MICRO CONTROLLLER ASSIGNMENT-II

Q1: Explain the following addressing modes of MSP 430:

Register mode:

This uses one or two of the registers in the CPU. It is the most straightforward addressing mode and is available for both source and destination. For example,

mov .w R5 , R6 ; move (copy) word from R5 to R6

The registers are specied in the instruction word; no further data are needed. It is also the fastest mode and this instruction takes only 1 cycle. Any of the 16 registers can be used for either source or destination but there are some special cases: The PC is incremented by 2 while the instruction is being fetched, before it is used as a source. The constant generator CG2 reads 0 as a source. Both PC and SP must be even because they address only words, so the lsb is discarded if they are used as the destination. SR can be used as a source and destination in almost the usual way although there are some details about the behavior of individual bits. For byte instructions, Operands are taken from the lower byte; the upper byte is not affected. The result is written to the lower byte of the register and the upper byte is cleared. The upper byte of a register in the CPU cannot be used as a source. If this is needed, the 2 bytes in a word must rst be swapped with swpb.

Indexed mode:

This looks much like an element of an array in C. The address is formed by adding a constant base address to the contents of a CPU register; the value in the register is not

changed. Indexed addressing can be used for both source and destination. For example, suppose that R5 contains the value 4 before this instruction:

mov .b 3(R5), R6 ; load byte from address 3+(R5)= 7 into R6

The address of the source is computed as 3 + (R5) = 3 + 4 = 7. Thus a byte is loaded from address 7 into R6. The value in R5 is unchanged. There is no restriction on the address for a byte but remember that words must lie on even addresses. Indexed addressing can be used for the source, destination, or both. The base addresses, just the single value 3 here because only one address is indexed, are stored in the words following the instruction. They cannot be produced by the constant generator. The instruction is not normally used like this with numerical constants. More typically the base is the address of the rst element of an array or table and the register holds the index. Thus the indexed address Message(R5) is equivalent to the element Message[i] of an array of characters in C, assuming that R5 is used for the index and R5 = i. Look back at Listing 4.15. However, there is an important difference between C and assembly. The CPU always calculates the indexed address in bytes, while C takes account of the size of the object when calculations are performed with pointers. Suppose that Words[] is an array of words. In C the two expressions Word[i] and *(Word+i) are equivalent. The corresponding indexed address would be Word(R5) with R5 = 2i because each word is 2 bytes long. These examples use general-purpose registers but the full power of indexed addressing is released when the special registers are used as well.

Indirect register mode:
This is available only for the source and is shown by the symbol @ in front of a register, such as @R5. It means that the contents of R5 are used as the address of the operand. In other words, R5 holds a pointer rather than a value. (The contents of R5 would be the operand itself if the @ were omitted.) Suppose that R5 contains the value 4 before this instruction: mov .w @R5 , R6 ; load word from address (R5)= 4 into R6

The address of the source is 4, the value in R5. Thus a word is loaded from address 4 into R6. The value in R5 is unchanged. This has exactly the same effect as indexed addressing with a base address of 0 but saves a word of program memory, which also makes it faster. This is very loosely the equivalent of r6 = *r5 in C, without worrying about the types of the variables held in the registers. Indirect addressing cannot be used for the destination so indexed addressing must be used instead. Thus the reverse of the preceding move must be done like this:

mov .w R6 ,0(R5) ; store word from R6 into address 0+(R5)= 4

The penalty is that a word of 0 must be stored in the program memory and fetched from it. The constant generator cannot be used.

Ubdurect ayti ubcrenebt register mode:
Again this is available only for the source and is shown by the symbol @ in front of a register with a + sign after it, such as @R5+. It uses the value in R5 as a pointer and automatically increments it afterward by 1 if a byte has been fetched or by 2 for a word. Suppose yet again that R5 contains the value 4 before this instruction:

mov .w @R5 +, R6

A word is loaded from address 4 into R6 and the value in R5 is incremented to 6 because a word (2 bytes) was fetched. This is useful when stepping through an array or table, where expressions of the form *c++ are often used in C. This mode cannot be used for the destination. Instead the main instruction must use indexed mode with an offset of 0, followed by an explicit increment of the register by 1 or 2. The reverse of this move therefore needs two instructions:

mov .w R6 ,0(R5) ; store word from R6 into address 0+(R5)= 4 incd .w R5 ; R5 += 2

This is undoubtedly a bit clumsy. Autoincrement is usually called postincrement addressing because many processors have a complementary predecrement addressing mode, equivalent to *–c in C, but the MSP430 does not. An important feature of the addressing modes is that all operations on the rst address are fully completed before the second address is evaluated. This needs to be considered when moving blocks of memory. The move itself might be done by a line like this:

mov .w @ R 5 + ,0 x0100 (R5)

Suppose as usual that R5 initially contains the value 4. The contents of address 4 is read and R5 is double-incremented to 6 because a word is involved. Only now is the address for the destination calculated as 0x0100 + 0x0006 = 0x0106. Thus a word is copied from address 0x0004 to 0x0106; the offset is not just the value of 0x0100 used as the base address for the destination. The

compiler takes care of these tricky details if you write in C, but you are on your own with assembly language.

Draw the simplified block diagram of ADC10 and SD16

Ans:

Figure 1 and 2.

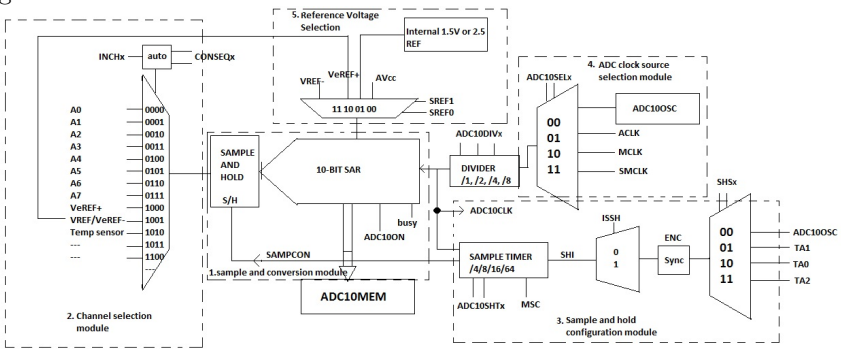


fig1:ADC10

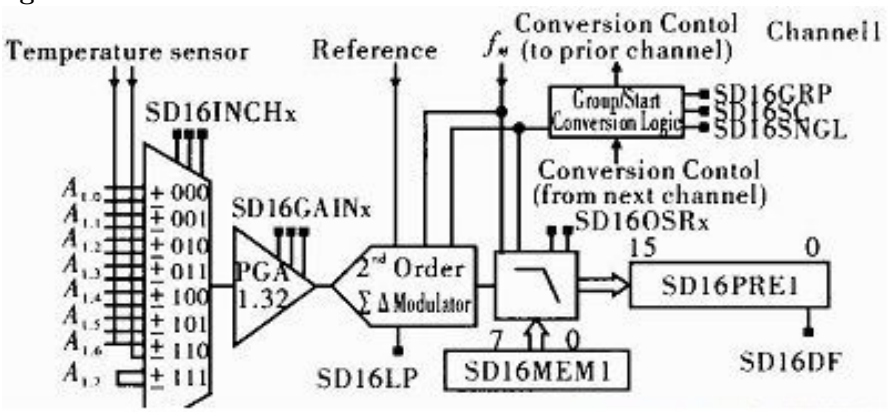


fig2:SD16

2)Explain the operation of Comparator_A+

Ans: The comparator can be used directly to compare a variable input voltage with a reference, which may be either one of the internal references or a second input. For example, the variable input might be from a temperature sensor and the comparator should detect when the equipment may freeze. Of course a microcontroller would be wasted on this task alone. It can be useful to connect a noisy input signal to the comparator so that the MSP430 detects when the input goes through a well-dened level, such as 0.5VCC. This is more reliable than the thresholds of the Schmitt trigger on the digital inputs. There is an example in the application note Ultrasonic Distance Measurement with the MSP430 (slaa136). A thermistor is a type of temperature sensor that is often used with the comparator. It is essentially a resistor made of a material whose resistance varies strongly with temperaturethe opposite of the property desired for a normal resistor. A wide variety of

devices is available. Some have a resistance that increases with temperature (positive temperature coefcient, or PTC), some the opposite (NTC). There is a broad selection of operating temperature, resistance, packaging, and accuracy. They are fairly cheap and easy to use because the change in resistance is large. Unfortunately it is also strongly nonlinear. A moderately accurate equation for a NTC thermistor is . B B .

R(T) = R0 exp(x)

T T0

, (9.1)

where T is the absolute temperature in kelvins. Thermistors are usually specied by their resistance R0 at a temperature T0, typically 25C. For example, a thermistor might be listed as having R0 = 10 kK at 25C with B = 3600 K. The equation shows that its resistance rises to about 30 kK at 0C and falls to 880 K at 100C. Thus the resistance changes by a factor of 30 between 0C and 100C, a conveniently large effect for a sensor. A simple detector for a single, xed temperature is shown in Figure 9.2(a). It is just a potential divider between VCC and ground with the midpoint connected to an input of the comparator (I show V+ but which one does not matter for this application). The other input of the comparator is connected to an internal reference VCAREF. The voltage from the potential divider is proportional to VCC in this circuit, so any changes in VCC affect V+. We do not want such variations to affect the temperature at which the output of the comparator changes. This can be achieved if the reference voltage on V changes in the same way. The internal references of VCC and VCC are derived from a potential divider and are therefore proportional to VCC, just like the potential divider with Another poor feature of these circuits is that their behavior is xed. For example, we can not change the temperature at which the output of the temperature sensor switches because it is determined by the resistance of the reference, Rref. This could be replaced by a potentiometer to make it adjustable but it would be far better to use the MSP430 to determine the value of Rth rather than whether it is larger or smaller than a single value. In other words, we want an analog-to-digital converter rather than a plain comparator. This goal can be achieved by adding a capacitor to form an

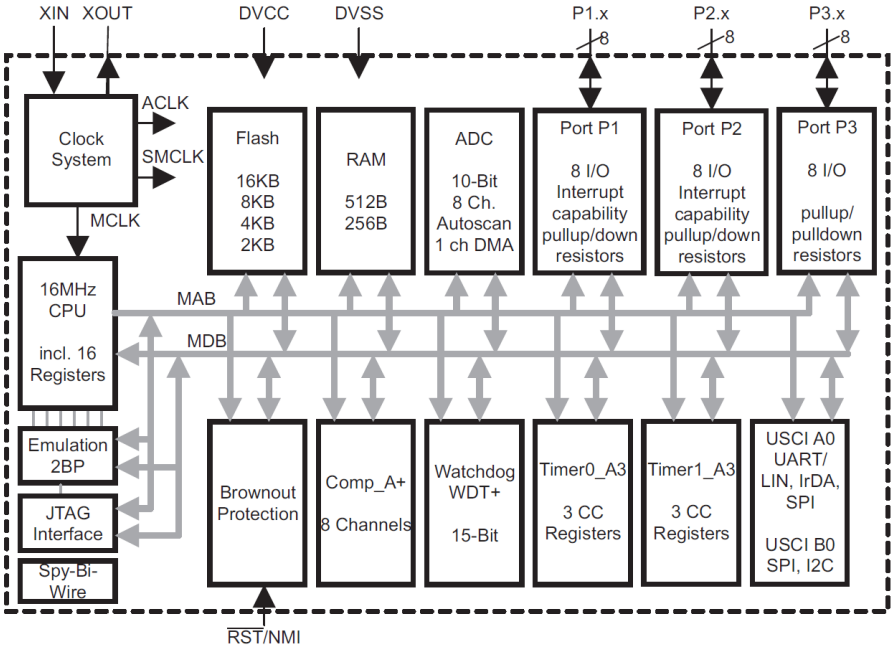
RC circuit and using a timer to measure its dynamic behavior. The operation can be performed in many ways:

A single transient of an RC circuit can be timed to determine R or C. Alternatively, the circuit can be made to oscillate and the frequency is measured rather than the duration of a single transient. An external voltage can be measured by comparing it with an RC transient. Alternatively, the capacitor is repeatedly charged and discharged through the resistor using rapid pulse-width modulation. The duty cycle is adjusted so that the voltage on the capacitor matches the input. This acts as a sort of integrating or sigmadelta ADC. I assume that the aim is to measure voltage or resistance but the same methods can be used for current or capacitance. The latter is growing in importance because of the burgeoning application of touch sensors. An RL circuit could be used to measure inductance but this has fewer practical applications. I pick out a couple of examples; others are described in the application notes An MSP430F11x1 SigmaDelta Type Millivoltmeter (slaa104) and Economic Measurement Techniques with the Comparator_A Module (slaa071). The code examples for the F20x1 contain a pretty example of a battery monitor, which compares 0.25 VCC with Vdiode. This sounds straightforward until you realize that both are reference voltages, which means that only one can be selected at a time. The trick is to use an external capacitor as a sample-and-hold circuit. It is rst charged to 0.25VCC by connecting it to the internal reference, then compared with Vdiode. A limitation is that only two comparisons are possible, Vdiode with either 0.25 VCC or 0.5 VCC.

3)Draw the functional block diagram of MSP430F2013 and briefly describe each region.

Ans:

The block diagram is:



The main blocks are linked by the memory address bus (MAB) and memory data bus (MDB). These devices have ash memory, 1 KB in the F2003 or 2 KB in the F2013, and 128 bytes of RAM. Six blocks are shown for peripheral functions (there are many more in larger devices). All MSP430s include input/output ports, Timer_A, and a watchdog timer, although the details differ. The universal serial interface (USI) and sigmadelta analog-to-digital converter (SD16_A) are particular features of this device. The brownout protection comes into action if the supply voltage drops to a dangerous level. Most devices include this but not some of the MSP430x1xx family. There are ground and power supply connections. Ground is labeled VSS and is taken to dene 0 V. The supply connection is VCC. For many years, the standard for logic was VCC = +5 V but most devices now work from lower voltages and a range of 1.83.6 V is specied for the F2013. The performance of the device depends on VCC. For example, it is unable to program the ash memory if VCC ¡ 2.2 V and the maximum clock frequency of 16 MHz is available only if VCC 3.3 V. TI uses a quaint notation for the power connections. The S stands for the source of a eld-effect transistor, while the C stands for the collector of a bipolar junction transistor, a quite different device. The MSP430, like most modern integrated circuits, is built using complementary metaloxidesilicon (CMOS) technology and eld-effect transistors. I doubt if it contains any bipolar junction transistors except possibly in some of the analog peripherals. There is only one pair of address and data

buses, as expected with a von Neumann architecture. Some addresses must therefore point to RAM and some to ash, so it is a good idea to explore the memory map next. The memory data bus is 16 bits wide and can transfer either a word of 16 bits or a byte of 8 bits. Bytes may be accessed at any address but words need more care. The address of a word is dened to be the address of the byte with the lower address, which must be even. Thus the two bytes at 0x0200 and 0x0201 can be considered as a valid word with address 0x0200, which may be fetched in a single cycle of the bus. This is shown in Figure 2.3. On the other hand, it is not possible to treat the two bytes at 0x0201 and 0x0202 as a single word because their address would be 0x0201, which is odd and therefore invalid. These two bytes straddle the boundary of two words. The memory system of some 16-bit processors can handle misaligned words like these, usually at the cost of an extra bus cycle, but the MSP430 does not. An important case is that instructions are composed of words and must therefore lie on even addresses.

4)Draw the memory map of MSP 430F2013 and describe each region.

Ans: The general memeory map is

Address	Type of memory
0xFFFF	interrupt and reset
0xFFC0	vector table
0xFFBF	flash code memory
0xF800	(lower boundary varies)
0xF7FF	
0x1100	
0x10FF	flash
0x1000	information memory
0x0FFF	*bootstrap loader*
0x0C00	(*not in F20xx*)
0x0BFF	
0x0280	
0x027F	RAM
0x0200	(upper boundary varies)
0x01FF	peripheral registers
0x0100	with word access
0x00FF	peripheral registers
0x0100	with byte access
0x000F	special function registers
0x0000	(byte access)

Random access memory: Used for variables. This always starts at address 0x0200 and the upper limit depends on the size of the RAM. The F2013 has 128 B.

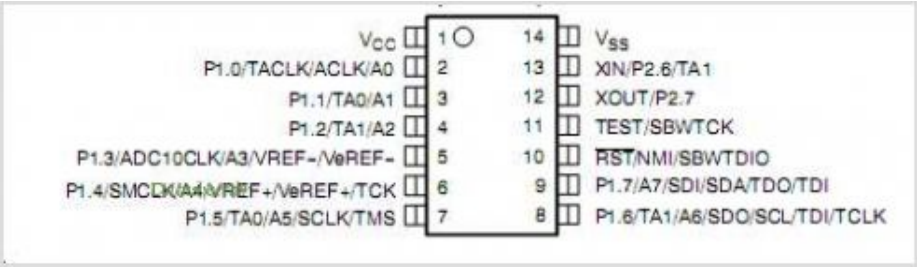
Bootstrap loader: Contains a program to communicate using a standard serial protocol, often with the COM port of a PC. This can be used to program the chip but improvements in other methods of communication have made it less important than in the past, particularly for development. Details are given in the application note Features of the MSP430 Bootstrap Loader (slaa089). All MSP430s had a bootstrap loader until the F20xx, from which it was omitted to improve security.

Information memory: A 256 B block of ash memory that is intended for storage of nonvolatile data. This might include serial numbers to identify equipmen-tan address for a network, for instanceor variables that should be retained even

when power is removed. For example, a printer might remember the settings from when it was last used and keep a count of the total number of pages printed. The information memory is laid out with smaller segments of ash than the code memory, which makes it more convenient to erase and rewrite. Segment A contains factory calibration data for the DCO in the MSP430F2xx family and is protected by default.
Code memory: Holds the program, including the executable code itself and any constant data. The F2013 has 2 KB but the F2003 only 1 KB. Interrupt and reset vectors: Used to handle exceptions, when normal operation of the processor is interrupted or when the device is reset. This table was smaller and started at 0xFFE0 in earlier devices. The range of addresses has been extended from 64 KB to 1 MB in the MSP430X. This means that addresses require 20 bits rather than 16 and the MAB is therefore 4 bits wider. The bottom 64 KB of memory from 0x00000 to 0x0FFFF is laid out in exactly the same way as in the original MSP430. The additional memory, from 0x10000 to 0xFFFFF, is available for additional ROM. This permits larger programs and tables to be stored.
Program counter, PC: This contains the address of the next instruction to be executed, points to the instruction in the usual jargon. Instructions are composed of 13 words, which must be aligned to even addresses, so the lsb of the PC is hard-wired to 0. Stack pointer, SP: When a subroutine is called, the CPU jumps to the subroutine, executes the code there, then returns to the instruction after the call. It must therefore keep track of the contents of the PC before jumping to the subroutine, so that it can return afterward. This is the primary purpose of the stack. Some processors use separate hardware for the stack but the MSP430 uses the top (high addresses) of the main RAM. The stack pointer holds the address of the most recently added word and is automatically adjusted as the stack grows downward in memory or shrinks upward.
Status register, SR: This contains a set of ags (single bits), whose functions fall into three categories. The most commonly used ags are C, Z, N, and V, which give information about the result of the last arithmetic or logical operation. The Z ag is set if the result was zero and cleared if it was nonzero, for instance. Decisions that affect the ow of control in the program can be made by testing these bits.
Setting the GIE bit enables maskable interrupts, which is explained in the section Interrupts on page 186. We do not use interrupts for the time being.
The nal group of bits is CPUOFF, OSCOFF, SCG0, and SCG1, which control the mode of operation of the MCU. All systems are active when all bits are clear. Setting various combinations of these bits puts the MCU into one of its low-power modes, which is described in the section Low-Power Modes of Operation on page 198. We keep the CPU active for now.
Constant generator: This provides the six most frequently used values so that they need not be fetched from memory whenever they are needed. It uses both R2 and R3 to provide a range of useful values by exploiting the CPUs addressing modes. This will be explained in the section Constant Generator and Emulated Instructions on page 131 but the compiler or assembler handles the details automatically.

5)Draw the pin-out of MSP430F2013 and describe the functions of each of the 14 pins:

Ans:



A0, A0+, and so on, up to A4, are inputs to the analog-to-digital converter. It has four differential channels, each of which has negative and positive inputs. VREF is the reference voltage for the converter.
ACLK and SMCLK are outputs for the microcontrollers clock signals. These can be used to supply a clock to external components or for diagnostic purposes.
SCLK, SDO, and SCL are used for the universal serial interface, which communicates with external devices using the serial peripheral interface (SPI) or inter-integrated circuit (I2C) bus.
XIN and XOUT are the connections for a crystal, which can be used to provide an accurate, stable clock frequency.
RST is an active low reset signal. Active low means that it remains high near VCC for normal operation and is brought low near VSS to reset the chip. Alternative notations to show the active low nature are _RST and /RST.
NMI is the nonmaskable interrupt input, which allows an external signal to interrupt the normal operation of the program.
TCK, TMS, TCLK, TDI, TDO, and TEST form the full JTAG interface, used to program and debug the device.
SBWTDIO and SBWTCK provide the Spy-Bi-Wire interface, an alternative to the usual JTAG connection that saves pins.
A less common feature of the MSP430 is that some functions are available at several pins rather than a single one. This applies to Timer_A in the F2013. Channel 0 (TA0) is brought out to pins shared with P1.1 and P1.5 but their functionality is not identical: P1.1 can be used for both a capture input and a compare output but P1.5 is available only for output. Similarly TA1 is shared with P1.2, P1.6, and P2.6. All can provide an output, but only P1.2 can be congured for input to the timer. The details are set out in the table of Terminal Functions in the data sheet and explained further in the section on Timer_A2.
One of the rst tasks in a program for a microcontroller is to congure the functions of each pin. All the pins can be used by a program except VCC and VSS for power and TEST/SBWTCK, which is reserved for debugging.