adheres to the requirements of the standard--a frequency accuracy of 20 ppm or better.

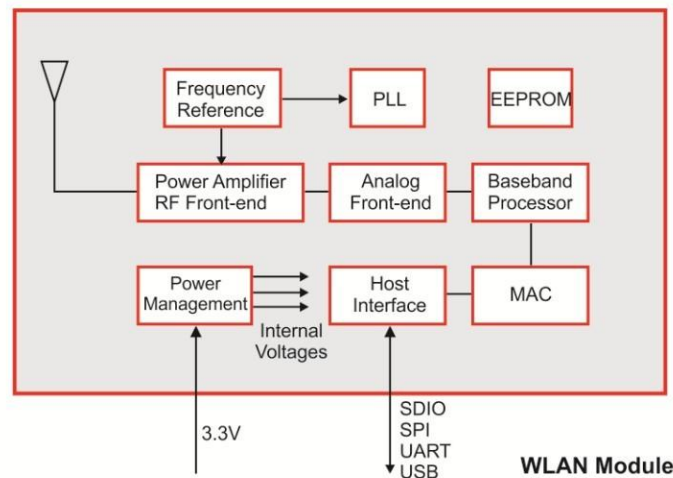


Figure 12 Components of WLAN Module

Source: <http://www.embedded.com/design/connectivity/4217738/2/Designing-Wi-Fi-connectivity-into-your-embedded-Internet-thing>.

Applications:

Computers, smart phones and tablets comprise the majority of wirelessly connected devices. There also applications within the smart home, energy grid, personal healthcare, medical and asset-tracking markets ready for Wi-Fi integration.

Embedded systems for a large variety of applications--including appliances, automation systems, medical devices, entertainment systems, and energy management--today already use or can potentially use a wireless interface.

Limitations

- Security
- Range
- Reliability
- Speed

5.8. Adding Wi-Fi to a Microcontroller-based system using cc3100 Simplelink Wi-Fi module

To illustrate the use of wireless connectivity in embedded networks, this section discusses the usage of Wi-Fi technology with a microcontroller. Wi-Fi is very widely used to provide connectivity between users and embedded systems. For example, a user can interact with utility systems (such as AC, Garage door, Coffee machine etc.) in a smart-home using a smartphone, provided both (smart-home and smartphone) are connected to the internet.

Texas Instruments provides low-power and easy-to-use Wi-Fi solutions that include battery-operated Wi-Fi designs with more than a year of battery life on two AA batteries. Texas Instruments' SimpleLink Wi-Fi CC3100 module is a wireless network processor with on-chip Wi-Fi, internet, and robust security protocols. It can be used to connect any low-cost microcontroller (MCU). A functional block diagram of CC3100 module is shown in Fig. 9.

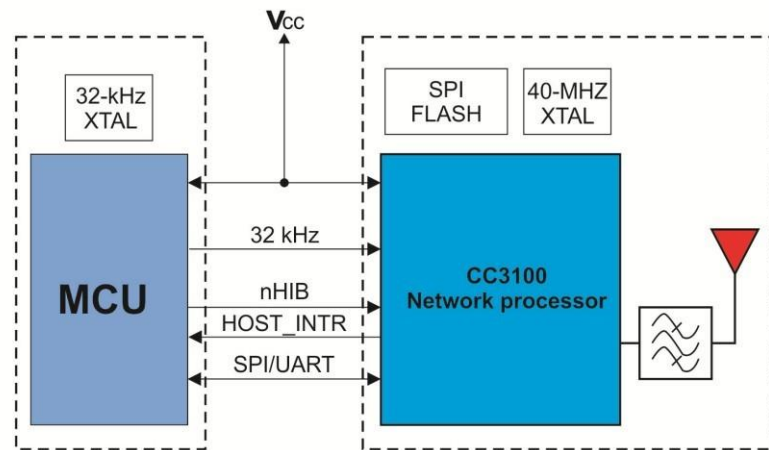


Fig. 15 Functional diagram of SimpleLink Wi-Fi CC3100 Module

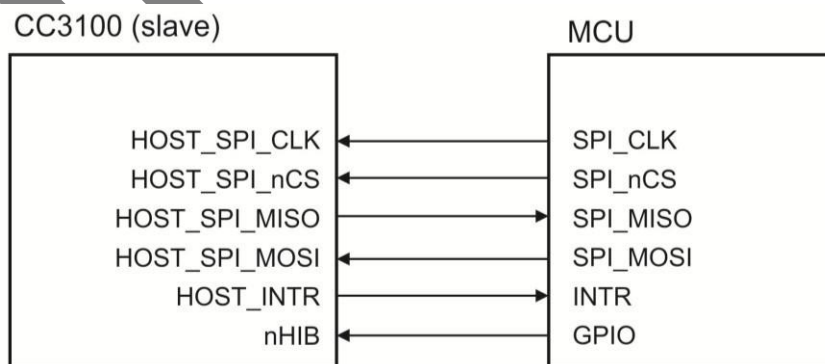


Fig 16: Connection of CC3100 with Msp430



Fig17: CC3100 mounted on MSP430

5.8.1. Architecture of SimpleLink Wi-Fi CC3100 Modu

It is important to understand the hardware and software architecture of any device before using it in a design. Fig. 11 shows the hardware architecture for SimpleLink Wi-Fi CC3100 module, that can be used to provide Wi-Fi connectivity to any micro-controller based system. It consists mainly of two parts:

- Wi-Fi Network Processor Subsystem
- Power-management Subsystem

Wi-Fi Network Processor Subsystem

The Wi-Fi Network Processor subsystem mainly consists of the following:

- Dedicated ARM MCU – It executes the Wi-Fi and Internet protocols required to communicate over the Internet using Wi-Fi connectivity.
- ROM – stores pre-programmed Wi-Fi driver and multiple Internet protocols
- TCP/IP Stack – supports communication with computer systems on the Internet
- Crypto Engine – provides fast, and secure Wi-Fi as well as Internet connectivity
- 802.11 b/g/n Radio, Baseband and Medium Access Control - for wireless transmission and reception of data
- SPI/ UART Interface – connects the module to the host MCU.

CC3100

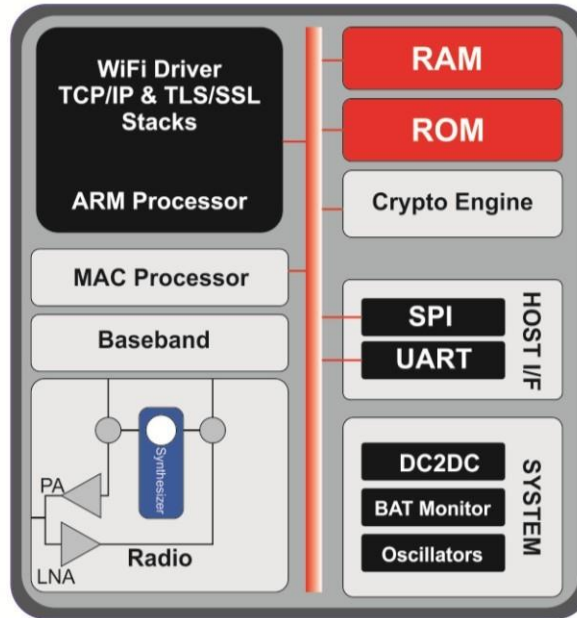


Fig. 17 Hardware Architecture for CC3100

Power Management Subsystem

The power management subsystem of CC3100 module provides the CC3100 module with an integrated DC-to-DC converter with a wide range of power supply from 2.3 to 3.6 V. This subsystem enables low-power consumption modes such as hibernate with RTC mode, which requires approximately 7 μ A of current.

Features of Wi-Fi supported by CC3100 chip

The Wi-Fi network processor sub-system in the SimpleLink Wi-Fi CC3100 device integrates all protocols for Wi-Fi and internet, greatly minimizing MCU software requirements. With built-in security protocols, SimpleLink Wi-Fi provides a simple yet robust security experience. This section discusses the features of Wi-Fi supported by the CC3100 device. A list of features and the functionality provided by them is in Table-1 below.

Table-1 Wi-Fi Features supported by SimpleLink CC3100 module

Sr. No.	Wi-Fi Feature	Function/ Utility
1	Supports 1-13 Wi-Fi channels	Provides 13 channels in 2.4 GHz frequency band
2	Support for WEP, WAP, WAP2	Secure Wi-Fi access
3	Enterprise Security	Provides additional security for enterprise networks
4	Wi-Fi Protected Set-up with WPS2	Provisioning methods to connect to Wi-Fi
5	Access Point mode with internal HTTP server	
6	SmartConfig technology	

7	802.11 Transceiver	Transmits and receives Wi-Fi packets
8	Supports IPv4	Internet Protocol
9	802.11 Power save and device deep sleep power with three user configurable policies	Low Power Operation
10	Up to 8 open sockets Up to 2 secured application sockets	User Application Sockets

5.8.2. CC3100 SimpleLink Driver and its Application Programming Interface (API)

In order to simplify development using the SimpleLink Wi-Fi devices, TI provides a simple and user friendly host driver software. This driver software allows any MCU (like TIVA platform) to interact with a SimpleLink device and performs the following functions:

- Provides a simple API for user application development.
- Handles the communication of MCU with the SimpleLink device.
- Provides flexibility in working with a MCU, with or without an OS.
- Works with existing UART or SPI physical interface drivers
- Compatible with 8-bit, 16-bit or 32-bit MCUs

The SimpleLink Host Driver includes a set of six logical and simple API modules:

- **Device API** – Manages hardware-related functionality such as start, stop, set, and get device configurations.
- **WLAN API** – Manages WLAN, 802.11 protocol-related functionality such as device mode (station, AP, or P2P), setting provisioning method, adding connection profiles, and setting connection policy.
- **Socket API** – The most common API set for user applications, and adheres to BSD socket APIs.
- **NetApp API** – Enables different networking services including the Hypertext Transfer Protocol (HTTP) server service, DHCP server service, and MDNS client/server service.
- **NetCfg API** – Configures different networking parameters, such as setting the MAC address, acquiring the IP address by DHCP, and setting the static IP address.
- **File System API** – Provides access to the serial flash component for read and write operations of networking or user proprietary data.

Programmer's model for CC3100 SimpleLink driver and its API

A programmer using a SimpleLink device needs to know about the different software blocks needed to build a networking application. The recommended flow for most applications is described below. However, program developers have complete flexibility on how to use the various software blocks.

Programs using the SimpleLink device consist of the following software blocks:

- **Wi-Fi subsystem initialization** – Wakes the Wi-Fi subsystem from the hibernate state.
- **Configuration** – WiFi sub-system. This phase refers to the initial time configuration that does not happen very often. For instance, changing the WiFi sub-system from a WLAN STA to WLAN soft AP, changing the MAC address and so forth.

- **WLAN connection** – The physical interface needs to be established. There are numerous ways to do so, all of which will be explained in this document. The simplest way is to manually connect to an AP as a wireless station.
- **DHCP** – Although not an integral part of the WLAN connection, you need to wait for the receiving IP address before continuing to the next step of working with TCP\UDP sockets.
- **Socket connection** – At this point, it is up to the application to set up its TCP\IP layer. This phase can be broken down into the following parts:
 - **Creating the socket** – Choosing to use TCP, UDP or RAW sockets, whether to use a client or a server socket, defining socket characteristics such as blocking\non-blocking, socket timeouts, and so forth.
 - **Querying for the server IP address** – On most occasions, when implementing a client side communication, you will not know the remote server side IP address, which is required for establishing the socket connection. This can be done by using DNS protocol to query the server IP address by using the server name.
 - **Creating socket connection** – When using the TCP socket, it is required to establish a proper socket connection before continuing to perform data transaction.
- **Data transactions** – Once the socket connection is established, it is possible to transmit data both ways between the client and the server; basically implementing the application logic.
- **Socket disconnection** – Upon finishing the required data transactions, it is recommended to perform a graceful closure of the socket communication channel.
- **Wi-Fi subsystem hibernate** – When not working with the Wi-Fi subsystem for a long period of time, it is recommended to put it into hibernate mode.

5.9. Building IoT Applications using CC3100 user API

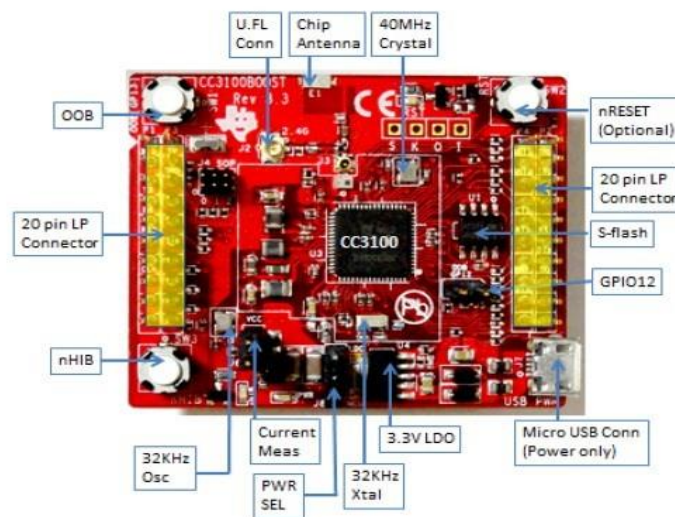
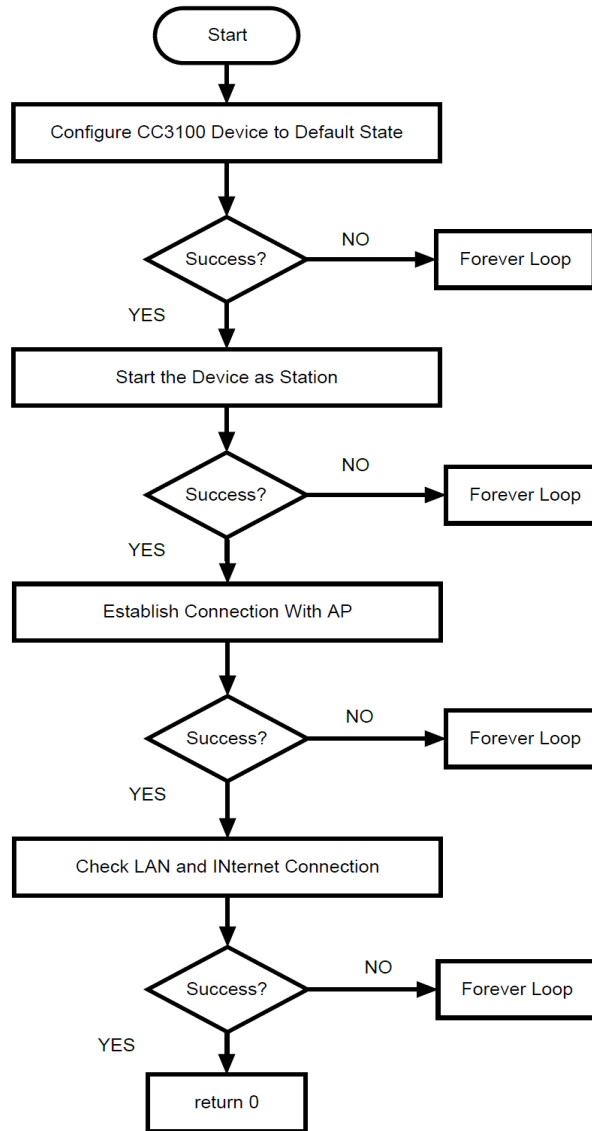


Figure 15 CC3100 Layout

Source: www.ti.com/lit/ug/swru371b/swru371b.pdf

Case Study IV: Configuration of CC3100 as a WLAN station by interfacing with MSP430F5529.

This case study demonstrates interfacing the MSP430 with CC3100 WiFi module. The following code attempts to configure CC3100 Booster Pack as a Wireless Local Area Network (WLAN) Station and connect to a Wi-Fi access-point. The application connects to an Access-Point and pings the gateway. It also checks for internet connectivity by pinging the website www.ti.com.



5.10. Implementing Wi-Fi Connectivity in a Smart Electric Meter

A smart meter is an electronic device that records energy consumption in some time intervals and communicates that information back to the utility for monitoring and billing. Around the world, electric meters are leading the way in smart meter deployments. Modern e-meters must meet certain criteria to play such a critical role in the smart grid rollout. First, meters need to report energy consumption information from houses and buildings back to the utilities. These meters need to deliver useful power consumption information into the home through an in-home display or a gateway. This information allows consumers to adapt energy behavior and lower utility bills.

Additionally, the MCU on a smart meter needs to support advanced functions like dynamic pricing/demand response, remote connect and disconnect, network security, over-the-air downloads and post-installation upgrades; so utility providers don't have to send out technicians to each meter. Texas Instruments has increased the availability of its field-tested metrology evaluation kits and grown its portfolio of metrology ICs with more memory, security and accuracy.

This case study demonstrates a three-phase energy meter with Wi-Fi connectivity. The three-phase energy meter is used to perform all metrology functions like RMS current, RMS voltage, power factor, frequency, active and reactive power and energies and to control the SimpleLink™ Wi-Fi transceiver. The smart meter data can then be displayed on any Wi-Fi connected device via a standard web browser.

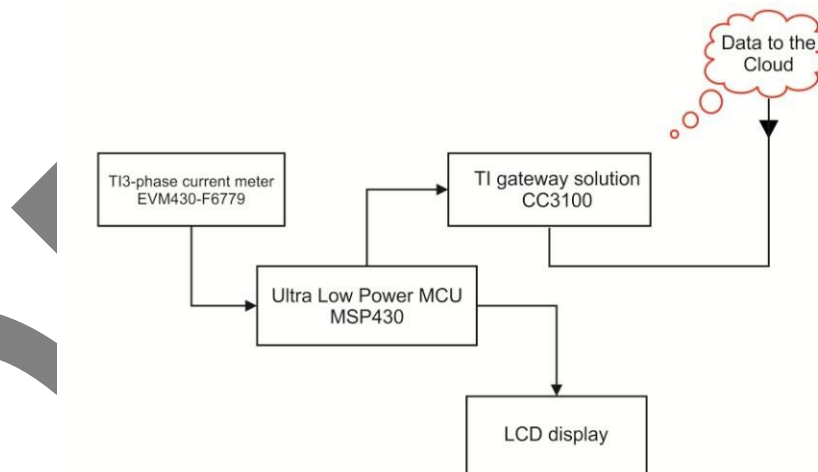
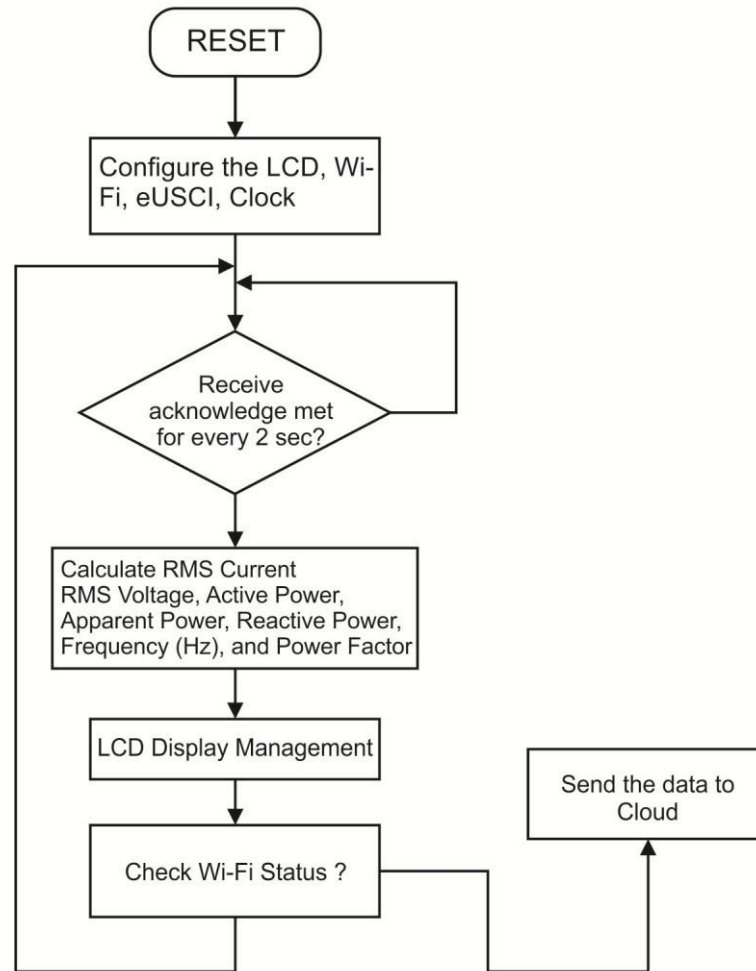


Figure 13 Smart electric meter system block diagram

Source: www.ti.com/lit/ml/slyb214/slyb214.pdf

In this case study TI's EVM430-F6779 3-phase current meter is used to measure all metrology functions. The measured energy consumption is displayed on the LCD every two seconds. For every parameter or metrology function, the metering is displayed on the LCD. In addition to displaying the metrology results on the LCD, the LCD is also used to display the Wi-Fi status.



5.11. Summary

In this chapter we covered the concept of Internet of Things and how the MSP430 microcontroller helps in realizing IoT applications. The concept of IoT is synonymous to connectivity. TI offers a wide-range of connectivity modules based on different wireless technologies to interface with the MSP430. The vast array of modules compatible with the MSP430 ensures that every IoT application makes use of the wireless technology that best suits the particular IoT application.

The potential that IoT offers is limitless, and the MSP430 and TI's portfolio of wireless modules are ideal candidates for diverse IoT applications.

5.12. Review Questions

DRAFT

5.13. Exercises

DRAFT