ST.ANNE'S COLLEGE OF ENGINEERING & TECHNOLOGY

EE8018 – MICROCONTROLLER BASED SYSTEM DESIGN

UNIT I INTRODUCTION TO PIC MICROCONTROLLER

Introduction to PIC Microcontroller–PIC 16C6x and PIC16C7x Architecture–PIC16cxx–-Pipelining - Program Memory considerations – Register File Structure - Instruction Set - Addressing modes – Simple Operations.

UNIT II INTERRUPTS AND TIMER

PIC micro controller Interrupts- External Interrupts-Interrupt Programming–Loop time subroutine - Timers-Timer Programming– Front panel I/O-Soft Keys– State machines and key switches– Display of Constant and Variable strings.

UNIT III PERIPHERALS AND INTERFACING

I2C Bus for Peripherals Chip Access– Bus operation-Bus subroutines– Serial EEPROM—Analog to Digital Converter–UART-Baud rate selection–Data handling circuit–Initialization - LCD and keyboard Interfacing - ADC, DAC, and Sensor Interfacing.

UNIT IV INTRODUCTION TO ARM PROCESSOR

ARM Architecture –ARM programmer's model –ARM Development tools- Memory Hierarchy –ARM Assembly Language Programming–Simple Examples–Architectural Support for Operating systems.

UNIT V ARM ORGANIZATION

3-Stage Pipeline ARM Organization– 5-Stage Pipeline ARM Organization–ARM Instruction Execution- ARM Implementation– ARM Instruction Set– ARM coprocessor interface–Architectural support for High Level Languages – Embedded ARM Applications.

TEXT BOOKS:

1. Peatman,J.B., "Design with PIC Micro Controllers"PearsonEducation,3rdEdition, 2004.
2. Furber,S., "ARM System on Chip Architecture" AddisonWesley trade Computer Publication, 2000.

UNIT I
INTRODUCTION TO PIC MICROCONTROLLER

INTRODUCTION

Simplicity and ease, which the higher programming languages bring for program writing as well as broader application of the microcontrollers, was enough to incite some companies as Micro engineering to embark on the development of BASIC programming language. What did we thereby get? Before all, the time of writing was shortened by employment of prepared functions that BASIC brings in (whose programming in assembler would have taken the biggest portion of time). In this way, the programmer can concentrate on solving the essential task without losing his time on writing the code for LCD display. To avoid any confusion in the further text, it is necessary to clarify three terms one encounters very often.

Programming language is understood as a set of commands and rules according to which we write the program and therefore we distinguish various programming languages such as BASIC, C, PASCAL etc. On the BASIC programming language the existing literature is pretty extensive so that most of the attention in this book will be dedicated to the part concretely dealing with the programming of microcontrollers.

Program consists of sequence of commands of language that our microcontroller executes one after another.

BASIC compiler is the program run on PC and it's task is to translate the original BASIC code into the language of 0 and 1 understandable to the microcontroller.

BASIC for PIC microcontrollers:

As a programming language, BASIC is since long time ago known to the PC users to be the easiest and the most widespread one.

Nowadays this reputation is more and more being transferred onto the world of microcontrollers. PIC BASIC enables quicker and relatively easier program writing for PIC microcontrollers in comparison with the *Microchip's* assembling language MPASM. During the program writing, the programmer encounters always the same problems such as serial way of sending messages, writing of a variable on LCD display, generating of PWM signals etc. All for the purpose of facilitating programming, PIC BASIC contains itsbuilt -in commands intended for solving of the problems often encountered inpraxis. As far as the speed of execution and the size of the program are concern, MPASM is in small advantage in respect with PIC BASIC (therefore exists the possibility of combining PIC BASIC and assembler). Usually, the part of the program in which the same commands are executed many times or time of the execution critical, are written in assembler. Modern microcontrollers such as PIC execute the instructions in a single cycle lasting for 4 tact of the oscillator. If the oscillator of the microcontroller is 4MHz, (one single tact lasts 250nS), then one assembler instruction requires 250nS x 4 = 1uS for the execution. Each BASIC command is in effect the sequence of the assembler instructions and the exact time necessary for its execution may be obtained by simply summing up the times necessary for the execution of assembler instructions within one single BASIC command.

PIC microcontrollers

The creation of PIC BASIC followed the great success of Basic stamp (small plate with PIC16F84 and serial eeprom that compose the whole microcontroller system) as its modification. PIC BASIC enables the programs written for the original Basic stamp to be translated for the direct execution on the PIC16xxx, PIC17Cxxx and PIC18Cxxx members of the microcontrollers family. By means of PIC BASIC it is possible to write programs for the PIC microcontrollers of the following families PIC12C67x, PIC14C000, PIC16C55x, PIC16C6x, PIC16C7x, PIC16x84, PIC16C9xx, PIC16F62x, PIC16C87x, PIC17Cxxx and PIC 18Cxxx. On the contrary, the programs written in PIC BASIC language cannot be run on the microcontrollers possessing the hardware stack in two levels as is for example the case of PIC16C5x family (that implies that by using the *CALL* command any subroutine can be called not more than two times in a row).

For the controllers that are not able to work with PIC BASIC there is an adequate substitution. For example, instead of PIC16C54 or 58, we can use pin compatible chips PIC16C554, 558, 620 and 622 also operating with PIC BASIC without any difference in price.

Currently, the best choice for application development, using PIC BASIC are microcontrollers from the family : PIC16F87x, PIC16F62X and of course the famous PIC16F84. With this family of PIC microcontrollers, program memory is created using FLASH technology which provides fast erasing and reprogramming, thus allowing faster debugging. By a single mouse click in the programming software, microcontroller program can be instantly erased and then reloaded without removing chip from device. Also, program loaded in FLASH memory can be stored after power supply has been turned off. The older PIC microcontroller series (12C67x, 14C000, 16C55x, 16C6xx, 16C7xx and 16C92x) have program memory created using EPROM/ROM technology, so they Basic for PIC Microcontrollers 5 can either be programmed only once (OTP version with ROM memory) or have glass window (JW version with EPROM memory), which allows erasing by few minutes exposure to UV light. OTP versions are usually cheaper and are used for manufacturing large series of products. Besides FLASH memory, microcontrollers of PIC16F87x and PIC16F84 series also contain 64-256 bytes of internal EEPROM memory, which can be used for storing program data and other parameters when power is off. PIC BASIC has built -in READ and WRITE instructions that can be used for loading and saving data to EEPROM. In order to have complete information about specific microcontroller in the application, you should get the appropriate Data Sheet or Microchip CD-ROM.

PIC Microcontrollers

PIC stands for Peripheral Interface Controller coined by Microchip Technology to identify its single chip microcontrollers. These devices have been phenomenally successful in 8-bit microcontroller market.

The main reason is that Microchip Technology has constantly upgraded the device architecture and added needed peripherals to the microcontroller to ’suit customers’ requirements. The development tools such as assembler and simulator are freely available on the internet.

Low-end Architectures

Microchip PIC microcontrollers are available in various types.

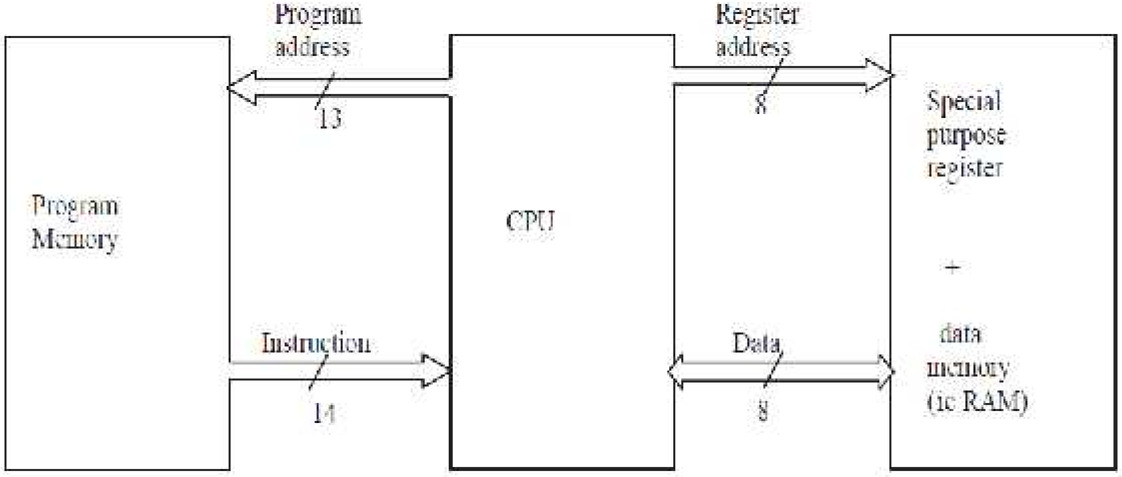
When PIC became available from General Instruments in early 1980’s, the microcontroller consisted of a very simple processor executing 12-bit wide instructions with basic I/O functions. These devices are known as low-end architectures. Some of the low-end device past numbers are 12C5XX, 16C5X, and 16C505

Mid-range Architectures

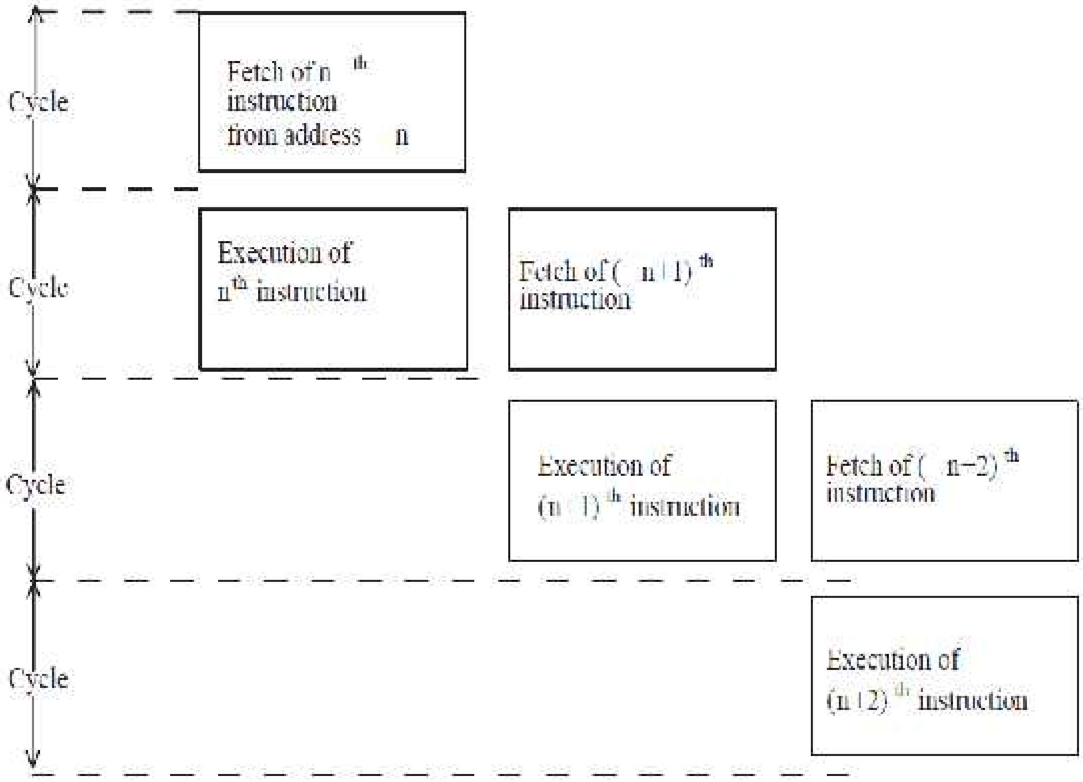
Mid-range Architectures are built by upgrading low-end architecture with more number of peripherals, more numbers of register and more data memory. Some of the mid-range devices are 16C6X 16C7X, 16F873 Program memory type
C = EPROM
F = Flash
RC = Mask ROM

Popularity of PIC microcontrollers is due to the following factors:
1. Speed: Harvard Architecture, RISC Architecture
1 instruction Cycle = 4 clock cycles.
For 20 MHz clock, most of the instructions are executed in 0.2*μs* or five instructions per microsecond.
2. Instruction Set Simplicity:
The instruction set consists of just 35 instructions (as opposed to 111 instructions for 8051)
3. Power on reset
Power-out reset
Watch-dog timer
Oscillator Options
• low-power Crystal
• Mid-range Crystal
• High-range Crystal
• RC Oscillator
4. Programmable timer options on chip ADC
5. Up to 12 independent interrupt sources
6. Powerful output pin control 25mA (max.) current sourcing capability.
7. EPROM/OTP/ROM/Flash memory options.
8. Free assembler and simulator support from microchip.

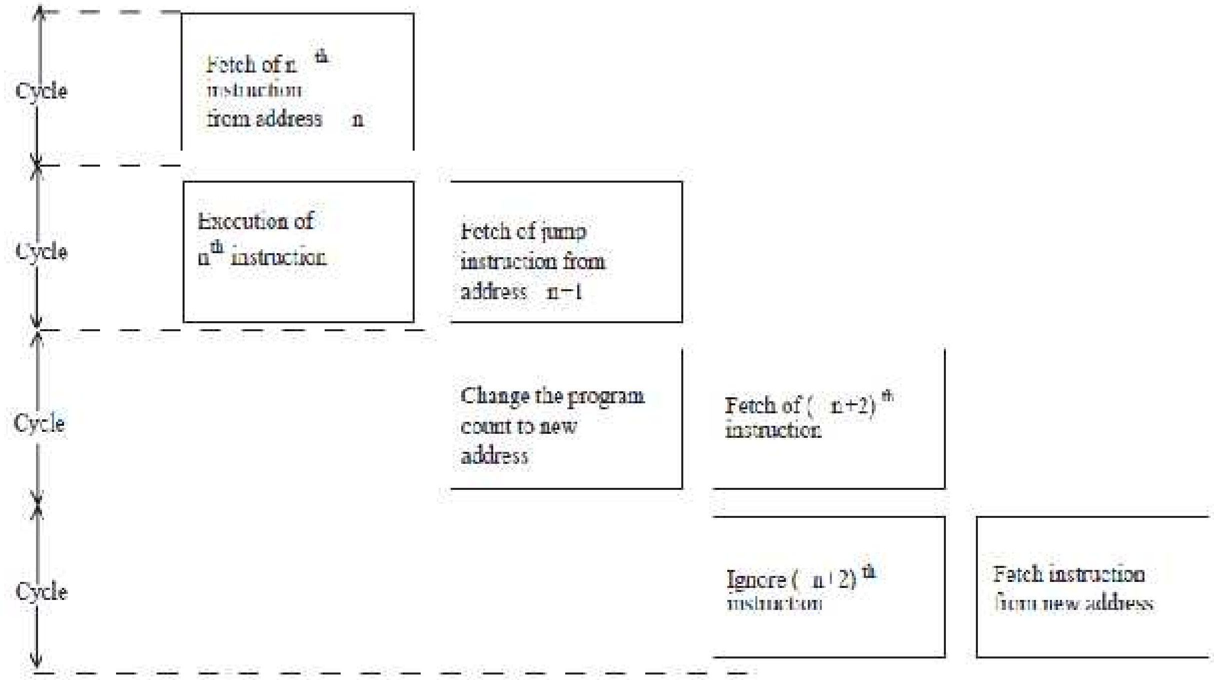
CPU Architecture and Instruction Set



Pipelining of instruction fetch successive addressing


Cycle
Fetch of n th instruction from address n
Cycle
Execution of n th instruction
Fetch of (n+1) th instruction
Cycle
Execution of (n+1) th instruction
Fetch of (n−2) th instruction
Cycle
Execution of (n+2) th instruction

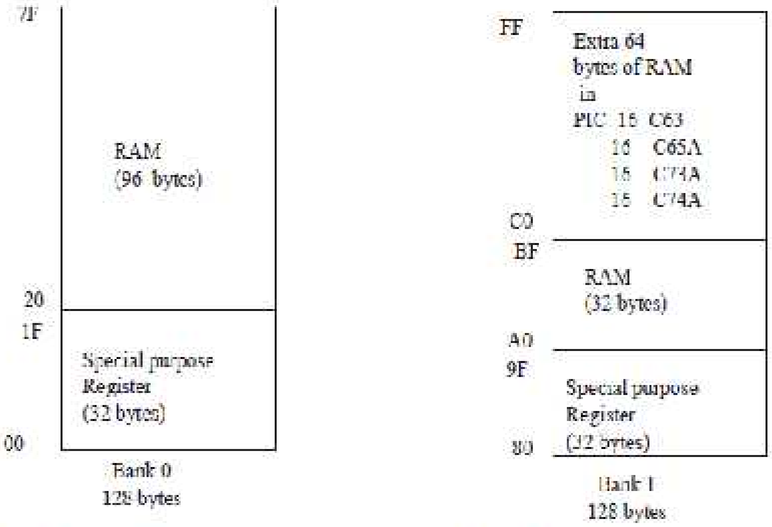
Introduction of extra cycle for a jump/goto instruction


Cycle
Fetch of n th instruction from address n
Cycle
Execution of n th instruction
Fetch of jump instruction from address n−1
Cycle
Change the program count to new address
Fetch of (n+2) th instruction
Cycle
Ignore (n+2) th instruction
Fetch instruction from new address

Register File Structure and Addressing Modes

Register file → locations that an instruction can access via an address.
Register file consists of two components.

1. General purpose register file (same as RAM)
2. Special purpose register file



RPO bit in the Status register detects the bank. 7 bit of direct address TRPO determines the absolute address of the register.
Indirect addressing mode
FSR contains the 8-bit address of the data/register.

CPU Registers

W, the working register, is used by many instructions as the source of an operand. It may also serve as the destination for the result of the instructions execution. It works as the accumulator.

W working register

7							0

STATUS
(address 03H,83H)

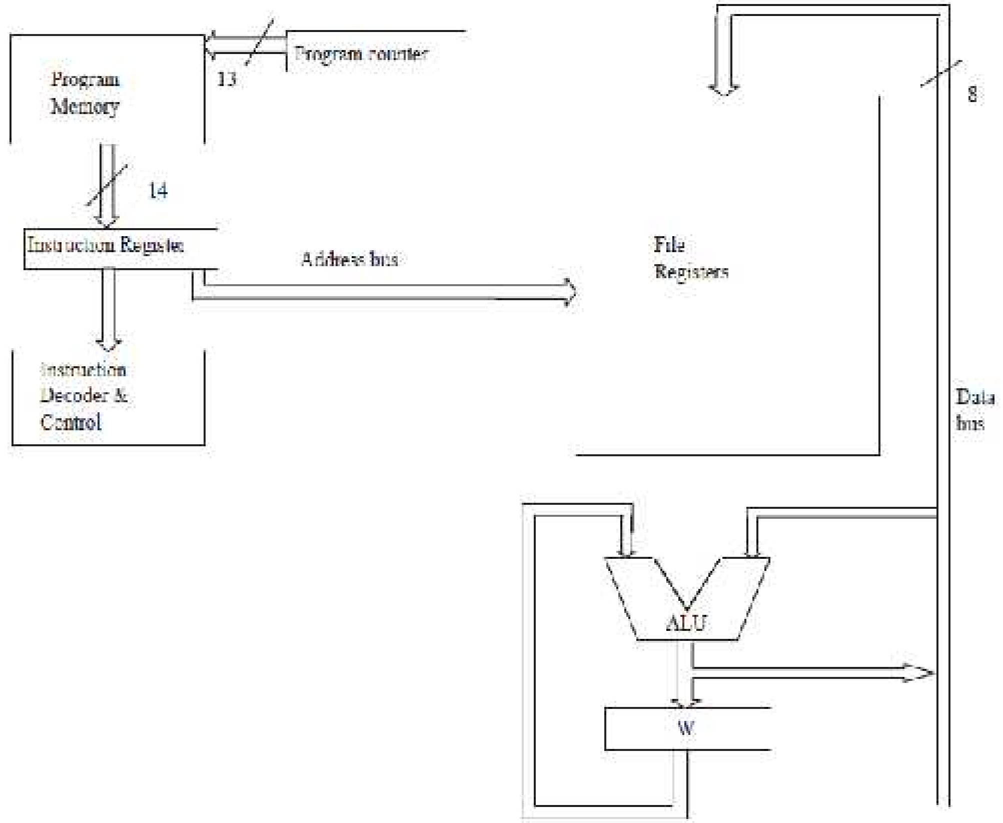
7	6	5	4	3	2	1	0
0	0	RP0	NOT_TO	NOT_PD	Z	DC	C

C = Carry bit
DC = Digit Carry (same as AC, Auxiliary Carry)
Z = Zero bit
NOT-TO, NOT-PD→ Used in conjunction with PIC's sleep mode
RPO → register bank select bit used in conjunction with the direct addressing mode.

Indirect data memory address points.

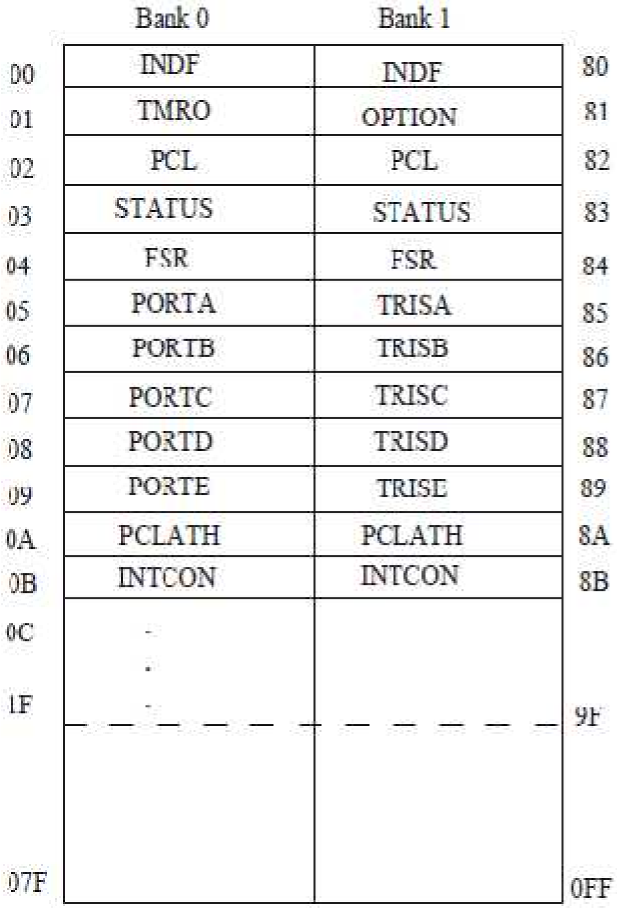
FSR is the pointer used for indirect addressing. The program is supported by an eight-level stack. When an interrupt occurs, the program counter is automatically pushed on to the stack. Since PIC microcontrollers programs are normally designed for handling one interrupt at a time, further

Basic Architecture of PIC Microcontroller



W → Temporary holding register, often called as an accumulator, cannot be accessed directly. Instead, contents must be moved to other registers that can be accessed directly.

Bank Addressing



	Bank 0	Bank 1	
00	INDF	INDF	80
01	TMRO	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07	PORTC	TRISC	87
08	PORTD	TRISD	88
09	PORTE	TRISE	89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C 1F	- . -		9F
07F			0FF

TRIS bit set → Post bit in I mode
Reset → Post bit in 0 mode.

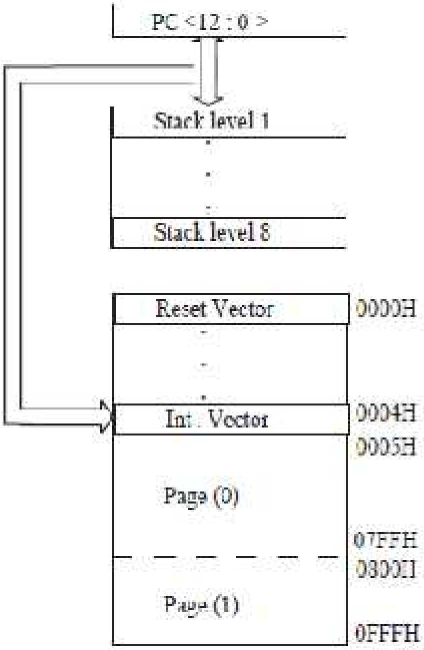
Memory Organization

The PIC 16C7X family has a 13-bit program counter capable of addressing 8k×14 program memory.

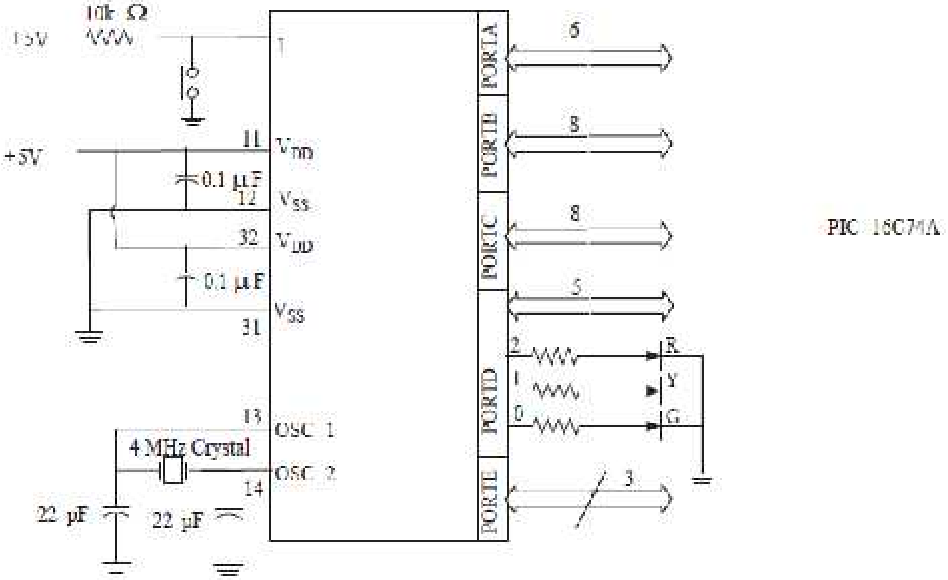
PIC16C74A has 4k×14 program memory. For those devices with less than 8k program memory, accessing a location above the physically implemented address will cause a wrap around.

Program memory map and stack

16C74A has 4k program memory. The address range is 0000H - 0FFFH. The reset vector is 0000H and the interrupt vector is 0004H.



LED Driver Example



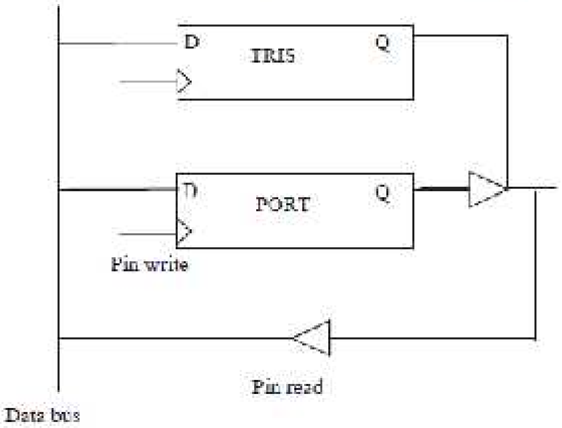
PIC 16C74A has five ports. Each port is a bidirectional I/O port. In addition, they have the following alternative functions.

PIC 16C74A has five ports. Each port is a bidirectional I/O port. In addition, they have the following alternative functions.

Port	Alternative uses of I/O pins	I/O pins	
		61A 65A 74A	62A 63A 73A
PORTA	A/D Converter inputs (PIC 16C7X parts)	6	6
PORTB	External interrupt inputs	8	8
PORTC	Serial port, Timer I/O	8	8
PORTD	Parallel slave port	8	0
PORTE	A/D Convertor inputs (PIC 16C 7X)	3	0
	Total I/O pins	33	22
	Total pins	40/44	28

Port D alternative function is parallel slave port which enables one PIC microcontroller to be connected to the data bus of another microprocessor. Since three LED's are connected to three pins of Port D to be used as normal I/O operation, the special alternative function is ruled out.

Structure of a port - pin of PIC microcontrolller



TRIS register controls the direction of data flow.
TRIS = 1 Sets the pin in the input mode.
TRIS = 0 Sets the pin in the output mode.

```
; Toggle the green LED every half second.
    List        P = PIC16C74A,       F = INHX8M,       C = 160,       N = 80
            ST = FF,       MM = OFF, R = DEC
    include "C:\MPLAB\P16C74A.INC"
      config ( CP OFF &  PWRTE ON & XT OSC &  WDT OFF &  BODEN OFF)
    error level -302
```

```
; Equates
    Bank0 RAM   equ     20H
    MaxCount   equ   50
    Green   equ   0000000HB
    TenMsH   equ   13
    TenMsL   equ   250
; Variables
    cblock   Bank0RAM      ;    Variables are declared
    BLNKCNT
    COUNTH
    COUNTL
    endc
; Vectors
    org      000H
    goto      Mainline
    org      004H
Stop:
    goto stop
Mainline:
```

```
   call   Initial       ; Initialize
Main loop:
   call   Blink       ; Blink LED
   call   TenMs       ; Inset ten millisecond delay
indent goto   Mainloop
;Initial Subroutine
   Initial:
      movlw   MaxCount       0.5 second
      movwf   BLNKCNT       BLKCNT ← N
      movlw   Green
      movwf   PORTD       PORTD ←   W
      bsf   STATUS,RPO       Set register access to bank 1
      clrf   TRISD       Set PORTD as O/P port
      bcf   STATUS,RPO       Set register access to bank 0
      return
; Blink Subroutine.  This subroutine blinks a green LED in evey 0.5 sec
   Blink:
      decfsz   BLNKCNT,F   ;   decrement loop counter and return if not zero
      goto   BlinkEnd
      movlw   MaxCount   ;   Reinitialize BLNKCNT
      movwf   BLNKCNT
      movlw   GREEN   w ← Green   Toggle green LED
      Xorwf   PORTD,F   w ← Green   Toggle green LED
   Blink End;
      return
; Ten Ms subroutine (delay of 10ms)
   Ten Ms:
      nop
```

Instructions		Instruction Cycles
Call ten Ms		2
nop		1
movlw 13 (TenMsH)		1
movwf COUNTH		1
movlw 250 (Ten MsL)		1
movwf COUNTL		1
decfsz COUNTL,F goto Ten_1	COUNTL:250 →249 →... →1	3 × 249 = 747
decfsz COUNTL,F	COUNTL: 1 → 0	2
decfsz COUNTH,F	COUNTH: 13 → 12	1
goto Ten_1		2
decfsz COUNTL,F goto Ten_1	COUNTL: 0→255→254...→1	
	255 × 3 = 765	
decfsz COUNTL,F	COUNTL:1 →0	2
decfsz COUNTH,F	COUNTH: 12 →11	1
goto Ten_1		2
		770
		770 ×11=8470
	Repeat this block 11 times as COUNTH:12→11→...→2→1	
decfsz COUNTL,F	COUNTL:0→255→...2→1	3×255=765
goto Ten_1		
decfsz COUNTL,F	COUNTL:1→0	2
decfsz COUNTH,F	COUNTH:1→0	2

UNIT II
INTERRUPTS AND TIMER

Overview of Timer Modules

Timer-0 Overview

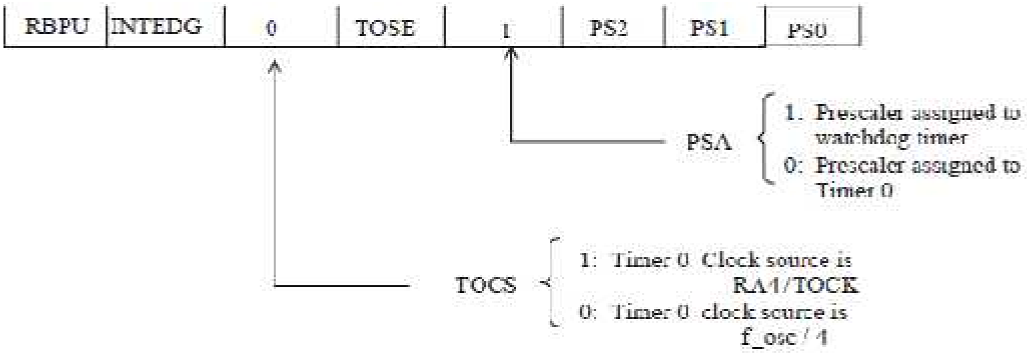
The Timer 0 module is a simple 8-bit overflow counter. The clock source can be either the internal clock (*fosc*/4) or an external clock. When the clock source is an external clock, the Timer 0 module can be selected to increment on either the rising or falling edge.

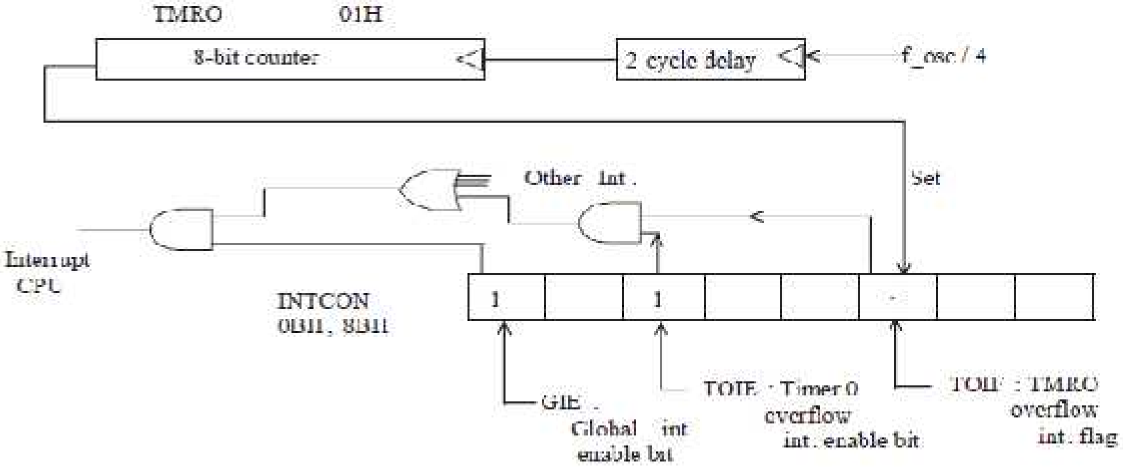
Timer-0 module also has a programmable prescalar option. This prescalar can be assigned either to Timer 0 or the watchdog Timer.

The counter sets a flag TOIF when it overflows and can cause an interrupt at that time if that interrupt source has been enabled (TOIF=1). Timer 0 can be assigned an 8-bit prescalar that can divide the input by 2,4,8,16,...,256. Writing to TMRO resets the prescalar assigned to it.

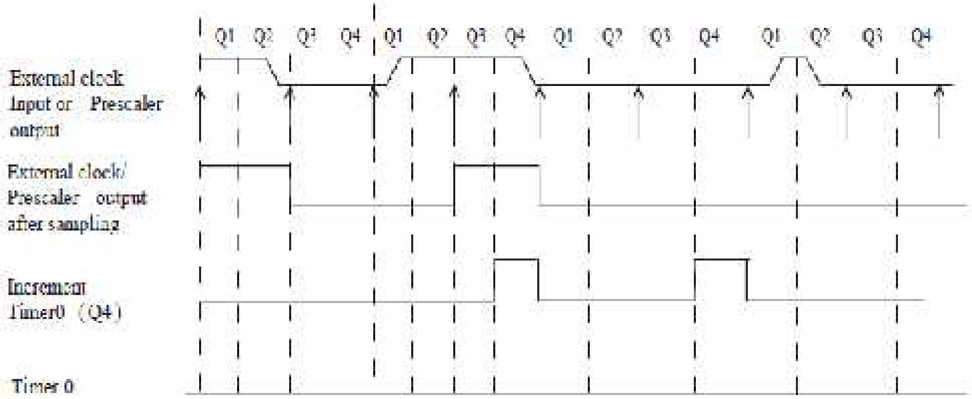
Timer-0, or its prescalar can be connected to either of two input sources.

1. *fosc*/4
2. RA4/ TOCKI, the input connected to bit 4 of PORTA.


RBPU
INTEDG
0
TOSE
1
PS2
PS1
PS0
PSA
1. Prescaler assigned to watchdog timer
0: Prescaler assigned to Timer 0
TOCS
1: Timer 0 Clock source is RA4/TOCK
0: Timer 0 clock source is f_osc / 4

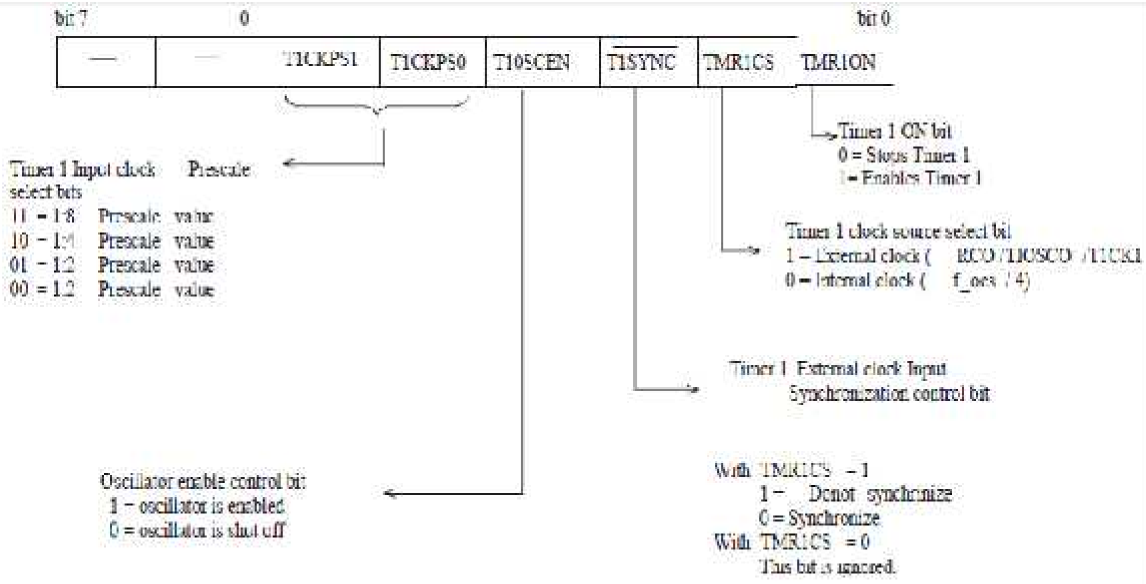

TMRO
01H
8-bit counter
2 cycle delay
f_osc / 4
Set
Other Int .
Interrupt CPU
INTCON
0BH , 8BH
1
1
GIE - Global int enable bit
TOIE : Timer 0 overflow int . enable bit
TOIF : TMRO overflow int . flag

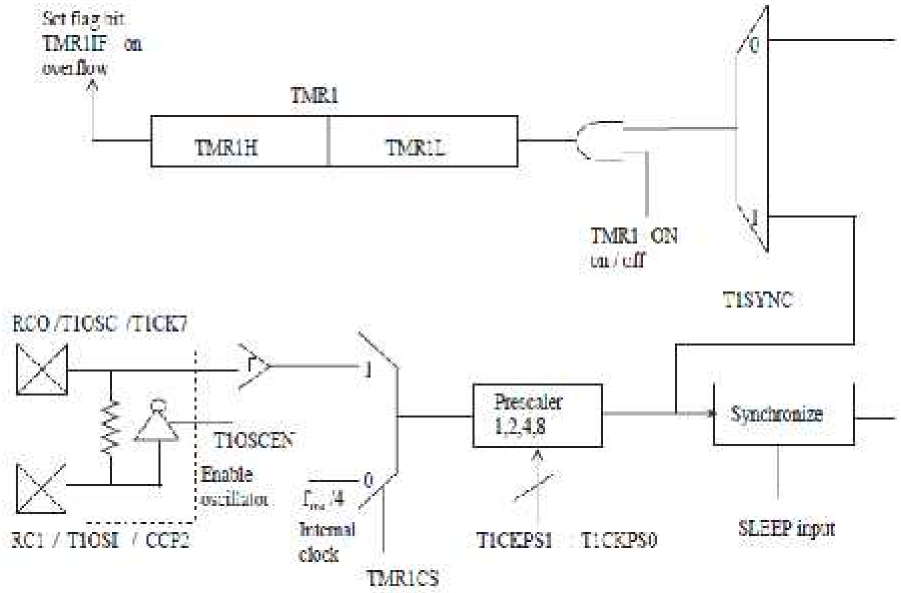
External clock synchronization


Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
External clock Input or Prescaler output
External clock/ Prescaler output after sampling
Increment Timer0 (Q4)
Timer 0

Timer-1 Module

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H: TMR1L) increments from 0000H to FFFFH and rolls over to 0000H. The TMR1 interrupt, if enabled, is generated on overflow which sets the interrupt flag bit TMR1IF-(PIR< 0 >). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE-(PIE < 0 >)
The operating and control modes of Timer 1 is determined by the special purpose register T1CON.
T1CON (10H)





Timer 1 can operate in one of the two modes.

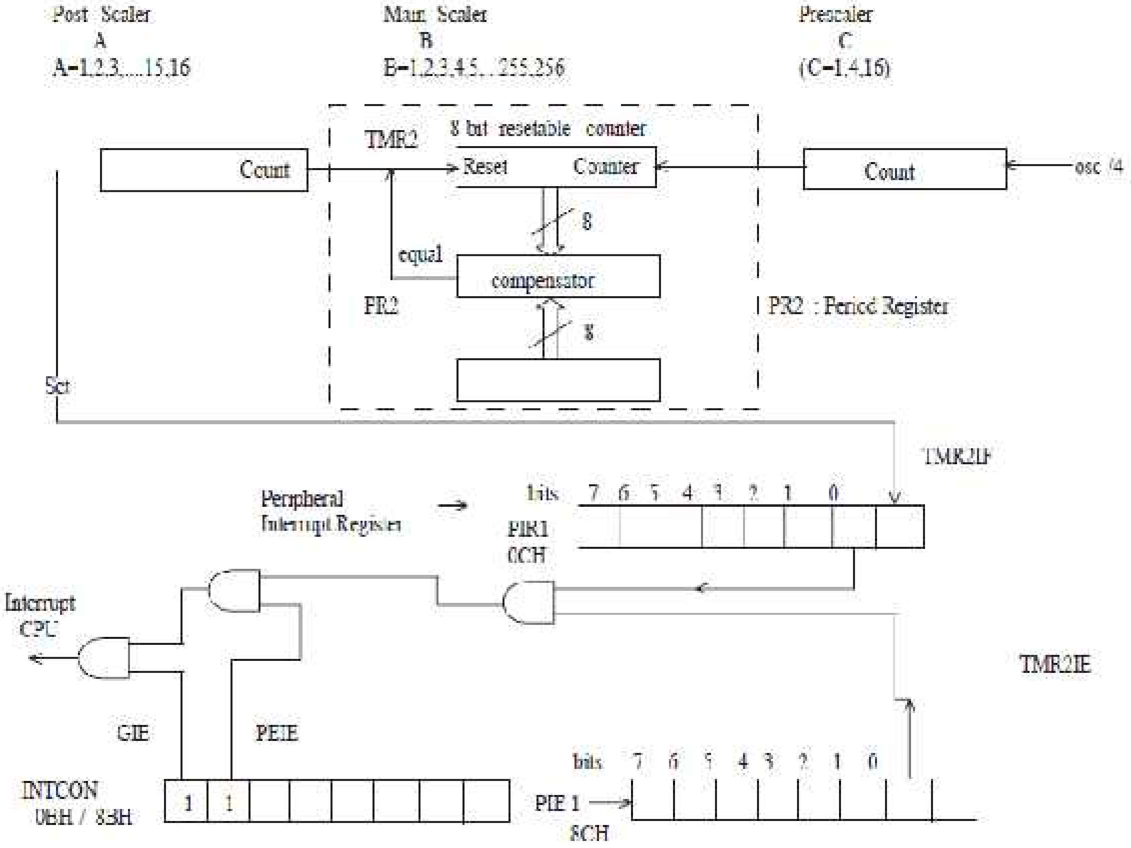
- As a timer. (TMR1CS = 0)
  In timer mode, Timer 1 increments in every instruction cycle. The Timer 1 clock source is $f_{osc}/4$. Since the internal clock is selected, the timer is always synchronized and there is no further need of synchronization.
- As a counter (TMR1CS = 1)

In counter mode, external clock input from the pin RC0/T1OSC/T1CKI is selected.

Use of Timer-2

Timer 0: 8 bit timer/counter with 8 bit prescalar
Timer 1: 16-bit timer/counter with prescalar, can be incremented during sleep
via external crystal/clock.
Timer 2: 8-bit timer/counter with 8-bit period register, prescalar, post scalar.

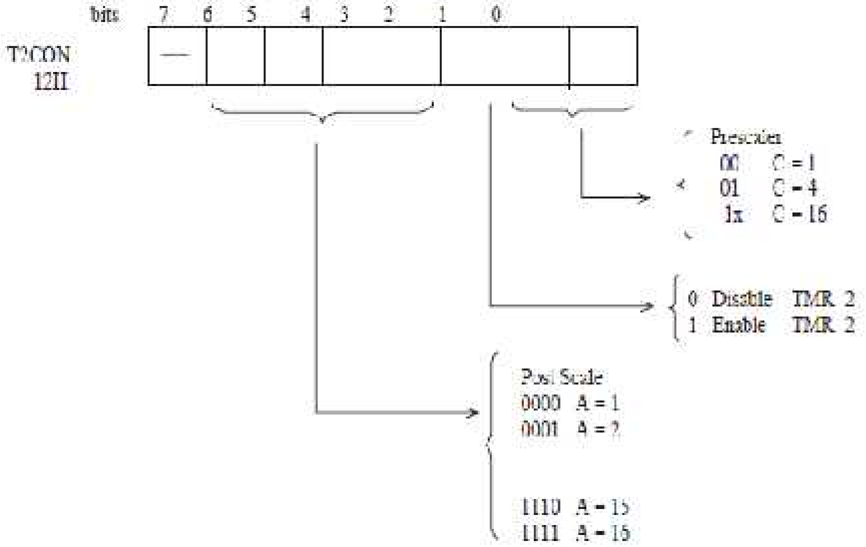
Timer 2 Circuitry



Timer 2 is an 8-bit timer with a prescalar and a port sclar. It can be used on the PWM mode of CCP modules. The TMR2 register is readable and writable and is cleared on any device reset. The input clock ($f_{osc}/4$) has a prescalar option of 1:1, 1:4 or 1:16 selected by bits 0 and 1 of T2CON register.

The timer 2 module has a 8-bit period register (PR2). timer 2 increments from 00H until it matches PR2 and then resets to 00H on the next increment cycle. PR2 is a readable and a writable register. PR2 is initialized to FFH on reset.

The output of TMR2 goes through a 4-bit post scalar (1:1, 1:2 to 1:16) to generate a TMR2 interrupt by setting TMR2IF flag.



CCP overview

The CCP module(s) can operate in one of the three modes: 16 bit capture, 16 bit compare, or upto 1-bit Pulse Width Modulation (PWM).

Capture mode captures the 16-bit value of TMR1 into CCPRxH: CCPRxL register pair. The capture event can be programmed for either the falling edge, rising edge, fourth rising edge, or the sixteenth rising edge of the CCPx pair.

Compare mode compares the TMR1H: TMR1L register pair to the CCPRxH: CCPRxL register pair. When a match occurs an interrupt can be generated, and the output pin CCPx can be forced to given state (High or Low), TMR1 can be reset (CCP1) or TMR1 reset and start A/D conversion (CCP2). This depends on the control bits < CCPxM3 : CCPxM0>

PWM mode compares the TMR2 register to a 10 bit duty cycle register (CCPRxH : CCPRxL< 5:4 >) as well as an 8-bit period register (PR2). When the TMR2 register= Duty cycle register, the CCPx pin will be forced low. When TMR2=PR2, TM2 is cleared to 00H, an interrupt can be generated, and the CCPx pin, if programmed in the O/P mode, will be forced high.

Compare Mode

Timer 1 is a 16-bit counter which can be used with CCP (Capture/compare/PWM) module to drive a pin high or low at precisely controlled time, independent of what the CPU is doing at that time. The pins are Port-C RC1/CCP2 and RC2/CCP1 pins.

Which Timer1 includes a prescalar to divide the internal clock by 1,2,4 or 8, the choice of divide-by-one gives the finest resolution in setting the time of an output edge.

Capture/Compare/PWM modules

Each CCP (Capture/compare/PWM) module contains a 16-bit register which can operate as a 16-bit capture register, as a 16-bit compare register or as a PWM master/slave duty cycle register. Both CCP1 and CCP2 are identical in operation, with the exception of the operation of the special event trigger.
The following shows the CCP mode timer resources.

CCP Mode	Timer Resource
Capture	Timer 1
Compare	Timer 1
PWM	Timer 2

CCP1 Module:

Capture/Compare/PWM Register 1 consists of two 8-bit register: CCPR1L (low byte) and CCPR2H (high byte). THe CCP1CON register controls the operation of CCP1. All are readable and writable.

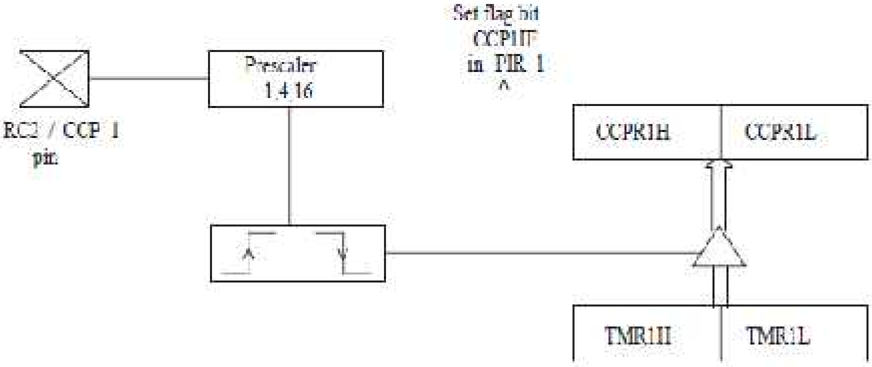
CCP2 Module:

Capture/Compare/PWM Register 2 consists of two 8-bit registers: CCPR2L (low byte) and CCPR2H (high byte). The CCP2CON register controls the operation of CCP2. Al are readable and writable.

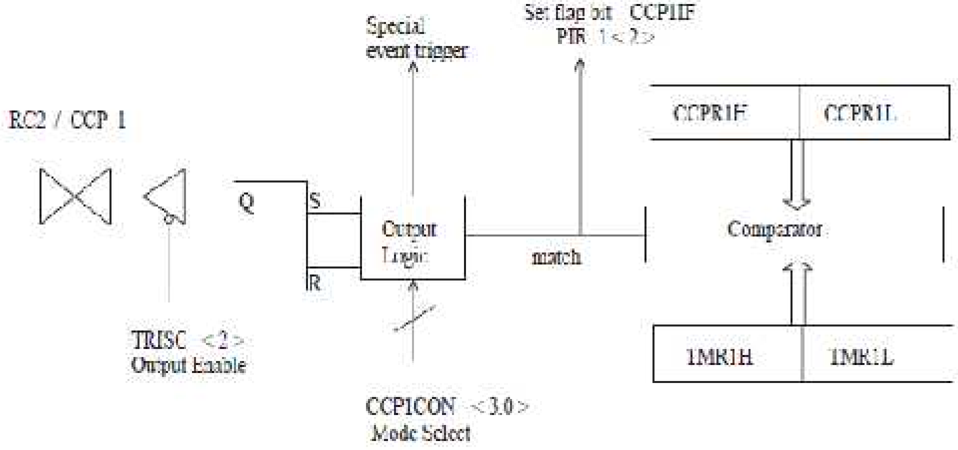
CCP1CON Register / CCP2CON Register

bit 7							bit 0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0

bit 5-4: CCPxX : CCPxY : PWM Least Significant bits.
Capture mode : Unused
Compare mode : Unused
PWM mode : These bits are the two LSBs of the PWM duty cycle. The eight MSBs are found in CCPRxL.
bit 3-0: CCPxM3 : CCPxM0 : CCPx Mode select bits.
Capture Mode



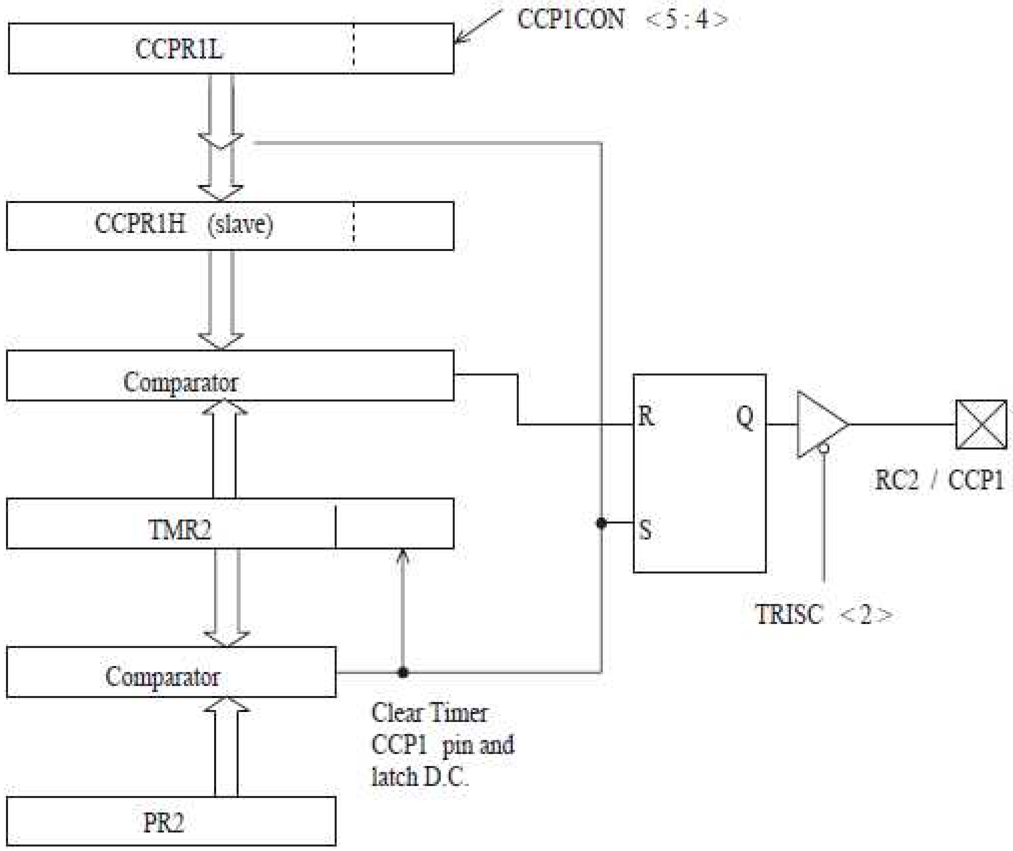
Compare Mode



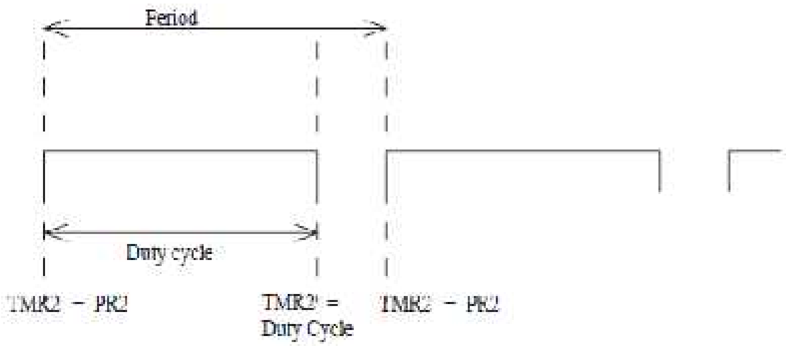
PWM Mode

In Pulse WIdth Modulation (PWM) mode, the CCPx pin produced upto a 10-bit resolution PWM output. Since CCP1 pin is multiplexed with PORT C data latch, the TRISC < 2 > pin must be cleared to make CCP1 pin an output.

Simplified PWM Block Diagram



PWM Output



A PWM output as ashown has a time period. The time for which the output stays high is called duty cycle.

PWM Period

The PWM period is specified by writing to PR2 register. The PWM period can be calculated using the following formula:

$$\text{PWM period} = [(PR2) + 1] \times 4 \times T_{osc} \times (\text{TMR2 prescale value})$$

$$\text{PWM frequency} = 1/ \text{ PWM period}$$

When TMR2 is equal to PR2, the following events occur on the next increment cycle.

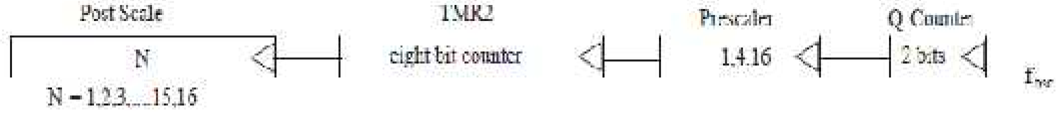
- TMR2 is cleared
- the CCP1 pin is set (if PWM duty cycle is 0
- The PWM duty cycle is latched from CCPR1L into CCPR1H

PWM duty cycle

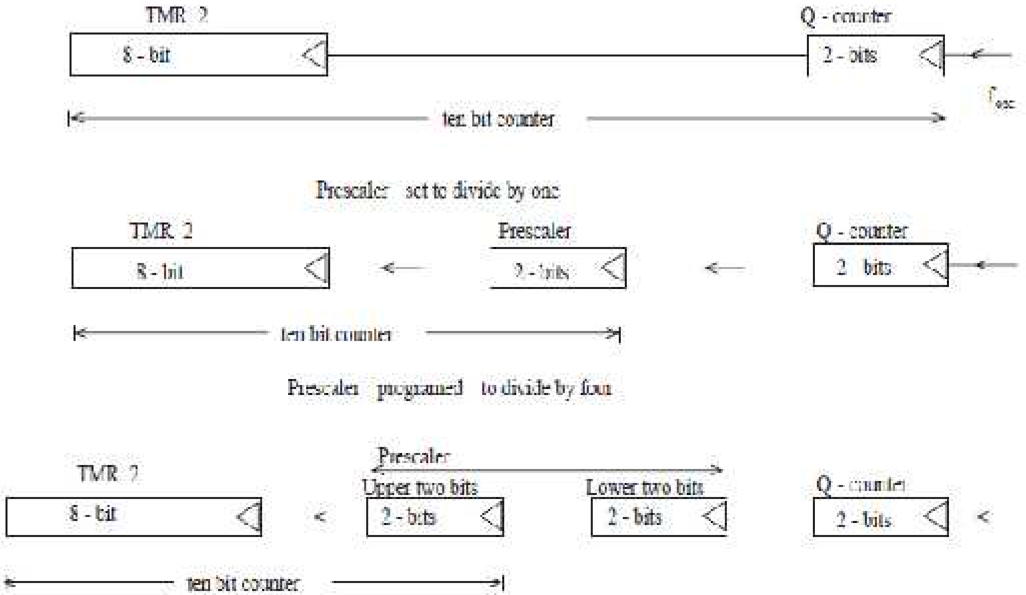
The PWM duty cycle is specified by writing to the CCPR1L register and to CCP1CON $< 5 : 4 >$ bits. Up to 10-bit resolution is available where CCPR1L contains the eight MSBs and CCP1CON $< 5 : 4 >$ contains the two LSB's. The 10 bit value is represented by CCPR1L : CCP1CON $< 5 : 4 >$. The PWM duty cycle is given by

PWM duty cycle = (CCPR1L : CCP1CON $< 5 : 4 >$). T_{osc} . (TMR2 prescale value)

Although CCPR1L and CCP1CON $< 5 : 4 >$ can be written to at anytime, the duty cycle value is not latched



Counting mechanism in Timer 2.



Prescaler set to divide by one

Prescaler programed to divide by four

into CCPR1H until a match between PR2 and TMR2 occurs. In PWM mode, CCPR1H is a read-only register.
The CCPR1H register and a 2-bit internal latch are used to double buffer the PWM duty cycle. This double buffering is essential for glitchless PWM operation. When the CCPR1H and 2-bit latch match TMR2 concatenated with an internal 2-bit Q clock or 2-bits of prescalar, the CCP1 pin is cleared.
Maximum PWM resolution (bits) for a given PWM frequency can be calculated as

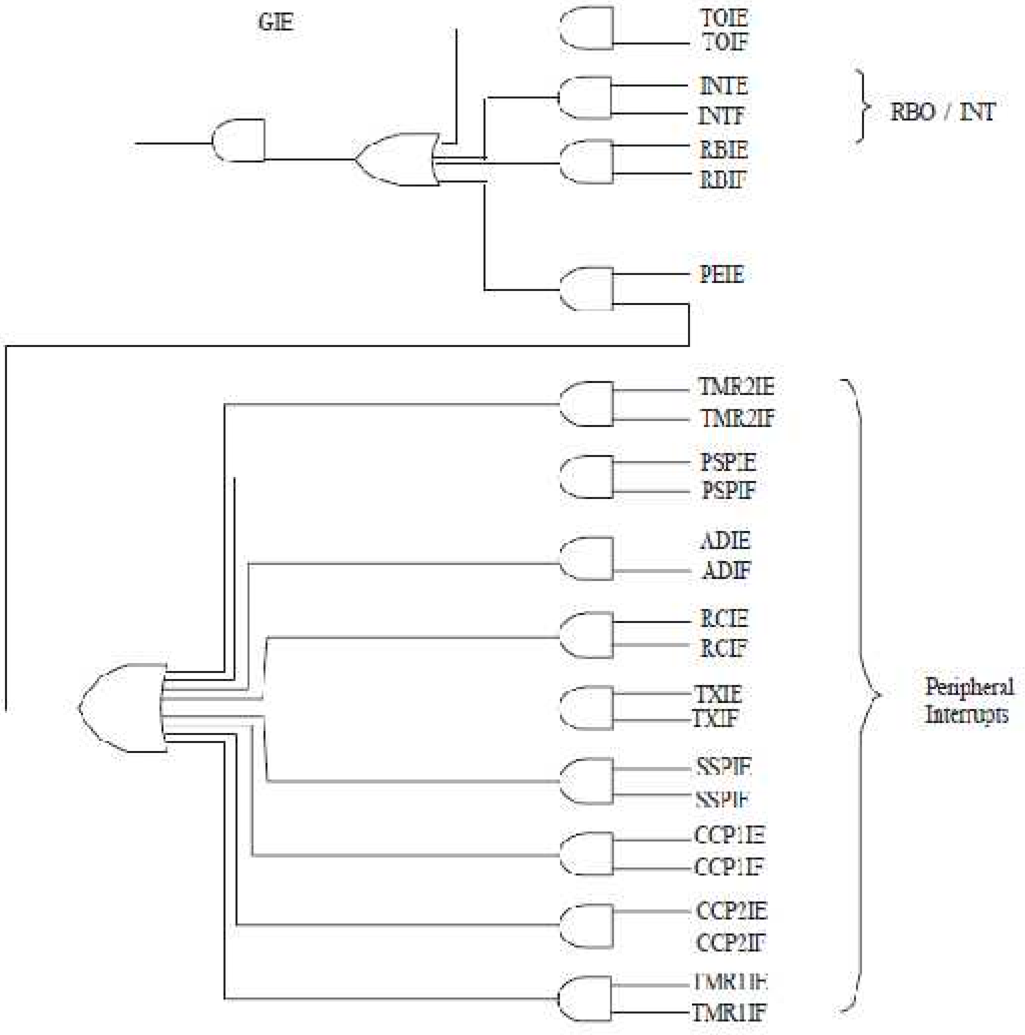
$$\frac{\log(\frac{f_{osc}}{f_{PWM}})}{\log_2}$$

If the PWM duty cycle is longer than the PWM period, then the CCP1 pin will not be cleared.

PWM Period and duty cycle calculation

Example Desired PWM frequency = 78.125 kHz
$f_{osc} = 20\text{MHz}$
TMR2 Prescalar = 1
$\frac{1}{78.125 \times 10^3} = (PR2+1)4 \times \frac{1}{20 \times 10^6}$ PR2 = 63
Find the maximum resolution of duty cycle that can be used with a 78.124 kHz frequency and 20 MHz oscillator.
$\frac{1}{78.125 \times 10^3} = 2^{\text{PWM Resolution}} \cdot \frac{1}{20 \times 10^6} \cdot 1$
$256 - 2^{\text{PWM Resolution}}$
PWM Resolution = 8
At most, an 8-bit resolution duty cycle can be obtained from a 78.125 kHz frequency and 20 MHz oscillator, ie, 0 < CCPR1L : CCP1CON < 5 : 4 >< 255. Any value greater than 255 will result in a 100°/o duty cycle. The following table gives the PWM frequency f_{PWM} if f_{osc}=20MHz

Interrupt Logic

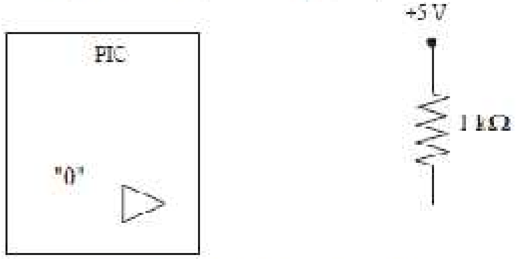


Four of PORTB’s pins RB7 : RB4 have an interrupt on change feature. Only pins configured on inputs can cause this interrupt to occur. The input pins (of RB7 : RB4) are compared with the old values on the last read of Port B. the ”mismatch” outputs of RB7 : RB4 are used together to generate the RB port change interrupt flag bit RB1F.

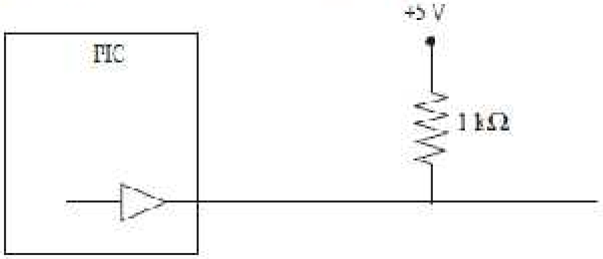
UNIT III
PERIPHERALS AND INTERFACING

I^2C Bus for Peripheral Chip Access

Requires two open-drain I/O pins.
Port-C of PIC IC can be used for I^2C communication.
SCL (Serial Clock) RC3/SCK/SCL
SDA (Serial Data) RC4/SDI/SDA



Low output on SCL or SDA I/O pin set to be an output with "0" written to it.



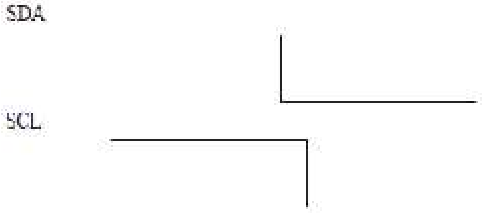
High output on SCL or SDA I/O pin set to be an input.
Transfers on the I^2C bus take place a bit at a time.

while every other chip on the I^2C bus is a receiver. During the acknowledgment bit time, the addressed chip is the only one that drives the SDA line, pulling it low in response to the masters pulse on SCL, acknowledging the reception of its chip address.

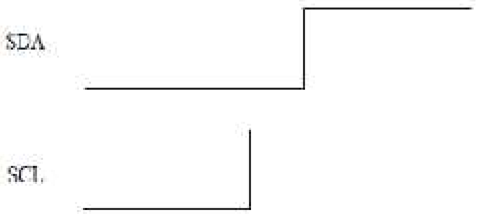
When the data transfer direction is reversed that is form a peripheral chip to the PIC, which is the master , the peripheral chip drives the eight daa bits in response to the clock pulse from PIC. In this case, the acknowledge bit is driven in a special way by the PIC, which is serving as receive but also as bus master. If the peripheral chip is one that can send the contents of successive internal address back to the PIC, then PIC completes the reception of each byte and signals a request for the next byte by pulling *SDA line low* in acknowledgment. After any number of bytes have been received by the master from the peripheral, the PIC can signal the peripheral to stop any further transfers by not pulling the SDA line low in acknowledgment.

SDA line should be *stable during high period of the clock (SCL).* When the slave peripheral is driving SDA line , either as transmiter or acknowledge, it initiates the new bit in response to the falling edge of SCL, after a specified time. It maintains that bit on SDA line until the next falling edge of SCL, again afte r a specified hold time.

I^2C bus transfers consist of a number of byte transfers framed between a START condition and either another START condition or a STOP condition. Both SDA and SCL lines are released by all drives and *float high* when bus transfers are not taking place. The PIC (I^2C bus controller) initiates a transfer with a START condition by first pulling SDA low and then pulling SCL as shown in the figure.



START Condition



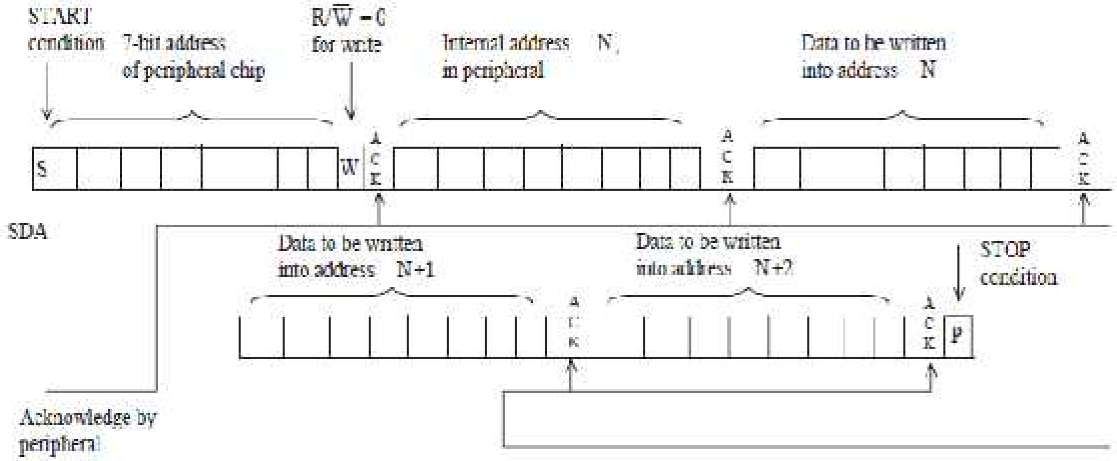
STOP Condition

Similarly, the PIC terminates a multiple byte transfer with the STOP condition. With both SDA and SCL initially low, it first releases SCL and then SDA. Both then occurrences are easily recognized by I^{2}C hardware in each peripheral chip since they both consist of a chage in SDA line which SCL is high, a condition that never happens in the middle of a byte transfer.

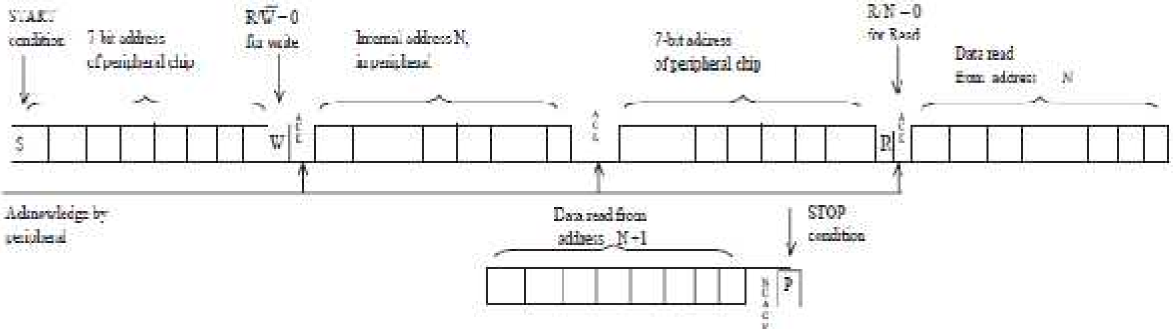
Data Communication protocol

In I^{2}C communication standard, there is one bus master and several slaves. It can be assumed here that the PIC microcontroller is the bus master and several peripheral devices connected to SDA and SCL bus are slaves.
Following a start condition, the master sends a 7-bit address of the slave on SDA line. The MSB is sent first. After sending 7 bit address of the slave peripheral a R/W bit (8^{th} bit) is sent by the master. If R/$\overline{W}$ bit is 0 the following byte (after the acknowledgment) is written by the master to the addressed slave peripheral. If R/$\overline{W}$ bit is 1, the following byte after the acknowledgment bit has to be read from the slave by the master. After sending the 7 bit address of the slave, the master sends the address of the internal register of the salve where from the data has to be used or written to. The subsegment access is automatically directed to the next address of the internal register.

The following diagrams give the general format to write and read from several peripheral internal registers.

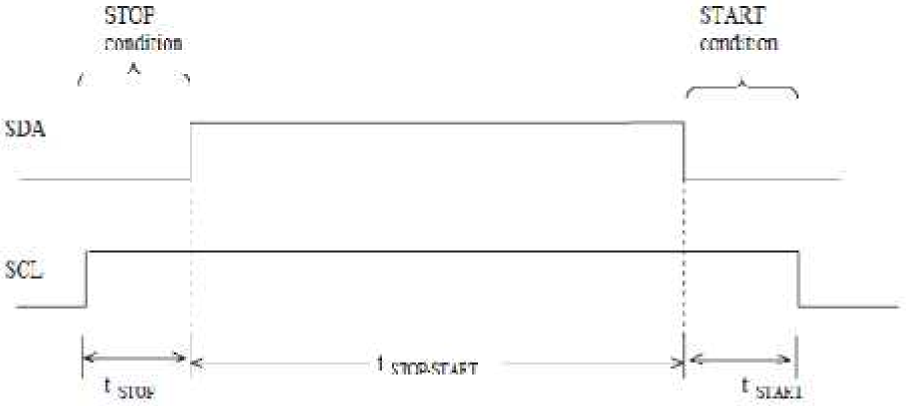

START condition
7-bit address of peripheral chip
R/$\overline{W}$ = 0 for write
Internal address N, in peripheral
Data to be written into address N
S
W
ACK
ACK
ACK
SDA
Data to be written into address N+1
Data to be written into address N+2
STOP condition
ACK
ACK
P
Acknowledge by peripheral

General format to write to several peripheral internal registers or addresses.

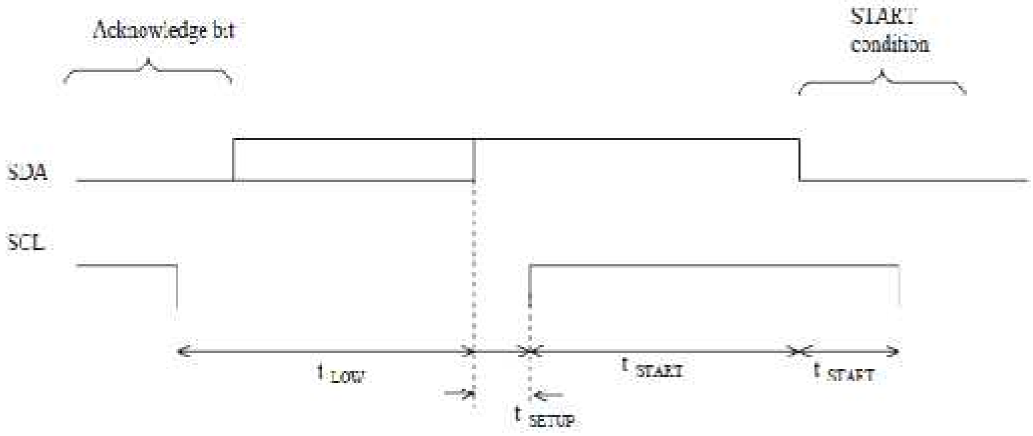

START condition
7-bit address of peripheral chip
R/$\overline{W}$ = 0 for write
Internal address N, in peripheral
7-bit address of peripheral chip
R/N = 0 for Read
Data read from address N
S
W
ACK
ACK
R
ACK
Acknowledge by peripheral
Data read from address N+1
STOP condition
NACK
P

I^{2}C Bus Subroutines:

I^{2}C bus fast-mode timing constraints.



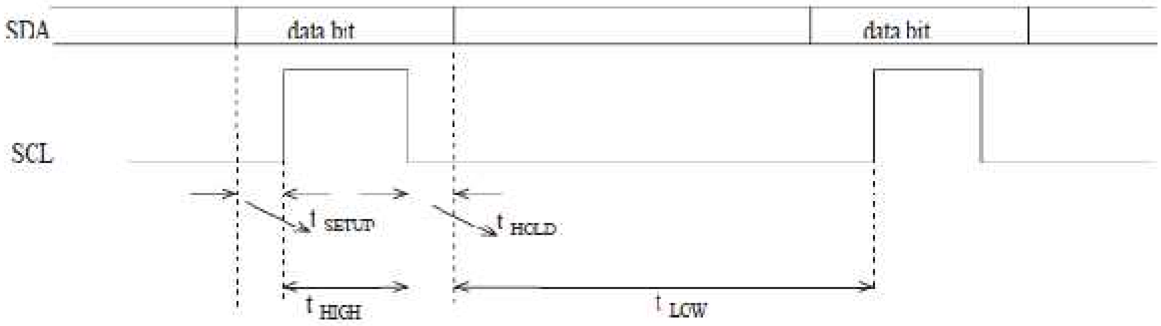
STOP - to - START Constraints



Acknowledge bit to START (restart condition)

t_{SETUP}

Acknowledge bit to START (restart condition)



Data bit to data bit

Parameter	Constraint	Cycles required to meet constraint		
		ocs = 4MHz Period − 1μs	osc = 10MHz Period − 0.4μs	osc = 20MHz Period − 0.2μs
t_{START}	> 0.6μ s	1	2	3
t_{SETUP}	> 0.1μ s	1	1	1
t_{HIGH}	> 0.6μ s	1	2	3
t_{HOLD}	> 0μ s	1	1	1
t_{LOW}	> 1.3μ s	2	4	7
t_{STOP}	> 0.6μ s	1	2	3
$t_{STOP-START}$	> 1.3μ s	2	4	7

Because the SCL pin must have an open pin output which the SDA pin must be either an input or have an open drain output, the I^{2}C subroutines will repeatedly access TRISC, the data direction register for PORTC, However, TRISC is located at the bank 1 address 87H, which cannot be accessed by direct addressing without changing RPO bit to 1.
bsf STATUS, RPO
Then required bit of TRISC can be changed followed by clearing RPO and reveting back to Bank 0.
bcf STATUS, RPO
Instead of doing this, the indirect pointer FSR can be loaded with the address of TRISC and the bit setting and bit clearing of TRISC can be done indirectly.
For example, with the following definitions

```
   SCL equ 3
   SDA equ 4
bsf INDF, SDA
```

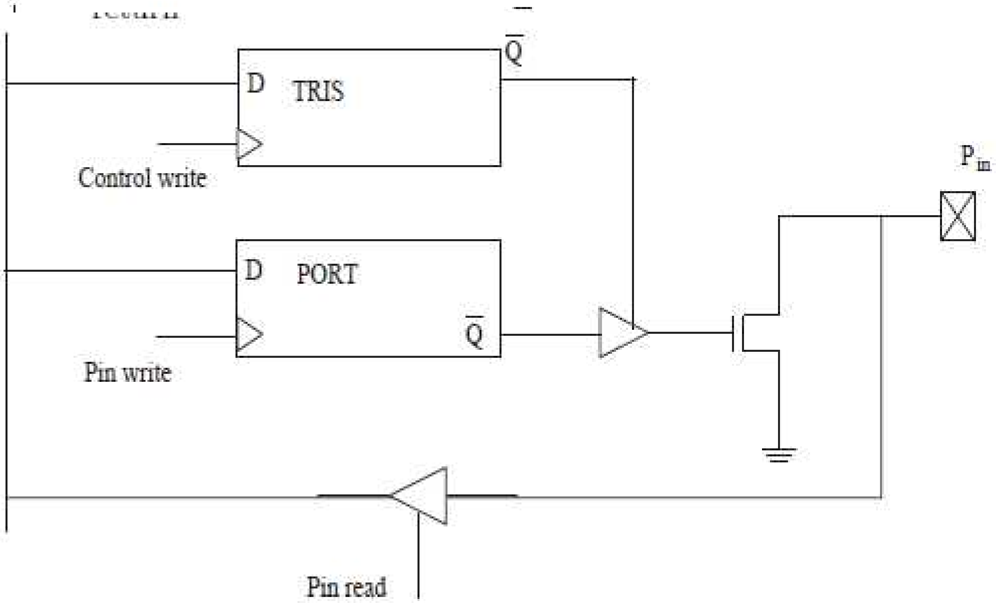
will release the SDA line, letting the external pull up register pull it high or some I^{2}C chp pull it low. When FSR is used for indirect addressing, care should be taken to restore FSR value when a subroutine is completed and the program returns to the mainline program.

I2C Subroutines

```
Freq          equ   4
SDA           equ   4
SCL           equ   3


cblock
.
.
DEVADD                  ;The I2Cout subroutine transfers out three bytes:
INTADD
DATAOUT
DATAIN
TXBUFF
RXBUFF
.
.
endc
;DEVADD, INTADD, and DATAOUT
```

```
I2C out :
        call start
        movf DEVADD, W ; Send peripheral address with R/W̄=0 (write)
        Call Tx
        movf INTADD, W
        Call Tx
        movf DATAOUT, W
        Call Tx
        Call Stop          ; Generate Stop condition
        return
; The I2C in subroutine transfers out DEVADD (with R/W̄=0)
; and INTADD, restarts, transfers out DEVADD (with R/W=1)
; and read one byte back into DATAIN.

I2C in:
        Call Start                ; Generate start condition
        movf DEVADD, W           ; Send peripheral address    R/W̄=0 (write)
        Call Tx
        movf INTADD, W           ; Send peripheral's internal address
        Call Tx
        Call ReStart              ; Re START
        movf DEVADD ,W            ; Send peripheral's address.
        iorlw 0000000.1 B        ; with R/W̄=1 (read)
        Call Tx
        bsf TXBUFF, 7            ; NOACK the following reading of one byte
        Call Rx                  ; Read byte
        movwf DATAIN            ; inte DATAIN
        Call stop               ; Generate stop condition
        return

; The start subroutine initializes the I2C bus and then
; generates the START condition on the I2C bus
; The ReStart entry point bypadd the initialization of the
; I2C bus

Start:
        movlw 00111011      ; Enable I2C Master mode.
        movwf SSPCON
        bcf PORTC, SDA      ; DRIVE SDA low when it is an output
        bcf PORTC, SCL      ; DRIVE SCL low when it is an output
        movlw TRISC         ; Set indirect pointer to TRISC
        movwf FSR
ReStart:
        bsf INDF, SDA            ; Make sure SDA is high - I/P mode
        bsf INDF ,SCL          ; Make sure SCL is high - I/P mode
        delay 0,1,2 not
        bcf INDF ,SDA           ; Make SDA low
        delay 0,1,2 nop
        bcf INDF, SCL          ; Make SCL low
        return

Stop:
        bcf INDF, SDA           ; Return SDA low
        bsf INDF, SCL          ; Drive SCL high
        delay 0,1,2
        bsf INDF, SDA           ; and then drive SDA high
        return

; The Tx subroutine sends out the byte passed to it in W.
; It returns with z = 1 if ACK occurs.
; It returns with z = 0 if NOACK occurs.

Tx:
        movwf TXBUFF
        bsf STATUS, C
```

```
Tx_1:
        rlf TXBUFF, F        ;   rotate TXBUFF left, through carry
        movf TXBUFF, F    ;   Set Z bit when all 8 bits have been transformed
        btfss STATUS, Z     ;   until z = 1
        Call Bitout    ;   Send carry bit then clear carry bit
        btfss STATUS, Z
        goto TX_1
        Call Bit In
        movlw 00000001 B
        End wf RXBUFF, W    ; z = 1 if ACK    z = 0 if NOACK
        return

; The Rx subroutine receives a byte from I2C bus into W,
; using RXBUFF buffer
; Call Rx with bit 7 of TXBUFF clear for ACK
; Call Rx with bit 7 of TXBUFF set for NOACK

Rx:
        movlw 00000001 B
        movwf RXBUFF
Rx_1:
        rlf RXBUFF, F
        Call Bit In
        btfss STATUS, C
        goto Rx_1
        rlf TXBUFF, F
        Call BitOut
        movf RXBUFF, W
        return
; The BitOut subroutine transmits, hthen clears, the carry bit
BitOut:
        bcf INDF, SDA          ; copy carrybit to SDA
        btfsc STATUS, c
        bcf INDF, SDA
        bsf INDF, SCL          ; pulse clockline
        delay 0,1,2            ; t: HIGH
        bcf INDF, SCL
        bcf STATUS, c
        return
; The bit In subroutine receives one bit into
; bit 0 of RXBUFF

BitIn:
        bsf INDF, SDA
        bsf INDF, SCL           ; Drive clock line high
        bcf RXBUFF, 0           ; copy SDA to bit 0 of RXBUFF
        btfsc PORTC, SDA
        bsf RXBUFF, 0
        bcf INDF, SCL           ; Drive clock low again
        return
```



Examples of I2C bus Interfacing

I. DAC Interfacing

Two digital to analog conveste outputs are easily added to a PIC with MAX518 eight pin DIP. Each output channel produces an output voltage that ranges from 0v to $\frac{255}{256}V_{DD}$ where V_{DD} is the power supply to the DAC chip. If $V_{DD} = 5V$, an output of 2.5V will appear on the OUT0 pin if the following three bytes are sent to the chip.

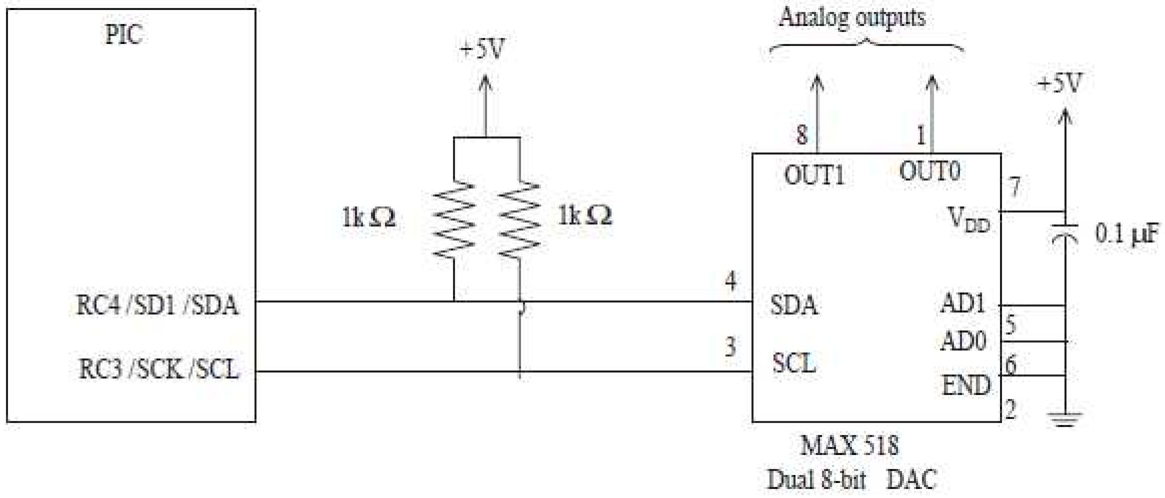
'01011000', '00000000' '10000000'

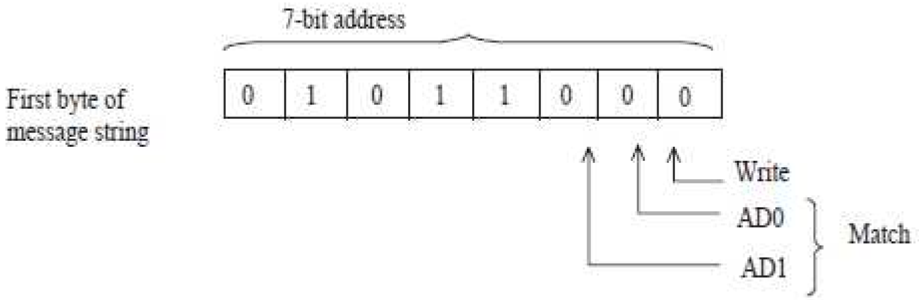
An output of 2.5 V will appear on the OUT1 pin by sending the following three bytes

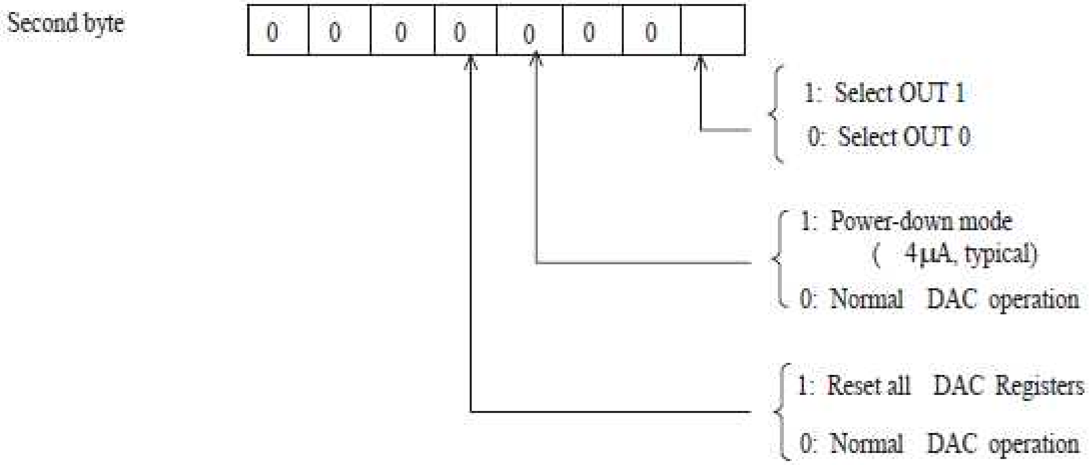
'01011000' '00000001' '100000000'

The MAX518 chip includes a power-on reset circuit that drives the two outputs to 0V initially. The two address inputs, AD1 and Ad0, provide an adjustable part of the chip's I²C address. With 5 bits fixed at 01011 and two adjustable bits, it is possible to connect four MAC518 chips to a PIC.

DAC Interfacing on I²C bus







Third byte

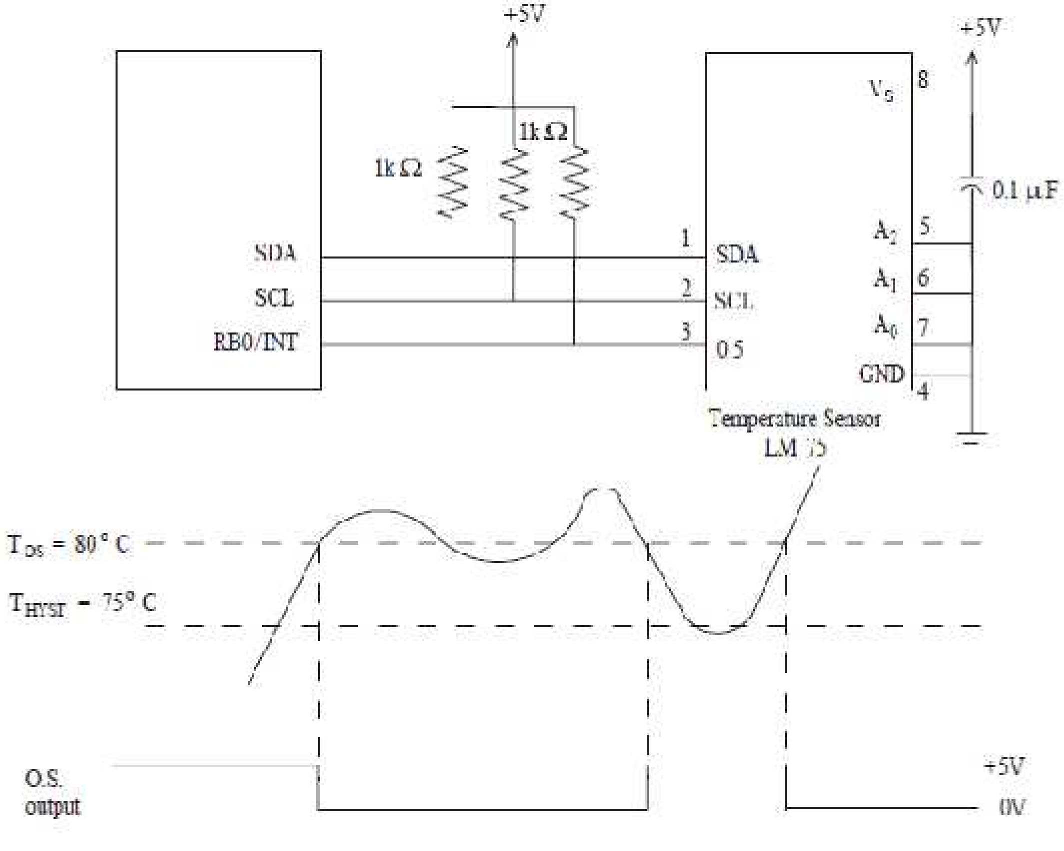
$$\text{Analog output voltage} = V_{DD}\frac{B}{256}$$

II. Interfacing a Temperature Sensor

National Semiconductor's LM 75 chip combines an analog temperature transducer, an analog-to-digital convertor (9-bit), and an I^{2}C bus interface, all in a tiny S)-8 surface mount package. The temperature range covered is -25°C to +100°C with +2°C accuracy. The two's complement form of the temperature is available from the 9-bit ADC. The resolution of the ADC is about 0.5°C.

Temperature	Digital Output	
	Binary	Decimal
125°C	01111 1010	250
25°C	00011 0010	50
0.5°C	000000001	1
0°C	00000 0000	0
-0.5°C	11111 1111	
-25°C	11100 1110	
-55°C	11001 0010	

LM 75 chip also includes a thermal watch dog that can be setup to interrupt PIC on its RBO/INT edge-triggered interrupt input when the temperature rises above a programmable, T$_{OS}$. It also includes programmable hysteresis so that the temperature must dip down below the setpoints T$_{OS}$ threshold to a lower T$_{HYST}$ threshold before rising againpast the T$_{OS}$ setpoint to generate another output edge.

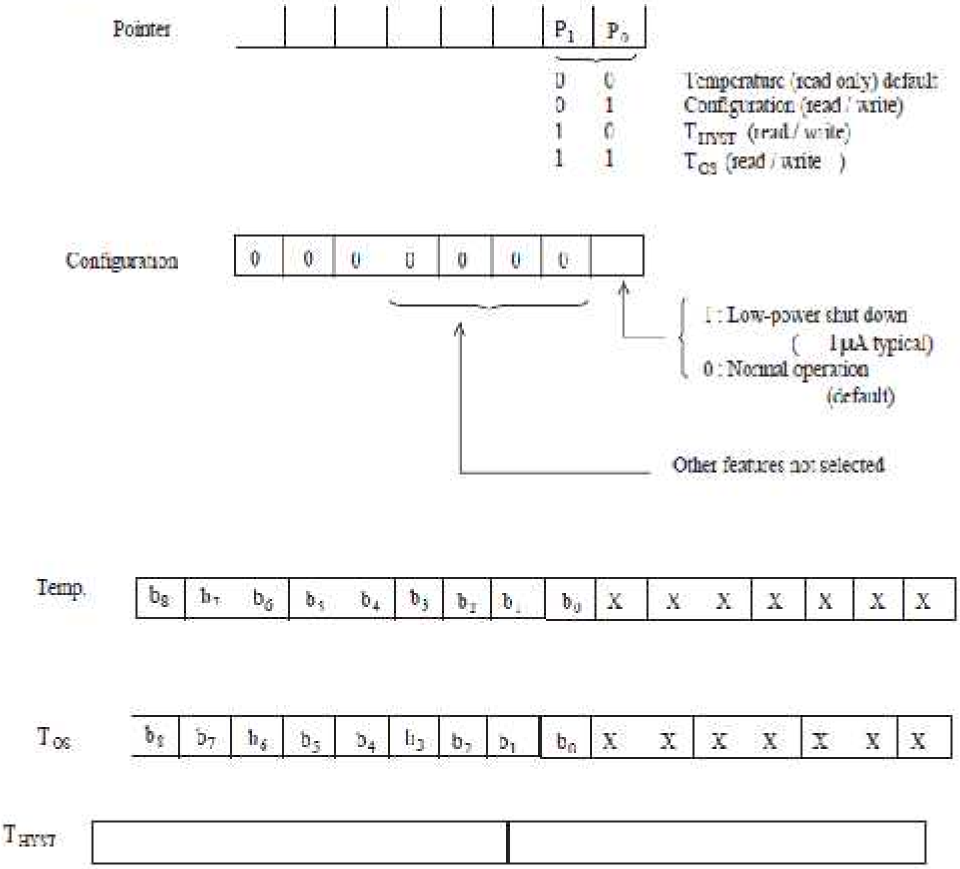


O.S. stands for over temperature shutdown

O.S. stands for over temperature shutdown.

Register Structure

When a "write" message string is sent, the first byte selects the chip for a write and the second byte loads the pointer register. The write message string can stop there or it can continue with a 2-byte write of T_{OS} (Over tem shutdown). Once the pointer has been set, any of their register can be read, reading two bytes for temperature, T_{OS}, or T_{HYST} or reading just 1 byte for the configuration register.


Pointer
P1
P0
0 0 Temperature (read only) default
0 1 Configuration (read / write)
1 0 THYST (read / write)
1 1 TOS (read / write)
Configuration
0 0 0 0 0 0 0
1 : Low-power shut down (1 μA typical)
0 : Normal operation (default)
Other features not selected
Temp.
b8 b7 b6 b5 b4 b3 b2 b1 b0 X X X X X X X
TOS
b8 b7 b6 b5 b4 b3 b2 b1 b0 X X X X X X X
THYST

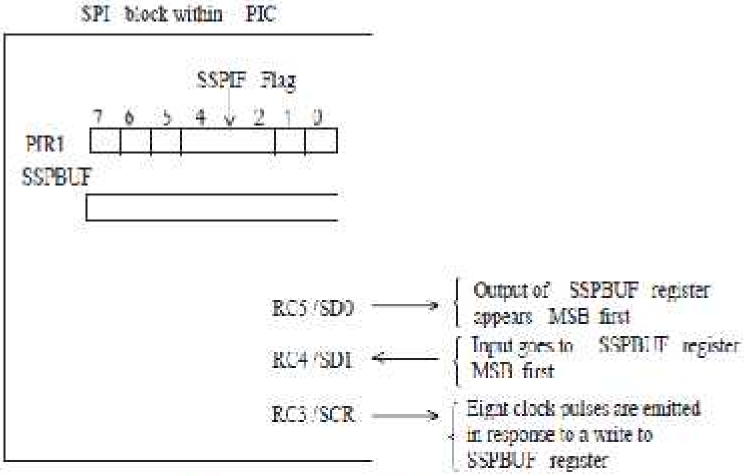
Synchronous Serial Port Module

Mid range PIC microcontroller includes a Synchronous Serial Port (SSP) module, which can be con figured into either of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)

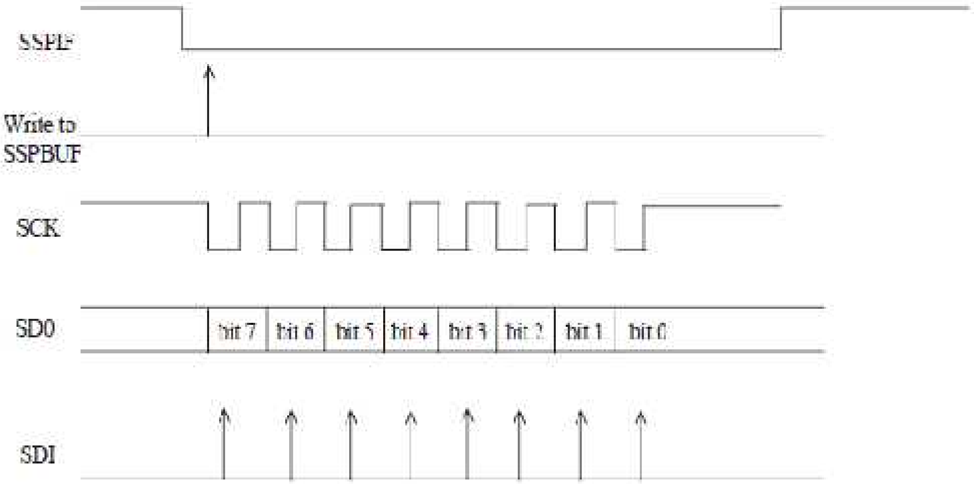
Either of these modes can be used to interconnect two or more PIC chips to each other using a minimal number of wires for interconnection. Alternatively, either can be used to connect a PIC chip to a peripheral chip. In this case of the I^2C mode, the peripheral chip must also include an I^2C interface. In contrast, the SPI mode provides the clock and serial data lines for direct connection to shift registers, adding an arbitrary number of I/O pins to a PIC chips.

Serial Peripheral Interface

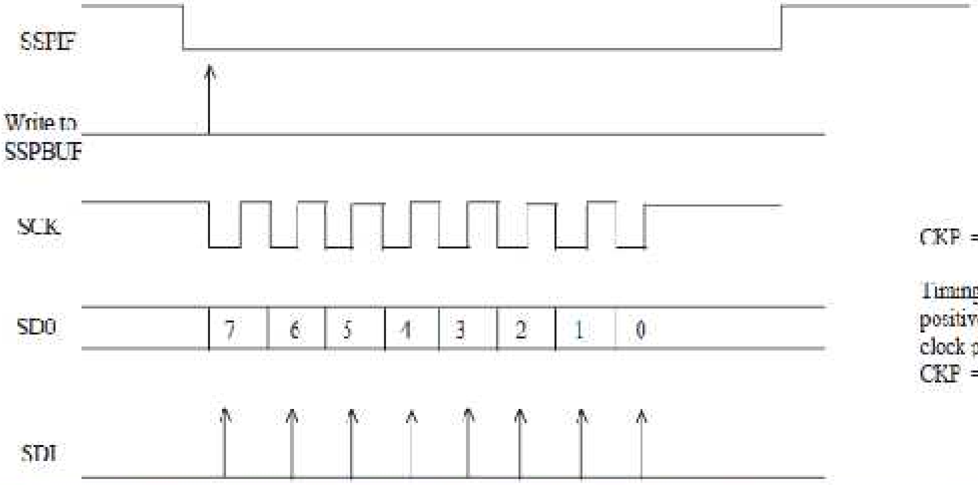

SPI block within PIC
SSPIF Flag
7 6 5 4 2 1 0
PIR1
SSPBUF
RC5/SD0
Output of SSPBUF register appears MSB first
RC4/SDI
Input goes to SSPBUF register MSB first
RC3/SCR
Eight clock pulses are emitted in response to a write to SSPBUF register

Portc three pins RC5, RC4 and RC3 are used for Synchronous Serial Interface. These pins revert to their normal general purpose I/O pins if neither of the two SSP modes is selected. The SPI port requires the RC3/SCK pin to be an output that generates the clock signal used by the external shift registers. This output line characterizes the SPI's master mode. In slave mode, RC3/SCK works as the input for the clock.

When a byte of data is written to SSPBUF register, it is shifted out the SDO pip in synchronism with the emitted pulses on the SCK pin. The MSB of SSPBUF is the first bit to appear on SDO pin. Simultaneously, the same write to SSPBUF also initiates the 8-bit data reception into SSPBUF of whatever appears on SDI pin at the time of rising edges of the clock on SCK pin.

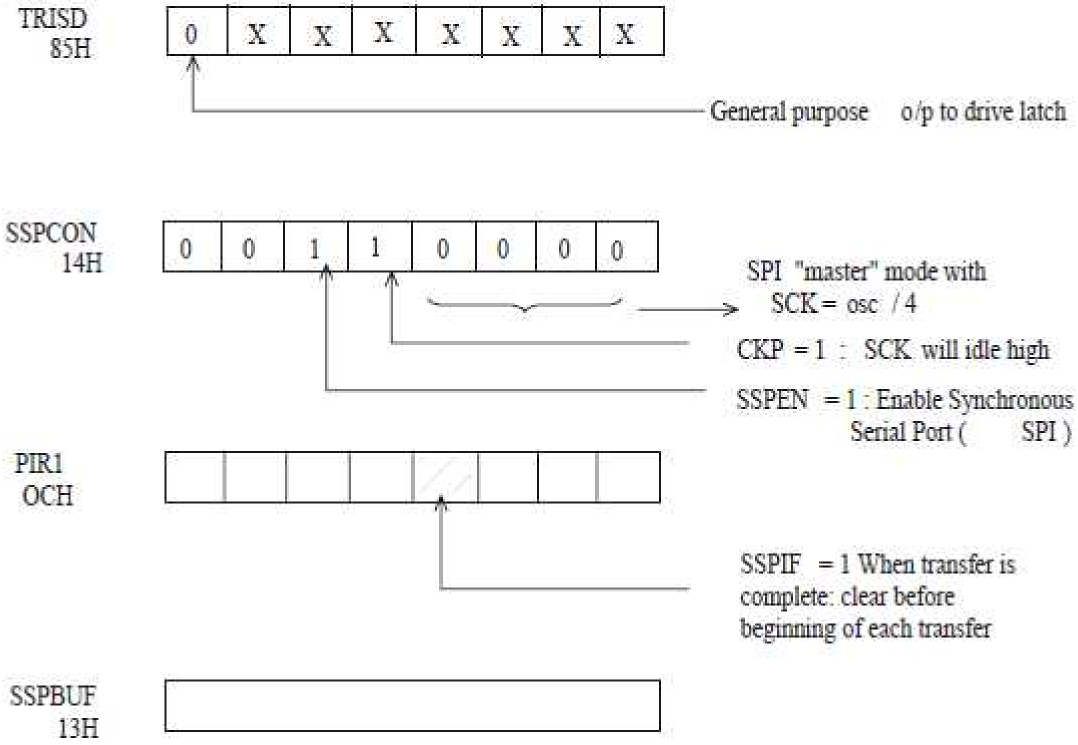


The SDI pin is read at these times
CKP = 1
Timing with negative going pulses.



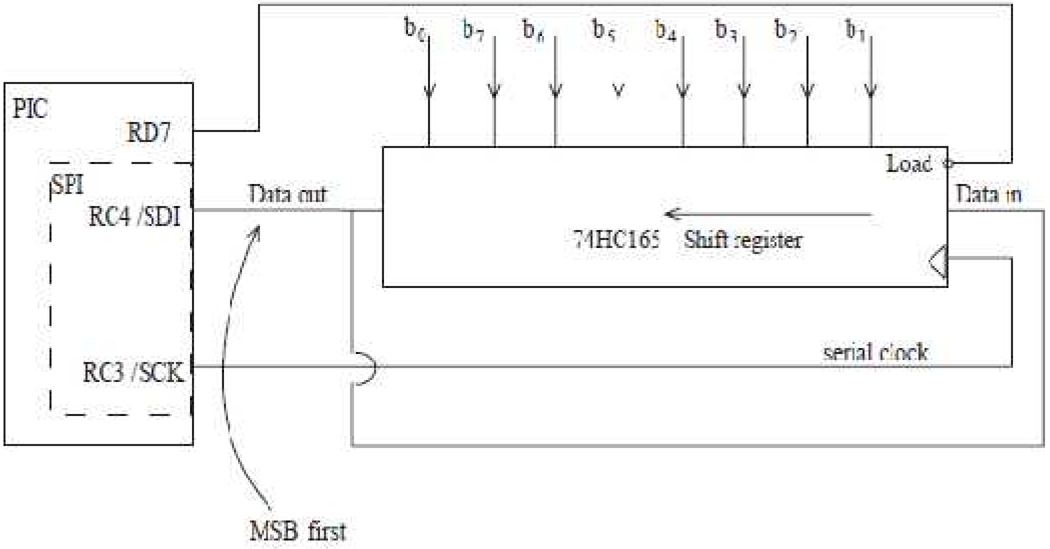
CKP = 0

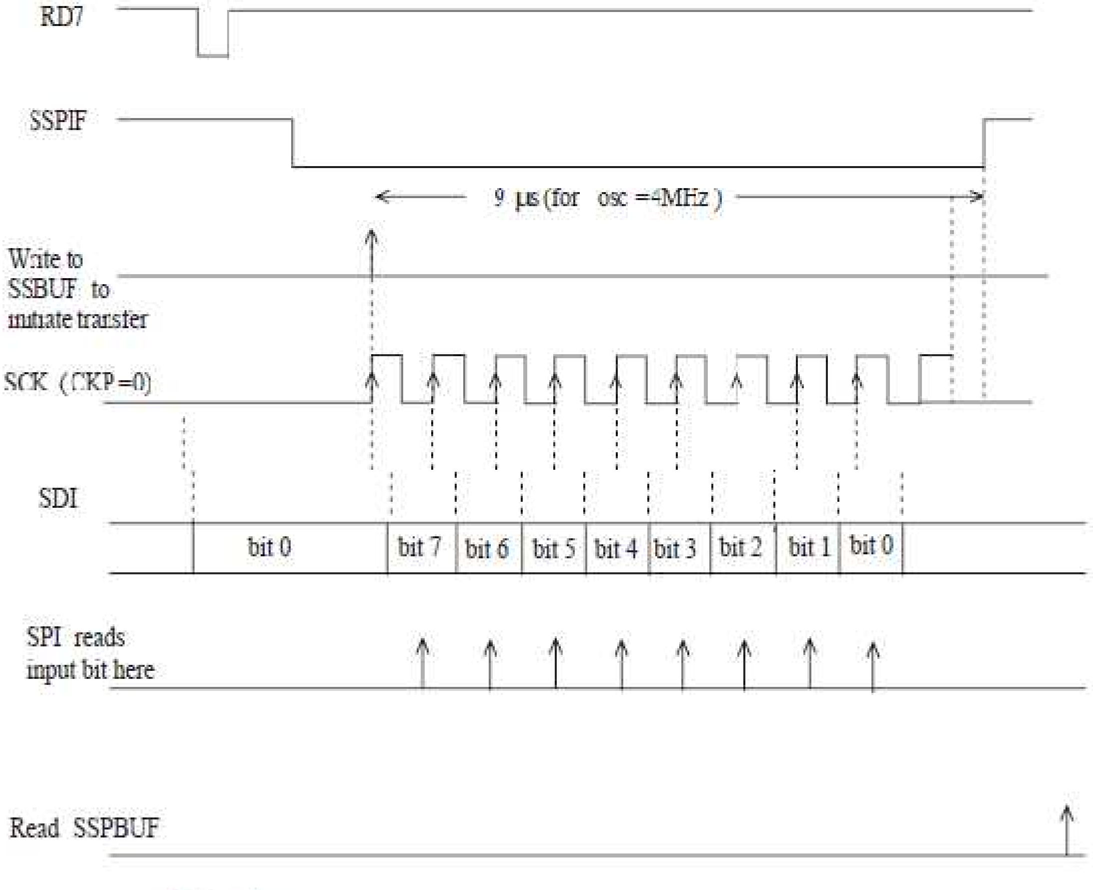
Timing with
positive going
clock pulse
CKP = 0


TRISD
85H
0
X
X
X
X
X
X
X
General purpose o/p to drive latch
SSPCON
14H
0
0
1
1
0
0
0
0
SPI "master" mode with
SCK = osc / 4
CKP = 1 : SCK will idle high
SSPEN = 1 : Enable Synchronous
Serial Port (SPI)
PIR1
OCH
SSPIF = 1 When transfer is
complete: clear before
beginning of each transfer
SSPBUF
13H

V

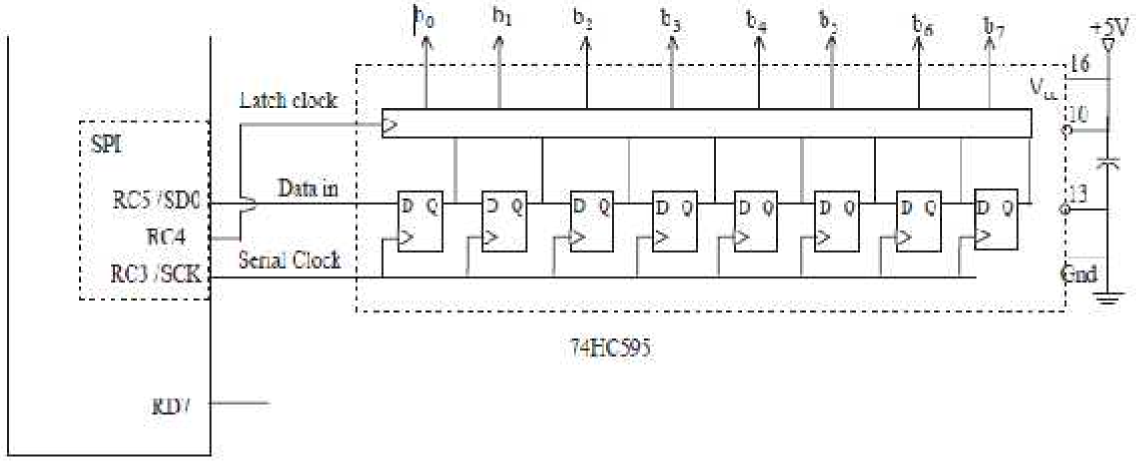
Input port expansion



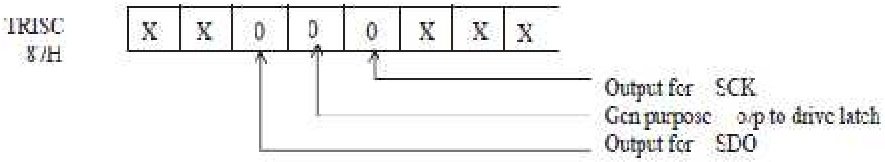


Timing diagram

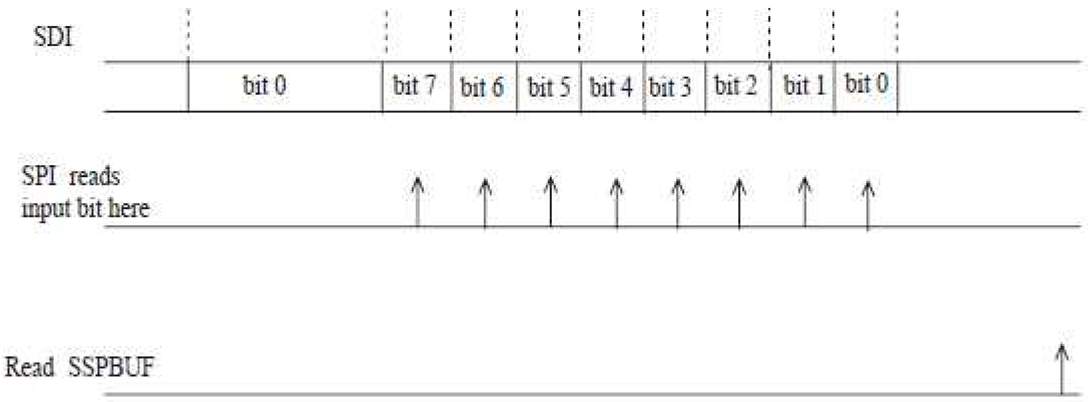
Output port expansion



Port configurations

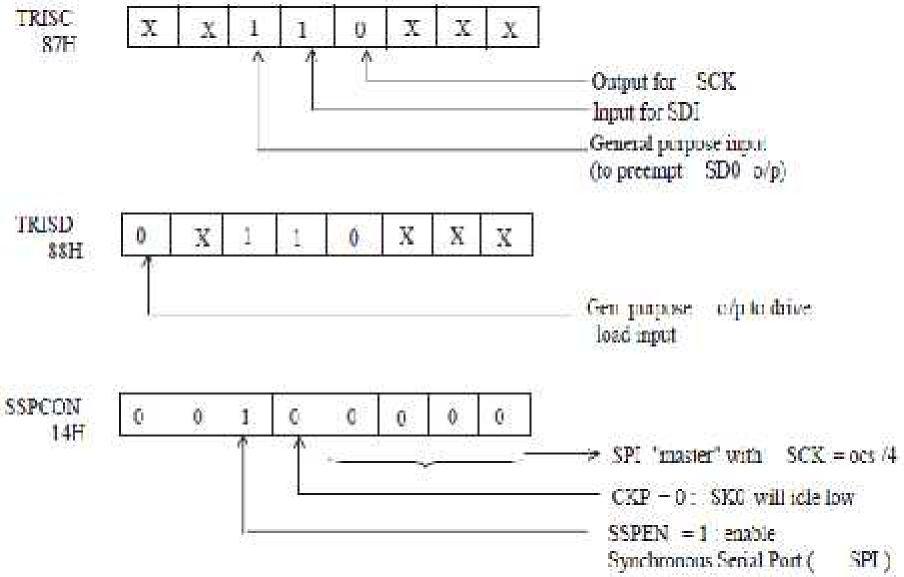


TRISC 87H	X	X	0	0	0	X	X	X



Timing diagram

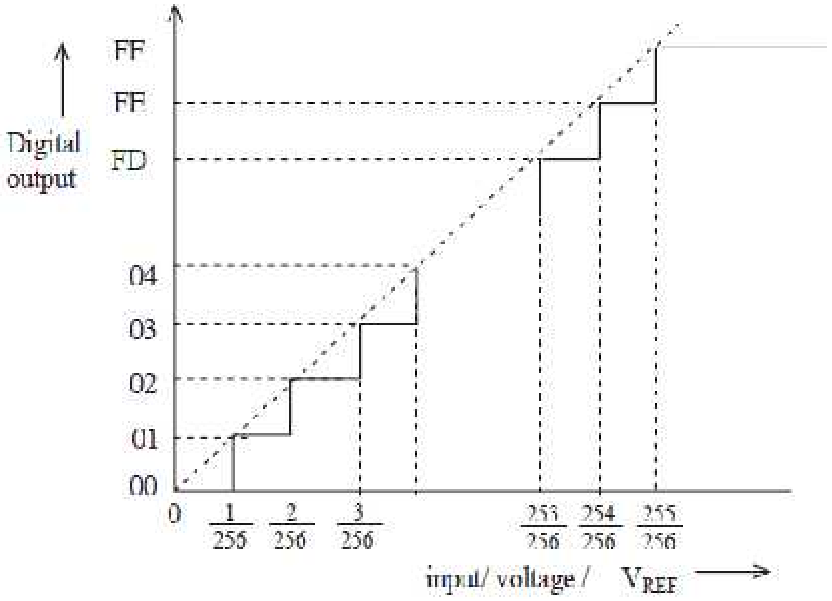
Port configurations


TRISC 87H
X X 1 1 0 X X X
Output for SCK
Input for SDI
General purpose input (to preempt SD0 o/p)
TRISD 88H
0 X 1 1 0 X X X
Gen purpose o/p to drive load input
SSPCON 14H
0 0 1 0 0 0 0 0
SPI "master" with SCK = ocs /4
CKP = 0 : SK0 will idle low
SSPEN = 1 : enable Synchronous Serial Port (SPI)

Analog-to-Digital Converter

Features (16C7X)

- Eight input channels
- An analog multiplexer
- A track and hold circuit for signal on the selected input channel



Port A and Port E pins are used for analog inputs/ Reference voltage for ADC.

Port A pins

RA0/AN0   Can be used as analog input 0

RA1/AN1 - Can be used as analog input-1

RA2/AN2 - Can be used as analog input-2

RA3/AN3/V_{REF} RA3 can be used as analog input 3 or analog reference voltage

RA4/TOCKI - RA4 can be used as clock input to Timer-0

RA5/$\overline{SS}$/AN4 - RA5 can be used as analog input 4 or the slave select for the sync serial port

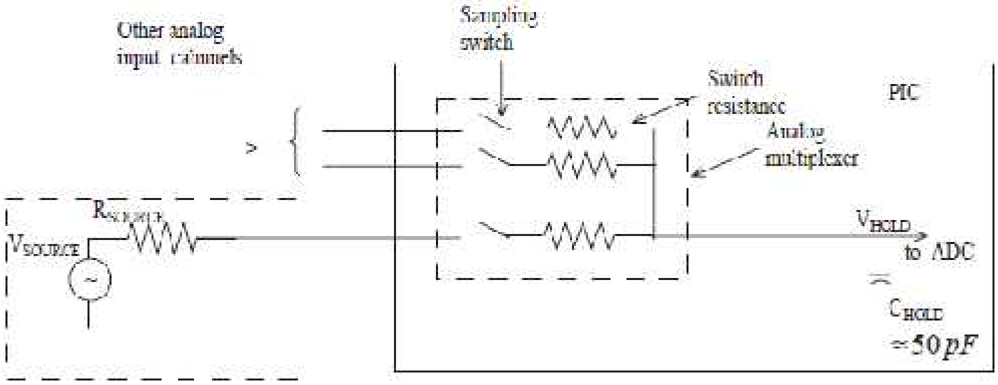
Port E pins

RE0/$\overline{RD}$/AN5   Can be used as analog input 5

RE1/$\overline{WR}$/AN5 - Can be used as analog input 6

RE2/$\overline{CS}$/AN7 - Can be used as analog input 7

PIC microcontroller has internal sample and hold circuit. The input signal should be stable across the capacitor before the conversion is initiated



After waiting out the sampling time, a conversion can be initiated. The ADC circuit will open the sampling switch and carry out the conversion of the input voltage as it was at the moment of opening of the switch. Upon completion of the conversion, the sampling switch is closed and V_{HOLD} again tracks V_{SOURCE}.

Using the A/D Converter

Registers ADCON1, TRISA, and TRISE must be initialized to select the reference voltage and the input channels. The first step selects the ADC clock source from among four choices (OSC/2, OSC/8, OSC/32, and RC). The constraint for selecting clock frequency is that the ADC clock period must be 1.6
mus or greater.
The A/D modules has three registers. These registers are

- A/D Result Register (ADRES)
- A/D Control Register 0 (ADCONO)
- A/D Control Register 1 (ADCON1)

The ADCONO register as shown here, controls the operation of A/D module.

7	6	5	4	3	2	1	0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	AD ON

<u>bit 7 - 6</u>

ADCS1 : ADCS 0
$00 = F_{osc}/2$
$01 = F_{osc}/8$
$10 = F_{osc}/32$
$11 = F_{RC}$ (clock derived from an internal RC oscillator)

<u>bit 5 - 3</u>

CHS2: CHS0
000 - channel 0 - AN0
001 - channel 1 - AN1
010 - channel 2 - AN2
011 - channel 3 - AN3
100 - channel 4 - AN4
101 - channel 5 - AN5
110 - channel 6 - AN6
111 - channel 7 - AN7

<u>bit 2</u> A/D Conversion Status bit

GO/$\overline{DONE}$
if ADON = 1
1 = A/D conversion is in progress (setting this bit starts the A/D conversion)
0 = A/D conversion is not in progress (this bit is automatically cleared by hardware when

bit 5 - 3

CHS2: CHS0
000 - channel 0 - AN0
001   channel 1   AN1
010 - channel 2 - AN2
011   channel 3   AN3
100 - channel 4 - AN4
101 - channel 5 - AN5
110 - channel 6 - AN6
111 - channel 7 - AN7

bit 2   A/D Conversion Status bit

GO/$\overline{DONE}$

if ADON = 1

1 = A/D conversion is in progress (setting this bit starts the A/D conversion)

0 = A/D conversion is not in progress (this bit is automatically cleared by hardware when

A/D conversion is complete.)

bit 1

Unimplemented

bit 0   ADON: A/D on bit

1 = A/D converter module is ON

0 = A/D converter module is OFF (not operating.)

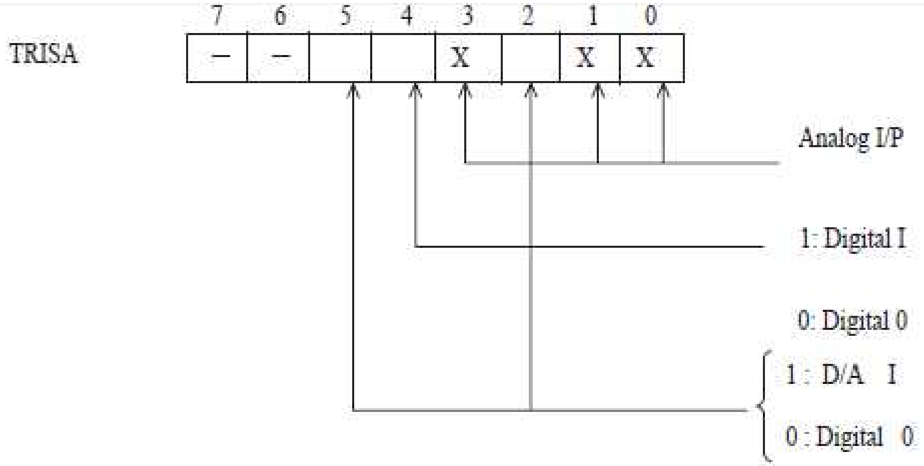
ADCON1 Register

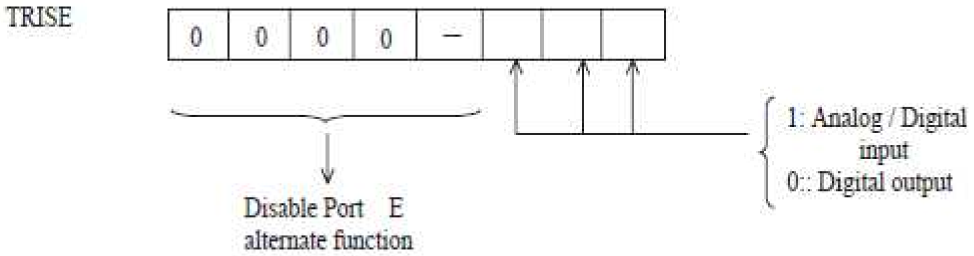
bit 7						bit 0
– –	–	–	PCFG2	PCFG1	PCFG0	

PCFG2 : PCFG0 : A/D Port Configuration Control bits

PCFG2 : PCFG0 : A/D Port Configuration Control bits

PCFG2 : PCFG0	RA0	RA1	RA2	RA5	RA3	RE0	RE1	RE2	V_{REF}
000	A	A	A	A	A	A	A	A	V_{DD}
001	A	A	A	A	V_{REF}	A	A	A	RA3
010	A	A	A	A	A	D	D	D	V_{DD}
011	A	A	A	A	V_{REF}	D	D	D	RA3
100	A	A	D	D	A	D	D	D	V_{DD}
101	A	A	D	D	V_{REF}	D	D	D	RA3
11X	D	D	D	D	D	D	D	D	-


TRISA
7
6
5
4
3
2
1
0
–
–
X
X
X
Analog I/P
1: Digital I
0: Digital 0
1 : D/A I
0 : Digital 0


TRISE
0
0
0
0
–
1: Analog / Digital input
0:: Digital output
Disable Port E alternate function

1. Configure A/D module
    - Configure analog pins/ voltage reference/ and digital I/O (ADCON1)
    - Select A/D channel (ADCON0)
    - Select A/D conversion clock (ADCONO)
    - Turn on A/D module (ADCONO)
2. Configure A/D interrupt (if required)
    - Clear AD—F bit in PIR 1 reg
    - Set AD—E bit in PIE 1 reg
    - Set G—E bit
3. Wait for required acquisitiion time
4. Start conversion
    - Set GO/$\overline{DONE}$
5. Wait for A/D conversion to complete by either
    - polling dor GO/$\overline{DONE}$ bit to be cleared
    - waiting for the A/D interrupt
6. Read A/D result register (ADRES) Clear AD—F if required.

Example Program

Example Program

A/D Conversion with Interrupt

```
bsf     STATUS, RPO     ;     Select Bank1
clrf    ADCON 1     ;     Configure A/D input
bsf     PIE1, ADIE     ;     Enable A/D interrupt
bcf     STATUS, RPO     ;     Select Bank 0
movlw 0811+ ; Select fosc/32, channel 0, A/D on movwf ADCONO
bcf     PIR1, ADIF
bsf     INTCON, PEIE
bsf     INTCON, GIE
; Ensure that the required sampling time for the
; selected input channel has elapsed.
; Then the conversion may be started
    bsf ADCONO, GO               ;  start A/D conversion
                                 ;  AD| F bit will be set
                                 ;  and GO/DONE bit is cleared
                                 ;  upon completion of A/D conversion
```

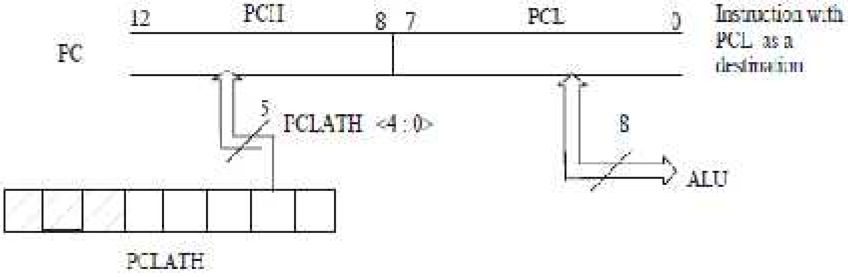
Code structure for large Programs

Memory paging is essential if the code exceeds 2k of program memory (2048). PIC 16C74A supports 4096 addresses and hence it is important to consider memory paging for this processor.

PCL and PCLATH

The program counter (PC) is 13-bit wide. The low byte comes from the PCL register, which is a readable and writable register. The upper bits (PC ¡12:8¿) are not readable, but are indirectly writable through the PCLATH register. On any reset , the upper bits of the PC will be cleared. PCL← 0 and PCLATH ← 0. Two situations for loading the PC following any reset are given here.

1. Any write to PCL register load the content of PCL to lower 8 bit of PC and content of PCLATH to higher 5 bits.
   mov wf PCL



2. PC is also loaded during a call or goto instruction

$$0 \leq k \leq 2047$$

Operation:

$$k \rightarrow PC \; < 10 : 0 >$$

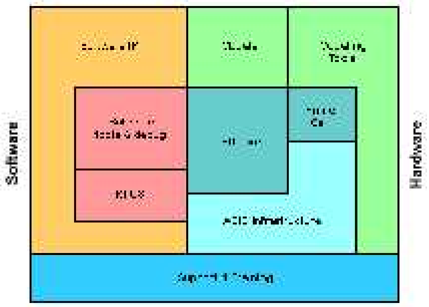
UNIT IV
INTRODUCTION TO ARM PROCESSOR

ARM designs microprocessor technology that lies at the heart of advanced digital products, from mobile phones and digital cameras to games consoles and automotive systems, and is leading intellectual property (IP) provider of high-performance, low-cost, power-efficient RISC processors, peripherals, and system-on-chip (SoC) designs through involvement with organizations such as the Virtual Socket Interface Alliance (VSIA) and Virtual Component Exchange (VCX).
ARM also offers design and software consulting services.
ARM's architecture is compatible with all four major platform operating systems: Symbian OS, Palm OS, Windows CE, and Linux. As for software, ARM also works closely with with its partners to provide optimized solutions for existing market segments.
These benefits are making the ARM company a complete solution provider.

The company offers a complete solution that is essential to the manufacturing process. Although ARM does not manufacture processors itself, ARM licenses its cores to semi-conductor manufacturers to be integrated into ASIC standards and then the company in using test chips manufactured by its partners to measure and validate the functionality of the core. ARM is able to accelerate OEM time-to-market by capitalizing on its architecture. By providing the IP and supporting services, customers can gain a jump on their design cycle and obtain a competitive edge in their targeted market segment. At that point, the architecture is portable to further product generations or applications as all code creation is directly compatible with any future architecture produced by ARM.

• ARM's Global Technology Partner Network is the largest in the industry, spanning from semiconductor manufacturers to distributors. ARM has worked diligently to ensure that the partnerships provide proven solutions in real-time operating systems (RTOS), EDA tools, development systems, applications software, and design consulting, all built around the ARM architecture.

This block diagram describes the ARM solution. The company recognizes that it cannot just present hardened macros and synthesizable CPUs to the industry, but it must also provide the ASIC infrastructure in the form of AMBA, the PrimeCell Peripherals, and models and modeling tools for the cores. There is also the need for ARM to pursue ports for RTOSs, develop debug hardware and software development tools, and, of course, embedded software for "off-the-shelf" integration. ARM combines all these futures together with support and training, to accelerate the design cycle and favour a successful product.



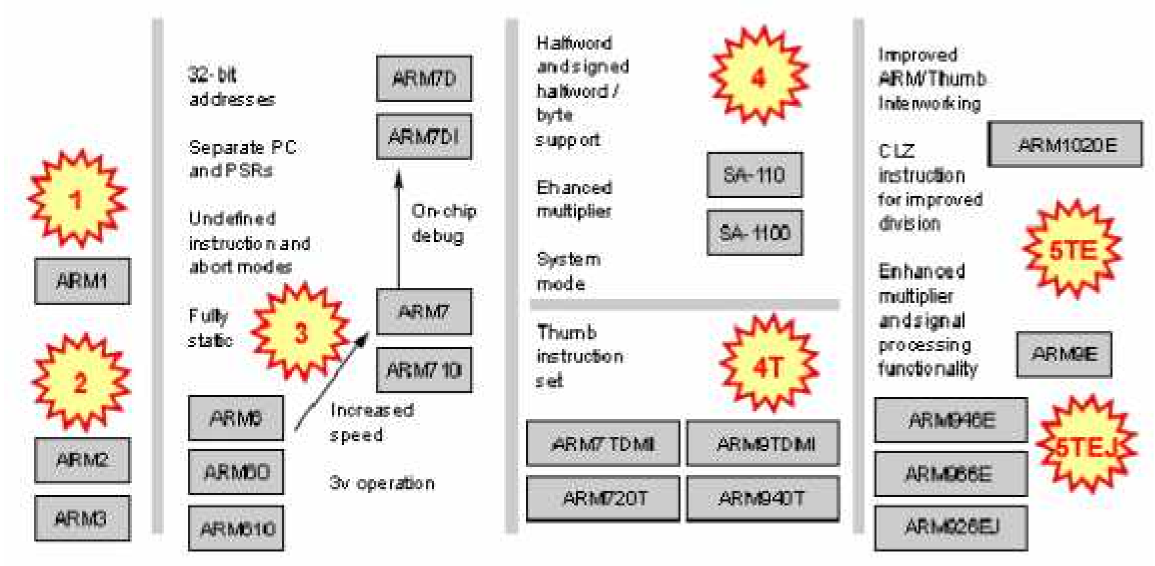
Introduction of the ARM's Core Families and their benefits

Overview of ARM's current families of main cores:

The ARM7 and ARM9 families have contributed to ARM's success. Each core family has several "children" that incorporate many different value-added features and combinations. Essentially, there are four main families available now for license: ARM7, ARM9, ARM9E-S, and ARM10. The ARM7 family features hardened and synthesizable macrocells with variants that incorporate cache with either a memory protection unit (MPU) or memory management unit (MMU). Other features include real-time debug (RTD) and real-time trace (RTT) technology. The ARM9 family consists of hardened macrocells with variants also including cache with an MPU or MMU, as well as the RTD and the RTT. Although the ARM9E-S family was released under a different architecture version, ARMv5TE, the fundamental design of the core is based on the ARM9TDMI family. The "E" identifies that the family is a DSP-enhanced architecture and the "S" identifies that the family is synthesizable.

The ARM10 family is the highest performance family to date and will also embody the "E" extensions that were developed for the ARM9E-S family. Finally, the StrongARM and XScale families are ARM compliant architectures available from Intel.

The Evolution of the ARM architecture:

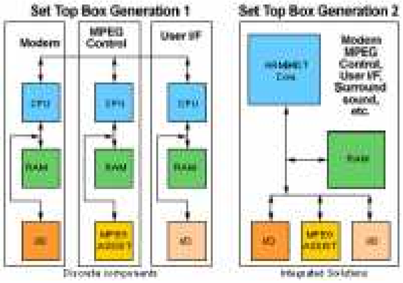


Architecture V1 was implemented only in the ARM1 CPU and was not utilized in a commercial

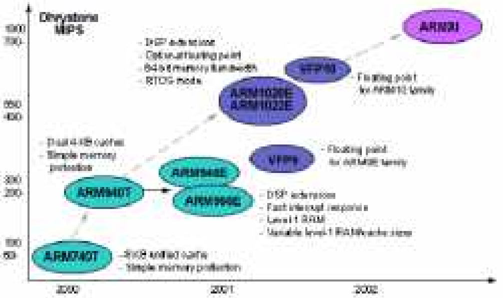
product. Architecture V2 was the basis for the first shipped processors. These two architectures were developed by Acorn Computers before ARM became a company in 1990. After that introduced ARM the Architecture V3, which included many changes over its predecessors.
These changes resulted in an extremely small and power-efficient processor suitable for embedded systems.
Architecture V4, co-developed by ARM and Digital Electronics Corporation, resulted in the Strong ARM series of processors. These processors are very performance-centric and do not include the onchip debug extensions. This architecture was further developed to include the Thumb 16-bit instruction set architecture enabling a 32-bit processor to utilize a 16-bit system. Today, ARM only licenses cores based on Architecture V4T or above. The latest architectures, version 5TE and 5TEJ, embody added instructions for DSP applications and the Jazelle-Java extensions, respectively. Currently, the ARM9E and 10E family of processors are the only implementations of these architectures.

From a development standpoint, ARM cores offer the advantage of a fully 32-bit processor designed specifically for embedded applications. An important feature is the embedded core debug facilities, which reduce the debugging stage of development. In some cases, this can be two-thirds of the overall development cycle.
Architecture compatibility allows code re-use and results in reduced design time. This in turn leads to reduced system cost, by eliminating investment in a second set of development tools to write code for a new processor architecture. The modular approach of the advanced micro-controller bus architecture, (AMBA), enables design reuse. This lowers the complexity of system on-chip (SoC) designs and reduces future design costs.
ARM and third parties offer the developer proven compiler technology and debug solutions. Multiple RTOSs and silicon sources mean that the developer will not need to change the preferred vendor in order to migrate to this architecture.

Redusing System Costs:



The ARM product roadmap:



Since the introduction of the ARM7 architecture, there has been huge leaps in core processing performance. As shown here, ARM families provide a wide range of performance, from 100 MIPS to 1000 MIPS.This increase in performance can be attributed to two main driving factors. The most obvious factor is the advances that have been made in new process technologies. The other is the engineering changes implemented in each subsequent generation of ARM processors and architectures.

Specific examples include a new pipeline in the ARM9 family, and the implementation of a Harvard bus architecture in the ARM 9 over the Von Neumann architecture in the ARM7. The result is that the ARM9 family doubles the performance of the ARM7 family. Recent developments include DSP and Jazelle-Java extensions to some of the new architectures.

These products enable feature rich applications to benefit from the high-performance and low power consumption intrinsic to ARM processor cores. Because of the fact that true embedded control applications typically require a processor with cache and memory protection to utilize real-time operating systems, ARM has developed a vertical expansion of CPUs to match these requirements. Each processor provides a unique, and in some cases configurable, amount of cache.

For example, the ARM9E-S family offers the ability to configure the size of instruction and data cache, as well as the ability to configure tightly coupled SRAM blocks. These features enable you to custom fit the CPU to specific application requirements. Many other features can be added via the co-processor interface, such as the Vector Floating Point
unit for the ARM10 and ARM9E families.

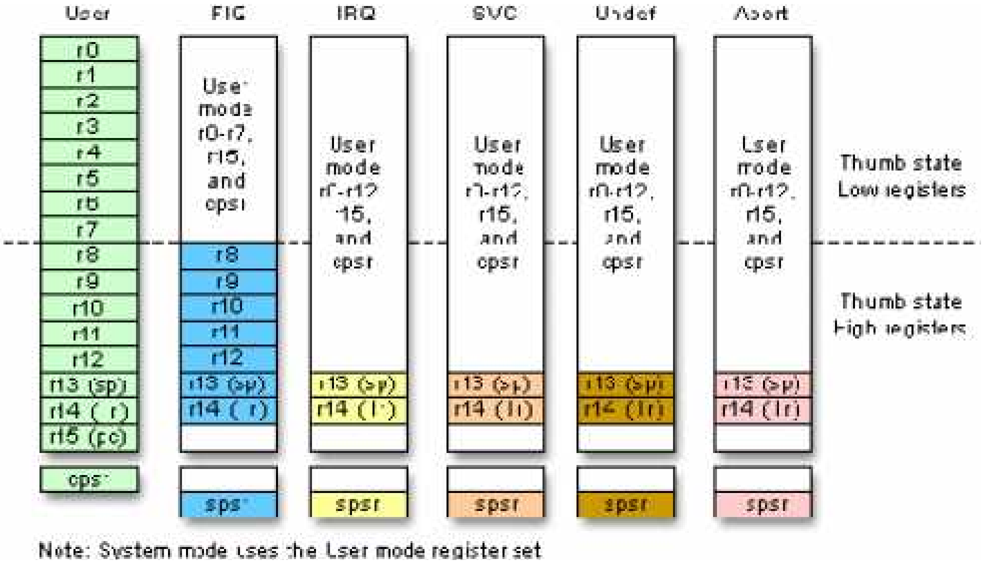
In other words, ARM has produced architectural families that are compatible, flexible, and encompass the full range of embedded requirements. Each product is designed to allow multi-sourcing at every level of development. ARM is now the de-facto standard in embedded IP.

Explanation of the ARM architecture

ARM cores use a 32-bit, Load-Store RISC architecture. That meanins that the core cannot directly manipulate the memory. All data manipulation must be done by loading registers with information located in memory, performing the data operation and then storing the value back to memory. There are 37 total registers in the processor. However, that number is split among seven different processor modes.

The seven processor modes are used to run user tasks, an operating system, and to efficiently handle exceptions such as interrupts. Some of the registers with in each mode are reserved for specific use by the core, while most are available for general use. The reserved registers that are used by the core for specific functions are r13 is commonly used as the stack pointer (SP), r14 as a link register (LR), r15 as a program counter (PC), the Current Program Status Register (CPSR), and the Saved Program Status Register (SPSR).

The SPSR and the CPSR contain the status and control bits specific to the properties the processor core is operating under. These properties define the operating mode, ALU status flags, interrupt disable/enable flags and whether the core is operating in 32-bit ARM or 16-bit Thumb state.



Note: System mode uses the User mode register set

There are 37 total registers divided among seven different processor modes. Fifgure 09 shows the bank of registers visible in each mode. User mode, the only non-privileged mode, has the least number of total registers visible. It has no SPSR and limited access to the CPSR. FIQ and IRQ are the two interrupt modes of the CPU.

Supervisor mode is the default mode of the processor on start up or reset. Undefined mode traps unknown or illegal instructions when they are passed through the pipeline. Abort mode traps illegal memory accesses as a result of fetching instructions or accessing data.

Finally, system mode, which uses the user mode bank of registers, was introduced to provide an additional privileged mode when dealing with nested interrupts.

Each additional mode offers unique registers that are available for use by exception handling routines. These additional registers are the minimum number of registers required to preserve the state of the processor, save the location in code, and switch between modes.

FIQ mode, however, has an additional five banked registers to provide more flexibility and higher performance when handling critical interrupts.

When the ARM core is in Thumb state, the registers banks are split into low and high register domains. The majority of instructions in Thumb state have a 3-bit register specifier. As a result, these instructions can only access the low registers in Thumb, R0 through R7. The high registers, R8 through R15, have more restricted use. Only a few instructions have access to these registers.

3.2 TDMI

T-D-M-I stands for:

- **T**humb, which is a 16-bit instruction set extension to the 32-bit ARM architecture, referred as states of the processor. "**D**" and "**I**" together comprise the on-chip debug facilities offered on all ARM cores.

These stand for the **D**ebug signals and Embedded**I**CE logic, respectively.

- The M signifies the support for 64-bit results and an enhanced multiplier, resulting in higher performance. This multiplier is now standard on all ARMv4 architectures and above.

Thumb 16-bit Instructions

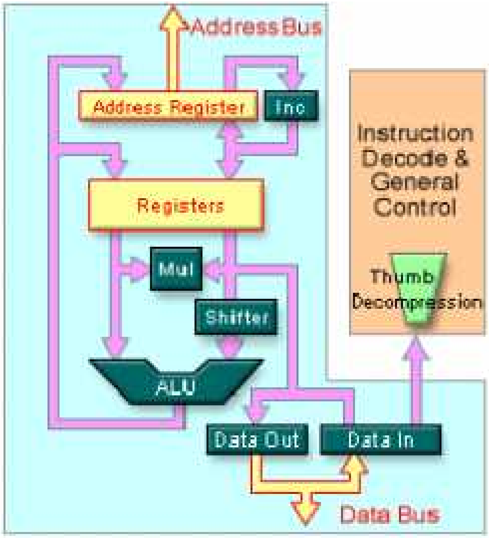
With growing code and data size, memory contributes to the system cost. The need to reduce memory cost leads to smaller code size and the use of narrower memory. Therefore ARM developed a modified instruction set to give market-leading code density for compiled standard

C language. There is also the problem of performance loss due to using a narrow memory path, such as a 16-bit memory path with a 32-bit processor.

The processor must take two memory access cycles to fetch an instruction or read and write data. To address this issue, ARM introduced another set of reduced 16-bit instructions labeled Thumb, based on the standard ARM 32-bit instruction set. For Thumb to be used, the processor must go through a change of state from ARM to Thumb in order to begin executing 16-bit code. This is because the default state of the core is ARM. Therefore, every application must have code at boot up that is written in ARM. If the application code is to be compiled entirely for Thumb, then the segment of ARM boot code must change the state of the processor. Once this is done, 16-bit instructions are fetched seamlessly into the pipeline without any result.

It is important to note that the architecture remains the same. The instruction set is actually a reduced set of the ARM instruction set and only the instructions are 16-bit; everything else in the core still operates as 32-bit.

An application code compiled in Thumb is 30% smaller on average than the same code compiled in ARM and normally 30% faster when using narrow 16-bit memory systems.



An example: ARM7TDMI Block Diagram Figure 10 shows the register bank in the center of the diagram, plus the required address bus and data bus. The multiplier, in-line barrel shifter, and ALU are also shown.

In addition, the diagram illustrates the in-line decompression process of Thumb instructions while in the decode stage of the pipeline. This process creates a 32-bit ARM equivalent instruction from the 16-bit Thumb instruction, decodes the instruction, and passes it on to the execute stage.

Debug Extensions

The Debug extensions to the core add scan chains to monitor what is occurring on the data path of the CPU. Signals were also added to the core so that processor control can be handed to the debugger when a breakpoint or watchpoint has been reached. This stops the processor enabling the user to view such characteristics as register contents, memory regions, and processor status.

Embedded ICE Logic

In order to provide a powerful debugging environment for ARM-based applications the Embedded ICE logic was developed and integrated into the ARM core architecture. It is a set of registers providing the ability to set hardware breakpoints or watchpoints on code or data. The

Embedded ICE logic monitors the ARM core signals every cycle to check if a breakpoint or Watch point has been hit. Lastly, an additional scan chain is used to establish contact between the user and the Embedded ICE logic.

Communication with the EmbeddedICE logic from the external world is provided via the test access port, or TAP, controller and a standard IEEE 1149.1 JTAG connection.

The advantage of on-chip debug solutions is the ability to rapidly debug software, especially when the software resides in ROM. This is critical in shortening the development cycle. The use of Multi- ICE and EmbeddedICE provides full debug capabilities for a processor integrated deep inside an ASIC, even in a production version of a consumer product.

Architecture details, features & comparison of the ARM7, ARM9, and ARM10 core families

4.1 ARM7TDMI Processor Core

- Architecture version 4T:

-- 3-stage pipeline

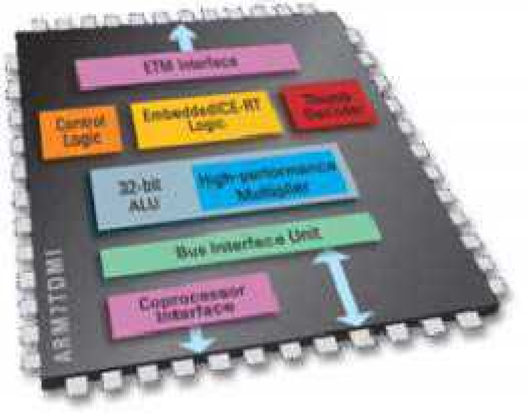
-- Unified bus architecture

-- 32-bit ARM ISA plus 16-bit Thumb extension

- Forward compatible code
- EmbeddedICE on-chip debug
- Hard Macrocell IP

-- Smallest Die Size: 0.53 mm$_2$ on 0.18 µm process

- Up to 110 MHz* on TSMC standard 0.18 µm
- Industry leading 0.25 mW/MHz



The ARM7TDMI has a core based on the fourth version of the ARM architecture. This implementation uses a three stage pipeline - a standard fetch-decode-execute organization.

It features a unified cache, as well as the Thumb extension permitting 32-bit and 16-bit operation. It is completely forward compatible, meaning that any code written for this core will be compatible with any new core releases, such as ARM9 or ARM10. This core also includes the on-chip debug extension discussed in the previous training module.

The core is successful mainly because of the extremely small but high performance processor - slightly more than 70,000 transistors in all an with extremely low power consumption.

ARM7TDMI-S

- Synthesizable RTL compliant with the ARM7TDMI

Custom Macrocell:

-- Fully compatible with the ARMv4T architecture.

-- Right denied to modify ARM7TDMI instruction set.
-- Coprocessor interface allows custom functions to
be added outside core.
-- EmbeddedICE support with "Multi-ICE" protocol
converter or third party device.
• Supports AMBA interface:
-- Standard interface, ideal for integration
of the core into an ASIC design.
• Supports full-scan and automatic test pattern generator.

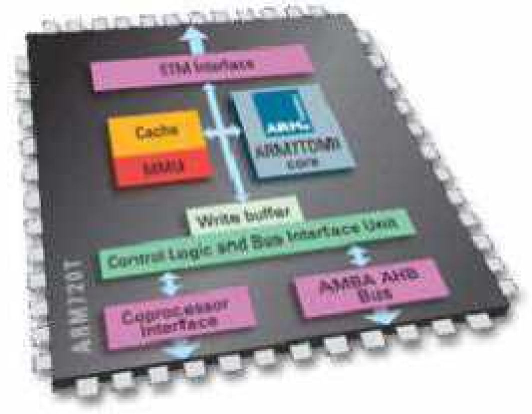


Figure 12 presents a model of the ARM7 processor that is a synthesizable version of the ARM7TDMI.
This version is fully compatible with the ARMv4T architecture and is functionally identical to the hardened ARM7TDMI macrocell. Although it is a synthesizable solution, the licensee does not have the right to change any feature of this core.

ARM720T

• Cached Macrocell for Platform OS Applications
• ARM7TDMI core:
-- ARM v4T ISA
-- THUMB 16-bit instruction set
-- Rev 3 onwards supports ETM7 for non-stop debug
• 8 KB cache:
-- High processor performance with low-speed
memory interface
• Memory Management Unit:
-- Full support for WindowsCE and Symbian OS
• ASB bus interface

ARM720Tcore offers 8 KB of unified instruction and data cache. Also included is a memory management unit (MMU) that offers virtual-to-physical address translation, 64-entry translation look aside buffer (TLB), two-level page tables stored in memory, and hardware page-table walking. There is also a highly flexible mapping scheme that supports 1 MB sections with permissions, 64 KB large pages with four sets of permissions, and 4 KB small pages with four sets of permissions. This processor enables up to 16 domains, each with individual access rights. It also features cyclic replacement and lockdown features to lock instructions or data into cache for critical real-time code.

The ARM720T was designed to be flexible and application-specific, especially for devices running complex operating systems such as Linux, Windows CE, Symbian OS, or PalmOS. It includes a system control coprocessor for cache and system initialization and the AMBA Advanced System Bus, or ASB, interface.

ARM7EJ

- New Jazelle-enhanced 32-bit processor core
- Thumb, Jazelle and DSP extensions
- Five stage pipeline and high performance multiplier
- Unified instruction and data bus
- v5TEJ architecture
- Real-time trace with the ETM9 macrocell
- Contact ARM for availability and characteristics data

The ARM7EJ solution is a compact CPU specifically designed for applications demanding low power consumption. It has a memory interface identical to that of the ARM7TDMI-S? core. It features the V5TEJ architecture instructions, including DSP extensions. This core implementation also features a five-stage pipeline similar to that of an ARM9 class processor, and supports easy integration of the Embedded Trace Macrocell-9 for real-time-trace capability.

ARM SC100 Secure Code

- Optimized processor family for smart card solutions
- Security enhanced ARM7TDMI design

-- ARMv4T compliant
-- Low power, high performance and small die size
-- Memory Protection Unit (MPU)
-- Anti-tampering/counterfeiting measures
-- JavaCard support
-- Standard coprocessor interface for incorporation of cryptographic solutions.

- SC100 - Small synthesizable IP:

-- 35K gates - 1 mm$_2$ area
-- 66 MHz* on 0.25 mm @2.5 V
-- Power: 0.7 mW/MHz

The ARM SecurCore family provides unique 32-bit RISC-based solutions for smart card development needs, offering system designers privileged access to ARM processor cores to create fast and secure e-commerce solutions.

The flexible Memory Protection Unit was specifically designed to ensure security of operating system and application data. This enables future generations of smart card solutions having multiple applications running on a single card. Special features in the core have been designed to help obscure

UNIT V
ARM ORGANIZATION

Processor activity and hide application program signatures, making SecurCore activity difficult to detect and observe.

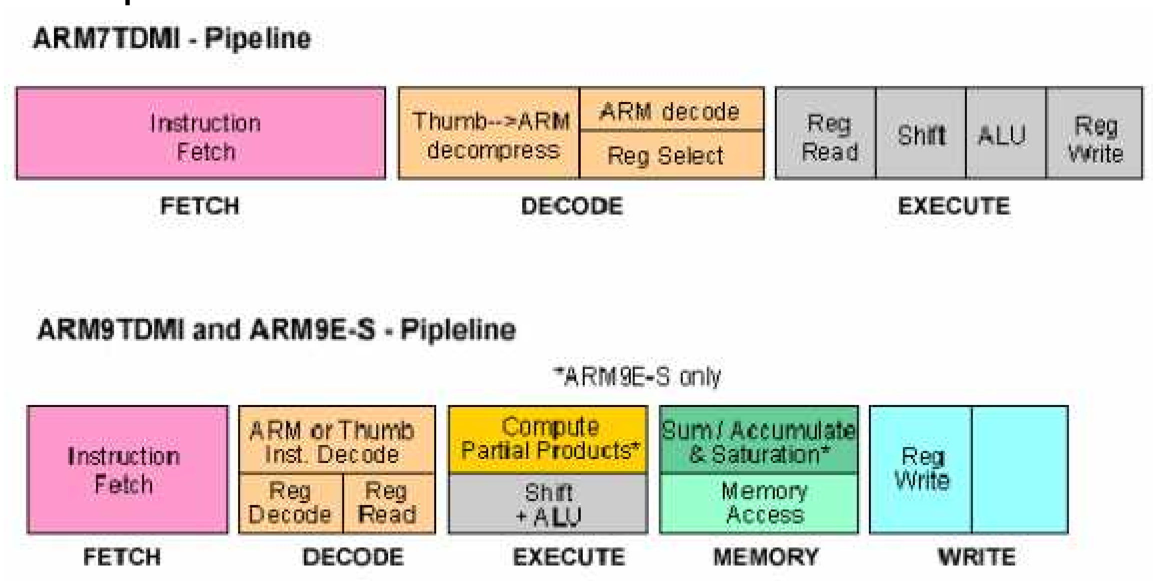
The SC100 runs all existing ARM JavaCard software implementations. Future SecurCore processors will include ARM's Jazelle technology for direct execution of Java byte codes to enable high performance low-power JavaCard applications. The advantages over a purely software-emulated JavaCard virtual machine are clear: significant reduction in execution time, improved responsiveness, and significantly power consumption.

The SecurCore family of processors also includes a standard coprocessor interface for simple incorporation of cryptographic coprocessors. A coprocessor can be designed for a very specific purpose and can contain as many registers and data paths as needed to implement the specific functions.

To provide one solution, ARM has integrated into the SC100 core a cryptographic accelerator, the Montgomery Multiplier Engine (MME). This engine is optimized for RSA calculations, providing five times the performance of software solutions without any restrictions on key length.

The SecurCore family offers all the benefits of ARM's industry leading high-performance, low-power architecture, with significant design differences that make the ARM approach ideal for secure applications.

4.3 Comparison of the ARM7TDMI with the ARM9TDMI families



To increase performance, the pipeline of the ARM9TDMI core was re-engineered from the three stage system used by the ARM7TDMI family to five stages.

Operations previously performed in the execute stage of ARM7 are spread across four stages in the ARM9 pipeline: decode, execute, memory, and write. The reorganization and removal of these critical paths resulted in a much higher clock frequency.

Another performance improvement is the reduced cycles per instruction rating of the processor. This is due to improved load and store instruction cycle counts. Single load and store instructions are now single-cycle operations. This is an enhancement over the ARM7 operation, which used the execute stage three times: first, to calculate the address; second, to access the memory and cache; and third, to write the data to the register bank. On ARM9, each step has a separate pipeline stage requiring only one cycle, avoiding pipeline stalls.

The ARM7TDMI family is popular with applications where small die size, high performance, and low power consumption help reduce system costs, especially when the system does not require cache. Applications include cellular phones, MP3 players, and mass storage.

The ARM9TDMI family are used for high performance applications that previously could not be implemented at the same cost. This family of cores was developed with twice the performance of the ARM7TDMI and without changes to the architecture. It is ideally suited for the next generation of cell phones, personal digital assistants, multi-function peripherals and fast printers, and set-top box applications.

ARM9TDMI Processor Core

- ARM 32-bit and Thumb 16-bit instructions (v4T ISA).
- Very high code compatibility with ARM7TDMI:

-- Only change is simplified data-abort handler

- Portable to 0.25, 0.18 µm CMOS and below.
- Harvard 5-stage pipeline implementation:

-- Higher performance from reduced cycle per instruction (1.5)

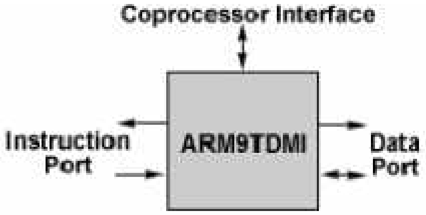
- Coprocessor interface for on-chip coprocessors:

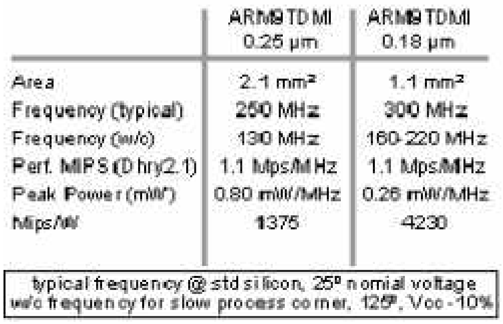
-- Allows floating point, DSP, graphics accelerators.

- EmbeddedICE debug capability with extensions:

-- Hardware single step

-- Breakpoint on exception.





	ARM9TDMI 0.25 µm	ARM9TDMI 0.18 µm
Area	2.1 mm²	1.1 mm²
Frequency (typical)	250 MHz	300 MHz
Frequency (w/c)	130 MHz	160-220 MHz
Perf. MIPS (Dhry2.1)	1.1 Mips/MHz	1.1 Mips/MHz
Peak Power (mW)	0.80 mW/MHz	0.26 mW/MHz
Mips/W	1375	4230

typical frequency @ std silicon, 25º nomial voltage
w/c frequency for slow process corner, 125º, Vcc -10%

The Harvard bus architecture creates separate instruction and data memory interfaces, enabling simultaneous access to instructions and data.

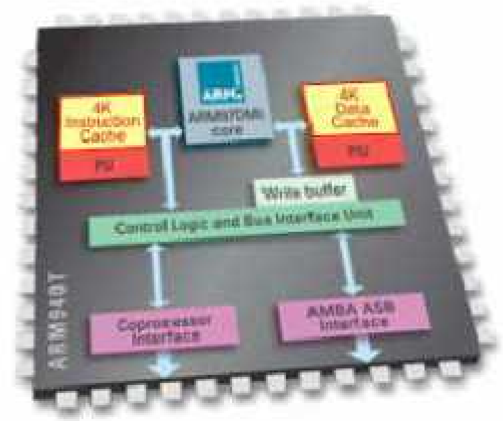
The ARM9TDMI represents a new family of CPU technology. The enhancements made to this core family doubles the performance of the ARM7TDMI family.

ARM940T Macrocell

- Processor for real-time embedded applications:

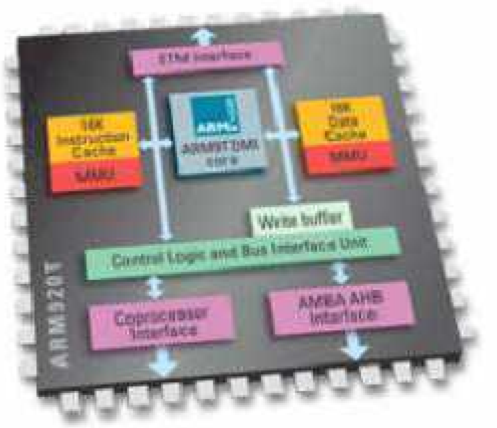
-- ARM9TDMI Core (v4T ISA)

-- 4 KB instruction and data cache with lock-down
-- Protection unit for RTOS
-- Code compatible from ARM7 Thumb CPUs
-- Hard Macro IP:
-- 4.2 mm_2 on 0.18 µm
-- Up to 200 MHz (worst case) on TSMC standard
0.18 µm
-- Power: 0.75 mW/MHz



The ARM940T represents the first sample of a cache-enabled ARM9TDMI core.
This core contains 4 KB each of instruction and data cache, with an MPU for use by real-time operating systems. This system makes the 940T an ideal CPU for embedded control applications, such as wireless networking devices, printers, or automotive control devices. The protection units allow definition of eight regions of memory, each with independent cache, write buffer enable, and access permissions. The protection unit is configured using on-chip registers, which provides a simple programmer's model. This eliminates the need for page-mapping tables stored in memory.

ARM's 940T Core Structure:
The core processor is about one-third of the die size. When other components are incorporated - the system control coprocessor, bus control, memory protection unit, and the cache itself - the integer unit becomes insignificant in total die area. This core has 4 KB caches, the smallest amount of cache used in the entire product family. One can visualize how small this integer unit and cache logic becomes when integrated with synthesized peripherals and the other features that complete the system-on-chip (SoC) design.



ARM920T Macrocell

- Cached processor for platform OS applications:

-- 16 KB instruction and data cache

-- ARMv4 MMU for Palm OS, Symbian OS,
Linux, and Windows CE

-- Code compatible from ARM7 Thumb CPUs

-- Hard Macro IP:

-- 11.8 mm_2 on 0.18 µm

-- Up to 200 MHz (worst case) on TSMC
standard 0.18 µm

-- Power: 0.8 mW/MHz

The ARM920T was created to address the needs of more complex systems using a platform operating system, such as Windows CE or Symbian OS. This core replaces the MPU of the 940T with a full memory management unit, and increases the instruction and data cache sizes to 16 KB for each. The performance, MMU, and cache of this core make it ideal for Personal Digital Assistants, smartphones, and set-top box applications.

ARM922T

Cached processor for Platform OS applications:

- 8 KB instruction and data cache
- ARMv4 MMU for: Palm OS, Symbian OS, Linux, and Windows CE
- Code compatible from ARM7 Thumb CPUs
- Hard Macro IP:

-- 8.1 mm_2 on 0.18 µm

- Up to 200 MHz (worst case) on TSMC standard 0.18 µm
- Power: 0.8 mW/MHz

The ARM922T core was created with half the amount of instruction and data caches of the 920T, resulting in smaller silicon overhead. Other than this simple difference, the two cores are fundamentally identical.

ARM920T and ARM922T MMU

- Two TLBs:

-- 64-entry instruction TLB

-- 64-entry data TLB

- Two-level page tables (stored in memory)

- Hardware page-table walking
- Cyclic replacement
- Lockdown features:

-- Lock instructions or data into cache for critical real-time code

Both the 920T and 922T core utilize an MMU with the same features. There are two, 64-entry translation look-aside buffers for instruction and data, two-level page tables, hardware page-table walking, and support for random or round robin replacement. Lockdown features are also included to secure critical real-time code. This cache architecture results in two solutions that are simpler to program and minimize power, area, and required memory.

ARM9E Family

The ARM9E family is currently comprised of four different units. The base ARM9E integer processor offers a high performance and low gate count synthesized solution in its most basic form.

The other units offer the true capabilities of the core when coupled with SRAM, cache, vector floating point acceleration, and the Jazelle Java extensions.

As a suite of synthesizable solutions, the final gate count and power consumption statistics of these cores depends on the implementation and the process technology used.

ARM9E Core Architecture

- 32-bit load/store RISC architecture
- Efficient 5-stage pipeline
- ARM andThumb instruction sets
- 37 x 32-bit registers
- 32-bit ALU and barrel shifter
- Enhanced 32-bit MAC block
- ETM9 interface
- AMBA AHB interface
- Coprocessor interface
- Synthesizable or soft IP

As mentioned hard macrocells always have been the ultimate answer for optimized performance and die size in any given processor design. But newer synthesized design flows are pushing the envelope for SoC applications.

The ARM9E family was built upon the standard set by the ARM9TDMI family, but it also provides freedom for defining the cache and tightly coupled SRAM configurations used by the core.

It was also the first family of CPUs designed to the AHB bus of the AMBA 2.0 specification.

Another key technological enhancement to this family of CPUs includes DSP extensions for true realtime systems. This improvement to the architecture introduces additional multiply and saturated math instructions for use by complex DSP algorithms. This family is also fully code compatible with ARMv4T architecture cores.

Lastly, to enhance the debug capabilities already common in ARM CPUs, the Embedded Trace Macrocell interface was added. This interface enables real-time debugging of complex real-time systems.

ARM966E-S

- Solution for hard real-time applications:
- ARM9E core (v5TE ISA).
- I and D TCM memory interfaces with 'wait' signal
- Selectable size Instruction and Data TCM
(0 KB - 64 MB)
- AMBA AHB bus interface
- Provides an "off-the-shelf" standard ARM9E
solution
- ETM9 interface for real-time trace

The ARM966E-S core was designed with hard real-time applications as the primary objective. An example is servo-motor control in hard disk drives. The key feature of this CPU over the base ARM9E-S is the tightly coupled memory interface that allows selectable SRAM sizes of up to 64 MB.

ARM946E-S

- Cached processor for embedded real-time applications:
- MPU to support RTOS: like µITRON and VxWorks
- Selectable size instruction and data caches and
TCMs:(0 KB, 4 KB, 8 KB ... 1 MB)
- Instruction and data TCM interfaces.
- 150 MHz* on TSMC 0.18 µm

The ARM946E-S core takes the developments made by the 966E-S and adds selectable instruction and data caches. Since the memory protection unit is integrated with cache, this processor is an excellent High performance solution for embedded real-time applications, such as engine management systems in automobiles and network appliances.

ARM946E-S Caches

- Cache is 4-way set associative:
- Can be built with compiled ASIC RAM.
- Sizes of 0 KB, 4 KB, 8 KB ? 1 MB supported:
- I and D cache sizes are independently selectable.
- Cache lock-down on per-set basis:
- Granularity is a quarter of the cache size.
- Software selectable replacement algorithm:
- Supports pseudo-random and round-robin
- Write through and write back s/w selectable
- Line length fixed at 8 words

The cache memory blocks of this core are selectable up to 1 MB. The cache is 4-way set associative and selectable up to 1 MB. It also features lock-down support on a per-set basis, random and round robin replacement support, software selectable options for write through and write back, and eight word cache lines.

ARM9EJ-S Core Architecture

- 32-bit load/store RISC architecture

- Efficient 5-stage pipeline:
- Fetch
- Decode
- Execute
- Memory
- Write back
- ARM, Thumb and Java instruction sets
- 31 x 32-bit registers
- 32-bit ALU and barrel shifter
- Enhanced 32-bit MAC block
- ETM9 interface

The ARM926EJ-S core, with full MMU support and selectable tightly coupled memory and cache sizes, introduces a new generation of Internet-enabled devices. For example, set-top-boxes and wireless communications products benefit from this single processor solution. This processor can be compared to the ARM920T or 922T cores in its base functionality and performance.

Now, with the added Jazelle enhancements, Java functions can be performed without the need for complicated coprocessors or slow software implementations.

ARM10E Architecture Enhancements

ARM10E implements:

- Harvard 6-stage pipeline
- Supports v5TE instruction set
- EmbeddedICE RTII debug logic
- Fully compatible with v4T architecture
- 390-700 MIPS integer performance based on Dhrystone 2.1
- Branch prediction:
- Eliminates 70% of branches on typical code sequences
- Separate load/store unit:
- 64-bit path to register bank - load two registers simultaneously
- Hit-under-miss caches:
- Significantly reduces pipe-line stalls
- Write buffer:
- Holds up to 8 double-words (16 register values)
- New energy saving power down modes

Anticipating the market's needs for multimedia digital consumer devices, ARM developed the ARM10 family of advanced microprocessor cores with 390-700 MIPS integer performance. To achieve this performance, additional features were added. The pipeline was widened to add an additional stage, and improvements were made to the EmbeddedICE logic to provide support for realtime debug. All the while, compatibility was maintained with ARMv5TE and v4T for ease of code migration.

Performance enhancements include the introduction of branch prediction, hit-under-miss support in the MMU and cache architecture, an improved write buffer that holds up to eight double-words, and a separate load and store unit. These features improve code performance by lowering the average number of cycles per instruction of the processor, and also help when code is heavily dependent on cache operations.

As an added enhancement, the architecture, circuits, layout, and software controlled power-down modes have been developed specifically to achieve low-power operation on high-performance processes. These enhanced features have been optimized to take advantage of clock gating and dynamic power reduction.

High Performance Features

- 64-bit accesses to on-chip I and D caches:
- Fetch two instructions/cycle
- Load/store two registers/cycle (LDM/STM)
- Dual 64-bit fast AHB bus:
- Separate buses for instruction and data
- >1 Gbyte/sec bandwidth @ 200 MHz (each)
- Split transaction extensions
- 64-bit coprocessor interface:
- Load/store double-precision operands in one cycle
- 32-bit integer data path saves area and power

The ARM10E is also the first family of processors designed with a 64-bit data bus. This feature combines the frugal power and die size characteristics of a 32-bit CPU with the bandwidth requirements of high performance systems. The 64-bit coprocessor interface also allows for increased performance of floating point operations when combined with the Vector Floating Point-10 coprocessor.

ARM1020E and ARM1022E

- Highest performance ARM processor cores:
- 1.3 MIPS/MHz
- 1.5x ARM9 performance
- Support for High Performance IEEE 754 Floating Point:
- 600-1200 MFLOPS
- 300MHz (worst case) on TSMC 0.15 µm
- Low Power: 0.7 mW/mips (0.15 µm)
- ARM1020E: 32K I and D cache
- 17.5 mm$_2$
- ARM1022E: 16K I and D cache
- 12 mm$_2$
- Roadmap to Jazelle enhanced cores

The ARM1020E and ARM1022E processor cores offer the highest performance per unit of power of any 32-bit processor running above 200 MHz. With an unprecedented 0.7 mW/MIPS power consumption ratio, worst case on 0.15 µm process technology, these processors offers ideal solutions for high end platform applications. Examples include MPEG4 videophones, smartphones, and Web pads.

Memory Management and caches are comparable to the ARM920T, ARM922T and ARM720T products - ensuring code portability and protection for existing software investments. Future implementation in this family will also integrate the Jazelle Java enhancements established by the ARM9EJ-S family.

Vector Floating Point (VFP10)

- High-performance IEEE 754 floating point:
- Single and double precision
- Vector operations (up to 8 values per vector)
- Thirty-two 32-bit (SP) registers (usable as sixteen DP registers)
- Single cycle FMAC throughput (single precision

- double precision FMAC in 2 cycles)

- 10-100x performance increase over software emulation
- Optional coprocessor:
- 1.6 mm2 in 0.15 ?m
- Target:
- Printers, imaging, graphics, embedded control

Many real-time control applications in the industrial and automotive fields benefit from the dynamic range and precision of floating-point offered by the ARM VFP10. Automotive power train, anti-lock braking, traction control, and active suspension systems are examples of mission-critical applications where precision and predictability are essential requirements. Incorporating the ARM VFP10 into a SoC design can reduce development time and provide reliable performance. The vector processing capability of the ARM VFP10 also offers increased performance for imaging applications, such as scaling, transforms, and font generation used in printing, 3D transforms, FFT, and graphic filtering.