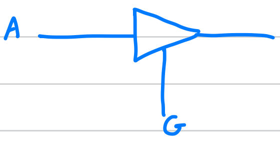
EICoM
ELECTRICAL COMPUTER MECHATRONICS

معالجات ومتحكمات دقيقة

د.محمد عبابنة

الطالب المبدع

محمد زغول



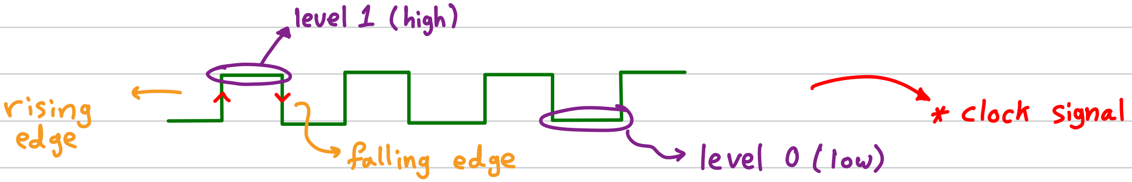
gated buffer
Active high

Allows info to go from A to Y
based on info from G
in the case of Active high, A goes to
Y if G is 1 (high), whereas in Active
low, A goes to Y if G is 0 (low).

A	G	Y
1	1	1
0	1	0
x	0	hi-Z

↳ high impedance

Kilobyte → 2^{10}
megabyte → 2^{20}
gigabyte → 2^{30}
terabyte → 2^{40}



Instruction cycles = 4 Tclk

$T_{clk} = \frac{1}{F_{osc}}$

ex: CLR → 1
self → 1
goto L_1 → 2
self portA → 1

∴ I.C = 5 x 4 x T_{clk}

Numbering systems

1. Binary number (0-1)
2. decimal number (0-9)
3. octal number (0-7)
4. hexadecimal (0-9)(A-F)

bit

nibble = 4 bits

byte = 8 bits

word = 16 bits

* How to convert from binary to Hexadecimal

1 Hex number takes 4 bits

$$(\underline{11}\,\underline{0\ 111}\overset{8421}{\underline{1110}}\ \overset{8421}{\underline{1110}})_2 = (37EE)_H$$

Decimal	Binary	Hexadecimal (H)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Microcontroller

- Micro processor
- prepherals

* we study PIC18FXX2

* Micro controller
  - I/o ports
  - Analog
  - Timers
  - pwm
  - serial port
  - memory

* Memory
  - program memory → for instructions
  - Data memory → for information
    - special function registers
    - General purpose register

* I/o ports
  - Digital I/o
  - Analog Input

I/O → input/output ports

Tris registers ⇒ configures the pins as input or output
($1 \rightarrow$ input, $0 \rightarrow$ output)

Lat registers ⇒ Connects the output pins to high or low voltage
($1 \rightarrow V_{dd}$, $0 \rightarrow V_{ss}$) ground

port registers ⇒ Reads the state of the pins
($1 \rightarrow V_{dd}$, $0 \rightarrow V_{ss}$)

→ There are 5 ports (A – E)

* EX: CLRF TRISA → TRISA = $\underbrace{0x}_{\text{Hex}}00$

* This way, we defined all pins on port A ($A_0 - A_7$) to be output

- port A ⇒ 7 bit ($A_0 - A_6$)
- port B ⇒ 8 bit ($B_0 - B_7$)
- port C ⇒ 8 bit ($C_0 - C_7$)
- port D ⇒ 8 bit ($D_0 - D_7$)
- port E ⇒ 3 bit ($E_0 - E_2$)

SETF TRISA
TRISA = 0X FF
$A_0 - 7_0$ are inputs

bcf TRISA, 3
bit clear file
11110111
↓
bit 3

($A_0 - A_2$) input
A_3 output
($A_4 - A_7$) input

to Read the new value on the port

bsf TRISB, 2
movf portB, 0 → Read the information and put it in the WREG

DATA RAM has → General Purpose RAM (GPR)
↳ Special Function Registers (SFRs)

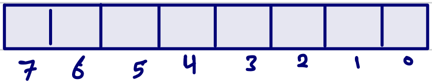
SFRs are fixed and must-have, GPR's size is what varies and changes the RAMS size.

GPR → for read/write and data manipulation
↳ sizes multiples of 256 B

* ROM is used to store the program

working register (WREG)

can store 8-bits



up to $2^8-1 = \underline{\underline{255}}$

used for arithmatic and logic operations

MOVLW

↳ moves 8-bit data into the working register

MOVLW X

size of x must be 8-bit

MOVLW 25H

↳ Hexadecimal

ADDLW

↳ ADD literal value K to WREG (K + WREG)

ADDLW k ⇒ takes the value stored in the WREG and adds k to that value then stores the resault back in the WREG

* Example

16H	MOVLW 12H	0	0	0	1	0	0	1	0	12H
0001 0110	ADDLW 16H	0	0	1	0	1	0	0	0	28H
0001 0001	ADDLW 11 H	0	0	1	1	1	0	0	1	39 H
0100 0011	ADDLW 43H	0	1	1	1	1	1	0	0	7C H

* After running the above code / instruction what's the value stored in the WREG? → 7CH

FIGURE 4-1: PROGRAM MEMORY MAP AND STACK FOR PIC18F442/242

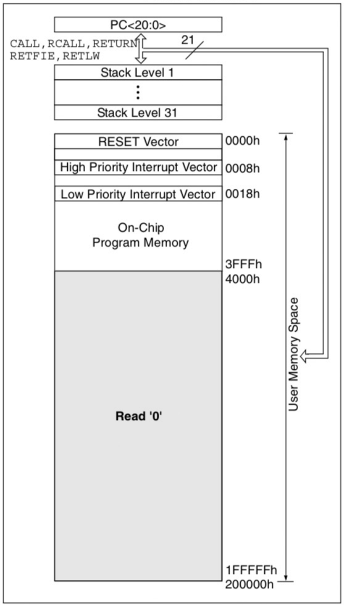
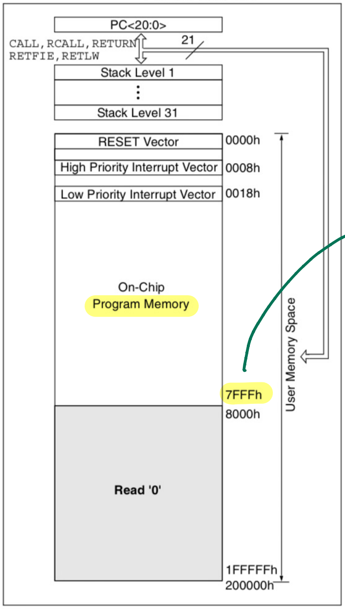


FIGURE 4-2: PROGRAM MEMORY MAP AND STACK FOR PIC18F452/252



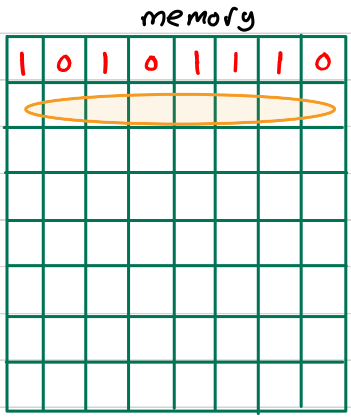
→ maximum address locations.

The PIC File Register = Data Memory

↳ Read / write (Static RAM)

→ a collection of registers

* Used for data storage



each location is considered a register and each location has an address →

$(111)_2$

7 address locations

3 address lines / bits → number of bits i need to represent the last address location.

Special Function Registers

↳ a bunch of registers inside the data memory.

* specific functions as timers, I/o ports, serial communication..etc.

The file register in PIC18 has maximum 4096 byte (4K), which has addresses 000-FFFH which is divided by banks, each bank has 256 bytes

$2^{12} = 4096$ bytes

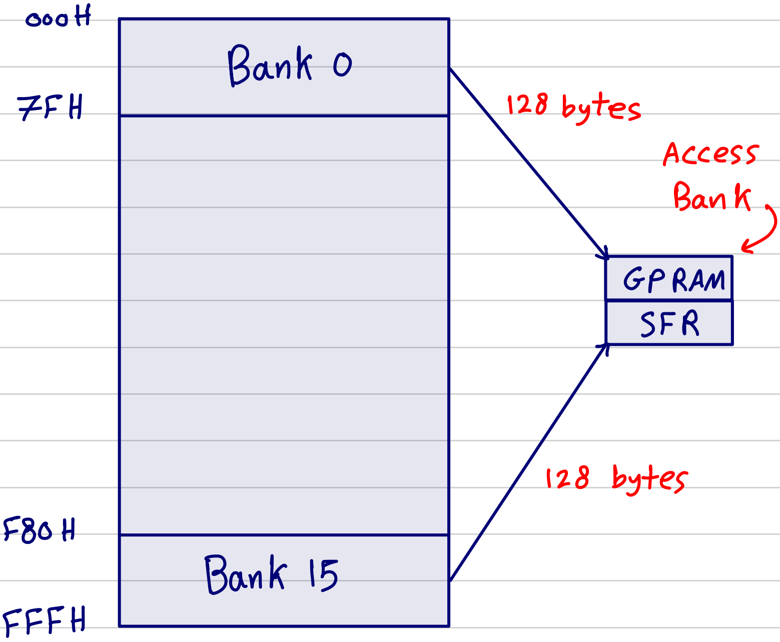
locations

$$\frac{4096}{256} = 16$$

maximum number of banks available

* There is at least 1 bank called the access bank which is the default bank.

* How can we access all banks? by a method called Bank switching



* The 256-byte access bank is divided by 2 equal sections of 128 bytes

*
- → Byte-oriented Instructions
- → Bit-oriented Instructions
- → control Instructions

* W → working register
F → File register

* Using instructions with the default access bank (to access other locations in the file register for ALU and other operations

MOVWF

→ tells the cpu to copy the source register (WREG) to a destination in the file register

MOVWF address
- → Can be in SFR or GP RAM
- → the address of the destination

* Example

```
MOVLW  99H
MOVWF  12H
MOVLW  85H
MOVWF  13H
MOVLW  3FH
MOVWF  14H
MOVLW  63H
MOVWF  15H
```

copies the contents ← MOVWF 12H
of WREG to a File register that has address 12 H

WREG
99 H
85H
3F H
63H

Address	Data
012 H	
013 H	
014 H	
015 H	

empty

Address	Data
012 H	99 H
013 H	85 H
014 H	3F H
015 H	63 H

* Note: we cannot move literal values (numbers) directly into the general purpose RAM location. it must be moved via WREG

ADDWF

↳ Adds together the content of WREG and a file register location

* takes two parameters: Address, D

↓

from Register File

D → 0, w or not specified → WREG

Result will be saved in WREG

↳ 1, F → address

Result will be saved in the same address

* Example: What are the contents of the WREG after running the following program

MOVLW 0	00H	{12H} = 00H
MOVWF 12H	00H	{12H} = 00H
MOVLW 22H	22H	{12H} = 00H
ADDWF 12H, F	22H	{12H} = 22H
ADDWF 12H, F	22H	{12H} = 44H
ADDWF 12H, F	22H	{12H} = 66H
ADDWF 12H, F	22H	{12H} = 88H

↑

contents of WREG after each instruction

COMF

↳ complements the contents of file register and places the result in WREG

COMF address, D → 1, F
↳ 0, W

* Example : write a simple program to toggle the SFR of portB continuosly forever.

```
      MOVLW   55H
      MOVWF   portB      → copies 55H from WREG to portB
   B1 COMF    portB,F
      GOTO    B1
```

LABEL ↙ B1

GOTO B1 → will go to this line (that has the B1 label)

* portB is just an address location that has a name, its address is F81H so we can also do MOVWF F81H which would be the same.

COMF portB,F → Complement the contents of portB

if was → 01010101

then becomes → 10101010 and gets stored in the File register (portB itself)

MOVF

MOVF address, D

→ D = 0 ∴ copies the contents of file register (from I/o pin) to WREG

→ 0 = 1 ∴ The content of file register is copied to itself

* Example: write a simple program to get data from the SFRs of portB and send it the SFRs of portC continuously

```
AGAIN  MOVF  PortB, w
       MOVWF portC
       GOTO  AGAIN
```

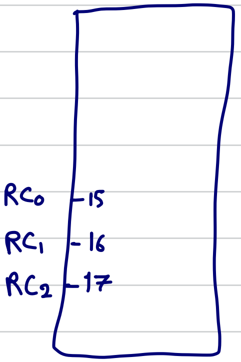
* To move values from portA to port B

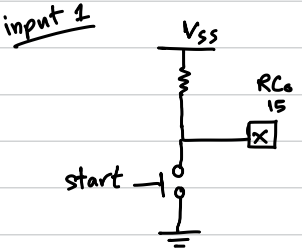
①
```
MOVFF portA, port B
```

or → ②
```
MOVF  portA, w
MOVWF portB
```

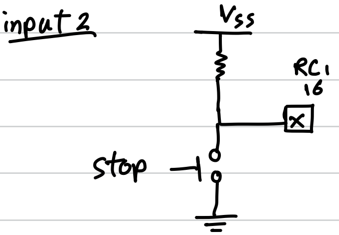
Input : 2 digital
output : 1 output

$RC_0 \Rightarrow$ start
$RC_1 \Rightarrow$ stop } → input (push Button)
$RC_2 \Rightarrow$ LED → output

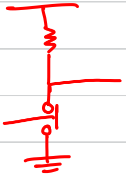


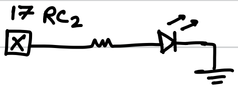


Active
low cct



active
high cct ↪





Main :

```
    ; Initialization
    bsf   TRISC, 0
    bsf   TRISC, 1
    bcf   LATC,2
    bcf   TRISC, 2
    CLRF  TRISD
```

can also be
done by :

```
bcf    LATC, 2
MoVlW  0x03
MoVWF  TRISC
CLRF   LATC
```

```
L1  btfsc   portc, 0
    goto    L1
    bsf     portc, 2
```

← to search for 'ON command'

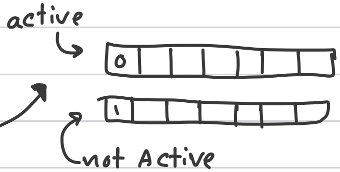
```
L2  btfsc   portc, 1
    goto    L2
    bcf     portc, 2
    goto    L1

    end.
```

* weak pull-up only available in portB
(which means internally there is a current limiting resistor)

* it's better to clear the LAT before defining any output incase it has anything from a previous file.

SFR ⇒ INTCON2

bcf INTCON2, 7 → Initialization

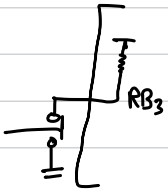


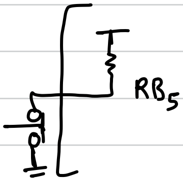
RB3 ⇒ Input → start

RB5 ⇒ Input → stop

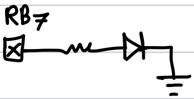
RB7 ⇒ output → LED

we will use weak pull-up





active High buttons



```
Main:
      ; Initialization
      bsf  TRISB,3
      bsf  TRISB,5
      bcf  LATB,7
      bcf  TRISB,7
      CLRF TRISD

L1    btfss   portB,2
      goto    L1
      bsf     portB,7

L2    btfss   portB,5
      goto    L2
      bcf     portB,7
      goto    L1

      end.
```

```
                                    I.C
delay                               2
     Movlw    N                     1
     Movwf    count                 1

L2   decfsz   count                 (N-1)*1+3
     goto  L2                       (N-1)*2
     Nop                            1
     Return                         2

                                    3N+7
```

assume $f_{osc} = 1\,MHz$

$T_{clk} = 1\,\mu s$

$T_{I.c} = 4\,T_{clk} = 4\,\mu s$

128 64 32 16 8 4 2 1

maximum value of N can be 256

∴ No I.C = 775

T_{Total} = No.I.C * $T_{I.c}$ = 3.1 mS

but what if we want to delay more? we can add another loop

```
                                    I.C
delay                               2
     Movlw    M
     Movf     count2
L3   Movlw    N                     1
     Movwf    count                 1          * M-1

L2   decfsz   count                 (N-1)*1+3
     goto  L2                       (N-1)*2
     decfsz   count2                (M-1)*1+3
     goto     L3                    (M-1)*2

     Return                         2
```

M= 2

N= 2

M = 1

DESCRIPTION

The HEF4511B is a BCD to 7-segment latch/decoder/driver with four address inputs (D_A to D_D), an active LOW latch enable input ($\overline{EL}$), an active LOW ripple blanking input ($\overline{BI}$), an active LOW lamp test input ($\overline{LT}$), and seven active HIGH n-p-n bipolar transistor segment outputs (O_a to O_g).

When $\overline{EL}$ is LOW, the state of the segment outputs (O_a to O_g) is determined by the data on D_A to D_D. When $\overline{EL}$ goes HIGH, the last data present on D_A to D_D are stored in the latches and the segment outputs remain stable. When $\overline{LT}$ is LOW, all the segment outputs are HIGH independent of all other input conditions. With $\overline{LT}$ HIGH, a LOW on $\overline{BI}$ forces all segment outputs LOW. The inputs $\overline{LT}$ and $\overline{BI}$ do not affect the latch circuit.

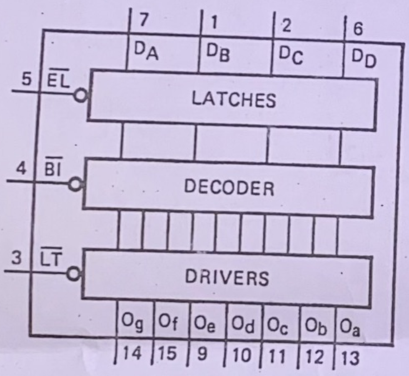


Fig.1 Functional diagram.

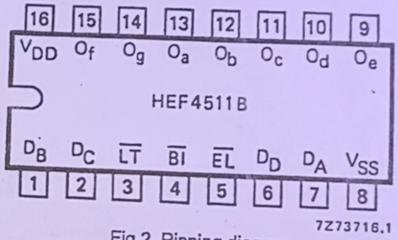


Fig.2 Pinning diagram.

HEF4511BP(N):	16-lead DIL; plastic (SOT38-1)
HEF4511BD(F):	16-lead DIL; ceramic (cerdip) (SOT74)
HEF4511BT(D):	16-lead SO; plastic (SOT109-1)

( ): Package Designator North America

PINNING

D_A to D_D	address (data) inputs
$\overline{EL}$	latch enable input (active LOW)
$\overline{BI}$	ripple blanking input (active LOW)
$\overline{LT}$	lamp test input (active LOW)
O_a to O_g	segment outputs

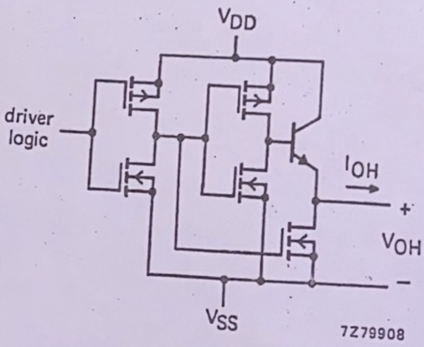


Fig.3 Schematic diagram of output stage.

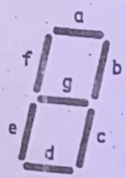


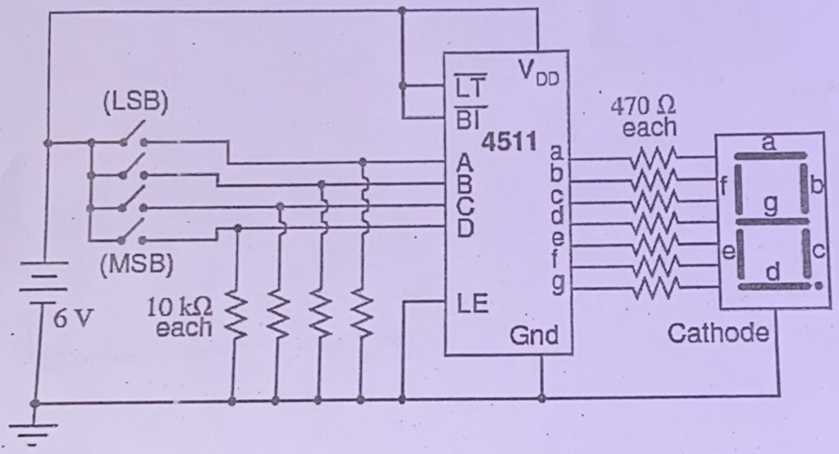
Fig.4 Segment designation.

FAMILY DATA, I_{DD} LIMITS category MSI

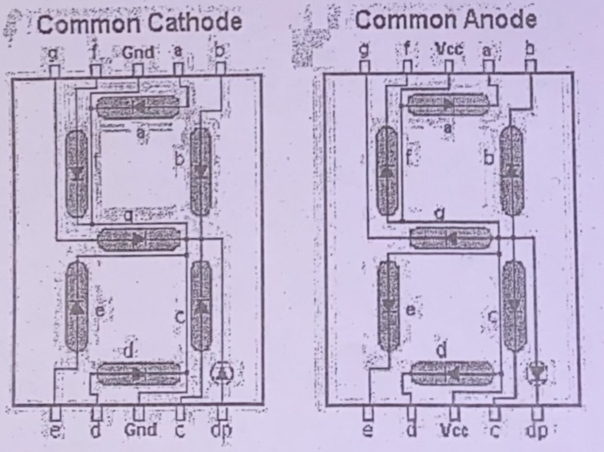
See Family Specifications

Design a circuit and write a code for a stopwatch timer. The maximum time that it is count 60s. The result should be display on a seven segment

SCHEMATIC DIAGRAM



active High



active low

Figure 9 – Common Cathode & Common Anode

ex: $45_{10} \longrightarrow (0100\ \ 0101)_{BCD}$

How can we get this using code ?

we need to count the number of 10s

w= 45

$$\begin{array}{r} 45\ - \\ 10 \\ \hline 35 \end{array}$$

x = 1 first 10

$$\begin{array}{r} 35\ - \\ 10 \\ \hline 25 \end{array}$$

x = 2

$$\begin{array}{r} 25\ - \\ 10 \\ \hline 15 \end{array}$$

x = 3

$$\begin{array}{r} 15\ - \\ 10 \\ \hline 5 \end{array}$$

x = 4

$$\begin{array}{r} 5\ - \\ 10 \\ \hline -5 \end{array}$$

since it's negative we add a 10 back and save the value in WREG

w = -5 + 10 = 5

SWAP X

$\boxed{x = 4} \rightarrow 0000\ \ 0100$

$\boxed{w = 5} \rightarrow 0000\ \ 0101$

After swap:

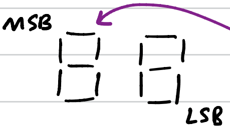
$\boxed{x = 40} \rightarrow 0100\ \ 0000$

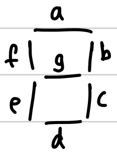
$\boxed{w = 5} \rightarrow 0000\ \ 0101$

we do XOR between x and w

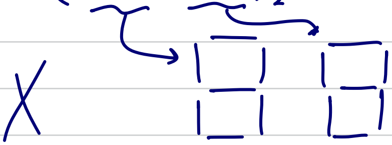
$= \underbrace{0100}_{4}\ \underbrace{0101}_{5}\ BCD$

2 7-Segments





$(20)_{10} = (\overset{10-90}{0001}\ \overset{0-9}{0100})_2$



we use BCD $(20)_{10} = (\underbrace{0010}_{2}\ \underbrace{0000}_{0})_{BCD}$

$(59)_{10} = (0101\ 1001)_{BCD}$

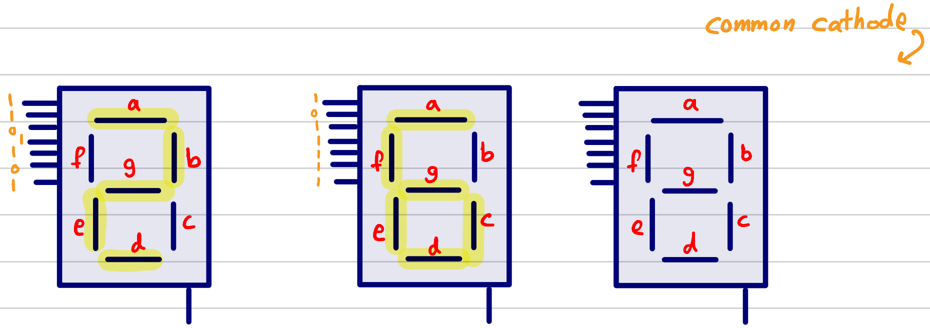
BCD

```
      Movwf    BCD1
      CLRF     w
      CLRF     count
      MOVLW    (10)10
L1    Subwf    BCD1,1    ; BCD1 = BCD1 - W
      BN       L2        ; will check if negative bit is 1, if it is
                           then branch to L2 (same as goto)
      INCF     count
      goto     L1

L2    ADDwf    BCD1, w   ; w = BCD1 + w
      SWAPF    count, F
      XORWF    count, F
      MovFF    count, portB
      return
```

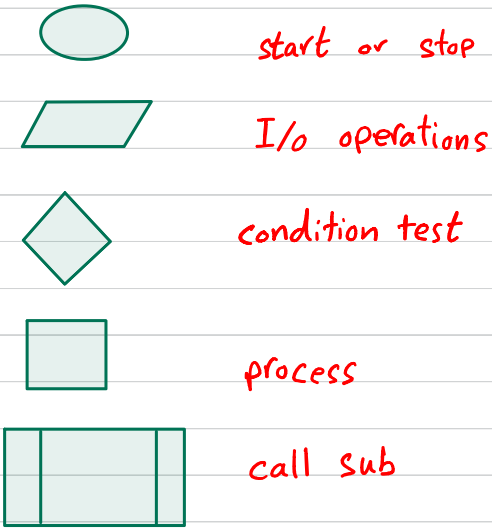
	Common Anode	Common cathode
Digit	a b c d e f g	a b c d e f g
0	0 0 0 0 0 0 1	1 1 1 1 1 1 0
1	1 0 0 1 1 1 1	0 1 1 0 0 0 0
2	0 0 1 0 0 1 0	1 1 0 1 1 0 1
3	0 0 0 0 1 1 0	1 1 1 1 0 0 1
4	1 0 0 1 1 0 0	0 1 1 0 0 1 1
5	0 1 0 0 1 0 0	1 0 1 1 0 1 1
6	0 1 0 0 0 0 0	1 0 1 1 1 1 1
7	0 0 0 1 1 1 1	1 1 1 0 0 0 0
8	0 0 0 0 0 0 0	1 1 1 1 1 1 1
9	0 0 0 0 1 0 0	1 1 1 1 0 1 1

↑ we need to supply Vcc and (Low) to turn on the segment

↑ we need to supply ground and (High) to turn on the segment



Flow charts

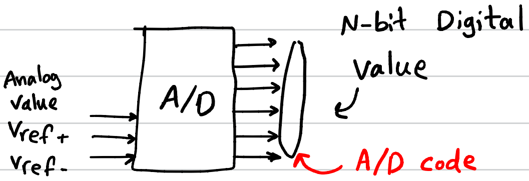


A/D

In PIC18FXX2 microprocessor, there is a built-in analog to digital module to convert an analog input signal (ex: voltage) to a 10-bit digital number.

* computers can only understand digital signals.

→ Resolution: number of discrete values produced over the range of analog values.



*Note:- AD code : the output of the A/D convertor Described by number of output bit.

$$\text{AD code} = \frac{2^N - 1}{V_{ref+} - V_{ref-}} * (V_{in} - V_{ref-})$$

$[\underset{\text{Low}}{V_{ref-}} , \underset{\text{high}}{V_{ref+}}]$ ⇒ The range of analog value that the A/D convertor can detect

Voltage resolution ≡ The smallest change in voltage an ADC can detect

$$\text{Voltage resolution} = \frac{V_{ref+} - V_{ref-}}{\boxed{2^N}}$$; N : number of bits

Quantization levels

* our ADC has a 10-bit resolution so N= 10.

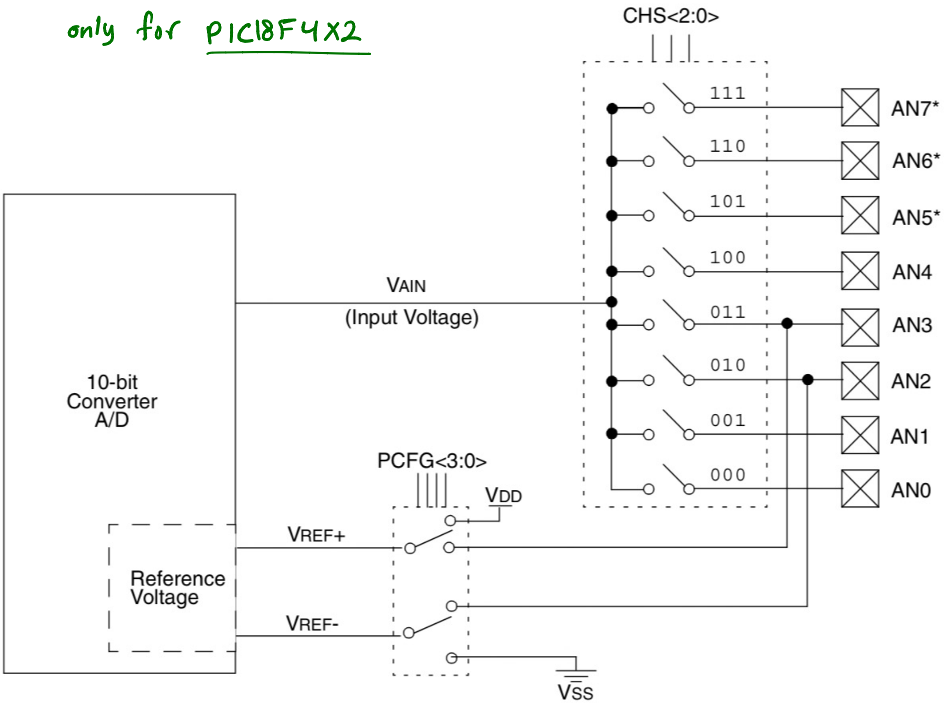
Example: Full scale measurment 0 → 10 v

ADC Resolution is 12 bit ⇒ 4096

ADC Voltage resolution ⇒ 10 - 0 /4096 = 0.00244 V

Vref → Internal (inside the PIC) $V_{ref+} \rightarrow +5V$, $V_{ref-} \rightarrow 0V$

Vref → External (must be provided by the user) only for PIC18F4X2



* The result of conversion is stored in SFRs → ADRESH
→ ADRESL

* SFRs used :-

TRISA , TRISE] for initialization
ADCON0 , ADCON1]

ADRESL] store the Result (stores the ADCODE)
ADRESH]

* we need to define port A and port E as input or output and as Analog or digital.

17.0 COMPATIBLE 10-BIT ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) converter module has five inputs for the PIC18F2X2 devices and eight for the PIC18F4X2 devices. This module has the ADCON0 and ADCON1 register definitions that are compatible with the mid-range A/D module.

The A/D allows conversion of an analog input signal to a corresponding 10-bit digital number.

SFRs

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1) *

controlles the operation of the A/D module

The ADCON0 register, shown in Register 17-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 17-2, configures the functions of the port pins. *

REGISTER 17-1: ADCON0 REGISTER

bit resolution

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

Don't care just put 0.

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	FOSC/2
0	**01**	FOSC/8
0	**10**	FOSC/32
0	**11**	FRC (clock derived from the internal A/D RC oscillator)
1	**00**	FOSC/4
1	**01**	FOSC/16
1	**10**	FOSC/64
1	**11**	FRC (clock derived from the internal A/D RC oscillator)

Table 1

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

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)

we choose and based on it, we specify the channel select bits.

Note: The PIC18F2X2 devices do not implement the full 8 A/D channels; the unimplemented selections are reserved. Do not select any unimplemented channel.

bit 2 **GO/DONE:** A/D Conversion Status bit

When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

→ conversion from Analogue to Digital

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit
1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

PCFG (0-3)

REGISTER 17-2: ADCON1 REGISTER

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	— X	— X	PCFG3	PCFG2	PCFG1	PCFG0

bit 7 bit 0

Do not care, just put 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in **bold**)

Table 1

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

if there is no VREF- or VREF+, it means it's internal

external VREF+ at AN3

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C / R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8 / 0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7 / 1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5 / 0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4 / 1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3 / 0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2 / 1
011x	D	D	D	D	D	D	D	D	—	—	0 / 0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6 / 2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6 / 0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5 / 1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4 / 2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3 / 2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2 / 2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1 / 0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1 / 2

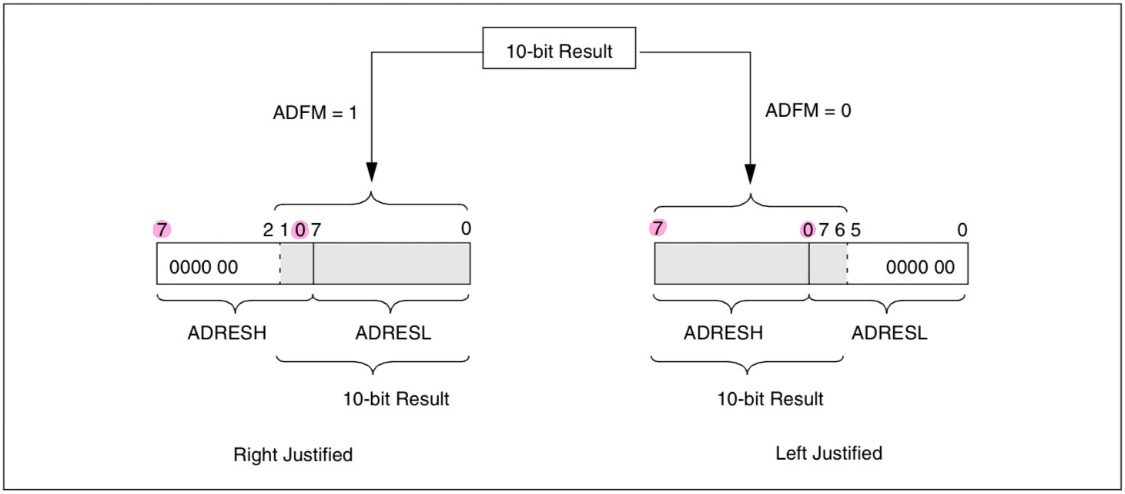
A = Analog input D = Digital I/O
C/R = # of analog input channels / # of A/D voltage references

Table 2

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: On any device RESET, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

FIGURE 17-4: A/D RESULT JUSTIFICATION



* Initialization procedure :

1. choose the analog pins → TRISA, TRISE
   → ADCON1 (0:3)

Bits from 0 to 3

2. choose Vref internal or external → ADCON1 (0:3)

3. choose A/D conversion clock → ADCON0 (7,6)
   → ADCON1 (6)

$T_{AD} \geqslant 1.6 \mu s$

$X \cdot T_{osc} \geqslant 1.6 \mu s$

Factor

* TAD is the time that the A/D convertor needs to finish converting from analog to digital. and in PIC18FXX2 it must be AT LEAST 1.6 Ms.

Example: $F_{osc} = 4\,MHz$

$$\therefore T_{AD} = X \cdot T_{osc} = X \cdot \frac{1}{4MHz} = X \cdot 0.25\,\mu s \geqslant 1.6\,\mu s$$

$$X \geqslant 6.4$$

* out of the options in Table 1, we can choose x to be 8, 16, 32, 64 which any of them works but the best thing is to choose the smallest.

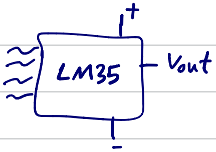
$\therefore X = 8 \Rightarrow F_{osc}/8$

→ we take ADCS0, ADCS1 and ADCS2 bits that corresponds to Fosc/8 and put them in bits 6 and 7 of ADCON0 and bit 7 of ADCON1

Q: in the A/D module for 18F452 what is the maximum frequency for the converion clock ($F_{osc} = 4MHz$)

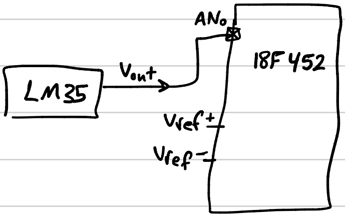
maximum → Fosc / 8

A/D Example :- a temperature sensor LM35 is used to measure the Room temp. the sensor gain is 10 mV/c° and the temp. range is from (0 - 100°c)

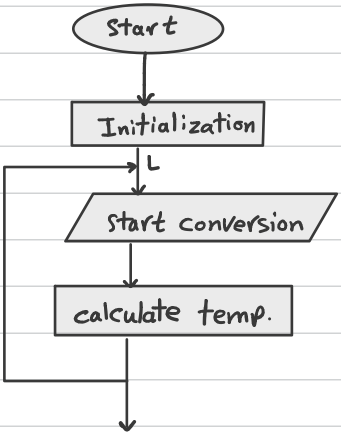


Sensor Voltage range =
temp range x sensor gain
= 0 - 1 v

Input is temp
output is Voltage



which is the input to the microcontroller.



- Read the analog value and store the temp in a register called "temperature".

Initialization procedure:

① we need to determine Vref, do we want internal or external.

Internal Vref ⇒ Voltage resolution = $5/1024$ = 0.0048 V/ADCODE
4.8 mV/ADCODE

⇒ ∴ temp resolution → $\dfrac{4.8\ \text{mV/ADCODE}}{10\ \text{mV}/C^\circ}$
= 0.48 C°/ADCODE

External voltage ⇒ (we need to choose the value)
let $V_{ref-} = 0$ and $V_{ref+} = 1$

∴ Voltage resolution = $1/1024$ = 0.00097 V/ADCODE
= 0.97 mV/ADCODE

∴ temp resolution = 0.097 c°/ADcode

☆ So we will choose the External voltage since it gives higher precision

② calculate conversion clock where fosc = 4MHz

$$T_{AD} \geq 1.6\,\mu s \qquad X \cdot T_{clk} \geq 1.6\,\mu s$$

$$X \cdot \frac{1}{4}\,\mu s \geq 1.6\,\mu s \qquad X \geq 6.4 \qquad \boxed{\therefore\ X = 8}$$

from the Data sheet

ADCS2	ADCS1	ADCS0	Fosc/8
0	0	1	←

③ choose the analog pin that will be used

⇒ we will choose analog pin AN0

* in this case, we could've used AN2 since we don't want an external Vref- BUT, since we want an external Vref+ then we can't use AN3.

⇒ now we need to choose a PCFG (from Table 2) that has an analog pin at AN0 and an external Vref+ at AN3. Available options : 0001

0011 ✓ ⟵ I can choose any.

0101

1010

④ ADFM

→ we want to store the conversion output using Left justification, so ADFM = 0

CS1	CS0	CHS2	CHS1	CHS0			
0	1	0	0	0	0	X_0	1

ADCON0

4 1 h

ADFM	CS2			PCFG3			PCFG0
0	0	X_0	X_0	0	0	1	1

ADCON1

0 3 h

* CHS2:CHS0 are analog channel select bit, which we get based on the analog pin we chose.

AN0 → channel 0 = 0 0 0

CH2 CHS0

→ calculate the relation between ADCODE and the room temperature

Let's say that the temp is 100°c which means that $V_{in} = 1$ Volt $\Rightarrow$ ADCODE $= (1111111111)_2 = (1023)_{10}$
but how will we read the temp from the ADCODE?
we need a way that converts ADCODE to temp.

$V_{in} =$ sensor gain * Room temp

$V_{in} = 10 \times 10^{-3} * T$

The input Voltage to the PIC which is the output voltage of the temp sensor.

$$\text{ADCODE} = \frac{2^{10}-1}{\boxed{1-0}} * V_{in} = 1023 * 10*10^{-3} * T$$

$V_{ref+} - V_{ref-}$

$$T = \frac{\text{ADCODE}}{1023 * 10 * 10^{-3}} = \frac{\text{ADCODE}}{10.23} \approx \frac{\text{ADCODE}}{10}$$

* Note: In the instruction set there is No divesion so will use RR (Rotate Right) ⇒ shift Right X by one bit $= X/2$

now we need to find a way to do ADCODE/10

RR 3 time → 5 times → 6 times

$$\frac{X}{10} = \frac{X}{8} - \frac{X}{32} + \frac{X}{64}$$

$0.125X \quad 0.03125X \quad 0.015625X \quad = 0.109X$

This is close to 0.1X that we're looking for but we can be more accurate.

```
Main:
   ; Initialization
   bsf TRISA, 0
   Movlw   0x41
   Movwf   ADCON0
   Movlw   0x03
   Movwf   ADCON1
L1 bsf     ADCON0, 2
L2 btfsc   ADCON0, 2
   goto    L2
   Movff   ADRESH, Value 1      ⇒ ADCODE/4
   RRNC    Value 1, F   }  Both of these will
   bcf     Value 1, 7   }  do a shift right       ⇒ ADCODE/8
   Movff   Value 1, Value 2
   RRNC    Value2, 1  }
   bcf     value2, 7  }  ⇒ ADCODE/16
   RRNC    Value 2, 1 }
   bcf     Value2, 7  }  ⇒ ADCODE/32
   Movff   value2, value3
   RRNC    value 3, 1 }
   bcf     value 3, 7 }  ⇒ ADCODE/64
   Movf    Value2, w                               Value2
   Subwf   Value1, w    →  w = Value1 - w
   ADDwf   Value3, w    →  w = Value3 + w = Value3 + Value1 - Value2
   Movwf   Temperature  ⇒  ≈ ADCODE/10
```

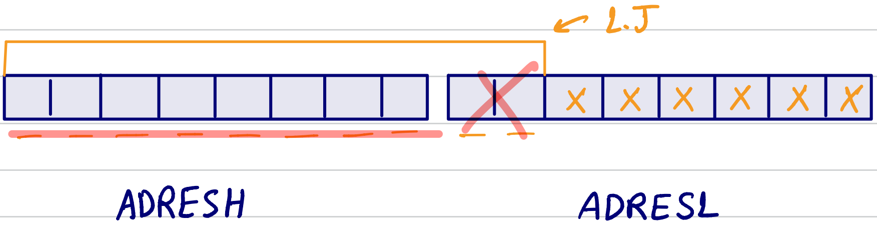
Value 1 = ADCODE/8
Value 2 = ADCODE/32
Value 3 = ADCODE/64

pin 0 of port A is set to be an input.

Some code explanations:-

①
```
Movff ADRESH, Value 1  ⇒ ADCODE/4
```

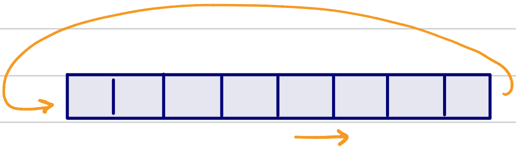
when initializing, we chose Left justification which would look like:



Usally when we do a shift Right (for division) we lose that first bit.
when we took only the contents of ADRESH, it's like we lost the 2 first bits (shift right twice) which means divided by 2 twice ∴ ADCODE/4.

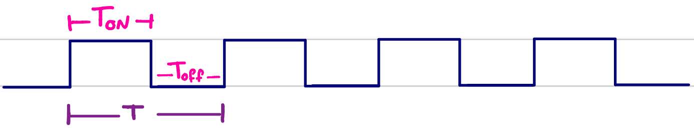
②
```
RRNC   Value 1, F  }
bcf    Value 1, 7  }  ⇒ ADCODE/8
```

in the instruction set, there is no shift right, but there is a rotate right.

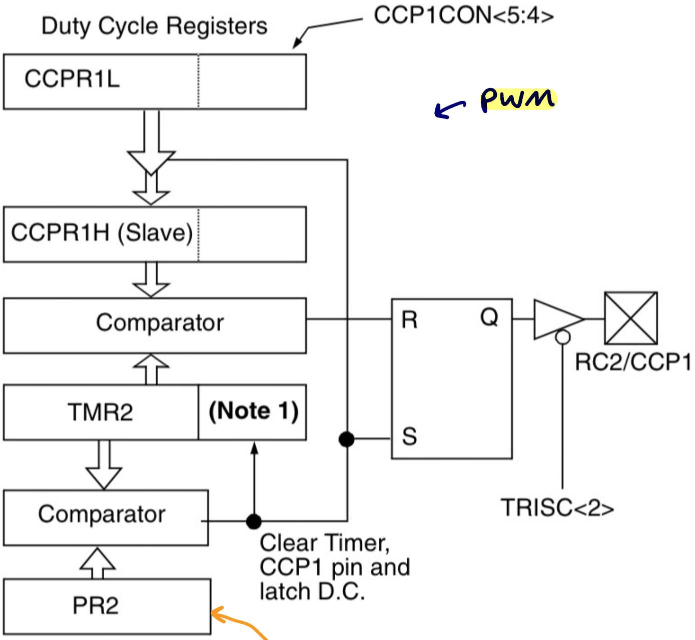


* everything shifts to the right, but the first bit goes to the last bit so we need to clear that last bit to do a real shift.

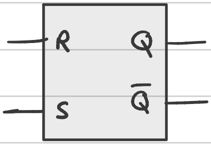
pulse width modulation (PWM)



$$Duty\ cycle = \frac{T_{ON}}{T}$$



← PWM



S	R	Q
0	1	0
1	0	1

← Built-in in PIC 18FXX2

must be less than 255 so we can put it in a 8-bit register

SFR's :

* <3:0> → bits
3,2,1,0

TRISC ,2

RC2 → The only pin that can output the pwm's waveform.

T2CON → prescale value <1:0>
T2CON → TMR2ON <2>

CCP1CON → LSB's of Duty cycle <5:4>
CCP1CON → pwm module <3:0>

CCPR1L → MSB's of Duty cycle

PR2 → A period register in the timer module.

$$T_{pwm} = [PR2 + 1] * 4 * T_{osc} * prescale$$

$$T_{ON} = [X] * T_{osc} * prescale$$

$$X \Rightarrow [\overset{b_9}{C}CPR1\overset{b_2}{L} : CCP1CON < \overset{b_1}{5} : \overset{b_0}{4} >]$$ a 10 bit number.

Example: $f_{osc} = 1\,MHz$ $T_{pwm} = 30\,\mu s$ Duty cycle = 50%

$$30\cancel{\mu}s = [\overset{??}{PR2} + 1] * 4 * 1\cancel{\mu}s * \overset{??}{prescale}$$

$$7.5 = [PR2 + 1] * prescale$$

$$PR2 = 6.5$$

→ if > 255 we should think of a prescale value that is larger than 1.

→ we don't have 0.5 so either PR2 = $\underline{7}$ or 6

* we choose the prescale Value
- 1 : 1
- 1 : 4
- 1 : 16

$T_{ON} = [X] * T_{osc} * \text{prescale}$

$15\,\mu s = [x] * 1\,\mu s * 1$

Duty cycle = 50%

$= 0.5 = \frac{T_{ON}}{T}$

$\therefore T_{ON} = 15\,\mu s$

$X = 15 = \underbrace{\overset{=3h}{00000011}}_{CCPR1L}\underbrace{11}_{CCP1CON<5:4>}$

Main:

```
bcf     Latc,2
bcf     TRISC,2
Movlw   0x7
Movwf   PR2
Movlw   0x03
movwf   CCPR1L
Movlw   0x3c
Movlw   CCP1CON
clRF    T2CON
bsf     T2CON,2
```

→ to get 00000100 and turn the timer ON.

T2CON

0					TMR2ON		
X	0	0	0	0	1	0	0
	postscale					prescale:	

prescale:

- 00 → 1:1
- 01 → 1:4
- 1X → 1:16

CCP1CON

				pwm mode			
X⁰	X⁰	1	1	1	1	X⁰	⁰X
3				c			

Example: $T_{pwm} = 300\,\mu s$ $f_{osc} = 4\,MHz$ Duty cycle = 50%

$$300\,\cancel{\mu s} = [PR2+1] * \cancel{4} * \frac{1}{\cancel{4}}\,\cancel{\mu s} * \text{prescale}$$

→ if we choose prescale 1:1 then PR2 = 299 which is > 255 so we need to choose another prescale, if we chose 1:4 then PR2 = 74 but if we chose 1:16 then PR2 = 17.75 we will choose a 1:4 prescale since it results in PR2 being an integer.

$$T_{ON} = [X] * T_{osc} * \text{prescale}$$

$$150 \cancel{\mu s} = [x] * \frac{1}{\cancel{4}} \cancel{\mu s} * \cancel{4}$$

$$X = 150 = \underbrace{00100101}_{25h}\underbrace{10}$$

T2CON	X^0	0	0	0	0	1	0	1
CCP1CON	X^0	X^0	1	0	1	1	X^0	X^0

Main:

```
bcf    LATC,2
bcf    TRISC,2
MovLw  0X4A
Movwf  PR2
Movlw  0X2C
movwf  ccp1coN
MovLw  0X25
Movwf  ccpR1L
movlw  0x05
Movwf  T2CON
```

12.0 TIMER2 MODULE

The Timer2 module timer has the following features:

- 8-bit timer (TMR2 register)
- 8-bit period register (PR2)
- Readable and writable (both registers)
- Software programmable prescaler (1:1, 1:4, 1:16)
- Software programmable postscaler (1:1 to 1:16)
- Interrupt on TMR2 match of PR2
- SSP module optional use of TMR2 output to generate clock shift

Timer2 has a control register shown in Register 12-1. Timer2 can be shut-off by clearing control bit TMR2ON (T2CON<2>) to minimize power consumption. Figure 12-1 is a simplified block diagram of the Timer2 module. Register 12-1 shows the Timer2 control register. The prescaler and postscaler selection of Timer2 are controlled by this register.

12.1 Timer2 Operation

Timer2 can be used as the PWM time-base for the PWM mode of the CCP module. The TMR2 register is readable and writable, and is cleared on any device RESET. The input clock (FOSC/4) has a prescale option of 1:1, 1:4 or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>). The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit TMR2IF, (PIR1<1>)).

The prescaler and postscaler counters are cleared when any of the following occurs:

- a write to the TMR2 register
- a write to the T2CON register
- any device RESET (Power-on Reset, $\overline{\text{MCLR}}$ Reset, Watchdog Timer Reset, or Brown-out Reset)

TMR2 is not cleared when T2CON is written.

REGISTER 12-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
— X	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0
0	0	0	0	0		prescale	

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

not used in pwm
so just put 0000

bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

14.0 CAPTURE/COMPARE/PWM (CCP) 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. Table 14-1 shows the timer resources of the CCP Module modes.

The operation of CCP1 is identical to that of CCP2, with the exception of the special event trigger. Therefore, operation of a CCP module in the following sections is described with respect to CCP1.

Table 14-2 shows the interaction of the CCP modules.

REGISTER 14-1: CCP1CON REGISTER/CCP2CON REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
— x	— X	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7 0	0						bit 0

← The LSBs of the Duty cycle

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **DCxB1:DCxB0**: PWM Duty Cycle bit1 and bit0

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode,
Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
1001 = Compare mode,
Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
1010 = Compare mode,
Generate software interrupt on compare match (CCPIF bit is set, CCP pin is unaffected)
1011 = Compare mode,
Trigger special event (CCPIF bit is set)
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

14.5 PWM Mode

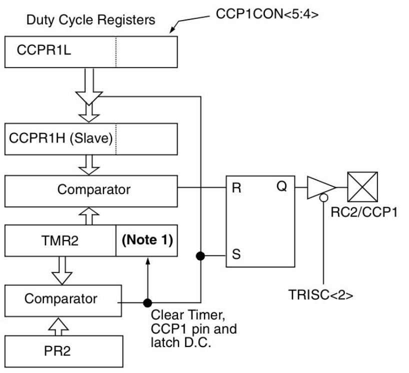
In Pulse Width Modulation (PWM) mode, the CCP1 pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with the PORTC data latch, the TRISC<2> bit must be cleared to make the CCP1 pin an output.

> **Note:** Clearing the CCP1CON register will force the CCP1 PWM output latch to the default low level. This is not the PORTC I/O data latch.

Figure 14-3 shows a simplified block diagram of the CCP module in PWM mode.

For a step-by-step procedure on how to set up the CCP module for PWM operation, see Section 14.5.3.

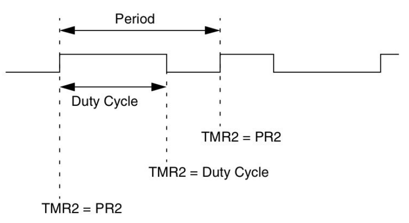
FIGURE 14-3: SIMPLIFIED PWM BLOCK DIAGRAM



Note: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

A PWM output (Figure 14-4) has a time-base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

FIGURE 14-4: PWM OUTPUT



14.5.1 PWM PERIOD

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

$$\text{PWM period} = (\text{PR2}) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 prescale value})$$

PWM frequency is defined as 1 / [PWM period].

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

- TMR2 is cleared
- The CCP1 pin is set (exception: if PWM duty cycle = 0%, the CCP1 pin will not be set)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

> **Note:** The Timer2 postscaler (see Section 12.0) is not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

14.5.2 PWM DUTY CYCLE

The PWM duty cycle is specified by writing to the CCPR1L register and to the CCP1CON<5:4> bits. Up to 10-bit resolution is available. The CCPR1L contains the eight MSbs and the CCP1CON<5:4> contains the two LSbs. This 10-bit value is represented by CCPR1L:CCP1CON<5:4>. The following equation is used to calculate the PWM duty cycle in time:

~~PWM duty cycle~~ Ton = $(\text{CCPR1L:CCP1CON<5:4>}) \cdot T_{OSC} \cdot (\text{TMR2 prescale value})$

CCPR1L and CCP1CON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR1H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). 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 the TMR2 prescaler, the CCP1 pin is cleared.

The maximum PWM resolution (bits) for a given PWM frequency is given by the equation:

$$\text{PWM Resolution (max)} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)}\ \text{bits}$$

> **Note:** If the PWM duty cycle value is longer than the PWM period, the CCP1 pin will not be cleared.

Interrupts

→ an interrupt is a sub routine (function) that gets called automatically when a certain condition is met and when it's done , the code resumes from the line it was last on in the main code.

(it will interrupt the main code when a condition is met)

Interrupt name :

~
~
~
~
~
~

retfie

→ "return from interrupt"

SFRs used :-

RCON
INTCON
INTCON2
INTCON3
PIR1 , PIR2
PIE1, PIE2
IPR1, IPR2

Interrupt → External → RB$_0$: RB2 → rising edge
RB$_0$: RB2 → falling edge
External → on change RB4: RB7

Internal → Timer, ccp , A/D ,
serial communication ....

The signal / condition that will trigger the interrupt to be called can be external or internal

in external interrupt, each bit is independent from bits 0 to 3 . whereas the onchange takes 4 bits (4 : 7) as one data , so if it was 0010 and became 0000 it will trigger the interrupt for example.

for example port B < 4: 7 >

$$\underbrace{0110}_{<4:7>}\ 0000$$

becomes $\underline{0010}$ 0000
changed → so there is an interrupt

* If there is at least one bit that changes from bits <4:7> an interrupt will occur.

For each interrupt has:

1. Enable bit	IE	
2. Flag bit	IF	→ Automatically set, manually reset
3. priority bit	Ip	

* The enable bit is to tell the microcontroller which interrupt registors we want to use

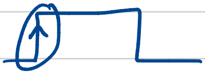
* if a certain condition is met, the microcontroller automatically sets the flag bit and we have to manually reset it (clear) before we return from the interrupt because if we didn't and we returned from the interrupt, the microcontroller will see that the flag bit is still set so it will enter the interrupt again.

→ To enable the priority feature : IPEN bit (RCON<7>)
↳ in general

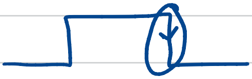
1: 2 levels of priority

Default → 0: 1 level of priority

rising edge



falling edge



INTEDG 0, 1, 2 ← in INTCON2

← Edge bits for port B0 : B2

0 → falling

1 → rising

→ Hardware decides the falling or rising

case 1 : IPEN (set)

- GIEH bit (INTCON<7>) enable all Int. high priority
- GIEL bit (INTCON<6>) " " " Low "

case 2 : IPEN (clear)

- GIE bit (INTCON<7>) enable all interrupt source
- PEIE bit (INTCON<6>) enable peripherals interrupt

* NOTE : There is no bit for priority selection of external INT0. Its always at high interrupt level and can't be changed to low

FIGURE 4-1: PROGRAM MEMORY MAP AND STACK FOR PIC18F442/242

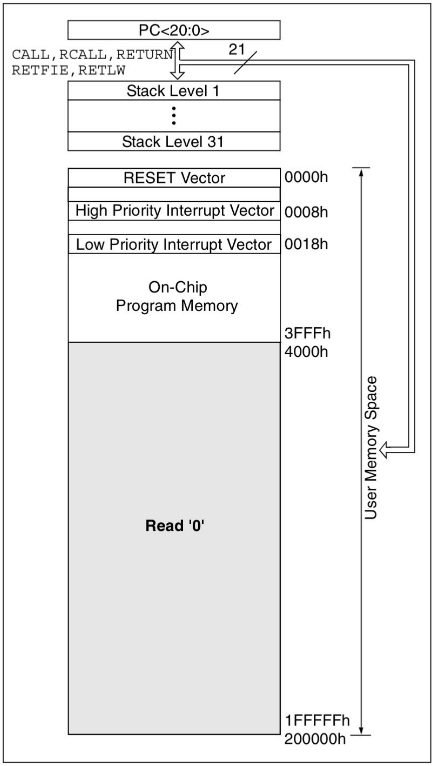
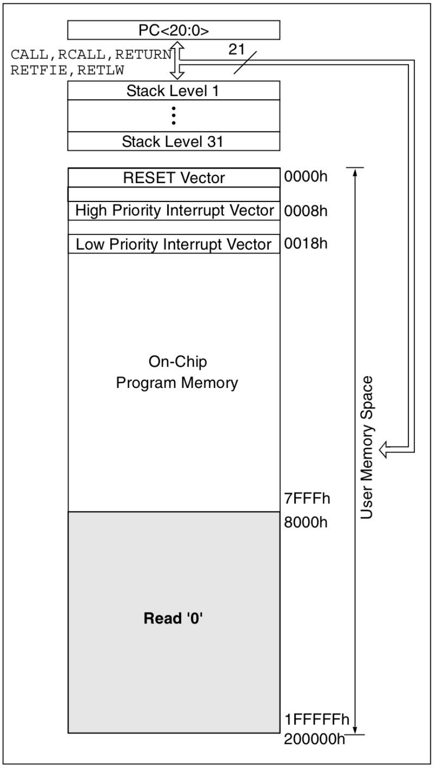


FIGURE 4-2: PROGRAM MEMORY MAP AND STACK FOR PIC18F452/252



8.0 INTERRUPTS

The PIC18FXX2 devices have multiple interrupt sources and an interrupt priority feature that allows each interrupt source to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 000008h and the low priority interrupt vector is at 000018h. High priority interrupt events will override any low priority interrupts that may be in progress.

There are ten registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/ compiler to automatically take care of the placement of these bits within the specified register.

Each interrupt source, except INT0, has three bits to control its operation. The functions of these bits are:

- Flag bit to indicate that an interrupt event occurred
- Enable bit that allows program execution to branch to the interrupt vector address when the flag bit is set
- Priority bit to select high priority or low priority

The interrupt priority feature is enabled by setting the IPEN bit (RCON<7>). When interrupt priority is enabled, there are two bits which enable interrupts globally. Setting the GIEH bit (INTCON<7>) enables all interrupts that have the priority bit set. Setting the GIEL bit (INTCON<6>) enables all interrupts that have the priority bit cleared. When the interrupt flag, enable bit and appropriate global interrupt enable bit are set, the interrupt will vector immediately to address 000008h or 000018h, depending on the priority level. Individual interrupts can be disabled through their corresponding enable bits.

When the IPEN bit is cleared (default state), the interrupt priority feature is disabled and interrupts are compatible with PICmicro® mid-range devices. In Compatibility mode, the interrupt priority bits for each source have no effect. INTCON<6> is the PEIE bit, which enables/disables all peripheral interrupt sources. INTCON<7> is the GIE bit, which enables/disables all interrupt sources. All interrupts branch to address 000008h in Compatibility mode.

When an interrupt is responded to, the Global Interrupt Enable bit is cleared to disable further interrupts. If the IPEN bit is cleared, this is the GIE bit. If interrupt priority levels are used, this will be either the GIEH or GIEL bit. High priority interrupt sources can interrupt a low priority interrupt.

The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (000008h or 000018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

The “return from interrupt” instruction, `RETFIE`, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB input change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bit or the GIE bit.

Note:	Do not use the `MOVFF` instruction to modify any of the Interrupt control registers while **any** interrupt is enabled. Doing so may cause erratic microcontroller behavior.

8.1 INTCON Registers

The INTCON Registers are readable and writable registers, which contain various enable, priority and flag bits.

> **Note:** Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

REGISTER 8-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

flag bit initially 0, if the condition is met, Automatically set, but before going bak to the main

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Automatically set, manually reset

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

REGISTER 8-2: INTCON2 REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP

bit 7 bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
`1` = All PORTB pull-ups are disabled
`0` = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0**:External Interrupt0 Edge Select bit
`1` = Interrupt on rising edge
`0` = Interrupt on falling edge

bit 5 **INTEDG1**: External Interrupt1 Edge Select bit
`1` = Interrupt on rising edge
`0` = Interrupt on falling edge

bit 4 **INTEDG2**: External Interrupt2 Edge Select bit
`1` = Interrupt on rising edge
`0` = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP**: TMR0 Overflow Interrupt Priority bit
`1` = High priority
`0` = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP**: RB Port Change Interrupt Priority bit
`1` = High priority
`0` = Low priority

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

REGISTER 8-3: INTCON3 REGISTER

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

8.2 PIR Registers

The PIR registers contain the individual flag bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Flag Registers (PIR1, PIR2).

Note 1: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

2: User software should ensure the appropriate interrupt flag bits are cleared prior to enabling an interrupt, and after servicing that interrupt.

REGISTER 8-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF(1):** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer, RCREG, is full (cleared when RCREG is read)
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit (see Section 16.0 for details on TXIF functionality)
1 = The USART transmit buffer, TXREG, is empty (cleared when TXREG is written)
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Master Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared in software)
0 = Waiting to transmit/receive

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 **TMR2IF:** TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = MR1 register did not overflow

Note 1: This bit is reserved on PIC18F2X2 devices; always maintain this bit clear.

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

REGISTER 8-5: PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	CCP2IF
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIF**: Data EEPROM/FLASH Write Operation Interrupt Flag bit
1 = The Write operation is complete (must be cleared in software)
0 = The Write operation is not complete, or has not been started

bit 3 **BCLIF**: Bus Collision Interrupt Flag bit
1 = A bus collision occurred (must be cleared in software)
0 = No bus collision occurred

bit 2 **LVDIF**: Low Voltage Detect Interrupt Flag bit
1 = A low voltage condition occurred (must be cleared in software)
0 = The device voltage is above the Low Voltage Detect trip point

bit 1 **TMR3IF**: TMR3 Overflow Interrupt Flag bit
1 = TMR3 register overflowed (must be cleared in software)
0 = TMR3 register did not overflow

bit 0 **CCP2IF**: CCPx Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

8.3 PIE Registers

The PIE registers contain the individual enable bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Enable Registers (PIE1, PIE2). When IPEN = 0, the PEIE bit must be set to enable any of these peripheral interrupts.

REGISTER 8-6: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE[1]:** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Master Synchronous Serial Port Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

To compare the value of PR2 and TMR2

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: This bit is reserved on PIC18F2X2 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

REGISTER 8-7: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	CCP2IE
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIE**: Data EEPROM/FLASH Write Operation Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 3 **BCLIE**: Bus Collision Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 2 **LVDIE**: Low Voltage Detect Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 1 **TMR3IE**: TMR3 Overflow Interrupt Enable bit
1 = Enables the TMR3 overflow interrupt
0 = Disables the TMR3 overflow interrupt

bit 0 **CCP2IE**: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

8.4 IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Priority Registers (IPR1, IPR2). The operation of the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

REGISTER 8-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PSPIP[1]	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
bit 7							bit 0

bit 7 **PSPIP[1]:** Parallel Slave Port Read/Write Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **ADIP**: A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP**: USART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP**: USART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP**: Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP**: CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP**: TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP**: TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: This bit is reserved on PIC18F2X2 devices; always maintain this bit set.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

REGISTER 8-9: IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER 2

U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	CCP2IP
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIP**: Data EEPROM/FLASH Write Operation Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **BCLIP**: Bus Collision Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **LVDIP**: Low Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR3IP**: TMR3 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **CCP2IP**: CCP2 Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

8.5 RCON Register

The RCON register contains the bit which is used to enable prioritized interrupts (IPEN).

REGISTER 8-10: RCON REGISTER

R/W-0	U-0	U-0	R/W-1	R-1	R-1	R/W-0	R/W-0
IPEN	—	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

we only need bit 7

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (16CXXX Compatibility mode)

1: The mc will look for which interrupt has a High priority and which has a low one

0: mc will not look for priorities and will generate interrupts at the same time with no bias.

bit 6-5 **Unimplemented:** Read as '0'

bit 4 **$\overline{\text{RI}}$:** RESET Instruction Flag bit
For details of bit operation, see Register 4-3

bit 3 **$\overline{\text{TO}}$:** Watchdog Time-out Flag bit
For details of bit operation, see Register 4-3

bit 2 **$\overline{\text{PD}}$:** Power-down Detection Flag bit
For details of bit operation, see Register 4-3

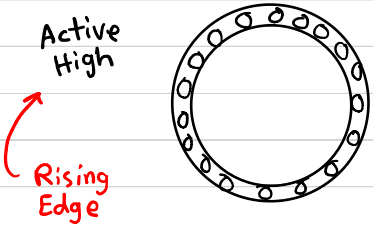
bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit
For details of bit operation, see Register 4-3

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
For details of bit operation, see Register 4-3

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

Example write Interrupt program to measure motor speed.

optical encoder $\boxed{1000 \text{ pulse / rev}} \rightarrow$ resolution



Every $\overset{T_s}{\underline{1 ms}}$ the program will find the speed

$F_{osc} = 4 \text{ MHz}$

number of revolutions

$$\text{Motor speed} = \frac{\text{Pulse count}}{\text{resolution}} * \frac{1}{T_s}$$

$$= \frac{\text{pulse count}}{1000} \times \frac{1}{1 \times 10^{-3}} \quad \text{rev/s}$$

$$\therefore \text{Motor speed} = \text{pulse count}$$

→ we want to count the number of pulses by counting each rising edge.

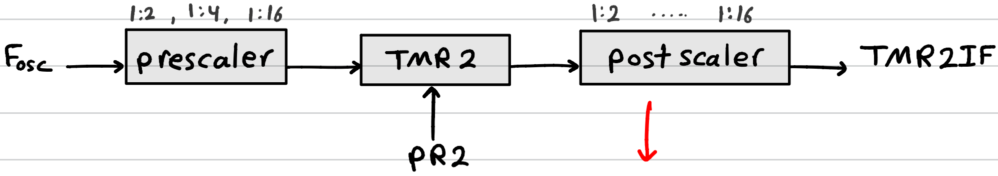
→ Each time period we want to see how many pulses we counted.

∴ we need 2 Interrupts

→ External ; to count the pulses. RB1 → to count rising edges

(since an optical sensor is not inside the micro controller itself).

→ Internal ; to access the timer. TMR2



we don't use it in the pwm but we do use it in the Interrupt.

$$T = [PR2+1] * 4 * T_{osc} * \text{prescale} * \text{postscale}$$

☆ Note: If want to use the pwm as well as the Interrupt first we need to find PR2 and initialize the pwm and keep the prescale as is without adding the post scale yet. when we start initializing the Interrupt we find the new PR2 by taking the same prescale and adding the postscale.

$$1\times10^{-3} = [PR2+1] * 1\times10^{-6} * \overset{1}{\text{prescale}} * \overset{10}{\text{postscale}}$$

$$[PR2+1] = 100$$

$$\therefore PR2 = 99$$

we chose 10 but we can also use 5 or 8

					ON	prescale	
X	1	0	0	1	0	0	0
	postscale						

T2CON = 48h

1:1 → 00

postscale

1:1 0000 → 1:16 1111

∴ 1:10 = 1001

Initialization:-

```
    BSF      INTCON3 , INT1IE ;
    BSF      PIE1 , TMR2IE     *
    BCF      RCON , IPEN       *
    BSF      INTCON2, INEDG1
   MOVLW     99
   MOVWF     PR2
   MOVLW     0x48
   MOVWF     T2CON
   BSF       INCON , PEIE    *
   BSF       INCON, GIE      *
   BSF       T2CON, 2       *

return
```

→ Because we chose External and port B1 we had to look for the bit that enables the interrupt for that bit in the data sheet and found that it's in INTCON3.

* we don't need priority so we clear that bit, it's clear by default but just to make sure.

→ we're assuming radix to be decimal which means if i put a number like 99 or 48 the program will consider it to be decimal so if we want Hexadecimal just put 0x48.

* Note that these file registers have named bits, so we can access them using the bit number or the name of the bit.

* To start the timer. (TMR2)

* To Activate timer and port B (enable).

* To Enable TMR2

```
    Int-Handler :

              BTFSC      INTCON3 , INT1IF
              goto       pulse count
              MOVFF      count, speed      ; count = No. of pulses
              CLRF       count
              BCF        PIR1 , TMR2IF        *
              goto       LL
pulse count   INCF       count , 1
              BCF        INTCON3 , INT1IF     *

      LL   RETFIE
```

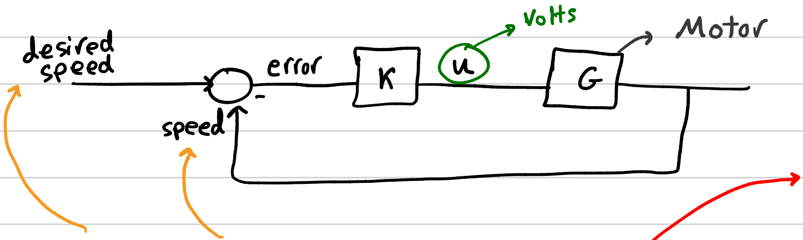
* The optical encoder's output is pulses and we found a relation based on the resolution to find the speed, in this example it's the same as the pulse count.

* There are 2 things that will cause an interrupt, external and the timer

* we need to clear the flag bit of each interrupt, otherwise the microcontroller will go back into the interrupt because it saw that the flag bit is set. (flag bit set = trigger an interrupt).

* Note ⟶ consider that pwm is already initialized

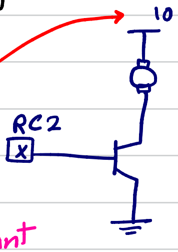
→ we will create a propotional controller to control the speed.



error = desired — actual

$U = V_{eff} = \frac{T_{ON}}{T} * V_s$

constant



* Note that $T = 10\ \mu s$

$T_{on} = [CCPR1L : CCP1CON<5,4>] \times \frac{1}{T_{osc}} \times \underset{1}{\text{prescale}}$

$T_{on} = \underbrace{\text{error} \times 4}_{} \times \frac{1}{4}\ \mu s \times 1$

Because we saved the error only in ccpR1L so it's like we did shift left twice

$T_{on} = \text{error}\ \mu s$

$$u = \frac{\text{error}\ \mu s}{10\ \mu s} \times 10$$

$u = \text{error}$ $\therefore K = 1$

```
Main :
     CALL      Initialization
     MOVLW     100          ; The desired speed we want (decimal)
     MOVWF     desired - speed
     MOVF      speed , w
     subwf     desired-speed , w  ; w= desired-speed - w
     MOVWF     CCPR1L
```

PID for Continuous Time

$$PID = K_p \, e + K_i \int e \, dt + K_d \frac{de}{dt}$$

Note that e is error

PID for Discrete Time

$$PID = K_p \, e_k + K_i \sum_{i=1}^{k} e_k \cdot T_s + K_d \frac{e_k - e_{k-1}}{T_s}$$

Note that k is sampling instant

Note that T_s is Sampling time

$T_s = 1ms$

$$GPR \begin{cases} \text{error i} \Rightarrow e_1 + e_2 + e_3 + e_4 \cdots + e_k \\ \text{error} \Rightarrow e_k \\ \text{error d} \Rightarrow e_{k-1} \end{cases}$$

errord = error - errorp

```
Main :
      CALL      Initialization
L1    MOVLW     100              ; The desired speed we want (decimal)
      MOVWF     desired - speed
      movF      speed , w
      subwf     desired-speed , w   ; w= desired-speed - w
      Movwf     error
      Addwf     errori , 1
      Movf      errorp , w                          previous error
                                                   ↗
      subwf     error , w        ;  w = eK - eK-1
                                              d
                                    p    ┌────┴────┐
      Movwf     errord
      addwf     error , w        ⇒ w = eK + eK - eK-1
      addwf     errori , w       ⇒ w = p + d + I
      movwf     ccpR1L
      MOVFF     error , error p

      goto      L1
```

Serial communication

USART ≡ Universal Synchronous Asynchronous Reciver Transmitter

1. Synchronous ⇒ Information + clock
2. Asynchronous ⇒ Just information

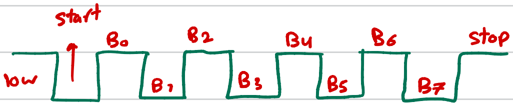
* Parallel and serial information

1. parallel: Information appear <u>at once</u>

Ex
```
CLRF   TRISB
MOVLW  0X08
MOVWF  portB
```

B7 ... B0

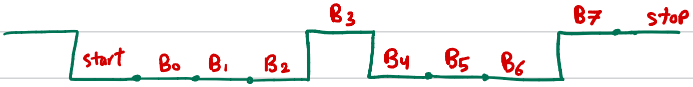
DATA 0 1 0 1 0 1 0 1



* we start from LSB to MSB

start → 8 bit → stop bit

DATA 1000 1000



2. series: Information Bit by Bit.

used SFRs :

TXSTA RCSTA SPBRG TxREG

used pins :

RC6 /TX → Transmitter (output)

RC7 / RC → Reciever (input)

parity Bit :
- odd parity → 1 if No. of ones is odd
- in DATA
- even parity → 0 if No. of ones is even

* The goal of the parity Bit is to make sure that the Reciever recieved the information correctly

Example

$X = \underline{1}0\underline{1}0\underline{1}000$ parity Bit = 1 (odd)

$X = \underline{1}0\underline{1}0\underline{1}0\underline{1}0$ parity Bit = 0 (even)

Baud Rate Generator

- Asynchronous low speed (BRGH = 0)

$$\text{Baud Rate} = \frac{f_{osc}}{64\,(x+1)}$$

- Asynchronous high speed (BRGH = 1)

$$\text{Baud Rate} = \frac{f_{osc}}{16\,(x+1)}$$

```
movlw     x
movwf     SPBRG
BCF       TXSTA, Sync
BSF       RCSTA, SPEN
BSF       TXSTA, TX9
BCF       TXSTA, 0        → The 9th data is zero
movlw     y
movwf     TXREG
BSF       TXSTA, TXEN
```

16.0 ADDRESSABLE UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI.) The USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices, such as CRT terminals and personal computers, or it can be configured as a half-duplex synchronous system that can communicate with peripheral devices, such as A/D or D/A integrated circuits, serial EEPROMs, etc.

The USART can be configured in the following modes:

- Asynchronous (full-duplex)
- Synchronous - Master (half-duplex)
- Synchronous - Slave (half-duplex)

In order to configure pins RC6/TX/CK and RC7/RX/DT as the Universal Synchronous Asynchronous Receiver Transmitter:

- bit SPEN (RCSTA<7>) must be set (= 1),
- bit TRISC<6> must be cleared (= 0), and
- bit TRISC<7> must be set (=1).

Register 16-1 shows the Transmit Status and Control Register (TXSTA) and Register 16-2 shows the Receive Status and Control Register (RCSTA).

REGISTER 16-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

Transmitter

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
x			0	0			

bit 7 bit 0

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode: → we will only use Asynchronous

Don't care

Synchronous mode:

1 = Master mode (clock generated internally from BRG)

0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit → we choose if we want 9 or 8 bits

1 = Selects 9-bit transmission

0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit

1 = Transmit enabled

0 = Transmit disabled

Note: SREN/CREN overrides TXEN in SYNC mode.

bit 4 **SYNC**: USART Mode Select bit

1 = Synchronous mode

0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:

1 = High speed

0 = Low speed

Synchronous mode:

Unused in this mode

bit 1 **TRMT**: Transmit Shift Register Status bit → Automatic

1 = TSR empty

0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data → if 9 Bit DATA is selected this is where we put the 9th bit

Can be Address/Data bit or a parity bit.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 16-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER

Receiver

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit

Asynchronous mode:
Don't care

Synchronous mode - Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode - Slave:
Don't care

bit 4 **CREN**: Continuous Receive Enable bit

Asynchronous mode:
1 = Enables receiver
0 = Disables receiver

Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN**: Address Detect Enable bit

Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enable interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data
This can be Address/Data bit or a parity bit, and must be calculated by user firmware.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

16.1 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode, bit BRGH is ignored. Table 16-1 shows the formula for computation of the baud rate for different USART modes, which only apply in Master mode (internal clock).

Given the desired baud rate and Fosc, the nearest integer value for the SPBRG register can be calculated using the formula in Table 16-1. From this, the error in baud rate can be determined.

Example 16-1 shows the calculation of the baud rate error for the following conditions:

- FOSC = 16 MHz
- Desired Baud Rate = 9600
- BRGH = 0
- SYNC = 0

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the FOSC/(16(X + 1)) equation can reduce the baud rate error in some cases.

Writing a new value to the SPBRG register causes the BRG timer to be reset (or cleared). This ensures the BRG does not wait for a timer overflow before outputting the new baud rate.

16.1.1 SAMPLING

The data on the RC7/RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin.

EXAMPLE 16-1: CALCULATING BAUD RATE ERROR

Desired Baud Rate = FOSC / (64 (X + 1))

Solving for X:

X = ((FOSC / Desired Baud Rate) / 64) – 1
X = ((16000000 / 9600) / 64) – 1
X = [25.042] = 25

Calculated Baud Rate = 16000000 / (64 (25 + 1))
= 9615

Error = $\frac{\text{(Calculated Baud Rate – Desired Baud Rate)}}{\text{Desired Baud Rate}}$
= (9615 – 9600) / 9600
= 0.16%

TABLE 16-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0 1	(Asynchronous) Baud Rate = FOSC/(64(X+1)) (Synchronous) Baud Rate = FOSC/(4(X+1))	Baud Rate = FOSC/(16(X+1)) N/A

Legend: X = value in SPBRG (0 to 255)

TABLE 16-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used by the BRG.

TABLE 16-3: BAUD RATES FOR SYNCHRONOUS MODE

BAUD RATE (Kbps)	FOSC = 40 MHz			33 MHz			25 MHz			20 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	NA	-	-	NA	-	-	NA	-	-
19.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
76.8	76.92	+0.16	129	77.10	+0.39	106	77.16	+0.47	80	76.92	+0.16	64
96	96.15	+0.16	103	95.93	-0.07	85	96.15	+0.16	64	96.15	+0.16	51
300	303.03	+1.01	32	294.64	-1.79	27	297.62	-0.79	20	294.12	-1.96	16
500	500	0	19	485.30	-2.94	16	480.77	-3.85	12	500	0	9
HIGH	10000	-	0	8250	-	0	6250	-	0	5000	-	0
LOW	39.06	-	255	32.23	-	255	24.41	-	255	19.53	-	255

BAUD RATE (Kbps)	FOSC = 16 MHz			10 MHz			7.15909 MHz			5.0688 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	NA	-	-	9.62	+0.23	185	9.60	0	131
19.2	19.23	+0.16	207	19.23	+0.16	129	19.24	+0.23	92	19.20	0	65
76.8	76.92	+0.16	51	75.76	-1.36	32	77.82	+1.32	22	74.54	-2.94	16
96	95.24	-0.79	41	96.15	+0.16	25	94.20	-1.88	18	97.48	+1.54	12
300	307.70	+2.56	12	312.50	+4.17	7	298.35	-0.57	5	316.80	+5.60	3
500	500	0	7	500	0	4	447.44	-10.51	3	422.40	-15.52	2
HIGH	4000	-	0	2500	-	0	1789.80	-	0	1267.20	-	0
LOW	15.63	-	255	9.77	-	255	6.99	-	255	4.95	-	255

BAUD RATE (Kbps)	FOSC = 4 MHz			3.579545 MHz			1 MHz			32.768 kHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	NA	-	-	NA	-	-	NA	-	-	0.30	+1.14	26
1.2	NA	-	-	NA	-	-	1.20	+0.16	207	1.17	-2.48	6
2.4	NA	-	-	NA	-	-	2.40	+0.16	103	2.73	+13.78	2
9.6	9.62	+0.16	103	9.62	+0.23	92	9.62	+0.16	25	8.20	-14.67	0
19.2	19.23	+0.16	51	19.04	-0.83	46	19.23	+0.16	12	NA	-	-
76.8	76.92	+0.16	12	74.57	-2.90	11	83.33	+8.51	2	NA	-	-
96	1000	+4.17	9	99.43	+3.57	8	83.33	-13.19	2	NA	-	-
300	333.33	+11.11	2	298.30	-0.57	2	250	-16.67	0	NA	-	-
500	500	0	1	447.44	-10.51	1	NA	-	-	NA	-	-
HIGH	1000	-	0	894.89	-	0	250	-	0	8.20	-	0
LOW	3.91	-	255	3.50	-	255	0.98	-	255	0.03	-	255

TABLE 16-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

Kilo byte per second

BAUD RATE (Kbps)	Fosc = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	2.40	-0.07	214	2.40	-0.15	162	2.40	+0.16	129
9.6	9.62	+0.16	64	9.55	-0.54	53	9.53	-0.76	40	9.47	-1.36	32
19.2	18.94	-1.36	32	19.10	-0.54	26	19.53	+1.73	19	19.53	+1.73	15
76.8	78.13	+1.73	7	73.66	-4.09	6	78.13	+1.73	4	78.13	+1.73	3
96	89.29	-6.99	6	103.13	+7.42	4	97.66	+1.73	3	104.17	+8.51	2
300	312.50	+4.17	1	257.81	-14.06	1	NA	-	-	312.50	+4.17	0
500	625	+25.00	0	NA	-	-	NA	-	-	NA	-	-
HIGH	625	-	0	515.63	-	0	390.63	-	0	312.50	-	0
LOW	2.44	-	255	2.01	-	255	1.53	-	255	1.22	-	255

BAUD RATE (Kbps)	Fosc = 16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)	7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	1.20	+0.16	207	1.20	+0.16	129	1.20	+0.23	92	1.20	0	65
2.4	2.40	+0.16	103	2.40	+0.16	64	2.38	-0.83	46	2.40	0	32
9.6	9.62	+0.16	25	9.77	+1.73	15	9.32	-2.90	11	9.90	+3.13	7
19.2	19.23	+0.16	12	19.53	+1.73	7	18.64	-2.90	5	19.80	+3.13	3
76.8	83.33	+8.51	2	78.13	+1.73	1	111.86	+45.65	0	79.20	+3.13	0
96	83.33	-13.19	2	78.13	-18.62	1	NA	-	-	NA	-	-
300	250	-16.67	0	156.25	-47.92	0	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	250	-	0	156.25	-	0	111.86	-	0	79.20	-	0
LOW	0.98	-	255	0.61	-	255	0.44	-	255	0.31	-	255

BAUD RATE (Kbps)	Fosc = 4 MHz		SPBRG value (decimal)	3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 kHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	0.30	-0.16	207	0.30	+0.23	185	0.30	+0.16	51	0.26	-14.67	1
1.2	1.20	+1.67	51	1.19	-0.83	46	1.20	+0.16	12	NA	-	-
2.4	2.40	+1.67	25	2.43	+1.32	22	2.23	-6.99	6	NA	-	-
9.6	8.93	-6.99	6	9.32	-2.90	5	7.81	-18.62	1	NA	-	-
19.2	20.83	+8.51	2	18.64	-2.90	2	15.63	-18.62	0	NA	-	-
76.8	62.50	-18.62	0	55.93	-27.17	0	NA	-	-	NA	-	-
96	NA	-	-	NA	-	-	NA	-	-	NA	-	-
300	NA	-	-	NA	-	-	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	62.50	-	0	55.93	-	0	15.63	-	0	0.51	-	0
LOW	0.24	-	255	0.22	-	255	0.06	-	255	0.002	-	255

TABLE 16-5: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

→ High

Actual Baud rate

Desired Baud rate

BAUD RATE (Kbps)	Fosc = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	9.60	-0.07	214	9.59	-0.15	162	9.62	+0.16	129
19.2	19.23	+0.16	129	19.28	+0.39	106	19.30	+0.47	80	19.23	+0.16	64
76.8	75.76	-1.36	32	76.39	-0.54	26	78.13	+1.73	19	78.13	+1.73	15
96	96.15	+0.16	25	98.21	+2.31	20	97.66	+1.73	15	96.15	+0.16	12
300	312.50	+4.17	7	294.64	-1.79	6	312.50	+4.17	4	312.50	+4.17	3
500	500	0	4	515.63	+3.13	3	520.83	+4.17	2	416.67	-16.67	2
HIGH	2500	-	0	2062.50	-	0	1562.50	-	0	1250	-	0
LOW	9.77	-	255	8,06	-	255	6.10	-	255	4.88	-	255

BAUD RATE (Kbps)	Fosc = 16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)	7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	2.41	+0.23	185	2.40	0	131
9.6	9.62	+0.16	103	9.62	+0.16	64	9.52	-0.83	46	9.60	0	32
19.2	19.23	+0.16	51	18.94	-1.36	32	19.45	+1.32	22	18.64	-2.94	16
76.8	76.92	+0.16	12	78.13	+1.73	7	74.57	-2.90	5	79.20	+3.13	3
96	100	+4.17	9	89.29	-6.99	6	89.49	-6.78	4	105.60	+10.00	2
300	333.33	+11.11	2	312.50	+4.17	1	447.44	+49.15	0	316.80	+5.60	0
500	500	0	1	625	+25.00	0	447.44	-10.51	0	NA	-	-
HIGH	1000	-	0	625	-	0	447.44	-	0	316.80	-	0
LOW	3.91	-	255	2.44	-	255	1.75	-	255	1.24	-	255

BAUD RATE (Kbps)	Fosc = 4 MHz		SPBRG value (decimal)	3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 kHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	0.30	+0.16	207	0.29	-2.48	6
1.2	1.20	+0.16	207	1.20	+0.23	185	1.20	+0.16	51	1.02	-14.67	1
2.4	2.40	+0.16	103	2.41	+0.23	92	2.40	+0.16	25	2.05	-14.67	0
9.6	9.62	+0.16	25	9.73	+1.32	22	8.93	-6.99	6	NA	-	-
19.2	19.23	+0.16	12	18.64	-2.90	11	20.83	+8.51	2	NA	-	-
76.8	NA	-	-	74.57	-2.90	2	62.50	-18.62	0	NA	-	-
96	NA	-	-	111.86	+16.52	1	NA	-	-	NA	-	-
300	NA	-	-	223.72	-25.43	0	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	250	-	0	55.93	-	0	62.50	-	0	2.05	-	0
LOW	0.98	-	255	0.22	-	255	0.24	-	255	0.008	-	255

16.2 USART Asynchronous Mode

In this mode, the USART uses standard non-return-to-zero (NRZ) format (one START bit, eight or nine data bits and one STOP bit). The most common data format is 8-bits. An on-chip dedicated 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. The USART transmits and receives the LSb first. The USART's transmitter and receiver are functionally independent, but use the same data format and baud rate. The baud rate generator produces a clock, either x16 or x64 of the bit shift rate, depending on bit BRGH (TXSTA<2>). Parity is not supported by the hardware, but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

Asynchronous mode is selected by clearing bit SYNC (TXSTA<4>).

The USART Asynchronous module consists of the following important elements:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

16.2.1 USART ASYNCHRONOUS TRANSMITTER

The USART transmitter block diagram is shown in Figure 16-1. The heart of the transmitter is the Transmit (serial) Shift Register (TSR). The shift register obtains its data from the read/write transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one TCY), the TXREG register is empty and flag bit TXIF (PIR1<4>) is set. This interrupt can be enabled/disabled by setting/clearing enable bit TXIE (PIE1<4>). Flag bit TXIF will be set, regardless of the state of enable bit TXIE and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While flag bit TXIF indicated the status of the TXREG register, another bit, TRMT (TXSTA<1>), shows the status of the TSR register. Status bit TRMT is a read-only bit, which is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty.

Note 1: The TSR register is not mapped in data memory, so it is not available to the user.

2: Flag bit TXIF is set when enable bit TXEN is set.

To set up an asynchronous transmission:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH (Section 16.1).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, set enable bit TXIE. → in PIE1
4. If 9-bit transmission is desired, set transmit bit TX9. Can be used as address/data bit.
5. Enable the transmission by setting bit TXEN, which will also set bit TXIF.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Load data to the TXREG register (starts transmission).

Note: TXIF is not cleared immediately upon loading data into the transmit buffer TXREG. The flag bit becomes valid in the second instruction cycle following the load instruction.

FIGURE 16-1: USART TRANSMIT BLOCK DIAGRAM

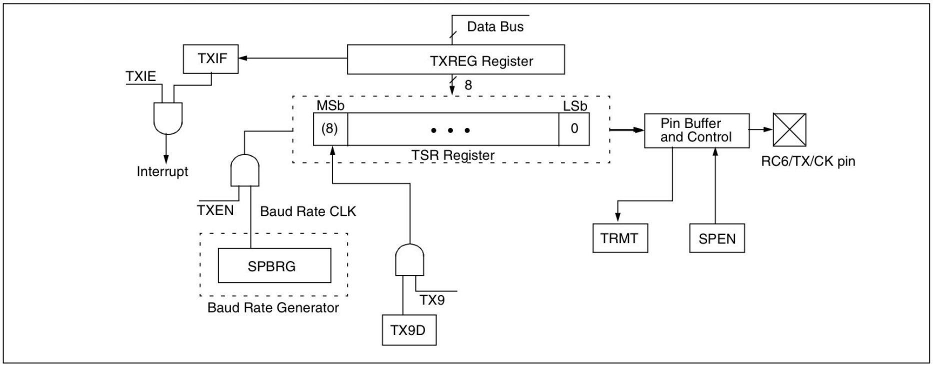


FIGURE 16-2: ASYNCHRONOUS TRANSMISSION

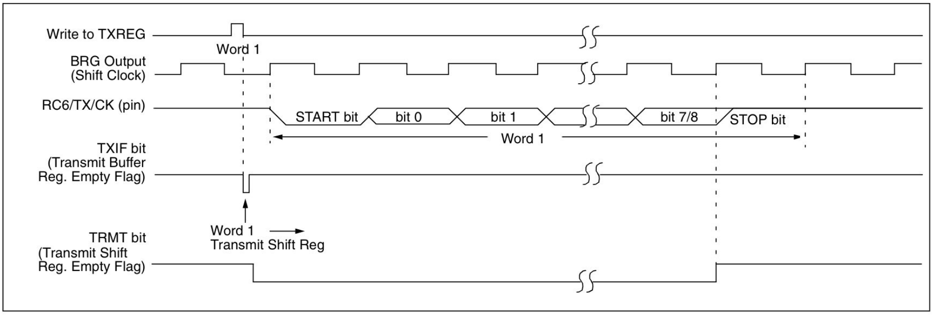
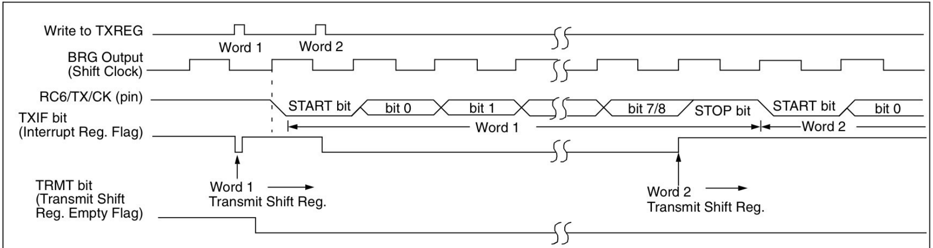


FIGURE 16-3: ASYNCHRONOUS TRANSMISSION (BACK TO BACK)



Note: This timing diagram shows two consecutive transmissions.

TABLE 16-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP[1]	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	0000 0000	0000 0000
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
TXREG	USART Transmit Register								0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'.
Shaded cells are not used for Asynchronous Transmission.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X2 devices; always maintain these bits clear.

16.2.2 USART ASYNCHRONOUS RECEIVER

The receiver block diagram is shown in Figure 16-4. The data is received on the RC7/RX/DT pin and drives the data recovery block. The data recovery block is actually a high speed shifter operating at x16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at FOSC. This mode would typically be used in RS-232 systems.

To set up an Asynchronous Reception:

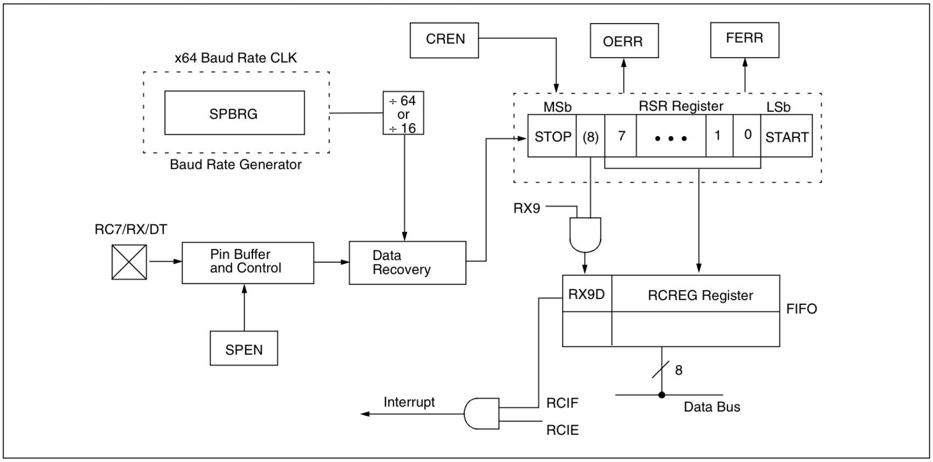
1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH (Section 16.1).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, set enable bit RCIE.
4. If 9-bit reception is desired, set bit RX9.
5. Enable the reception by setting bit CREN.
6. Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE was set.
7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
8. Read the 8-bit received data by reading the RCREG register.
9. If any error occurred, clear the error by clearing enable bit CREN.
10. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

16.2.3 SETTING UP 9-BIT MODE WITH ADDRESS DETECT

This mode would typically be used in RS-485 systems. To set up an Asynchronous Reception with Address Detect Enable:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is required, set the BRGH bit.
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are required, set the RCEN bit and select the desired priority level with the RCIP bit.
4. Set the RX9 bit to enable 9-bit reception.
5. Set the ADDEN bit to enable address detect.
6. Enable reception by setting the CREN bit.
7. The RCIF bit will be set when reception is complete. The interrupt will be acknowledged if the RCIE and GIE bits are set.
8. Read the RCSTA register to determine if any error occurred during reception, as well as read bit 9 of data (if applicable).
9. Read RCREG to determine if the device is being addressed.
10. If any error occurred, clear the CREN bit.
11. If the device has been addressed, clear the ADDEN bit to allow all received data into the receive buffer and interrupt the CPU.

FIGURE 16-4: USART RECEIVE BLOCK DIAGRAM



Questions :-

Q1: write a code that subtract the value of the working register from a file register called VALUE and returns the result to the file register VALUE.

```
 MOVLW    5H
 MOVWF   VALUE
MOVLW     10H ; put 10H in the WREG as an example
SUBWF   VALUE,F
```

Q2: write a code that divid the value of a file register called count by 2, and returns the result to the working register.

```
 MOVLW   E6H   ⟶    11100110
MOVWF   COUNT
RRCF    COUNT, w
```

Count will have value of $(01110011)_2$ which is 73 H (E6H ÷ 2 = 73H)

Q3: write a code that compares Count1 and Count2 file registers and returns the greater value to the WREG.

```
Main:
      MOVF     COUNT1 , w    ; WREG = COUNT1
      CPFSGT   COUNT2        ; skip if COUNT2 > COUNT1
      GOTO     LAST
      MOVF     COUNT2, w
LAST  end
```

skip if f > W

Q4: write a code that Multiply the value of file register called Reg1 and file register called Reg2 and store the results in file registers called RegH and RegL respectively.

```
MOVLW    50H
MOVWF    Reg1
MOVLW    A5H
MULLWF   Reg1
MOVFF    PRODH, RegH
MOVFF    PRODL, RegL
```

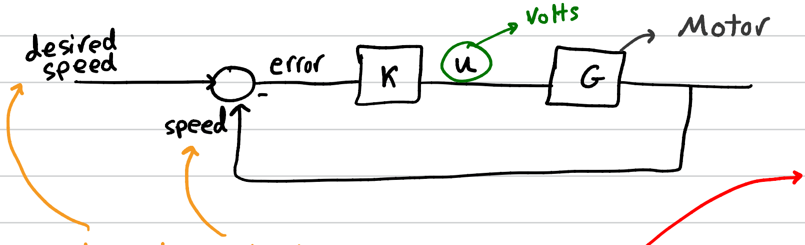
* For USART for Asynchronous mode of the BRGH = 1, what value should be stored in the SPFRG if the baud rate is 9600

$X = 214$ (from the tables)

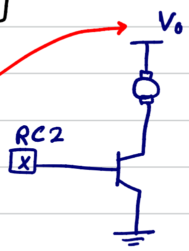
Q5: for A pressure sensor that is connected to AN7 as input where the output voltage of the sensor analog $V_o = 40\ mV/P$ (P is the pressure in psi)

A) if the pressure set point is 20 psi, writ a code that controls the motor speed such that the duty cycle will be equal to the error signal

$$V_o = \text{pressure} \times \text{sensitivity}$$
$$= 20\ psi \times 40\ mV/psi$$
$$= 2\ mV$$



error = desired — actual



$$U = V_{eff} = \frac{T_{ON}}{T} * V_0$$

$$\text{duty cycle} = \frac{T_{on}}{T} = \text{error}$$

كمل حل السؤال ←

B) write initialization code for the A/D module. Assume that vref is internal   Fosc = 4MHz

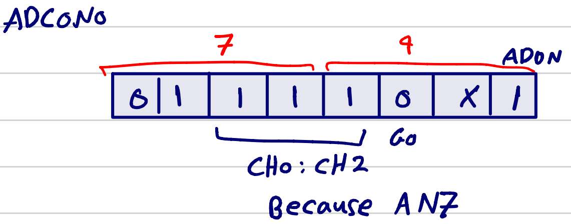
$T_{AD} \geqslant 1.6\ \mu s$

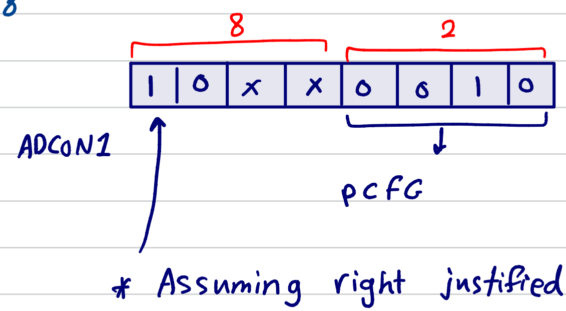
$x \cdot T_{osc} \geqslant 1.6\ \mu s$

$x \geqslant 1.6 \times 4$

$x \geqslant 6.4$

$\therefore\ x \rightarrow F_{osc}/8$





Initiailization :

```
Movlw   oX79
movwf   ADCONo
movlw   oX 82
movwf   ADCoN1

return
```

c) write initialization code such that pwm module will operate at 20 KHz assuming $F_{osc} = 4\,MHz$ & Duty cycle = 50%

↘ F_{pwm}

$$T_{pwm} = [PR2+1] * 4 * T_{osc} * prescale$$

$$\frac{1}{20} \times 10^{-3} = [PR2+1] * 4 * \frac{1}{4} * 10^{-6} * 1$$

PR2 = 49

Duty cycle = $\frac{T_{on}}{T}$

$$T_{ON} = [X] * T_{osc} * prescale$$

$$0.5 * \frac{1}{20} \times 10^{-3} = X * \frac{1}{4} \times 10^{-6} \times 1$$

$$X = 100 = \underbrace{\overset{=19h}{00\,0110\,01}}_{CCPR1L}\,\underbrace{00}_{CCP1CON<5:4>}$$

Initialization:

```
bcf    latc, 2
bcf    Trisc, 2
MovLw   49
movwf  PR2
movlw  0x19
movwf  CCPR1L
movlw  0x0C
movlw  CCP1CON
clRF   T2CON
BSF    T2CON, 2    ; to turn on the timer
return
```

TM2ON

T2CON								
	X	0	0	0	0	1	0	0

post scale = 0 at first

prescale = 1

and to turn on → bit 2 set

CCP1CON								
	X	X	0	0	1	1	X	X

= 0 0 c h

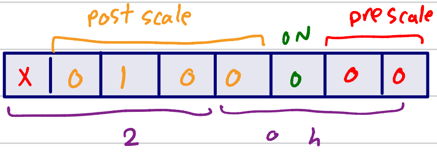
12

Q6: write initialization code TM2 interrupt to read the analog input every 500 μs and store the pressure value $F_{osc} = 4$ MHZ

$$T = [PR2 + 1] * 4 * T_{osc} * prescale * postscale$$

$$500 \times \cancel{10^{-6}} = [PR2 + 1] * \cancel{4} * \frac{1}{\cancel{4}} \times \cancel{10^{-6}} * 1 * 5$$

$$\therefore PR2 = 99$$



post scale = 1:5 = 0100

Initialization:

```
BSF      INTCON3 , INT1IE
BSF      PIE1 , TMR2IE
BCF      RCON , IPEN
BSF      INTCON2, INEDG1
MOVLW    99
MOVWF    PR2
MOVLW    0X 20
MOVWF    T2CON
BSF      INCON , PEIE
BSF      INCON , GIE
BSF      T2CON , 2
```

return

Q: write a code to solve the function
f(w) = 5 + 5w
↳ = x

```
movlw     x
mullw     5        ;    w = 5x
movwf     f(w)     ;    f(w) = 5x
movlw     5
addwf     f(w),1   ;    f(w) = 5 + 5x
```

Q: write a subroutine that read port B (B0 : B7) and store the second complement of port B in a file register called " second-complement ", don't use
(second complement = first complement + 1)

```
Comp:

    movlw     0xFF                         → ones = 1111 1111
    movwf     ones                                   F    F  h
    movf      portB, w   ; read contents of port B and save
    XORWF     ones, w      it in WREG
    movwf     first-comp
    movlw     0x01                         if port B  = 00101101
    Addwf     first-comp, w                   XOR       11111111
    movwf     second-complement            first-comp → 11010010
                                                               1 +
    return                                 second-comp → 11010011
```