Page: lecture ① Date:

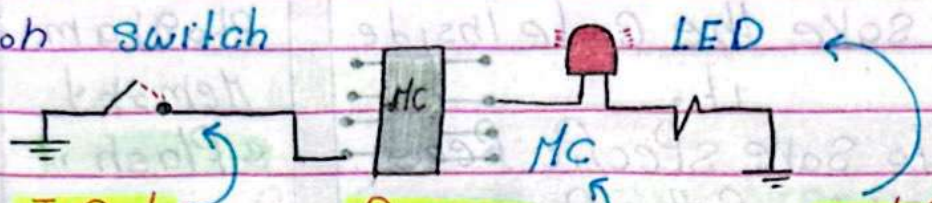
what's the Micro controller interfacing

This the shape of the MC is considered as ← PIC

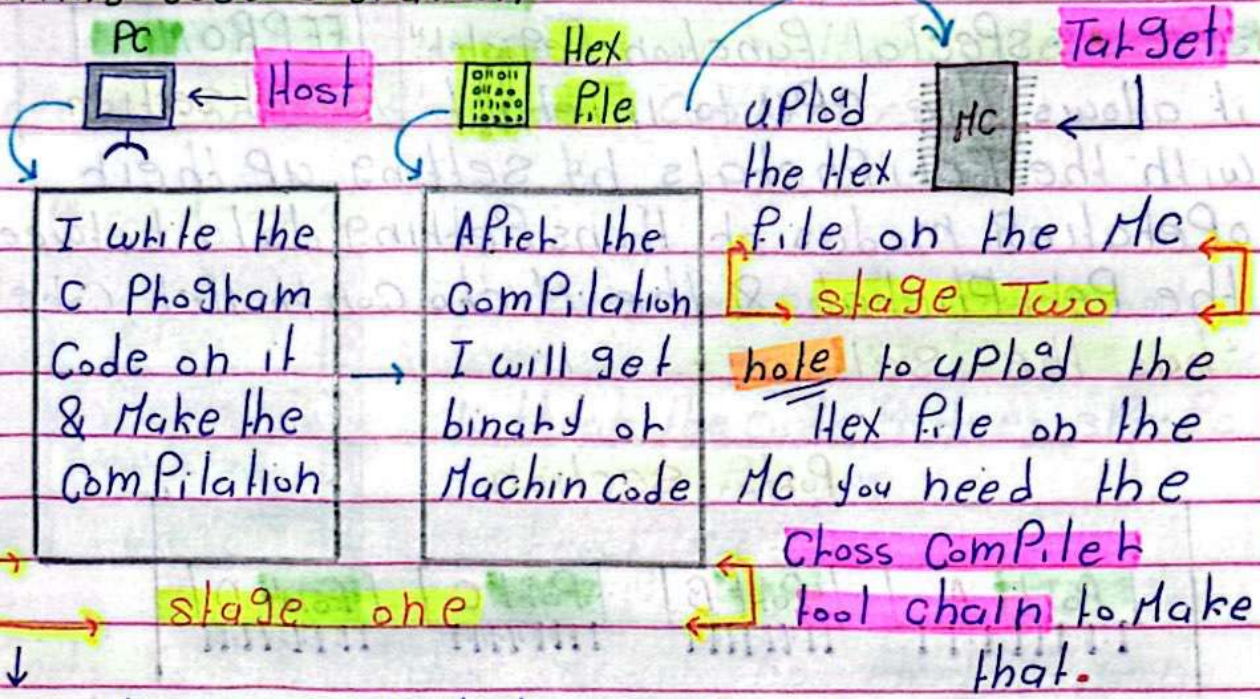
IC "integrated circuit" I can control with it

what's the interfacing

Connecting external devices with the MC & those devices may be are input action or output action switch



writting code operation



hole to upload the Hex File on the MC you need the Cross compiler tool chain to make that.

hole the compiler that is responsible for generating Hex File on the Host is called native compile & the compiler that takes the Hex File from the Host "PC" to the Target "MC" it's called the Cross compiler « Inv Q »

hole

we need a programer to flash the hex File on the Micro controller « Debugger »

Page: Date:

Micro Controller Internal block

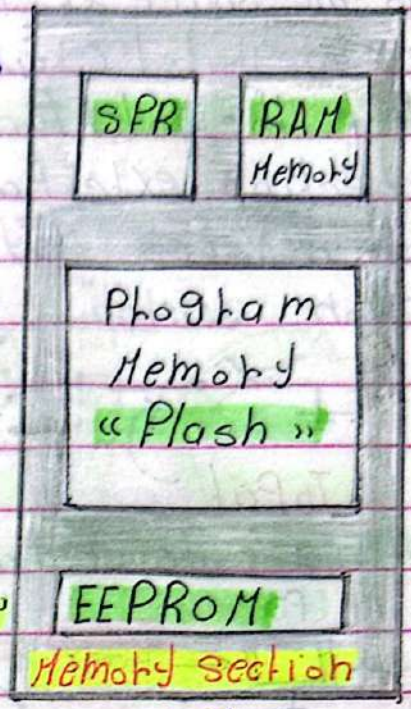
The first section at MC is the **Memory section** this section has all the Memories that are could be exsisted at the MC for ex:

RAM: to save the Variabiles

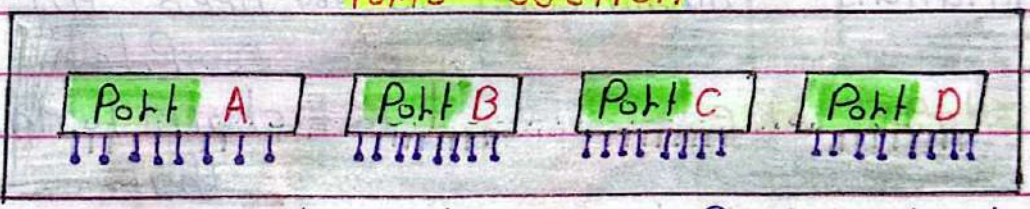
Flash: to save the Code inside it.

EEPROM: to save specific Read. Inside it even if the Powe is off.

SFR: "special Function Register" it allows the CPU to interact with the Peripherals by setting up their operating modes or transferring data between the Peripheral & the Micro Controller Core. "has the setting"



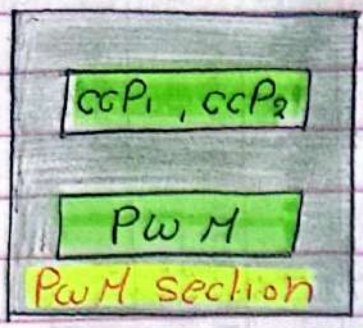
Ports section



* This section has the Pins of the MC devided into some Ports & each Port has number of Pins is equaled to other Ports.

ADC section Analog to digital Converter this section is responsible for converting the Analog signal to Digital

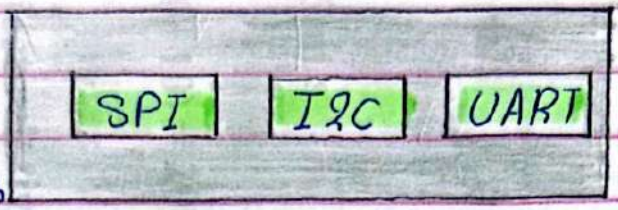
Page: Date:



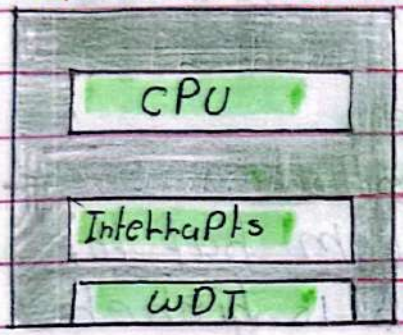
Puls width Modulation section I conthol with it With the signal on the Pin by Conthelling on the size of Puls width by this I Can Conthol in speed of Motor or lighting intensity of LED.

Communication Phot Col section

This section for the different Communication Phot Cols to Communicate with other external Devices or other Micho Conthollers



Processor section

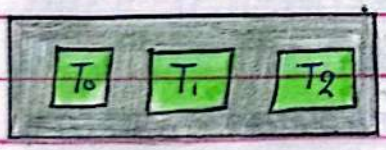


→ it's Consider as the Phocessor

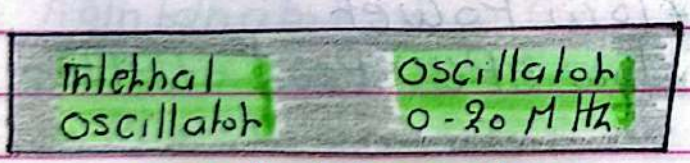
→ to bheak the Code & does specific function

→ to reset the Code to start

Timers section



This section has timers I Can use one or more of them to do specific functions each all those Timers.

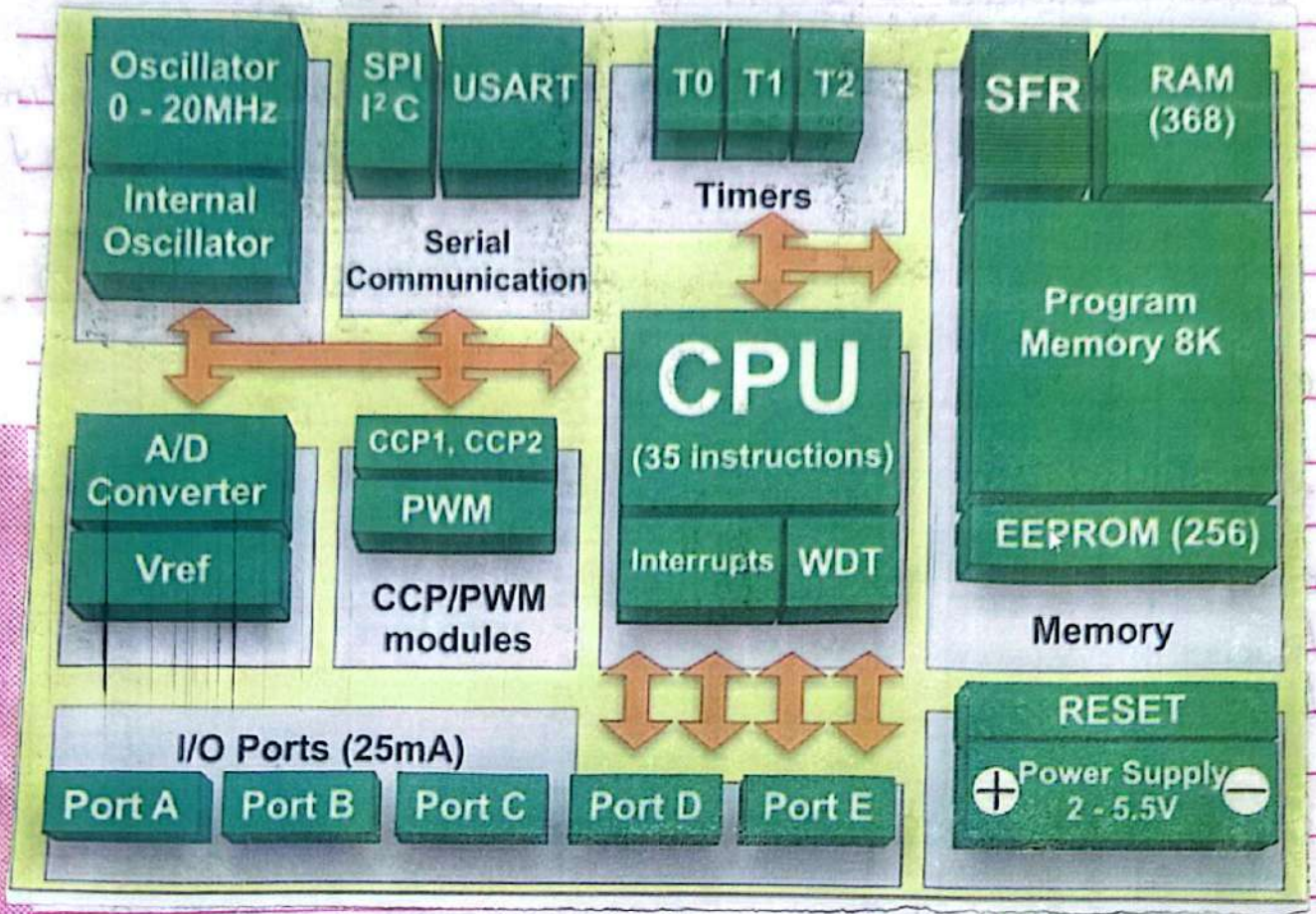


is hesponsible for gehenate the clock Pulses.

Oscillator section

Page:..... Date:.....

Micro Controller block



what's the different between the:
Micro Processor & Micro Controller
all of them are computing system but the techniques of the Micro Processor is higher than of the Micro Controller for ex:

PC

MC

*CPU = Processor	*CPU=Micro Processor
*General Tasks	*Specific Task
*high Power	*low Power
*high CPU speed 1GHz	*low CPU speed 16MHz

Page: Date:

the characteristics of ES Applications

* Single Function :
  execute a single Program repeatedly
* Tightly - Constrained : " هفيزة باه كلام "
  low Cost - low Power - fast - small
* Reactive & real - time :
  what is the time that the task will take it to be executed.
  ex : the Airbag to work what is the time that it will take it.
* Uint Cost :
  the Cost of the system without NRE Cost
  NRE Cost : the salary for the engineers
  the Cost of Component to Make the system .... ect
* NRE Cost :
  the Missing Cost during the implemention of the system
* Flexibility :
  Publish new Versions of the old system & Add new features
* Time to Prototype : Time needed to Make a Version of the system or a Model to see How the system works but this Version is not the Main system
* Time to Market
* Maintainability : the ability to Modify the system after is already in the Market
* safty & security

Page: Date:

What's the CPU : Central Processing Unit

* it Responsible for executing the code
* it Consits of 3 main elements :
  - CU : Control uint .
  - ALU : Arithmetic & logic uint .
  - Register bank :
    - → Program Counter .
    - → Instruction Register .
    - → Address Register .
    - → Accumalator Register .

the explaining of 3 main elements

CU : Control uint

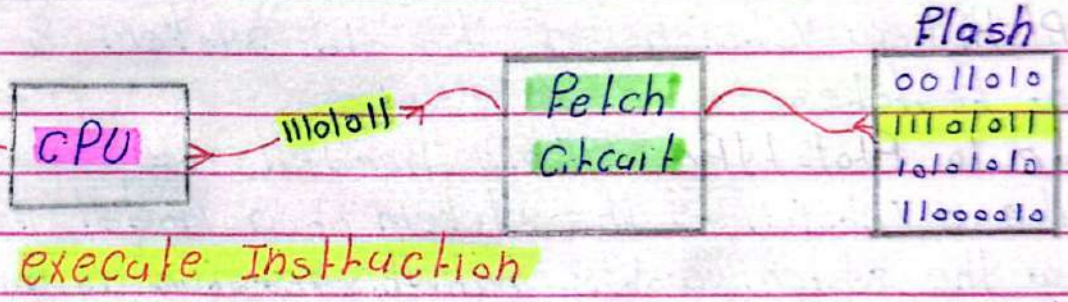
this unit has tw circuit are :

- → Fetch Circuit
- → Instruction Decoder circuit "ID"

CU

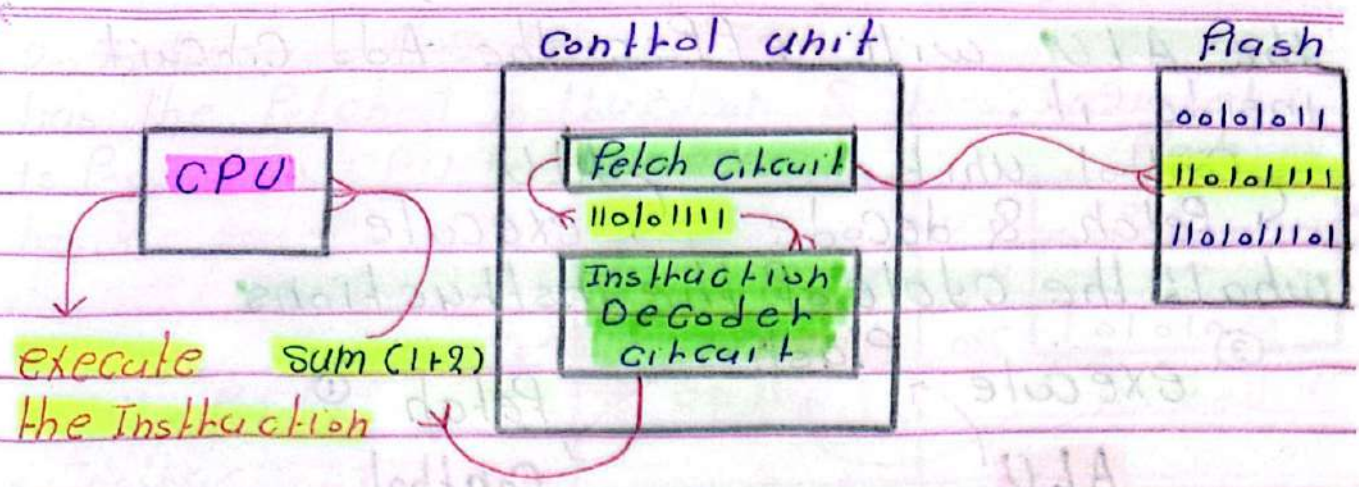
Fetch cir
ID cir

Fetch Circuit : Fetchs or takes the Instructions from the Flash Memory to execute those Instructions



Instruction Decoder Circuit : it translate the Instruction that is Consider as Zeros & ones into Meaning is under stood to the CPU after this Instruction Passed or Fetched by the Fetch Circuit in the Control Unit .

Page:..... Date:.....



How the Instruction Decoder got that this Instruction is sum (1+2) A3
By the Instruction set

① operand ② operand

011 001 010

Add 1 2

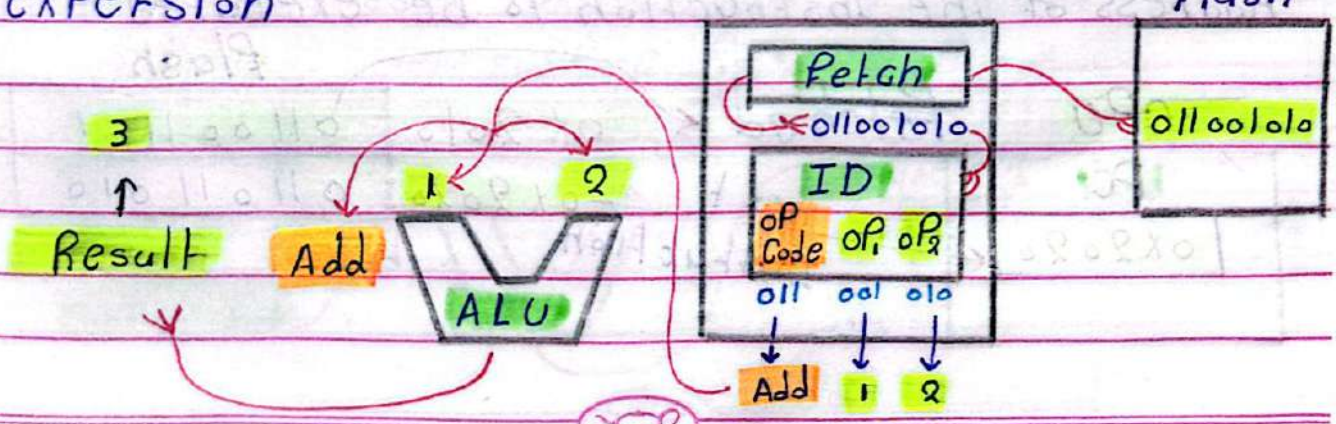
Instruction set

opCode	Instruction
011	Add
101	Sub
110	AND

The Meaning to the CPU to execute it

note the opCode : is a binary code
Cosiders as the expersion ex:. Add ..... ect

after the Instruction Decoder circuit Mode the expersion of the Instruction Pass this expersion to the ALU to execute this expersion

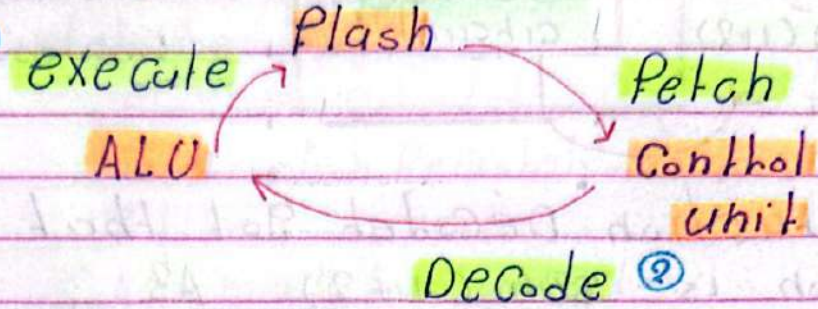


Page: Date:

the ALU will effec the Add circuit inside it.

Control unit → Fetch & decode
ALU → execute

what's the cycle of the Instructions

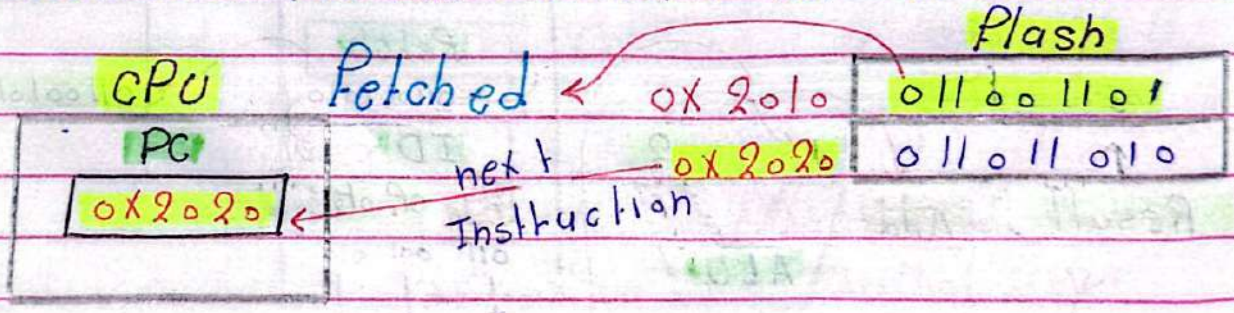


what's the register :
is a high speed Memory storing units
8 - 16 - 32 bits :. the size of the Register

note
any CPU has two types of registers :
1- General PurPose Register
2- special PurPose Register

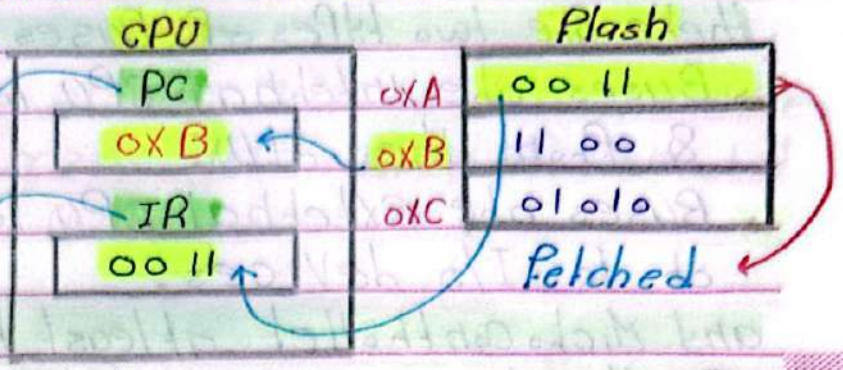
General PurPose Register :
has the data to operate on it

special PurPose Register :
is alocation inside the CPU for special uses
those registers are
1- Program Counter Register : "PC" : has the next Address of the Instruction to be executed



Page:..... Date:.....

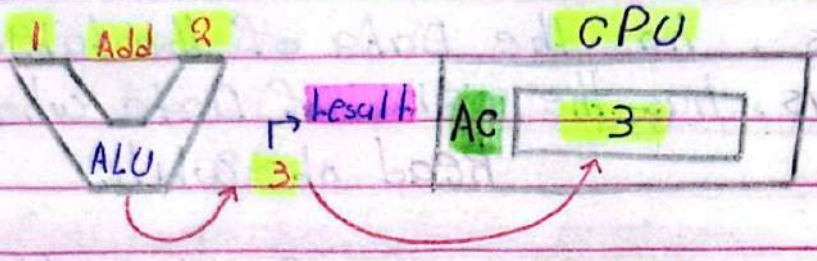
2- Instruction Register : "IR"
has the fetched Instruction & this register is found in CPU too



has the Address of the next Instruction

has the fetched Instruction

3- Accumulator Register "AC"
has the Result that is generated by the ALU

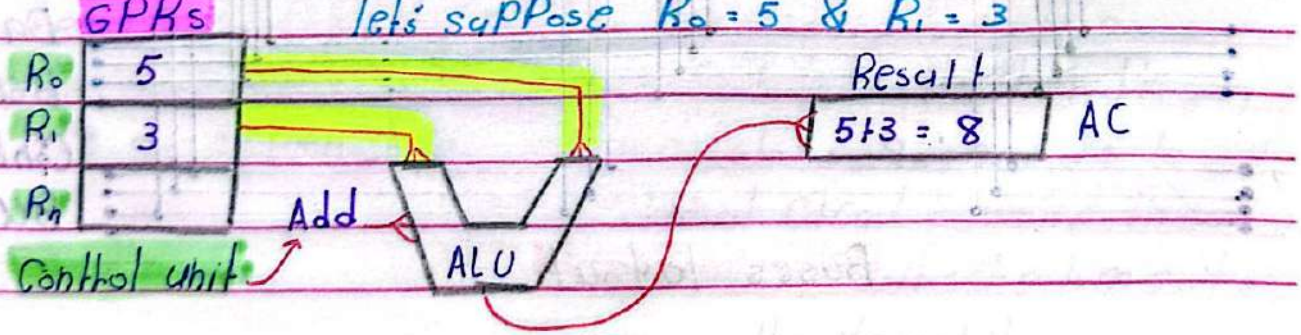


4- Processor status word "Psw"
hold the status of the last operation done by ALU

---

note: the register bank has : Special Register PC & IR & AC & Psw
& the register bank is inside the CPU.

General PurPose registers May be I store inside it the Data to be closed to the CPU to access it Quicly

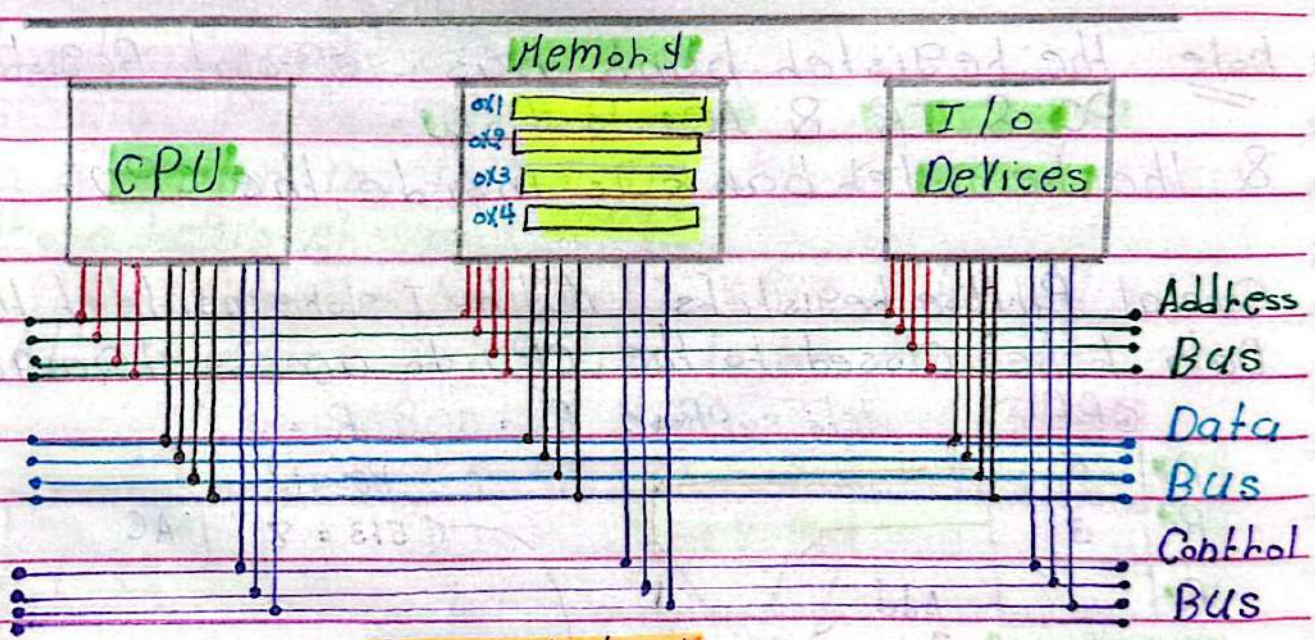


Page:..... Date:.....

The Bus is a Commah electrical Pathway between Multiple devices
there are two types of Buses inside the MC
→ Buses are internal CPU to transport data to & From the ALU.
→ Buses are external CPU to Connect it to Memory or to I/o devices.
any Micro Controller atleast has three main types of the buses :
→ Address bus. has the Address of the target location
→ Data bus . has the Data of the target location
→ Control bus. has the signal of Data what ever is Read or write

Note
= the number of wires for the Address & Data & Control buses are NOT equaled

"QIV"



Buses layout

Page: Date:

Note when the size of the Data bus increases = the ability to transfer the Data increases.

Note when the size of the Address bus increases = the ability to access large size of Memory increases.

Memory organization

You need to have a Memory inside your MC from different types Volatile & non Volatile

- **Volatile** : miss the data when the Power is off.
- **non Volatile** : doesn't miss the data when the Power is off.

Three types of storage needed for a Program :

- Code Memory "Flash or ROM".
- Data Memory "SRAM".
- Run time state of Program "Registers".

Note

= the size of buses inside the MC is changed from a MC to another.

Volatile Memory:

Very Fast but Can't hold data without Power

SRAM	DRAM
→ Faster than DRAM "4" times.	→ it's Made from the Capacitors rather than Transistors so needs to external circuit to Make the refreshing all the time to the Memory
→ needs More transistors So it's so expensive.	

Page: lecture ② Date:

How to read the Data sheet

PIC 18F 2k20 / 4x k20 → serial number
MP ↓ L→ 18 family

note
= the current of the Pin of MC Pic is 25 mA. input or output

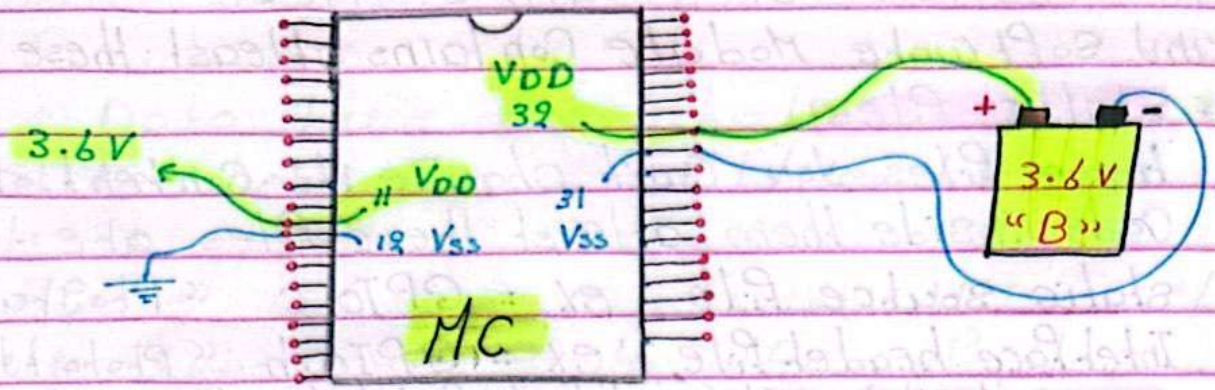
	Program Memory		Data Memory	
Device	Flash	single-word	SRAM	EEPROM
PIC18F46k20	(bytes)	Instruction	(bytes)	(bytes)
	64 k	32768	3936	1024
	↓ the size of the Flash to save the code	↓ After the code is considered as Instruction 32768 Instructions	↓ size of the RAM	↓ size of the EEPROM

I/O	A/D	CCP / ECCP PWM	MSSP SPI	MSSP I2C	UART	ComP	Timers 8/16 bit
36 Pins	14 channel	1/1 CCP/ECCP	yes	yes	1	2	1/3 Timers

36 Pins For I/O
14 Pins For A/D
one For CCP & one For ECCP
existed
has one UART Mod
has Two
has 4 Timers one is 8 bits & Three are 16 bits

Page:..... Date:.....

the Connecting Power to the Micro controller



layer based on Embedded Software Design

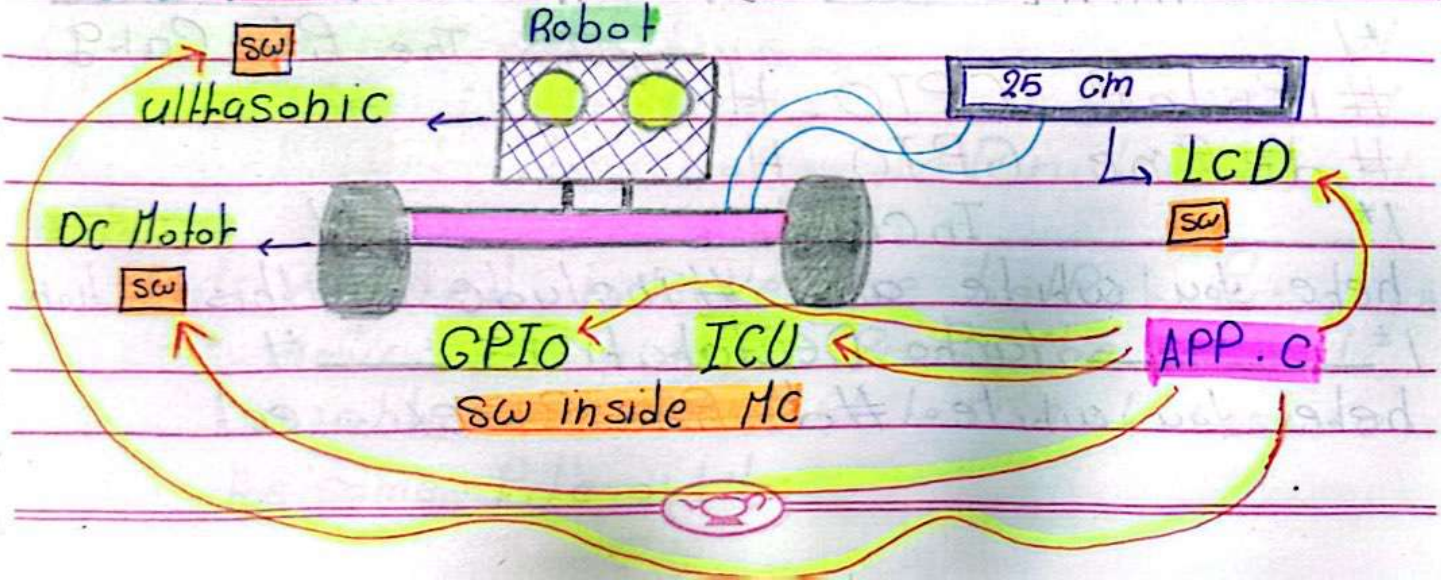
Modular Programming

is a software design technique :- you organize the software of your Project rather than you write all the software inside the main.c file you arrange the code into files Interface & Program & main.

the Modular Programming

how I want to Make a Project Firstly I have to know the target Drivers that will be used in the Project & I will call the functions of those drivers inside the APPlication "main.c" file.

For example I want to Make a Robot determines the distance & shows it on the LCD



Page:..... Date:.....

all of those Software are called by the Application or main.c file.

any Software Module Contains atleast these files

* Static Files
  these files you can't change it's content of the code inside them atleast these files are:-
  Static Source file ex :- GPIO.c "Program"
  Interface header file ex :- GPIO.h "Prototypes"
* Configration Files
  these files you can set the Configration of your Module inside it as you want.
* Generators:
  you use inside it a scripting language to make the configration files like "Python"
* Make Files
  are to make the build of the files too easy

the structure of the header file "Interface.h"

```
/*
 * File :- GPIO.h                    <- The documention
 * Author : Ziad Osama
 * https://www.linkedin.com
 * Created on : May 21, 2024, 10:00 PM
 */
#ifndef GPIO_H                       <- The File Guard
#define GPIO_H
/*_____Includes_____*/
here you write any #include at this Position
/*_____Macro Declarations_____*/
here you write #define for ex .... ect
```

Page: Date:

/* Macro Function Declaration */
You write the functions like Macro at this Position
/* Data type Declaration */
You write here the data types like struct or enum or union
/* Software Interfaces Declation APIs */
You write here the Photo types of the function that used for the Driver
#endif ← The end of File Guard

the structure of source file "Program.C"

/* ← The Documention
* File : GPIO.C
* Author : Ziad Osama Mahmoud
* http://www.linkedin.com
* Created on May 21, 2024, 10:00 PM
*/
/* Includes */
#include "GPIO.h" ← Call the Interface File
/* Global Variables Definition */
static uint8 Flag = 0;
/* Helper functions declaration */
/* APIs Definitions */
/* ISR Definitions */
/* Helper Functions definitions */

Note
it's Prefered to give the helper functions "static" data type & for the global variables if you want to use it just in the same file only.

Page: ..... Date: .....

note at least a simple embedded APPlication has 4 layes :.

* APPlication layer
* Electronic Control unit Abstraction layer
* Hardware Abstraction layer
* Micro Controller Abstraction layer «low level»

* Maybe I use diffeRent MC & I want this doesn't effect on other layers ←

* any layer is independent because if I want to edite at any layer this doesn't effect on other layers

layout is clear for the Architecture layer there are two schools for that school ① is 3 layers & schoo ② is 4 layers :.

3 layers Arch

	APP L
Library	HAL
	MCAL

* Library is shared for the 3 layers.
* MCAL : has the sw Module related to the PeriPherals inside MC.
* HAL : has sw Module of Connected devices in the MC.

4 layers Arch

	APPL
Library	ECU AL
	HAL
	MCal

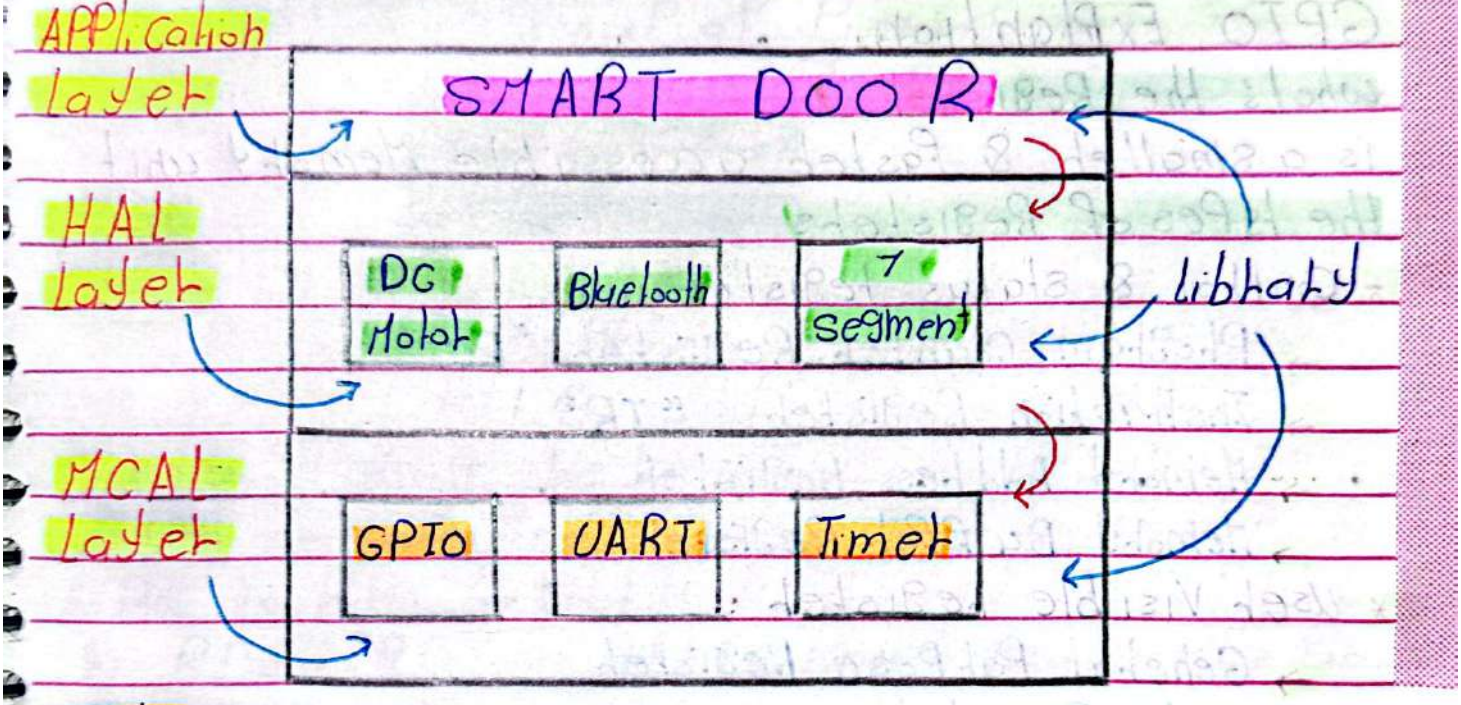
* the same thing in the 3 layers but we Consider the HAL & Mcal layers are inside the MC but the Mcal is so near to the register level of the MC & ECUAL layer is for the connected devices.

Page:..... Date:.....

note
= Application layer always call the layer that before it whatever ECUAL or HAL layer.

note
= Application layer can't call MCal layer directly but it has to call the HAL layer firstly after that calls the MCAL layer with indirect way.

The Architecture Layer:



note
= smart Door layer will call the HAL layer & HAL layer will call the MCAL layer but the smart Door layer can't call the MCAL layer direct #.

note
= library is shared for all layers.

Page: ..... Date: .....

New Topic

Interfacing

GPIO Explantion

What's the Register :

is a smaller & faster accessible Memory unit

the types of Registers

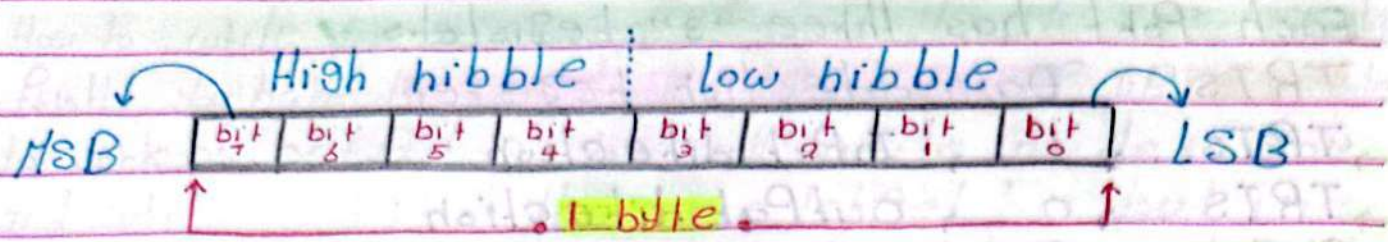
* Control & status register :-
  - → Program Counter Register "PC"
  - → Instruction Register "IR"
  - → Memory Address Register
  - → Memory Buffer Register
* User Visible register :-
  - → General Purpose Register
  - → Data Register
  - → Address Register
  - → Conditional Register Code

note

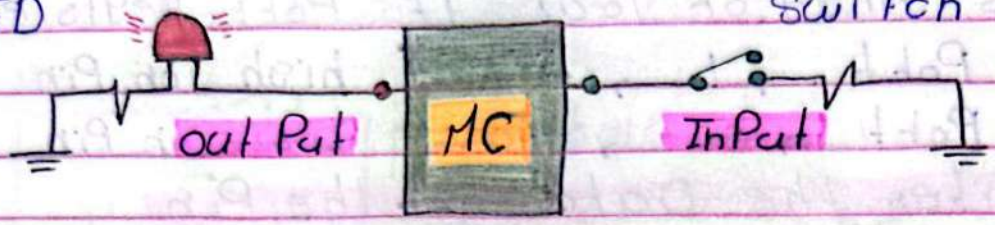
the register Maybe 8 or 16 or 32 bits according to the Architecture of the Micro Controller.

Page: ..... Date: .....

Register



GPIO "Digital InPut outPut"



what's the Port ::

it Consists of a group of Pins of the MC

note In our MC PIC18F46k20 we have 5 Ports
= Port A - Port B - Port C - Port D - Port E
From you reading the layout of the data sheet of MC you will find RA0 to RA7 for Port A & RB0 to RB7 for Port B & RC0 to RC7 for Port C & RD0 to RD7 for Port D finaly RE0 to RE3 for Port E.

note
= if the target Microcontroller from 8 bits family the size of Most Registers is 8 bits & if the MC from 16 bits family the size of Most register is 16 bits & .... ect till 32 & 64 bits family.

Page:..... Date:.....

Each Port has three "3" registers :

TRIS : Data direction register

→ TRIS → 1 → Input direction

→ TRIS → 0 → output direction

PORT : Reads the Data of a Pin

this Register reads the status of the Pin whatever this Pin has high or low if Port register reads :

→ Port → 1 → "5V" high on Pin

→ Port → 0 → "0V" low on Pin

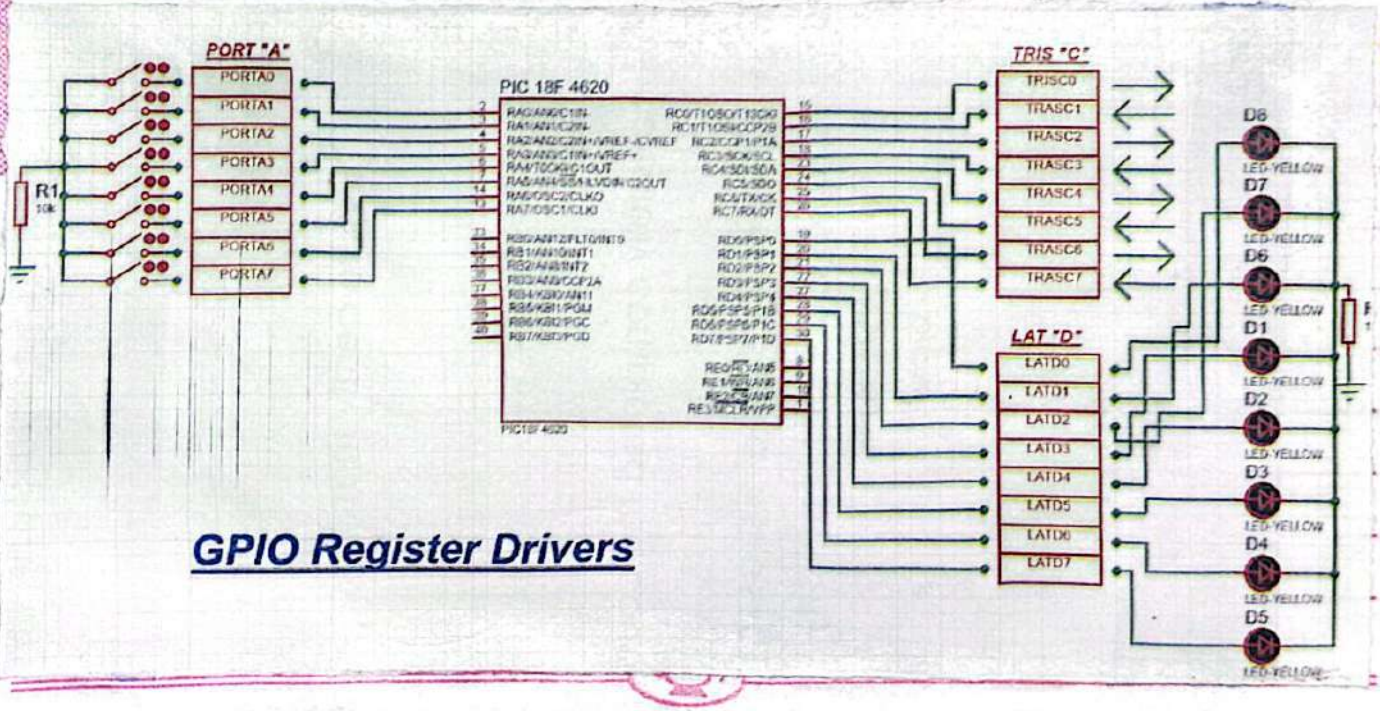
LAT : writes the Data on the Pin

is for the out Put action its called "latch out Put".

→ LAT → 1 → the Pin has "5V" high

→ LAT → 0 → the Pin has "0V" low

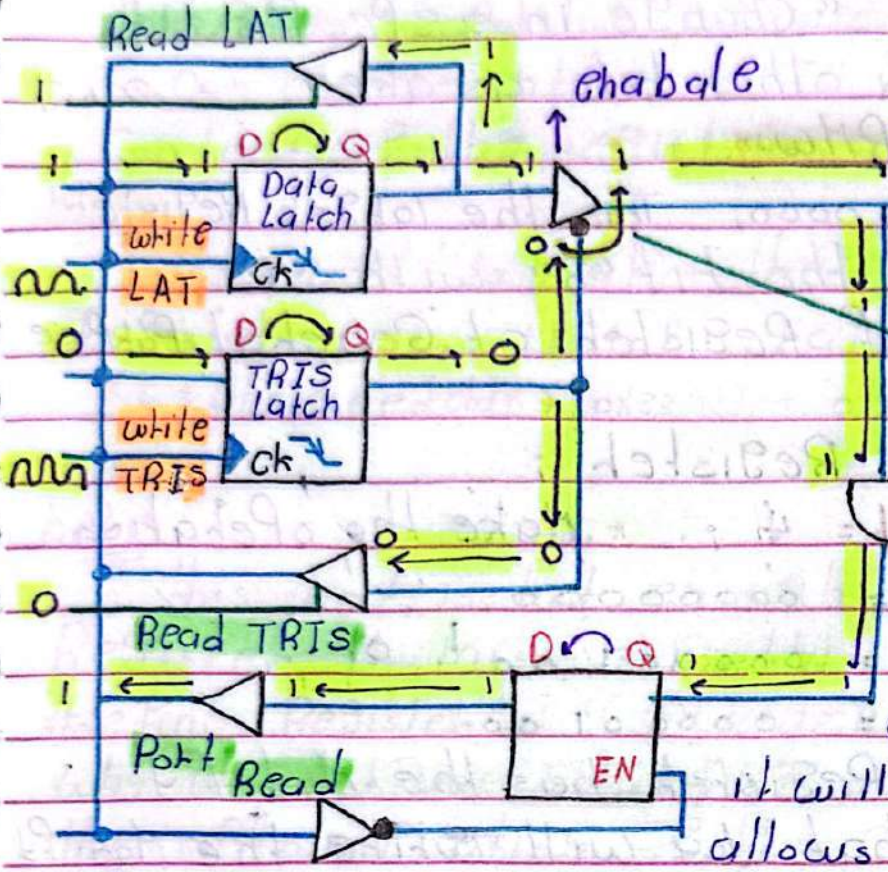
Ports	Port A	Port B	Port C	Port D	Port E
Direction	TRIS A	TRIS B	TRIS C	TRIS D	TRIS E
write	LAT A	LAT B	LAT C	LAT D	LAT E
Read	PORT A	PORT B	PORT C	PORT D	PORT E



GPIO Register Drivers

Page: Date:

How to write 5V or 0V on a Pin of the Mc "logic circuit"
Firstly you have to select the target bit of TRIS register to work as out Put So you will write inside it "0" and write 0 or 1 at the same target bit but in the LAT Register to Make out the action on the target Pin see the next logic circuit
This circuit is For each Pin at the Micro Controller



note
Read TRIS Path This is just For Reading what you write on TRIS write the same Concept is applied on Read LAT Path.

note
the Data at node "D" transfers to node "Q" after the operating the Clock on each latch.

Page:..... Date:.....

hote if the TRIs is works as InPut the LAT latch can't Pass any data to the Pin because the Inverter will be disable " X ".

whats the Read Modify write oPeration "RMW" : its oPeration ensure that you Modify only the sPecific bits in a system register that you want to change "change in a sPecific bit in the Register & other bits as are"

the steps to use "RMW"

Register = 0x 0000 0000; This the Target Register I want to chang the bit "2" with "1"

* CoPy the value of Register at General PurPose register

Temp_Register = Register ;

Temp_Register |= 4 ; * Make th oPeration

Temp_Register = 00000000

4 = 00000100 ! or

Temp. Register = 00000100

* now the Temp_Register has the Modify value after that you will write the Modify value at the Main register.

Register = Temp. Register;

Register = 0x 0000 0100 ;

hote

You use RMw if you want to change the values of sPecific bits at one time

" X " PortC0 = 1;
PortC1 = 1;

" ✓ " PortC = Value;

Page:..... Date:.....

GPIO Device Driver

How to Pick up the Registers of the Driver

```
#define Register (*((unsigned char *)(0x33)))
```

- Register: * This the target Register
- (*(...)): * This is the dereference operation to can change in the Content of the register
- (unsigned char *): * explaced Casting to say this a Pointer for Address
- (0x33): * the Address of Register

note

this way to Make the Matter is so easy Register = ob oollolol; rather than use *((unsigned char*)(0x33)) = ob oo ll olol; each time.

note

this way is not enough to Pick up the Register it miss something "Volatile"

```
#define Register = (*((Volatile char*)(0x33)))
```

why we did use Volatile?

the optomization of the Compiler when see this expersion without Volatile thinks that expersion is not important to use so it deletes this expersion from your code to decrease the number of the Instructions inside the Flash Memory at the Micro Controller.

the values that Could be changed without using Volatile:

⇒ Memory-Mapped Peripherals registers

Page:..... Date:.....

→ Global Variables modified by an interrupt service routine.

→ Global Variables accessed by Multiple tasks within a Multiple threaded application.

the syntax of Volatile

* For Variables:

Volatile uint32 Var1 ; → both are the
uint32 Volatile Var2 ; same

* For Pointers

Volatile uint32 * Ptr1 ; → both are the
uint32 Volatile * Ptr2 ; same

* Volatile Pointer to non Volatile Data or Var

uint32 * Volatile Ptr ;

* Volatile Pointer to Volatile Variable

uint32 Volatile * Volatile Ptr ;

the Volatile with the struct:

#define IO Volatile : May be not used

typedef struct {
_IO uint8 CSR ; → has Volatile
_IO uint8 CCR ; → has Volatile
uint8 AD ; → doesn't have Volatile
_IO uint8 CDR ; → has Volatile
} ADC_Command_TYPEDEF;

↳ here I Can Control in giving the Volatile to the elements as I want

typedef Volatile struct {
uint8 RC0
uint8 RC1 → here all elements
} Port_C ; have Volatile

Page: Date:

Creating Software Interfaces for HW:

→ we consider for the design software to be independent of the Architecture & Platform to can use the same software to other Platforms or other MCs.

Interface Methods for the Hw "Access Registers"

→ Register definition Files : May be from the company.
→ Macro Functions : Macro functions to interface.
→ Specialized C-Functions : the APIs to interface.

note
= the name of the function is not changed with the changing of the Platform.

Register definition Files → "Pic18F4620"
it Provides : "Platform Dependancy" F80h PORTA
→ Address list for Peri Pherals Register.
→ Access Methods.
→ Defines for Bit Fileds & Bit Mask.

Macro functions to access the Registers
you can Make it as a Configration to be independent of the any Platform ex →

```
#define  Platform  PIC18F
#if  Plat_Form == PIC18F
     #define HwReg(_x) (*((volatile uint8 *)(_x)))
#elif Plat_Form == DIS_PIC
     #define HwReg(_x) (*((volatile uint16 *)(_x)))
#elif Plat_Form == PIC32
     #define HwReg(_x) (*((volatile uint32*)(_x)))
#else
     #error "not supported Platform"
#endif
```

Page:..... Date:.....

* to define or Pick up the Register after the function Macros Configration

#define PORTA (HWReg(F80h))

#what ever the target platform I am right#

PORTA = 0b11111111;

Proteus :

→ Target Mc :: "PIC18F46k20"

→ Flash size :: 64 kB

→ RAM size :: 3936 B

→ EPROM size :: 1024 B

The Bit Manipulation : go to a spec fic bit at the register & change its values

	7	6	5	4	3	2	1	0
Register	0	1	0	1	1	0	1	0

I want to Make Bit3 = 0 & other Bits ake the sam just I change in Bit3 Value.

this called operation & May be is

set or clear or Toggle (1→0, 0→1)

① ⓪

the operation equations

Set equation :

Ineed to set bit (4&5) at the Register = 0xCF

Register = 0xCF → 1100 1111 → Main Value

bit(4&5) ones = 0x30 → 0011 0000 → Mask Value

Register = Register | Mask Value

↳ new Register ↳ Main Value

Value Register = 1111 1111

Register | = Mask Value

Page: Date:

Clear equation

I need to clear bit (4 & 5) of the Register = 0xFF

Register = 0xFF → 1111 1111 → Main Value

bit(4&5) ones = 0x30 → 0011 0000 → Mask Value

bit(4&5) Zeros = ~(0x30) → 1100 1111 → ~(Mask Value)

Register = Register & (~(Mask Value))

↳ new Value ↳ Main Value

Register = 1100 1111 Register &= ~(Mask Value)

Toggle equation

I need to toggle bit (4 & 5) of the Register = 0x00

Register = 0x00 → 0000 0000 → Main Value

bit (4 & 5) ones = 0x30 → 0011 0000 → Mask Value

Register = Register ^ (Mask Value)

↳ new Value ↳ Main Value

Register = 00 11 0000 Register ^= Mask Value

---

note the size of the register at our plate form is 8 bits so we have 8 Property MaskValues

	7	6	5	4	3	2	1	0
#define Bit0_Mask (uint8) 0x1 ←	0	0	0	0	0	0	0	1
#define Bit1_Mask (uint8) 0x2 ←	0	0	0	0	0	0	1	0
#define Bit2_Mask (uint8) 0x4 ←	0	0	0	0	0	1	0	0
#define Bit3_Mask (uint8) 0x8 ←	0	0	0	0	1	0	0	0
#define Bit4_Mask (uint8) 0x10 ←	0	0	0	1	0	0	0	0
#define Bit5_Mask (uint8) 0x20 ←	0	0	1	0	0	0	0	0
#define Bit6_Mask (uint8) 0x40 ←	0	1	0	0	0	0	0	0
#define Bit7_Mask (uint8) 0x80 ←	1	0	0	0	0	0	0	0

note if I want to Make Mask Value for bit5 & bit3 & bit7 at one Mask Value you will or all of them & use the one Mask Value with the target equation

Page: Date:

Ex →

Mask Value = (Bit3_Mask | Bit5_Mask | Bit7_Mask);

Set :

Register | = Mask Value

Clear :

Register & = ~(Mask Value)

Toggle :

Register ^ = Mask Value

rather than use the Mask Value we will use the shift left operation :

Set equation Mask Value

Register = Register | (1 << Bit_num)

Register | = (1 << Bit_num)

Clear equation Mask Value

Register = Register & ~(1 << Bit_num)

Register & = ~(1 << Bit_num)

Toggle equation Mask Value

Register = Register ^ (1 << Bit_num)

Register ^ = (1 << Bit_num)

Make those equations as function Macro to be easy to use the operation

```
#define Set_Bit (Reg, Bit_Posn)
        (Reg | = (1 << Posn))

#define Clear_Bit (Reg, Bit_Posn)
        (Reg & = ~(1 << Bit_Posn))

#define Toggle_Bit (Reg, Bit_Posn)
        (Reg ^ = (1 << Bit_Posn))
```

Page:..... Date:.....

Macho Functions Issues : "InV Q"

→ no type checking.
→ Bug Instruction.
→ Complex / Confusing layers of Machos Calling Machos.
→ Code size & Duplication.

what's the encapsulation

is consider as a group of variables data & some operation are inside a frame it's called function.

Using "inline" key words with function

I say to the compile don't generate the Assembly Code for this function just Post the operation at the calling function place. "InV Q"

→ inline unsigned int return_3 (void) {
    return (3);
}
3 ↳ This is inline function #

note

May be "inline" key word is not supported in other compilers.

what's the difference between the function like Machos & the inline function : "InV Q"

all of them is Code size Duplication but inline function has type checking & functions like Machos doesn't have the type checking.

note

it's Prefered to the code of the inline function be small size.

what's the Variadic function : "InV Q"

is a function with unlimited arguments

void Print (char * name, ...);

Page: ..... Date: .....

GPIO implemention tools

to Make the layout of the register you have to Make an union has a struct for the bits of the register & other variable from uint8 type as the register overall.

note

union will support to you use the struct of the variable and the target use only will be allocated at the Memory

The implemention

```
typedef union {
    struct {                          <-- this struct is
        unsigned TRISC0 : 1;          presented to the
        unsigned TRISC1 : 1;          Bits of TRIS
        unsigned TRISC2 : 1;          register from
        unsigned TRISC3 : 1;          Bit0 to Bit7
        unsigned TRISC4 : 1;
        unsigned TRISC5 : 1;          <-- here I say that
        unsigned TRISC6 : 1;          each element at
        unsigned TRISC7 : 1;          the struct takes
    };                                1 Bit size.
    uint8 TRISCRegister;
} TRISC_t;                            <-- this to access the register overall
```

note

the size of the union at the Memory will be from uint8 size because the large element in the union is "TRIS_Register" from uint8 size

Page: ..... Date: .....

#define TRISC_Reg ((Volatile TRISC_t *)(0xF94))

Note

where the dereference ? If you want to Theate with the union as a Pointer you will not Make the dereference

Ex →

Pass By Reference

TRISC_Reg → TRISC0 = 0 ;

TRISC_Reg → TRISC_Register = 0x55 ;

Note

If you want to Theat with the union as Pass by Value you will Make the dereference

#define TRISC_Reg (*((Volatile TRISC_t *)(0xF94)))

TRISC_Reg.TRISC0 = 0 ; Pass By

TRISC_Reg.TRISC_Register = 0x55; Value

May be I have a group of bits at specific register as you see from SSPM3 to SSPM0 you will create them as one element of struct but you will Mak the size for it is 4 Bits only according to the Configration at the Data sheet other elements will be 1 Bit & the Register over all Value #

REGISTER 17-2: SSPCON1: MSSP CONTROL REGISTER 1 (SPI MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV(1)	SSPEN(2)	CKP	SSPM3(3)	SSPM2(3)	SSPM1(3)	SSPM0(3)
bit 7							bit 0

```
typedef enum{
    SPI_MASTER_MODE_FOSC_DIV_4,
    SPI_MASTER_MODE_FOSC_DIV_16,
    SPI_MASTER_MODE_FOSC_DIV_64,
    SPI_MASTER_MODE_FOSC_TMR2,
    SPI_SLAVE_MODE_SS_ENABLED,
    SPI_SLAVE_MODE_SS_DISABLED
}SSPMx_Values_t;

typedef union{
    struct{
        unsigned SELF_SSPMx : 4;
        unsigned SELF_CKP   : 1;
        unsigned SELF_SSPEN : 1;
        unsigned SELF_SSPOV : 1;
        unsigned SELF_WCOL  : 1;
    };
    uint8 SSPCON1_REGISTER;
}SELF_SSPCON1_REG;
```

```
#define SELF_SSPCON1_REG ((volatile SELF_SSPCON1_REG *)0xFC6)
#define P_SELF_SSPCON1_REG ((volatile SELF_SSPCON1_REG *)0xFC6)

SSPMx_Values_t SSPMx_val = SPI_SLAVE_MODE_SS_DISABLED;
SELF_SSPCON1_REG->SELF_SSPMx = SSPMx_val;
```

SSPCON1	0x0FC6	0b00000101
SSPMx	0x0FC6	5
CKP	0x0FC6	0
SSPEN	0x0FC6	0
SSPOV	0x0FC6	0
WCOL	0x0FC6	0

bit 3-0 **SSPM3:SSPM0:** Master Synchronous Serial Port Mode Select bits(3)

0101 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control disabled, $\overline{SS}$ can be used as I/O pin
0100 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control enabled
0011 = SPI Master mode, clock = TMR2 output/2
0010 = SPI Master mode, clock = Fosc/64
0001 = SPI Master mode, clock = Fosc/16
0000 = SPI Master mode, clock = Fosc/4

Note 1: In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.
2: When enabled, these pins must be properly configured as input or output.
3: Bit combinations not specifically listed here are either reserved or implemented in I^2C™ mode only.

Page: ..... Date: .....

*

Note

In our Micro Controller we have some Pins & Some Ports & types of the logic & types of the directions we need to present that to Make all the Information More readable May be Make that by the #define or the enum user data type

the Pin Index enum

typedef enum {
    GPIO_Pin0 = 0,
    GPIO_Pin1,
    GPIO_Pin2,
    GPIO_Pin3,
    GPIO_Pin4,
    GPIO_Pin5,
    GPIO_Pin6,
    GPIO_Pin7,
} Pin_Index_t;

* This enum is for the Bit index rather than use a number I don't know what does Mean I use just from this enum to Make the operation is More readable to the developer.

the logic enum

typedef enum {
    GPIO_low = 0,
    GPIO_High
} logic_t;

* This enum is for to the out Put action on the Pin this Pin will get low "0V" or high "5V".

the direction enum

typedef enum {
    GPIO_direction_outPut = 0,
    GPIO_direction_InPut
} direction_t;

* This enum is for setting the direction of the Pin.

the Port index enum

Page: Date:

```
typedef enum {
    PORTA = 0,
    PORTB,
    PORTC,
    PORTD,
    PORTE,
} PORT_index_t;
```

* This enum is for setting the target port to set a specific pin at the port to Make it is Input or output direction of High or Low logic

let's see the function what does work

```
Void GPIO_Pin_Initialize (
    PORT_index_t PORT,
    Pin_Index_t Pin,
    direction_t Direction,
    type logic_t logic);
{ /* your code here */ }
```

those are the Parameter

How to use this function

```
GPIO_Pin_Initialize ( PORTA,
                      GPIO_Pin0,
                      GPIO_direction_Input,
                      GPIO_High );
```

as you see you use the elements of the enums

note

later than Make those Parameters to the function can collect them in a struct & Pass the struct object to the function

as you see

```
typedef struct {
    uint8 Port : 3;
    uint8 Pin : 3;
    uint8 direction : 1;
    uint8 logic : 1;
} Pin_Config_t
```

```
Pin_Config_t led1 = {
    .Port = PORTA,
    .Pin = GPIO_Pin0,
    .direction = GPIO_Input,
    .logic = GPIO_High,
};
```

→ create & initialize object of struct Config initialized to enums

Page:..... Date:.....

note

we divided the elements of the struct Pin_Config_t to Bits this to less the size of the struct inside the Memory let's see

way ①

```
typedef struct {
    uint8 Port      : 3 ;
    uint8 Pin       : 3 ;
    uint8 direction : 1 ;
    uint8 logic     : 1 ;
} Pin_Config_t ;
```

This way will take 3+3+1+1 = 8 Bits = 1 byte at the Memory

way ②

```
typedef struct {
    uint8 Port ;
    uint8 Pin ;
    uint8 direction ;
    uint8 logic ;
} Pin_Config_t ;
```

This way will take 1+1+1+1 = 4 bytes at the Memory

after you created the struct Pin_Config_t & an object of it led1 & Make the initialization how I wan to Make the GPIO_initalization_Pin Function & give it the led1 object as a Pass Patameter May be I Make this by Two ways

way ① : "Pass By Value"

```
void GPIO_initialize_Pin ( Pin_Config_t obj );
GPIO_initialize_Pin ( led1 );
```

way ② : "Pass By Refernce"

```
void GPIO_initialize_Pin (Pin_Config_t * obj );
GPIO_initialize_Pin (&led1 );
```

note

to Make your Configrations are Goveth to don't allow to chang the Values use "Const"

Page:..... Date:.....

Void GPIO_initialize_Pin(Const PinConfig_t * obj);
↳ GPIO_initialize_Pin(&led1);

---

the APIs for the GPIO Driver :-

1- GPIO Pin direction intialize :-

```
/*_____ GPIO_PIN_DIRECTION_INTIALIZE Function
std_ReturnType GPIO_PIN_DIRECTION_INTIALIZE
(
    const Pin_Config_t * PIN_TARGET_CONFIG
);
```

→ this function takes the Address of the created object of Pin_Config_t struct.
→ it intializes the direction of the specific Pin.

2- GPIO Get Pin direction status

```
/*_____ GPIO_PIN_GET_DIRECTION_STATUS Function

std_ReturnType GPIO_PIN_GET_DIRECTION_STATUS
(
    const Pin_Config_t * PIN_TARGET_CONFIG ,
    Direction_t * DIRECTION_STATUS
);
```

→ this function is to read the direction of the specific Pin you will read the direction by a Pointer from Direction_t Data type because the function return data from std_ReturnType data type.

Q why the data type of Pointer Direction_status from Direction_t not uint8 ?
because I need to read direction of one Pin not a complete Port.

Page:..... Date:.....

3- GPIO write logic on Pin

```
/*_____ GPIO_PIN_WRITE_LOGIC Function

std_ReturnType GPIO_PIN_WRITE_LOGIC
(
    const Pin_Config_t * PIN_TARGET_CONFIG,
    Logic_t LOGIC_PIN
);
```

→ This function to get out high "5V" or low "0V" on a target Pin accordding to the Pin element at Pin_Config_t struct.
→ takes the configration of the Pin.
→ takes low or HIGH accordding to logic_t enum.

4- GPIO Read logic from the Pin

```
/*_____ GPIO_PIN_READ_LOGIC Function

std_ReturnType GPIO_PIN_READ_LOGIC
(
    const Pin_Config_t * PIN_TARGET_CONFIG,
    Logic_t * LOGIC_PIN
);
```

→ This function reads the logic whatever is high or low from the Pin you will read the logic By a Pointer because this function return another data from std_ReturnType Data type.

5- GPIO Toggle the logic at the Pin

```
/*_____ GPIO_PIN_TOGGLE_LOGIC Function

std_ReturnType GPIO_PIN_TOGGLE_LOGIC
(
    const Pin_Config_t * PIN_TARGET_CONFIG
);
```

Makes the high is low and low is high.

Page:..... Date:.....

6- GPIO Intialize the direction of Port

```
/*_____ GPIO_PORT_DIRECTION_INTIALIZE Function

std_ReturnType GPIO_PORT_DIRECTION_INTIALIZE
(
    PORT_Index_t PORT_TARGET,
    uint8 DIRECTION
);
```

→ This function sets the direction of the Port overall rather than set the directions of the Pins individual

→ takes the Target Port from Port.Index.t ehum & the Value from 8 bits are Presented to Zeros & ones.

7- GPIO Get the direction of the Port

```
/*_____ GPIO_PORT_GET_DIRECTION_STATUS Function
std_ReturnType GPIO_PORT_GET_DIRECTION_STATUS
(
    PORT_Index_t PORT_TARGET ,
    uint8 * DIRECTION_STATUS
);
```

→ this function Just Gets the direction of the Port you will Make this by a Point because this function Return Data from std.ReturnType Data types.

8- GPIO write logic on a Port

```
/*_____ GPIO_PORT_WRITE_LOGIC Function

std_ReturnType GPIO_PORT_WRITE_LOGIC
(
    PORT_Index_t PORT_TARGET,
    Uint8 _t LOGIC_PORT
);
```

this Function is for writing logic on a Port overall

Page:..... Date:.....

9- GPIO Read the logic from the Port

```
/*_____ GPIO_PORT_READ_LOGIC Function

std_ReturnType GPIO_PORT_READ_LOGIC
(
    PORT_Index_t PORT_TARGET,
    uint8 * LOGIC_PORT
);
```

→ This function to Read the logic at the Port overall but with a Pointer too because this function returns data from std.ReturnType Data type.

10- GPIO Toggle logic Port

```
/*_____ GPIO_PORT_TOGGLE_LOGIC Function

std_ReturnType GPIO_PORT_TOGGLE_LOGIC
(
    PORT_Index_t PORT_TARGET
);
```

→ this function to toggle the logic at the Port overall at one time.

Led interfacing

hole

= You have to read & know the electrical characteristics of the Micro Controller like temperature - Max Voltage - Current - Power.

hole

= Max Current enters into a Pin is 25 mA & the output Max Current from the Pin is 25 mA.

hole

= Max Current for the all Ports in MC is 200 mA output & Input.

Page: ..... Date: .....

Note

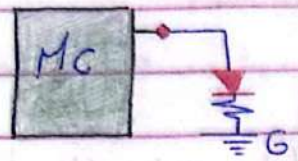
Max Current out of V_{SS} Pin = 300 mA.

Max Current into V_{DD} Pin = 250 mA.

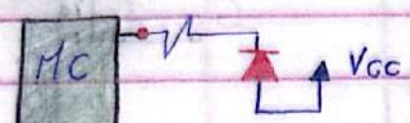
Connecting LED with MC

there two types for the connection:

Source



Sink



the Law to Calculate the used resistor with a LED

$$R = \frac{(V_s - V_{led})}{I_{led}}$$

where:

R → the resistance value.

V_s → the Voltage of the Micro Controller.

V_{led} → the Voltage of LED.

I_{led} → the Current of LED.

Note

from the Data sheet of the LED $V_{led} = 2.2V$ & $I_{led} = 10\,mA$ & the Voltage of Micro Controller May be is 3.3V or 5V So $V_s = 3.3V$ or $V_s = 5V$ So we have to options to Calculate the Value of the resistance when

$V_s = 3.3V$	$V_s = 5V$
$R = \frac{(3.3 - 2.2)}{10 * 10^{-3}} = 110\,\Omega$	$R = \frac{(5 - 2.2)}{10 * 10^{-3}} = 280\,\Omega$

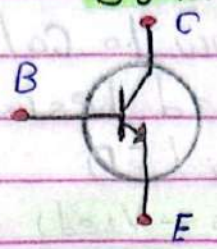
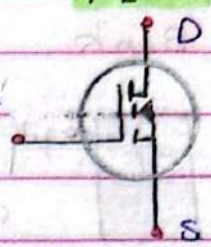
How to Connecte a group of LED at one Pin

to Control the loads with high Current like.. "Motor, Relays, More than one LED"

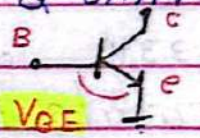
Page:..... Date:.....

The Current of Mc "Pin" will not support that (25 mA) Transistors used as (control switche) for that

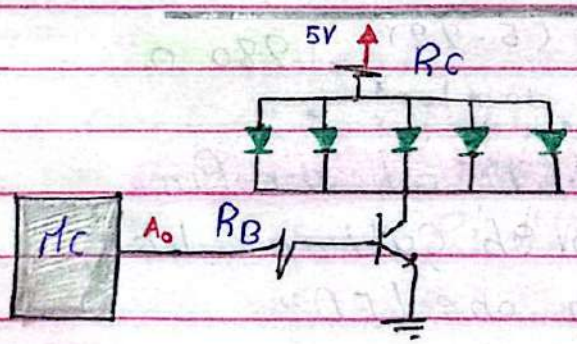
Types of used Transistors "Depends on load"

BJT	FET
	
→ Control the current from base "I_B".	→ Controlled through the voltage at Gain "V_G".
→ switching speed ~ 200 MHz.	→ switching speed ~ 10 times faster than BJT.
→ More affected by heat.	switching speed = 200×10 MHz
→ used with low current loads.	→ less affected by heat
	used with high current loads.

How to Make the Transistor works as a switch it Must the Voltage between the base & emitter is higher than 0.7 V → $V_{BE} \geq 0.7$ V



* when $V_{BE} < 0.6$ V → this Means off states.
* when $V_{BE} \geq 0.6$ V → this Means on states.



when A_o = 5V high the "T" will works as short circuit the leds will be lighted

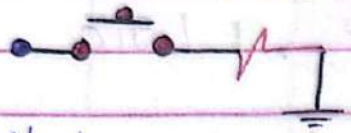
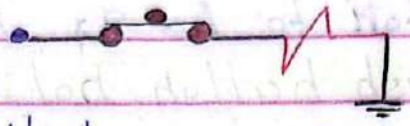
→ short circuit

Page: Date:

when A0 : 0V low the "T" will work as open circuit

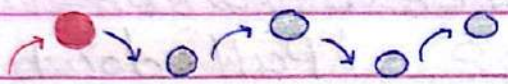
the leds will be turned off → open circuit no connection

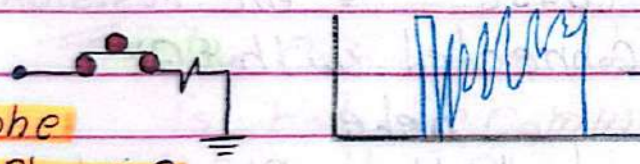
Switches & Push buttons interfacing

not Pressed	Pressed
	
there's no current Pass	there's a current Pass

note

there's a bouncing occours when you Press on the Push button.

 this is the bouncing when you Press on a Push button is connected on a MC the MC reads zeros & ones by just one Pressing with the Push button

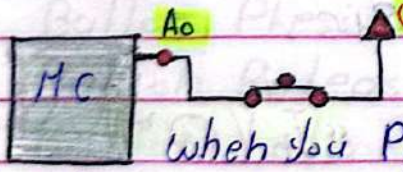


there's a bouncing

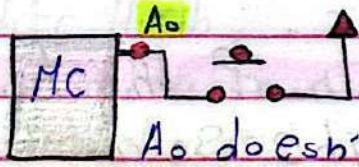
one Pressing

How to connect the Push button with the MC

let's see some states



when you Press on Push button A0 reads (5V) → 1 logic.

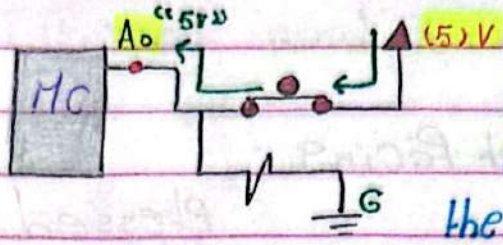


A0 doesn't read (0V) → Zero logic. "floating state"

Page:..... Date:.....

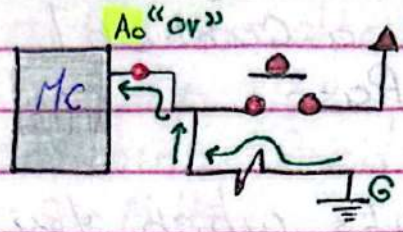
hole floating state is unlimited voltage at the Pin I don't know is 1 logic or 0 logic

So what's the solution



By Connecting ground with resistor Parallel with the Push button at this case the current will Pass through the Push button to Ao so Ao reads "5V" → 1 logic

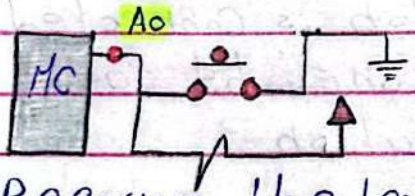
if the Push button not Pressed



here the Push button is not Pressed & it's Pin Ao is not at the floating state because it's Connected with Ground so it reads "0V" → 0 logic

what's the difference between the

Pull up

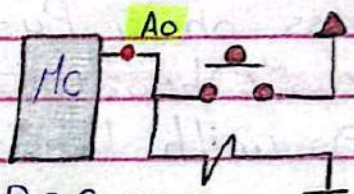


Because the resistor is Connected with "5V" here

→ Push button Pressed
Ao = 0 logic.

→ Push button not Pressed
Ao = 1 logic.

Pull down



Because the resistor is Connected with "G" here

→ Push button Pressed
Ao = 1 logic.

→ Push button not Pressed
Ao = 0 logic.

"InVQ"

the Solution of the bouncing

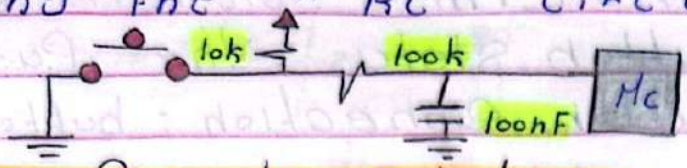
→ Hardware Solution.

→ Software Solution.

Page: Date:

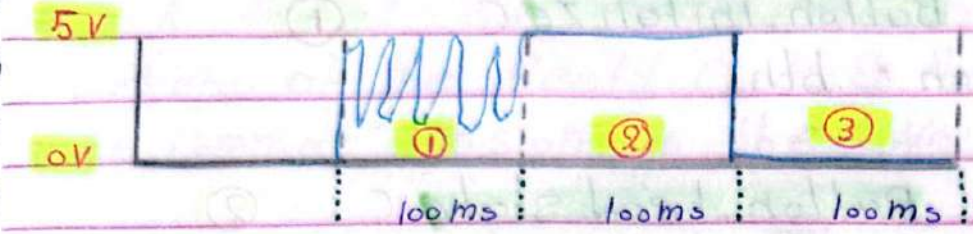
the Hard wake Solution

* Using the "Rc" circuit



the Soft wake Solution

the bouncing occurs during 100 msec



we will slot the Period of the bouncing into 20 slots this mehs I have Max Counter = 20 so I will increase the Counter each 5msec if the Counter = 20 this means the bouncing is finished & red 1 logic after that from the second Period of time at the third Period the Pin reads 0 logic

First Period	second Period	third Period
Counter ! = 20	Counter = 20	Counter = 0

hote

You Can use the delay but it is hot recommended because you stop the Processor

Push Button inter face header file

typedef enum {	typedef enum {
Button_Pressed,	Active_High,
Button_Released	Active_low
} Button_state;	} Button_Active;
Check the state	Check if PullUP or Pull down resistor

Page: ..... Date: .....

```
typedef struct {                    Collection
    Pin_Config_t  Button_Pin ;      elements of
    Button_state  Button_status ;   the Push
    Button_Active Button_Connection ; button
} Button ;
```

the interface function are

```
std_ReturnType Button_Initalize (          (1)
    Const Button * btn
) ;
std_ReturnType Button_Read_State (         (2)
    Const Button * btn ,
    Button_state * btn_state
) ;
```

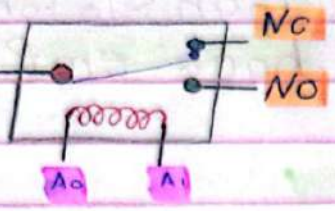
---

the solution of the bouncing debouncing

```
Return_std_type Ret = E_NOT_ok ;
uint8  btn_status = Released
uint8  btn_Valid , btn_Valid_status ,
while (1) {
    Ret = button_Read_status (&btn , &btn_status);
    if( btn_status == Pressed) {
        btn_Valid ++ ;          threshold value
        if (btn_Valid > 500) {
            btn_Valid_status = Pressed ;
        }                       here you will use the
    }                           btn Valid status for the
    else {                      checking.
        btn_Valid = 0 ;
        btn_Valid_status = Released;
    }
}
```

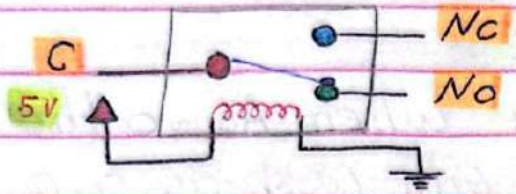
Page:..... Date:.....

Relay :: electromagnetic Relay

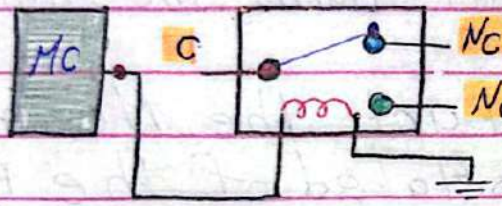


normally closed
normally opened

hole when there is a current pass from A0 to A1 there's an electromagnetic filed happens at the Realy Coil & this action Pulls the Common tension to the "NO" normally opened

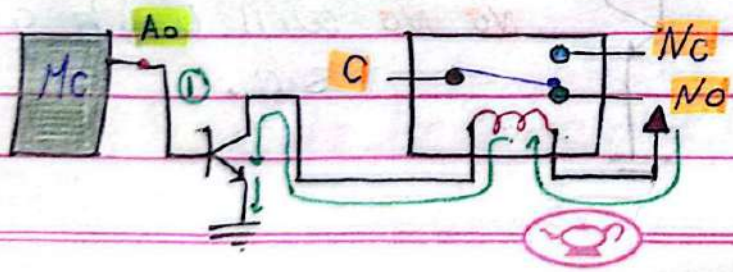


the Problem of interfacing the realy with MC
the Coil of the Realy is the load because one of its terminal will be connected with the MC.



the Pin of MC supports "5V" & this voltage doesn't work the Coil of the realy because it requires voltage is higher more than 5V.

what's the Solution
use a Transistor to work as a switch & support the relay with high Voltage



* when A0 = 1 the Transistor works as S.C so the Coil will work

Page:..... Date:.....

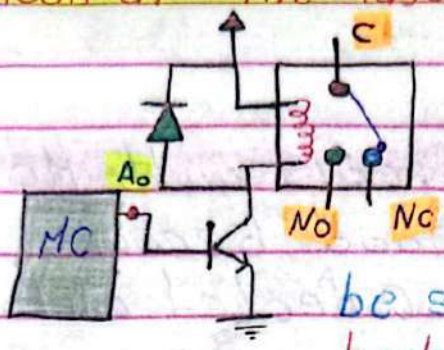
note

when Ao = 0 the Transistor will be burn because the **back emf** from the Coil at the realy.

what's the Solution :

* we will Connect a Diode with a Coil of the realy has the same Voltage of the realy.
* the Connection of the diode will be as reversal way.

look at the layout

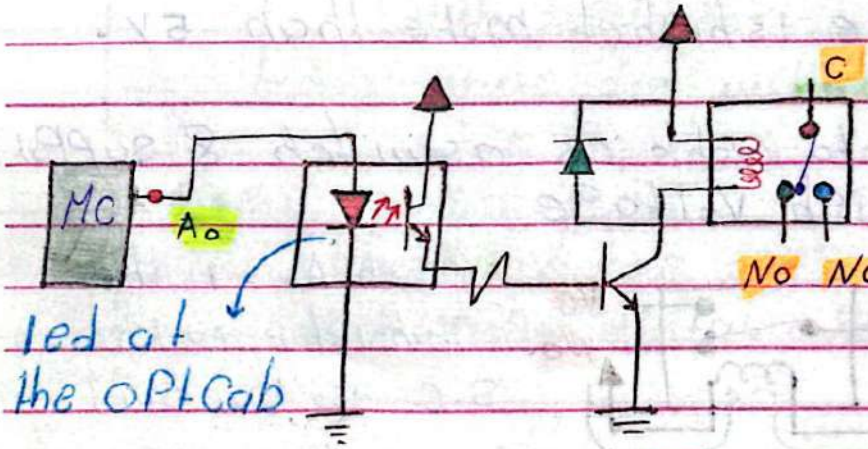


* when Ao = 0 the Transistor will work as O.C & the Diod will Pull the back emf from the Coil for a time till be stopped rather than the back emf burns the Micro Controller

note

we need to Protect the MC more By Making it more isolated of the Power Circuit so we will use the opto Cabler.

* when Ao = 1 the led of the the opto Cabler will be light & the Transistor will work as S.C.



Page: Date:

Relay interface header file

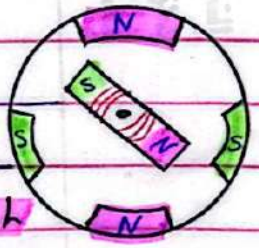
```
#define  Relay_on_status   0x01
#define  Relay_off_status  0x00
typedef struct {
    uint8  relay_port  : 4 ;
    uint8  relay_pin   : 3 ;
    uint8  relay_status : 1 ;
} Relay_t ;
```

→ to create objects in main file

The APIs

```
Std_ReturnType Relay_Initialize(
    Const Relay_t * relay
);
Std_ReturnType Relay_turn_on
    Const Relay_t * relay
);
Std_ReturnType Relay_turn_off(
    Const Relay_t * relay
);
```

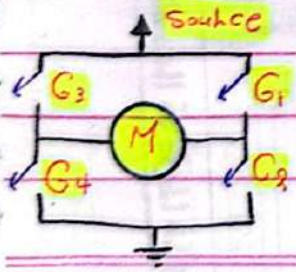
The DC Motor "Direct Current Motor" is used for Producing Continous Movement & whose speed rotation Can easily be Controlled



the fixed Part ← rotor ←
the Moving Part ← stator ←

How to Control in the direction of DC Motor

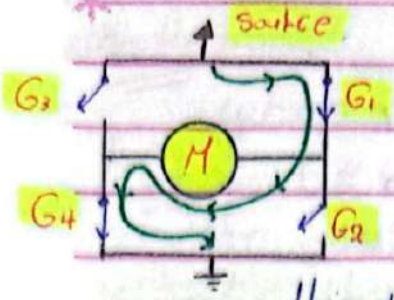
By using the H-Bridge Circuit



Open Case

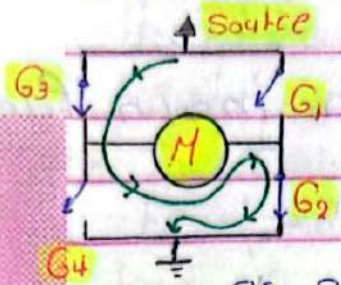
there's no Current Pass through any gate because allof them is open Circuit.

Page:..... Date:.....



Forward Case

G1 & G4 are closed so the current will pass from the source to G1 after that will pass through the Motor then G4 & this is consider as short circuit & the Motor will rotate forward.

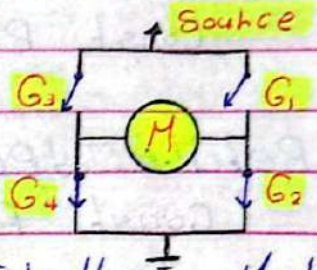


Backword Case

G3 & G2 are closed so the current will pass from the source to G3 after that will pass through the Motor then G2 & this is consider as short circuit & the Motor will rotate backword.

Note

the best case to stop the Motor is the Braking Case.

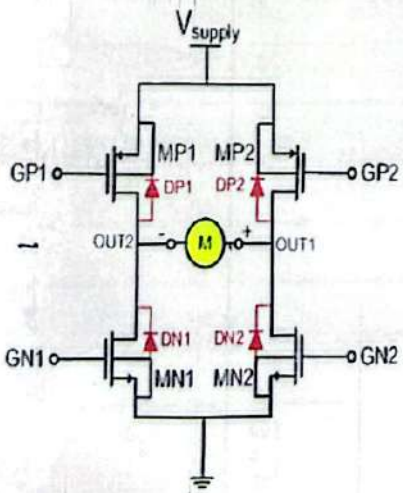


close the gates that are closed to the ground & open the gates that are closed to the Power source & there's no current will pass through the Motor

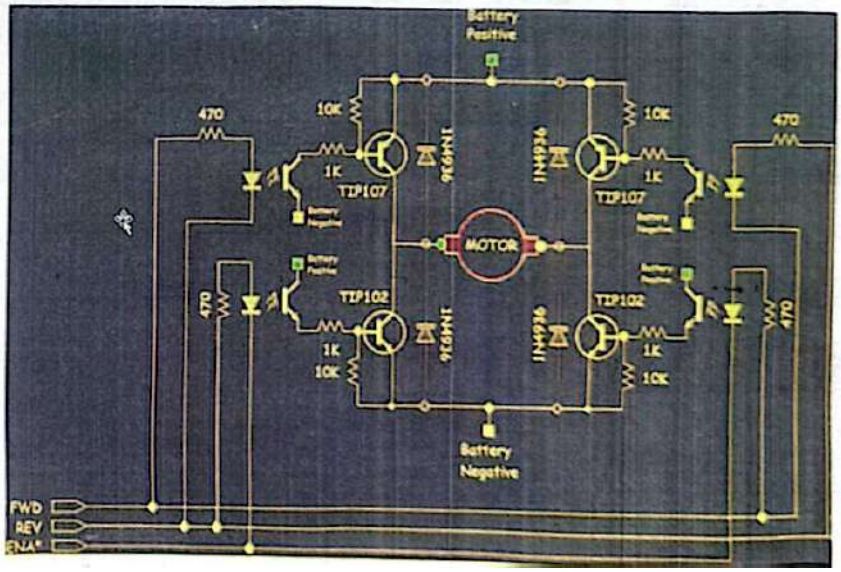
Note

to control we will use the Transistor rather than the gates opto coupler to protect the MC.

Simple



Complex Design



Page:..... Date:.....

the Motor driver

Support high voltage to the Motors & leads from the Micro Controller just "5V"

note

to Make the H-bridge circuit with the Dc Motor you have to get 2 Pins for the control & Pin for the enable & disable the H-bridge.

How to Connect the Motor driver with DcMotor

DC Motor Drivers (L298 Dual Motor Driver Module 2A) Connections

IN3 & IN4
- ✓ Pins are used to control spinning direction of Motor B.
- ✓ When one of them is HIGH and other is LOW, the Motor B will spin.
- ✓ If both the inputs are either HIGH or LOW the Motor B will stop.

ENB
- ✓ Pins are used to control speed of Motor B. PwM → enable Pin
- ✓ Pulling this pin HIGH(Keeping the jumper in place) will make the Motor B spin, pulling it LOW will make the motor stop.
- ✓ Removing the jumper and connecting this pin to PWM input will let us control the speed of Motor B.

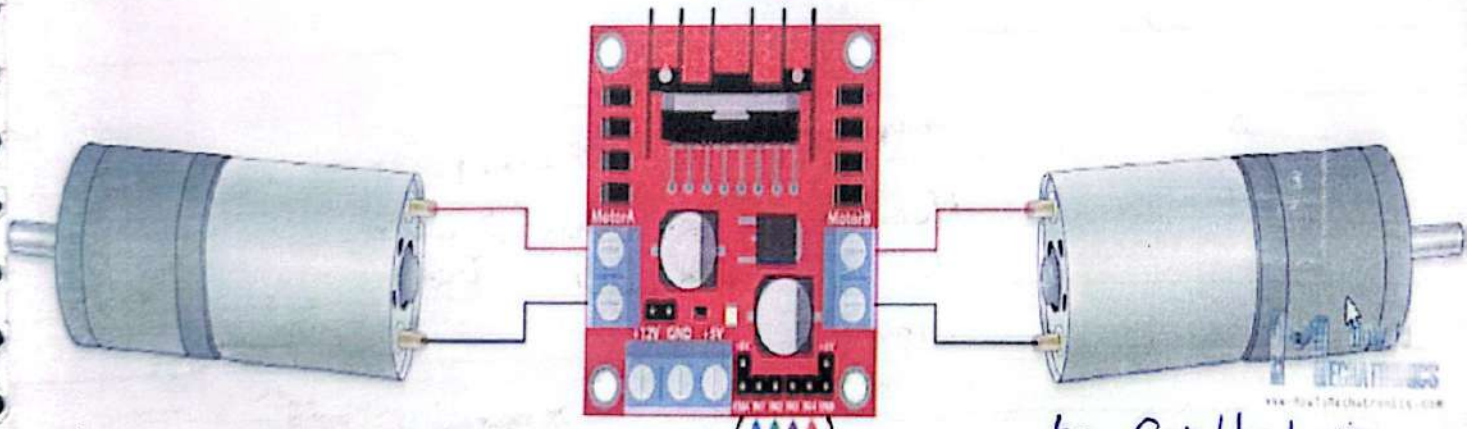
OUT1 & OUT2
- ✓ Pins are connected to Motor A.

OUT3 & OUT4
- ✓ Pins are connected to Motor B.

the best Case to stop the Motor

Input1	Input2	Spinning Direction
Low(0)	Low(0)	Motor OFF
High(1)	Low(0)	Forward
Low(0)	High(1)	Backward
High(1)	High(1)	Motor OFF

Direction Control Pins



to Control in the speed of the Motor ← PWM Signal

PWM Signal →

MotorA MotorB Control Pins

to Control in the logic of the Motor

Page:..... Date:.....

the interfacing of the DC Motor

```
#define DC_Motor_on_status 1
#define DC_Motor_off_status 0
```
to initalize the logic of the Pin with them.
```
#define DC_Motor_PIN1 0
#define DC_Motor_PIN2 1
```
to access the Array of Pin_Config_t struct with them.

```
typedef struct {
    Pin_Config_t DC_Motor_Pin[2];
} DC_Motor_t;
```

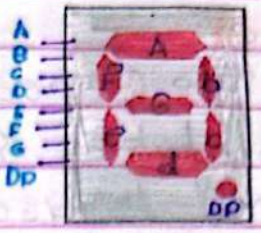
the APIs interfacing

```
Std_ReturnType DC_Motor_initalize(
        Const DC_Motor_t * dc_motor
);
Std_ReturnType DC_Motor_MoveRight(
        Const DC_Motor_t * dc_motor
);
Std_ReturnType DC_Motor_Move_left(
        Const DC_Motor_t * dc_motor
);
Std_ReturnType DC_Motor_stop(
        Const DC_Motor_t * dc_motor
);
```

Page:..... Date:.....

7-Segments Interfacing

- Is a Collection of Group of LEDs.
- It Prints the numbers from 0 to 9.

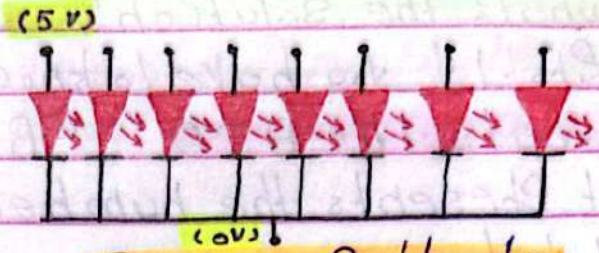


the types of 7-segment :

- Comman Anode.
- Comman Cathode.

the different is at the Connection way

Comman Anode



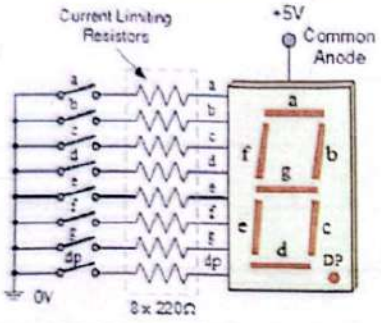
Comman Cathode

the tables to Print the numbers & the Connection

A	B	C	D	E	F	G	DP
1	1	1	1	1	1	0	0

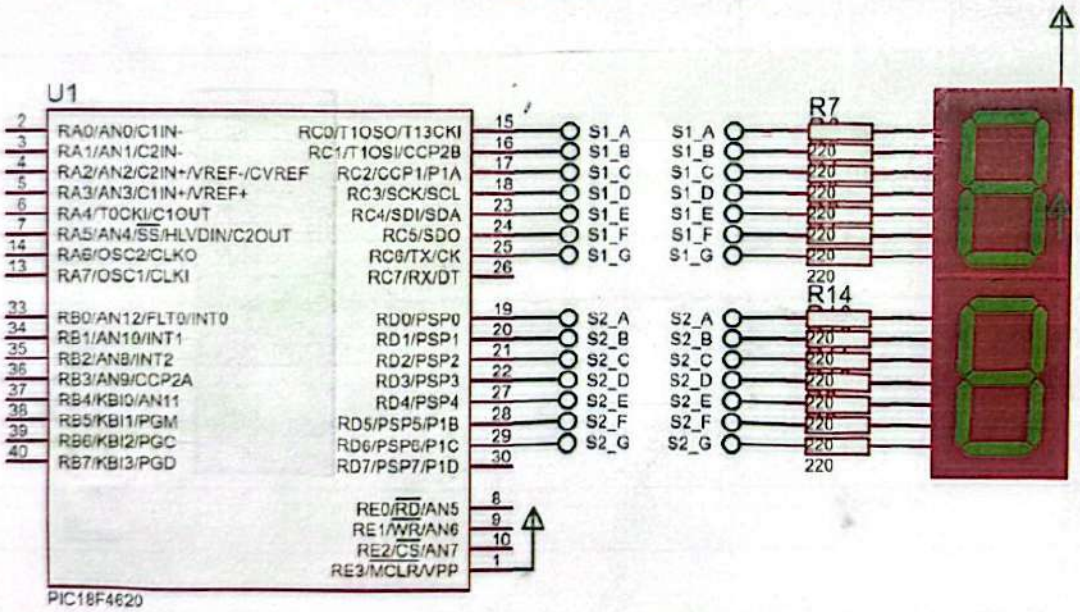
Table with display codes of all the digits with decimal point OFF.

Numbers	Common Cathode		Common Anode	
	(DP)HGFEDCBA	HEX Code	(DP)HGFEDCBA	HEX Code
0	00111111	0x3F	11000000	0xC0
1	00000110	0x06	11111001	0xF9
2	01011011	0x5B	10100100	0xA4
3	01001111	0x4F	10110000	0xB0
4	01100110	0x66	10011001	0x99
5	01101101	0x6D	10010010	0x92
6	011111101	0x7D	10000010	0x82
7	00000111	0x07	11111000	0xF8
8	01111111	0x7F	10000000	0x80
9	01101111	0x6F	10010000	0x90



each led at 7 seg is Connected with a Pin & to Display the nums use the table according to your Connection.

Anode or Cathode



Page:..... Date:.....

note

to Connecte the 7 segment with the MC just takes 8 Pins for one 7segment so if I heed to use 6 7segments in My Project so I heed 48 Pins for GPIO & this humbeh is hot available in PIC 18F 46k20.

what's the Solution?

Firstly you have to know the BCD

what's the BCD "Binary Coded Decimal it Presents the humbehs from 0 to 9 at just 4 bits

BCD Table

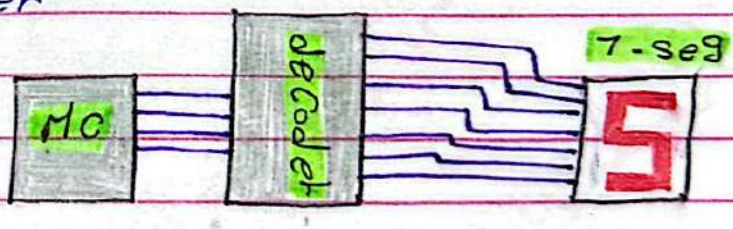
Decimal	Binary Pattehn				BCD
	8	4	2	1	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9

the Binary Pattehn is Consideh as four Pins of the MicroControlleh are Connected with the **decodeh** the Pins of otheh side of the decodeh are will be Connected with 7 segments

So that is Make the function of the BCD is Called the decodeh

note

the decodeh to Save the humbeh of Pins



Page:..... Date:.....

as you see at the data sheet of the DeCoder 7447 & 7448 to the function table :-

LT & BI/RBo :- ake High at all Cases

RBI :- is high at 0 decimal only & it's not important for the test Values so I will Make it high too.

(A, B, C, D) :- will be Connected with the Pins of the Micro Controller

Function Table

46A, 47A

Decimal or Function	Inputs						BI/RBO (Note 6)	Outputs							Note
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	(Note 7)
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	(Note 8)
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	(Note 9)
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	(Note 10)

H = High level, L = Low level, X = Don't Care

Note 6: BI/RBO is a wire-AND logic serving as blanking input (BI) and/or ripple-blanking output (RBO).

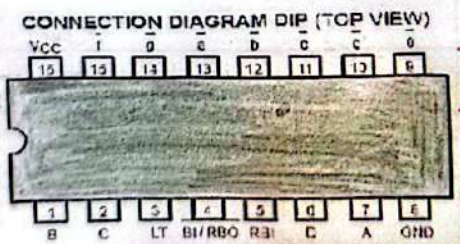
Note 7: The blanking input (BI) must be open or held at a high logic level when output functions 0 through 15 are desired. The ripple-blanking input (RBI) must be open or high if blanking of a decimal zero is not desired.

Note 8: When a low logic level is applied directly to the blanking input (BI), all segment outputs are high regardless of the level of any other input.

Note 9: When ripple-blanking input (RBI) and inputs A, B, C, and D are at a low level with the lamp test input high, all segment outputs go H and the ripple-blanking output (RBO) goes to a low level (response condition).

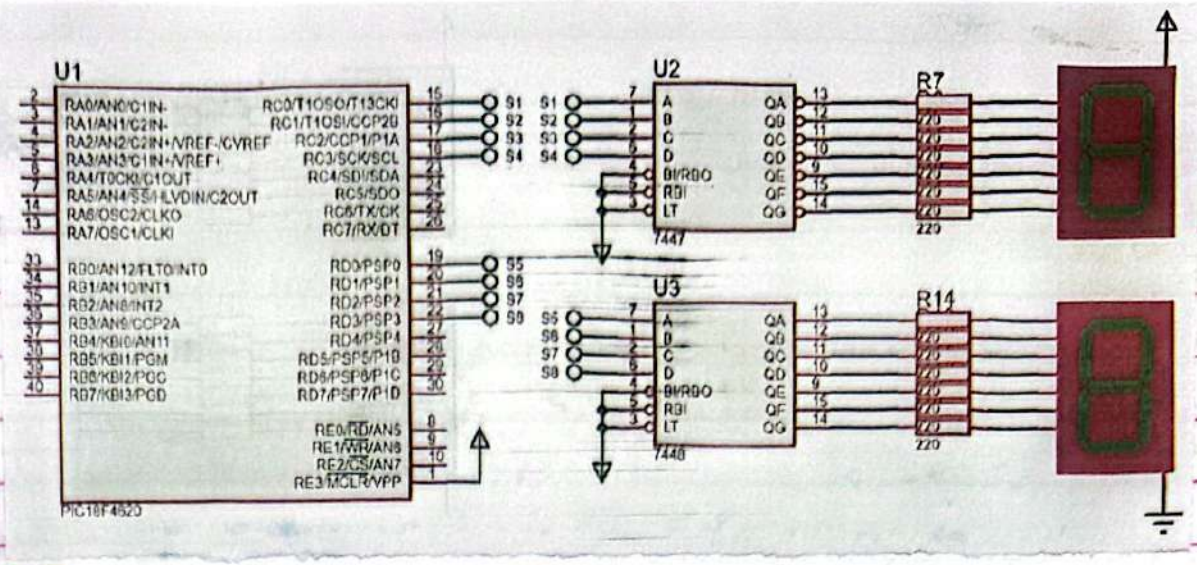
Note 10: When the blanking input/ripple-blanking output (BI/RBO) is open or held high and a low is applied to the lamp-test input, all segment outputs are L.

acordding to this table



you will Connecte the Pins of the decoder

Page:..... Date:.....

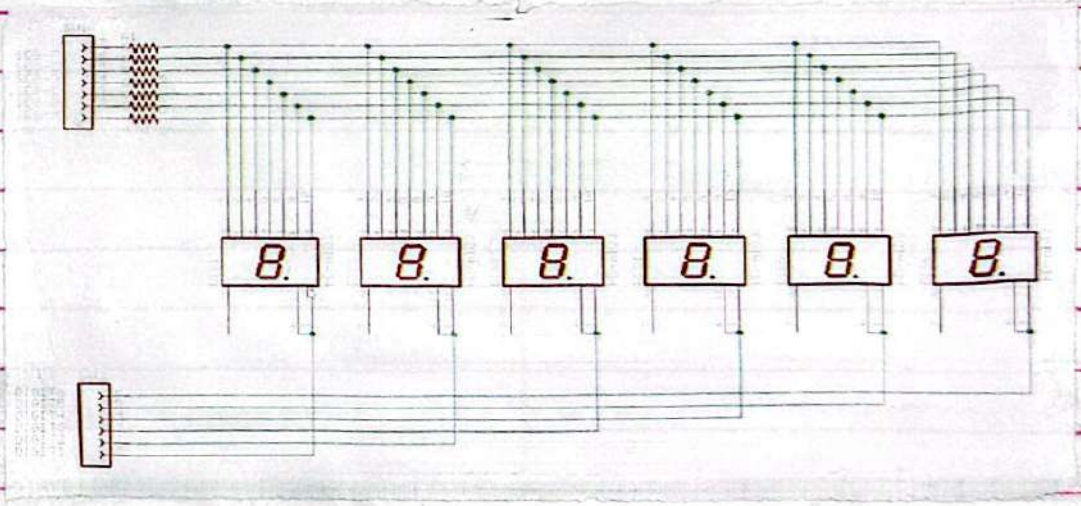


hote

7448 is Pot 7seg Commah Cathode.
7447 is Pot 7seg Commah Anode.

the MatiPlexing SeVeh Segment

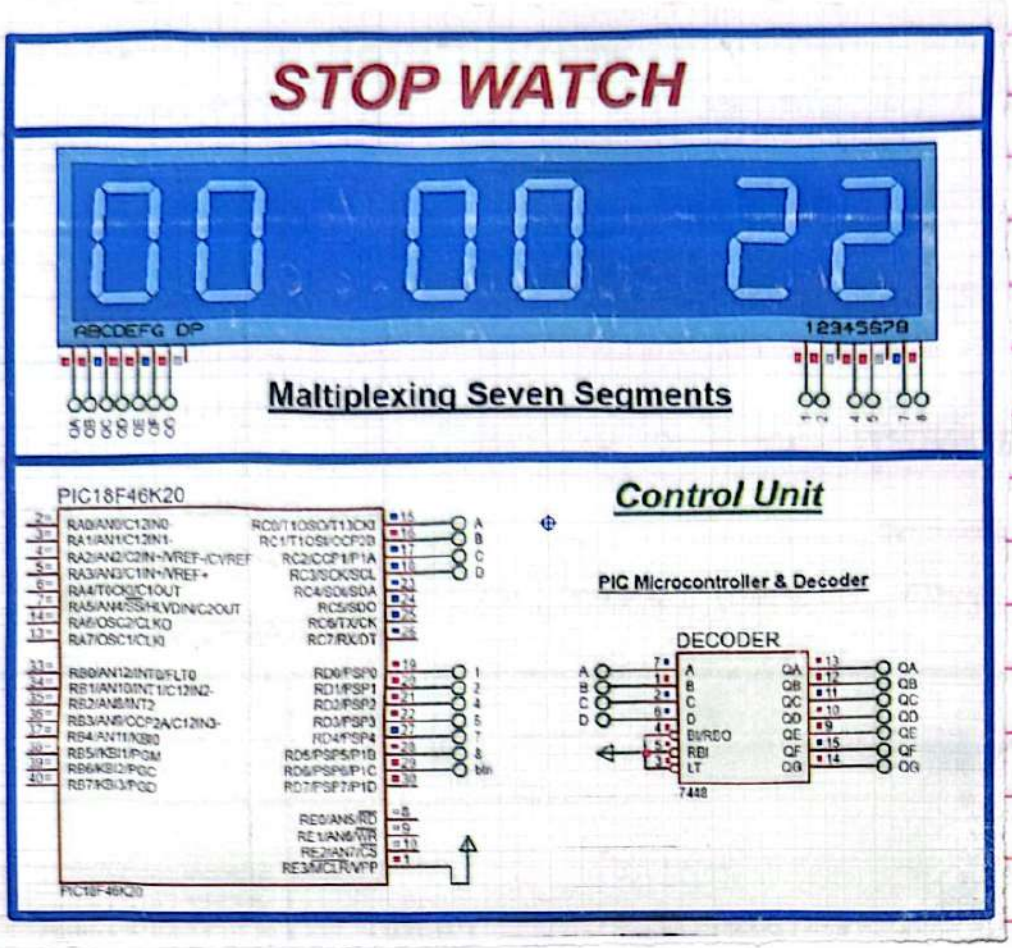
You will use some seVeh segments what eVeh it's humbet but you will Conthol with it with just two Ports in the MC Conttotlet Portto lights the LEDs of seVeh segment & anothe to disable & ehable the tatget seVeh segment & this Ptocess will dePend on the Petsistehce of Vision.



Page:..... Date:.....

note

You will connect the common of each seven segment with the Pins of the MC to control the enable & disable. as you see in this Project:



the interface file of seven segments

```
#define  seg_Pin0
#define  seg_Pin1
#define  seg_Pin2
#define  seg_Pin3
typedef enum {
    Comma_Anode,
    Common_Cathode
} 7seg_type_t;
```

0 these Macros to define the four Pin's for the 7seg connected with the decoder

this enum to Present the type of 7 seg

Page: Date:

typedef struct {
    Pin_Config_t segment_Pin[4];
    7seg_type_t segment_type;
} segment_t;

this struct to create with it an object in the APPlication file it has the 4 Pins of the Decoder with the seven segment & the type of the used seven segment.

the APIs of 7 seg Module

Std_ReturnType 7seg_initalize(
    Const segment_t * seg
); function to initalization

Std_ReturnType 7seg_write_number(
    Const segment_t * seg,
    uint8 number
); function to write a number on 7 seg

---

note

= there's a way to show the number of two digits on two seven segment as you see the next expaining

uint8 number = 23;

((uint8)(number / 10)) = 2 → 0010

((uint8)(number % 10)) = 3 → 0011 ↓ Connected at the decoder

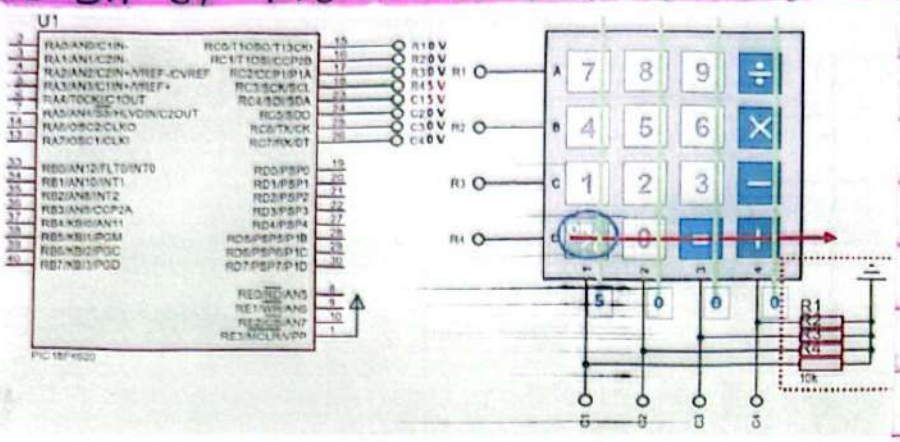
these equation when they will be Passed will Present its numbers by hex so what ever you write any number will be on 7seg.

Page:..... Date:.....

keyPad interfacing Matrix keyPad

May be is 4 x 4 4 rows x 4 Columens & is Consider as a Collection of Push buttons are Connected by specific way.

the design of the Matrix key Pad 4 x 4



hote
the busses of rows & Columns will be Connected with the Micro Controller.

* Make the rows work as outPut.
* Make the Columens work as InPut.
* you will active the row & check the Columeh & see which button is Pressed

hote
keyPad is Consider as Active a row & check the Columens.

the interface file of the keyPad

```
#define  kPD.Rows      4
#define  kPD.Columns   4
```

↳ to set the number of Pins for the rows & the Columns.

```
typedef struct {
    Pin_Config_t  kPD_Row_Pins [kPD.Rows];
    Pin_Config_t  kPD_Column_Pins [kPD.Columns];
} keyPad_t  → to create the object
```

Page:..... Date:.....

The APIs Functions For key Pad

```
Std_ReturnType Kpd_Initialize(
    Const keyPad_t * kPd
);
Std_ReturnType Kpd_Get_Value(
    Const keyPad_t * kPd,
    uint8 * value
);
```

Note

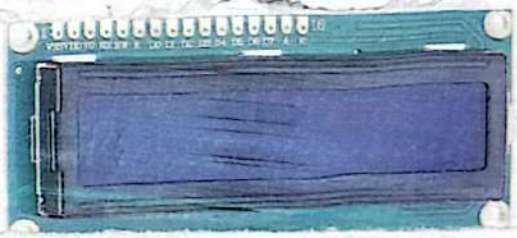
= the idea of the KPD depends on activing one row & disable others & see what's the Column will gives 5V to the MC

LCD interfacing

* is used to show the characters that you write it on the LCD or send for it
* Most of character LCDs have "16 Pins"
* it Can display all the letters of alphabet
* it Can display the Custom characters

the Pins of the LCD

* there are Pins for the Power & Ground & Pins for the Control & Pins to recive the Data on the LCD



6. Interface pin description

Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7-10	DB0-DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11-14	DB4-DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

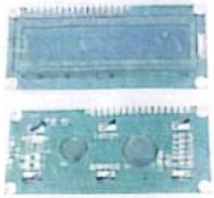
Page:..... Date:.....

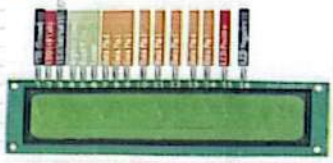
PIC18F4620 GPIO (Character LCD Interfacing)

16*2 Character LCD Interfacing with PIC Microcontroller in 8-bit Mode → Pins

✓ There are in total of **14 pins** marked with numbers (16 if there is a backlight)

FUNCTION	PIN NUMBER	NAME	LOGIC STATE	DESCRIPTION
Ground	1	V_{ss}		0V
Power supply	2	V_{dd}		+5V
Contrast	3	V_{ee}		0 - V_{dd}
Control of operating	4	RS	0 1	D0 - D7 are interpreted as commands D0 - D7 are interpreted as data
	5	R/W	0 1	Write data (from controller to LCD) Read data (from LCD to controller)
	6	E	0 1 From 1 to 0	Data Operation (Read - Write) Enable signal
Data / commands	7	D0	0/1	Bit 0 LSB
	8	D1	0/1	Bit 1
	9	D2	0/1	Bit 2
	10	D3	0/1	Bit 3
	11	D4	0/1	Bit 4
	12	D5	0/1	Bit 5
	13	D6	0/1	Bit 6
	14	D7	0/1	Bit 7 MSB

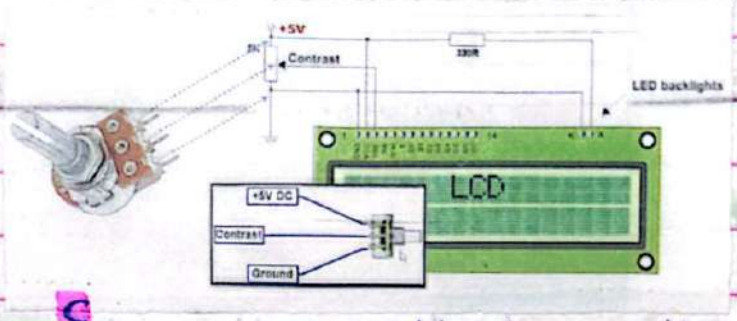




V_{ss} : is connected with the ground

V_{DD} : is with the Powet "5V"

V_o : is connected with a Potientmeter to Conthol in the lighting of th LCD



Pin 15 & 16 : to Conthol in the brightness of the LCD Contholed by the PWM

note

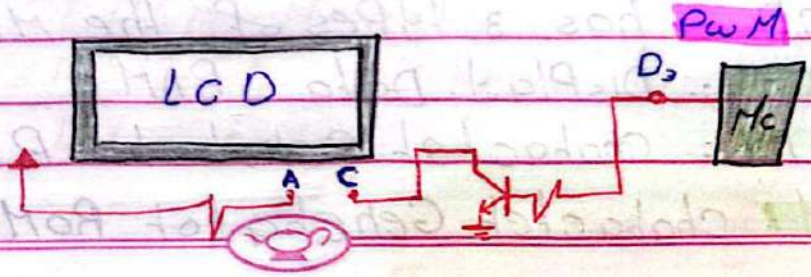
if you don't want to Conthol in the brightness you will Connecte

Anode with "5V"

Cathode with Ground

note

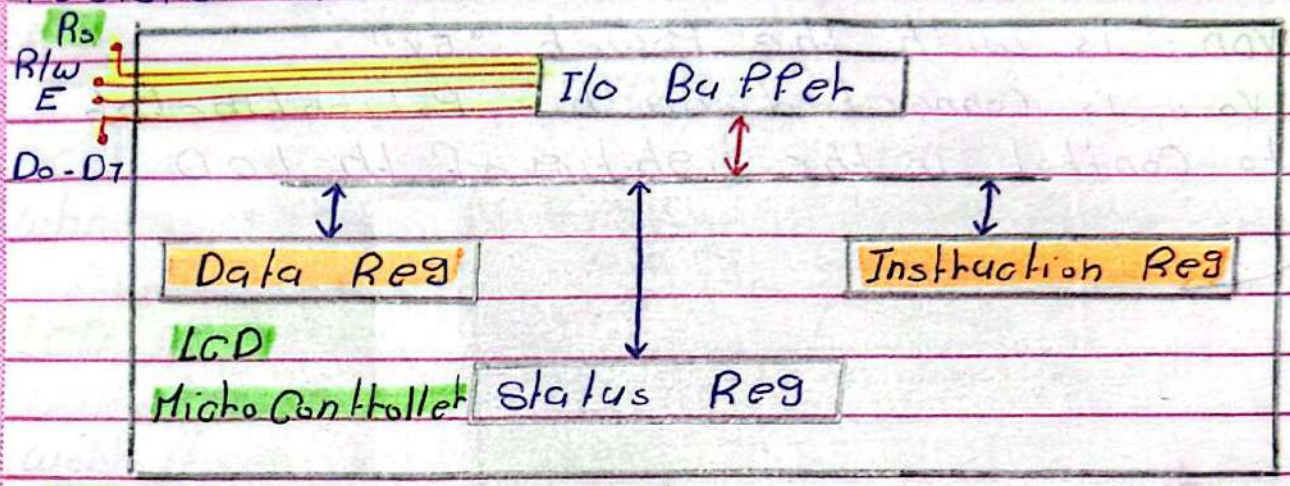
if you want to Conthol in the brightness



Page:..... Date:.....

Pin 4 : Register Select "Rs"

the Pins from D0 to D7 are used to send Data or send Command the Data will be stored at Data Register & the Command will be stored at Instruction Register & those registers are existed at the MC of LCD the thing that limits what is coming through Data Pins goes to the Data register or Instruction register is "Rs"



Rs Pin will connected with MC if the out Put of it

0 → this means Command the Data will be at the Instruction register

1 → this means Data Display the Data will be at the Data register. according to the Data sheet of the LCD

note

HD44780 this the Micro Controller of the LCD :. has 3 types of the Memory

DDRAM :. Display Data RAM

CGRAM :. Character Generator RAM → special character

CGROM : character Generator ROM

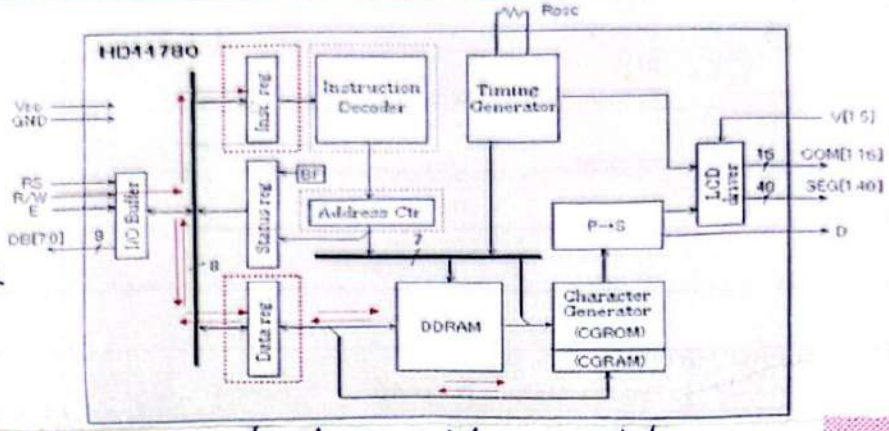
Page: Date:

R/w Pin : is for if you want to Read or write Data & its Value stored at the status Register R/w = 0 while R/w = 1 Read
Follow the block diagram of the Micro Controller of the LCD

Content :
1) Command
2) Address (DDRAM or CGRAM)

* the instruction Decoder Processes the instruction Code written into the instruction register

#define CGRAM 0x40
#define DDRAM 0x80



note the Address Ctr : it has the Address of DDRAM & CGRAM if you want to access these Memories according to the Command that is stored at the Instruction Register

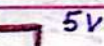
note From the Data sheet of the LCD
the start Address of CGRAM 0x40
the start Address of DDRAM 0x80

note
the Address Counter will be incremented when you write specific data in the DDRAM & CGRAM Memory.
&
the Address Counter will be decremented when you Read specific data from the DDRAM & CGRAM Memory
Follow the next table to know the role of Rs Pin with R/w Pin

Page: Date:

Rs	R/w	operation
0 → IR	0 → w	→ Instruction register writes as internal operation (display - clear - ect). Command
0 → IR	1 → R	→ Read bus flag DB7 & Address Counter (DB0 to DB6)
1 → DR	0 → w	→ Data register writes as internal operation to the DDRAM & GCRAM.
1 → DR	1 → R	→ Data register Reads as internal operation from the DDRAM & GCRAM.

E Pin : Enable signal

is falling Edge : high to low Pulse 

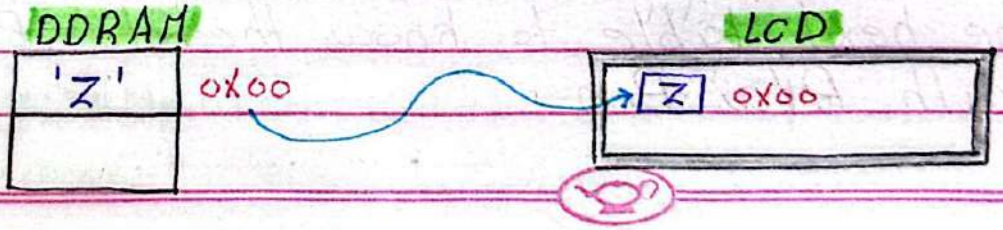
it's responsible for transfering the Data to the Instruction register or the Data register

the Memories at the MC of LCD

DDRAM

→ the size is 80 * 8 bits this means can store 80 character. 80 bytes

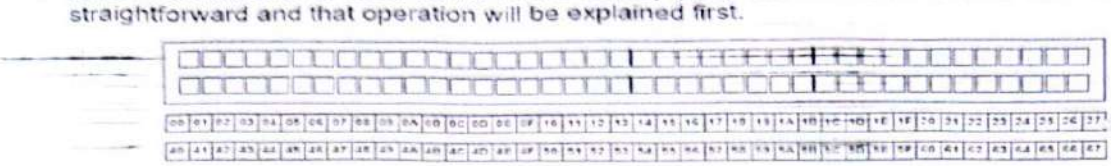
→ the Data that will be shown at the LCD will be stored in the DDRAM.



Page:..... Date:.....

DDRAM [Display Data] RAM → LCD DDRAM Addressing

- ✓ The HD44780 type controller chip is used with a wide variety of Liquid Crystal Displays.
- ✓ These LCDs come in many configurations each with between 8 and 80 viewable characters arranged in 1, 2, or 4 rows.
- ✓ The problem is that **there is no way to inform the controller of the configuration of the display that it is driving**.
- ✓ The controller operates exactly the same way for all displays and it is up to the programmer of the device that is controlling the LCD controller (usually a host microcontroller) to deal with this situation.
- ✓ When the controller is used with a 40 x 2 display (forty characters on each of two rows) the operation is quite straightforward and that operation will be explained first.



Display data RAM (DDRAM) 80 × 8 bits

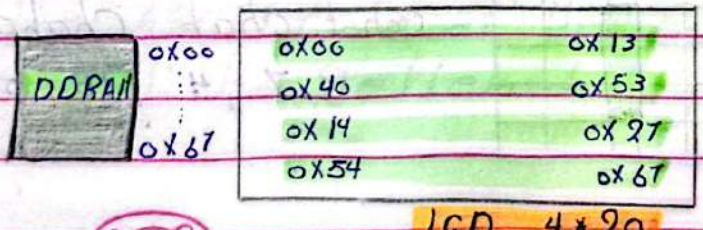
* For example 40×2 LCD where 2 is the number of rows & 40 of the number of Columns.
* the first Address of the first row is 0x00 & the end Address of the first row is 0x27.
* the first Address of the second row is 0x40 & the end Address of the second row is 0x67.
* there's a gape between the first row & the second row 0x27 ..... 0x40.
* the Data that are stored in DDRAM will be shown on the LCD according to the specific location Address in the DDRAM will be shown at the same Address at the LCD.

note

= you can use another LCD like 4×20 LCD where 4 : number of rows & 20 number of Column the size of DDRAM is fixed 80 * 8 bits but the difference will be at the start & end Address of rows as you see

note

= the lines Address of DDRAM is 2



So you will have with any LCD by 2 rows

Page: Date:

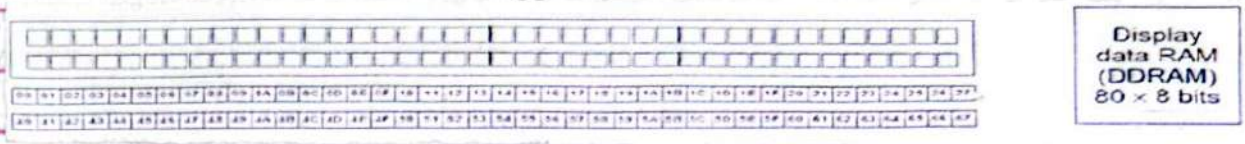
2 x 40 LCD

left Side			Right side
0x00	0x13	0x14	0x27
0x40	0x53	0x54	0x67

4 x 20 LCD

0x00	0x13
0x40	0x53
0x14	0x27
0x54	0x67

as you see the left side as it was & the right side will be under the left side & this the dividing 4x20 LCD



This is the DDRAM memory in the MC of the LCD is fixed with its Addresses.

how I will work on 2x20 LCD

2: rows & 20 Column

0x00	0x13
0x40	0x53

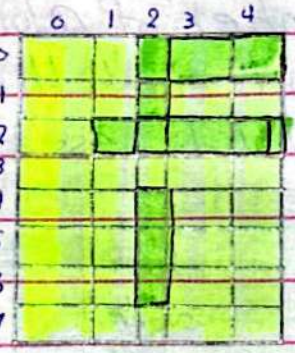
→ This the 2x20 LCD

the same thing for the 2x16 LCD

0x00	0x0F
0x40	0x4F

Note

= you have options to set the one dot of the LCD to work as 5x8 or 5x11 you control with that by the Data sheet.



this "i" character will be Presented as an Array like that

```
const char Character[] = {
    7, 4, 15, 0, 4, 4, 4, 0 };
```

Page: Date:

CGRoM

has the standat Patteth of the English Chatacter so it underStands the ASCII Code of Chatacter that is existed in English language you don't heed to cheate for the English Chatachter a hew Patteth of an Attay fotit.

How the CGRoM wotks

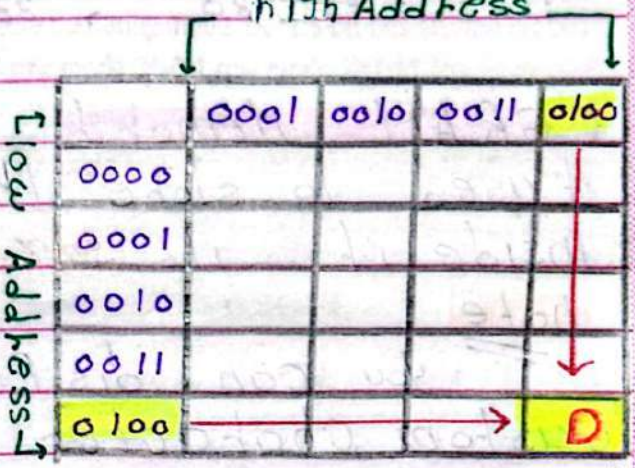
hote

= the ASCII Code of any Chatacter is Consider as it's Address at the CGRoM Memoty

* the Design of the CGRoM Memoty is devide-ing the Address into low Address & high Addtes

ex: ASCII Code of 'D' is 44 → high 4 4 low

this is the high Address ← 0100 0100 ↓ this is the low Address



CGRoM Memoty

it will access the low Address & the high Address & sees what is Patteth between these Addresses.

hote

= the CGRAM doesn't underStand the humbet but gets the ASCII Code of the humbet & the ASCII Code will be getteth by the Address

Page:..... Date:.....

ASCII	hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3

if you want to show "0" you will send 30 or 48 so you can make this API

```
char ConVert_num_to_ASCII (char input){
    return input + 0x30 ;
}
```

note the ASCII & hex is arranged Conterously in the CGRAM

```
ConVert_num_to_ASCII (3);
    ↳ 3 + 0x30 ⟶ 33 ⟶ 3 Symbol
```

CGRAM Memory

it used to store the custom character inside it its **size 64 bytes**

note

you can display only 8 specialor Custom character in the CGRAM why

```
uint8 character[] = {7,4,15,0,4,4,4,0};
                    1B 1B 1B 1B 1B 1B 1B 1B
                    ↳ = 8 bytes
```

and the total size is 64 bytes so

$$\text{char} \times \frac{8 \text{ bytes}}{8 \text{ bytes}} = \frac{64 \text{ bytes}}{8 \text{ bytes}}$$

∴ char = 8

Page: Date:

the Commands of the LCD

Instruction	Opcode b7	b6	b5	b4	b3	b2	b1	b0	Description
Clear Display	0	0	0	0	0	0	0	1	Fills DDRAM with 0x20 and set DDRAM address 00h to the address counter. ↳ ASCII of space
Cursor Home	0	0	0	0	0	0	1	*	Sets DDRAM address 00h to the address counter. *:Don't care
Entry Mode Set	0	0	0	0	0	1	I/D	S	Sets the direction of address counter and specifies display shift (updating display offset register) on data read/write. I/D=1:Increment, S=1:With display shift
Display ON/OFF	0	0	0	0	1	D	C	B	Sets display, under-line cursor and block cursor on/off. D=1:Display ON, C=1:Under-line cursor ON, B=1:Block cursor ON
Move cursor and Shift display	0	0	0	1	S/C	R/L	*	*	Increment/decremet address counter and display offset register. S/C=1:Shift display, S/C=0:Move cursor, R/L=1:Right shift, R/L=0:Left shift
Function Set	0	0	1	DL	N	F	*	*	Configure operating mode. DL=1:8-bit bus, DL=0:4-bit bus N=1:2-row mode, N=0:1-row mode, F=1:11-line mode, F=0:8-line mode
dress Set (CGRAM)	0	1	Address(00h..3Fh)						Sets CGRAM address to the address counter. After this instruction, CGRAM is accessed via data register
ress Set (DDRAM)	1	Address(00h..67h)							Sets DDRAM address to the address counter. After this instruction, DDRAM is accessed via data register

Clear Display: write space at all the Addresses of the DDRAM & will start from the 0x00 Address so the LCD will be cleared.

Cursor home: **Note** * : it's not important it goes to the home location at the DDRAM row number ① & Column number ①.

Entry Mode Set:

I/D: Increment / decrement **S:** shift
↳ ① ↳ ⓪ ↳ 1 : shift
↳ 0 : no shift

I/D	S	
0	0	→ decrement shift off
0	1	→ decrement shift on

Page: Date:

I/D	S	
1	0	→ Inchement shift off
1	1	→ Inchement shift on

hole

the Command is for the Initialization of the LCD Inchement :- means when I write the Address Counter will be Inchemented Dechement :- means when I write the Address Counter will be dechemented. this for at the DDRAM Memory & Make the shift is enable to don't Make an over write on the stored Data inside the DDRAM.

Move Cursor & shift display

the Data is shown from the right or left But firstly you have to enable the shift at the entry Mode set Command

S/C :- shift Display / Cursor Move.

R/L :- Right shift / left shift.

S/C	R/L	
0	0	→ Cursor Move / left shift.
0	1	→ Cursor Move / right shift.
1	0	→ shift Display / left shift.
1	1	→ shift Display / right shift.

Display on/off

D :- Display on / Display off.

C :- under line Cursor on/off.

B :- Block Cursor on/off.

this Command just display on/off the data that are existed at the DDRAM but doesn't clear it.

Page:..... Date:.....

Function set Instruction

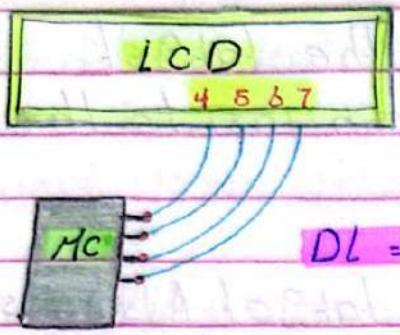
is used for the Initialization of the LCD

DL :: Data length

↳ limits if we work 4 or 8 bit Mode

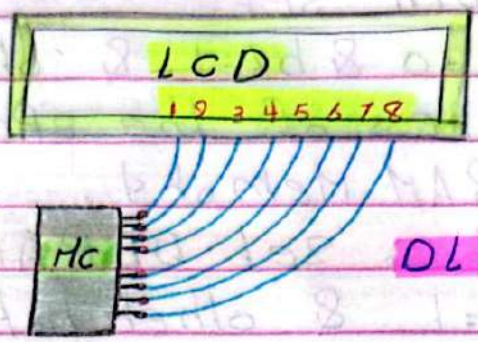
↳ I send the Data by 4 Pins of Data or 8 Pins of Data.

4 bit Mode



* here when you send a character you will send it during 2 times
Char → 8 bits
* takes 4 Pins but is slow for sending the data to LCD

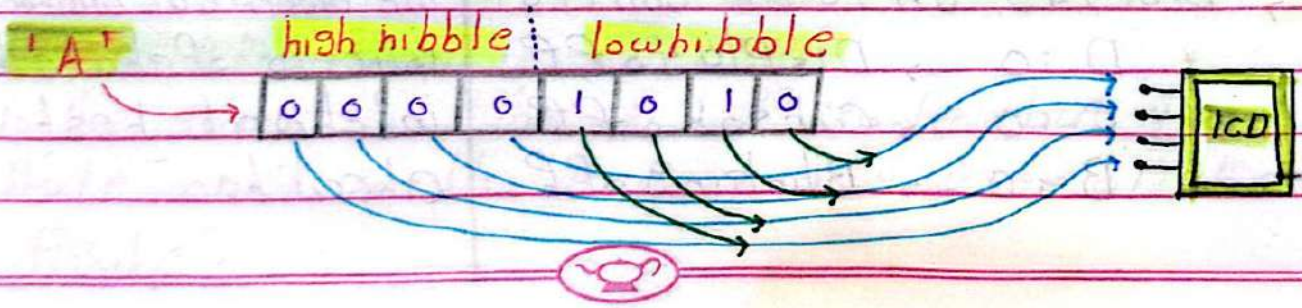
8 bit Mode



* here when you send a character you will send it though just one time Char → 8 bits
* takes 8 Pins opposite 4 bits Mode. takes 4 Pis but it is faster than 4 bits Mode

note

= if you work on 4 bit Mode to send the Data you will send the high nibble firstly after that you will send the lower.



Page: Date:

N: limits the LCD is one line or two lines
F: Character Font
5*8 Dot Matrix F=0
5*11 Dot Matrix F=1

Address Set CGRAM

If you want to access the CGRAM Memory start Address is 00h end Address is 3Fh & you will set
b7:0 & b6:1 & other bits has the target Address of the specific location inside the CGRAM Memory.

Address Set DDRAM

b7:1 & other bits has the target Address.

the Initialization of LCD

*there's two ways to Make that :
→ by internal test Circuit. not recommended
→ Sending set of the commands.
↳ here you can Make what you want in the LCD but the internal test Circuit governs you by fixed Initialization like that

→ Display clear
→ Function set
* DL:1 → 8 bit Mode
* N:0 → 1 line
* F:0 → 5*8 dot
→ Display on/off Control
* D:0 → Display off
* C:0 → Cursor off
* B:0 → Blinking off
→ Entry Mode set
* I/D:1 → Increment
* S:0 → no shift

note
You Must Follow the electrical characteristic to operate the internal test Circuit.

Page: Date:

Note at the Internal test circuit there's a flag inside HD 44780 Mcof LCD is Called Busy flag "BF" if the Instruction Decoder execute an Instruction this flag will has "1" & you can't send another Instruction till the last one will be executed & the executing time is 10ms so this way not recomended.

How to initialize LCD by Commands

8 Bit - Interface

→ Give the Power to LCD

→ wait 15ms delay

→ Send this Command

→ wait 4.1ms delay

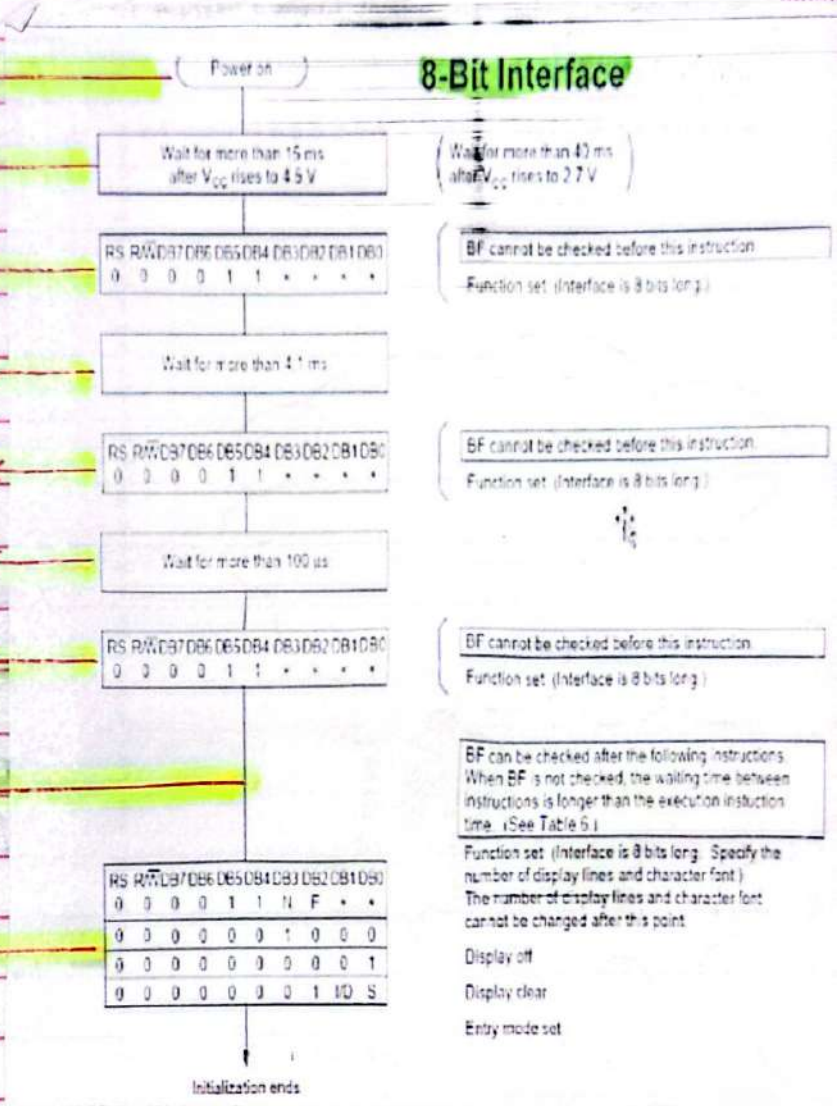
→ Send this Command

→ wait 100µs delay

→ Send this Command

→ here 8 bit Mode is Active

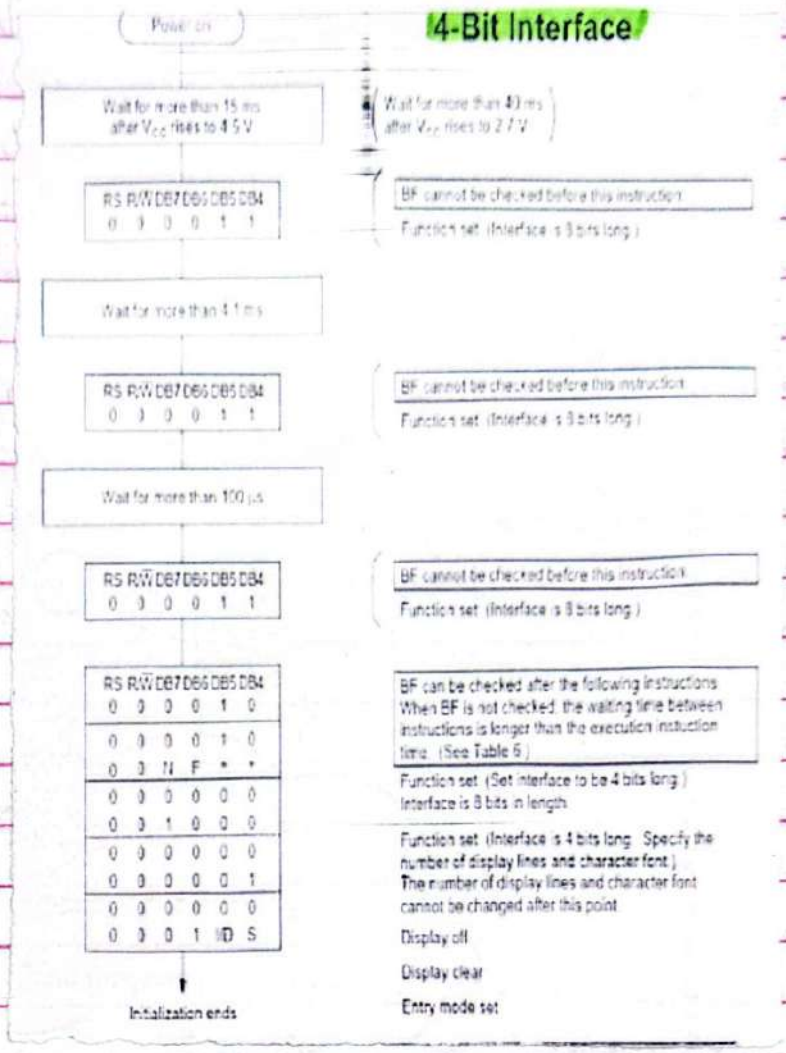
→ extra Configurations



4 - Bit - Interface

Is the same at the 8 bit Mode but you have only with 4 bits as you see at the figure.

Page:..... Date:.....



the Interface file of LCD

* Firstly you will make #define for all the command that are existed at the Data sheet of the LCD.

typedef struct { → this struct if you will
PinConfig_t LCD_Rs; work on 4 bit Mode
PinConfig_t LCD_E;
PinConfig_t LCD_Data[4];
} Char_LCD_4bits_t;

Page: Date:

typedef struct { → this struct if you
Pin_Config_t LCD_RS ; work on 8 bit
Pin_Config_t LCD_E ; set Mode
Pin_Config_t LCD_Data[8] ;
} chat_LCD_8bits_t;

the APIs

note
the APIs are common for the 4 & 8 bits Mode Interface LCD.

* Std_ReturnType LCD_Initialize(
Const chat_LCD_4or8_bits_t * lcd
);

* Std_ReturnType LCD_send_Command(
Const chat_LCD_4or8_bits_t * lcd,
uint8 Command
);

* Std_ReturnType LCD_send_Data(
Const chat_LCD_4or8_bits_t * lcd,
uint8 Data
);

* Std_ReturnType LCD_send_Data_Position(
Const chat_LCD_4or8_bits_t * lcd
uint8 row,
uint8 Columh,
uint8 Data,
);

* Std_ReturnType LCD_send_string(
Const chat_LCD_4or8_bits_t * lcd,
uint8 * str
);

Page: ..... Date: .....

* std_ReturnType LCD_send_string_Position(
    Const char_lcd_4or8 bits_t * lcd,
    uint8 row,
    uint8 column,
    uint8 *str,
);

* std_ReturnType LCD_send_custom_char(
    Const char_lcd_4or8 bits_t * lcd,
    uint8 row,
    uint8 column,
    Const uint8 _chr[],
    uint8 Pattern
);

The Interrupt Feature

is an event generated by the hardware that requires the CPU to stop the normal Program & Perform some service code related to the event.

Event Types **exceptions**

→ **Asynchronous event :**

is hardware interrupts : is Raised by a hardware device Maybe is:

→ internal to the MCU

→ external to the MCU

Why is Called Asynchronous ? because the CPU doesn't know the time of the interrupt event is generated at random times with respect to the CPU clock signal.

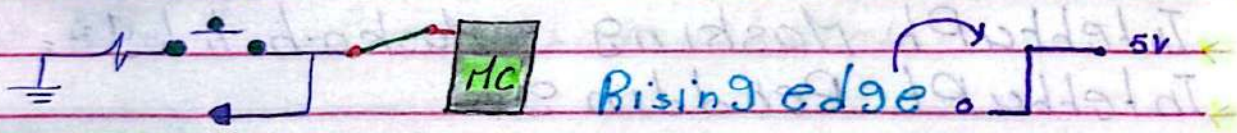
Page: ..... Date: .....

→ Synchronous event
is an exception :: Produced by the CPU Control unit while executing instructions the CPU Makes interrupt according to illegal instruction was been executed so this interrupt is made

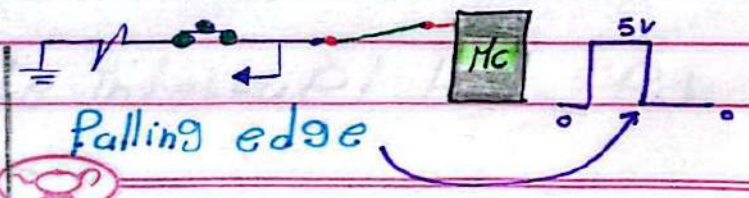
why is Called Synchronous?
because the CPU when it will Make the interrupt is according to the illegel instruction.

the types of synchronous event
Faults . Traps . Aborts

Asynchronous event "hard wake interrupt"
Internal to the MCU :
→ the EUSART Module received anew byte of data.
→ the Timer Module Complate the required time.
→ the ADC Module finishes the ADC Conversion
→ the SPI Module received / transmitted a Complate byte
→ the I2C Module received / transmitted a Complate byte
External to the MCU :
→ the "INTx" Pins : Rising / Falling edge detection



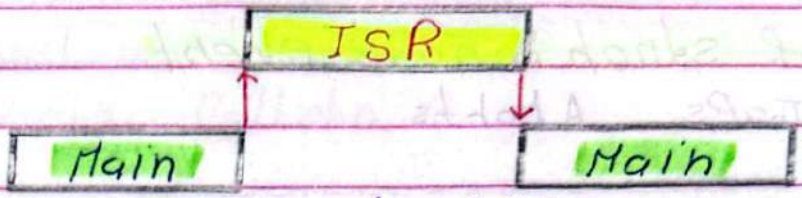
→ the "on change" Pins :
Interrupt on ever voltage change on these Pins



Page: ..... Date: .....

the Interrupt :.

the Processor execute the code at the Main function suddenly there's a flag in a Register has ① to Interrupt the Processor will stop the Main function & goes to the Interrupt service routine function ISR to execute its task after that the Processor will goes one more time to the Main to complete the rest of code.



Synchronous events types

→ Faults :. the saved value in the register is the Address of the instruction in your code that caused the fault. & Fault

→ Traps :. the values of the Address of the instuction in your code that caused the fault * Fault

→ Aborts :. is a failat by the hard ware error by the Hw makes the Micro Controller to make Aborts.

what's the difference between the :.

→ Interrupt Masking "Maskability".

→ Interrupt Penddings.

Page:..... Date:.....

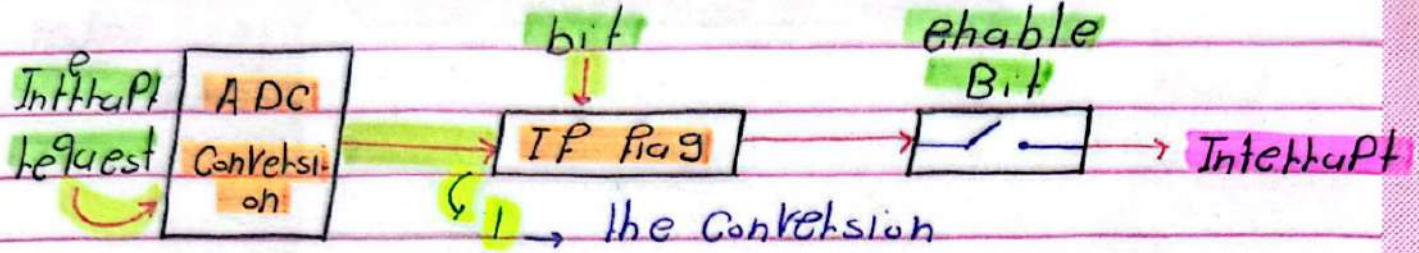
* Maskable interrupt

→ this kind of the interrupt can be Enable or disable

→ there's a specific bit at some Register thought you configer if you want to enable the Interrupt or disable the Interrupt

ex :

↳ Most of the MCU Peripherals "Timer - ADC - UART"



↳ 1 → the Conversion done

↳ 0 → the Conversion is not done yet

hote

= when the flag bit has ① this means the Conversion Process is done but the Processor will not Make an Interrupt till you enable the bit enable  enable bit Interrupt

* non - Maskable interrupt

→ this kind of the interrupt Must be enabled the enable bit is not exasited

ex :

↳ the Reset of Micro Controller

---

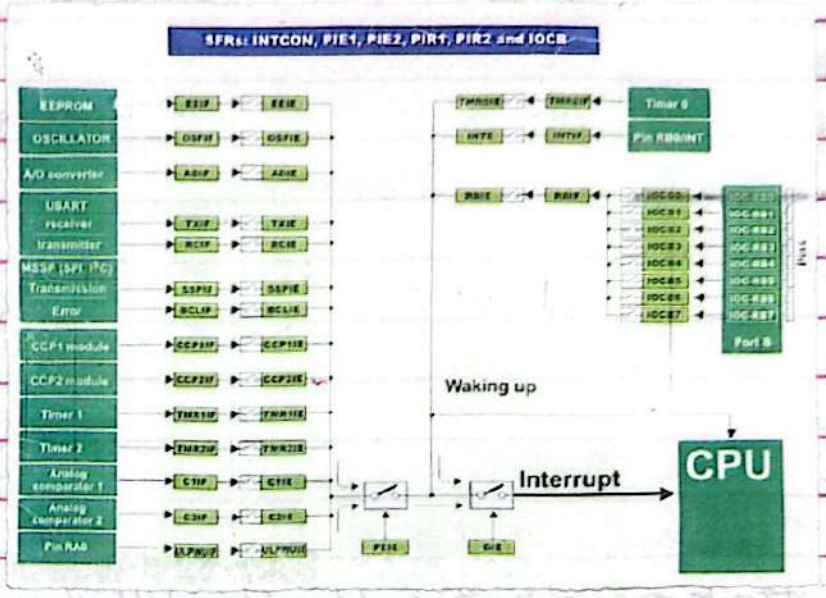
hote

= there are enable Interrupt bits for

Page: Date:

all the Modules inside the MicroController & there's an enable Interrupt bits for the Peripherals Modules over all & there's an enable Interrupt bit for the general Interrupts that are existed at the MicroControlleer

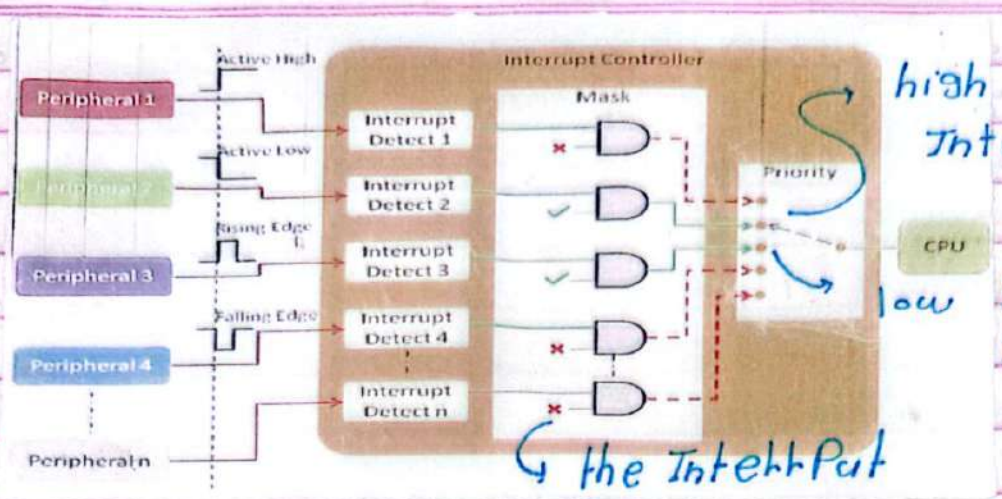
PEIE bit : Peripheral Interrupt enable.
GIE bit : General Interrupt enable.



hole:
as you see at this figure each Module has Interrupt bit enable if you want to Make the Interrupt for any Module you Must enable it's Interrupt bit after that you Mast enable PEIE bit after that GIE bit to teach the Interrupt to the Processor "CPU".

what's the Interrupt Pending
is an Interrupt is happen but is not be excuted because there's another Interrupt has high Priority about it.

Page:..... Date:.....



hote

= the low Priority Interrupt its called the Pending Interrupt.

---

the Software Interrupt

you can trigger "إستثناء" exceptions or Interrupt using the Software Code.

what's ISR

Interrupt service routine is a function the Processor execute it in the case of Interrupt by the hardware "Asynchronous"

what's ESR

Exception service routine is a function the Processor execute it in the case of Interrupt of "synchronous type".

hote

= it's Preferred for the code of ISR / ESR function to be few lines.

hote

= the first Address of ISR / ESR function Must be known. the start Address its called **Vector**

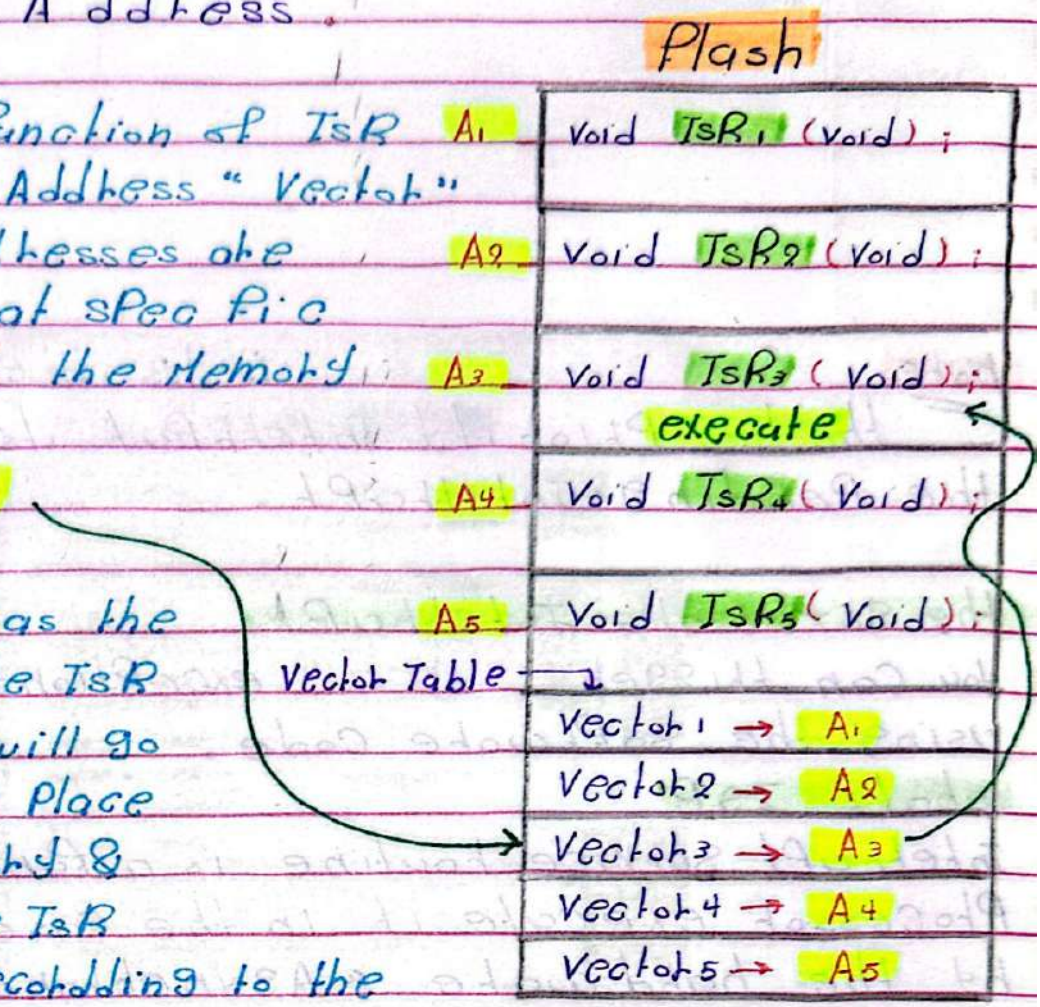
Page: Date:

how you have 5 functions are stored at the Memory of ISR each function has an start Address.

* each function of ISR has start Address "Vector" & these Addresses are allocated at specific location at the Memory

CPU

the CPU has the Vector of the ISR function it will go to the Vector Place at the Memory & execute the ISR function according to the target Vector.



what will happen when the Interrupt Case

- suspend Main function
- save the CPU state Value at the stack
- execute the Interrupt function
- Restor the CPU state Value from the stack
- resume or Compelete the Main function

those 5 Main steps at the case of the Interrupt.

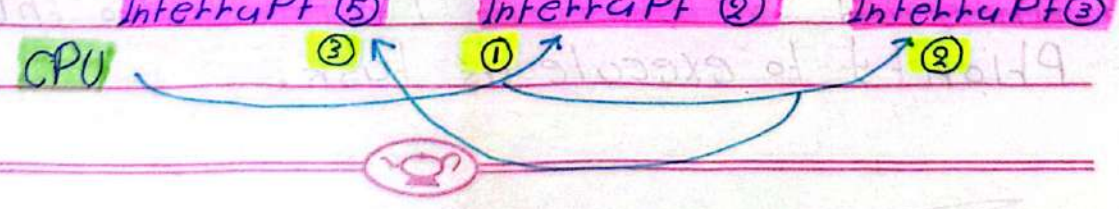
Page:..... Date:.....

what's the difference between the

Polling	Interrupt
the CPU doesn't Make any thing else checking on the indecators of the events are happened or not this Process is done by Periodic way the CPU just checks each x of time on the flag of the Interrupt & doesn't Make any thing else that.	the CPU execute the functions of Main in the case of the Interrupt will break the Main & go to the ISR or ESR to execute it's function at the case of the Interrupt.
the Advantage: efficient if the event are rapidly "high rate".	
the Disadvantage: CPU takes time even when no Pending request.	check each 1 sec  if the flag has the Interrupt task case or not case of Polling

what's the Interrupt Priority

is a number given to the task of ISR function Interrupt if I have some Interrupts at the MC & there are 2 Interrupts are done at the same time whose Interrupts the CPU will execute it's function firstly the high Priority will be executed firstly



Page:..... Date:.....

hole

= There are some Peripherals are fixed Priority & Peripherals you can set its Priority at your code or the Run time.

the types of Priority :

→ Configured Priority.
→ Run time Priority.
→ Fixed Priority. by the hardware

let's know some Questions

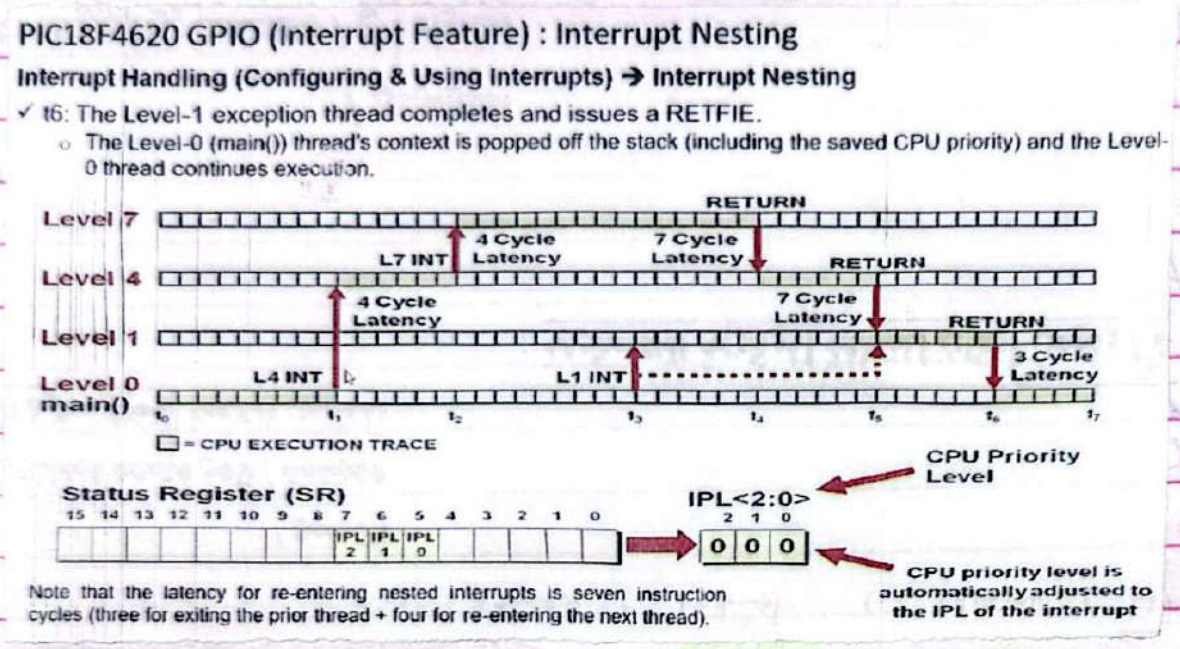
*what will happen if the CPU was executing low Priority Interrupt & suddenly there an high Priority Interrupt has been happen ?

as CPU at nested Interrupt schedule Mode I will stop the executing of the low Priority Interrupt & go to the high Priority Interrupt to execute its function after that I will go back one more time to the low Priorit Interrupt to resum the rest of function after that I will go to the Main function.

as CPU at non nested Interrupt schedule Mode I will Complate the low Priority task till be finished after that I will go to the high Priority to execute its task.

Page: ..... Date: .....

look at this figure to see the sequence of the nested Interrupt by levels



what's the Interrupt vector table IVT

→ is a section at the flash Memory is like the kind of an array its elements is increased or decressed according to the type of the Micro controller & these elements are consider the Addresses of ISR functions.

what's the Content of IVT table

1- vector number
2- Program Address
3- Source : INT - Timer ... ect
4- Interrupt definition

Page:..... Date:.....

hote

= the writing of ISR function different from a compiler to another each compiler has away is specific to write the ISR function.

hote

= the number of the vectors inside the Interrupt vector table differences from Micro controller to another for ex:.

ATmega32 MC support 42 different Interrupt sources Assume address size is 2 bytes for each vector so the size of IVT = 42 × 2 = 84 bytes so the IVT at ATmega32 MC will take 84 bytes from the flash Memory.

hote

= IVT determines the Priority level of the Interrupt.

hote

= the lowest Address at IVT is high Priority

IVT

Vector No	Program Address	Source	Interrupt Definition
1	0x0000	Reset	Reset
2	0x0002	INT0	external Interrupt 0
3	0x0004	INT1	external Interrupt 1

* Reset > INT0 > INT1 Priority.

↓ lowest Address

↓ highest Address

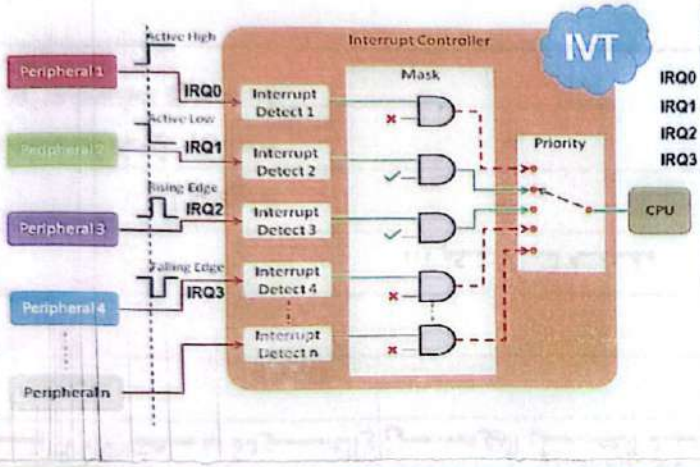
Page: ..... Date: .....

The Interrupt Controller

→ it determines if you made enable or disable to the Interrupt of the Peripheral.

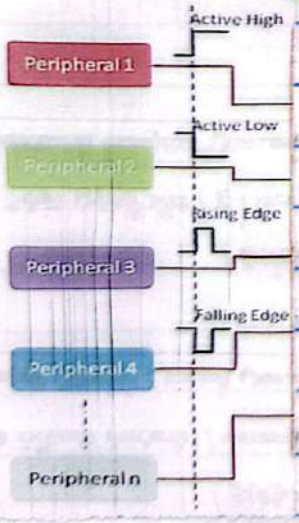
→ it has some Interrupt request lines are connected with the Peripherals to receive the signal of the Interrupt.

→ it converts the signal at the Interrupt request line into Vector no to go to the Vector table & get the Address of the target ISR function.

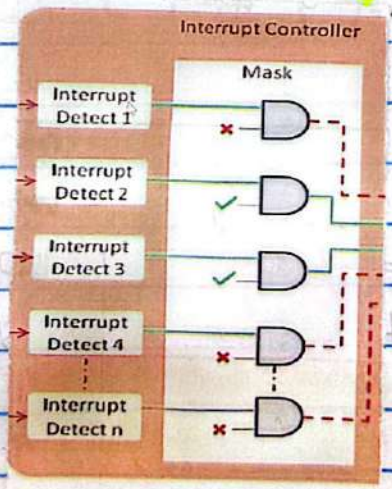


	Vector No	Program Address	Source
	1	0x0000	RESET
IRQ0	2	0x0002	INT0
IRQ1	3	0x0004	INT1
IRQ2	4	0x0006	INT2
IRQ3	5	0x0008	INT3
	6	0x000A	Reserved
	7	0x000C	Reserved
	8	0x000E	INT6
	9	0x0010	Reserved
	10	0x0012	PCINT0
	11	0x0014	USB General
	12	0x0016	USB Endpoint
	13	0x0018	WDT
	14	0x001A	Reserved
	15	0x001C	Reserved

Interrupt Controller

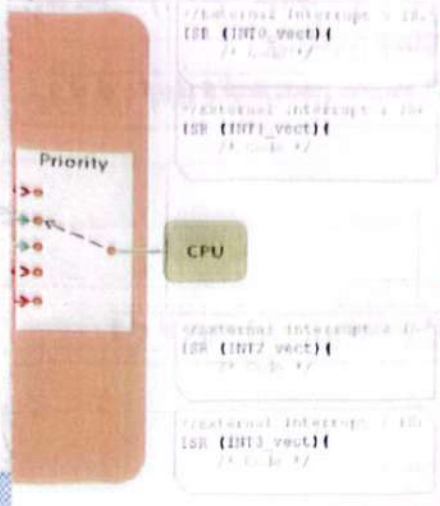


These are the Interrupt request lines are the Inputs of the Interrupt Controller has the signal of the Interrupt Peripherals.



this is the Maskability checking checks which Interrupt Peripheral is enable & disable.

Page: ..... Date: .....



→ the Peripherals that are enable Interrupt will go to the Priority section to the CPU can execute the high Priority task after that will execute the low Priority task.

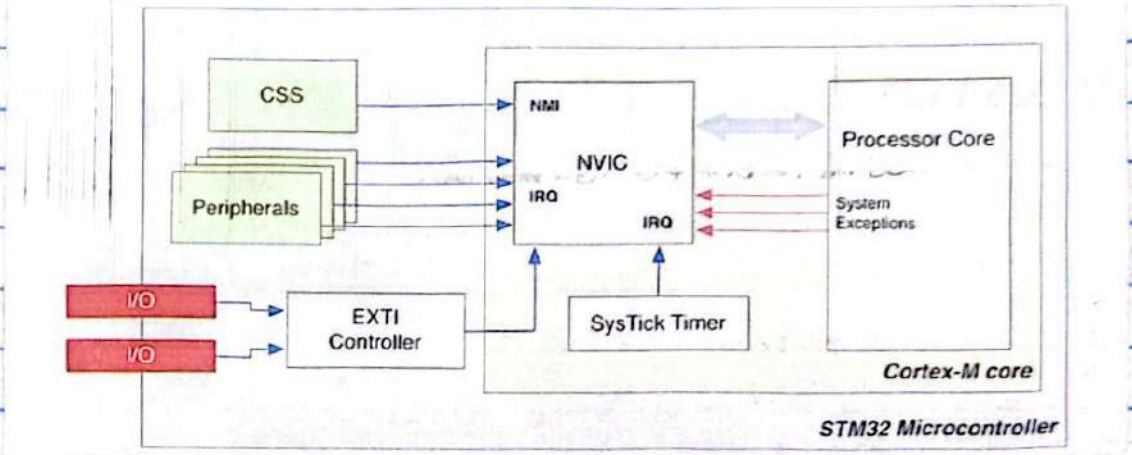
Interrupt handling & IVT in STM32F4 MC based on ARM Processor :

PIC18F4620 GPIO (Interrupt Feature) : General illustration

What is Interrupt Vector Table (IVT) ? → Interrupt Handling & IVT in STM32F4 (CortexM4 Processor)

NVIC : Nested Vector Interrupt Controller

- A **hardware unit** responsible of the exceptions handling.



NVIC : it's an Interrupt Controller it's Called Nested Vector Interrupt Controller.

* it supports the nested Interrupts.

IRQ : are the Interrupt request lines are Connected with « Peripherals - external Interrupt Controller - SysTicTimer clock » & System exceptions

Page: ..... Date: .....

NMI : non Masking Interrupt the Interrupt gate must be enabled.

CSS : Clock security system : if there are some issues of the MC this clock will generate some Pulses to the NVIC IC.

I/O : are the Pins of the MicroControllet if there are external events & give their signal to the external Interrupt Controllet then the NVIC IC.

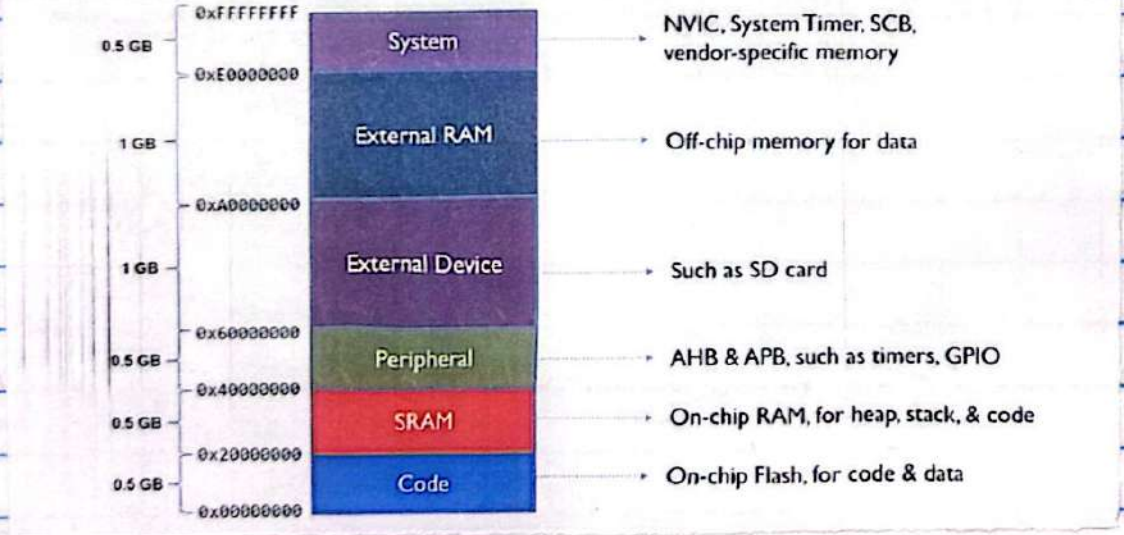
System exceptions : are the sychronous Interrupts by the CPU go to the NVIC IC.

Sys Tick Timer : the clock of the MC system.

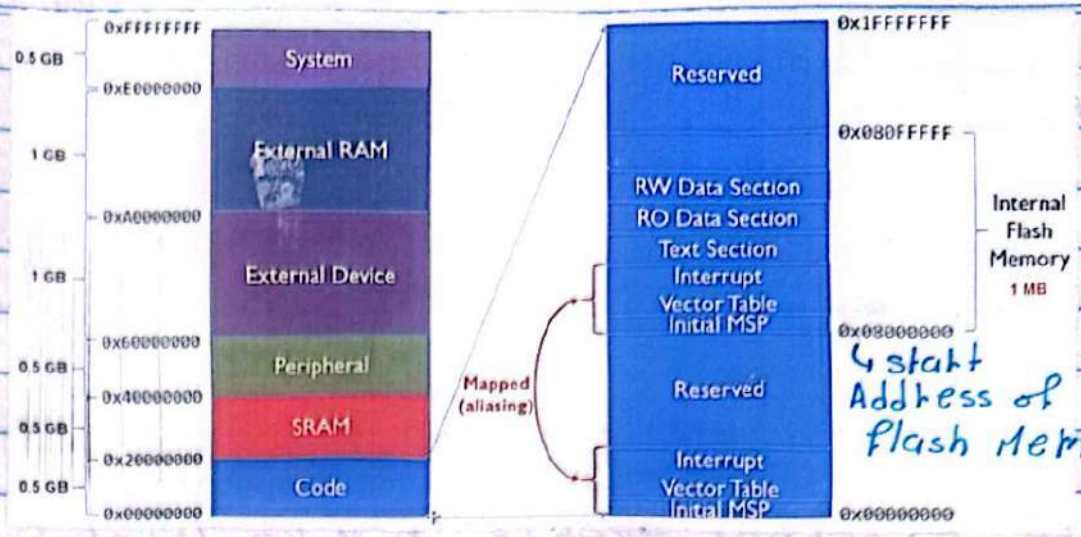
this is the Memory MaP of STM32F4 MC we Interested in the Code segment because it has the IVT

PIC18F4620 GPIO (Interrupt Feature) : General illustration

What is Interrupt Vector Table (IVT) ? → Interrupt Handling & IVT in STM32F4 (CortexM4 Processor)



Page: ARM Memory Date:



hote
= the Code segment is existed at the less Position at the Memory & the IVT will be at the less Position at the Code segment.

hote
= any Processor based on Cortex M4 when you restart the MicroControlleR goes to the 0x0 4 Address inside the IVT to execute the reset handeler function

what's the Memory aliasing :.
the Processor goes to an specific Address at the less Position at Code segment to access it's Content after that will jump to this Content Address at the flash Part.

let's see the vector table of Cortex M4 MicroControlleR based on the ARM Processor

Page: ..... Date: .....

Cortex-M4 Devices

Generic User Guide

Exception number[a]	IRQ number[a]	Exception type	Priority	Vector address or offset[b]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	HardFault	-1	0x0000000C	-
4	-12	MemManage	Configurable[c]	0x00000010	Synchronous
5	-11	BusFault	Configurable[c]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	UsageFault	Configurable[c]	0x00000018	Synchronous
7-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable[c]	0x0000002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable[c]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[c]	0x0000003C	Asynchronous
16	0	Interrupt (IRQ)	Configurable[d]	0x00000040 +	Asynchronous

hote

as software you will create the Interrupt Vector table as a kind of an Array of Function Pointer.

hote

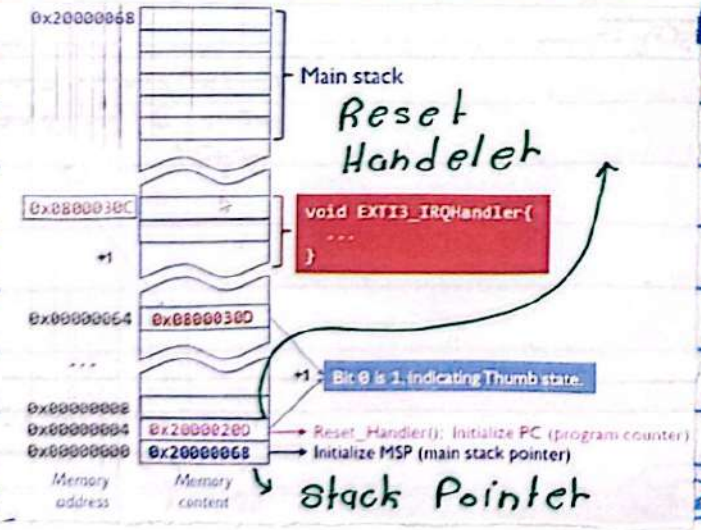
the Vector table it will be at the startup file code & it Consider as Array of Function Pointer.

How the Processor jumps to the target Address from the IVT to the this Position at the Flash Memory ... by this equation :

Vector Address = 64 + 4 * IRQ

(IRQ → Interrupt request number)

↳ because the size of the Address location is 4 bytes



hote

Stack Pointer is first element & Reset Handelet function is the second element at IVT

Page:..... Date:.....

* What's the role of the Reset Handler Function :

1- set the stack Pointer.
2- Copy the data segment initialization from flash to SRAM.
3- Init .bss.
4- Call the clock system initialization function.
5- go to Main.

Note
= the stack Pointer is a Register has the Address of the last data element added to the stack.

Note
= the Reset Handler function : "start up Code"
* is the first piece of software to execute after a system reset
* is used for setting up Configuration data for "C" start up Code :

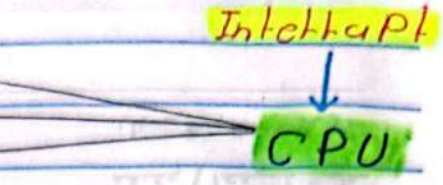
Ex :-
Address range for stack & heap Memories
* it's Address is existed at the second location of the IVT after the stack Pointer location.

Non - Vector Interrupt system :
this means that I don't have any Interrupt Vector table.
here the Question
How the CPU will go to the ISR ?
at this case I have inside the MicroController

Page: ..... Date: .....

at least 3 interrupt vector :

1- reset vector.
2- High Priority Vector.
3- low Priority Vector.



Note
= at non Vector Interrupt system in the case of Interrupt the Processor will jump to one of these vectors.

Note
= the types of ISR function in the case of the non Vector Interrupt system are two function :

1- High Priority ISR.
2- low Priority ISR.

Note
= PIC18F2ok46 is non Vector Interrupt system.

How to write the ISR function High Priority & low Priority.

High Priority ISR

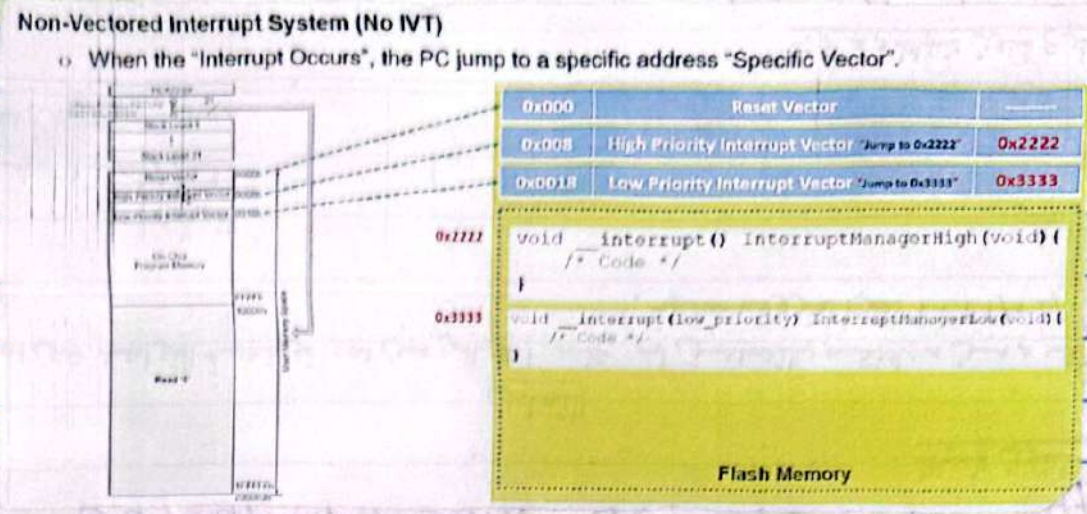
```
Void __interrupt() InterruptManager High(Void){
    /* Code */
}
```

low Priority ISR

```
Void __interrupt(low_Priority) InterruptManager
low(Void){
    /* Code */
}
```

Page:..... Date:.....

Non Vector Interrupt system with Flash Memory



These locations inside the Flash Memory

1. Reset Vector at 0x000 Address
2. High Priority Interrupt Vector at 0x008 Address
3. low Priority Interrupt Vector at 0x0018 Address

Note

→ the start Address of high Priority ISR is written at 0x008 Address is the Address of high Priority Interrupt Vector & the Processor when access the high Priority Interrupt Vector & finds the Address of high Priority ISR Address will jump to it's start Address to execute it.

→ the same thing at the low Priority ISR but it's start address will written at 0x0018 Address is the Address of low Priority Interrupt Vector.

How I Can Support Multiple ISR & I have just 2 Interrupt Vector at My MC

Page:..... Date:.....

any Interrupt at My Microcontroller has a flag at it's register when this flag has "1" this means it's ISR will be executed & when this flag has "0" this means the CPU will not execute it's ISR so you write the code as you see:

```
Void __interrupt() InterruptManager High (Void) {
    if (INT0_flag == 1) {          → for INT0 Interrupt
        INT0_service_code();
    } else {/*nothing*/}           High Priority.
    if (INT1_flag == 1) {          → for INT1 Interrupt
        INT1_service_code();
    } else {/*noting */}
    .....
}
```

note
the same thing for the low Priority ISR

the disadvantage of non IVT is very slow & large delay because the flags checking Interrupt & execute the code service of the Interrupt task.

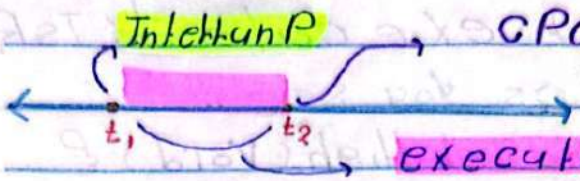
→ the Interrupt handling
what will happen when the Interrupt occurs & till be finished.

1- as CPU during the main function executing there's an exception occurred whatever is Synchronous or Asynchronous event.

2- the Interrupt was detected by the Processor.

Page: Date:

3- the Processor will execute the current instruction firstly after that will suspend the Main Program execution & jump to the ISR Interrupt Function.



4- before the Processor goes to the ISR firstly :.

* save the state register value of the Process of the Instruction this Process it's Called the Context switch.

Note

= the state register it's called the Program status word / register "PSW" / "PSR" & this value will be copied at the Shadow Register.

* save the Address of the next instuction that will be executed at the main function after executing the ISR this Address Called Return Address → Pushing to the stack section

* the Processor will disable the Interrupt global enable till execute the current ISR to don't allow to other Interrupt to be available to the Processor & this case at the Priority is disabled

Page:..... Date:.....

if the Pitority is enable .. * if the cattent executing Intettupt is high Pitority so the Processot will disable the othet high Pitority Intettupt till finsh the cattent high Pitority Intettupt

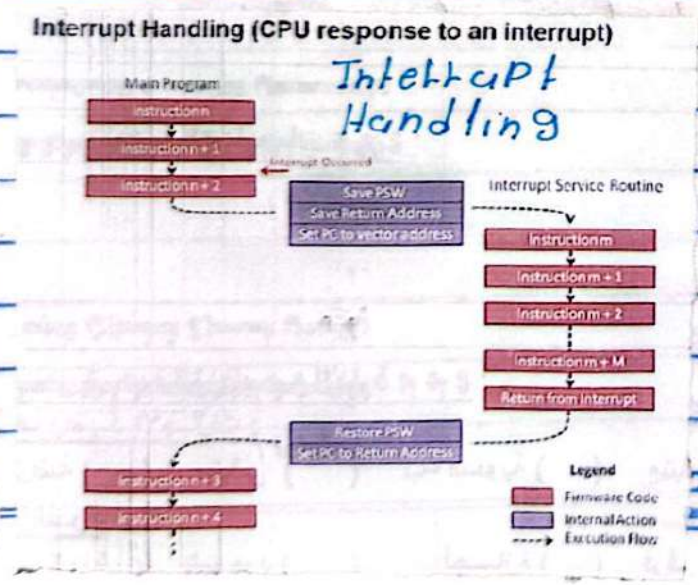
* if the cattent executing Intettupt is low Pitority Intettupt so the Processot will disable the othet low Pitority intettupts till finsh the cattent low Pitority Intettupt.

5- the Progtam Countet is set to the vectot address specific to this intettupt.

6- execute the ISR till teaching to the RETI "tetutn from intettupt instruction" & go to the main.

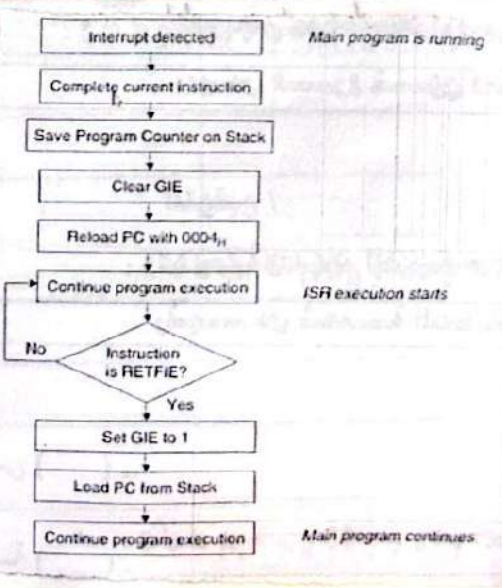
7- testot the Psw & tetutn Address to Compelete the Main

8- enable the globle Intettupt attet you disable it when you went the ISR



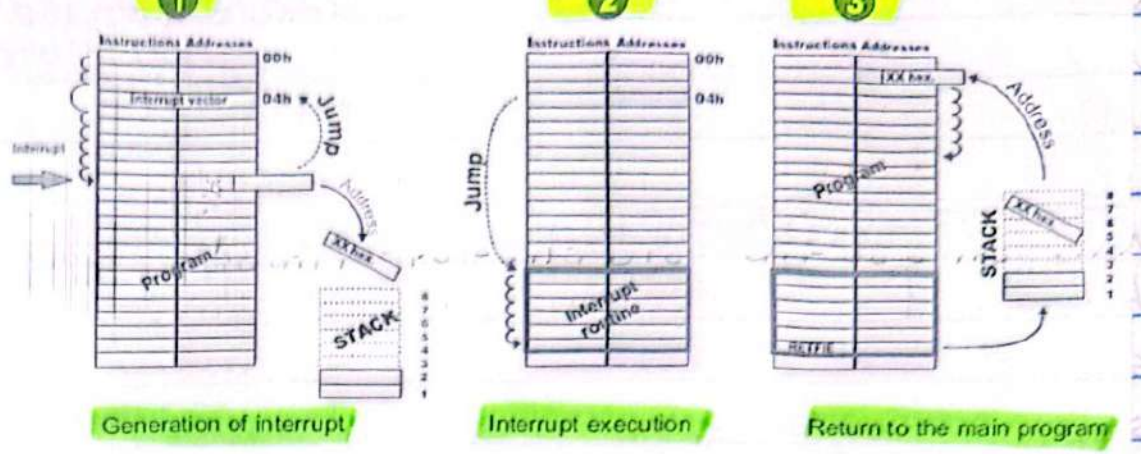
Intettupt Handling

this flow chatt to show the steps of the Intettupt Handling

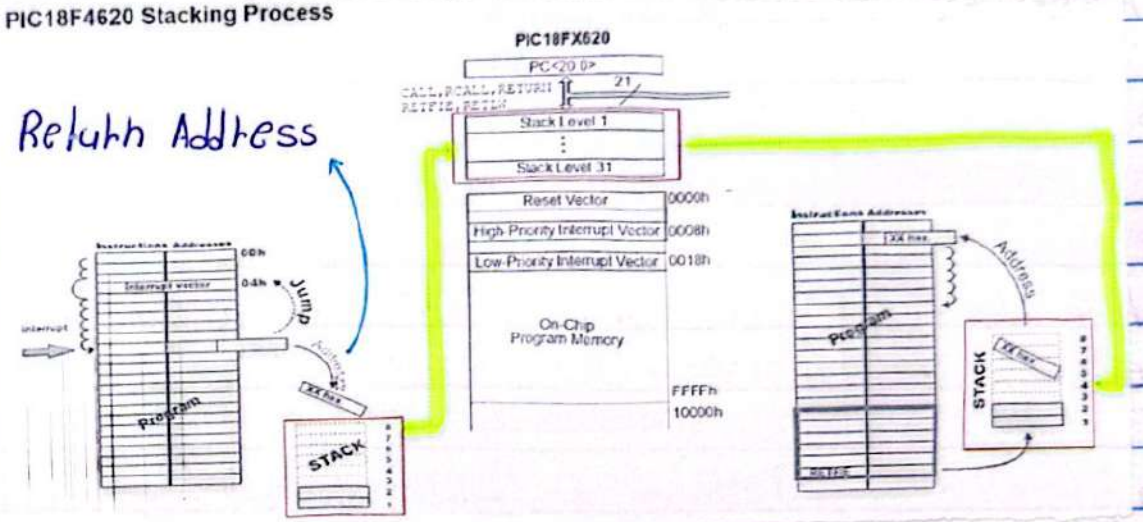


Page:..... Date:.....

another figure to show the steps



another figure with the Memory



what's the Interrupt latency or Interrupt response time :

is the delay of the start Interrupt request till the start ISR execution.

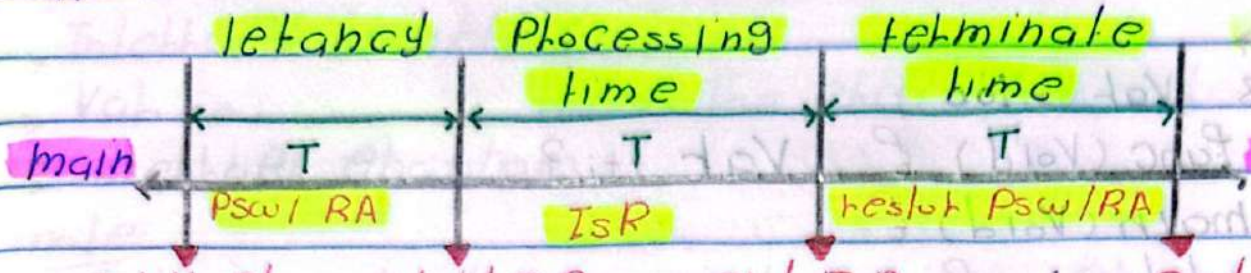
Interrupt Processing time :

the delay from the start of ISR execution to the end of ISR execution.

Page: ..... Date: .....

Interrupt terminate time

the delay from the end of ISR execution to the start continoue execution Instruction at the main function.



the shared data

for example I have a globle variable is commah at two different functions.

ex →

```
uint8 var;
void func1(void){ var++; }
void func2(void){ var--; }
```

the shared resource:

you have inside the Micro controller Memory or Register & there's many locations thy to access this Memory or register.

the shared Data & Race Condition:

what's the Race Condition occurs at the interrupt is a globle variable is commah at the main function or normal function and the ISR function during the case of executing this variable or making about an operation at the main function or normal function there's an Interrupt has

Page: ..... Date: .....

accured the Processing Variable operation is not Completed & the CPU will jump to the ISR function to make it's task about this Variable this will Make Race Condition

Ex →

```
uint8 Var = 100;
void Func(void) { Var ++; }
int main(void) {
    while(1) {
        if(Var == 200) { Func(); break; }
    }
    return 0;
}
```

there's an Interrupt has accured at this instruction

lets say Func(): takes 3 instructions to be executed
1) read Var
2) increment Var
3) store Var

the CPU will not increment Var & store Var and will jump to the ISR function & the value of Var lets say = 101 not stored

```
ISR() { Var--; }
```

the value will b 99 not 100 because there's the Race Condition

note
= it's not recommended to use Globle Variable at the ISR & normal functions

---

what's the Critical section
the shard data that I make about it operation at the normal & ISR function.

Page:..... Date:.....

How to Protect the Critical section

→ Make disable Interrupt before the critical section & Enable Interrupt after the critical section

→ Interrupt_disable();
→ Var--; ——→ this is the critical section
→ Interrupt_Enable()

Note

This way Maybe Make some Problems at your system if you have more than one Interrupt.

the Solution: Make the critical section is very simple.

→ you can Protect the critical section by another way by using the temp & the bit is responsible for the Interrupt Enable or disable

→ Global_Interrupt_status = INTCONbit.GIE;
→ Interrupt_disable() # the temp
→ Var--; → critical section way
→ INTCONbit.GIE = Global_Interrupt_Status

Note

the Problem of the critical section will beat the Projects dependen on RTOS.

Page:..... Date:.....

note at PIC 18F46k20 in general Interrupt sources have 3 bits to control:

1- Flag bit:
to indicate that an interrupt event occured or not.

2. Enable bit
if you want to enable an Specific interrupt write at this bit "1".

3- Priority bit
select from it if the interrupt is low interrupt or high interrupt

note
the flag bit is not cleared by itself you must Make that by using the soft wake

Interrupt → Flag bit [1] → if (flagbit == 1) { flagbit = 0 ; }

Enable the Priority feature by using Data sheet of PIC 18F46k20 MC

note knowing the bit of a Register
bit (Register <1>)
↳ the name ↳ the name ↳ the number of bit of register of bit

to enable the Priority feature

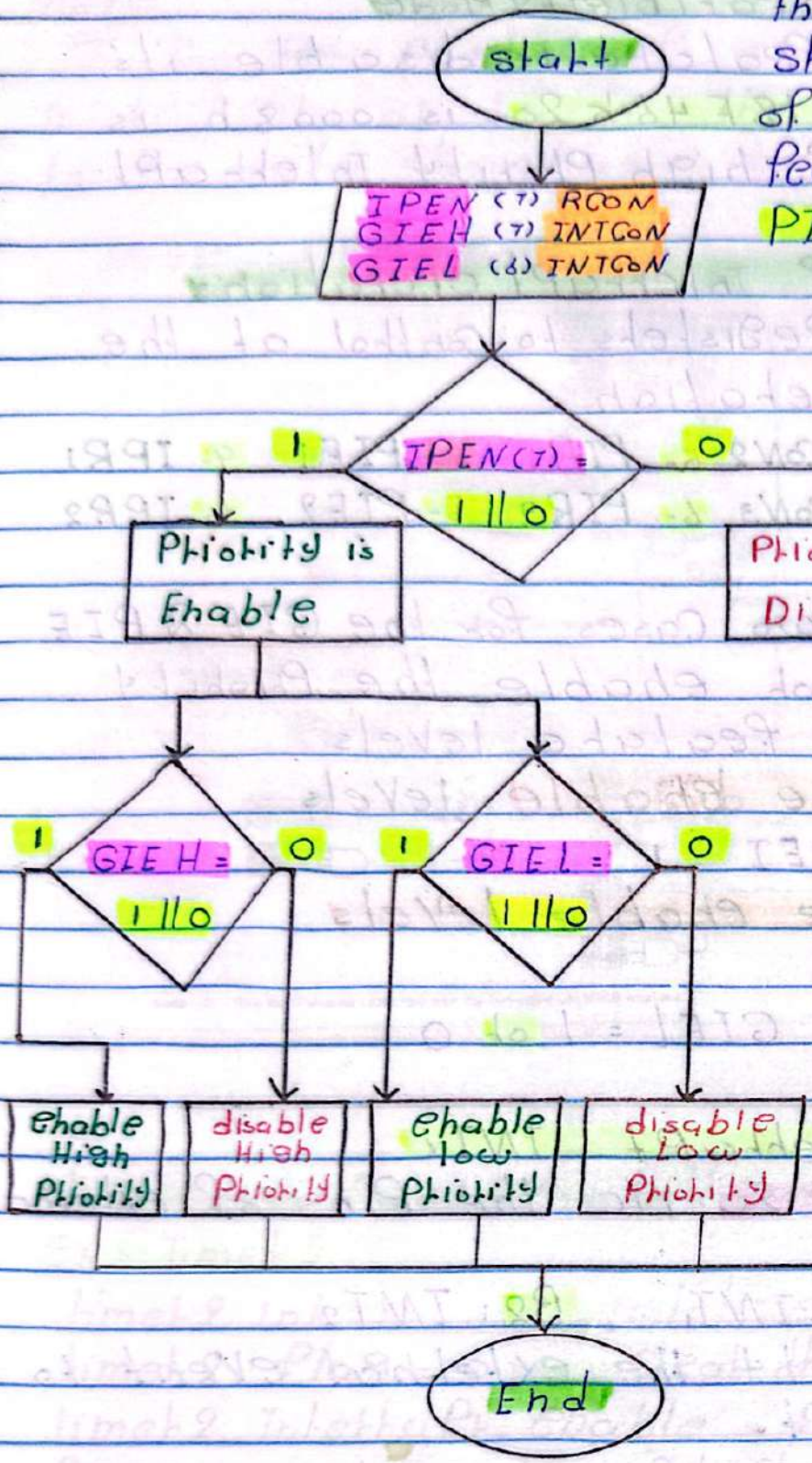
IPEN (RCON<7>) → '0' disable Priority
'1' enable Priority

GIEH (INTCon<7>)
↳ '0' disable high Priority
↳ '1' enable high Priority

GIEL (INTCon<6>)
↳ '0' disable low Priority
↳ '1' enable low Priority

Page:..... Date:.....

the flow chart to show the status of Priority Interrupt feature at MC PIC 18F46k20



Note = when you Make enable for an Interrupt clear its flag bit.

Page: Date:

what's the Compatibility Mode

the Interrupt feature is disable it's Address at PIC18F46k20 is 0008h is the Address of high Priority Interrupt vector.

The Registers of Interrupt operations

there are 10 Registers to control at the Interrupt operation.

1- RCON 3- INTCON2 5- PIR1 7- PIE1 9- IPR1
2- INTCON 4- INTCON3 6- PIR2 8- PIE2 10- IPR2

note

there are two cases for the GIE & PIE if you disable or enable the Priority feature levels

Priority feature disable levels
GIE = 1 & PEI = 1

Priority feature enable levels
PEI = 1
GIEH = 1 or 0 & GIEL = 1 or 0

---

the external Interrupt INTx

at PIC18F46k20 MC the Pins of external Interrupt are

B0 : INT0 , B1 : INT1 , B2 : INT2
these Pins work by the external event to do the Interrupt.

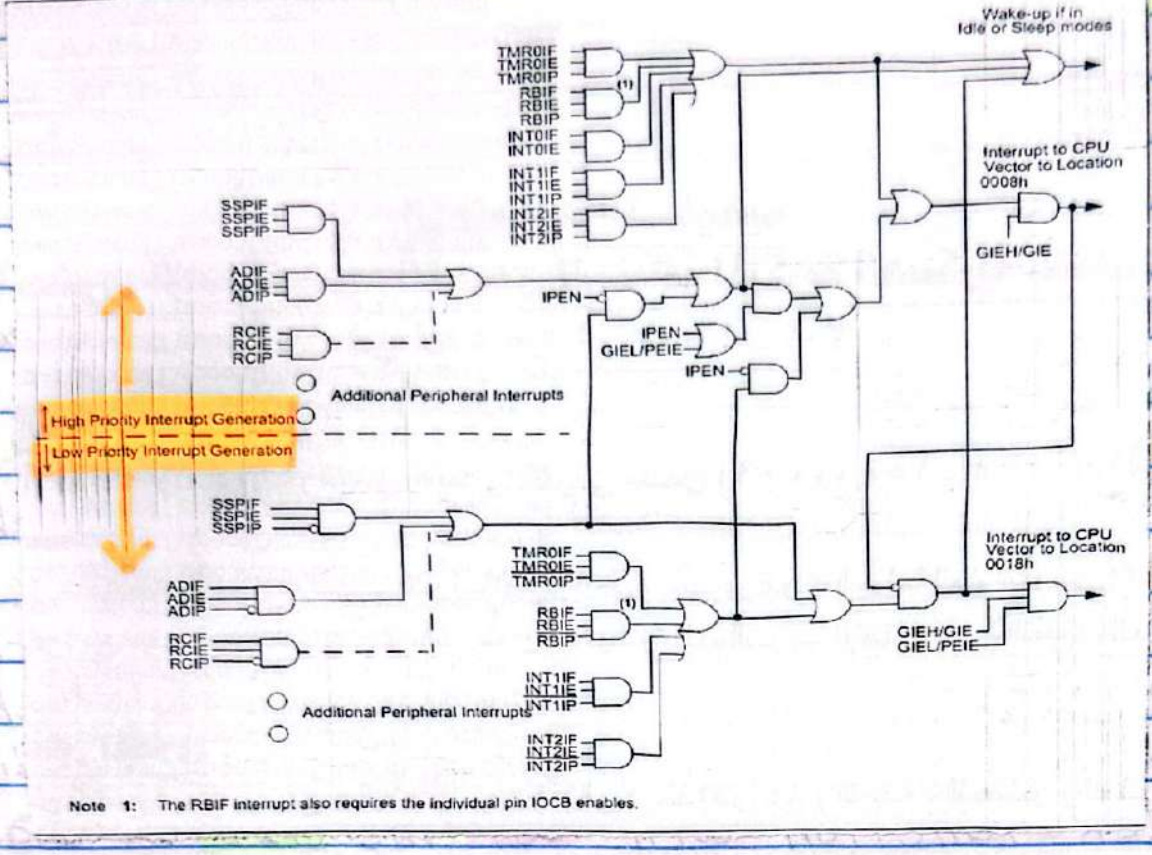
also :-

B4 : kBI0 , B5 : kBI1 , B6 : kBI2 , B7 : kBI3
these also work by external event

Page:..... Date:.....

note
= the Peripheral of fixed Priority, is it high or low ? the ans according to the data sheet.

FIGURE 9-1: PIC18 INTERRUPT LOGIC



Note 1: The RBIF interrupt also requires the individual pin IOCB enables.

Show the sequence of the Interrupt for
ex: Timer2

```
timer2_int();          → initalize timer2 function.
timer2_flag = 0;       → clear the flag of Timer2.
timer2_Interrupt_enable = 1; → enable Interrupt.
PEIE = 1;              → close Peripheral Interrupt gate.
GIE = 1;               → close General Interrupt gate.
timer2_on = 1;         → operate timer2.
```

Page:..... Date:.....

note when the Interrupt occurs so the flag Interrupt of timer2 will be "1" & the CPU will jump to the ISR function to execute the task at the ISR function you must clear the flag by the software

ex →

```
Void ISR_func (Void) {
    if(timer2_flag == 1) {
        /* Perform ISR timer2 task */
        timer2_flag = 0;
    } clear the flag by the software
} to get out from the Interrupt state.
```

note the Priority of the Interrupt is option.

note when the Interrupt occurs the Processor will Clear '0' the GIE "Globle Interrupt enable" & when the Processor finshed the execution of ISR function will set '1' the GIE to enable the Interrupt receiving to the MicroController.

---

the Implemention of the Interrupt:

this Module will have 3 types of the implemention:

1- MCal_interrupt_Manger:
has the ISR function low & high Priority

2- MCal_External_Interrupt:
the Interrupt that is operated by external action to the MC like INTx or RBIx

Page:..... Date:.....

3- MCal_interrupt_internal:
the Interrupt that is operated by the internal Peripheral inside the MC like ADC or UART .... ect.

Note
when you initialize the Interrupt Module you have to disable the Interrupt firstly after that Make your initializations & Configurations after that enable the Interrupt Module in the end of function Interrupt init #
look at the steps :-
1- disable the interrupt.
2- Clear the flag interrupt.
3- Configure the edge if the interrupt is external.
4- Configure the Priority.
5- Configure the I/o Pin if the interrupt is external.
6- Configure Default interrupt Call back function.
7- enable the interrupt.

Note at the on change external Interrupt
when the Pin of onchange is changed from high to low or low to high the Processor will Make an Interrupt this the Meaning of the Interrupt.

Page:..... Date:.....

Note

if you enable the Pirority levels you will this means :.

IPEN = 1 → GIEH = 1 for high
GIEL = 1 for low

after that enable the bit of the target Interrupt ex INT0
INT0E = 1

Note

if you disable the Pirority levels this means

IPEN = 0 →
GIE & PEIE Must be = 1

to enable the Interrupt without the Pirority levels.

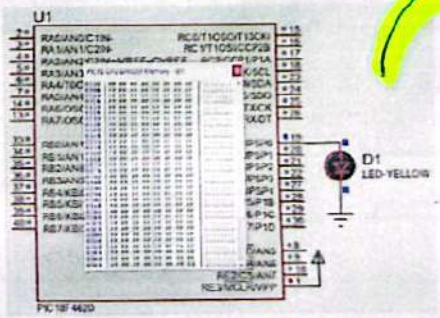
EEPROM Data Memory

this Memory is used to save the old value inside it even the system is Power off there are two types of it

→ intrna : inside the MC .
→ extrna : outside the MC .

EEPROM Data Memory

extrnal



*this extrnal EEPROM I use it if I don't have one is internal at My MCU or I need More of non Volitle Memories.

*there's one inside the Micro Controller

"I2C Protocal"

Page: ..... Date: .....

* the internal EEPROM :
Can be accessed for reading/writing operations by the code.
EEPROM : electrically erasable Programmable Read only Memory.
is non Volatile Memory. save the last data even the system is Power off.
the SFRs of internal EEPROM at PIC18F46k20 Micro controller are used to read/write data EEPROM
EECON1 - EECON2 :
Control the access to the data EEPROM
EEData :
hold the data to be written or to be retrieved
EEADR - EEADRH :
Pair used to address the data EEPROM for read & write operations.
note
= the life time "cycle endurance" is 1000 000 Read/write for the EEPROM.
note
= EEPROM allows to read "byte" or write "byte" one time.
note
= when you write data inside the EEPROM this location will be cleared & receive the new byte of data this Process it's called "erase-before-write".

Page:..... Date:.....

Note

the size of the EEPROM is 1k bytes the start Address of the EEPROM from 0X00 to 0X3FF

Note

0X3FF in binary →

9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1

you need at least 10 bits to save inside it the Address of the last location of the EEPROM so to access the location by the Address you have 2 Register to Make that each one is 8 bits the Registers are :-

EEADR

7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1

EEADRH

7	6	5	4	3	2	1	0
-	-	-	-	-	-	1	1

How to write the Address inside the two registers :-

uint16 bAdd = 0X3FF → 00000011 11111111

EEADRH = (bAdd >> 8) & 0X03

↳ 00000011 & 00000011 = 00000011

∴ EEADRH = 00000011

EEADR = bAdd & 0XFF →

↳ 00000011 11111111 & 00000000 11111111

= 11111111

∴ EEADR = 11111111

Page: ..... Date: .....

the steps to write Data inside the EEPROM

1- Store the Address location that you want to write inside it at EEADRH & EEADR registers

2- write the data inside the data register EEDATA

3- write "0" inside EEPGD bit at EECON1 register to accesses the EEPROM Data Memory if you write "1" you will access the flash Memory

4- write "0" inside CFGS at EECON1 register to select the rate of EEPROM Memory rather than the Configuration bits

5- write "1" inside WREN to enable the write operation to the EEPROM Memory

6- Disable the Globle interrupt GIE or GIEH at INTCON register.

7- Make unlocking operation to the EEPROM to write inside it by write 0x55 & 0xAA at EECON2 register (unlocking sequence)

8- to start the whitting operation you have to set WR bit inside EECON1 register & this bit when be Zero this means write Cycle to EEPROM is Completed. → this is done by the Processor

the steps to Read the Data from EEPROM the same thing at the whiting but there's

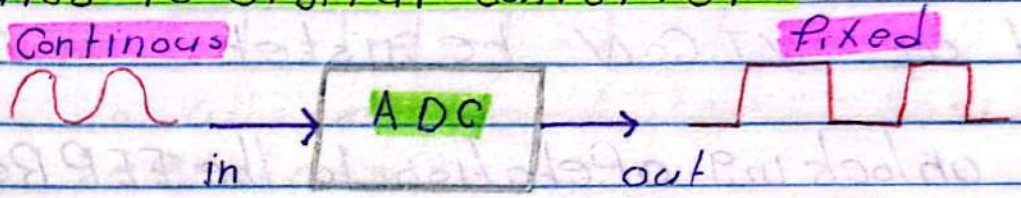
Page:..... Date:.....

a bit is responsible for the reading operation it's called RD bit while inside it "1" to start the Reading operation & will be "zero" when the read cycle is completed → this Made by the Processor

note
the internal EEPROM needs the Interrupt to enable & disable it at it's initialization function.

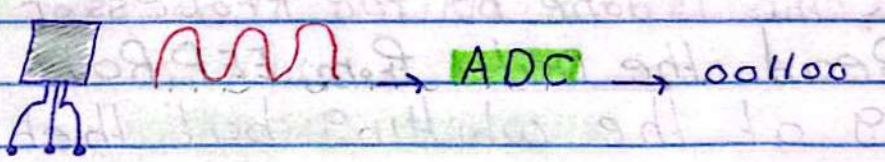
Q: why we disable the Global Interrupt when we write data inside the EEPROM?
to Prevent the data corruption at the case of the Interrupt.

Analog to digital Convertor:



note
there's an external ADC IC

what's the ADC
is a circuit converts a continuous voltage value "analog" to a binary value "digital" to the Processor can understand this value

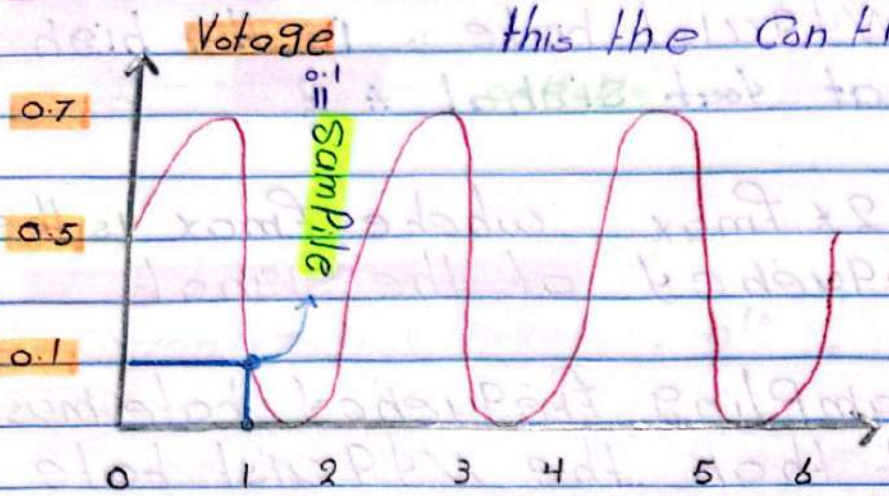


Page: ..... Date: .....

note the rasbetty Pi has no ADC circuit

DAC :: Digital to Analog Converter

Sampling



this the continous signal

Sample
Quantization
Encoding

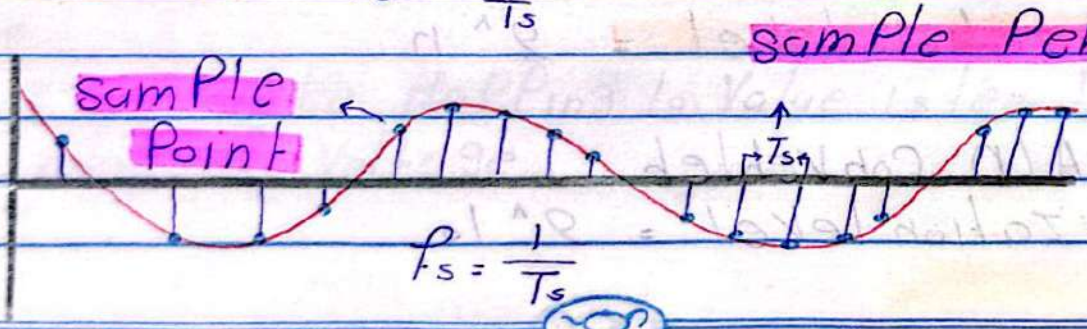
→ we take a sample at specific time to know the value of the voltage.

→ after that we Make the Quantization is assigen the value at a specific time to sample

→ after that we Make the encoding converts the value of the specific time into Patterh Group of zeros & ones 0011010101

note

we take the sample by specific rate. & this rate is determined by the sample frequency $f_s = \frac{1}{T_s}$.



Page: Date:

*

Note: the sample frequency must be with high rate to the conversion process be perfect.

how to know the high sampling frequency

→ know the Nyquist rate :: is the high frequency at your signal * 2

this means ::

$f_{nyquist} = 2 \times f_{max}$ where f_{max} is the highest frequency at the signal

→ the high sampling frequency rate must be greater than the Nyquist rate frequency $f_s > f_N$ where $f_N = 2 * f_{Max}$.

Note: if the sample rate frequency is less than the nyquist rate this is called aliasing result :: distortion of your signal.

Quantizing

I take the sample & make for it Mapping with the voltage levels.

How to know the number of the Quantization level ?

n-bit A/D Converter Module

Quantization level = 2^n

ex →

10-bit A/D Converter

Quantization level = 2^{10}

Page: ..... Date: .....

What's the role of resolution?

→ determines the Quantization levels.

→ determines the number of bits for the output Zeros & ones from ADC.

Sensor → ADC → 0011011 → How Many bits for those

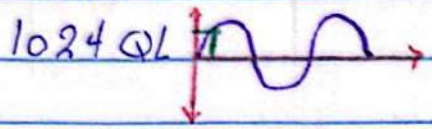
ex →

10-bit A/D system

→ the Quantization levels = 2^{10} = 1024

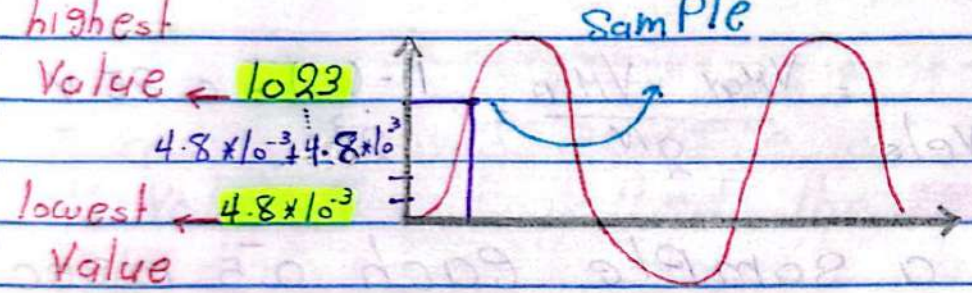
note

one sample has 1024 Quantization levels.



→ the Value of one Quantization level

$$q = \frac{V_{Max} - V_{Min}}{2^{resolution}} = \frac{5-0}{2^{10}} = \frac{5}{1024} = 4.8 \times 10^{-3} \text{ V}$$



note: the Q levels from 0 to $2^N - 1$

note

1024 Q levels = 5 V

∴ $\frac{1024}{2}$ = 512 Q levels = 2.5 V → $4.8\times10^{-3} \times 512 = 2.5$ V

note

I can't Make Mapping to Value is less than 4.8×10^{-3} to Value of Voltage.

Page: ..... Date: .....

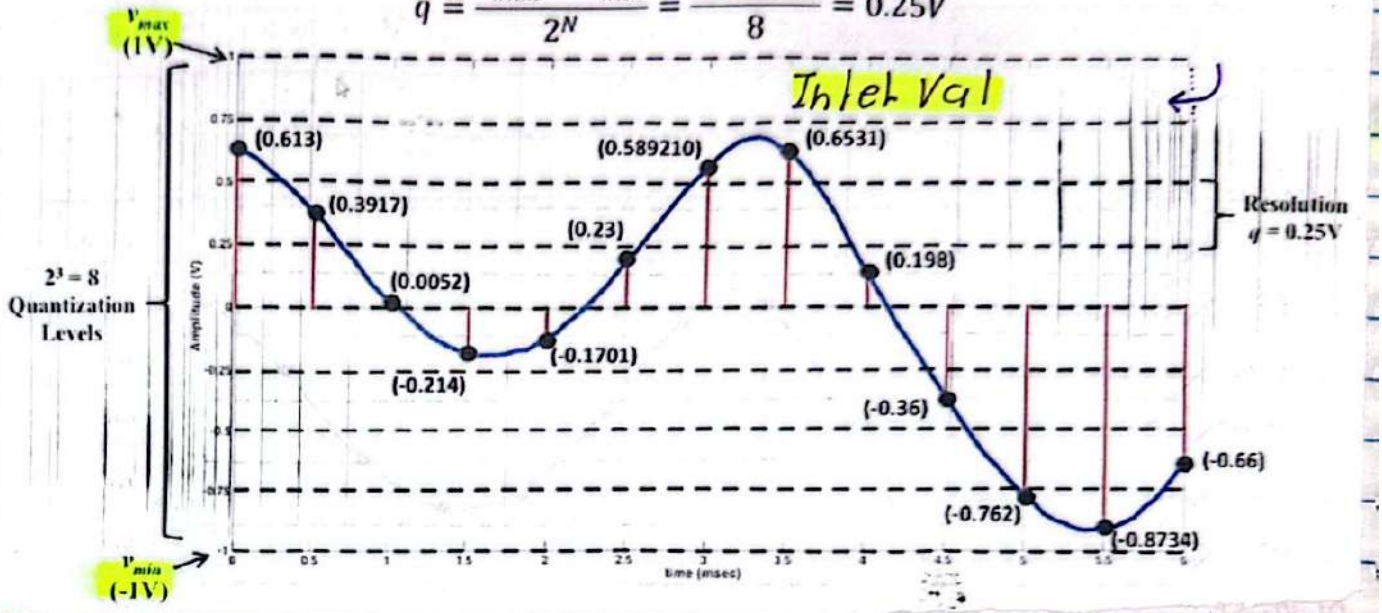
Note

when the resultion increases the number of Quantization levels increases & the Pherformance of Conversion will be More Perfect.

Analog To Digital Converter (ADC) – Introduction (Quantizing)

✓ The following figures represent conceptually how a 3-bit A/D converter converts an analog signal into bits.

$$q = \frac{v_{max} - v_{min}}{2^N} = \frac{1-(-1)}{8} = 0.25V$$



Inlet Val

the value of Quantization levels $= \frac{V_{Max} - V_{Min}}{2^N} = \frac{1-(-1)}{2^3} = 0.25\ V$

Note

we take a sample each 0.5 msec

$\rightarrow 0.5 \times 10^{-3}$ sec

the Sampling rate "f_s" $= \frac{1}{T_s} = \frac{1}{0.5 \times 10^{-3}}$

$= 2000$ sample this means I take each second 2000 sample & each 0.5 sec I take $\frac{2000}{2} = 1000$ sample & each 0.5msec

I take $1000 \times 10^{-3} = 1$ sample.

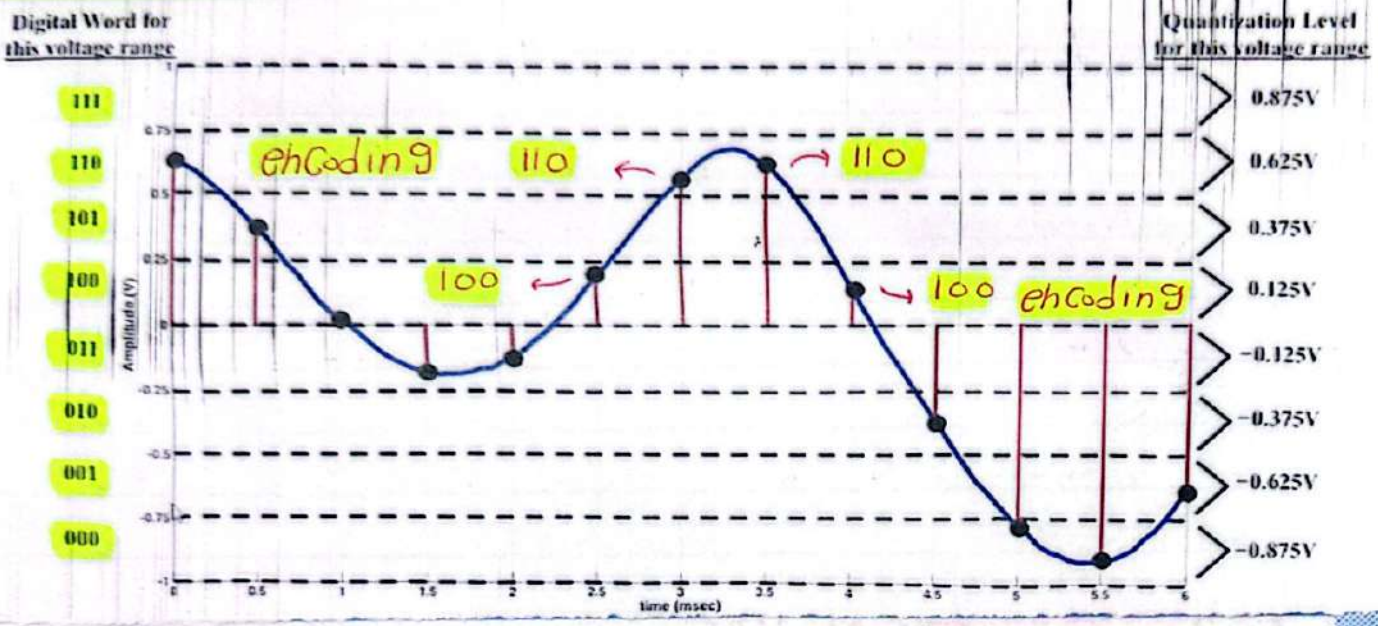
Page: ..... Date: .....

note I divide the analog signal into $2^N = 2^3 = 8$ IntetVal according to the resolution of A/D Convetter ithe width of one IntetVal is $\frac{V_{Max} - V_{Min}}{2^N}$

$= \frac{1 - (-1)}{2^3} = 0.25V$ ↕ 0.25 IntetVal

Analog To Digital Converter (ADC) – Introduction (Quantizing)

✓ Each of the voltage intervals is assigned an N-bit binary number representing the integers from 0 to (2^N)-1 → 3-bit A/D converter



I assigen each sample at the IntetVal to it's value this it's called the Encoding

note

when thete ate two ot Mote samples inside one IntetVal this Makes Phecetage of Ettot so when the humbet of the IntetVal incteaces the Ettot Ptecentage decteases & the humbet of the IntteVals dePends on the N-bit A/D C the tesly tion ↵

Page: ..... Date: .....

ADC types

* SAR : Successive Approximation Register
  - → Middle-Fast : speed < 5 MsPs .
    hote : sPs → sample Pet second
  - → Middle resolution : 8 : 16 Bits .

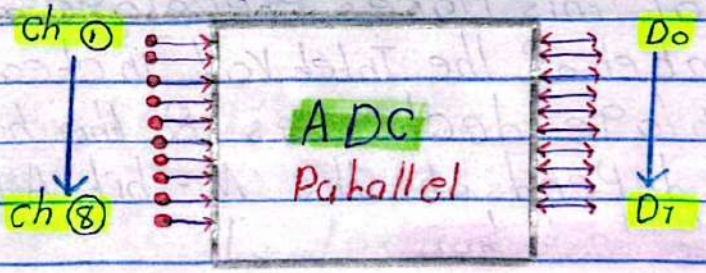
* SD : Sigma - Delta
  - → low - Middle : speed « few ksPs ».
  - → High resolution : 24 bits .

* Flash :
  - → Fast speed than SAR & SD .
  - → lowet resolution : 12 Bits .

hote
= SAR ADC is Mote used at MCUs .

What's the diffetence between the Patallel & setties ADC .

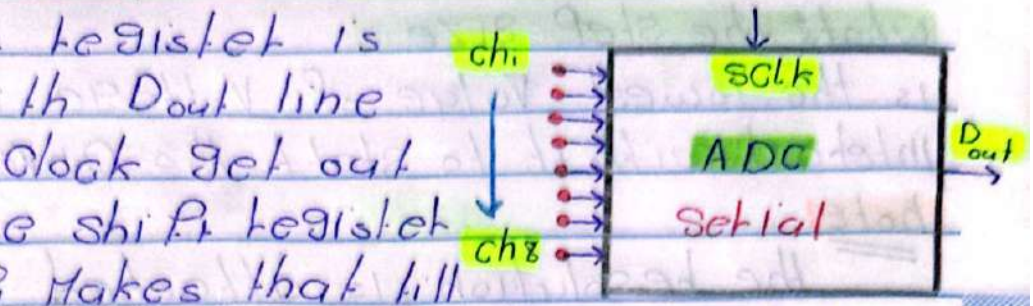
Patallel :
it dependes on the tesolution if the ADC has 8-bits this Means I heed 8 outPut lines fot the Citcuit & the Data will get out at the same time

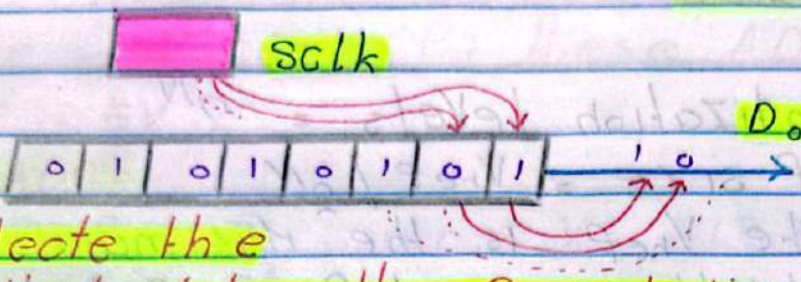


hote this Makes Hatdwate Complexity is high .

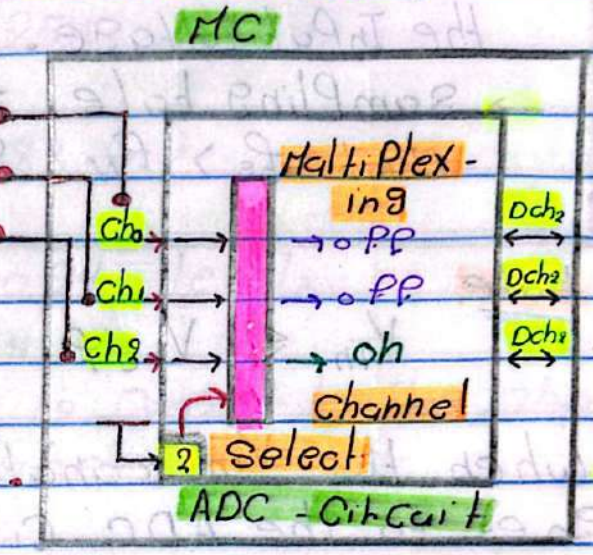
Page: ..... Date: .....

Serial :- has just one line for output data.
here the Question :
How I get 8 bits of Data on one line ?
there's a shift register is connected with Dout line when each clock get out 1 bit from the shift register to the Dout & makes that till reachs to the last bit of the shift register





How to selecte the Channel that makes the conversion ?
there's a bit inside the ADC circuit it's called Select channel value is connected with Multiplexer with other Channels to select the target channel from the select channel bit.



Ex →
If the select channel bit has "2" so so channel 2 will make the conversion & other channels will be off.

Page: Date:

hote
the value of the Quantization level it's called the step size.

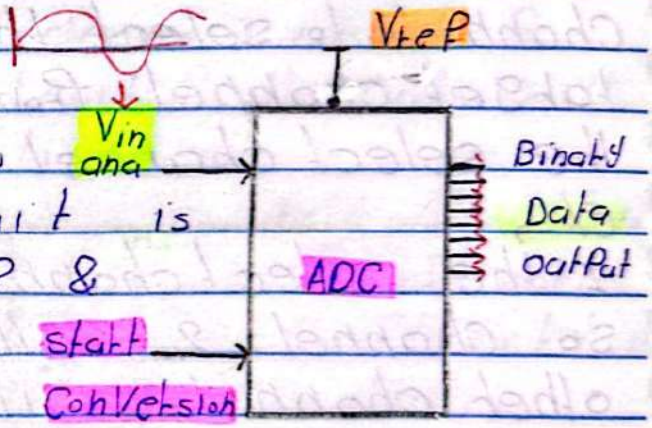
what's the step size :
is the lowest value of voltage the V_{in} can interact with it to start the conversion process. (channel)

hote
the resolution is attached to the design of the ADC circuit.

ADC laws

- Quantization levels = 2^N
- step size = $V_{ref} / 2^N$
  where V_{ref} is the maximum input voltage used for comparing the input voltage.
- sampling rate $f_s = 1/T_s$
  Must $f_s > f_N$ & $f_N = 2 * f_{max}$

hote
$V_{in} \leq V_{ref}$



* when the V_{in} "analog" goes to the ADC circuit is compared with the V_{ref} & start the conversion to get out the V_{in} analog as Binary Data output.

$V_{ref} = V_{max} - V_{min}$

Page: Date:

Q/ I need to Make the step size is 10mV & the ADC circuit is 8 bits what's the Value of Vref

$$\text{step size} = \frac{V_{ref}}{2^N} \rightarrow 10 \times 10^{-3} = \frac{V_{ref}}{2^8}$$

$\therefore V_{ref} = 10 \times 10^{-3} * 2^8 = 2.56V$

Note

$\therefore V_{in\ analog} \leq {}^{V_{ref}}2.56\ V$ #

Q/ determine the Quantization levels & the step size for these ADCs where $V_{ref} = 5V$

① 8 bits ADC :

→ $SS = \frac{V_{ref}}{2^N} = \frac{5}{2^8} = 0.019\ V = 19.53\ mV$

→ $QL = 2^N = 2^8 = 256$ levels

② 10 bits ADC

→ $SS = \frac{V_{ref}}{2^N} = \frac{5}{2^{10}} = 4.88\ mV$

→ $QL = 2^N = 2^{10} = 1024$ levels

what's the time Conversion

is the time take ADC to Convert the analog input to a digital number.

the Digital output data

8-bit ADC : Digital output $D_0 : D_7$

10-bit ADC : Digital output $D_0 : D_9$

How to Calculate the Digital output data

$$D_{out} = (V_{in} / \text{step size})_{-1}$$

Page: ..... Date: .....

Ex: Calculate the Dout if the V_{in} = 5V for 10 bit-ADC & V_{ref} = 5V

$$D_{out} = (V_{in} / \text{stepsize}) - 1$$

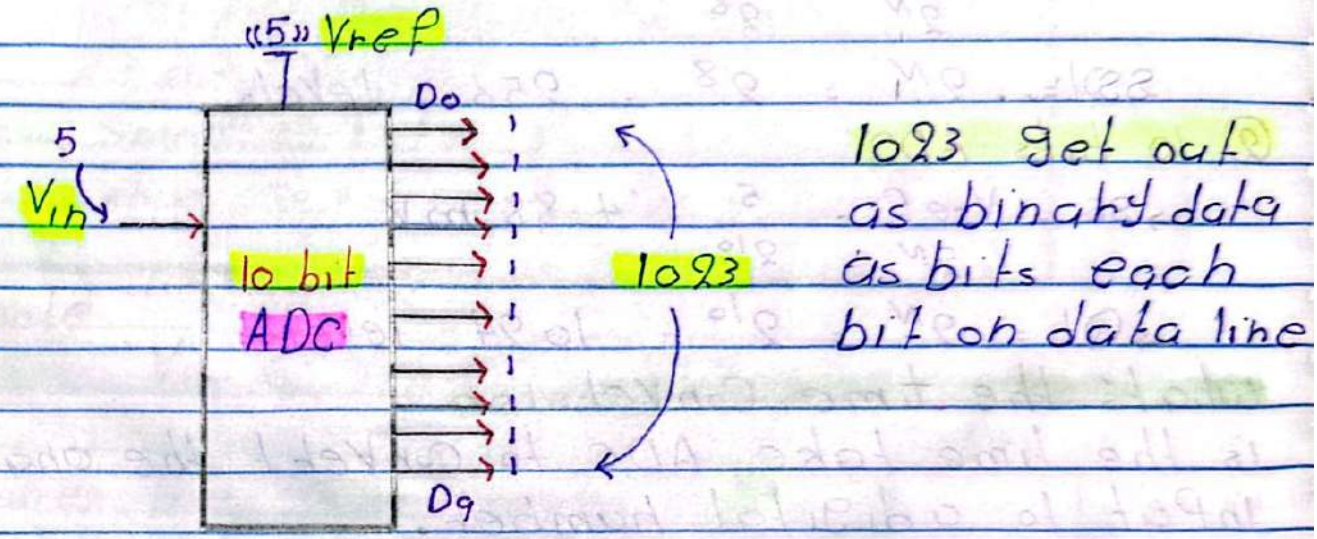
$$\text{stepsize} = \frac{V_{ref}}{2^N} = \frac{5}{2^{10}} = 4.8828125 \times 10^{-3} V$$

$$\therefore \text{stepsize} = 4.8828125 \times 10^{-3} V$$

$$D_{out} = (5 / 4.8828125 \times 10^{-3}) - 1 = 1023$$

$\therefore D_{out}$ = 1023 Digital Value

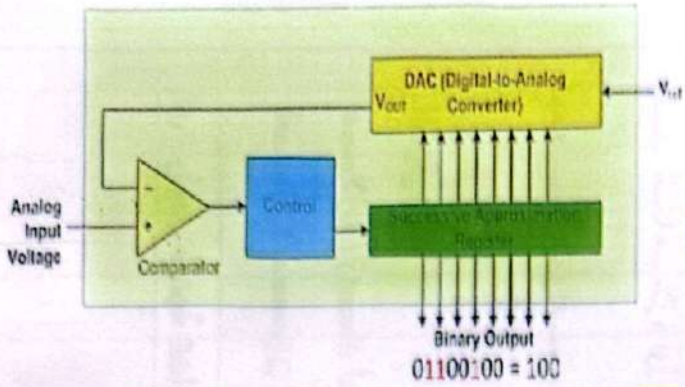
9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1



Successive APProximate ADC has 4 components

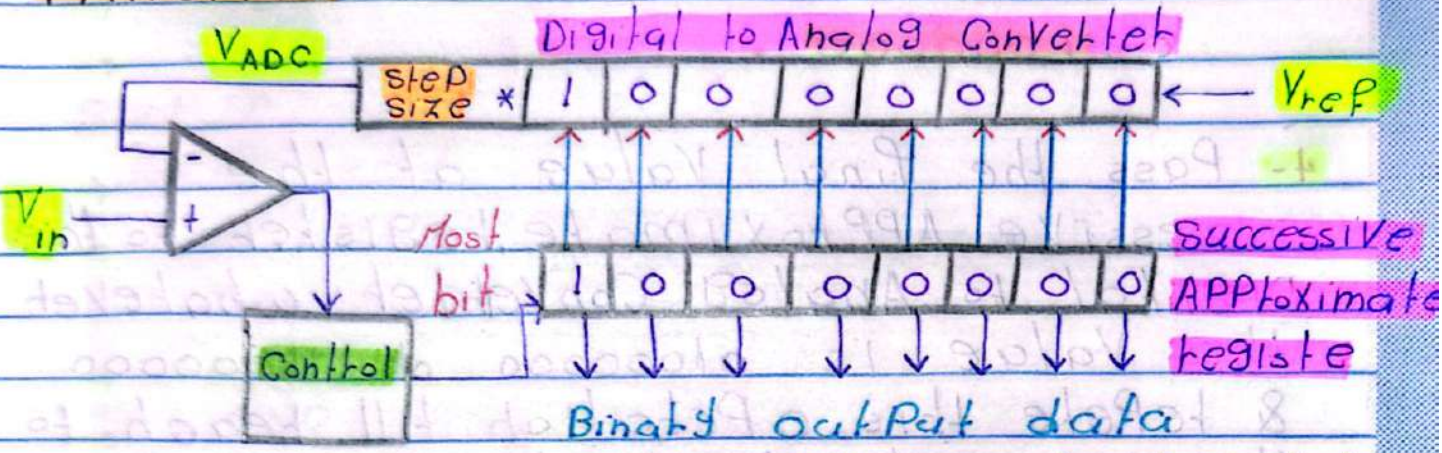
- Successive APProximate register
- Comparator
- Digital to analog Converter
- Control unit

Page: ..... Date: .....



Dout = Vin / StepSize
Dout = 1 V / 0.01 V = 100

Successive APPtoximate ADC wotking PtinCiPle



1- I go to successive apptoximate tegistet & wtite at the Most bit "1" & othet bits will be Zetos 10000000.

2- I Pass 10000000 value to the DAC & it will Multiple this value by the step size to detet mine the V_{ADC} = (10000000) * step size
let's suPPose that step size = 10mV
$V_{ADC} = 128 * 10 * 10^{-3} = 1.28\ V$
↳ whete 128 → 10000000.

3- ComPate V_{ADC} with V_{in} by the ComPatatot.

Page: Date:

if $V_{ADC} > V_{in}$
You will clear the Most bit at the successive APProximate register & set bit6 at the same register
01000000 .
if $V_{ADC} < V_{in}$
You will keep the Most bit at the successive APProximate register by "1" & set bit6 at the same register
11000000 .

4- Pass the final value at the successive APProximate register to the Digital to Analog Converter whatever this value is 01000000 or 11000000 & repete this operation till reach to the less bit at the successive APProximate register.

example
8-bit ADC
$V_{in} = 1$ V
step size = 10 mV

How ADC IC works

step ①
→ SAR = 10000000 = 128
→ DAC = SAR * step size = $128 * 10 * 10^{-3}$
= 1.28 V ∴ $V_{ADC} = 1.28$ V
→ $V_{ADC} > V_{in}$ ∴ 1.28 > 1 ∴ yes
→ ∴ SAR = 01000000
step ②
→ SAR = 01000000 = 64

Page: Date:

→ DAC = SAR * step size = $64 * 10 * 10^{-3} = 0.64V$

∴ $V_{ADC} = 0.64V$

→ $V_{ADC} \gtrless V_{in}$ ∴ $0.64 > 1$: NO

→ ∴ SAR = 01100000

Step ③

→ SAR = 01100000 = 96

→ DAC = SAR * step size = $96 * 10 * 10^{-3} = 0.96V$

∴ $V_{ADC} = 0.96V$

→ $V_{ADC} \gtrless V_{in}$ ∴ $0.96 > 1$ ∴ NO

→ ∴ SAR = 01110000

Step ④

→ SAR = 01110000 = 112

→ DAC = SAR * step size = $112 * 10 * 10^{-3} = 1.12V$

∴ $V_{ADC} = 1.12V$

→ $V_{ADC} \gtrless V_{in}$ $1.12 > 1$ ∴ yes

→ ∴ SAR = 01101000

Step ⑤

→ SAR = 01101000 = 104

→ DAC = SAR * step size = $104 * 10 * 10^{-3} = 1.04V$

∴ $V_{ADC} = 1.04V$

→ $V_{ADC} \gtrless V_{in}$ $1.04 > 1$ ∴ yes

→ ∴ SAR = 01100100

Step ⑥

→ SAR = 01100100 = 100

→ DAC = SAR * step size = $100 * 10 * 10^{-3} = 1$

∴ $V_{ADC} = 1V$

→ $V_{ADC} \gtrless V_{in}$ $1 \gtrless 1$ ∴ yes

→ ∴ SAR = 01100110

Step ⑦

Page: Date:

→ SAR = 01100110 = 102

→ DAC = SAR * step size = 102 * 10 * 10^{-3} = 1.02

∴ V_{ADC} = 1.02 V

→ $V_{ADC} \geq V_{in}$: 1.02 ≥ 1 ∴ yes

→ ∴ SAR = 01100101

Step (8)

→ SAR = 01100101 = 101

→ DAC = SAR * step size = 101 * 10 * 10^{-3} = 1.01

∴ V_{ADC} = 1.01 V

→ $V_{ADC} \geq V_{in}$ ∴ 1.01 ≥ 1 ∴ yes

→ ∴ SAR = 01100100

∴ the final value of SAR is 01100100 = 100

Successive Approximation register

	D7	D6	D5	D4	D3	D2	D1	D0
100 =	0	1	1	0	0	1	0	0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

Binary out Put Data

ADC Module at Data sheet

the registers

→ ADRESH :- A/D result high register.

→ ADRESL :- A/D result low register.

→ ADCON0 :- A/D Control register 0.

→ ADCON1 :- A/D Control register 1.

→ ADCON2 :- A/D Control register 2.

Page: ..... Date: .....

REGISTER 19-1: ADCON0: A/D CONTROL REGISTER 0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON
bit 7							bit 0

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 unknow

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

bit 5-2 **CHS3:CHS0:** Analog Channel Select bits → to select which channel you will operate it as analog pin.

0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)(1,2)
0110 = Channel 6 (AN6)(1,2)
0111 = Channel 7 (AN7)(1,2)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)
1101 = Unimplemented(2)
1110 = Unimplemented(2)
1111 = Unimplemented(2)

bit 1 **GO/DONE:** A/D Conversion Status bit → start ADC Conversion & check if is done or not.
When ADON = 1:
1 = A/D conversion in progress
0 = A/D Idle

bit 0 **ADON:** A/D On bit → if you want to enable or disable ADC.
1 = A/D Converter module is enabled
0 = A/D Converter module is disabled

Note 1: These channels are not implemented on 28-pin devices.
2: Performing a conversion on unimplemented channels will return a floating input measurement.

note
at bit ① Go : for the starting ADC Conversion operation
Done : for checking if the operation is done or not.

REGISTER 19-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-q(1)	R/W-q(1)	R/W-q(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

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

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

bit 5 **VCFG1:** Voltage Reference Configuration bit (VREF- source)
1 = VREF- (AN2)
0 = VSS

bit 4 **VCFG0:** Voltage Reference Configuration bit (VREF+ source)
1 = VREF+ (AN3)
0 = VDD

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

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O → 1 AN, 0 I/O

Note 1: The POR value of the PCFG bits depends on the value of the PBADEN Configuration bit. When PBADEN = 1, PCFG<2:0> = 000; when PBADEN = 0, PCFG<2:0> = 111.
2: AN5 through AN7 are available only on 40/44-pin devices.

note
form 0:3 bit is for select which channels you want to operate it as Analog or digital.

note
bit 4:5 is for if you want the Vref is the Voltage of MC or with external Battery is connected with the Micro Controller.

note
You can't select more than one channel at the same time.

Page:..... Date:.....

REGISTER 19-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

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

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified
0 = Left justified

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

bit 5-3 **ACQT2:ACQT0:** A/D Acquisition Time Select bits
111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)

bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Select bits
111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2

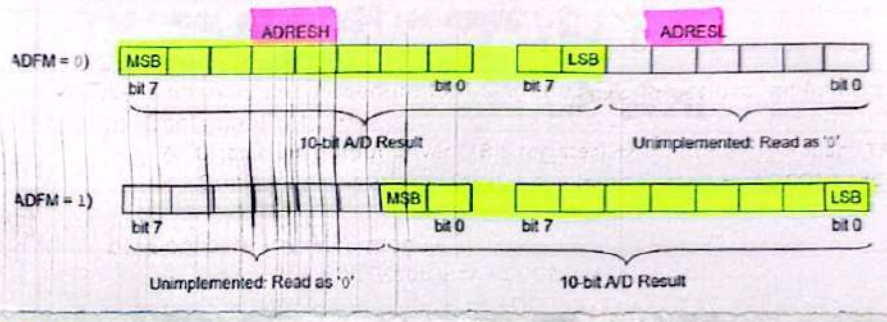
Note 1: If the A/D FRC clock source is selected, a delay of one TCY (instruction cycle) is added before the A/D clock starts. This allows the SLEEP instruction to be executed before starting a conversion.

bit 7 :- ADFM :- A/D Result Format
there are 10 bits to store the data of the Conversion & this bit is for to make the first 8 bits at the right & other 2 bits at another register or the opsite.

ADC Configurations (Result Formatting)

- ✓ The 10-bit A/D conversion result can be supplied in **two formats**.
  - Left justified
  - Right justified
- ✓ The **ADFM** bit of the **ADCON2** register controls the output format.

bit 7	**ADFM:** A/D Result Format Select bit 1 = Right justified 0 = Left justified



hote
it's recommended to use the right
I want to write 1023 inside 2 register & read it. value = 1023 → 11 11111111 right
ADRESL = 11111111 = 255
ADRESH = 3 × 256 = 768 ADC_Val = 255 + 768
↳ 11 → the rest of bits = 1023

ADC_Value = ADRESL + 256 × ADRESH

Page: Date:

REGISTER 19-4: ADRESH: ADC RESULT REGISTER HIGH (ADRESH) ADFM = 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ADRES9	ADRES8	ADRES7	ADRES6	ADRES5	ADRES4	ADRES3	ADRES2
bit 7							bit 0

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

bit 7-0 **ADRES<9:2>:** ADC Result Register bits
Upper eight bits of 10-bit conversion result

REGISTER 19-5: ADRESL: ADC RESULT REGISTER LOW (ADRESL) ADFM = 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ADRES1	ADRES0	—	—	—	—	—	—
bit 7							bit 0

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

bit 7-6 **ADRES<1:0>:** ADC Result Register bits
Lower two bits of 10-bit conversion result

bit 5-0 **Reserved:** Do not use

REGISTER 19-6: ADRESH: ADC RESULT REGISTER HIGH (ADRESH) ADFM = 1

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	—	—	—	ADRES9	ADRES8
bit 7							bit 0

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

bit 7-2 **Reserved:** Do not use
bit 1-0 **ADRES<9:8>:** ADC Result Register bits
Upper two bits of 10-bit conversion result

REGISTER 19-7: ADRESL: ADC RESULT REGISTER LOW (ADRESL) ADFM = 1

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ADRES7	ADRES6	ADRES5	ADRES4	ADRES3	ADRES2	ADRES1	ADRES0
bit 7							bit 0

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

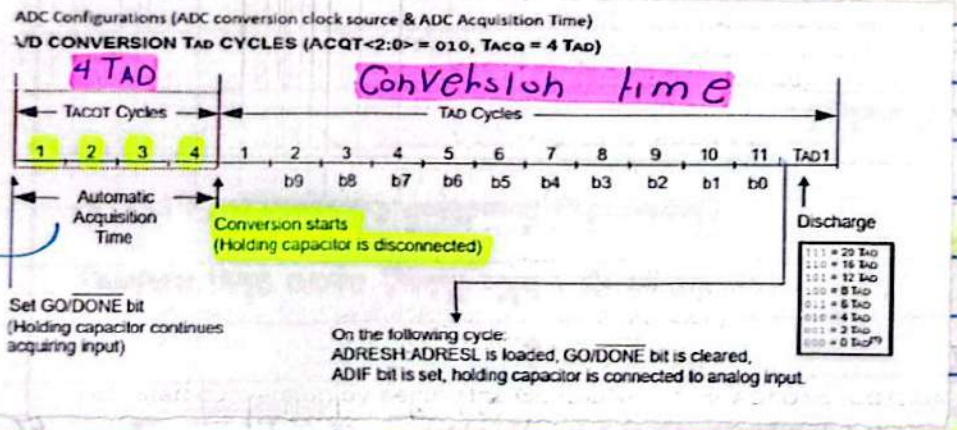
bit 7-0 **ADRES<7:0>:** ADC Result Register bits

as you see here the right & left result format to store the Data according to the value of ADFM bit at ADCON2 register this the way of the storge binary Data out Put

bit 5:3 at ADCON2 register is for the Acquisition time.

what's the Acquisition time:

is the required time to charge the holding capacitor to capture the Vin during the sampling.



wait till charge the capacitors

note

you can select the Acquisition time but it must be not reduced about the Manule Acquisition time. at our MC Pic18f46k20 Acquisition Manule time = 2.4 M/sec

Page: ..... Date: .....

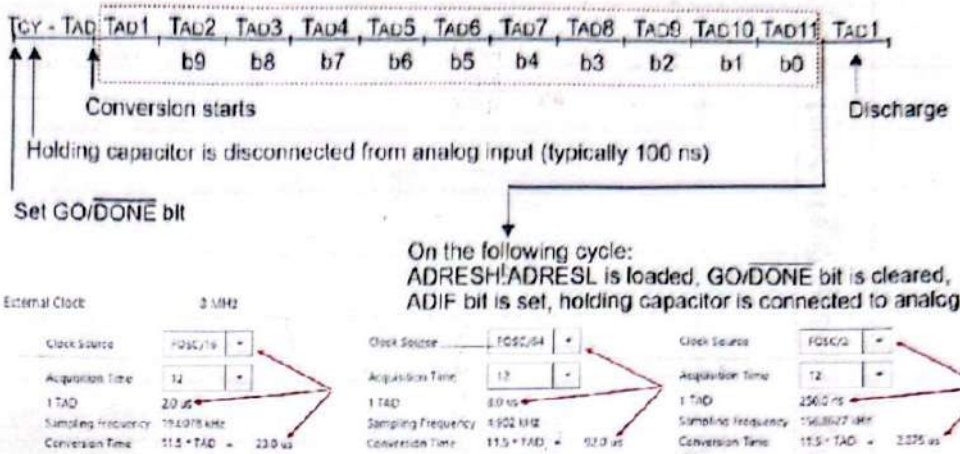
note

one full 10-bit conversion requires 11 TAD periods this means that one bit requires one TAD.

note

after the conversion time finshed the capacitors will be discharged.

- ADC configurations (ADC conversion clock source & ADC Acquisition Time)
  - ✓ The time to complete one bit conversion is **defined as Tad**.
  - ✓ One full 10-bit conversion requires **11 TAD periods**



note

the value of TAD is depends on the used clock source & is selected from bit 2:0 at ADCON2 register.

example ↳ fosc

→ external clock = 8 MHz = $8 * 10^6$ Hz

→ Clock source = fosc / 16 = $8 * 10^6 / 16 = 500$ kHz

→ $TAD = \frac{1}{f_{clock\ source}} = \frac{1}{500 \times 10^3} = 2$ µsec

the steps to implement the ADC Module

Page: ..... Date: .....

ADC Conversion Procedure

✓ This is an example procedure for using the ADC to perform an Analog-to-Digital conversion

a) Configure Port
- ✓ Disable pin output driver (TRIS register) → Input Configuration (TRISx register)
- ✓ Configure pin as analog (Disable the Input Buffer) → ADCON1 (PCFG3:PCFG0)

b) Configure the ADC module
- ✓ Disable the ADC Module → ADCON0 (ADON = 0)
- ✓ Select ADC input channel → ADCON0 (CHS3:CHS0 = Channel)
- ✓ Select ADC conversion clock → ADCON2 (ADCS2:ADCS0 = Conversion Clock)
- ✓ Select acquisition time → ADCON2 (ACQT2:ACQT0 = Acquisition time)
- ✓ Configure voltage reference → ADCON1 (VCFG1 + VCFG0)
- ✓ Select result format → ADCON2 (ADFM)
- ✓ Turn on "Enable" ADC module → ADCON0 (ADON = 1)

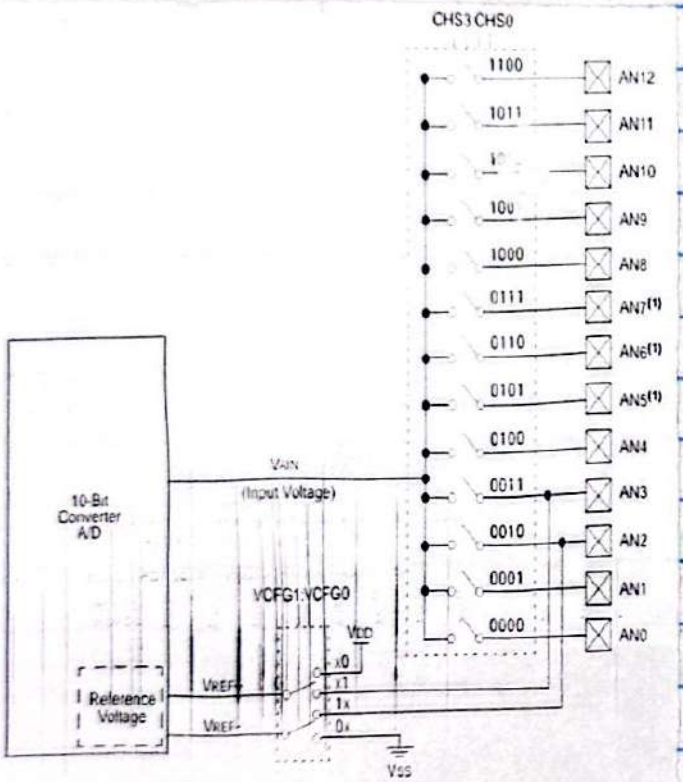
c) Configure ADC interrupt (Optional)
- ✓ Clear ADC interrupt flag → PIR1 (ADIF = 0)
- ✓ Enable ADC interrupt → PIE1 (ADIE = 1)
- ✓ Configure the interrupt priority if needed → IPR1 (ADIP = Interrupt Priority)
- ✓ Enable peripheral interrupt → INTCON (GIE/GIEH)
- ✓ Enable global interrupt → INTCON (PEIE/GIEL)

d) Start conversion by setting the GO/DONE bit. → ADCON0 (GO/DONE = 1)

e) Wait for ADC conversion to complete by one of the following
- ✓ Polling the GO/DONE bit (Periodically check the GO/DONE bit == 0)
- ✓ Waiting for the ADC interrupt (Interrupts enabled) → PIR1 (ADIF = 1)

f) Read ADC Result → ADRESL + ADRESH

g) Clear the ADC interrupt flag (Required if interrupt is enabled) → PIR1 (ADIF = 0)



Note 1: Channels AN5 through AN7 are not available on 28-pin devices.
2: I/O pins have diode protection to VDD and VSS.

* this the block digram of the ADC Module accordding to the CHS3 : CHS0 you select the target channel & VCFG1 : VCFG0 to select if you will use the voltage of the MC or external battery for the Vref

hote

ADIF Must cleat by Software.

Page:..... Date:.....

Timers

a timer Peripheral is a Counter Register & Control Register to determine its usage.

note

to operate any Module inside the MC this Module must has a Clock to be operated

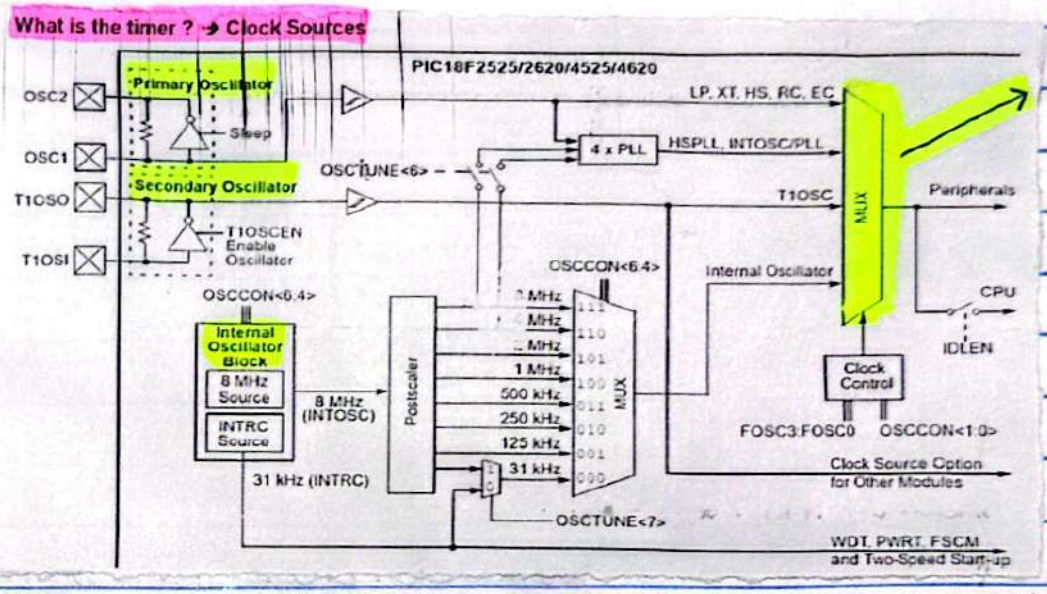
note

every timer needs a clock source:

- → internal clock source.
- → external clock source.

the internal clock source is

- → Primary oscillator.
- → secondry oscillator.
- → Internal oscillator block.



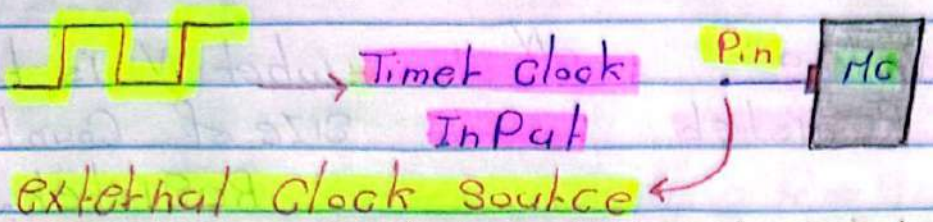
note

this Multiplexer to determine which the oscillator will Make the clock source.

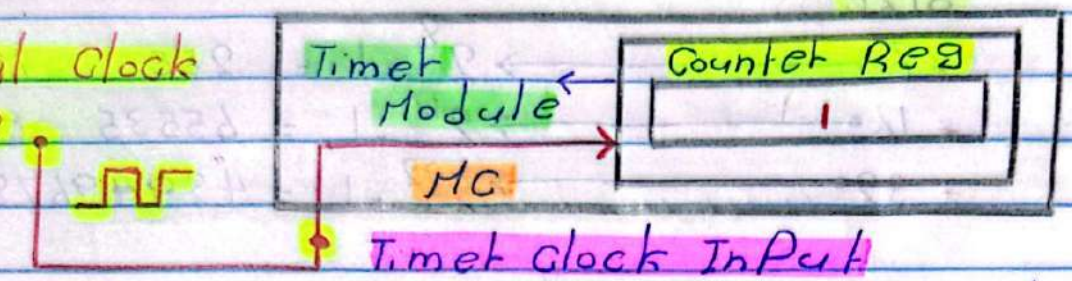
the external oscillator or clock source any thing else the Primary & secondry oscillator & the Internal oscillator block is consider as external clock source How it's Connected

Page: ..... Date: .....

the MicroControllet has a Pin is Configuted as an exteknal Clock to the Timet Module



the Countet Registet with the exteknal Clock



with each exteknal clock teached to the Timet Module the Countet Registet will be inCkemented by 1

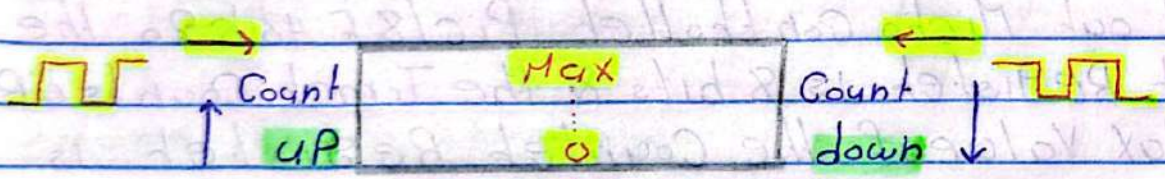
Countet Registet ++

hote

the Countet Registet Counts the teceiVed clocks.

hote

the Countet Registet May be is Count uP ot Count down



what's the timet tesolution

it deteknines & limits the Max Value of the Countet tegistet.

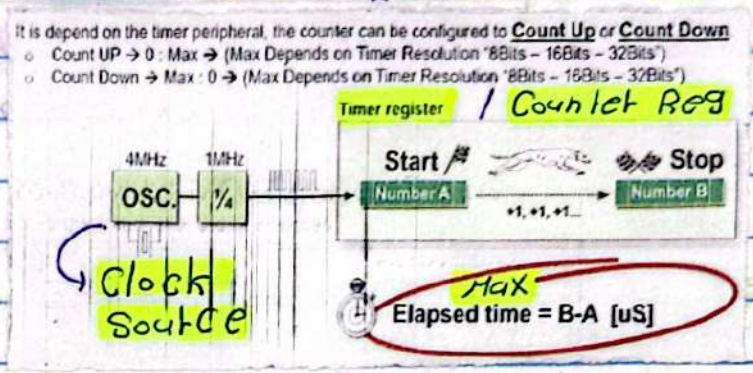
Page: ..... Date: .....

the timer resolution

the Counter register may be 8 - 16 - 32 bits

* the Max Value of Counter Register $= 2^N - 1$
* what N is the size of Counter Register.

Regisle Counter size	Max Value
* 8	$2^8 - 1 = 255$ Max
* 16	$2^{16} - 1 = 65535$ Max
* 32	$2^{32} - 1 = 4294967295$ Max



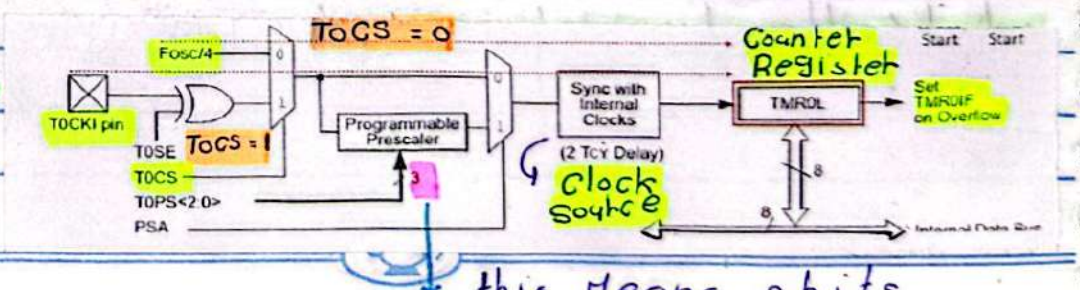
note

the timer is counting up or down is according to the design of the target Micro Controller.

note

at our Micro Controller Pic18F46k20 the Counter Register is 8 bits & the Timer Counts up the Max Value of the Counter Register is $(2^8 - 1) = 255$ Max Value size 8 bit.

the block diagram of Timer0



this means 3 bits to Control

Page: ..... Date: .....

with each clock the Timer0 will increment the value of the Counter Register by one till reach to the over flow value where over flow value = Max value + 1 the TMR0IF flag will Make Interrupt & Make the value of the Register Counter is Zero.

note

Fosc/4 :: the Internal Clock source.

T0CS :: the external Clock source at a specific Pin in the Mc.

ex

Let's suppose that Fosc = 4 MHz

F Clock = Fosc / 4 = 4/4 = 1 MHz

the time of one Clock (T) = $\frac{1}{F_{clock}} = \frac{1}{1 \times 10^6}$

∴ the time of one Clock (T) = 1 Micro sec

the Max value of the Counter Register of Timer0 of Pic18F46k20 is 255

∴ the Max Time of timer0 is = $1 * 10^{-6} * 255$

= 255 Micro sec this Means the Interrupt of Timer0 will be happen each 255 Micro second.

What's the timer Modes

any timer is operated with 2 Main Modes

→ Timer Mode :: Counter is known.

→ Counter Mode :: Counter is unknown.

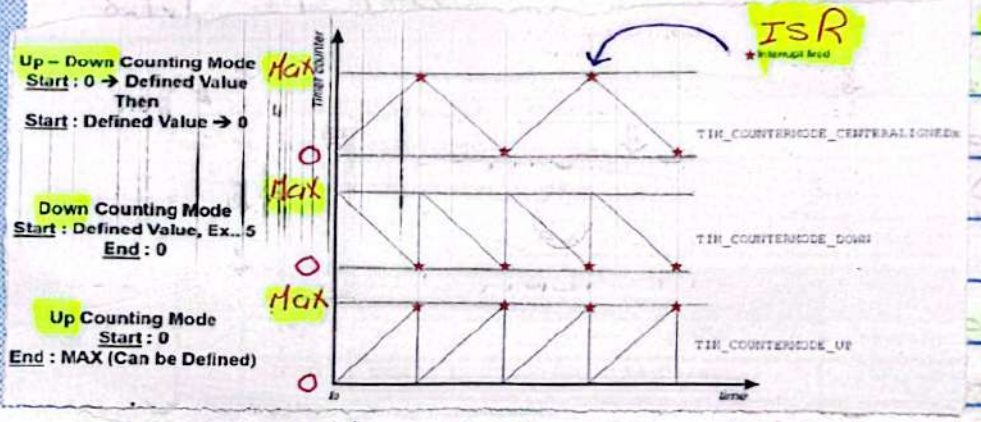
Page: ..... Date: .....

Note

The Timer Mode : I increment the value at the register Counter regularly with each clock so the Timer Mode is used with the Internal Source Clock

the Counter Mode is used with the external source clock because the clock source may be Push button or sensors generate the clock I don't know when the clock will be generated to increment the value at the Counter Register opposite the internal clock source "fosc/4".

three Major Counting Modes at Timer



* UP-down Mode will start from to the Max Value & dechemented to reach the Zero one More time

* Down Mode will start from the Max Value till reach to the Zero & is set with the Max Value one More time

* UP Mode will start from the Zero till reach to the Max Value & is set with Zero one More time.

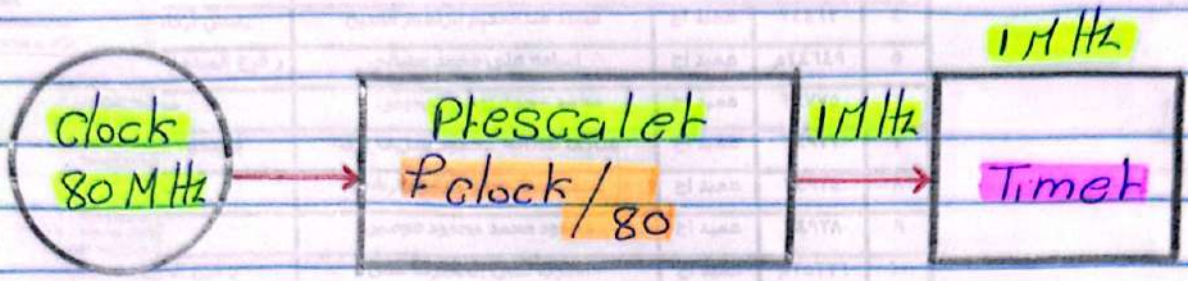
★ → ISR function executed.

Page:..... Date:.....

what's the timer Prescaler

is a frequency divider to reduce the value of the InPut frequency.

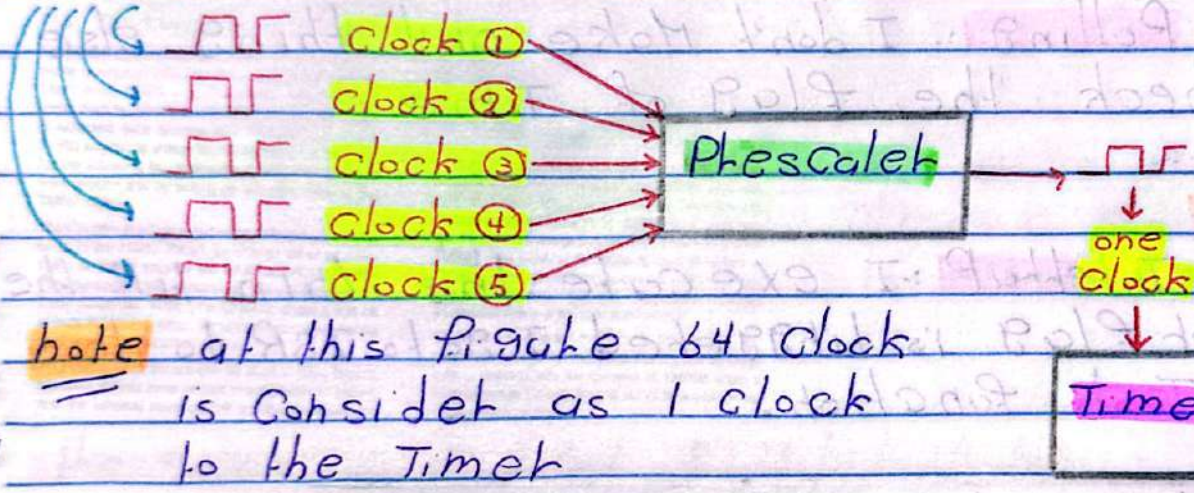
frequency / Prescaler Value



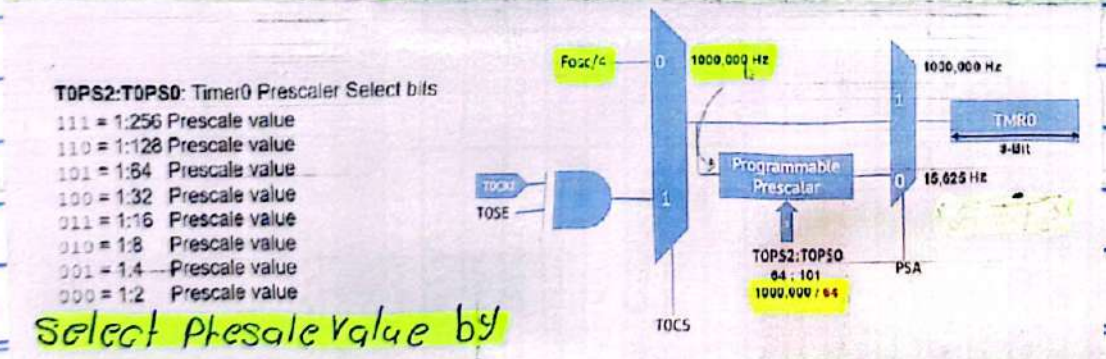
hote

oPPsite the Prescaler is the Muli Plet fclock * value

what's the behnfit of the Prescaler if you want to Make humber of Clock as one Clock what means ?!



hote at this figure 64 clock is Consider as 1 clock to the Timer



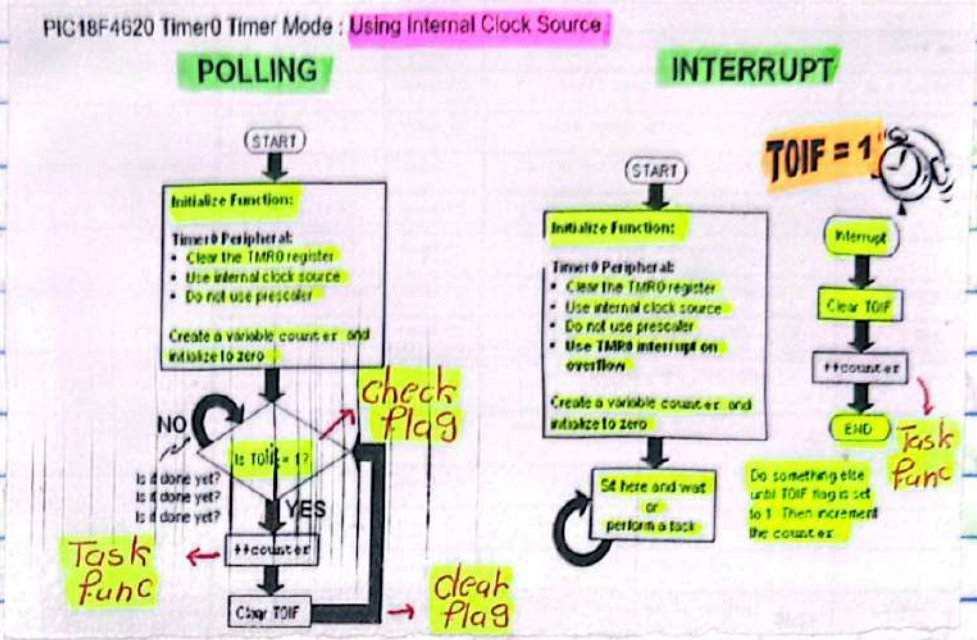
Select Presale value by Data Sheet

Page:..... Date:.....

there are two ways to theate with Timer to do the Task each say second

→ the Polling .

→ the Interrupt.



hote

Polling :. I don't Make any thing else I check the flag of Time

hote

Interrup :-I execute the Main till the Timer flag is triggered I go to ISR to do the Task function.

Timer0 Module at Data sheet PIC18F46k20

→ operate Timer/Counter in 8 bit / 16 bit.

→ 8 bit Prescaler Max value $2^8 = 256$.

→ selectable Clock source « internal/external».

→ Interrupt on over flow value.

Page: ..... Date: .....

the Registers

Timer0 Control Register

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

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

bit 7 **TMR0ON**: Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin → Clock external Counter Mode
0 = Internal instruction cycle clock (CLKO) → Timer Mode Internal Clock

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Max Value = 256

Min Valu = 2

hote

Transition Means Falling to Raising edge of oPsize is determine by bit 4 select edge T0SE.

hote

this register you set by it all the Configurations of the timer0

Timer0 Counter Registers TMR0L & TMR0H

timer0 oPeration

11.1 Timer0 Operation

Timer0 can operate as either a timer or a counter, the mode is selected with the T0CS bit (T0CON<5>). In Timer mode (T0CS = 0), the module increments on every clock by default unless a different prescaler value is selected (see **Section 11.3 "Prescaler"**). If the TMR0 register is written to, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

The Counter mode is selected by setting the T0CS bit (= 1). In this mode, Timer0 increments either on every rising or falling edge of pin RA4/T0CKI/C1OUT. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (T0CON<4>); clearing this bit selects the rising edge. Restrictions on the external clock input are discussed below.

An external clock source can be used to drive Timer0; however, it must meet certain requirements to ensure that the external clock can be synchronized with the internal phase clock (Tosc). There is a delay between synchronization and the onset of incrementing the timer/counter.

11.2 Timer0 Reads and Writes in 16-Bit Mode

TMR0H is not the actual high byte of Timer0 in 16-bit mode; it is actually a buffered version of the real high byte of Timer0 which is not directly readable nor writable (refer to Figure 11-2). TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides the ability to read all 16 bits of Timer0 without having to verify that the read of the high and low byte were valid, due to a rollover between successive reads of the high and low byte.

Similarly, a write to the high byte of Timer0 must also take place through the TMR0H Buffer register. The high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows all 16 bits of Timer0 to be updated at once.

* this section at the Data sheet explains How Timer0 Module is oPerated.

the Selection of 8 bit & 16 bit Modes

8 bit Mode

the Counter Register

TMR0L | 0 → 255

0

$2^8 - 1$

= 255

16 bit Mode

the Counter Register

TMR0H + TMR0L

65535 ← ——— 0

$2^{16} - 1$

0

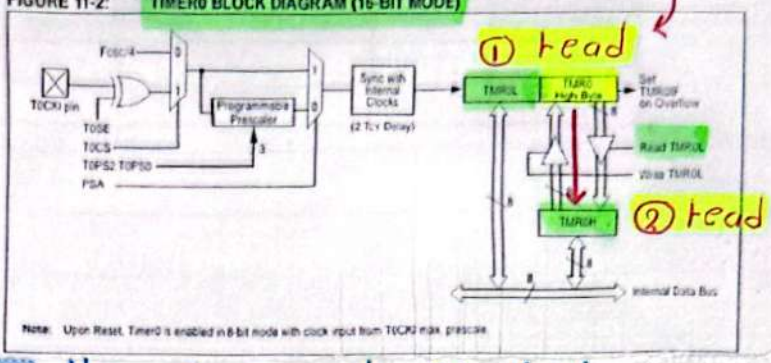
= 65535

Page: ..... Date: .....

hote

there's a specific way to read the value at the Counter register if you configure Timer0 is operated with 16 bit Mode

FIGURE 11-2: TIMER0 BLOCK DIAGRAM (16-BIT MODE)



* when you read the value at low register the value at the TMR0 High byte gets down to the TMR0H register & you can read the High value from it.

this case at 16 bit Mode to read the Data at Counter Register

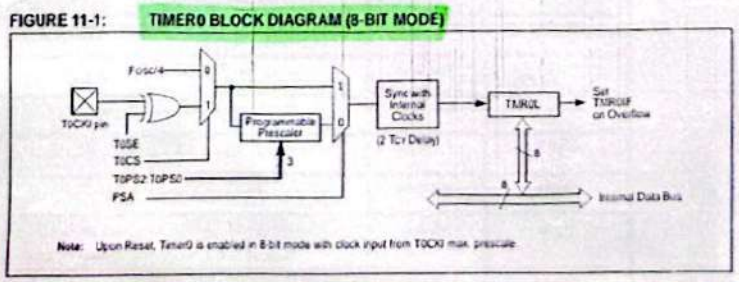
hote

the same thing at the whitting. but the value at TMR0H will go up to the TMR0 High byte register when you write Data inside the low Counter Register.

hote you must prepare the value at the TMR0H Register to be ready to go up to TMR0 High byte when you write data at the low Counter Regster "TMR0L".

the block digram of 8 bit Mode

FIGURE 11-1: TIMER0 BLOCK DIAGRAM (8-BIT MODE)



is too simple to use this Mode

-# HAHA

Page: ..... Date: .....

hote

I use the Counter Mode with the external clock if I want the Timer Counter Register counts the clocks by external events what ever is falling or raising edge.

hote

You can set the Prescaler at Counter Mode.

TABLE 11-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMR0L	Timer0 Register Low Byte								50
TMR0H	Timer0 Register High Byte								50
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	50
TRISA	RA7(1)	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0	52

Legend: Shaded cells are not used by Timer0.
Note 1: PORTA<7:6> and their direction bits are individually configured as port pins based on various primary

Timer0 Peripheral Configuration

* Timer Mode
  - → Disable Timer0.
  - → Select Timer0 resolution 8 bit / 16 bit.
  - → Select Internal clock source.
  - → Enable the Prescaler if you need it.
  - → Enable the Interrupt if you need it.
  - → Enable Timer0.

hote

You have to initialize the Counter Register by value

* Counter Mode
  - → Disable Timer0.
  - → select Timer0 resolution 8 bit / 16 bit.
  - → select external clock source.
  - → select the edge of the external clock source "rising . falling".
  - → Enable the Prescaler if you need it.
  - → Enable the Interrupt if you need it.
  - → Enable Timer0.

Page:..... Date:.....

* How to Calculate the time of the Timer to do a task each this time

TIMER0 L/H = 2^N - ((delay * Fosc) / (4 * Prescaler))

where

N → the resolution

delay → time to do task Must be in second

Fosc → the system frequency

Prescaler →

→ if you enable the Prescaler the value of it from 2 to 256

→ if you disable the Prescaler equals = 1

note

the final value Must be +ve without fraction.

ex →

we need a delay every 250 msec

Fosc = 4 MHz

Timer0 resolution is 16 bit

Timer0 L/H = 2^N - ((delay * Fosc) / (4 * Prescaler))

= 2^{16} - ((250 * 10^{-3} * 4 * 10^6) / (4 * 4))

= 3036 → decimal

= BDC → Hex

Timer0H ↘ Timer0L

Timer0 L = 0xDC

Timer0 H = 0x0B

note

why we need to initialize the Counter register by value in the first? to Make the wanted delay to do a task every this delay.

Page: ..... Date: .....

hote

If you initialize the Counter register by value this means the Timer when receives a clock will increment on this value till reach to the Max Value of the Counter register.

ex →

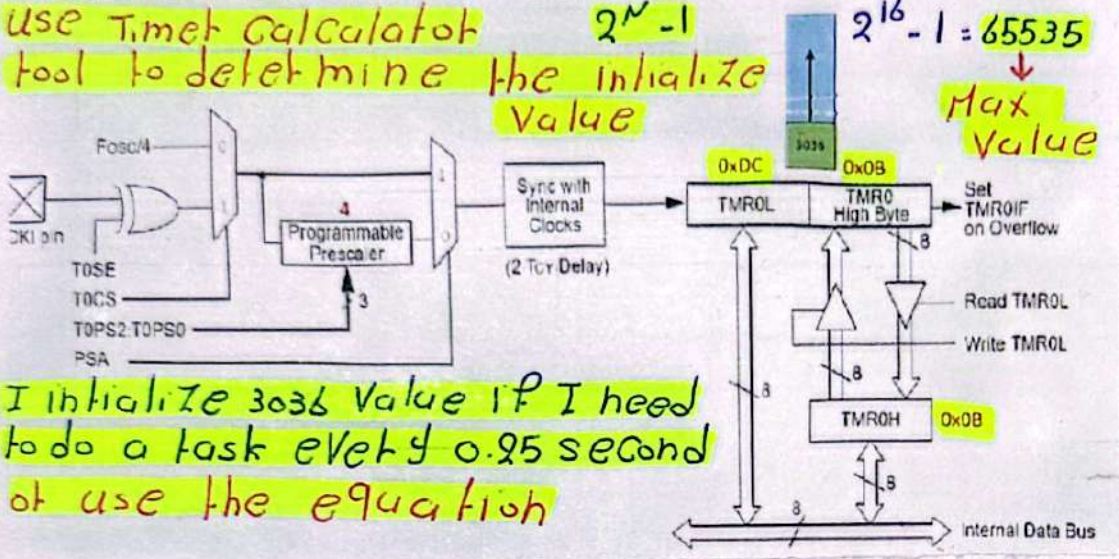
I initialize Timer0 L/H = 0x0BDC = 3036
the Counter Register will be incremented from 3036 to Max Value every received one Clock to the Timer0 Module 3036 ++ ← Clk

PIC18F4620 Timer0 Timer Mode : Generate a Time Delay
- Example, we need a delay every 250 ms "0.25 second", the Fosc = 4MHz and timer 0 resolution is 16-bit.

use Timer Calculator tool to determine the initialize value

$2^N - 1$

$2^{16} - 1 = 65535$ ↓ Max Value



I initialize 3036 value if I need to do a task every 0.25 second or use the equation

hote

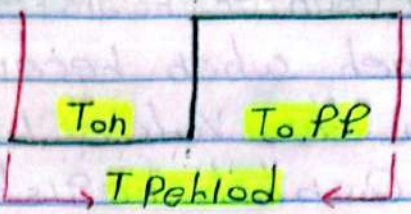
Initialization Counter Register Value is not important at the Counter Mode. Timer0 L/H = 0

How to generate the square wave form of the Timer firstly this the square wave form

→ firstly what's the frequency of the square wave form.

Page:..... Date:.....

→ second thing Is Ton = Toff



$f = \frac{1}{T_{Period}}$

ex →

I need to generate a square wave form with 50 Hz frequence & Fosc = 4MHz & the Timer resolution is 16 bit & Ton = Toff what is the Initialization Value to the Counter register

f = 50 Hz

Fosc = 4×10^6 Hz Ton = Toff

resolution = 16 bit

$T_{square\ form} = 1/f = \frac{1}{50} = 0.02$ sec

∴ $T_{square\ form} = 20$ m sec

$T_{square\ form} = T_{on} + T_{off}$ & Ton = Toff

∴ $T_{on} = T_{off} = T_{square\ form}/2 = 20/2 = 10$ m sec

∴ delay = 10 m sec selected by you ↑

Timer 1/H = $2^N - ((delay \times Fosc)/(4 \times Prescaler))$

= $2^{16} - ((10 \times 10^{-3} \times 4 \times 10^6)/(4 \times 1)) = 55536$

Positive value without any fraction

∴ to generate a square form with 50 Hz you must initialize Timer 1/H with 55536 where Timer 1/H is Counter Register.

Page:..... Date:.....

note
= when the frequency decreased the time increased. $T = \frac{1}{F}$

Timer1 Module

The Timer1 timer/counter module incorporates these features:

- Software selectable operation as a 16-bit timer or counter
- Readable and writable 8-bit registers (TMR1H and TMR1L)
- Selectable clock source (internal or external) with device clock or Timer1 oscillator internal options
- Interrupt-on-overflow
- Reset on CCP Special Event Trigger
- Device clock status flag (T1RUN)

shown in Figure 12-1. A block diagram of the module's operation in Read/Write mode is shown in Figure 12-2.

The module incorporates its own low-power oscillator to provide an additional clocking option. The Timer1 oscillator can also be used as a low-power clock source for the microcontroller in power-managed operation.

Timer1 can also be used to provide Real-Time Clock (RTC) functionality to applications with only a minimal addition of external components and code overhead.

Timer1 is controlled through the T1CON Control register (Register 12-1). It also contains the Timer1 Oscillator Enable bit (T1OSCEN). Timer1 can be enabled or disabled by setting or clearing control bit TMR1ON (T1CON<0>).

REGISTER 12-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

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

bit 7 **RD16: 16-Bit Read/Write Mode Enable bit**
1 = Enables register read/write of Timer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations

bit 6 **T1RUN: Timer1 System Clock Status bit**
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS1:T1CKPS0: Timer1 Input Clock Prescale Select bits**
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN: Timer1 Oscillator Enable bit**
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **T1SYNC: Timer1 External Clock Input Synchronization Select bit**
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS: Timer1 Clock Source Select bit**
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge only)
0 = Internal clock (Fosc/4)

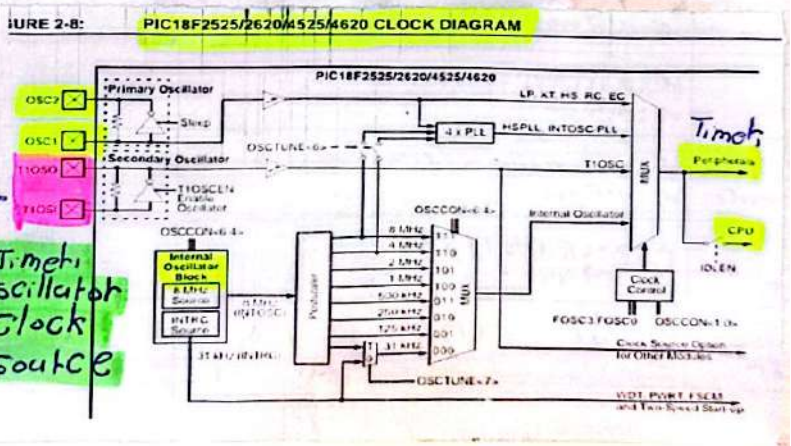
bit 0 **TMR1ON: Timer1 On bit**
1 = Enables Timer1
0 = Stops Timer1

note
= timer1 is just 16 bit Counter Register has three Clock Sources:
→ internal → Fosc
→ external → RIP
→ Timer1 Oscillator
↓
if you want to operate the MC at the low Power Mode.
where: RIP
Raising / Falling edge

note
= if you use external clock source at timer1 it will work at the raising edge only

Timer1 Oscillator

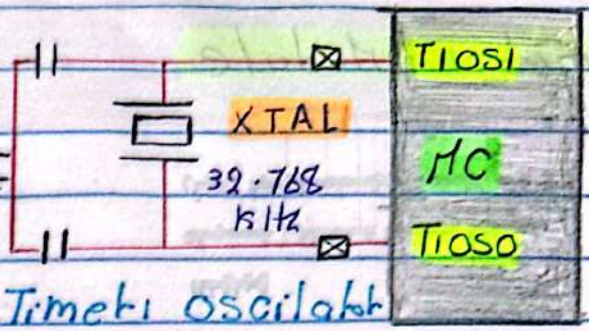
note when the
= Clock Sources increases the Power Consumption increase



Page:..... Date:.....

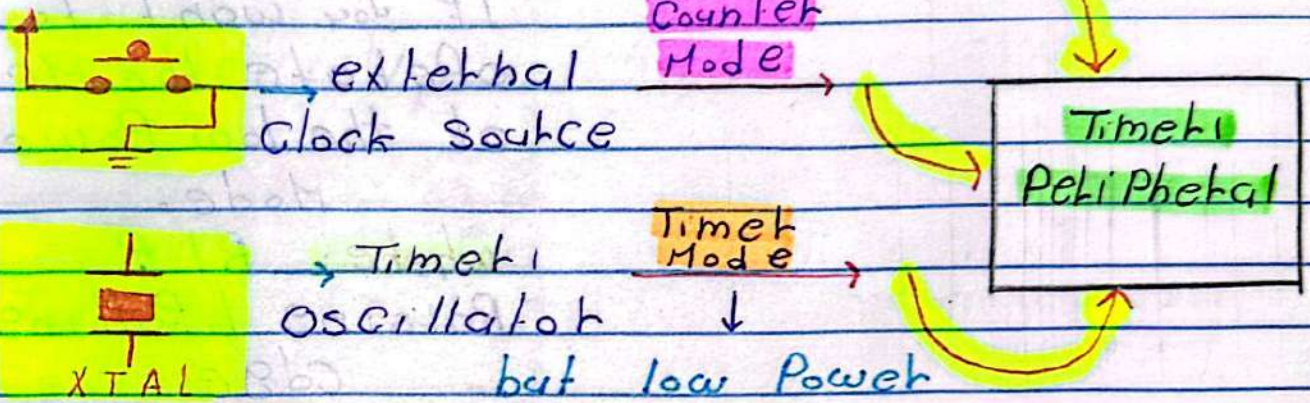
Note

If you want to give to your MC low frequency to make the MC at the low Power Mode use the timer1 oscillator is a crystals circuit for 32 kHz crystals.



MicroController low Power ← Timer1 oscilator

Timer1 Clock Sources :



Note

we use the low Power Mode if the system APPlication works at a battery.

Note

If you want to reduse the Power Consamption use the timer1 oscillator Clock Source Mode.

hote

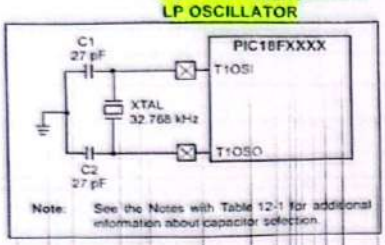
at timer1 I don't have bit to disable the Prescaler but I have a Prescaler Value 0:0 → Fosc/1 this like disable the Prescaler

12.3 Timer1 Oscillator

An on-chip crystal oscillator circuit is incorporated between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting the Timer1 Oscillator Enable bit, T1OSCEN (T1CON<3>). The oscillator is a low-power circuit rated for 32 kHz crystals. It will continue to run during all power-managed modes. The circuit for a typical LP oscillator is shown in Figure 12-3. Table 12-1 shows the capacitor selection for the Timer1 oscillator.

The user must provide a software time delay to ensure proper start-up of the Timer1 oscillator.

FIGURE 12-3: EXTERNAL COMPONENTS FOR THE TIMER1 LP OSCILLATOR



Note: See the Notes with Table 12-1 for additional information about capacitor selection.

TABLE 12-1: CAPACITOR SELECTION FOR THE TIMER OSCILLATOR

Osc Type	Freq	C1	C2
LP	32 kHz	27 pF(1)	27 pF(1)

Note 1: Microchip suggests these values as a starting point in validating the oscillator circuit.
2: Higher capacitance increases the stability of the oscillator but also increases the start-up time.
3: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.
4: Capacitor values are for design guidance only.

12.3.1 USING TIMER1 AS A CLOCK SOURCE

The Timer1 oscillator is also available as a clock source in power-managed modes. By setting the clock select bits, SCS1:SCS0 (OSCCON<1:0>), to '01', the device switches to SEC_RUN mode; both the CPU and peripherals are clocked from the Timer1 oscillator. If the IDLEN bit (OSCCON<7>) is cleared and a SLEEP instruction is executed, the device enters SEC_IDLE mode. Additional details are available in **Section 3.0 "Power-Managed Modes"**.

Whenever the Timer1 oscillator is providing the clock source, the Timer1 system clock status flag, T1RUN (T1CON<6>), is set. This can be used to determine the controller's current clocking mode. It can also indicate the clock source being currently used by the Fail-Safe Clock Monitor. If the Clock Monitor is enabled and the Timer1 oscillator fails while providing the clock, polling the T1RUN bit will indicate whether the clock is being provided by the Timer1 oscillator or another source.

12.3.2 LOW-POWER TIMER1 OPTION

The Timer1 oscillator can operate at two distinct levels of power consumption based on device configuration. When the LPT1OSC Configuration bit is set, the Timer1 oscillator operates in a low-power mode. When LPT1OSC is not set, Timer1 operates at a higher power level. Power consumption for a particular mode is relatively constant, regardless of the device's operating mode. The default Timer1 configuration is the higher power mode.

As the low-power Timer1 mode tends to be more sensitive to interference, high noise environments may cause some oscillator instability. The low-power option is, therefore, best suited for low noise applications where power conservation is an important design consideration.

*Timer1 oscillator section at the Data sheet.

Timer1 Modes

→ Timer Mode : Internal Clock Source.

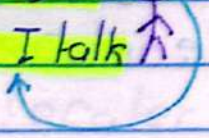
→ Synchronous Counter Mode: external Clock Source with timer1 oscillator.

→ Asynchronous Counter Mode: external Clock Source.

what's the Meaning of Synchronous

happening or existing at the same time

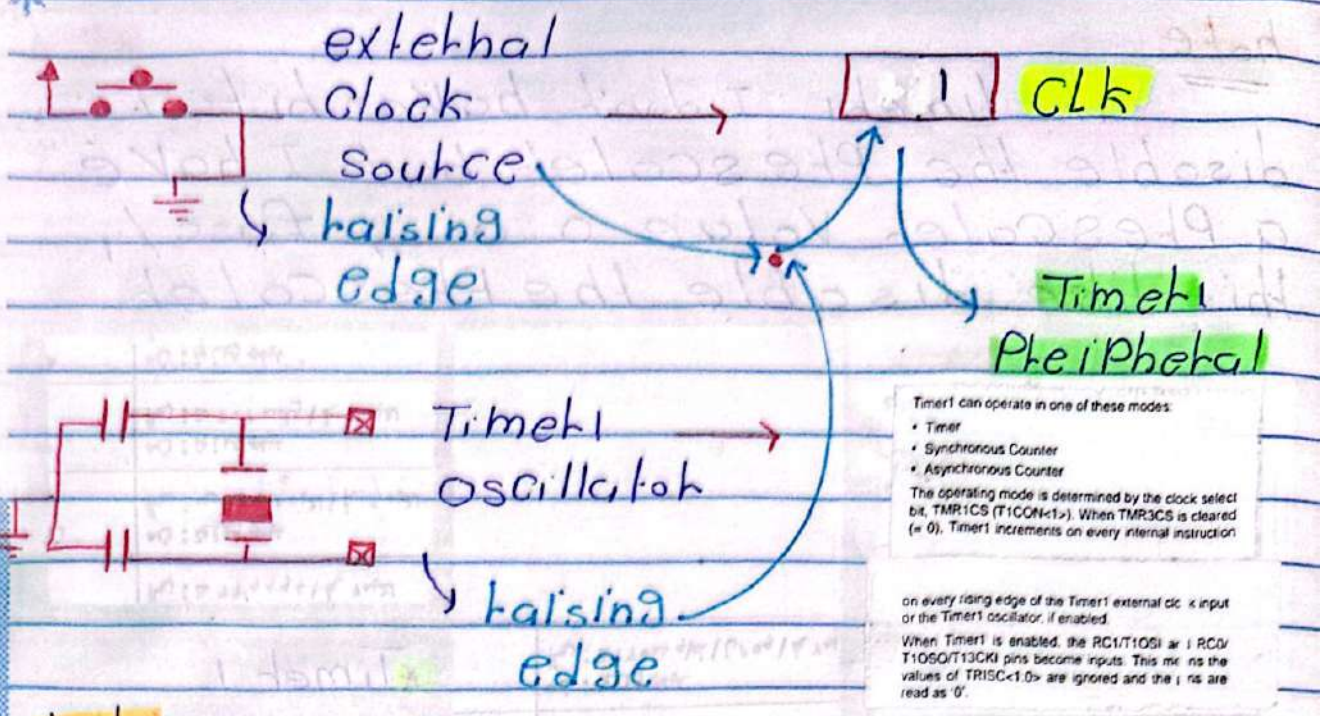
when you talk I talk



hote

the Counter Synchronous Mode here is related with the timer1 oscillator

Page: Synchronous Counter Mode Date:



Timer1 can operate in one of these modes:
- Timer
- Synchronous Counter
- Asynchronous Counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>). When TMR3CS is cleared (= 0), Timer1 increments on every internal instruction

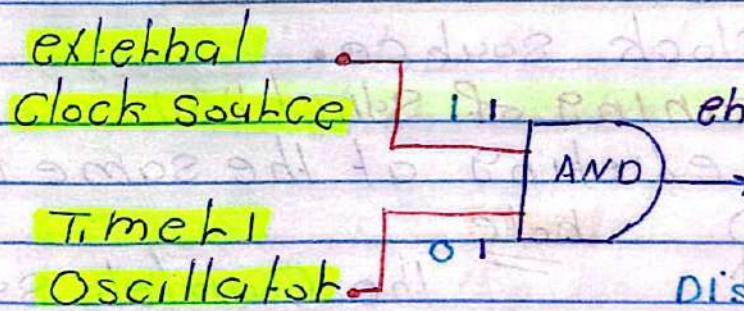
on every rising edge of the Timer1 external clock input or the Timer1 oscillator, if enabled.

When Timer1 is enabled, the RC1/T1OSI and RC0/T1OSO/T13CKI pins become inputs. This means the values of TRISC<1:0> are ignored and the pins are read as '0'.

hole

→ if the external clock source generates falling edge & the timer1 oscillator generates falling edge this means there's a lic clock.

→ if the external clock source generates falling edge & Timer1 oscillator generates falling edge this means there's no lic clock.



Page:..... Date:.....

Show timer1 block digram is only act with 16 bit Mode

12.2 Timer1 16-Bit Read/Write Mode

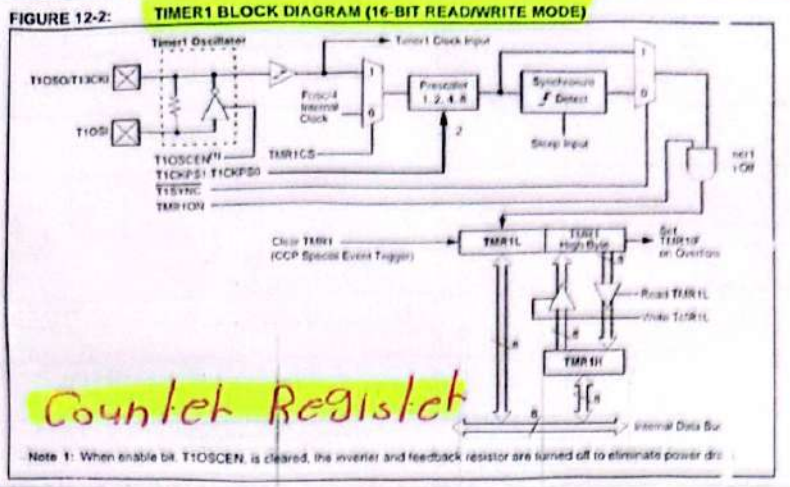
Timer1 can be configured for 16-bit reads and writes (see Figure 12-2). When the RD16 control bit (T1CON<7>) is set, the address for TMR1H is mapped to a buffer register for the high byte of Timer1. A read from TMR1L will load the contents of the high byte of Timer1 into the Timer1 high byte buffer. This provides the user with the ability to accurately read all 16 bits of Timer1 without having to determine whether a read of the high byte, followed by a read of the low byte, has become invalid due to a rollover between reads.

A write to the high byte of Timer1 must also take place through the TMR1H Buffer register. The Timer1 high byte is updated with the contents of TMR1H when a write occurs to TMR1L. This allows a user to write all 16 bits to both the high and low bytes of Timer1 at once.

The high byte of Timer1 is not directly readable or writable in this mode. All reads and writes must take place through the Timer1 High Byte Buffer register. Writes to TMR1H do not clear the Timer1 prescaler. The prescaler is only cleared on writes to TMR1L.

How to R/w at Counter Register

FIGURE 12-2: TIMER1 BLOCK DIAGRAM (16-BIT READ/WRITE MODE)



the same thing at Timer0

the equation of delay Timer1

TIMER1 L/H = 2^n - ((delay * Fosc) / (4 * Prescaler))

the same equation at Timer0

Registers are related to the Timer1 Module for the Interrupt & the Poriority of the Interrupt.

TABLE 12-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	52
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	52
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IF	52
TMR1L	Timer1 Register Low Byte								50
TMR1H	Timer1 Register High Byte								50
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	50

Legend: Shaded cells are not used by the Timer1 module.

Timer2 Module

this timer is 8 bit only & works as timer Mode only used with PWM has two types of the Prescaler :-

- Prescaler
- Postscaler

the different will be at the delay equation

The Timer2 module timer incorporates the following features:

- 8-bit Timer and Period registers (TMR2 and PR2, respectively)
- Readable and writable (both registers)
- Software programmable prescaler (1:1, 1:4 and 1:16)
- Software programmable postscaler (1:1 through 1:16)
- Interrupt on TMR2 to PR2 match
- Optional use as the shift clock for the MSSP module

The module is controlled through the T2CON register (Register 13-1), which enables or disables the timer and configures the prescaler and postscaler. Timer2 can be shut off by clearing control bit, TMR2ON (T2CON<2>), to minimize power consumption.

A simplified block diagram of the module is shown in Figure 13-1.

In normal operation, TMR2 is incremented from 00h on each clock (FOSC/4). A 4-bit counter/prescaler on the clock input gives direct input, divide-by-4 and divide-by-16 prescale options; these are selected by the prescaler control bits, T2CKPS1:T2CKPS0 (T2CON<1:0>). The value of TMR2 is compared to that of the Period register, PR2, on each clock cycle. When the two values match, the comparator generates a match signal as the timer output. This signal also resets the value of TMR2 to 00h on the next cycle and drives the output counter/postscaler (see Section 13.2 "Timer2 Interrupt").

The TMR2 and PR2 registers are both directly readable and writable. The TMR2 register is cleared on any device Reset, while the PR2 register initializes at FFh. Both the prescaler and postscaler counters are cleared on the following events:

- 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.

Page: ..... Date: .....

TIMR2 = 2^N - ((delay * Fosc) / (4 * Prescaler or * Postscaler))

↳ the Counter Register is only 8 bit there's no low & High register.

the Control register

REGISTER 13-1: 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
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

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

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

bit 6-3 **T2OUTPS3:T2OUTPS0:** Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

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

note

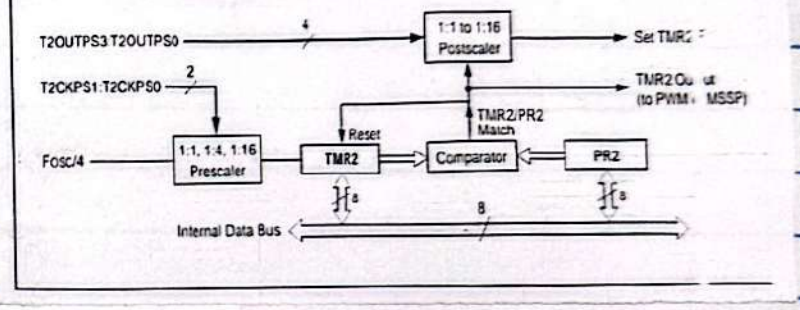
at the Prescaler & Postscaler you sleate what you want to deride on it

$\frac{D * Fosc}{4 * Pre}$ or $\frac{D * Fosc}{4 * Post}$

Timer2 block digram

* as you see is just work on the Fosc is the internal clock source so is works only as timer Mode operation.

FIGURE 13-1: TIMER2 BLOCK DIAGRAM



* these registers is related to timer2 Module to Make the interrupt feature & set the low & high Priority levels feature

TABLE 13-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Value on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1I	52
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1I	52
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1I	52
TMR2	Timer2 Register								50
T2CON	—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS	50
PR2	Timer2 Period Register								50

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.
Note 1: These bits are unimplemented on 28-pin devices and read as '0'.

Page: ..... Date: .....

Timer3 Module

14.0 TIMER3 MODULE

The Timer3 module timer/counter incorporates these features:

- Software selectable operation as a 16-bit timer or counter
- Readable and writable 8-bit registers (TMR3H and TMR3L)
- Selectable clock source (internal or external) with device clock or Timer1 oscillator internal options
- Interrupt-on-overflow
- Module Reset on CCP Special Event Trigger

A simplified block diagram of the Timer3 module is shown in Figure 14-1. A block diagram of the module's operation in Read/Write mode is shown in Figure 14-2.

The Timer3 module is controlled through the T3CON register (Register 14-1). It also selects the clock source options for the CCP modules (see Section 15.1.1 "CCP Modules and Timer Resources" for more information).

→ May be timer & Counter Mode is just 16 bit Counter register timer3 L/H.

→ May be is internal or external clock source with timer1 oscillator clock source.

the Control register

REGISTER 14-1: T3CON: TIMER3 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	T3SYNC	TMR3CS	TMR3ON
bit 7							bit 0

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

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for the CCP modules
01 = Timer3 is the capture/compare clock source for CCP2;
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for the CCP modules

CCP Module

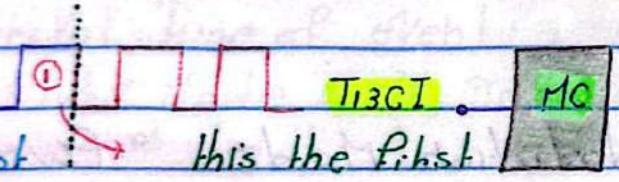
bit 5-4 **T3CKPS1:T3CKPS0:** Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **T3SYNC:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CKI (on the rising edge after the first falling edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

hote
the external clock source on the rising edge after the first falling edge of T13CKI this means the first rasing edge is not Count to the register Counter after this raising edge the falling edge after that the raising edge this raising edge will be Count to the Counter registe



this is not Count to the Counter register

this the first raising edge

hote
the explaining at bit2 is the same at timer1 Counter Mode synchronous at not

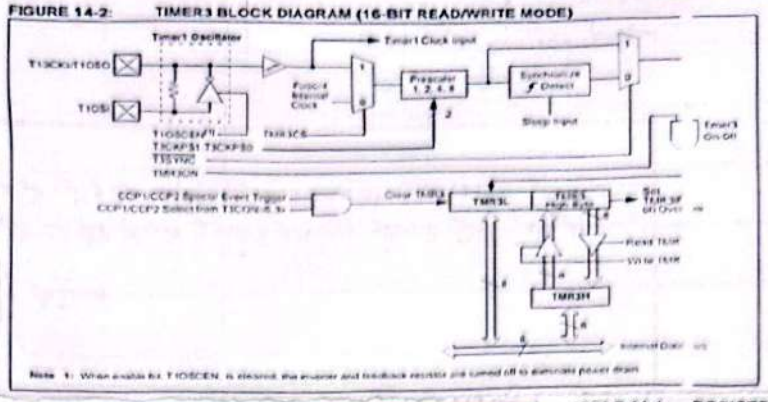
14.1 Timer3 Operation
Timer3 can operate in one of three modes:
- Timer
- Synchronous Counter
- Asynchronous Counter

the equation delay for timer3

timer3 L/H = 2^N - ((delay * Fosc) / (4 * Ptescaler))

Page:..... Date:.....

FIGURE 14-2: TIMER3 BLOCK DIAGRAM (16-BIT READ/WRITE MODE)



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off to eliminate power drain.

* this block digram to show the operation of the timer 3 Module.

* these Register are related to the timer 3 Module.

TABLE 14-1: REGISTERS ASSOCIATED WITH TIMER3 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
PIR2	OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	52
PIE2	OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	52
IPR2	OSCFIP	CMIP	—	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP	52
TMR3L	Timer3 Register Low Byte								51
TMR3H	Timer3 Register High Byte								51
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	50
T3CON	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	51

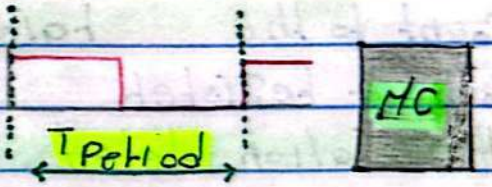
Legend: — = unimplemented, read as '0'. Shaded cells are not used by the Timer3 module.

CCP Module CaPture - ComPare - PWM Module

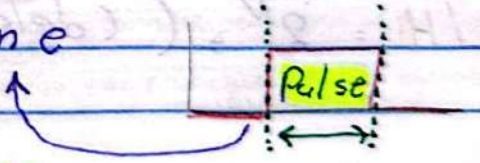
* at Pic18F46K20 MCUs have two CCP Module:
  - → Normal CCP Module.
  - → Enhanced CCP Module.
* each Module has three Modes and each Mode can be used for Many application:
  - → CaPture Mode.
  - → ComPare Mode.
  - → Pulse width Modulation Mode "PWM".

the CaPture Mode:

- → Measuring Period time.
- → Measuring the Pulse width



what's the Pulse width
the high Period of time

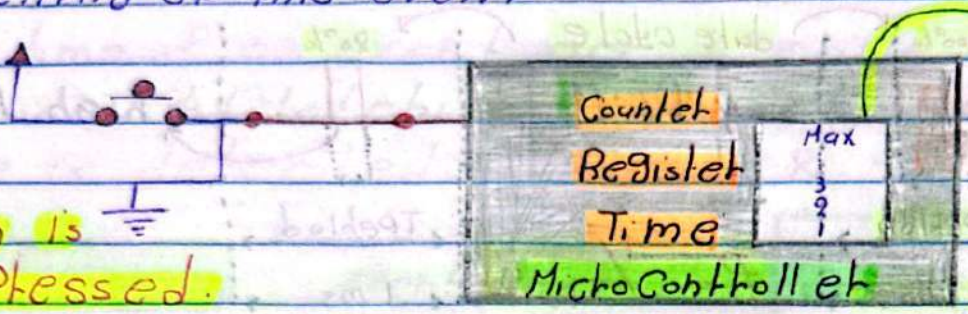


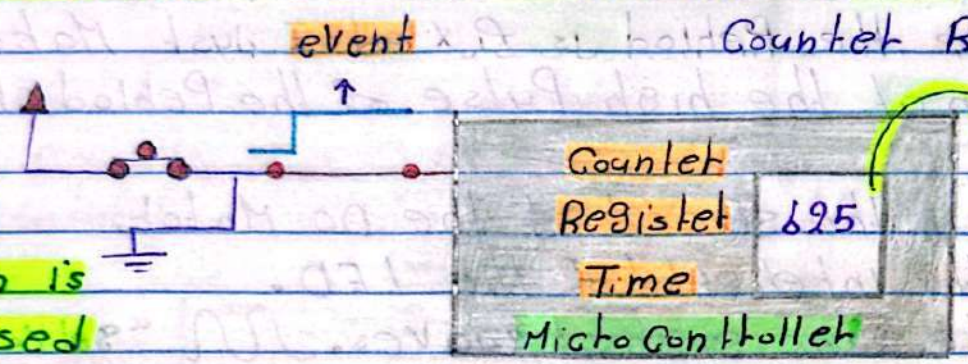
Page: ..... Date: .....

→ Capture the arrival time of an event

this means

If you connect push button or any thing makes raising or falling edge & the timer counts time when the push button is pressed or the MC receives raising edge or falling edge the capture mode will get the value of time at happening of this event





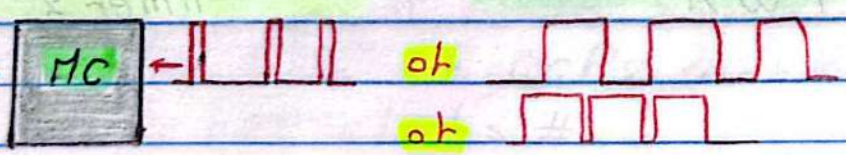
∴ the arrival time of event = 625 Register Value

∴ the Capture Mode will get it #

the Compare Mode

→ Generate a wave form of various duty cycles like PWM. this means: I generate specific wave at a Pin of the Micro Controller.

according to the Application



Page: ..... Date: .....

→ Generate specific time delay:-
May be I toggle LED after a delay of time.
→ Trigger an event when the pre-determined time expires:-
May be after each half hour I go to do a task.

the Pulse width Modulation Mode

* I Control in the width of the time of high Pulse without changing in the Period time

* as you see the Period is fixed I just Make the Modulation at the high Pulse of the Period only.

→ Control in the speed of the DC Motor.
→ Change the intensity of the LED.
→ Generate sine signal waves. ∿∿ "Servo Motor

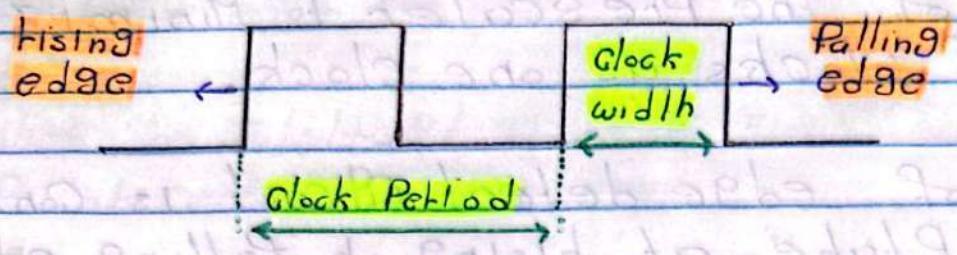
hote
= all Modules of the CCP work with the timers.

CCP - ECCP Mode	Timer resources
Capture	Timer1 or Timer3
Compare	Timer1 or Timer3
PWM	Timer2

Page: ..... Date: .....

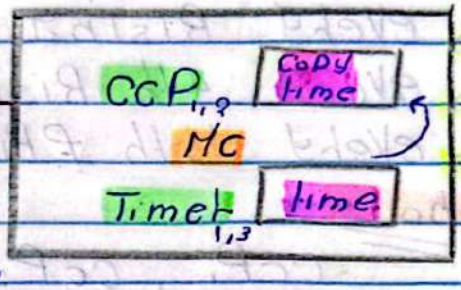
the CaPture Mode exPalination

→ an event Can beRePresented by the rising & falling edge of a Pulse.

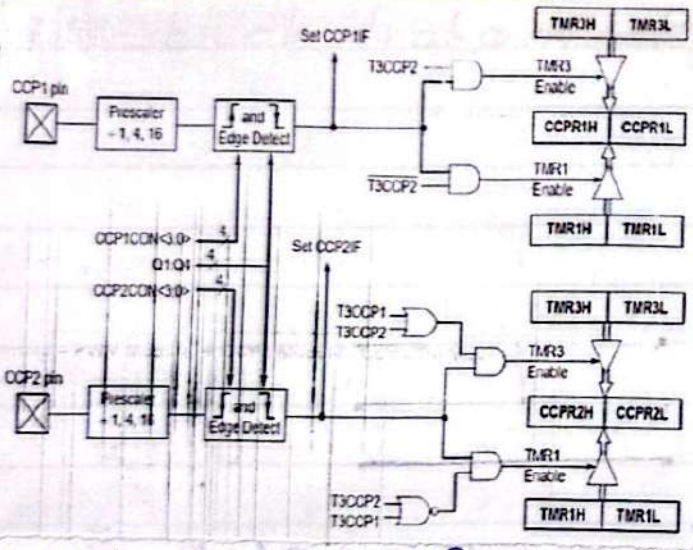


→ the time of an event occurrence Can be recorded by latching the Count when an edge is detected : Rising & Falling edge

Rising or Falling event → CCP1,2 Pin



* In CaPture Mode the InPut Pulse should be Connected to CCP1,2 which is MultiPlexed with PorTc (RC2)



CaPture Mode block diagram

note

the registers of the CaPture Mode are low & high as one register is 16 bit as it's existed at Timer1 & Timer3 Module

* as you see CCP1 works with timer1 & timer3 with the ccp1 Registers & CCP2 works with them too with it's registers #.

Page:..... Date:.....

* according to your Configration you select the Capture Mode works with timer 1 or

* the role of the Prescaler to Minimize a group of clocks as one clock.

* the role of edge detect circuit is Confige I will Capture at rising or falling edge event.

as event is defined as one of the following

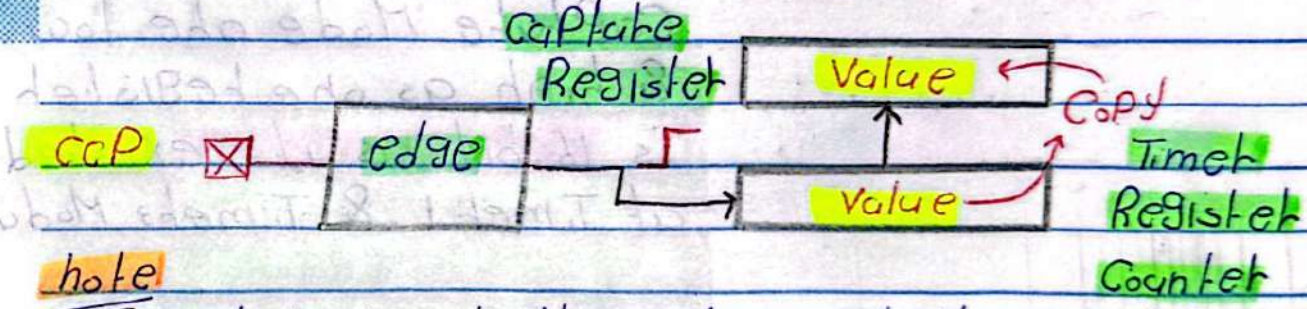
→ every falling edge Fosc/1.
→ every Rising edge Fosc/1.
→ every 4th Rising edge Fosc/4.
→ every 16th Rising edge Fosc/16.

Note

CCP1, CCP2 Must be Input to receive the sent edge to the MicroControlleR.

Note

if another Capture occurs before the Va in register CCPR1 is read the old Capture Value is over written by the new Capture Value.



Note

You Must read the Value at the Capture register before the next edge.

Page: ..... Date: .....

the Compare Mode explination

it Compare the value of CCPR register with the timer register & when the values of both registers are Matched it Make an Interrupt.

note

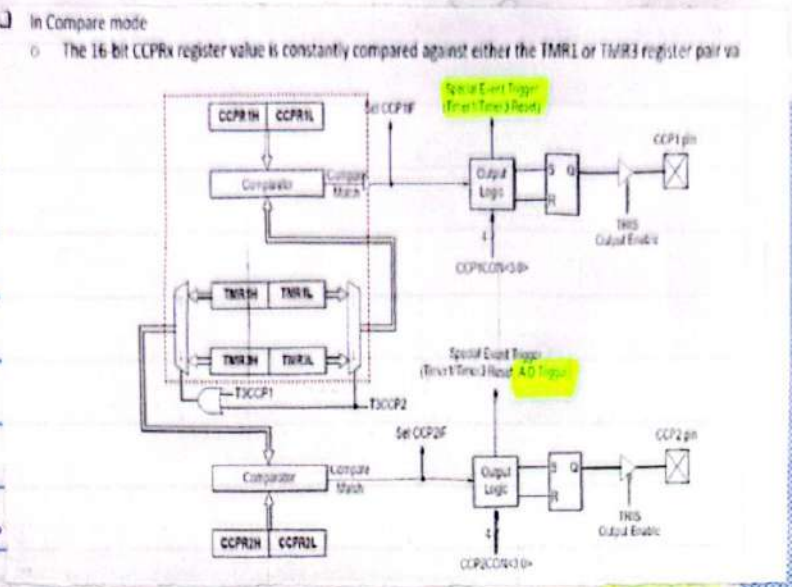
when the Interrupt is done Make an event on CCP1 or CCP2 Pins

note

You Must Configure Pins CCP1 & CCP2 Pins work as out Put Direction Mode.

note

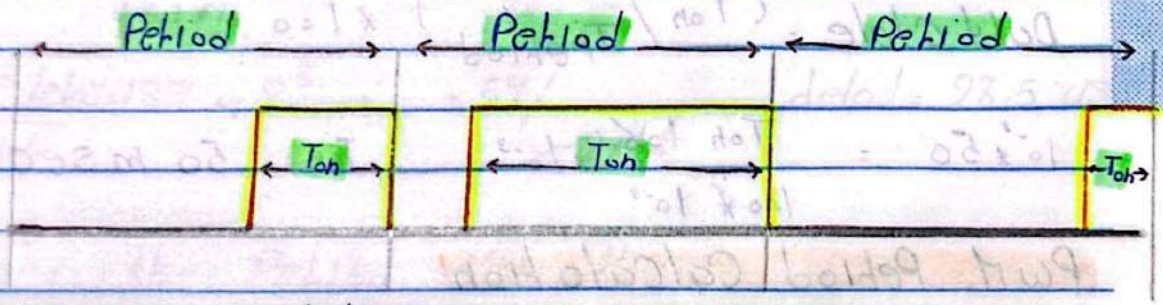
You can start ADC Conversion by the Compare Mode.



the PWM Module explination

is defined as Pulse width Modulation.

→ it Controls in the time of an Pulse without changing at the time of the Period at the Frequency



what's the Duty cycle

is the Precentage of the on Pulse time Per the total Period of time. Duty cycle = (Ton / T Period) * 100

Page:..... Date:.....

Ex → if we have a Pulse width a Period of 10 ms and the on Pulse is 2 ms. what's the duty cycle?

Duty cycle = $(T_{on} / T_{Period}) * 100 = \left(\frac{2 * 10^{-3}}{10 * 10^{-3}}\right) * 100 = 20\%$

Note if you want to know the Voltage at the Pin:

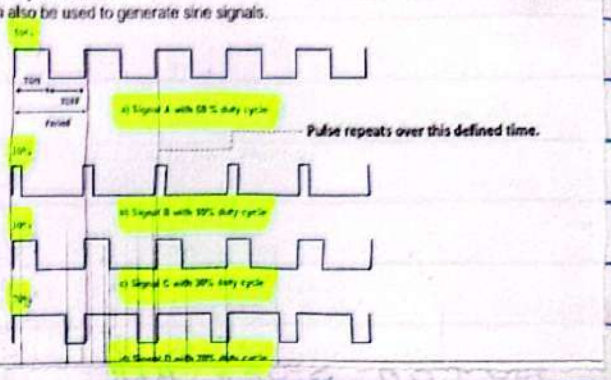
$V_{Pin} = \frac{\text{Duty cycle}}{100} * 5$

where 5 is the Max Voltage at the Pin

Ex → Duty cycle = 70%

Pin Voltage = $\frac{70}{100} * 5 = 3.5\ V$

With the Pulse Width Modulation Mode:
- Control the power delivered to the load by using the ON-OFF signal. Control the speed of DC motors.
- Change the intensity of the LED.
- Moreover, it can also be used to generate sine signals.



Ex → assume we need to generate a PWM with a frequency 10 Hz & the duty cycle is 50%

$T_{Period} = 1/f = 1/10 = 100\ m\,sec$

$\text{Duty cycle} = (T_{on} / T_{Period}) * 100$

$10^{-3} * 50 = \frac{T_{on}}{100 * 10^{-3}} * 10^{-3} \rightarrow T_{on} = 50\ m\,sec$

PWM Period Calculation

$\text{PWM Period} = (PR2 + 1) * 4 * T_{OSC} * \text{Timer2 Prescaler Value}$

where

PR2 → Counter Register for Timer 2 & $T_{OSC} = 1/F_{OSC}$

Page:..... Date:.....

$PR_2 = (F_{osc} / (F_{PWM} * 4 * Timer_2\ Prescale\ Value)) - 1$

where

Timer2 Prescaler Value = Prescaler * Postscaler Values

F_{PWM} : PWM frequency.

note

the value of PR2 must be not over flow 256 because the size of PR2 is 8 bit. & (+ve) value.

ex →

we need to generate a PWM with 10kHz frequency assume clock frequency : 8 MHz Sol →

$F_{osc} = 8 * 10^6$ Hz

$F_{PWM} = 10 * 10^3$ Hz

suppose : Prescaler = 1

Postscaler = 1

$PR_2 = (8 * 10^6 / (10 * 10^3 * 4 * 1 * 1)) - 1 = 199$ → Initialize Value to PR2

$PR_2 = 2^N - ((delay * F_{osc}) / 4 * Timer2\ Prescale\ Value)$

$199 = 2^8 - \frac{8 * 10^6\ delay}{4 * 1 * 1}$ → $199 = 2^8 - 2 * 10^6\ delay$

$\frac{-2 * 10^6\ delay}{-2 * 10^6} = 199 - 2^8 = -59 / -2 * 10^6$ delay = 28.5 us

note

199 is the Period over all.

How to Calculate the Duty cycle of the PWM

in PWM Module the CCPx Pin Produces up to 10-bit resolution PWM output this means the Duty cycle will varies from 0 to $(2^N - 1)$ → 0 to 1023

$2^{10} - 1$

Page: ..... Date: .....

* The following equation is used to calculate the PWM duty cycle in time:

PWM Duty Cycle = (CCPRxL:CCPxCon<5:4>) * Tosc * Timer2 Prescale Value

↳ the resolution values from 0 to 1023

Note

How to write the resolution values at the register according to PIC18F46K20 data sheet

CCPRxL	CCPxCon 5	CCPxCon 4
11111100	0	1

↳ register ↳ Bits at another register

ex → 1009 : 1111110001 → CCPxCon4, CCPxCon5 (CCPRxL)

ex → Calculate the duty cycle resolution we need to generate a PWM with 10 kHz frequency & Duty Cycle is 20% assume the clock frequency is 8 MHz. Sol →

Note: 160 is 20% Ton of the Period

$F_{osc} = 8 \times 10^6$ Hz

$F_{PWM} = 10 \times 10^3$ Hz

Duty Cycle Percentage = 20%

$PR2 = (F_{osc} / (F_{PWM} * 4 * Timer2\ Prescaler\ Value))^{-1}$

$= (8 \times 10^6 / (10 \times 10^3 * 4 * 1 * 1)) - 1 = 199$

$(CCRxL:CCPxCon<5:4>) = (PR2+1) * \frac{Duty\ Cycle}{100} * 4$

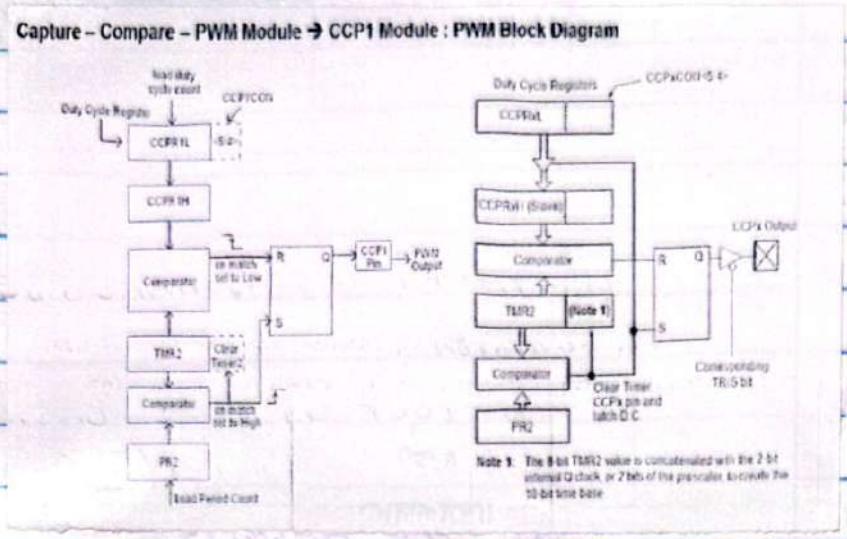
$= (199+1) * \frac{20}{100} * 4 = 160$ resolution value

Page: ..... Date: .....

the equation to calculate the duty cycle resolution:

$$(CCPRxL : CCPxCon<5:4>) = (PR2+1) \times \frac{\text{Duty cycle}}{100} \times 4$$

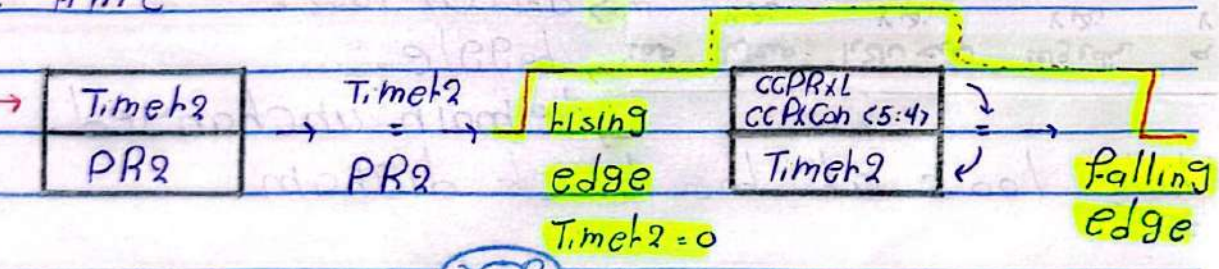
PWM block digram



→ firstly you write value at PR2 register to set the time of the period & TIMR2 counts values.

→ when PR2 = TIMR2 this means compare match will make rising edge at ccPx Pin & TIMR2 will be 0 & counts value again.

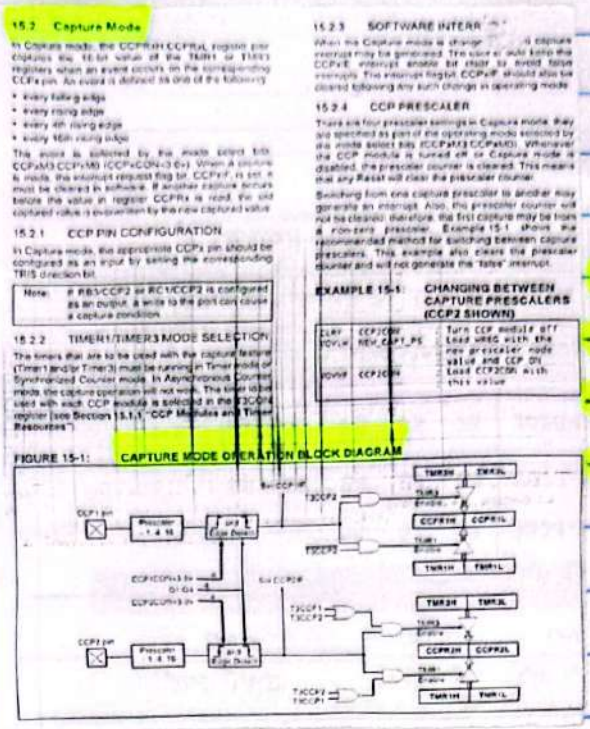
→ when the value at CCPRxL : CCPxCon<5:4> = Timer2 this means compare match will make falling edge at ccPx Pin & TimR2 will continue counts till the Max value & repeate this operation one more time



Page:..... Date:.....

CCP Module CaPture & ComPare & PWM at the Data sheet

PIC 18f 46 k 80



15.2 Capture Mode

In Capture mode, the CCPRxH:CCPRxL register pair captures the 16-bit value of the TMR1 or TMR3 registers when an event occurs on the corresponding CCPx pin. An event is defined as one of the following:

- every falling edge
- every rising edge
- every 4th rising edge
- every 16th rising edge

The event is selected by the mode select bits CCPxM3:CCPxM0 (CCPxCON<3:0>). When a capture is made, the interrupt request flag bit, CCPxIF, is set. It must be cleared in software. If another capture occurs before the value in register CCPRx is read, the old captured value is overwritten by the new captured value.

15.2.1 CCP PIN CONFIGURATION

In Capture mode, the appropriate CCPx pin should be configured as an input by setting the corresponding TRIS direction bit.

Note: If RB3/CCP2 or RC1/CCP2 is configured as an output, a write to the port can cause a capture condition.

15.2.2 TIMER1/TIMER3 MODE SELECTION

The timers that are to be used with the capture feature (Timer1 and/or Timer3) must be running in Timer mode or Synchronized Counter mode. In Asynchronized Counter mode, the capture operation will not work. The timer to be used with each CCP module is selected in the T3CON register (see Section 13.1.1 "CCP Modules and Timer Resources").

15.2.3 SOFTWARE INTERR...

When the Capture mode is change... a capture interrupt may be generated. The user should keep the CCPxIE interrupt enable bit clear to avoid false interrupts. The interrupt flag bit CCPxIF should also be cleared following any such change in operating mode.

15.2.4 CCP PRESCALER

There are four prescaler settings in Capture mode. They are specified as part of the operating mode selected by the mode select bits (CCPxM3:CCPxM0). Whenever the CCP module is turned off or Capture mode is disabled, the prescaler counter is cleared. This means that any Reset will clear the prescaler counter.

Switching from one capture prescaler to another may generate an interrupt. Also, the prescaler counter will not be cleared; therefore, the first capture may be from a non-zero prescaler. Example 15-1 shows the recommended method for switching between capture prescalers. This example also clears the prescaler counter and will not generate the "false" interrupt.

EXAMPLE 15-1: CHANGING BETWEEN CAPTURE PRESCALERS (CCP2 SHOWN)

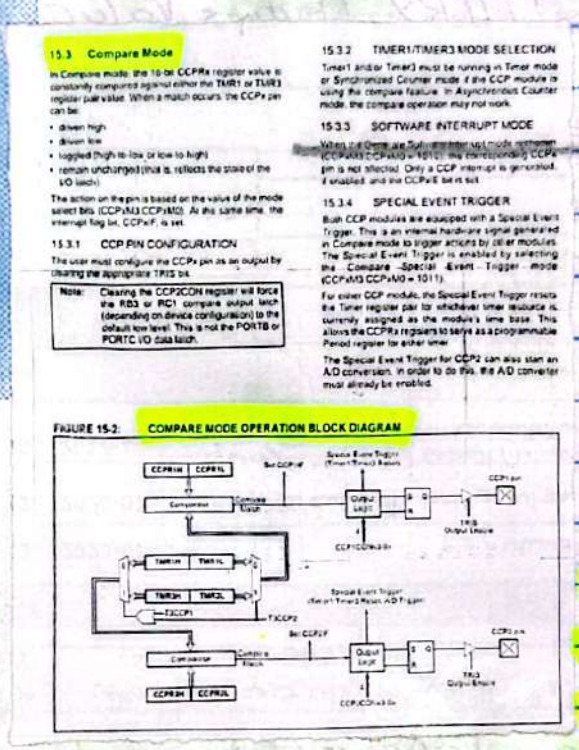
```
CLRF    CCP2CON     ; Turn CCP module off
MOVLW   NEW_CAPT_PS ; Load WREG with the
                    ; new prescaler mode
                    ; value and CCP ON
MOVWF   CCP2CON     ; Load CCP2CON with
                    ; this value
```

FIGURE 15-1: CAPTURE MODE OPERATION BLOCK DIAGRAM

→ the CaPture Mode it works with timer 1 & 3 & it caPture the value at the timer Reg at some cases as you see:

→ every falling edge.
→ every rising edge.
→ every 4th rising edge.
→ every 16th rising edge.

hote
look at the block digham of the caPture Mode to know all the details.



15.3 Compare Mode

In Compare mode, the 16-bit CCPRx register value is constantly compared against either the TMR1 or TMR3 register pair value. When a match occurs, the CCPx pin can be:

- driven high
- driven low
- toggled (high-to-low or low-to-high)
- remain unchanged (that is, reflects the state of the I/O latch)

The action on the pin is based on the value of the mode select bits (CCPxM3:CCPxM0). At the same time, the interrupt flag bit, CCPxIF, is set.

15.3.1 CCP PIN CONFIGURATION

The user must configure the CCPx pin as an output by clearing the appropriate TRIS bit.

Note: Clearing the CCP2CON register will force the RB3 or RC1 compare output latch (depending on device configuration) to the default low level. This is not the PORTB or PORTC I/O data latch.

15.3.2 TIMER1/TIMER3 MODE SELECTION

Timer1 and/or Timer3 must be running in Timer mode or Synchronized Counter mode if the CCP module is using the compare feature. In Asynchronous Counter mode, the compare operation may not work.

15.3.3 SOFTWARE INTERRUPT MODE

When the CCPx are Software Interrupt mode (CCPxM3:CCPxM0 = 1010), the corresponding CCPx pin is not affected. Only a CCP interrupt is generated, if enabled and the CCPxIE bit is set.

15.3.4 SPECIAL EVENT TRIGGER

Both CCP modules are equipped with a Special Event Trigger. This is an internal hardware signal generated in Compare mode to trigger actions by other modules. The Special Event Trigger is enabled by selecting the Compare -Special Event - Trigger - mode (CCPxM3:CCPxM0 = 1011).

For either CCP module, the Special Event Trigger resets the Timer register pair for whichever timer resource is currently assigned as the module's time base. This allows the CCPRx registers to serve as a programmable Period register for either timer.

The Special Event Trigger for CCP2 can also start an A/D conversion. In order to do this, the A/D converter must already be enabled.

FIGURE 15-2: COMPARE MODE OPERATION BLOCK DIAGRAM

→ the ComPare Mode it works with timer1 & timer3 it ComPares the value at the register timer with the value at it's register & at the Matching ComPare Case takes an action as you see at the data sheet get out specific logic at CCPx Pins:

→ driven high.
→ driven low.
→ toggle.
→ remain unchanged.

hote look at the block digham

Page: ..... Date: .....

PWM

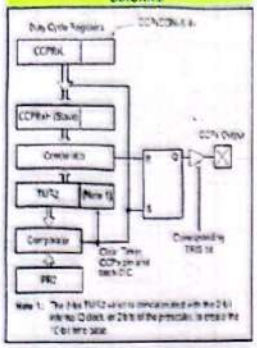
In Pulse-Width Modulation (PWM) mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with a PORTB or PORTC data latch, the appropriate TRIS bit must be cleared to make the CCP2 pin an output.

> **Note:** Clearing the CCP2CON register will force the RB3 or RC1 output latch (depending on device configuration) to the default low level. This is not the PORTB or PORTC I/O data latch.

Figure 15-3 shows a simplified block diagram of the CCP module in PWM mode.

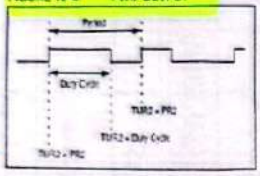
For a step-by-step procedure on how to set up the CCP module for PWM operation, see **Section 15.4.4 "Setup for PWM Operation"**.

FIGURE 15-3: SIMPLIFIED PWM BLOCK DIAGRAM



A PWM output (Figure 15-4) 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 15-4: PWM OUTPUT



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

EQUATION 15-1:

$$\text{PWM Period} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

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 CCPx pin is set (exception: if PWM duty cycle = 0%, the CCPx pin will not be set)
- The PWM duty cycle is latched from CCPRxL into CCPRxH

> **Note:** The Timer2 postscalers (see **Section 13.0 "Timer2 Module"**) are 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.

15.4.2 PWM DUTY CYCLE

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

EQUATION 15-2:

$$\text{PWM Duty Cycle} = (CCPRxL:CCPxCON<5:4>) \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

CCPRxL and CCPxCON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR2H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). In PWM mode, CCPRxH is a read only register.

hote

look at the block digram

the PWM Mode it wokks with timer2 only & it's Conttol in the time of Puls is on without change in the total Pertod as you see at the block digram of the PWM PR2 registert to set the total Pertod of time & CCPRxL registert & 2 bits CCPx Con <5:4> to set the duty cycle Precentage to Conttol in the time of on Pulse the value of PR2 registert is not changed duting the run time but you Can change at the duty cycle

TABLE 15-2: INTERACTIONS BETWEEN CCP1 AND CCP2 FOR TIMER RESOURCES

CCP1 Mode	CCP2 Mode	Interaction
Capture	Capture	Each module can use TMR1 or TMR3 as the time base. The time base can t different for each CCP.
Capture	Compare	CCP2 can be configured for the Special Event Trigger to reset TMR1 or TMF (depending upon which time base is used). Automatic A/D conversions on tr ger event can also be done. Operation of CCP1 could be affected if it is using the sam timer as a time base.
Compare	Capture	CCP1 can be configured for the Special Event Trigger to reset TMR1 or TMF (depending upon which time base is used). Operation of CCP2 could be affe ed if it is using the same timer as a time base.
Compare	Compare	Either module can be configured for the Special Event Trigger to reset the tir base. Automatic A/D conversions on CCP2 trigger event can be done. Conflicts m occur if both modules are using the same time base.
Capture	PWM(1)	None
Compare	PWM(1)	None
PWM(1)	Capture	None
PWM(1)	Compare	None
PWM(1)	PWM	Both PWMs will have the same frequency and update rate (TMR2 interrupt)

Note 1: Includes standard and Enhanced PWM operation.

* these the Instruction Cases for the CCP1 & CCP2 Pins at the different Modes:
  - → Capture Mode.
  - → Compare Mode.
  - → PWM Mode.

Page:..... Date:.....

the Communication Protocols

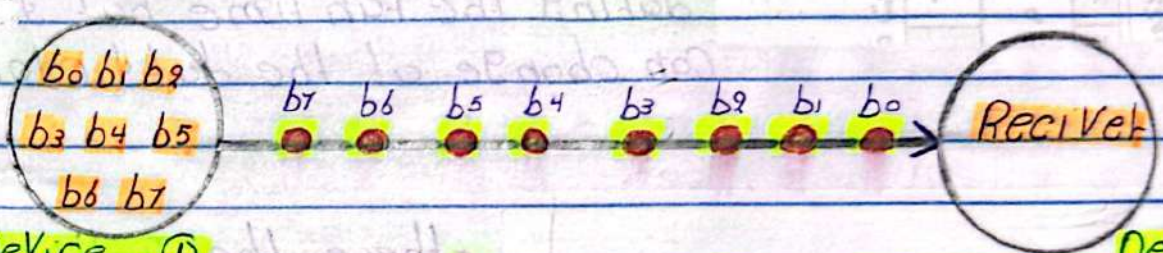
is a specific way to communicate between 2 devices & these devices can understand each other.

hole
May be you connect MC with MC or MC with external device & set the suitible COM Protocol for it.

the types of the communication

→ Serial Communication wired

* the data will be sent bit by bit, one bit on a time.
* it's called sequentially transmission.
* needs one channel or one wire for transmitting.



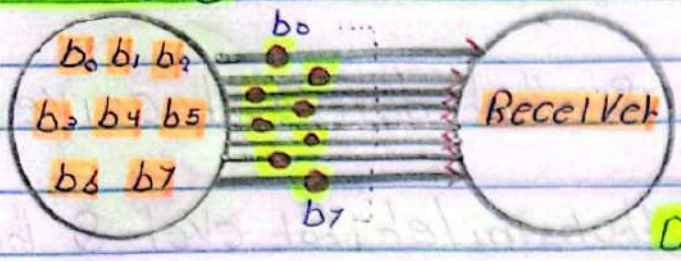
* ex: UART - I2C - SPI - USB - FlexRay - CAN - MOST - LIN.

→ Parallel Communication wired

* More than one bit can be transmitted at the same time
* needs more than one channel or wires for the transmission.
* ex: PCI bus «PeriPheral ComPonent interConnect» Connects the CPU and exPansion boards like Modem Cards & network Cards & Sound Cards.

Page: ..... Date: .....

Device ①



Device ②

note

as you see the Data is sent at the same time

Ex: for the wireless communication this is done by: IR "Infrared" & RF "Radio frequency".

Ex:

Bluetooth - wifi - ZigBee - IEEE 802.11 - GSM - 2G 4G - 3G ... ect.

what's the advantage & disadvantage of the serial & parallel communication

Serial Communication

advantage	disadvantage
one wire to send or receive the Data no hard ware complexity	slow sending or receiving the data one channel for the Data

Parallel Communication

advantage	disadvantage
fast sending or receiving Data it has more than one channel for the Data	high hard ware complexity it has more than one channel for the Data

what's the Communcation system

the Communcation system it describes the connection way of the communication protocol & it's types are:

simplex & half duplex & full duplex

Page:..... Date:.....

the Communication System types :-

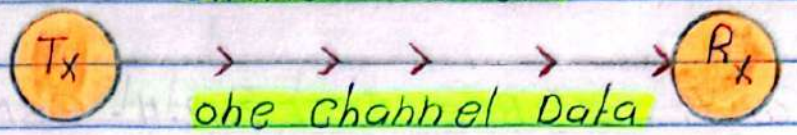
→ **Simplex :**

* one device transmits & other devices receive

note

the transmitter is transmitter for ever & the receiver is receiver for ever can't be changed.

Simplex system



* ex : the Radio

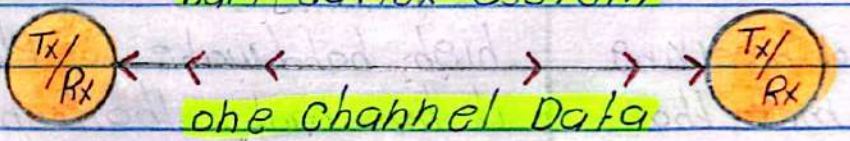
→ **half Duplex :**

* two devices can send & receive from eachother

note

when a device works as a transmitter the another Device can't work as a transmitter too & the same thing at the receiving case when a device works as a receiver the another device can't work as a receiver too this means one is transmitter & another is receiver because I have one channel for data

half duplex system



ex : walkie - talkie جهاز لاسلكي

→ **Full Duplex :**

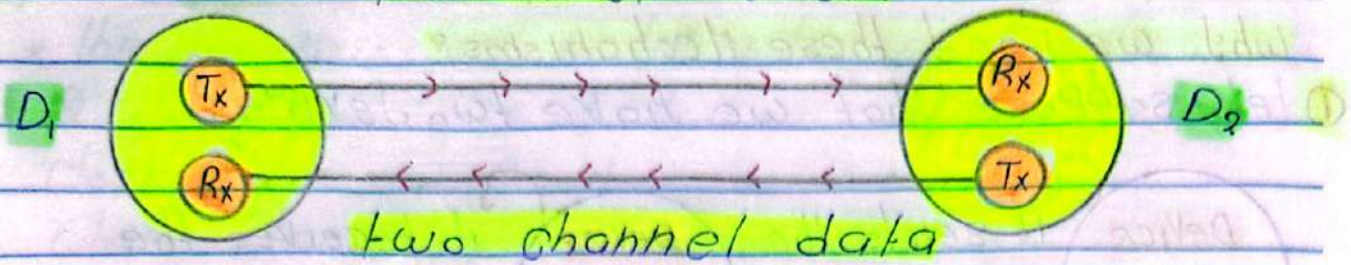
* two devices can send & receive from each other.

note

the communication between the two devices "send or receive" is done at the same time. because I have two channel for Data

Page: Date:

Full DuPlex system

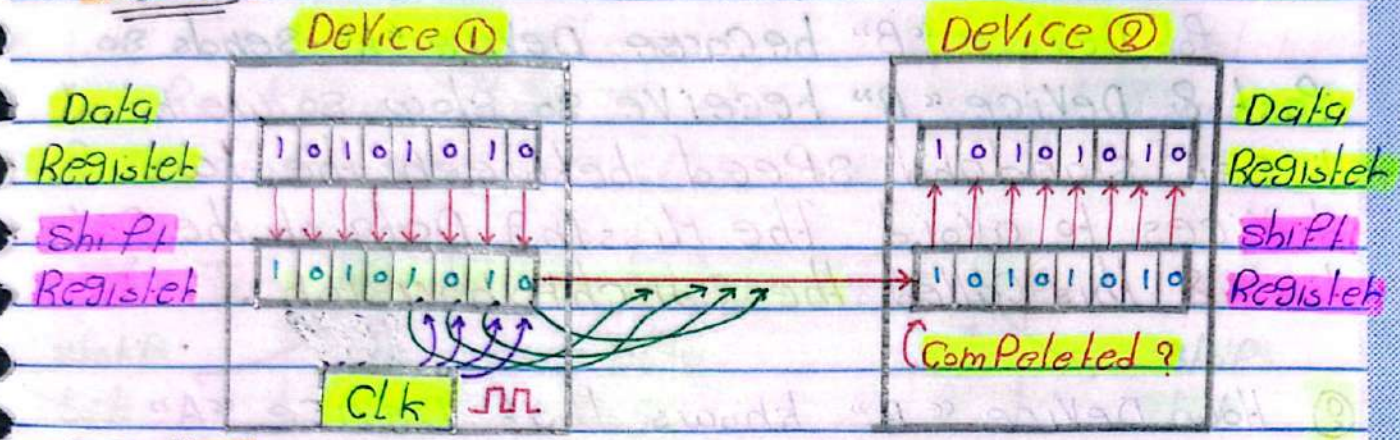


two channel data

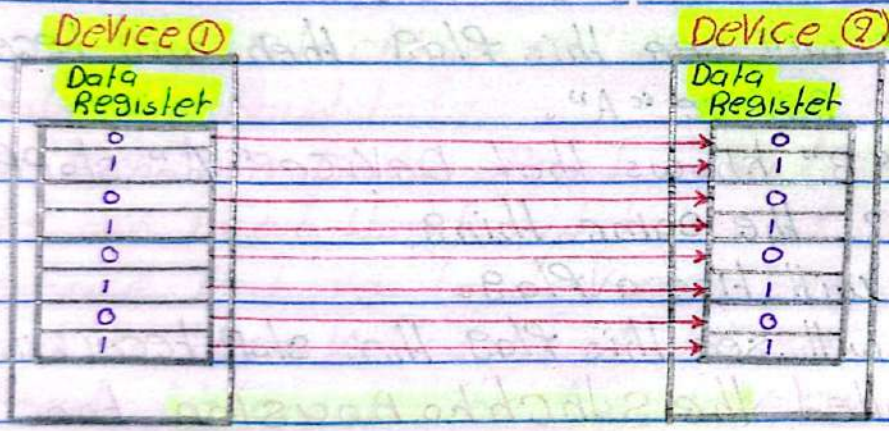
ex: telePhone Call

How the data is transfered at the serial & Parallel Communication

Serial



Parallell



note

May be there is a noise at the wires at the Parallel Communication & this Makes Data loss

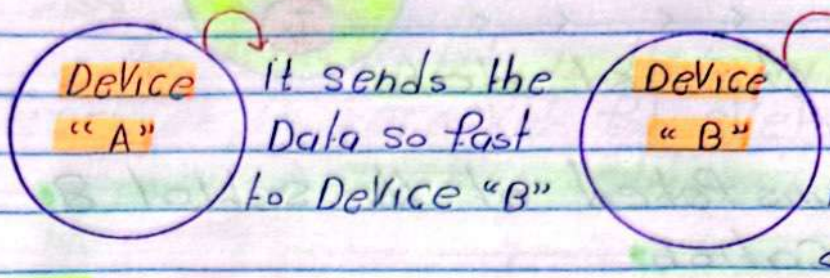
* May be there's a wire is not suitible to transfere the Data is damaged or cutted

Page: ..... Date: .....

Asynchronous & Synchronous Mechanisms

Why we need these Mechanisms?

① let's suppose that we have two devices



This will make data loss for Device "B" because Device "A" sends so fast & Device "B" receive so slow so we need to set standard speed between the two devices to avoid the Missing Data or the Data loss & this called the Synchronousion.

② How Device "B" knows that Device "A" wants to send for it Data?

1- Device "A" will rise a flag.

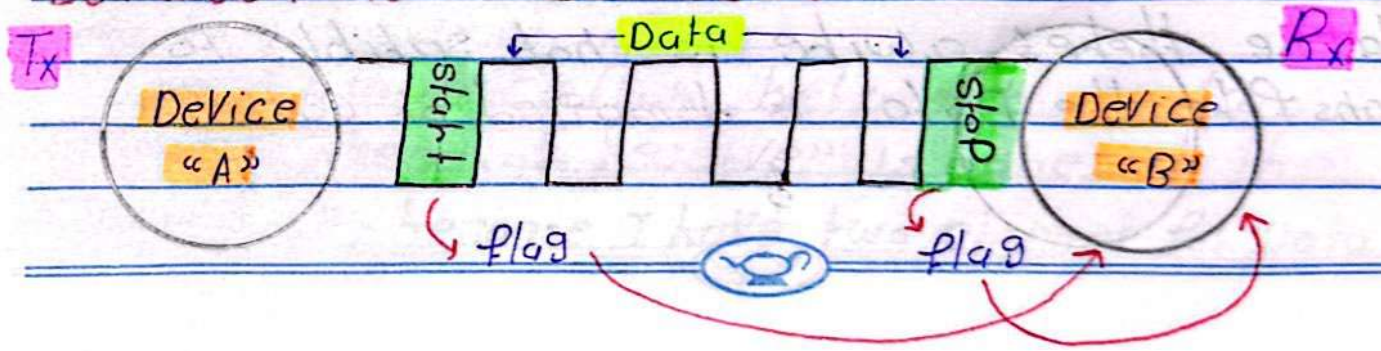
2- Device "B" will see this flag then will receive the Data from Device "A".

How Device "B" knows that Device "A" stopped sending Data? the same thing

1- Device "A" will rise a flag.

2- Device "B" will see this flag then stop receiving.

So this it's called the Synchronousion too between the two devices.



Page: ..... Date: .....

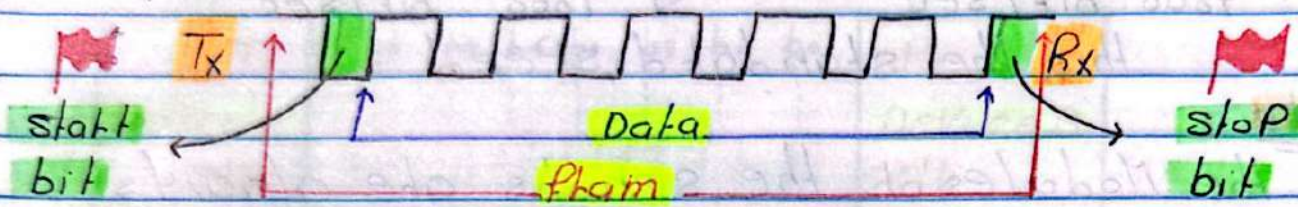
→ **Asynchronous**

* the transmission is depended on the framing & the baud rate & don't use the external clock.

what's framing

is a group of bits between the two devecives to send & receive the data is devided into the start bit to send & the data bits & the stop bit to stop sending

why we use the framing at the Asynchronous to Rx knows that Tx starts to send the Data & Tx stops sending the Data



note

the start bit & stop bit are called the synchronization bits

what's the baud rate

the sending Data speed. bit / sec

why we need the baud rate at Asynchronous because I have to devices one is sender & other is receiver I need to set the standard speed between the two devices or the suitible speed.

note

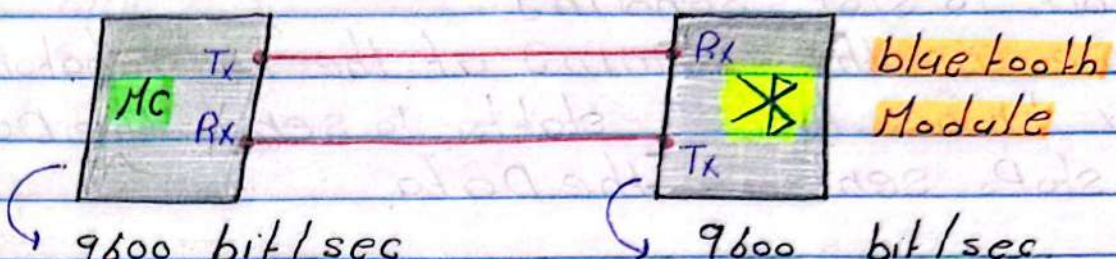
the Mc support different baud rates ex:- 9600, 2400 ... bit/sec.

Page: ..... Date: .....

*

note

how I have blue tooth Module with 9600 bit/sec baud rate I need to connecte it with the Micro Controlleh so I Must set the baud rate of the Micro Controlleh is 9600 bit/sec to Make the standard speed between the two devices & this at the Case of Asynchronous



this the standard speed

note

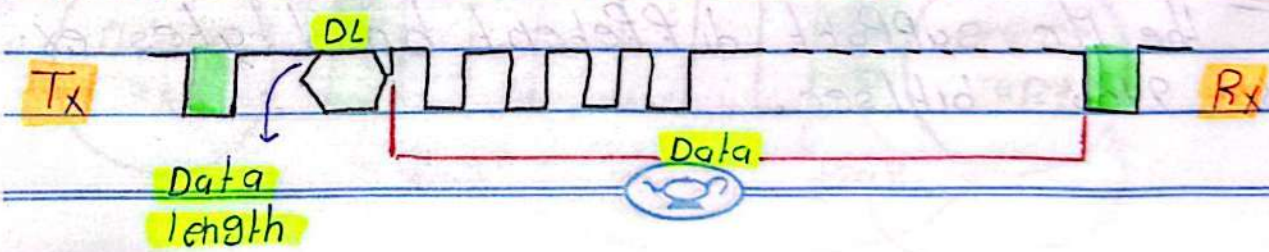
the Modules or the sensors are always fixed baud rate so you need to set the Micro Controlleh by this baud rate at the Case of Asynchronous.

note

Asynchronous Mechanism is byte oriented Communication this Means the Max data to send is 1 byte at the Frame.

note

there are some Asynchronous Photocols support sending More than 1 byte at the frame & the frame will be like that



Page: Date:

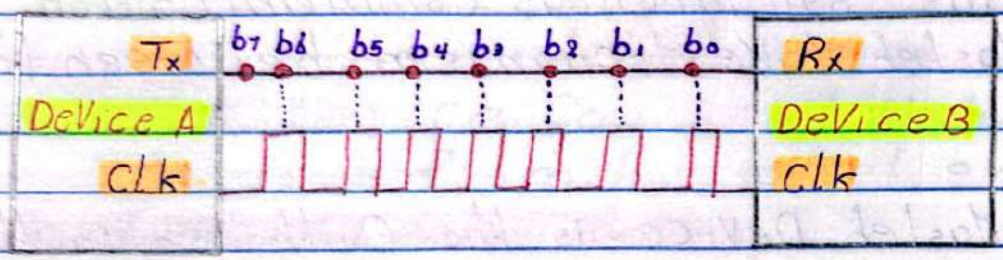
what's the Data length.. is number of sending bytes of Data DL = 3 this means I send 3 bytes over the frame.

ex for the Asynchronous Mechanism UART

UART : universal Asynchronous Receiver Transmitter.

→ Synchronous

* the transmission of the data uses external clock.
* the transmission is synchronized over clock This means the transmitter sends the Data with the clock



note

at synchronous Mechanism I can have channel for Data & Channel for the clock.

note

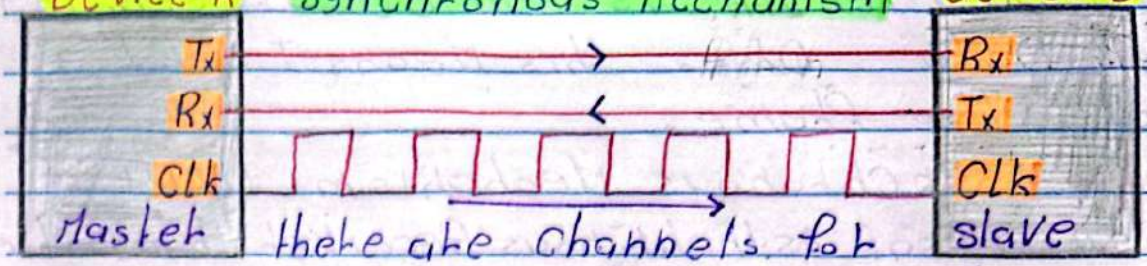
the synchronous Ization can be at rising or falling edge.

* synchronous Mechanism used at short Distance Communication & it's more Data to send.
* it's faster than Asynchronous Mechanism
* ex:

SPI, I2C, USART, USB.

Page: Date:

Synchronous Mechanism



there are channels for the Data & other for the clock.

Note

let's suppose that device A is sending the data with high frequency & Device B can't receive this Data by this high frequency so we need to know the Master slave Mechanism

Note

at any synchronous Communication there's Master slave Mechanism between the Devices.

Note

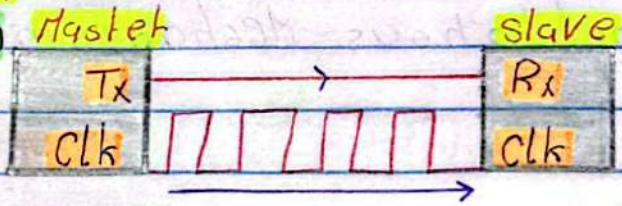
the Master Device is the Controller in the Clock is generating the clock to the slave Device & says to the slave Device I will send or receive the data & I will stop send or receive the data

Note

the direction of the clock is always in one direction from the Master to the slave

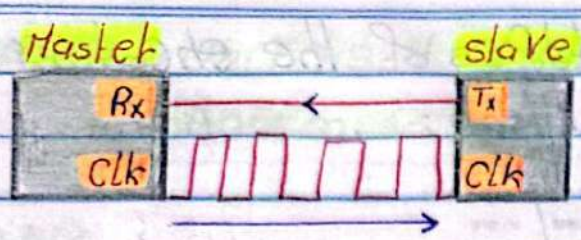
let's see Some Cases for the Master slave Mechanism

→ **Single Master single slave**

① 

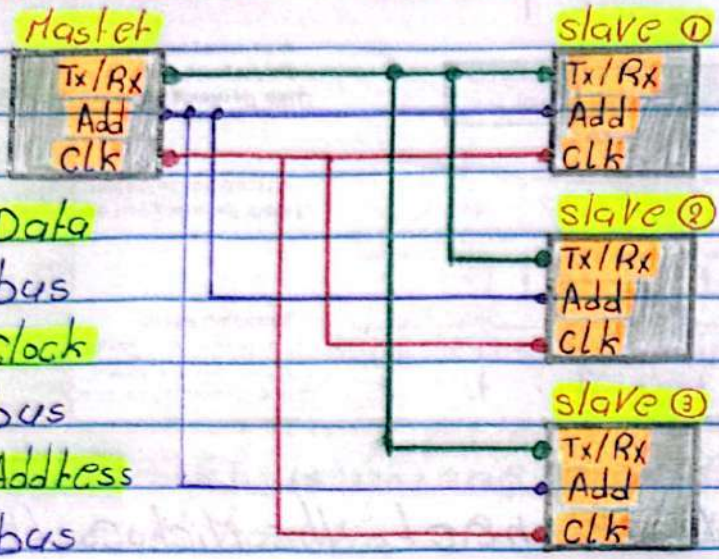
the Master generates the clock to send the Data to the slave & start & stop the Communication.

Page: ..... Date: .....

② 

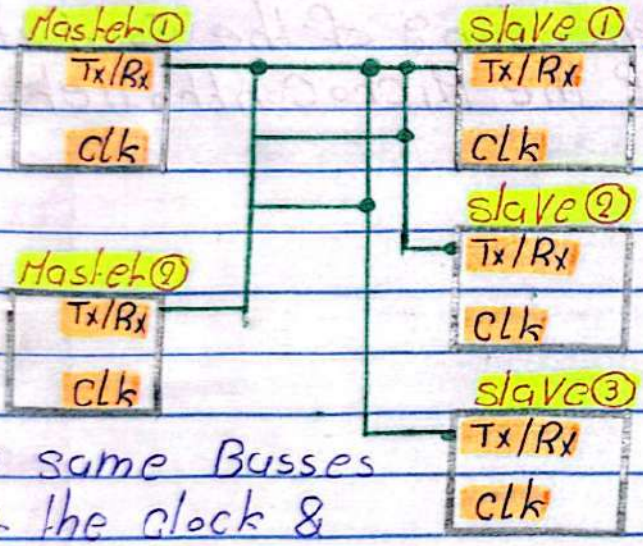
the Master will generate the clock to read the Data from the slave Device.

→ Single Master Multy slaves

① 

if the Maste wants to S... with any slave device will generate the clock to the target slave Device & set it's Address on the Address bus to know the target slave Device after that the Master Device can send or receive the data to or from the target slave device.

→ Multy Masters Multy slaves



the same Busses for the clock & Address

hote
there's too a type is the Multy Masters single slave.

How the data is transmitted between the Devices by the line encoding.

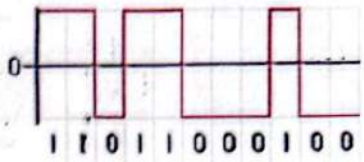
Page:..... Date:.....

* as you see these some types of the encoding methods with it's description :-

ne Encoding for data transmission :-

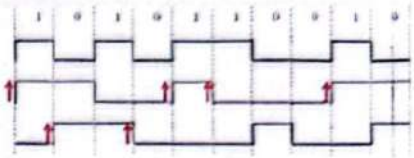
How to represent the data over a protocol or a wire :-

- ✓ Ex: (Non returned to Zero "NRZL") encoding method.
  - o At logic (1) → +ve (voltage)
  - o At logic (0) → -ve (voltage)



Code name	Alternate name	Complete name	Description
NRZ(L)	NRZL	Non-return-to-zero level	Appears as raw binary bits without any coding. Typically binary 1 maps to logic-level high, and binary 0 maps to logic-level low. Inverse logic mapping is also a type of NRZ(L) code
NRZ(I)	NRZI	Non-return-to-zero inverted	Refers to either an NRZ(M) or NRZ(S) code. Ex. RS-232
NRZ(M)	NRZM	Non-return-to-zero mark	Serializer mapping (0: constant, 1: toggle)
NRZ(S)	NRZS	Non-return-to-zero space	Serializer mapping (0: toggle, 1: constant)
NRZ(C)	NRZC	Non-return-to-zero change	

https://en.wikipedia.org/wiki/Non-return-to-zero

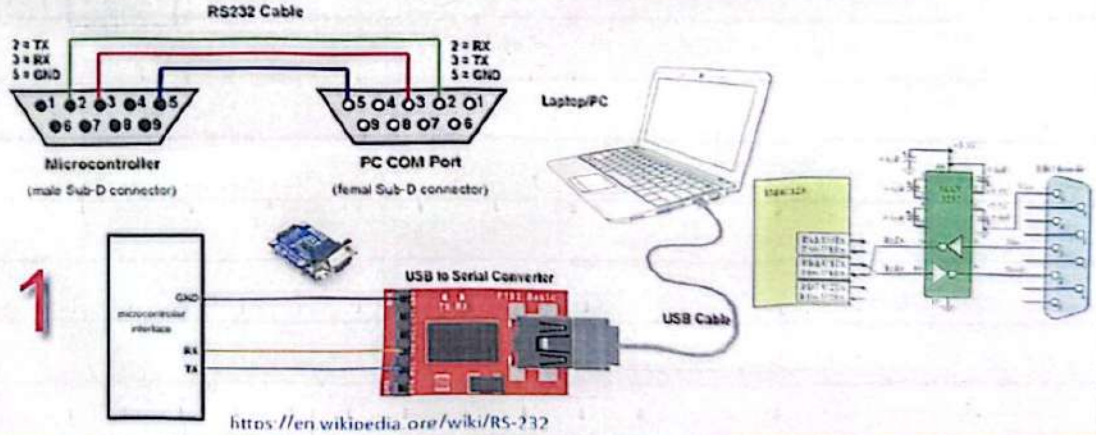


hote

= if you want to connect the MicroControlle with the Computer, the MicroControllet understa a line encoding & the Computer understand another line encoding so you need a Convert to translate the line encoding of the Compute to the line encoding of the MicroController.

ne Encoding for data transmission (NRZI Encoding Method) :-

Valid signals are either in the range of +3 to +15 (0) volts or the range −3 to −15 (1) volts with respect to the "Common Ground" (GND) pin; consequently, the range between −3 to +3 volts is not a valid RS-232 level.



https://en.wikipedia.org/wiki/RS-232

Page: Date:

UART : Universal Asynchronous Receiver / Transmiter

USART : Universal Synchronous / Asynchronous Receiver / Transmiter

what's the Main difference between

UART : it supports Asynchronous Communication only.

USART : it supports Synchronous / Asynchronous Communication.

Some Characteristics about UART / USART

→ UART / USART is not a defined Communication Protocol because it has no specifications, document, references like I2C or CAN ... ect.

→ is just a Circuit or Module inside the MicroController or a sensor or any device to support only the Serial Communication between the devices.

note

the Circuit of UART / USART you can configure it if this Circuit is inside the MCU only, you can configure the speed or the Data size.

note

if the Circuit of UART / USART is inside a sensor or any device can't be Programmable you can't configure it because it's already configured by the speed & the Data length.

the specifications of the UART & USART Modules at the MicroController

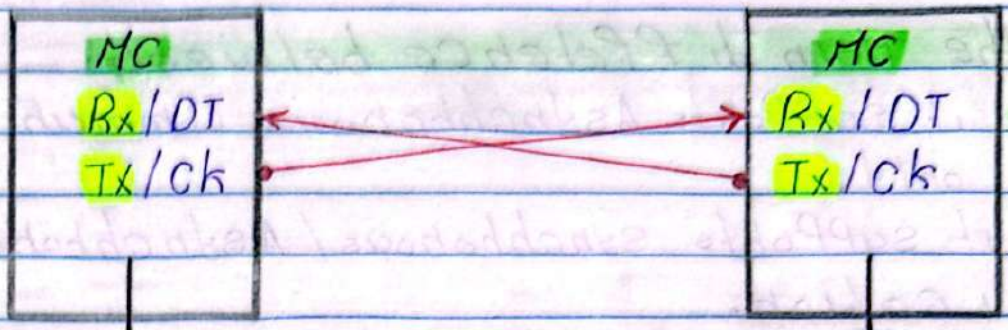
Page:..... Date:.....

→ **if the MC has UART Module :**

* it supports only Asynchronous functionality.

This Means

* we need to define frame format & baud rate "speed" & it has no clock



This Device has Tx & Rx → I Can send & receive ← This Device has Rx / Tx at the same time

↓

Full Duplex Communication

where :

Rx : receiver DT : sending Data

Tx : transmitter Ck : sending clock

→ **if the MC has USART Module :**

it supports the synchronous & Asynchronous functionality together.

* Asynchronous : is explained above ☺ ↑

* **Synchronous :**

This Means :

* we need the clock & the master slave Mechanizem

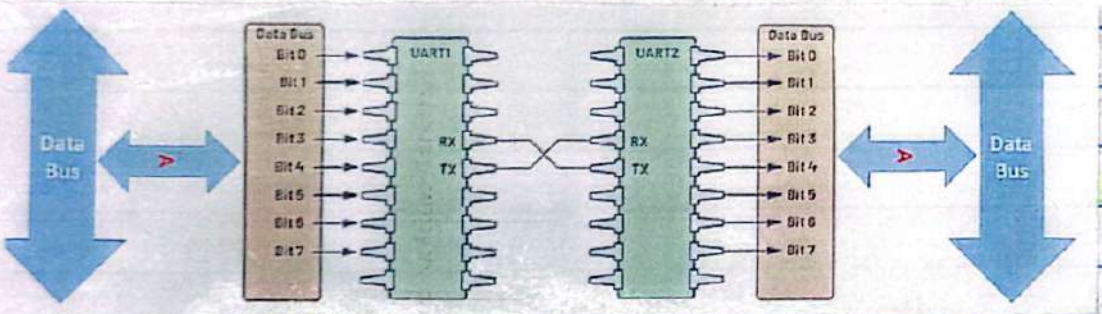
Page: ..... Date: .....

* as you see one Data channel between the two devices & both devices can't receive or transmite together at the same time because I have just one channel for the Data

* as you see Device "1" can work as Tx / Rx but is not the same time & Device "2" too So this means half duplex communication

* you will follow the master slave mechanizem to set the master Device that will genelate the clock ⊓⊓⊓ & set the slave Device.

→ USART characteristics "Asynchronous Mode" UART the transmission here is done by one of devices that is master or slave gets the Data from the Data bus of the Micro Controller & save this data in a register & send this data bit by bit over the serial wire to another devices.



note

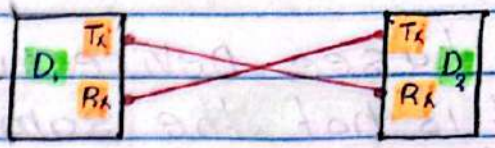
you have to set the baud rate between the two devices to avoid the Data loss.

Page:..... Date:.....

hote

Asynchronous Mode UART is single Master single slave why? because I don't have a clock channel to generate a clock to choose which device I want to speak with it.

Just the Data channels ←



→ the frame of Asynchronous Mode UART "Packet"

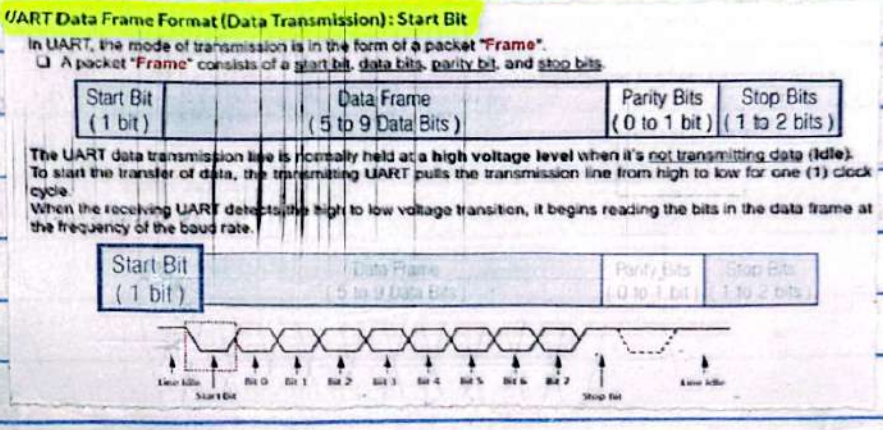
any UART Module Must have at his fram

1- start bit : to start the transmission.

2- Data frame : it has the bits of Data.

3- Parity Bit : is optionally is to check if the Data at the Data frame is transmitted true or it has some losses.

4- stop bits : to stop the transmission.



hote

1/ baud ra

as you see : the start bit is 1 bit & the Data fram is from 5 to 9 Data bits on the frame & Parity bit is 0 to 1 bit & 0 means may be is not existed finally the stop bits is from 1 to 2 bits.

explaining the elements of the frame

* line Idle status : when there's no communication between the two devices.

Page: ..... Date: .....

Note the Idle status at the UART is high

Idle  has 5 V → logic "1"

* **the start bit :**

is Consider as a flag to start the transmission is low has 0V → logic "0"

Idle status 

Note the frame is high at the first this the Idle status to start the transmission send low bit

* **the Data frame**

the range of bits at this frame from 5 to 9 bits but notice :

→ if the Data size 5 ~ 8 bits this Means I have the Parity bit.

→ if the Data size is 9 bits this Means I don't have the Parity bit

Note

the sender & receiver devices Must align to the size of Data filed

* Must be the same numbers of bits
* if they are not the same this will Make issue ⊗.

how I want to send 'A' chat How will be Presented on the fram?

'A' → Askll Code = 65 → 0 1000001

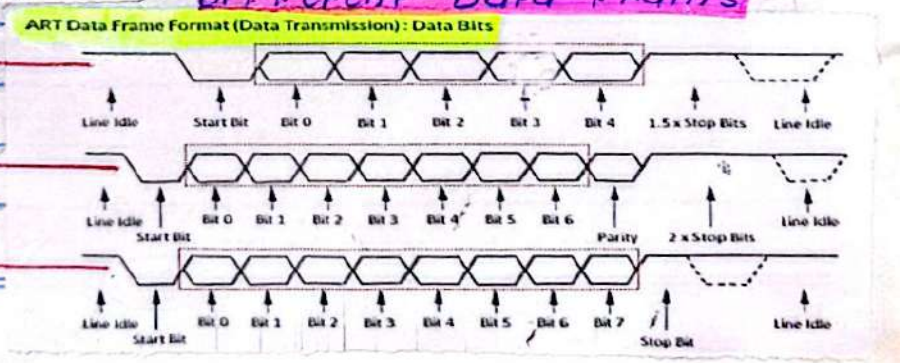
MSB (7) LSB (0)

Different Data frams:

5 bits Data frame ←

6 bits Data frame ←

8 bits Data frame ←



Page:..... Date:.....

Note the two Devices Must know the shape of the fram what it is.

* what's the types of Data loss Problems:
  - → electromagnetic radiation.
  - → Mismatched baud rate for the Devices.
  - → long distance data thans fers.

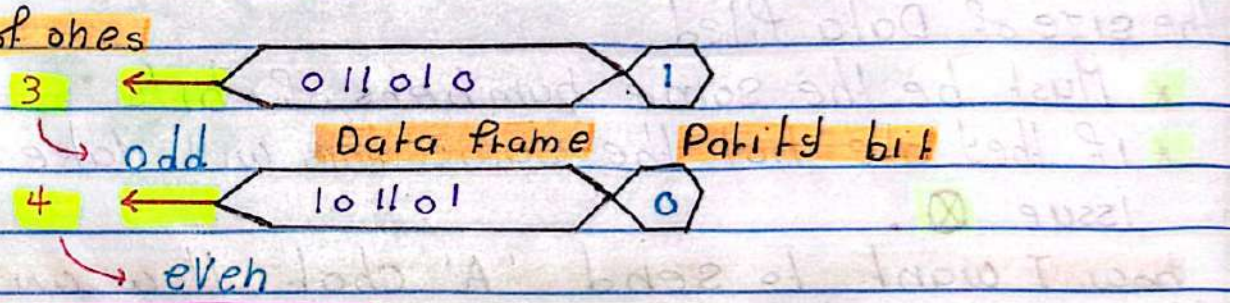
here the role of the Parity bit Comes to check if there's a Data Corruption or not

* the Parity bit

there's two types of the Parity
even Parity & odd Parity

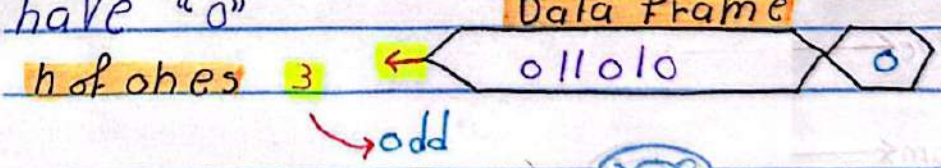
the even Parity:

the Data fram is Consider as a group of zeros & ones the Parity bit Counts the number of ones at the Data fram if this numbe is even the Parity bit will have "0" if this number is odd the Parity bit will have "1"

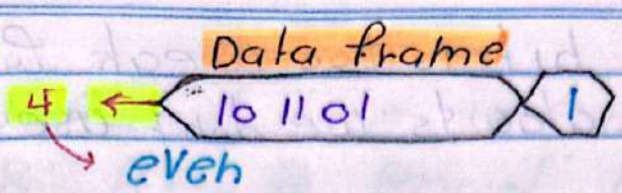


the odd Parity:

oppsite the even Parity the Parity bit Counts the number of ones a the data fram if this number is even the Parity bit will have "1" if this number is odd the Parity bit will have "0"



Page: ..... Date: .....

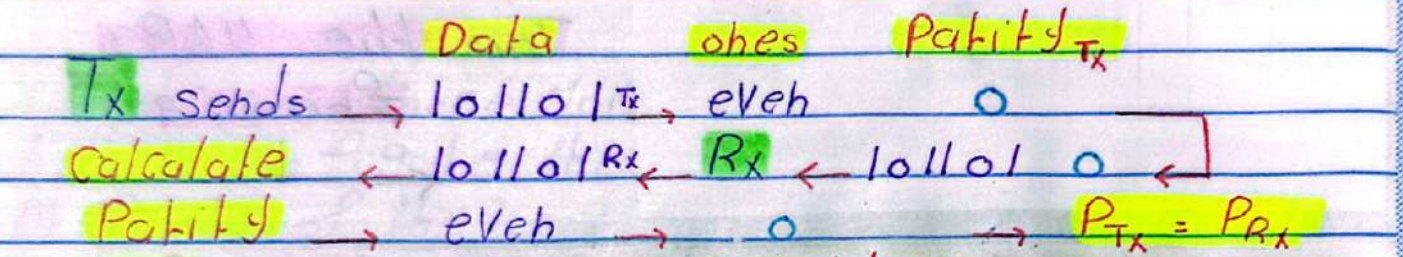


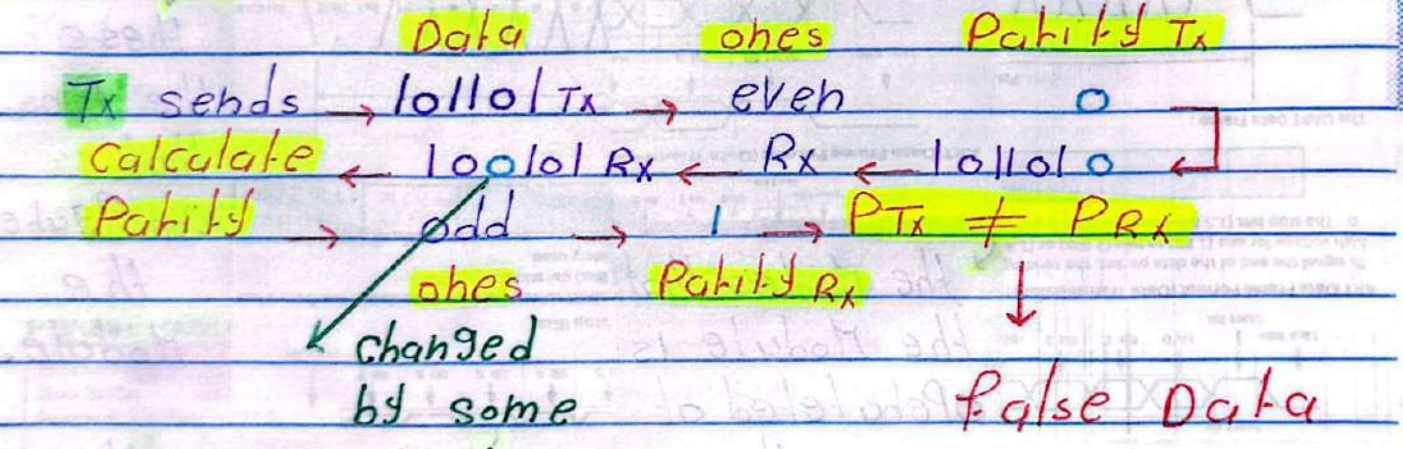
How to know if the data is sent true or not by the Parity bit ?

let's suppose that the frame is even Parity

* Tx will send the Data to Rx but before the Tx send the Data it will calculate the Parity of data & send it to the Rx at the Rx the Parity of the data will be calculated too if the Parity at Tx equals the Parity at Rx this means the Data is True #

if the Parity at Tx is not equaled to the Parity at the Rx this means the data is False #





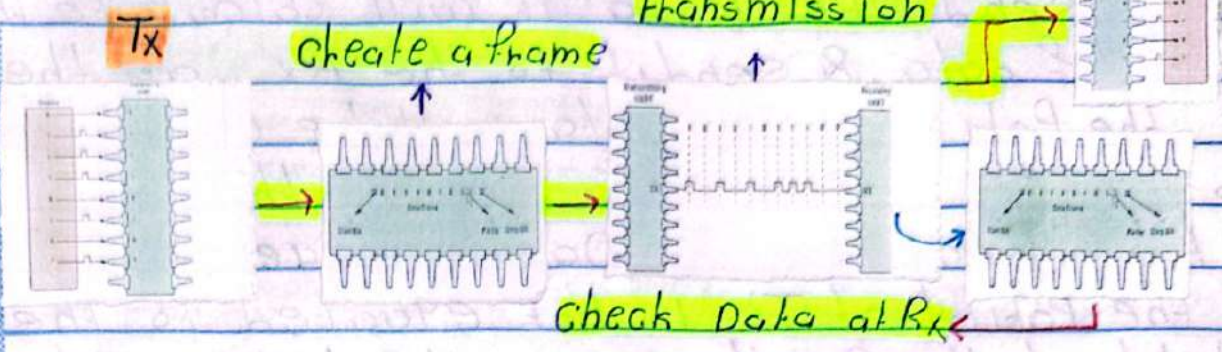
note if you have noise at the bits you can't know if the Data is True or false

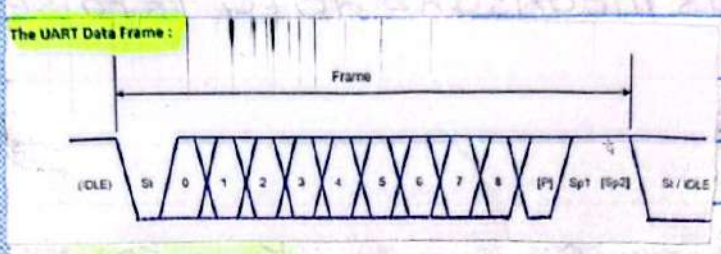
Page:..... Date:.....

* So the Parity bit is too weak for checking the Data always we don't use it.

* Stop bits:
are always after the Parity bits, have 5V → logic "1" → are high, they are to end the transmission frame.

UART transmission steps over frame





This the UART frame

USART Module at the Data sheet Pic18 F4680

18.0 ENHANCED UNIVERSAL SYNCHRONOUS RECEIVER TRANSMITTER (EUSART)

The Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) module is one of the two serial I/O modules. (Generically, the USART is also known as a Serial Communications Interface or SCI.) The EUSART can be configured as a full-duplex asynchronous system that can communicate with peripheral devices, such as CRT terminals and personal computers. It can also 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 Enhanced USART module implements additional features, including automatic baud rate detection and calibration, automatic wake-up on Sync Break reception and 12-bit Break character transmit. These make it ideally suited for use in Local Interconnect Network bus (LIN bus) systems.

Some Information about the Module.

The EUSART can be configured in the following modes:

- Asynchronous (full duplex) with:
  - Auto-wake-up on character reception
  - Auto-baud calibration
  - 12-bit Break character transmission
- Synchronous – Master (half duplex) with selectable clock polarity
- Synchronous – Slave (half duplex) with selectable clock polarity

the Modes that the Module is operated at them.

The pins of the Enhanced USART are multiplexed with PORTC. In order to configure RC6/TX/CK and RC7/RX/DT as a USART:

- SPEN bit (RCSTA<7>) must be set (= 1)
- TRISC<7> bit must be set (= 1) Init USART
- TRISC<6> bit must be set (= 1)

Note: The EUSART control will automatically reconfigure the pin from input to output as needed.

The operation of the Enhanced USART module is controlled through three registers:

- Transmit Status and Control (TXSTA)
- Receive Status and Control (RCSTA)
- Baud Rate Control (BAUDCON)

These are detailed on the following pages in Register 18-1, Register 18-2 and Register 18-3, respectively.

these the Pins that Configure the Module.

these are the Control Registers of the Module.

Page: ..... Date: .....

hote

the UART Module is Consider as Tx & Rx & Baud rate.

→ the BRG " baud rate generator"

18.1 Baud Rate Generator (BRG)

The BRG is a dedicated 8-bit or 16-bit generator that supports both the Asynchronous and Synchronous modes of the EUSART. By default, the BRG operates in 8-bit mode; setting the BRG16 bit (BAUDCON<3>) selects 16-bit mode.

The SPBRGH:SPBRG register pair controls the period of a free-running timer. In Asynchronous mode, bits, BRGH (TXSTA<2>) and BRG16 (BAUDCON<3>), also control the baud rate. In Synchronous mode, BRGH is ignored. Table 18-1 shows the formula for computation of the baud rate for different EUSART modes which only apply in Master mode (internally generated clock).

Given the desired baud rate and Fosc, the nearest integer value for the SPBRGH:SPBRG registers can be calculated using the formulas in Table 18-1. From this, the error in baud rate can be determined. An example calculation is shown in Example 18-1. Typical baud rates and error values for the various Asynchronous modes are shown in Table 18-2. It may be advantageous to use the high baud rate (BRGH = 1) or the 16-bit BRG to reduce the baud rate error, or achieve a slow baud rate for a fast oscillator frequency.

Writing any value (even the same value) to the SPBRGH:SPBRG registers immediately reloads the BRG timer. This may corrupt a transmission or reception already in progress. This ensures the BRG does not wait for a timer overflow before outputting the new baud rate.

18.1.1 OPERATION IN POWER-MANAGED MODES

The device clock is used to generate the desired baud rate. When one of the power-managed modes is entered, the new clock source may be operating at a different frequency. This may require an adjustment to the value in the SPBRG register pair.

18.1.2 SAMPLING

The data on the RX 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 when SYNC is clear or when BRG16 and BRGH are both not set. The data on the RX pin is sampled once when SYNC is set or when BRGH16 and BRGH are both set.

↷ the Information about the baud rate is used for the shift register to send the Data bit by bit with the clock that is generated by the baud rate

* there are two Registers to write the desire baud rate inside it SPBRGH : SPBRG.

* there are three Main bits to Configure the Properties of the baud rate & Choose the equation to Calculate the desire baud rate Value to write it at SPBRGH : SPBRG registers.

→ SYNC :: to set Synchronous & Asynchronous mode

→ BRG16 :: select H&L Register or L Register only.

→ BRGH :: Asynchronous Mode High & low speed.

TABLE 18-1: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n + 1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n + 1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n + 1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGH:SPBRG register pair

EXAMPLE 18-1: CALCULATING BAUD RATE ERROR

For a device with Fosc of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

Desired Baud Rate = Fosc/(64 ([SPBRGH:SPBRG] + 1))

Solving for SPBRGH:SPBRG:

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

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

Error = (Calculated Baud Rate - Desired Baud Rate)/Desired Baud Rate
= (9615 - 9600)/9600 = 0.16%

X = is the value that will be whitten at the register

↷ this equation to Calculate the desire baud rate Value & the Precentage error of this Value.

Page:..... Date:.....

→ the Data sheet Provides some desire baud rate values with different Mode & different internal frequency Fosc

→ accordding to the Data sheet this table

SYNC = 0 Asynchronous
BRGH = 0 low speed
BRG16 = 0 8 bit register
suPPorts : Asynchronous Mode low speed 8 bit register SPBRG register only

SYNC = 0, BRGH = 0, BRG16 = 0

BAUD RATE (K)	Fosc = 40.000 MHz			Fosc = 20.000 MHz			Fosc = 10.000 MHz			Fosc = 8.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	—	—	—	—	—	—	—	—	—	—	—	—
1.2	—	—	—	1.221	1.73	255	1.202	0.16	129	1.201	-0.16	103
2.4	2.441	1.73	255	2.404	0.16	129	2.404	0.16	64	2.403	-0.16	51
9.6	9.615	0.16	64	9.766	1.73	31	9.766	1.73	15	9.615	-0.16	12
19.2	19.531	1.73	31	19.531	1.73	15	19.531	1.73	7	—	—	—
57.6	56.818	-1.36	10	62.500	8.51	4	52.083	9.58	2	—	—	—
115.2	125.000	8.51	4	104.167	9.58	2	78.125	-32.18	1	—	—	—

SYNC = 0, BRGH = 0, BRG16 = 0

BAUD RATE (K)	Fosc = 4.000 MHz			Fosc = 2.000 MHz			Fosc = 1.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	0.300	0.16	207	0.300	-0.16	103	0.300	-0.16	51
1.2	1.202	0.16	51	1.201	-0.16	25	1.201	-0.16	12
2.4	2.404	0.16	25	2.403	-0.16	12	—	—	—
9.6	8.929	-6.99	6	—	—	—	—	—	—
19.2	20.833	8.51	2	—	—	—	—	—	—
57.6	62.500	8.51	0	—	—	—	—	—	—
115.2	62.500	-45.75	0	—	—	—	—	—	—

← SYNC = 0 Asynchronous
BRGH = 1 high speed
BRG16 = 0 8 bit register
suPPorts : Asynchronous Mode high speed 8 bit register SPBRG register only

SYNC = 0, BRGH = 1, BRG16 = 0

BAUD RATE (K)	Fosc = 40.000 MHz			Fosc = 20.000 MHz			Fosc = 10.000 MHz			Fosc = 8.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	—	—	—	—	—	—	—	—	—	—	—	—
1.2	—	—	—	—	—	—	—	—	—	—	—	—
2.4	—	—	—	—	—	—	—	—	—	2.441	1.73	255
9.6	9.766	1.73	255	9.615	0.16	129	9.615	0.16	64	9.615	-0.16	51
19.2	19.231	0.16	129	19.231	0.16	64	19.531	1.73	31	19.230	-0.16	25
57.6	58.140	0.94	42	56.818	-1.36	21	56.818	-1.36	10	55.555	3.55	8
115.2	113.636	-1.36	21	113.636	-1.36	10	125.000	8.51	4	—	—	—

SYNC = 0, BRGH = 1, BRG16 = 0

BAUD RATE (K)	Fosc = 4.000 MHz			Fosc = 2.000 MHz			Fosc = 1.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	—	—	—	—	—	—	0.300	-0.16	207
1.2	1.202	0.16	207	1.201	-0.16	103	1.201	-0.16	51
2.4	2.404	0.16	103	2.403	-0.16	51	2.403	-0.16	25
9.6	9.615	0.16	25	9.615	-0.16	12	—	—	—
19.2	19.231	0.16	12	—	—	—	—	—	—
57.6	62.500	8.51	3	—	—	—	—	—	—
115.2	125.000	8.51	1	—	—	—	—	—	—

SYNC = 0, BRGH = 0, BRG16 = 1

BAUD RATE (K)	Fosc = 40.000 MHz			Fosc = 20.000 MHz			Fosc = 10.000 MHz			Fosc = 8.000 MH		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	0.300	0.00	8332	0.300	0.02	4165	0.300	0.02	2082	0.300	-0.04	1665
1.2	1.200	0.02	2082	1.200	-0.03	1041	1.200	-0.03	520	1.201	-0.16	415
2.4	2.402	0.06	1040	2.399	0.03	520	2.404	0.16	259	2.403	-0.16	207
9.6	9.615	0.16	259	9.615	0.16	129	9.615	0.16	64	9.615	-0.16	51
19.2	19.231	0.16	129	19.231	0.16	64	19.531	1.73	31	19.230	-0.16	25
57.6	58.140	0.94	42	56.818	-1.36	21	56.818	-1.36	10	55.555	3.55	8
115.2	113.636	-1.36	21	113.636	-1.36	10	125.000	8.51	4	—	—	—

SYNC = 0, BRGH = 0, BRG16 = 1

BAUD RATE (K)	Fosc = 4.000 MHz			Fosc = 2.000 MHz			Fosc = 1.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	0.300	0.04	832	0.300	-0.16	415	0.300	-0.16	207
1.2	1.202	0.16	207	1.201	-0.16	103	1.201	-0.16	51
2.4	2.404	0.16	103	2.403	-0.16	51	2.403	-0.16	25
9.6	9.615	0.16	25	9.615	0.16	12	—	—	—
19.2	19.231	0.16	12	—	—	—	—	—	—
57.6	62.500	8.51	3	—	—	—	—	—	—
115.2	125.000	8.51	1	—	—	—	—	—	—

→ SYNC = 0 Asynchronous
BRGH = 0 low speed
BRG16 = 1 16 bit register here
SPBRGH : SPBRG register together.

Page: ..... Date: .....

SYNC = 0 Asynchronous
BRG16 = 1 16 bit register
BRGH = 1 high speed

or

SYNC = 1 Synchronous
BRGH = X
BRG16 = 1 16 bit register

SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1

BAUD RATE (K)	Fosc = 40.000 MHz			Fosc = 20.000 MHz			Fosc = 10.000 MHz			Fosc = 8.000 MH		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SP v ue (de mal)
0.3	0.300	0.00	33332	0.300	0.00	16665	0.300	0.00	8332	0.300	-0.01	6
1.2	1.200	0.00	8332	1.200	0.02	4165	1.200	0.02	2082	1.200	-0.04	1
2.4	2.400	0.02	4165	2.400	0.02	2082	2.402	0.06	1040	2.400	-0.04	2
9.6	9.606	0.06	1040	9.596	-0.03	520	9.615	0.16	259	9.615	-0.16	1
19.2	19.193	-0.03	520	19.231	0.16	259	19.231	0.16	129	19.230	-0.16	1
57.6	57.803	0.35	172	57.471	-0.22	86	58.140	0.94	42	57.142	0.79	4
115.2	114.943	-0.22	86	116.279	0.94	42	113.636	-1.36	21	117.647	-2.12	2

SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1

BAUD RATE (K)	Fosc = 4.000 MHz			Fosc = 2.000 MHz			Fosc = 1.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	0.300	0.01	3332	0.300	-0.04	1665	0.300	-0.04	832
1.2	1.200	0.04	832	1.201	-0.16	415	1.201	-0.16	207
2.4	2.404	0.16	415	2.403	-0.16	207	2.403	-0.16	103
9.6	9.615	0.16	103	9.615	-0.16	51	9.615	-0.16	25
19.2	19.231	0.16	51	19.230	-0.16	25	19.230	-0.16	12
57.6	58.824	2.12	16	55.555	3.55	8	—	—	—
115.2	111.111	-3.55	8	—	—	—	—	—	—

TABLE 18-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Value on page
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	51
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	51
BAUDCON	ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN	51
SPBRGH	EUSART Baud Rate Generator Register High Byte								51
SPBRG	EUSART Baud Rate Generator Register Low Byte								51

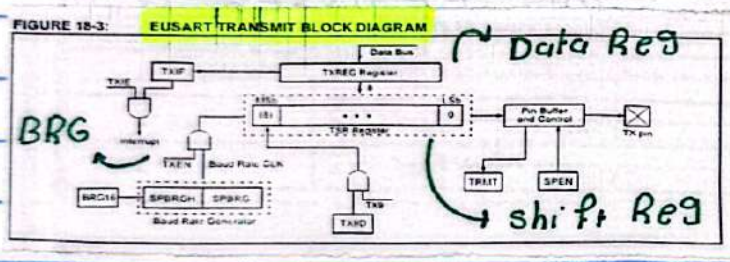
Legend: — = unimplemented, read as '0'. Shaded cells are not used by the BRG.

* these the registers that is related to the baud Rate Generator

→ Asynchronous Mode Transmitter

the block digram Asynchronous Mode Transmitter:

FIGURE 18-3: EUSART TRANSMIT BLOCK DIAGRAM



* the Data register will take the Data from the Data bus & store it inside & Pass it to the shift Register.

note

you will know that the Data register has no data & is empty after it Passed the Data to the shift register b TxIF flag.

note

the BRG is responsible for generate the clock to transfer the Data from the shift register to Tx Pin bit by bit.

Page:..... Date:.....

*

hote

Msb (8) May be is Consider a Data bit or the Parity bit it will work as a Data bit if you Configure that your Frame is 9 bits Data Frame.

hote

there's no hard ware circuit inside the UART Module to Calculate the Parity you will Calculate it by yourself.

hote

the Data Register will not take Data from the Data bus till the stop bit has been transmitted.

hote

TRMT : is to say if the Data has been transmitted or not & the shift register is empty how this bit is Read only.

REGISTER 18-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

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

bit 7 **CSRC**: Clock Source Select bit
Asynchronous mode:
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
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC**: EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB**: Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care

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
1 = TSR empty
0 = TSR full

bit 0 **TX9D**: 9th Bit of Transmit Data
Can be address/data bit or a parity bit

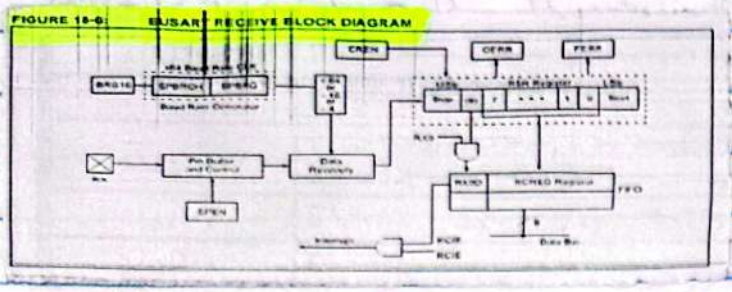
Note 1: SREN/CREN overrides TXEN in Sync mode.

this Register is to Control the Configuration of the UART transmit Mode.

Page:..... Date:.....

→ Asynchronous Mode Receiver

the block digram of Asynchronous Mode Receiver



the Data is received to the Rx Pin at the MicroControlloer after that checking if you set the Serial Port Enable or not by SPEN bit after that the data will go into Data Recovery which Makes the baud rate is fast it Multiples BR×16 after that the Data will be formed at the shift Register & the Frame.

Shift Register

↑

hote

→ OERR : over Run error : if the SR is over flow

→ FERP : framing error : if there's an error at frame

→ CREN : is a bit to Control to reset the receiver Mode : enable receiver or disable receiver.

after that the data will go into the Data Register then over the Data bus of the MC.

this the Receive Control Register to Configure all the Interfacing to the Receiver Mode.

REGISTER 18-1: RCSTA: RECEIVE STATUS AND CONTROL REGISTER

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

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

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 (held in Reset)

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, enables interrupt and loads the receive buffer when RSR<8> is = 1
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit
Asynchronous mode 9-Bit (RX9 = 0):
Don't care.

bit 2 FERR: Framing Error bit
1 = Framing error (can be cleared by reading RCREG register and receiving 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.

Page:..... Date:.....

REGISTER 18-3: BAUDCON: BAUD RATE CONTROL REGISTER

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDE
bit 7							bit 0

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

bit 7 **ABDOVF:** Auto-Baud Acquisition Rollover Status bit
1 = A BRG rollover has occurred during Auto-Baud Rate Detect mode (must be cleared in software)
0 = No BRG rollover has occurred

bit 6 **RCIDL:** Receive Operation Idle Status bit
1 = Receive operation is Idle
0 = Receive operation is active

bit 5 **RXDTP:** Received Data Polarity Select bit (Asynchronous mode only)
Asynchronous mode:
1 = RX data is inverted
0 = RX data received is not inverted

bit 4 **TXCKP:** Clock and Data Polarity Select bit
Asynchronous mode:
1 = Idle state for transmit (TX) is a low level
0 = Idle state for transmit (TX) is a high level
Synchronous mode:
1 = Idle state for clock (CK) is a high level
0 = Idle state for clock (CK) is a low level

bit 3 **BRG16:** 16-Bit Baud Rate Register Enable bit
1 = 16-bit Baud Rate Generator – SPBRGH and SPBRG
0 = 8-bit Baud Rate Generator – SPBRG only (Compatible mode), SPBRGH value ignored

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

bit 1 **WUE:** Wake-up Enable bit
Asynchronous mode:
1 = EUSART will continue to sample the RX pin – interrupt generated on falling edge; bit cleared in hardware on following rising edge
0 = RX pin not monitored or rising edge detected
Synchronous mode:
Unused in this mode.

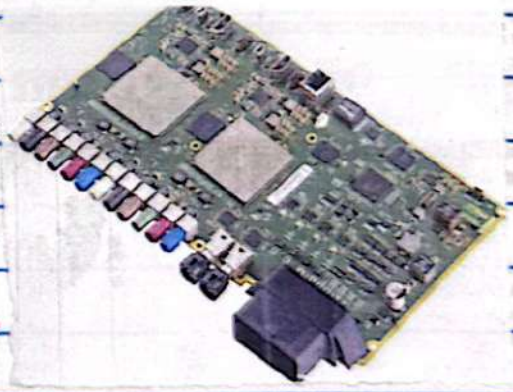
bit 0 **ABDEN:** Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Enable baud rate measurement on the next character. Requires reception of a Sync field (55h); cleared in hardware upon completion.
0 = Baud rate measurement disabled or completed
Synchronous mode:
Unused in this mode.

This register to control with the baud rate configurations.

Serial Peripheral Interface "SPI"

SPI specifications:

- Developed by Motorola 1980s.
- Synchronous Serial Communication Protocol.
- Master-slave topology.
- used for short distance communication. Between MCUs & Memory that are on the system on chip.
- May be it works at full & half Duplex Modes
- *Single Master single slave
  *Single Master Multi slave
  *Multi Master this depends on the type of MCU ex STM32 f407 MC.



Note the Master Provides the Communication Clock scl & clk.

Page:..... Date:.....

SPI Applications

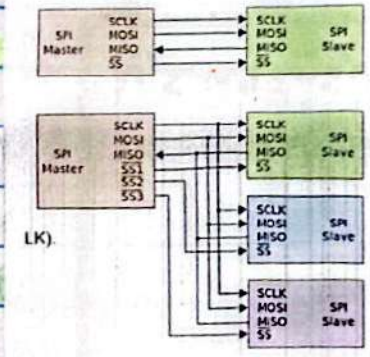
- MCUs Communications.
- SD Cards.
- external EEPROM.
- I/O expander Modules.
- wireless Communication Modules.

How SPI works as full & half duplex Modes?

there are SPI supports 4 Pins & other supports 3 Pins.

SPI 4 Pins : MOSI MISO SCLK SS

- MOSI : Master output slave Input: data going from Master to slave
- MISO : Master Input slave output: data going from slave to Master
- SCLK : serial clock : it's generated by the Master
- SS or CS : slave select or chip select the Master is limiting through which slave device will speak with it.



note

as you see this figure is 4 Pins SPI the Master has "Tx & Rx" & the slaves have "Tx & Rx" so the 4 Pins SPI is full duplex Mode.

the disadvantage of 4 Pis SPI takes More Pins : because the number of SS Pins at the Master will be increased if the number of the slaves Devices increases because each slave Device will take a Pin from the Master to the Mast can speak with the slave Devices

Page: ..... Date: .....

hote $\overline{SS}$ this Means active low. T = 0

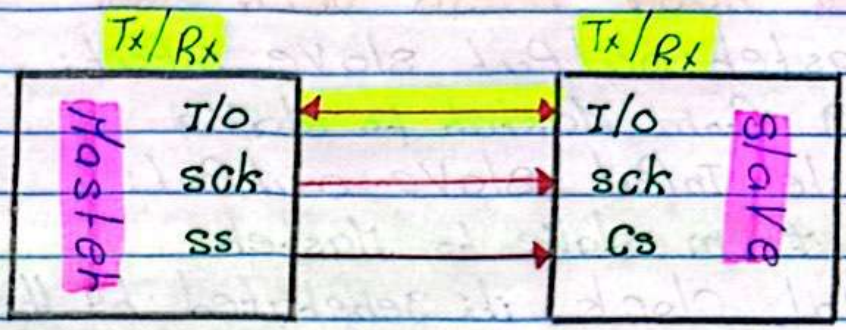
SPI 3 Pins I/O sck SS

→ I/O : is Data Pin works as "Rx & Tx" Can't be receiving & thansmitting at the same time.

→ SCk : is clock Pin it's genenated by the Master Device.

→ SS : slave sleck Pin : is to selecat which slave Device the Master will speak with it.

hote



as you see at the 3 Pins SPI there's just on line for the Data & line fhor the clock by the Master so 3 Pins SPI is half DaPlex Mode.

hote

the Master Can't thansmitte & Receive at the same time & the slave Device too.

the AdVantage & Disad Vantage of SPI

→ GPIO Push-Pull Configurations : Advantage

↳ there's no standard set of speeds defined for the SPI.

Is high speed Photocol depends on Fosc 10 Mbs ~ 20 Mbs - 50 Mbs this Means SPI Module has a direct line to the Fosc.

Page: ..... Date: .....

→ Higher throughput :- has no than there's no overhead of bits like stop & start bits.
→ lower Power requirements than I2C Protocol
→ Simple hard ware design than USART & I2C

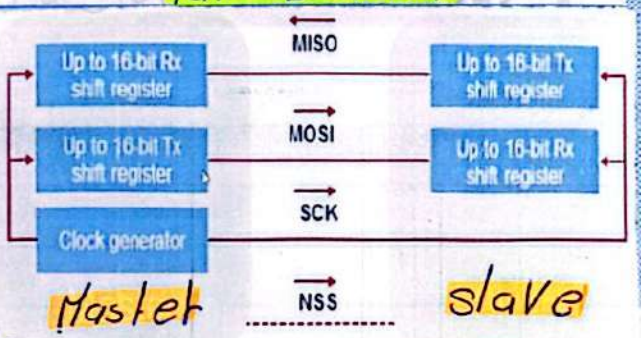
the disadvantage :-
→ no flow control, Master & slave must determine the communication speed.
the Master transmittes the Data with high speed & the slave can't receive the Data with this high speed ∴ the slave can't say to the Master decrease the speed this is the meaning of no flow control.

note
the flow control at the SPI is made Manually if you need it.

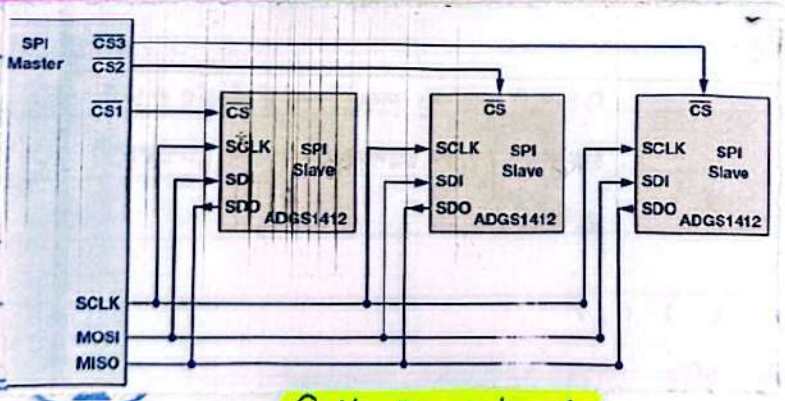
SPI Connection types

→ Direct Connection
each Pin at the Master is connected to another at the slave
Single Master single slave

full Duplex



→ Direct Connection single Master Multi slave
the Master will send "0" logic at th Cs of the target slave & the others will send for them "1" logic.

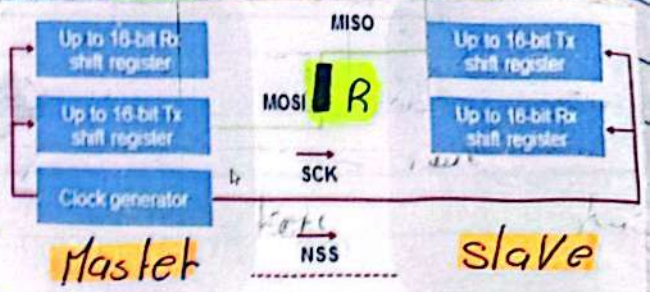


full Duplex

Page:..... Date:.....

→ Quasi Bidirectional half Duplex

3 Pins SPI #



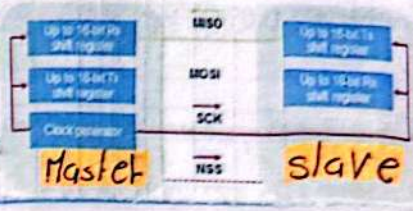
hole

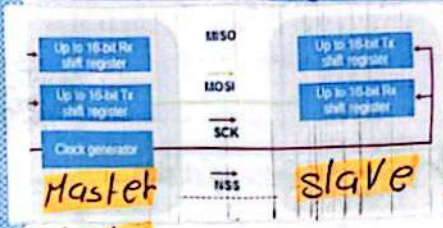
the resistance that over the Data line is to avoid the short circuit if the Master & slave send Data at the same time to don't burn the SPI Module inside the Micro Controller.

→ Unidirectional Mode "Simplex Mode"

the Tx is always Transmitter & the Rx is always Receiver.

Rx → Master
Tx → slave
for ever #



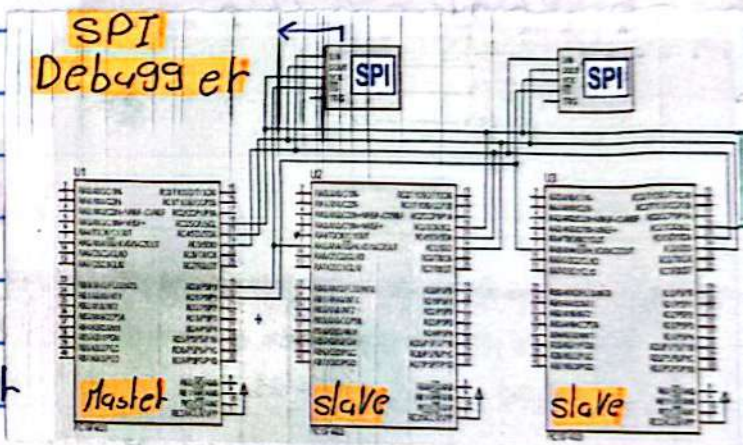


Tx → Master & Rx → slave for ever simplex Mode

hote

$\overline{SS}$ Pin is a GPIO Pin at the MC send over it "0" logic if you want to speak with the slave Device.

This way of the Connection is Single Master Multi slaves with SPI debugger

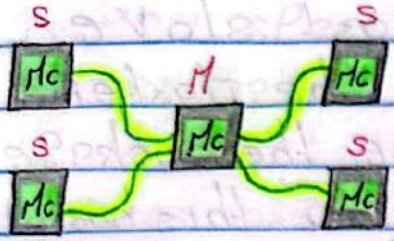


hote

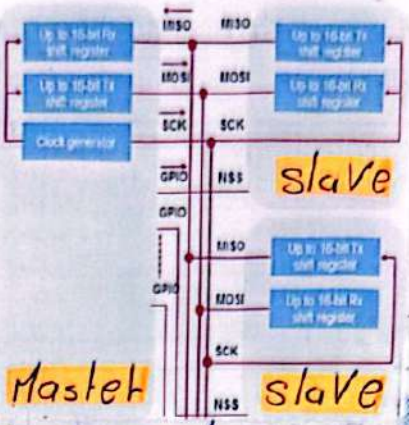
the debugger acts as the slave to see the Transfering of the Data.

Page:..... Date:.....

What's the stat topologies Connection



* Is Consider as a Master Device Connected to other slave Devices & the Master Control in every thing to the slave Devices.

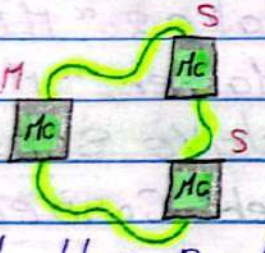


hote

at the SPI stat topologies Connection each slave MC Could be has different Configurations about the others slave Mc, & the Master when wants to speak with any slave device Must sent for it the suitable Clock according to the Configurations of the slave Device.

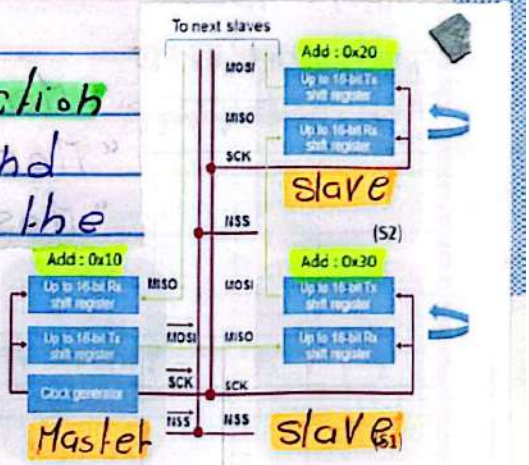
* slave nodes Can have different Clocks & different data formate.
* according to that the slaves Can have different Configurations.

What's the Circular topology Connection



Daisy Chain

* the Master send the Address of the slave Device firstly... after that send the Data Data slaveAddress

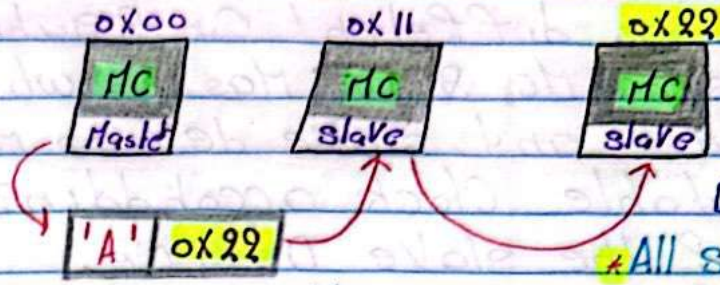


hote

each Device at the Daisy chain topology has an Address. the Master send the Address of the target slave after that send the Data

Page: Date:

note lets suppose that the Master wants to send Data to the last slave Device at the Connection the Master will send the backage where the backage is the slave Address & the Data this backage will go to the slave Device 1, the first slave Device at the Connection to the second to the third ..... till reach to the last slave Device at the Connection



note all nodes must support a common Data & Common Clock

* All slave Devices have the same Configurations.

note Comman ss/cs signal for all nodes the Communications occurs at the same time.

note * Master "MOSI" Connected to slave1 "MISO"
* Slave1 "MOSI" Connected to slave2 "MISO"
* Slave2 "MOSI" Connected to Master "MISO"

* Master clock line Connected to all slave SCKs
* Master ss "only one SS from the Master" Connected to all slaves SSs.

note the Daisy chain topology of the Circular topology SPI is so slow for transfering the Data so is not recommended to use it Very slow.

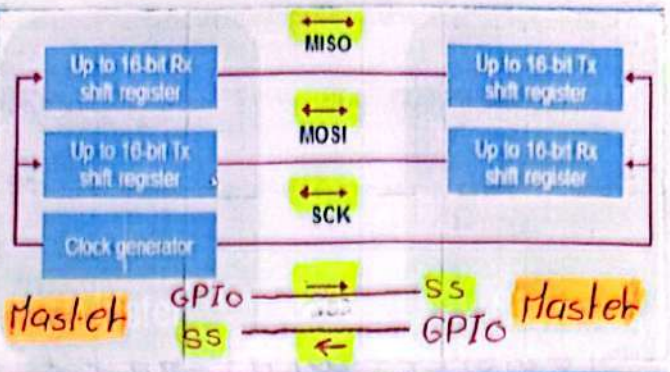
Page:..... Date:.....

Multi Master topology

is MCU specific

ex : STM32F4

by default the two Devices are at the slave Mode.



note if the node needs to communicate the node switch itself to active Master to control the bus.

→ responsible of clock generation & slave selection & start communication session.

it depends on the switching between the two devices but firstly the two devices go to the slave mode firstly.

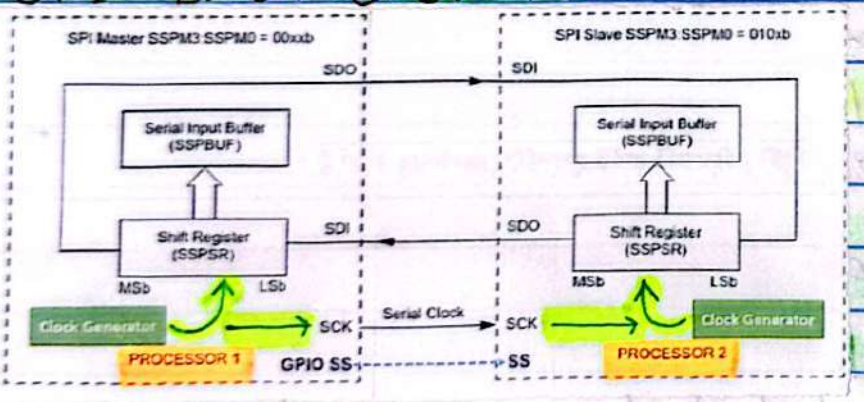
note

SS Pin : used as input to detect Potential bus Collisions → indicating start of communication

note

the Master node returns to the slave mode at the end of communication.

SPI block digram



note

any serial communication Module Must have shift register & internal clock

note

the buffer Register is the Data Register

Page:..... Date:.....

→ the Serial Input buffer register "Data Register" will take the Data from the Data bus after that download it at the shift register.

→ after that check ss Pin "0" logic low.

→ with each Clock to the shift register the register will shift the Data bit by bit.

→ after that the slave Device receive the sent byte & save it at it's shift register of the SPI Module.

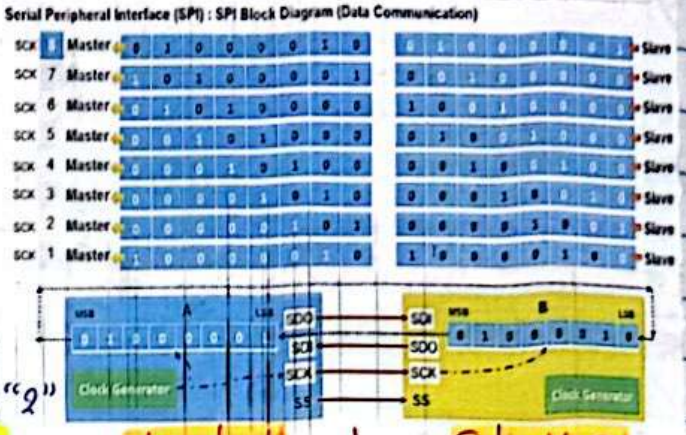
→ after the shift Register of the slave Device receive the completed byte it will rise a flag & save this byte of Data at Serial Input buffer " the Data Register of SPI slave Device".

Note

the same too if the slave Device will send to the Master Device.

Note

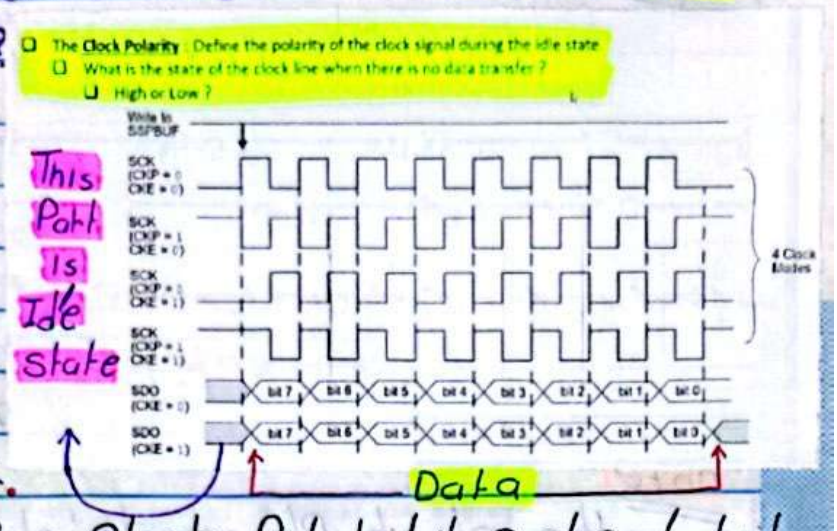
as you see at the block digram of SPI Module the SPI shift register of Device "1" is connected to the shift register of Device "2" there's exchanged Data between the two Devices.



this figure shows How the Data are transfered from the Master to the slave & the slave to the Master look at what's doing with each clock between the two shift registers.

Page:..... Date:.....

What's the Clock Polarity: it's the clock at the Idle state :: is defined as what's the state of the clock line when there's no data trans fer the ans is low or high?

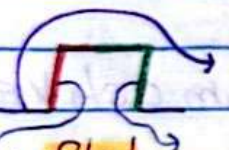


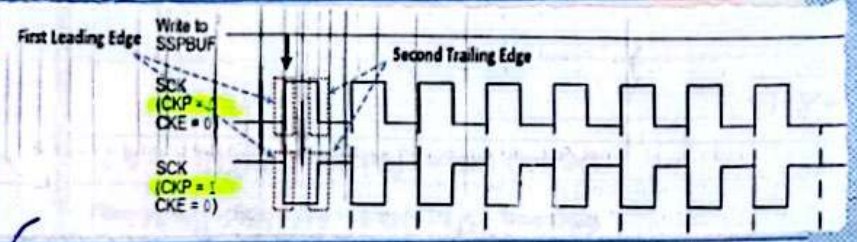
note

the idle state of the clock line is Configured low or high as you want by the Data sheet of the Micro Controller.

PIC18F46K20 Micro Controller : CKP : Clock Polarity select bit 1 = Idle high level 0 = Idle low level

what's the effect of determine the Clock Polarity select? the effect will be on the first leading edge & second trailing edge after the idle state.





↳ this figure shows the low & high

assume that Idle state "low" ⎍

→ the first leading edge = rising edge.

→ the second trailing edge = falling edge.

Idle state "high" ⎍

→ the first leading edge = falling edge.

→ the second trailing edge = rising edge.

Page:..... Date:.....

* the clock Phase :- is the shape of the clock. "شكل الـ clock عامل ازاى"

hote the clock Polarity & clock Phase together determine :- when the data be latched at the data line

hote the clock Phase determine the location that I want to transmit for it

hote any SPI Module has 2 important bit Clock Polarity & Clock Phase

hote the clock Phase determine when I will send the Data.

Pic18f46k20 MC Data sheet

CkP :- Clock Polarity Select bit

1 = idle state clock high level

0 = idle state clock low level

CkE :- SPI clock select bit

1 = transmit occours on transition from active to idle clock.

0 = transmit occours on transition from Idle to active clock state.

from these two bit I have 4 Modes on transmitting the Data

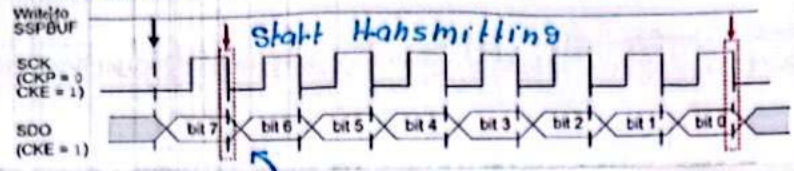
→ low idle transmit from active to idle
→ low idle transmit from idle to active
→ high idle transmit from active to idle
→ high idle transmit from idle to active

hote active is high if the idle low & active is low if the idle high

Page:..... Date:.....

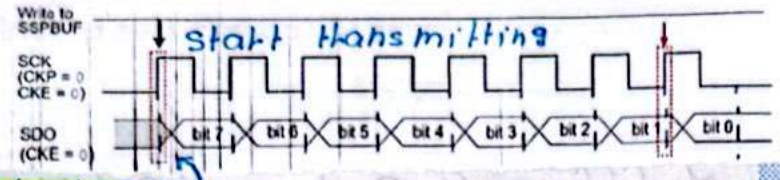
* here the idle is low & Tx from active H to Idle & idle is low so the Tx from High to low

The clock polarity & phase together determine when will the data be latched on the data line.
1) Clock Idle is "Low" and transmitting occurs on transition from active "H" to Idle "L" clock state.
- SSPCON1 <CKP : Bit-4> = 0 & SSPSTAT <CKE : Bit-6> = 1



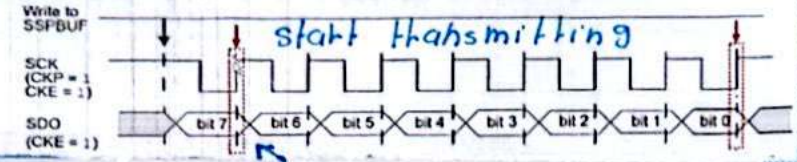
* here the idle is low & Tx from the Idle to active high & idle is low so the Tx from low to high

The clock polarity & phase together determine when will the data be latched on the data line.
2) Clock Idle is "Low" and transmitting occurs on transition from idle "L" to active "H" clock state.
- SSPCON1 <CKP : Bit-4> = 0 & SSPSTAT <CKE : Bit-6> = 0



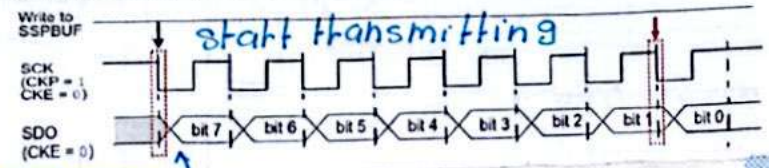
* here the idle is high & Tx from the active L to th idle & idle is high so Tx from the low to high

The clock polarity & phase together determine when will the data be latched on the data line.
3) Clock Idle is "High" and transmitting occurs on transition from active "L" to Idle "H" clock state
- SSPCON1 <CKP : Bit-4> = 1 & SSPSTAT <CKE : Bit-6> = 1



* here the idle is high & Tx from the idle to active L & the idle is high so the Tx from high to low

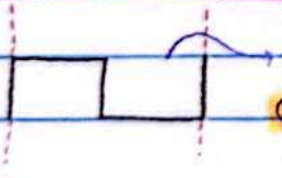
The clock polarity & phase together determine when will the data be latched on the data line.
4) Clock Idle is "High" and transmitting occurs on transition from idle "H" to active "L" clock state
- SSPCON1 <CKP : Bit-4> = 1 & SSPSTAT <CKE : Bit-6> = 0



SPI at receiving

to receive Data from the Master to slave or slave to Master there's a bit is responsible for take a sample at the stable form of the generated Clock from the Master it's called Sampling bit

Reading the Data



SSPSTAT

bit 7 SMP: Sample bit

SPI Master mode:

1 = Input data sampled at end of data output time

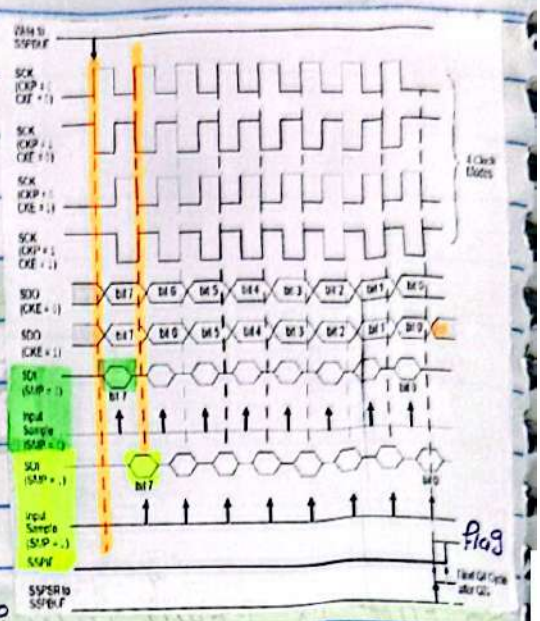
0 = Input data sampled at middle of data output time

SPI Slave mode:

SMP must be cleared when SPI is used in Slave mode.

Page: Date:

- Reading the Data at the Middle of Data out Put time.
- Reading the Data at the end of Data out Put time



Note

it's recommended to take the sample at the Middle of Data out Put time

Note

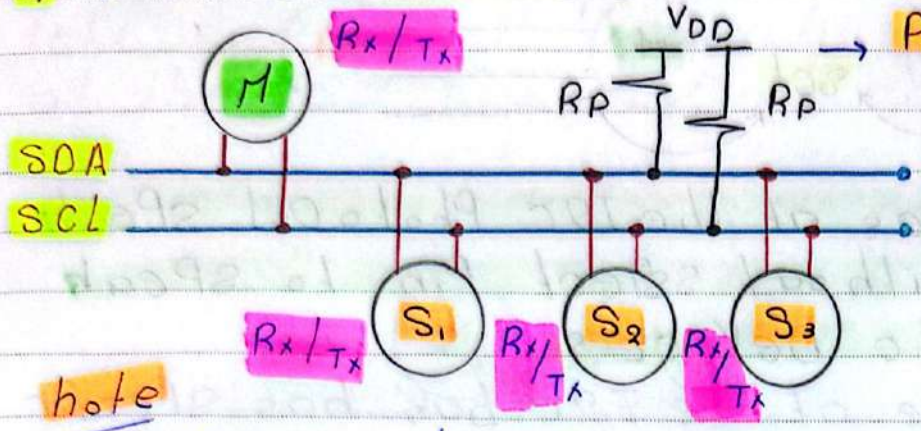
Sample :. Read Data.

Setup :. write Data.

I2C or IIC "inter integrated circuit"

I2C characteristics :

→ Serial Protocol :: one Data wire :: send bit by bit
→ Half duplex - Bidirectional



note

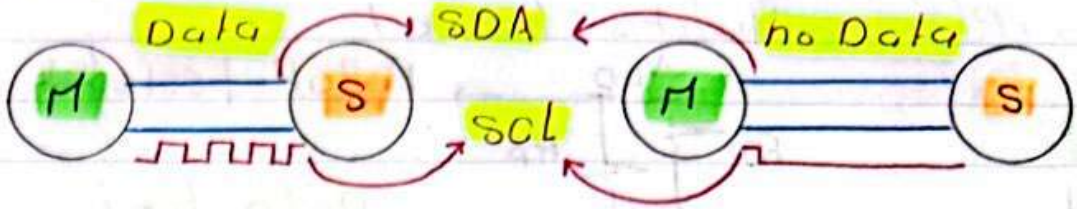
I2C is half Duplex Communication because I have just one channel for the Data & other for the clock ..

when

* the Master transmites Data the slaves can't transmite any thing and oppsite.
* the Master receives Data the slaves can't receive any thing and oppsite.

→ I2C is 2 wire bus :: Synchronous
* SDA :: serial Data. Protocol
* SCL :: serial clock. I2C
→ I2C is synchronous Protocol
→ Byte oriented "8 bits" send byte each time
→ used to make the Communication on the same PCB :: so is used for the short Distance.
→ I2C is Master slave Protocol

→ Master generates the clock at data transfer.
→ Master terminates the clock when no data transfer.



How the Devices at the I2C Protocol speak each other without select Pin to speak with a specific Device?

→ each Device at the I2C bus has specific Address the Address May be :
* **fixed Address** : for the sensor ... ect
* **Addressable Address** : for the MCUs is Software Addressable "**unique Address**" Can't be repeted #.

note
each Device on the I2C bus must have unique Address.

→ Master Can work as a Master Transmitter or Master receiver. Master Tx/Rx.

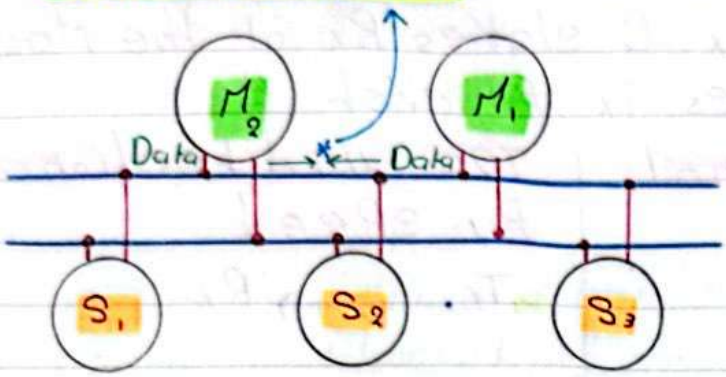
→ I2C supports "**Multi Master**" :- any Device Can generate the clock is Consider as Master Device. (M) ⎍⎍⎍ is Consider as a slave Device if it Can't generate the clock to send or receive the Data.

(S) no clock

note any Protocol supports Multi Master there are two important things must you know :-

* **Collision Detection**

lets suppose that we have two Masters at the I2C bus & each Master will send Data at the same time This will Make Data Collision



note I2C Protocol at the Multi Master Case support to you Collision Detection

* tells you if there's Data Collision or not ←

* **Bus Arbitration** → Bus الإمتلاك ال

lets suppose that both Masters want to send Data to different slave .

M1 → S1 & M2 → S2 & there's just one wire for the Data :-

each slave Device has Periority specific Address the slave that is high Periority Address it will receive the Data from its Master & other Master will be neglected.

Speed transfering Data by I2C

I2C supports Multiple speeds.

→ **I2C Bidirectional bus speeds**

note Bidirectional bus Means the Devices that are over the bus can act as Tx/Rx

1- Standard Mode. SM up to 100 kbit/sec
2- Fast Mode. FM up to 400 kbit/sec
3- Fast Mode Plus. FM+ up to 1 Mbit/sec
4- High Speed Mode. HSM up to 3.4 Mbit/sec

→ I2C unidirectional bus speed :)

1- Ultra Fast Mode U-FM up to 5 Mbit/sec

Note

unidirectional bus means that the Master will act as Tx & slaves Rx or the Master acts as Rx & slaves Tx for ever

I2C Bidirectional Bus speed	I2C unidirectional Bus speed
→ RX/TX ⟷ TX/RX	→ Tx ⟶ Rx
→ Support Multi Master.	→ doesn't support Multi Master option

Note

all Devices at the bus must have the same speed. ↳ I2C bus

Note

at Multi Master option I2C I have :

→ Collision Detection : Data crashed
→ Bus Arbitration : which Master send or Receive
→ Clock Synchronization : for the clock generation

at no Multi Master option I2C

I just have the clock synchronization only

Note

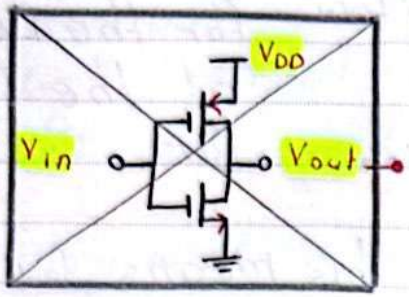
the Pins of I2C bus "SDA - SCL" must be open drain or open Collector output drivers.

whats the different between the open drain & Push Pull :-

note

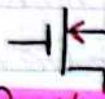
each Pin at the MC has hardware implemention.

This is the Pin of the MC



note

the hardware Pin implemention May be Mosfet transistor or BJT transistor.

 PMOS  NMOS

Push Pull Configrations

firstly Consider the typical CMOS "inverting" output stage CMOS :- Complementary Metal Oxide Semiconductor.

Inverting Means :- $V_{in} = 0 \rightarrow V_{out} = 1$ & $V_{in} = 1 \rightarrow V_{out} = 0$

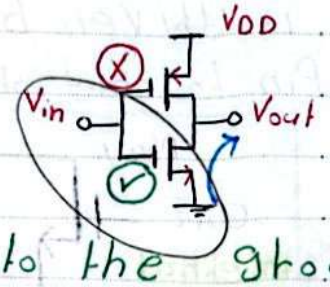
if the Input is logic high

the PMOS is off

the NMOS is ON

$V_{in} = 1 \rightarrow$ Inverting $\rightarrow V_{out} = 0$

because the Vout is Connected to the ground



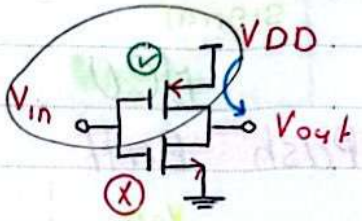
if the Input is logic low

the PMOS is ON

the NMOS is off

$V_{in} = 0 \rightarrow$ Inverting $\rightarrow V_{out} = 1$

because the Vout is Connected to the VDD



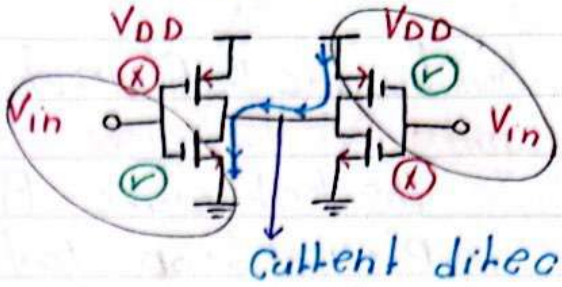
note

SDA & SCL Pins Must be open drain

note

at the Push Pull the Pin has PMOS & NMOS

why SDA & SCL Pins Must be oPen dtain if we use the Push Pull :



hote

at this Case Maybe happening damage for the DeVices at the I2C Bus.

hote

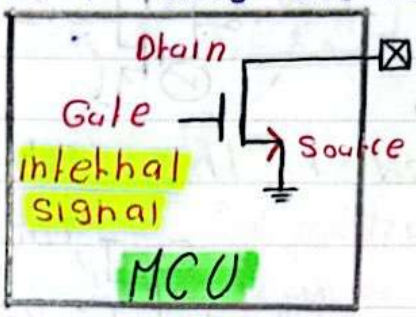
If you use the Push Pull this means you use the unidiTectional I2C Bus & use the ulthal fast Mode sPeed & you just haVe one Mastet DeVice.

hote

If you use the oPen dtain this means you use the BidiTectional I2C Bus & use the sPeeds like : SM fM - fM+ - HSM & suPPott Multi Mastet.

oPen dtain Con fiGutations :

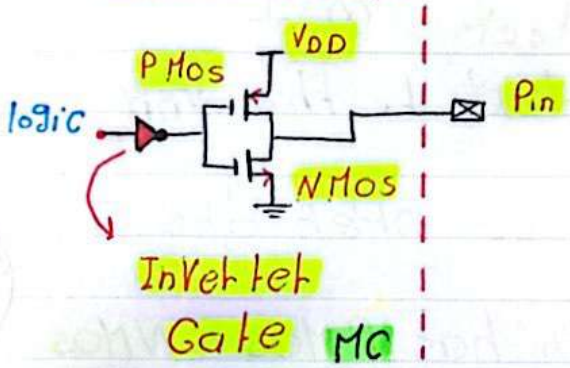
Pin is dtiVen by a single thansistot which Pulls the Pin to only one Voltage " 5V, Ghoung"



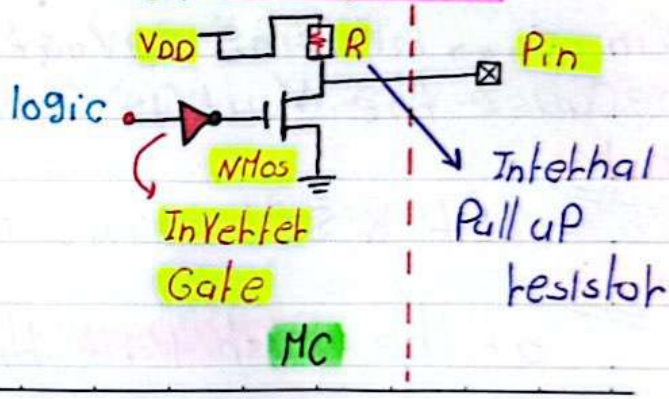
hote

at this figute the Voltage May be logic one ot Ghound.

Push Pull

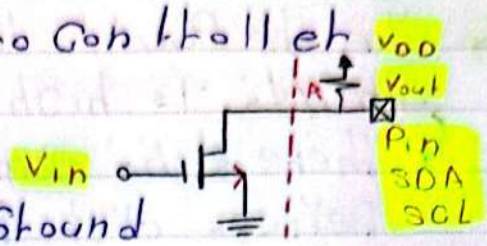


oPen dtain



oPen drain Con figrations

→ the PMos thansistor has been tePlaced by a tesistor extethal to the Micho Con tholl eh



if the Vin is logic high
the NMos will active & Vout will Connected to the Ground & the Pin will have logic "0".

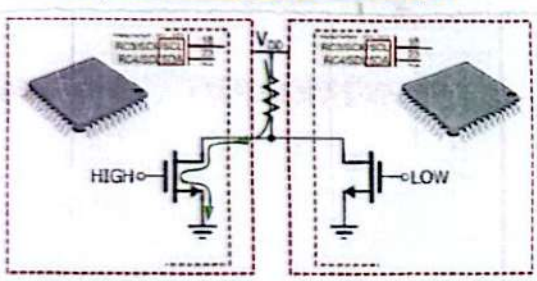
if the Vin is logic low
the NMos will deactive & Vout will Connected to the VDD & the Pin will have logic "1"

hote
the tole of the tesistor is to Contholl in the Value of Cuttent that will be on the Pin I/O.

hote
we don't have Intethal Pull uP tesistance at PIC18f46k20 Micho Conthollet so we Make it is extethal on the Pin if we heed it.

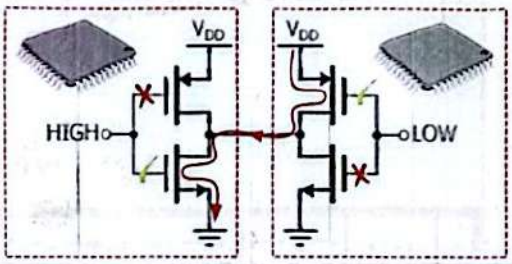
How the Cuttent Pass at the Push Pull & the oPen drain

oPen drain



hete the NMos of just one DeVice will be active & othet is deactive.

Push Pull

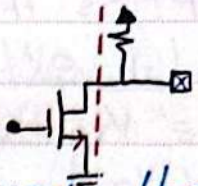


hete the DeVices May be damaged if the Vin of one of these DeVices gets a fault logic.

Here three important implications of the open drain bus

الآثار المترتبة

a→ the default logic on the Pin is always is high at the idle state.



↳ there is no communication between the Devices over the bus.

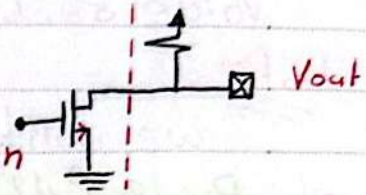
↳ if the slave Devices has no received Data it will read logic high.

Note

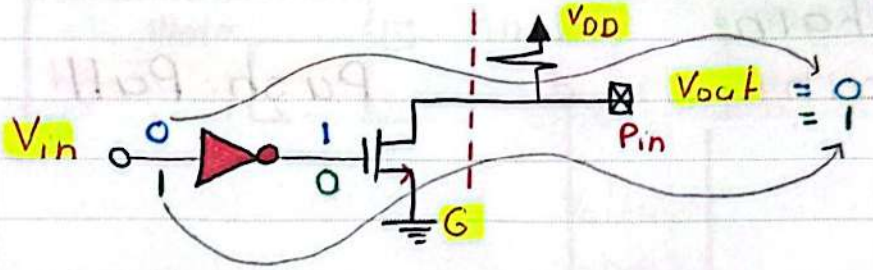
when Vin = 0 the NMos will be Deactive
when Vin = 1 the NMos will be Active

But

when Vin = 0 the Vout = 1
when Vin = 1 the Vout = 0



↳ this is not logic so I will use the Inverter to make the function of the open drain is Note logic



b→ the And wiring feature :

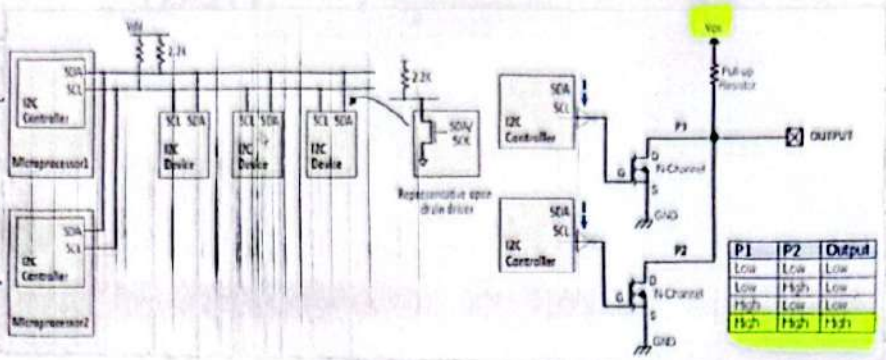
as we know all Devices at the I2C bus are connected at the two wires :

the Data bus & Clock bus when Connect SCL_1 with SCL_2 thes Makes And output

Device Device for the two Devices.

look at to this figure to know

→ here two Masters at the I2C bus each one has I2C Module & both are connected together at the scl line & Makes out Put line goes to the slaves Devices #



P1	P2	Output
Low	Low	Low
Low	High	Low
High	Low	Low
High	High	High

If you have MC3 you will get P3

note

the And wiring support to me the Clock synchronization & clock stretching I2C feature.

→ you can select different voltages are connected to the Devices that are over the I2C bus Make V_{DD} = 5V or V_{DD} = 3.33V or you can use the Bi directional logic level Convertet → 5V → [BID Conv] → 33.V

I2C start & stop Condition

note

the only Device can send the start & stop Condition is just the Master Device.

start Condition

there's Data is sent on the bus.

stop Condition

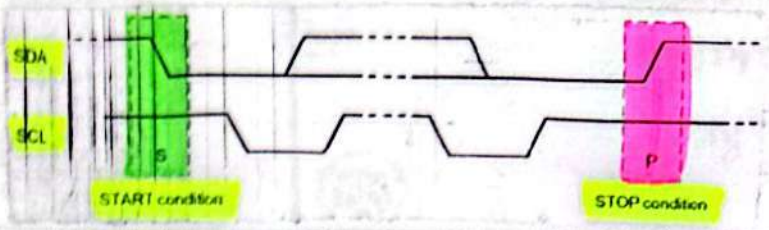
there's no Data is sent on the bus.

note

the start & stop Condition are associated by the SCL & SDA lines.

Note any Data is transmitted on the I2C bus must start with start Condition & end with end Condition.

* as you see this figure →



Start Condition: the transition from high to low at the Data line while the Clock line is high.
↳ SDA ↳ SCL

Stop Condition: the transition from low to high at the Data line while the Clock line is high.
↳ SDA ↳ SCL

Note the I2C bus is consider to be busy after the start Condition.

Note
it's recommended at the Multi Master Case to don't break the communication between the Devices by the stop Condition but you Make the repeted start Condition #.

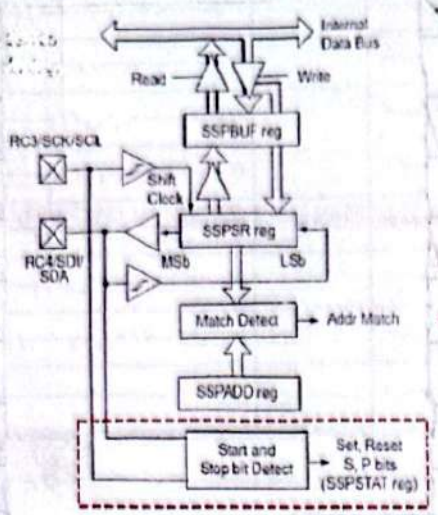
Note
If you have just one Master you can send the stop Condition & send the start Condition one Make time.

Note
the repeated start Condition is simalar to the start Condition. "same specifications"

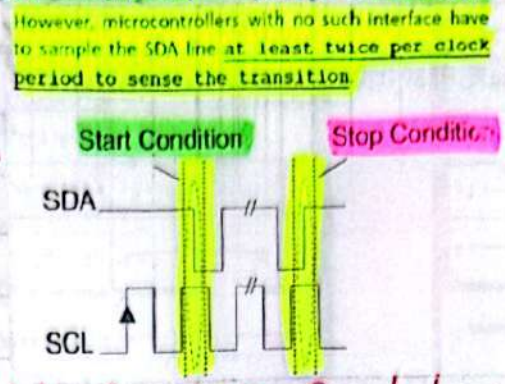
Note
the I2C bus will be free after the stop Condition.

How the Devices at the I2C bus know if the Master send the start or stop Condition

each Device has I2C Module & inside the I2C Module there's IC is responsible for Detect the start & stop Condition.



what will you do if the Device don't support the start & stop Condition Detection? you will Make it by the **soft wake** by take sample at the transition at the Data line & clock line.

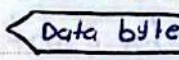


Acknowledge bit

note

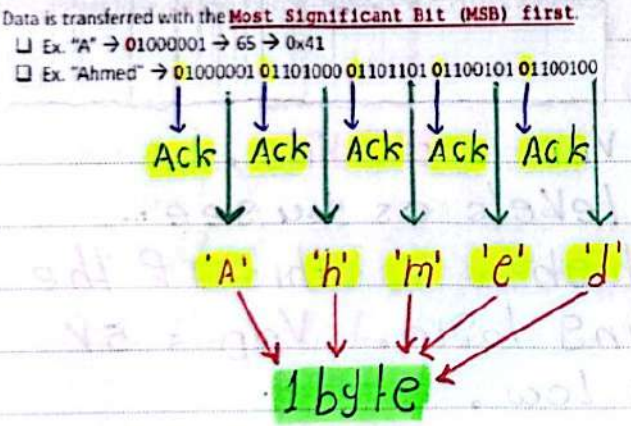
I2C is byte oriented Mechanism this means send the Data bit by bit & the first bit is sent is the **MSB** till reach to the **LSB**.

→ **the Acknowledge bit**

this bit says if the byte of Data has been sent successfully or not. So the sent byte is Consider as:- byte of Data + Ack bit 

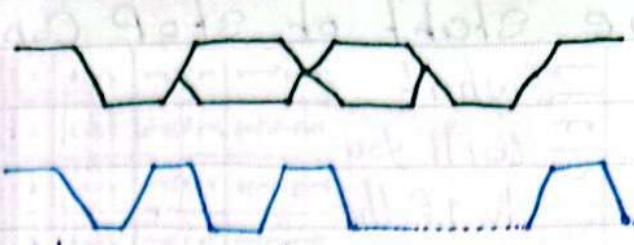
.

← look at this example



→ each chatachters of those is Consider as 8 bits of the Data the next bit is Ack #.

SDA & SCL logic levels :-



* as you see the SDA & SCL genetate group of Pulses Pulses are high & Pulses are low.

* here the Question : what's the value of high Pulse & the Value of low Pulse in the Voltage ? this it's called Data & clock lines logic levels #

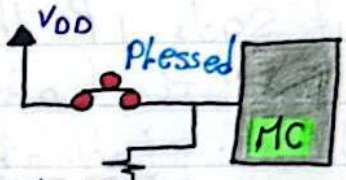
note as we know that there are different Devices are connected at the I2C bus & each Device it works with different V_{DD} & this Due to the different techology of Devices like Devices have : CMos - NMos - BiPolar according to the technology of the Device the V_{DD} of the Device May be 5V or 3.33 V according to the Device is CMos - NMos - BiPolar.

the InPut refereh ce levels

How Much of the V_{DD} "Value" % V_{DD}

ex → Pressed → Pin = V_{DD}

unPressed → Pin = 0



* as a Micro Controller there's a different definition for the high & low value.

→ Voltage InPut low = 30% V_{DD} = 0.3 V_{DD}.

→ Voltage InPut high = 70% V_{DD} = 0.7 V_{DD}.

So the range of the logic levels as you see :-

3.5 ⟶ 5 V ⟶ logic high. this if the

1.5 ⟶ 3.5V ⟶ floating logic. V_{DD} = 5V

1.5 ⟶ 0 V ⟶ logic low.

if the VDD = 3.33 V

2.3 ⟶ 3.33 V ⟶ logic high.

0.99 ⟶ 2.3 V ⟶ floating logic.

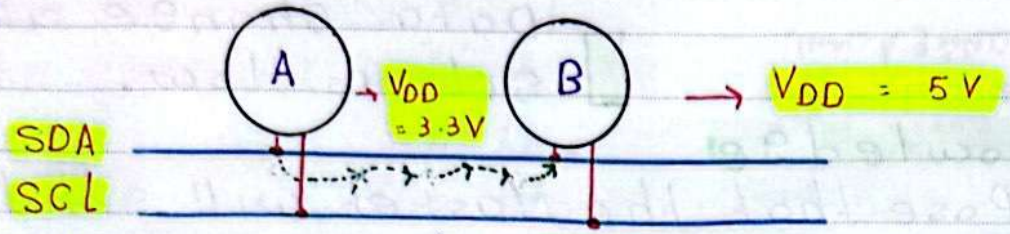
0.99 ⟶ 0 V ⟶ logic low.

hole

May be the DeVices at the I2C bus have diffeRent Voltage InPut low & high.

How will we Connect these DeVices at the I2C bus with diffeRent Voltage InPut low & high VDD?

→



DeVice "A" sends high Pulse to DeVice "B"

↳ VDD = 3.3V

↳ high Range : 2.3 → 3.3 ⟶ ↳ VDD = 5 V

↳ float Range : 1.5 : 3.5

as you see the high Range foR DeVice "A" is ConsideR as float Range to the DeVice "B" so the DeVice "B" will not undeRstand the high Pulse by the DeVice "A".

So we Must Use the Bi-diRectional logic leVel ConVeRteR

logic leVel EQ

Voltage InPut low = 30% VDD

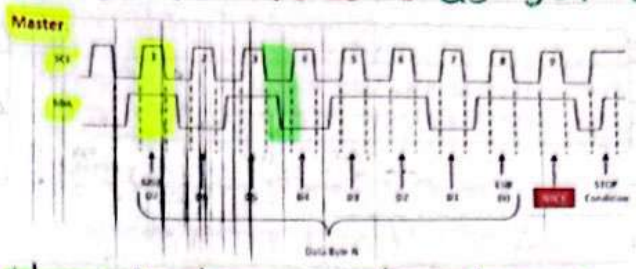
Voltage InPut high = 70% VDD

foR the MicRoContRolleR Pin InPut.

the Data Validity

→ the Data on the SDA line Must be stable during the high Period of the clock

→ the high or low state of the data line can only change when the clock signal on the SCL line is low. as you see the following figure



Data is stable at SCL is high.

Data change at SCL is low.

the Acknowledge

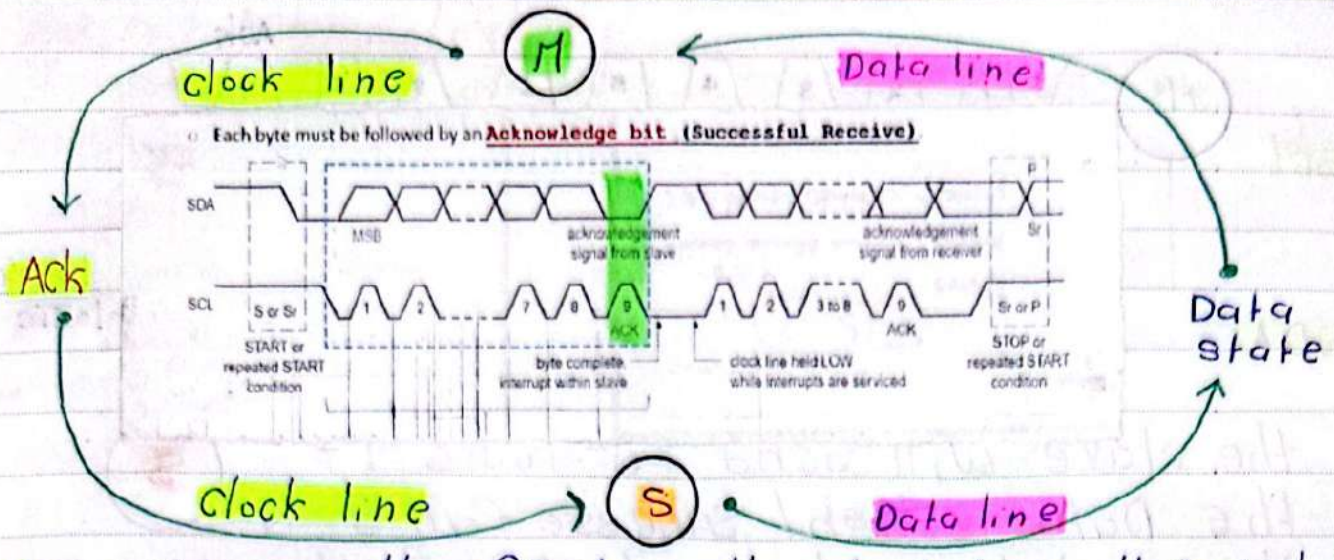
let's suppose that the Master will send Data "byte" to the slave the Master will send 8 bits of Data & bit9 will be for the ACK bit when the slave receives the Data byte from the Master the slave will send the ACK to the Master to tell the Master if the Data has been received successfully or not.

note

after each sent byte there's ACK.

note

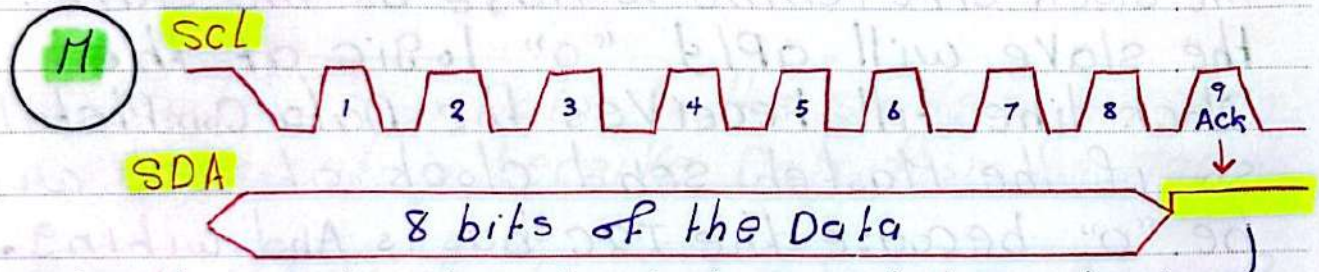
the Master allows to the slave the ACK bit to the slave tells the Master if the Data was sent successfully or not. the Master generates all Clock Pulses including the ACK bit9 at the clock line.



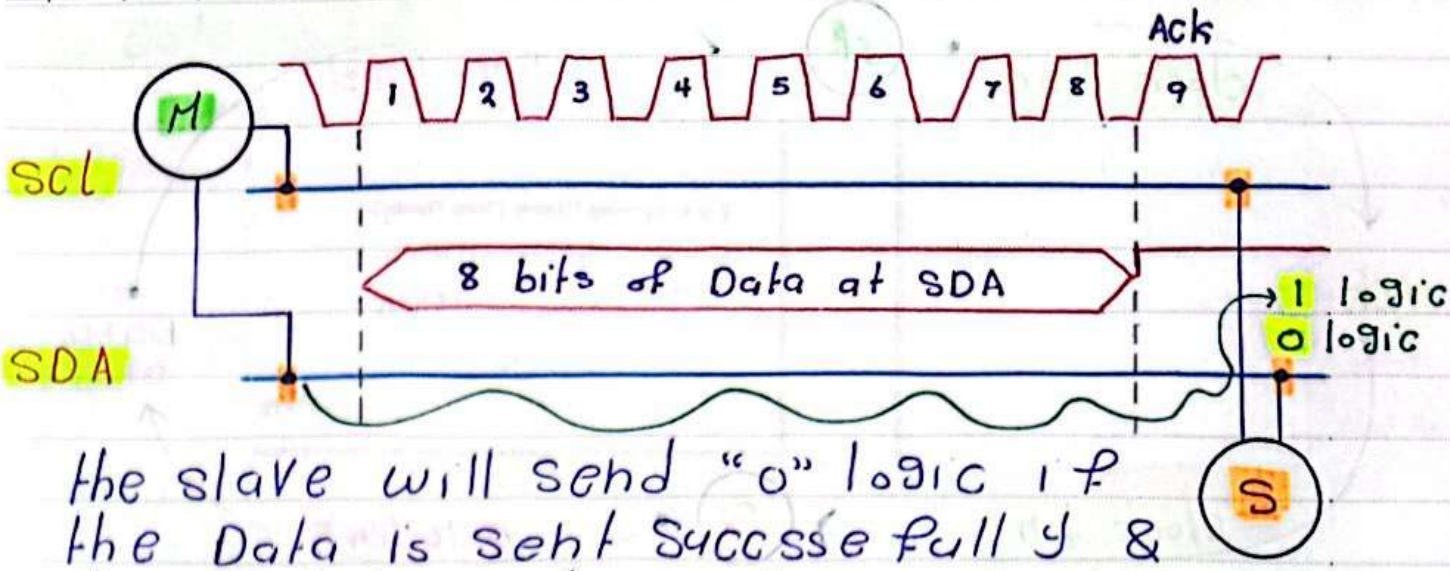
* as you see the figure the Master will generate the clock at SCL for 8 bits of Data & the bit9 is for ACk at the clock line to the slave after that the slave will send at the Data line the result of the ACk clock to the Master & the result May be logic high or logic low.
* what's the difference between the Acknowledge & Not Acknowledge ::

the Acknowledge ::

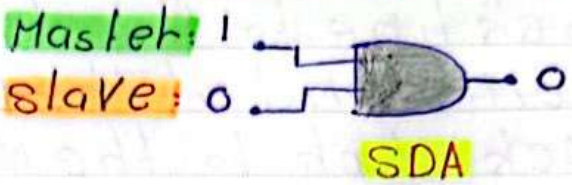
the Master will generates the clock of 8bits & the bit9 for Ack at SCL during the bit9 Ack at SCL the Master will send logic "1" at the SDA

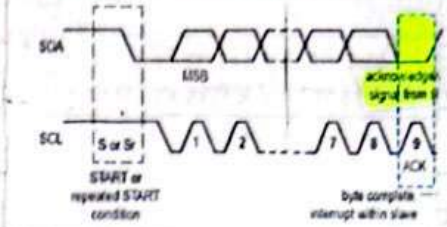


* at the Ack the Master send logic high at the Data line "SDA"
* if the slave received the Data Successfully will send logic "0" at the SDA line at Ack bit9



the slave will send "0" logic if the Data is sent successfully & the Master send "1" logic & the I2C bus is And wiring so the final result is "0" logic





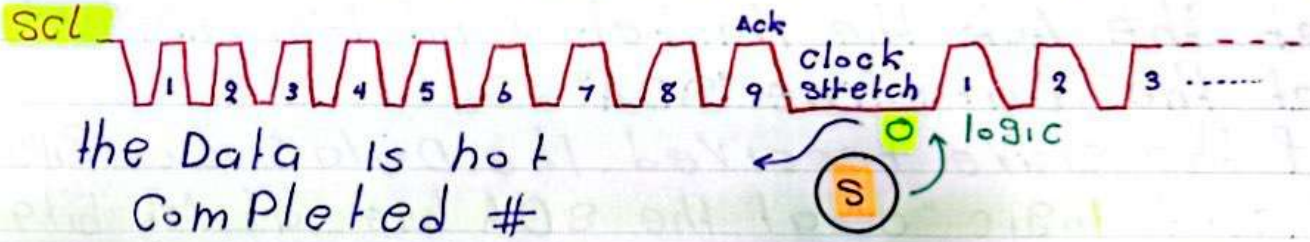
the final result of the Ack

note

May be the slave is slow to receive the Data so is not ready to receive another byte of Data in normal the slave will send not Ack but rather than the slave sends not Ack will Make **Clock stretching**

what's the Clock stretching

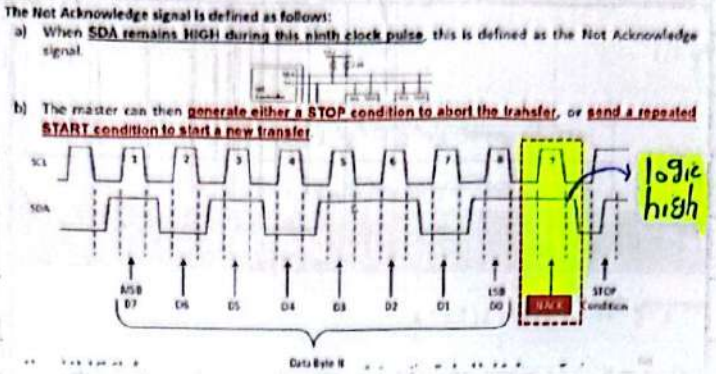
the Clock stretching is Made by the slave. the slave will aply "0" logic at the Clock line till receives the Data Completely so if the Master send clock at SCL will be "0" because the I2C bus is And wiring.



the Data is not Completed #

the hot Acknowledge :

the same thing at the ACk the Master will send 8 bits clock at SCL & bit9 is for Ack & the Master will apply logic high at SDA if the slave can't receive the Data will send the NACk as white logic high at SDA line & the I2C bus is And wiring so the result will be logic high at SDA line.



the NACk reasons :

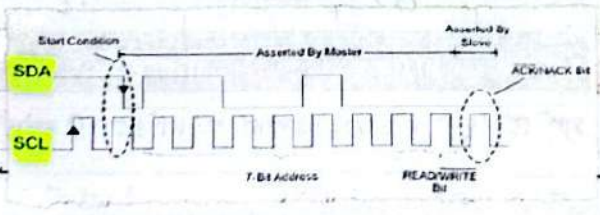
→ the Master sends an Address of a slave Device is not existed at the I2C bus.

→ the Master sends Data to the slave & the slave is busy for Performing a specific task like ISR so the slave can't receive the Data.

→ the Master sends Data is not understood to the slave Device.

→ the Master sends Data to the slave & the slave doesn't want to receive Data from the Master.

→ the Master doesn't want to receive Data from the slave.



hote

the reciVer whateVer is Master or slaVe that is resPonsible for generating the ACK.

the slaVe Addressing :

the Master when wants to sPeak with a slaVe DeVice it will send firstly the Address of the slaVe DeVice after that it willsend the Data at the bus.

hote

the I2C bus has some slaVe DeVices & each slaVe DeVice has uniQue Address.

there are two slaVe Addressing schemes

are Made by NXP ComPany which Made the I2C bus. the schemes are 7-bit & 10 bit Addressing

hote

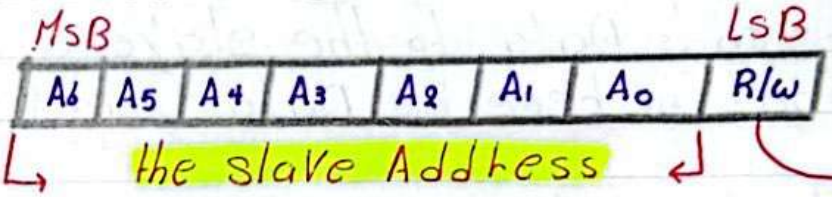
the diffeRent is in the size of the slaVe Address.

hote NXP ComPany Made 10 bit Addressing scheme. to increase the number of the slaVe DeVices to able to get More slaVes at the I2C bus.

7-bit Addressing scheme

the Master will send the start Condition after that the Control byte.

Control byte



hote

write ⟶ R/w = 0

Read ⟶ R/w = 1

this is determined by the Master DeVices.

Q: if we have 7 bit Address range that Means that we have $2^7 = 128$ Address?

ans: No, because we have reserved Addresses at the I2C bus.

Example of these [Reserved Addresses] → Will be discussed soon

Slave Address	R/W Bit	Description
000 0000	0	General call address
000 0000	1	START byte
000 0001	X	CBUS address
000 0010	X	Reserved for different bus format
000 0011	X	Reserved for future purposes
000 01XX	X	Hs-mode master code
111 10XX	X	10-bit slave addressing
111 11XX	X	Reserved for future purposes

note from the reserved Addresses is the 10 bit slave Addressing

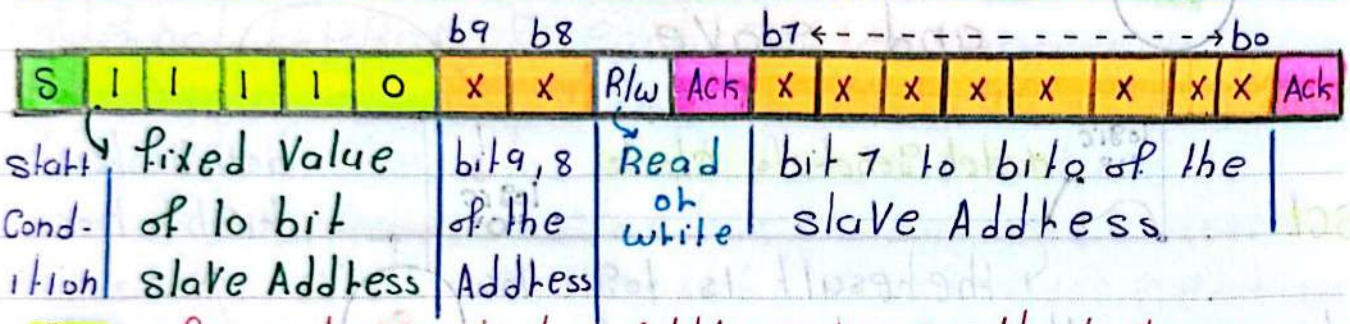
10 bit slave Addressing scheme

111 10 xx ⇝ bit 1:0 May be have zero or one fixed value for the 10 bit Addressing

note

it Possible for the I2C bus has slaves work with 7 bit & 10 bit Address at the same Project there's no Problem.

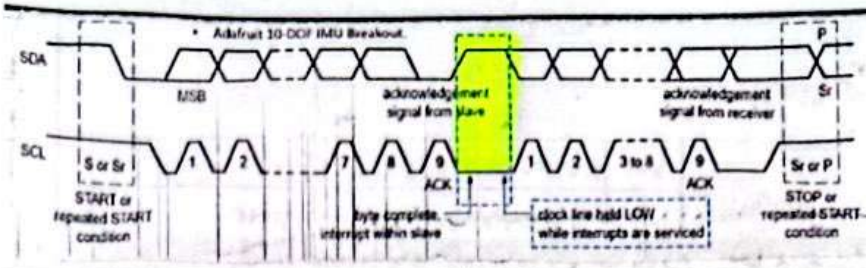
How the 10 bit Address is sent at the I2C bus



Q: if we have 10 bit Address range that Means that we have $2^{10} = 1024$ Address?

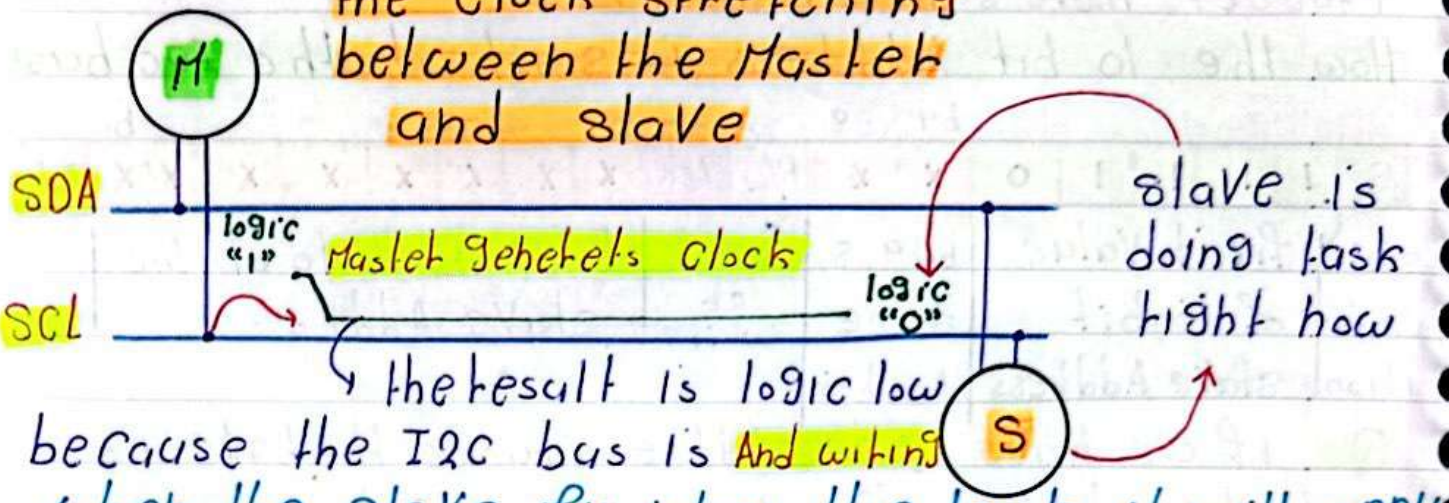
ans: No, because we have reserved Addresses at the I2C bus too.

Clock stretch is rather than the NACK Indication. the reciver send logic low at SCL line.



How the Clock stretching works ?

if you have at the I2C bus a slave Device can't receive the Data from the Master because the slave Device is busy for execute a specific task it will Make a Clock stretching rather than send NACK as an indication to stop the Master sending the Data till the slave Device finshes it's task the slave Device will send logic low at the clock line SCL so even the Master genetates a clock it will be not understood because the slave device is applying logic low.



because the I2C bus is And wiring

when the slave finishes the task it will apply logic high at the clock line.

hote

there are some slave Devices doesn't support the clock stretching so when the Master send for them Data they will send NACK what's the solution ?

you will the speed of the Master with the speed of these Devices #.

---

the clock synchronization & bus Arbitration

these two concepts must be existed at any Protocol supports the Multi Master. like I2C

note

→ if I have just one Master at the bus so I don't need the clock synchronization & bus Arbitration.

note

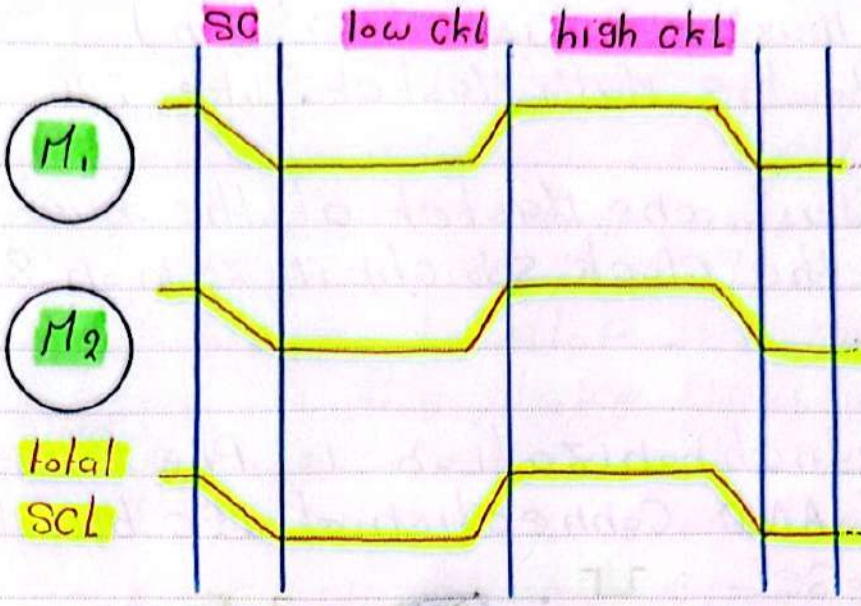
the clock synchronization is performed using the wired AND connection of I2C bus to the clock line SCL.

ex let's suppose that we have two Masters at the I2C bus & slave Device like the external EEPROM & each Master wants to write Data inside the EEPROM & each Master has generated the clock at the same time. how I have two clock at the clock line Master1 clock & Master2 clock so I need to Make a Clock synchronization to the two Master & the slave Device " the external EEPROM".

note

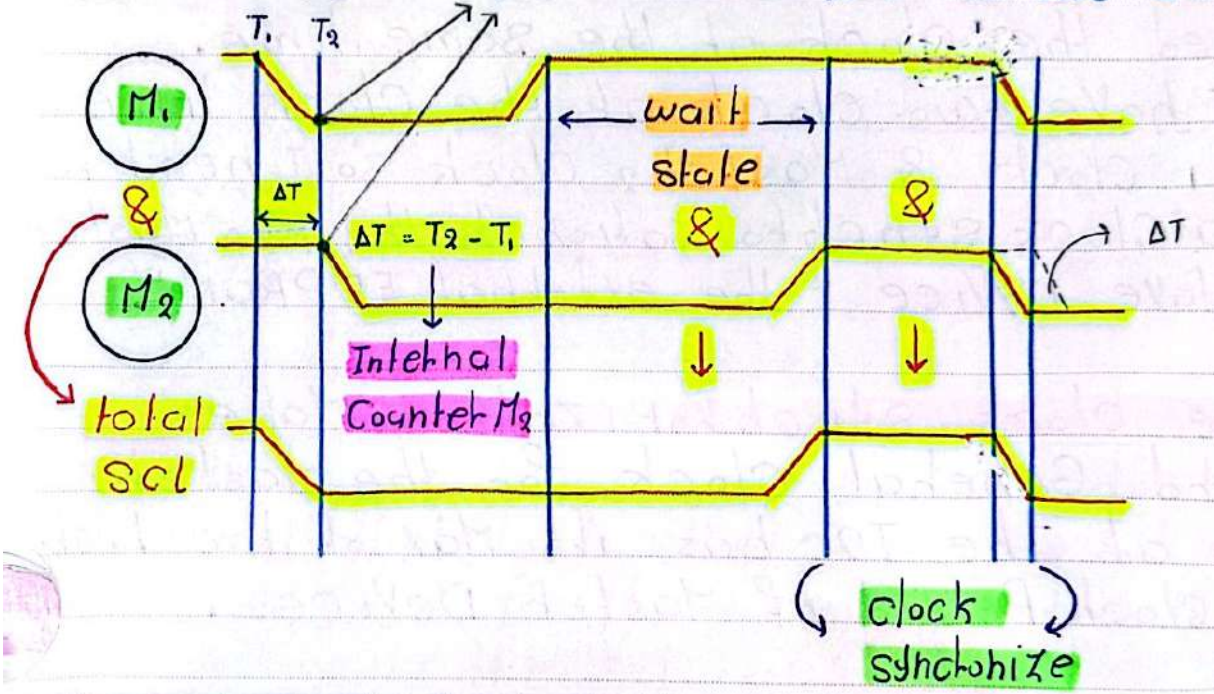
the clock synchronization Makes standard General clock for the Master Devices at the I2C bus it's Mix of the high & low clock Period of Master Devices.

example ① I have two Masters at the I2C bus & each one generates the clock at the same time & both Masters have the same speed Clock generation :-



As you see the total SCL is the same at Master1 CkL & Master2 CkL because Master1 CkL = Master2 CkL & I2C bus is And wiring so the Clock Synchronization is the same at the total SCL . every thing is the same SC & low & high clock Period

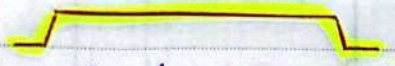
example ② I have two Masters at the I2C bus & each one generates the clock but not at the same time :- the clock is not at the same time



→ Master1 generates the clock at T_1 & Master2 generates the clock at T_2 the time is different for the two Masters.

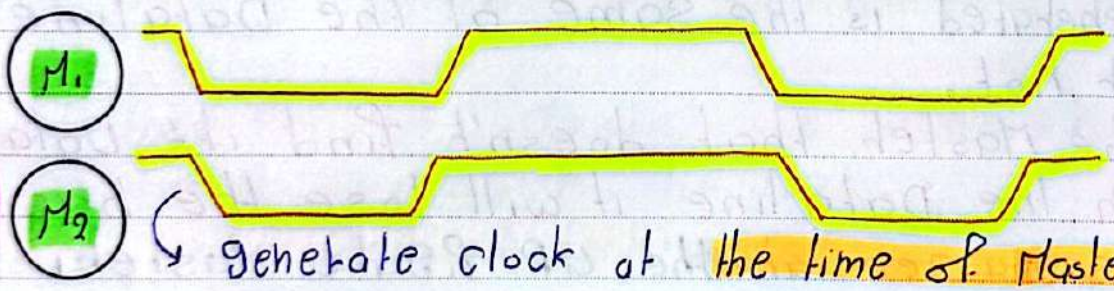
→ Master2 has time delay about Master1 so there is an internal Counter at Master2 Counts the time from the logic high till the transition to the logic low at the clock generation of Master2 Device is ΔT where $\Delta T = T_2 - T_1$

→ how Master1 generates clock high & Master2 still generating logic low so the Master1 clock will go in the wait state Mode till Master2 generates clock high.

→ how Master1 generated biggest clock high Period  & Master2 generated biggest clock low Period  so after this backage of Periodes the CPU will Make the clock synchronization by low & high Period of Master1,2 clock  CKL

→ how Master2 knew when the Master1 will generate the clock by calculating the ΔT

after the clock synchronize



↳ generate clock at the time of Master2 $- \Delta T$.

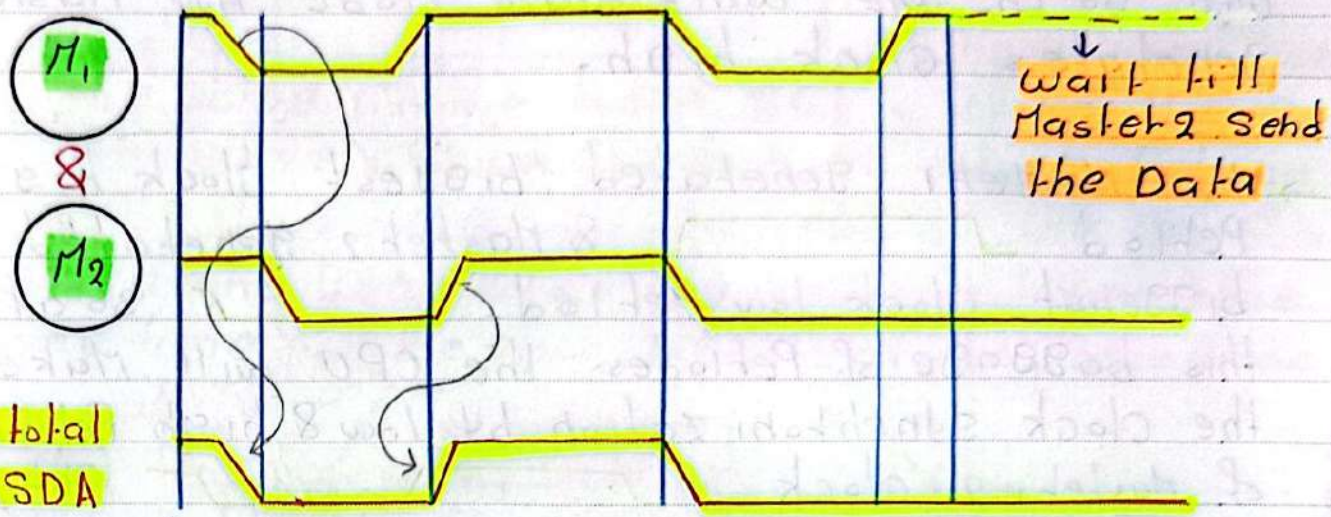
hole the clock synchonization improves the clock at the clock line SCL.

hole the bus Arbitration determines which Master will send or receive the Data.

The Bus Arbitration

if there are two Master Devices want to send Data at the same time on the Data line SDA the Bus Arbitration feature that determines which Master Device will send the Data.

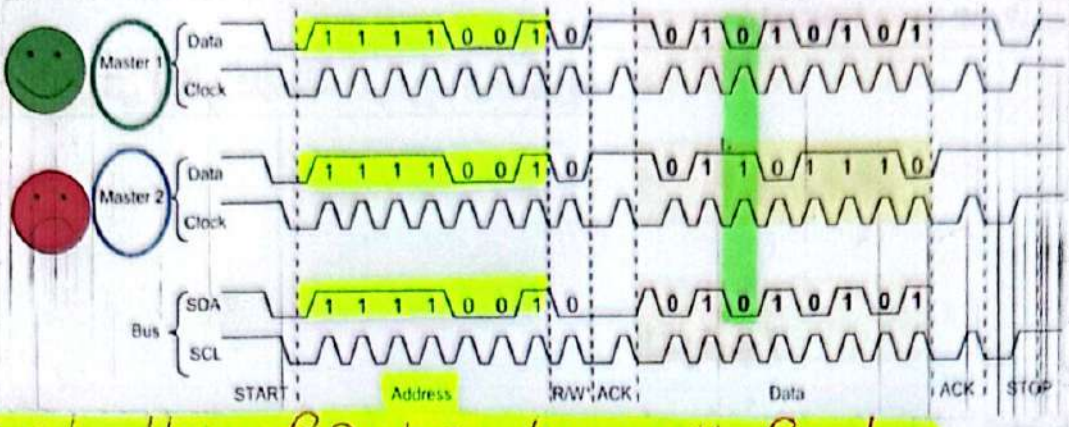
example :



→ each Master Device will generate Data Pulse & will go to the Data line to see if the Pulse Data that the Master Device has generated is the same at the Dataline SDA or not.

the Master that doesn't find it's Data Puls on the Data line it will lose the bus as you see at the UP figure Master1 lose the bus.

hote the Master that lose the bus will wait till the other Master send the Completed Data after that the loser Master can send it's Data at the Data line SDA

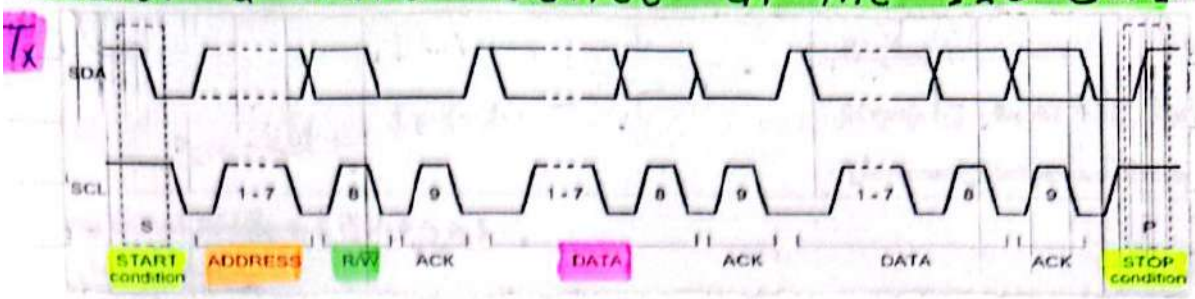


look at this figure you will find :
two Master Devices want to send Data.
which one will send the Data?
all of them send the Address of the slave Device firstly after that determines if the Master Read or write by R/w bit & send the Ack bit after that each Device is Master will send the Data bit by bit & will see if the generated bit of Data by the Master is the same at total SDA Data line, the Master that send Data & will not find the same Data at the Dataline will lose the Bus & will send the Data after the winner Master Device sends the Completed Data.

hote
at this figure Master1 is winner & Master2 is loser the bus.

This is the Bus Arbitration

the Communication Processing between the Master & slave Device at the I2C bus



look at the steps of the I2C bus transmission :-
here the Master Device wants to speak with slave Device

1- the Master will send the start Condition.
2- the Master will send the Address of the target slave Device that the Master wants to speak with it.
3- the Master will send R/w bit to determine if the Master wants to Read or write from or in the target slave Device :-
   * R/w = 0 :- Master needs to send Data to the slave.
   * R/w = 1 :- Master needs to receive Data from the slave.
4- the Master will send Ack bit & the slave or the Rx Device will reply if it received the Address or not at the Data line SDA.
   * Ack → low :- Rx has received the Address successfully.
   * Ack → high :- Rx has not received the Address.
5- after that the Master will send the 8 bit of Data
6- the Master send the Ack bit to Make sure if the slave Device has received the Data or not.
7- finally the Master will send the Stop Condition to end the Communication between the Master & slave Device or Tx & Rx.

note

the Master & slave May be Tx & Rx.

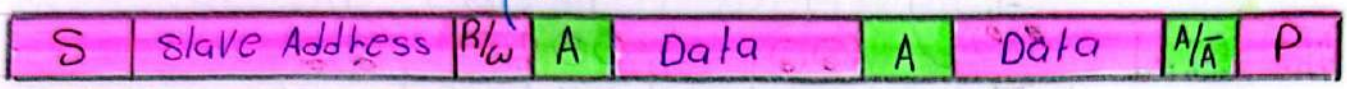
note
the slave Address May be 7 bit or 10 bit.

note
the Master Device May be send repeted start Condition if it wants to send another Data to the same slave or another slave or even receive rather than the stop Condition to avoid losing the I2C bus by the Master.

let's see some examples to see the I2C frame

- From Master to slave
- From slave to Master
- Sr → repeted start Condition
- A → Acknowledge
- $\bar{A}$ → Not Acknowledge
- S → start Condition
- P → stop Condition

* Master is Tx & slave is Rx



* Master is Rx & slave is Tx

note
the target of repeated start Condition to avoid losing the I2C bus by the Master at Multi Master Case.

slave 10 bit Address

* Master is Tx & slave is Rx

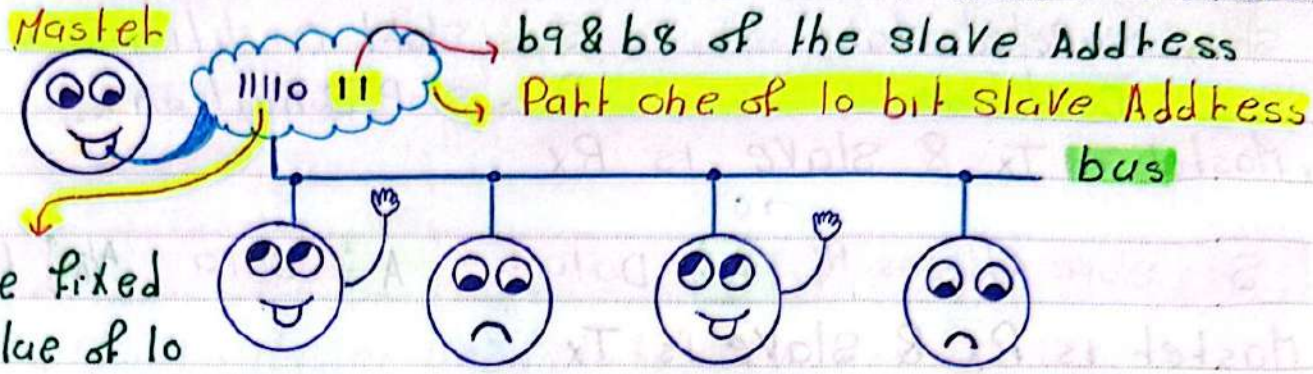
the fixed value of 10 bit slave Address

note A_1 all the slave Devices that have the same b9 & b8 from their Address -es will send it

hote A2 is sent by the target slave because it's determined by the Part 2 of 10 bit slave Address b7 → b0 slave Address. & then A & A/Ā bit.

hote

the slave Device knows that the Master wants to speak with it after the Part 2 of the slave Address before this Part all the slave Devices at the bus that has the same b9 & b8 from the slave Address Value will be ready to receive the Data from the Master.

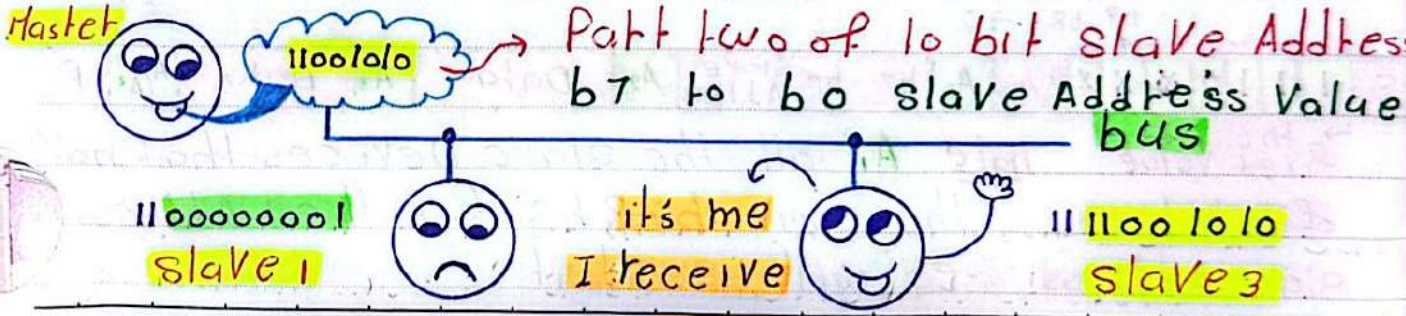


the fixed Value of 10 bit slave Address

slave1 11000000001 slave2 00 11110100 slave3 11110010 10 slave4 01 1001111 01

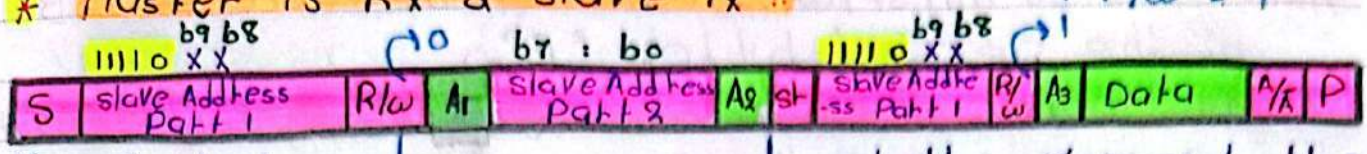
slave Devices Addresses

as you see slave1 & slave3 has the same Address Value for b9 & b8 that are sent by the Master so slave 2 & slave 4 are not important & the Master will send the Part 2 of the slave Address Value b7 ..... b0.



* the Master wants to Read Data from the slave Device.

* Master is Rx & slave Tx ::



the Master will write because it wants to send the slave Address firstly

the fixed Value of 10 bit slave Address

at this Moment the target slave knew that the Master wants to access it

hote

A_1 & A_2 & A_3 are Made by the slave because the Master still sending the Address of target slave firstly.

hote

the Master sends Sr because after it accesses the slave it wants to Read the Data from it.

hote

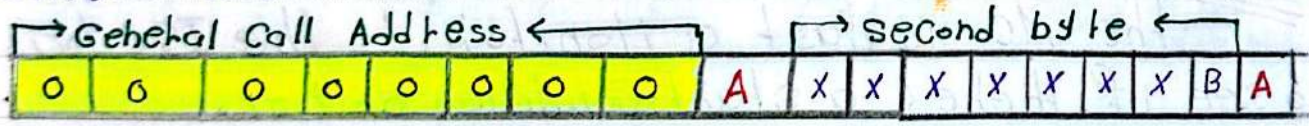
$A/\bar{A}$ is Made by the Master because it is the Rx at this Moment.

I2C Reserved Addresses ::

→ General Call Address

Slave Address = 0000000 & Rw bit = 0

* this Address will go to all the slave Devices at the I2C bus "slave Address + Rw bit" if any slave Device is interested with this byte will generate Ack bit at the Data line. & others that are not interested will send NAck.



hole it's so important to know the value of LSB of the second byte. "B"

When LSB "B = 0" at the second byte after the General Call Address :

if I have slave Device with an Address & I want to Make to change this Address with another Address if this Mode is supported at the slave Device I will send after the General Call Address this Value 06h

06h → Rest & write Programable Part of slave Address by the hardware "Soft reset"

General Call Address "06h"

04h → write Programable Part of slave Address by the hardware "no Soft resent"

↳ just Change the Address of the slave #

General Call Address "04h"

00h → this Code is not allowed to be used as the second byte

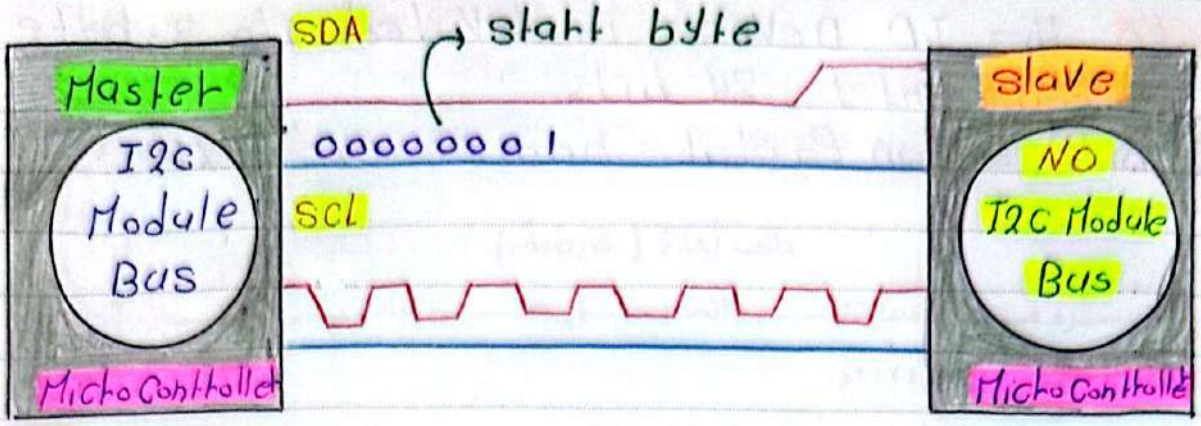
→ **Start byte Address :**

Slave Address = 0000000 & Rw bit = 1

* This Address is for the Micro Controllers that has no I2C bus interface inside it So it use this Address & Some Software to support this Micro Controller with the I2C Interface bus.

hole

the MC is just a Monitor to the I2C bus to see if there's a start Condition or not.

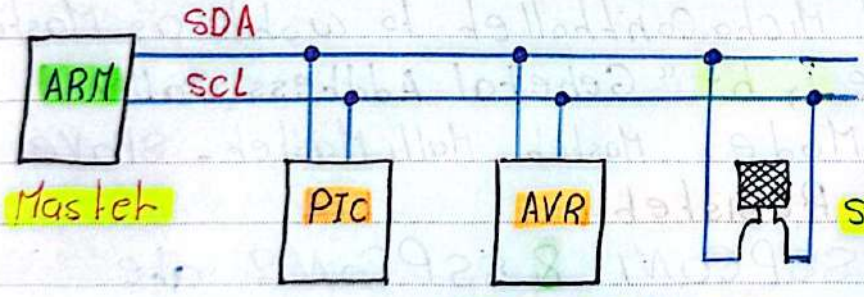


hote the Master Mc suPPortes the I2C bus Module inSide it So it will Make all the sPeficaTion of the I2C bus & May be other fanctions like tatn on & off led Make an InterruPt ..... ect & the slaVe Micro Controller doesn't suPPort the I2C Module inside it So it just will check if the Master sends to it start I2C byte or not it like the Polling InterruPt Processing & this will be just the fanctionof the slaVe Micro Controller to oPerate the slaVe at the I2C Mode.

→ DeVice ID Address

Slave Address = 1111 11xx & Rw bit = x

this Address is used to Connecte the Master DeVice with anothe S.laVe DeVice Sensor or MC at the I2C bus



hote Master Send the ID Address of the Sensor slaVe DeVice

hote the ID Device is divided into 3-byte read only 24 bits

12 bit :- Manfacture name ex:- NXP

Manufacturer bits												Company
11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	NXP Semiconductors
0	0	0	0	0	0	0	0	0	0	0	1	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	0	1	0	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	0	1	1	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	1	0	0	Ramtron International
0	0	0	0	0	0	0	0	0	1	0	1	Analog Devices
0	0	0	0	0	0	0	0	0	1	1	0	STMicroelectronics
0	0	0	0	0	0	0	0	0	1	1	1	ON Semiconductor
0	0	0	0	0	0	0	0	1	0	0	0	Sprintek Corporation
0	0	0	0	0	0	0	0	1	0	0	1	ESPROS Photonics AG
0	0	0	0	0	0	0	0	1	0	1	0	Fujitsu Semiconductor
0	0	0	0	0	0	0	0	1	0	1	1	Flir
0	0	0	0	0	0	0	0	1	1	0	0	O2Micro
0	0	0	0	0	0	0	0	1	1	0	1	Atmel

9 bit:- Part identification is like the serial number ex:- PCA9698

3 bit:- die revision ex:- Rev X

I2C bus at Data sheet Pic18f46k20

→ is existed at the MSSP is consider as Master Synchronous Serial Port has SPI Mode & I2C Mode

I2C Mode

→ I Can Make My Microcontroller to work as Master or slave Mode → by "General Address Call"

→ Multi Master Mode. Master - Multi Master - Slave.

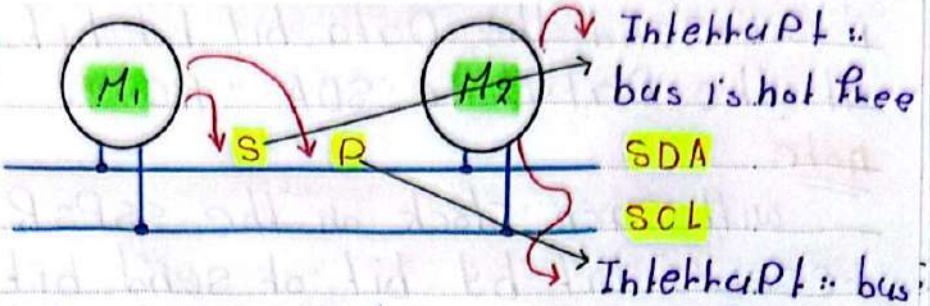
MSSP Control Register:

SSPSTAT & SSPCON1 & SSPCON2 are operated at one Mode SPI or I2C.

I2C Data sheet information

→ our MicroController may be is Master or Slave if the MCU is slave Must send the General Call Address support.

→ at the Multi Master Mode Provides Interrupts on Start & Stop bits in Hw to determine if the bus is free or not.



where
S :- Start bit.
P :- Stop bit.

→ I2C MicroController is operated just at the standard Mode → speed up to 100 kbit/sec

→ Supports 7bit & 10 bit slave Address

→ Two Pins are used for data transfer
* Serial clock SCL RC3 / SCK / SCL
* Serial Data SDA RC4 / SDI / SDA → I2C

note
according to you will send or receive Data you will change the direction of the Pin during the run time.

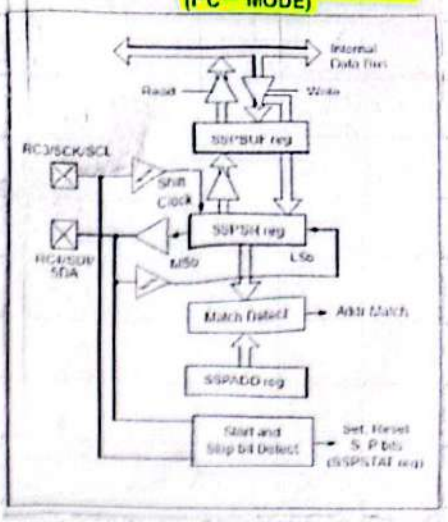
MSSP I2C Registers overall:

→ SSPCON1 → Control
→ SSPCON2 → Control
→ SSPSTAT → Control & status
→ Serial Receiver / Transmit buffer Register SSPBuf
→ MSSP Shift register SSRSR → not directly accessible
→ MSSP Address register SSPADD

note not directly accessible Means has no Address at the MicroController.

the blockdigham of the I2C Module

FIGURE 17-7: MSSP BLOCK DIAGRAM (I²C™ MODE)

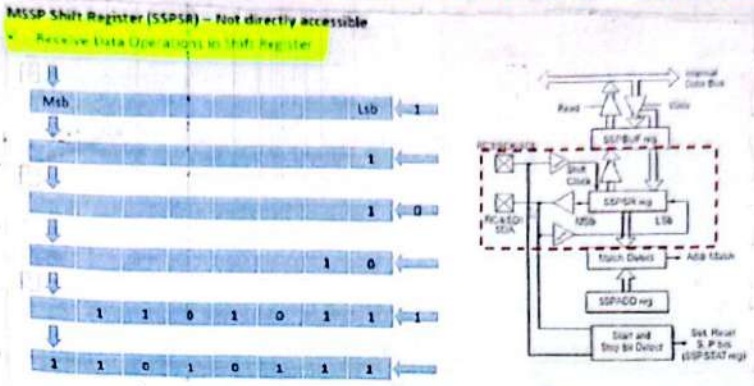


→ SSPSR reg : is the shift register of the Data is used to store the Data bit by bit at the receiving Case & the Data will be written as overall at the transmitting Case & it will send the Data bit by bit at the Dataline SDA "RC4".

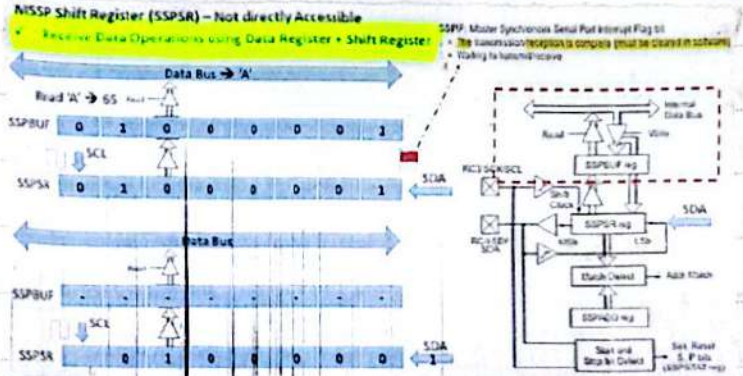
note

with each Clock on the SSPSR reg the Master will receive bit by bit or send bit by bit & the source of the Clock is SCL "RC3".

Receive Data operation at the shift register



→ with each Clk the shift register will receive bit by bit & the first bit will be stored at the LSB & the next bit will be stored at the LSB & the first bit will be shifted to left till reach to the MSB when the first received bit is at the MSB this means the shift register is completed so the value of the shift register at this moment will be stored at the SSPBuf reg.



note

the SSPSR & SSPBUF together create a **double-buffered receiver**

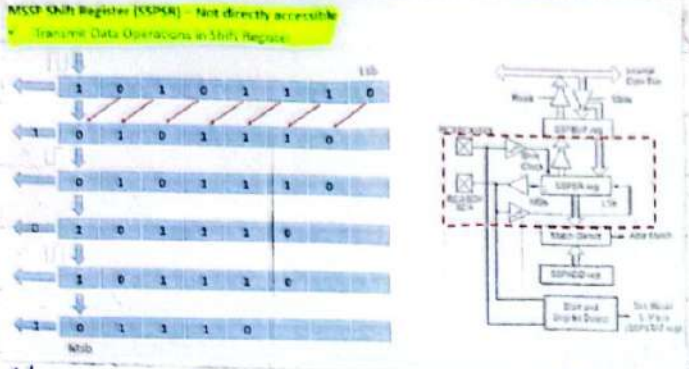
the Double buffeted is I have old Data at the SSPBUF Register & still receive Data at the SSPSR Register I have old Data & I receive new Data.

hote

at the receiving Mode when the shift register is completed there's a flag is set it's called SSPIF :. 1 :. receiving is complete
0 :. waiting to receive.

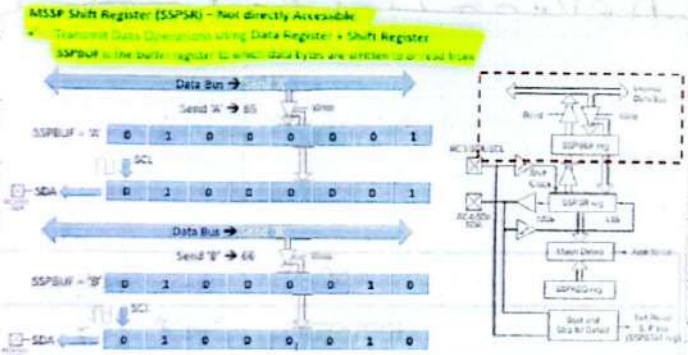
when SSPIF flag has "1" the Data at the shift register will be at the buffer Register from here I can read the Data from SSPBUF.

Transmit Data operation at the shift register



→ the Data that I want to transmit it is ready at the shift register with each Clk ⎍⎍ the shift register will send the Data bit by bit at the Data line SDA from the MSB to the LSB.

hote when the shift register send the bit that is existed at the MSB will make shift left to other bit at the register till send all byte of Data at the Data line.



hote

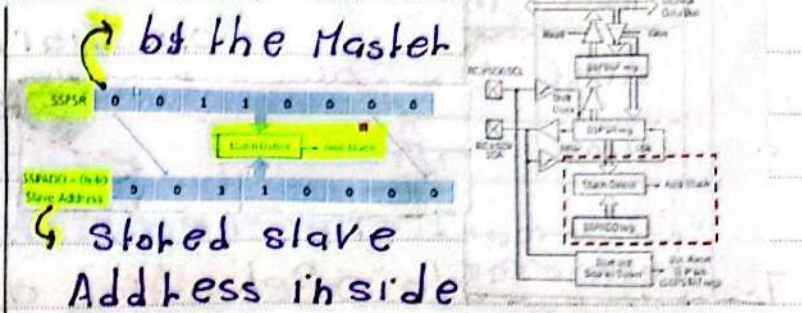
SSPBUF is hot double Buffeted :. this means the Data is written at the sspBuf will beat the sspSR directly.

note the SSPBUF Register takes or gives the Data from or to the Data bus at the Device.

→ MssP Address Register "SSPADD reg" at the slave Mode: this register has the slave Device address.

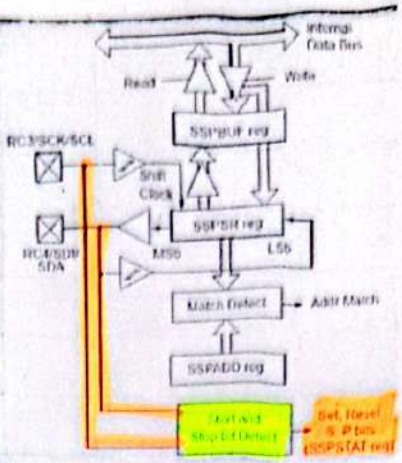
How the slave Device knows that the Master wants to speak with it slave Address

the Master Device when it wants to speak with target slave Device firstly it will send the Address of the slave Device at the Data line SDA this Address will be stored at the shift register of the slave Device & the slave Device has it's Address is stored at the slave Address register the Match Detect block will Compare the Address that is existed at the shift register with the Address that is existed at the slave Address register when the two Addresses are equaled the Addr Match flag will be set at this Moment the slave Device has known that the Master Device wants to speak with it.



note the Match Detect Compare at shift register from bit1 to bit7 because bit0 is for the R/w bit & at slave Address reg from bit0 to bit6 & bit7 is not important.

note the function of the start and stop bit Detect bit block is to Detect if there is stop or start Conditions at the I2C bus as you see at the figure is Connected to the Pins Transfere of Data & Clock.



→ MSSP Address Register "SSPADD reg" at the Master Mode: this register will act as the Baud Rate Generator reload value.

↳ the speed of Data Transfering "Clock"

note the value is SSPADD <6:0> 7bit only.

Master Mode

- SSPADD register is used to decide or generate the clocks to the SCL pin

TABLE 17-3: I²C™ CLOCK RATE W/BRG

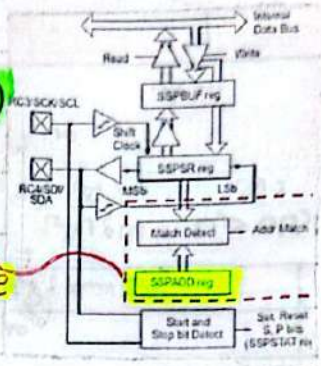
Fcy	Fcy	Fcy*2	BRG Value	FSCL (2 Rollovers of BRG)
10 MHz	10 MHz	20 MHz	18h	400 kHz
10 MHz	10 MHz	20 MHz	19h	392.16 kHz
10 MHz	10 MHz	20 MHz	1Ah	384.62 kHz
10 MHz	10 MHz	20 MHz	1Bh	377.36 kHz
10 MHz	10 MHz	20 MHz	1Ch	370.37 kHz
10 MHz	10 MHz	20 MHz	1Dh	363.64 kHz
10 MHz	10 MHz	20 MHz	1Eh	357.14 kHz
10 MHz	10 MHz	20 MHz	1Fh	350.88 kHz
10 MHz	10 MHz	20 MHz	20h	344.83 kHz
10 MHz	10 MHz	20 MHz	21h	339.00 kHz
10 MHz	10 MHz	20 MHz	22h	333.33 kHz
10 MHz	10 MHz	20 MHz	23h	327.87 kHz
10 MHz	10 MHz	20 MHz	24h	322.58 kHz
10 MHz	10 MHz	20 MHz	25h	317.46 kHz
10 MHz	10 MHz	20 MHz	26h	312.50 kHz
10 MHz	10 MHz	20 MHz	27h	307.69 kHz
10 MHz	10 MHz	20 MHz	28h	303.03 kHz
10 MHz	10 MHz	20 MHz	29h	298.51 kHz
10 MHz	10 MHz	20 MHz	2Ah	294.12 kHz
10 MHz	10 MHz	20 MHz	2Bh	289.86 kHz
10 MHz	10 MHz	20 MHz	2Ch	285.71 kHz
10 MHz	10 MHz	20 MHz	2Dh	281.69 kHz
10 MHz	10 MHz	20 MHz	2Eh	277.78 kHz
10 MHz	10 MHz	20 MHz	2Fh	273.97 kHz
10 MHz	10 MHz	20 MHz	30h	270.27 kHz
10 MHz	10 MHz	20 MHz	31h	266.67 kHz
10 MHz	10 MHz	20 MHz	32h	263.16 kHz
10 MHz	10 MHz	20 MHz	33h	259.74 kHz
10 MHz	10 MHz	20 MHz	34h	256.41 kHz
10 MHz	10 MHz	20 MHz	35h	253.16 kHz
10 MHz	10 MHz	20 MHz	36h	250.00 kHz
10 MHz	10 MHz	20 MHz	37h	246.91 kHz
10 MHz	10 MHz	20 MHz	38h	243.90 kHz
10 MHz	10 MHz	20 MHz	39h	240.96 kHz
10 MHz	10 MHz	20 MHz	3Ah	238.10 kHz
10 MHz	10 MHz	20 MHz	3Bh	235.29 kHz
10 MHz	10 MHz	20 MHz	3Ch	232.56 kHz
10 MHz	10 MHz	20 MHz	3Dh	229.89 kHz
10 MHz	10 MHz	20 MHz	3Eh	227.27 kHz
10 MHz	10 MHz	20 MHz	3Fh	224.72 kHz
10 MHz	10 MHz	20 MHz	40h	222.22 kHz
10 MHz	10 MHz	20 MHz	41h	219.78 kHz
10 MHz	10 MHz	20 MHz	42h	217.39 kHz
10 MHz	10 MHz	20 MHz	43h	215.05 kHz
10 MHz	10 MHz	20 MHz	44h	212.77 kHz
10 MHz	10 MHz	20 MHz	45h	210.53 kHz
10 MHz	10 MHz	20 MHz	46h	208.33 kHz
10 MHz	10 MHz	20 MHz	47h	206.19 kHz
10 MHz	10 MHz	20 MHz	48h	204.08 kHz
10 MHz	10 MHz	20 MHz	49h	202.02 kHz
10 MHz	10 MHz	20 MHz	4Ah	200.00 kHz
10 MHz	10 MHz	20 MHz	4Bh	198.02 kHz
10 MHz	10 MHz	20 MHz	4Ch	196.08 kHz
10 MHz	10 MHz	20 MHz	4Dh	194.17 kHz
10 MHz	10 MHz	20 MHz	4Eh	192.31 kHz
10 MHz	10 MHz	20 MHz	4Fh	190.48 kHz
10 MHz	10 MHz	20 MHz	50h	188.68 kHz
10 MHz	10 MHz	20 MHz	51h	186.92 kHz
10 MHz	10 MHz	20 MHz	52h	185.19 kHz
10 MHz	10 MHz	20 MHz	53h	183.49 kHz
10 MHz	10 MHz	20 MHz	54h	181.82 kHz
10 MHz	10 MHz	20 MHz	55h	180.18 kHz
10 MHz	10 MHz	20 MHz	56h	178.57 kHz
10 MHz	10 MHz	20 MHz	57h	176.99 kHz
10 MHz	10 MHz	20 MHz	58h	175.44 kHz
10 MHz	10 MHz	20 MHz	59h	173.91 kHz
10 MHz	10 MHz	20 MHz	5Ah	172.41 kHz
10 MHz	10 MHz	20 MHz	5Bh	170.94 kHz
10 MHz	10 MHz	20 MHz	5Ch	169.49 kHz
10 MHz	10 MHz	20 MHz	5Dh	168.07 kHz
10 MHz	10 MHz	20 MHz	5Eh	166.67 kHz
10 MHz	10 MHz	20 MHz	5Fh	165.29 kHz
10 MHz	10 MHz	20 MHz	60h	163.93 kHz
10 MHz	10 MHz	20 MHz	61h	162.60 kHz
10 MHz	10 MHz	20 MHz	62h	161.29 kHz
10 MHz	10 MHz	20 MHz	63h	160.00 kHz
10 MHz	10 MHz	20 MHz	64h	158.73 kHz
10 MHz	10 MHz	20 MHz	65h	157.48 kHz
10 MHz	10 MHz	20 MHz	66h	156.25 kHz
10 MHz	10 MHz	20 MHz	67h	155.04 kHz
10 MHz	10 MHz	20 MHz	68h	153.85 kHz
10 MHz	10 MHz	20 MHz	69h	152.67 kHz
10 MHz	10 MHz	20 MHz	6Ah	151.52 kHz
10 MHz	10 MHz	20 MHz	6Bh	150.38 kHz
10 MHz	10 MHz	20 MHz	6Ch	149.25 kHz
10 MHz	10 MHz	20 MHz	6Dh	148.15 kHz
10 MHz	10 MHz	20 MHz	6Eh	147.06 kHz
10 MHz	10 MHz	20 MHz	6Fh	145.99 kHz
10 MHz	10 MHz	20 MHz	70h	144.93 kHz
10 MHz	10 MHz	20 MHz	71h	143.88 kHz
10 MHz	10 MHz	20 MHz	72h	142.86 kHz
10 MHz	10 MHz	20 MHz	73h	141.84 kHz
10 MHz	10 MHz	20 MHz	74h	140.85 kHz
10 MHz	10 MHz	20 MHz	75h	139.86 kHz
10 MHz	10 MHz	20 MHz	76h	138.89 kHz
10 MHz	10 MHz	20 MHz	77h	137.93 kHz
10 MHz	10 MHz	20 MHz	78h	136.99 kHz
10 MHz	10 MHz	20 MHz	79h	136.05 kHz
10 MHz	10 MHz	20 MHz	7Ah	135.14 kHz
10 MHz	10 MHz	20 MHz	7Bh	134.23 kHz
10 MHz	10 MHz	20 MHz	7Ch	133.33 kHz
10 MHz	10 MHz	20 MHz	7Dh	132.45 kHz
10 MHz	10 MHz	20 MHz	7Eh	131.58 kHz
10 MHz	10 MHz	20 MHz	7Fh	130.72 kHz
10 MHz	10 MHz	20 MHz	80h	129.87 kHz
10 MHz	10 MHz	20 MHz	81h	129.03 kHz
10 MHz	10 MHz	20 MHz	82h	128.21 kHz
10 MHz	10 MHz	20 MHz	83h	127.39 kHz
10 MHz	10 MHz	20 MHz	84h	126.58 kHz
10 MHz	10 MHz	20 MHz	85h	125.78 kHz
10 MHz	10 MHz	20 MHz	86h	125.00 kHz
10 MHz	10 MHz	20 MHz	87h	124.22 kHz
10 MHz	10 MHz	20 MHz	88h	123.46 kHz
10 MHz	10 MHz	20 MHz	89h	122.70 kHz
10 MHz	10 MHz	20 MHz	8Ah	121.95 kHz
10 MHz	10 MHz	20 MHz	8Bh	121.21 kHz
10 MHz	10 MHz	20 MHz	8Ch	120.48 kHz
10 MHz	10 MHz	20 MHz	8Dh	119.76 kHz
10 MHz	10 MHz	20 MHz	8Eh	119.05 kHz
10 MHz	10 MHz	20 MHz	8Fh	118.34 kHz
10 MHz	10 MHz	20 MHz	90h	117.65 kHz
10 MHz	10 MHz	20 MHz	91h	116.96 kHz
10 MHz	10 MHz	20 MHz	92h	116.28 kHz
10 MHz	10 MHz	20 MHz	93h	115.61 kHz
10 MHz	10 MHz	20 MHz	94h	114.94 kHz
10 MHz	10 MHz	20 MHz	95h	114.29 kHz
10 MHz	10 MHz	20 MHz	96h	113.64 kHz
10 MHz	10 MHz	20 MHz	97h	113.00 kHz
10 MHz	10 MHz	20 MHz	98h	112.36 kHz
10 MHz	10 MHz	20 MHz	99h	111.73 kHz
10 MHz	10 MHz	20 MHz	9Ah	111.11 kHz
10 MHz	10 MHz	20 MHz	9Bh	110.50 kHz
10 MHz	10 MHz	20 MHz	9Ch	109.89 kHz
10 MHz	10 MHz	20 MHz	9Dh	109.29 kHz
10 MHz	10 MHz	20 MHz	9Eh	108.70 kHz
10 MHz	10 MHz	20 MHz	9Fh	108.11 kHz
10 MHz	10 MHz	20 MHz	A0h	107.53 kHz
10 MHz	10 MHz	20 MHz	A1h	106.95 kHz
10 MHz	10 MHz	20 MHz	A2h	106.38 kHz
10 MHz	10 MHz	20 MHz	A3h	105.82 kHz
10 MHz	10 MHz	20 MHz	A4h	105.26 kHz
10 MHz	10 MHz	20 MHz	A5h	104.71 kHz
10 MHz	10 MHz	20 MHz	A6h	104.17 kHz
10 MHz	10 MHz	20 MHz	A7h	103.63 kHz
10 MHz	10 MHz	20 MHz	A8h	103.09 kHz
10 MHz	10 MHz	20 MHz	A9h	102.56 kHz
10 MHz	10 MHz	20 MHz	AAh	102.04 kHz
10 MHz	10 MHz	20 MHz	ABh	101.52 kHz
10 MHz	10 MHz	20 MHz	ACh	101.01 kHz
10 MHz	10 MHz	20 MHz	ADh	100.50 kHz
10 MHz	10 MHz	20 MHz	AEh	100.00 kHz
10 MHz	10 MHz	20 MHz	AFh	99.50 kHz
10 MHz	10 MHz	20 MHz	B0h	99.01 kHz
10 MHz	10 MHz	20 MHz	B1h	98.52 kHz
10 MHz	10 MHz	20 MHz	B2h	98.04 kHz
10 MHz	10 MHz	20 MHz	B3h	97.56 kHz
10 MHz	10 MHz	20 MHz	B4h	97.09 kHz
10 MHz	10 MHz	20 MHz	B5h	96.62 kHz
10 MHz	10 MHz	20 MHz	B6h	96.15 kHz
10 MHz	10 MHz	20 MHz	B7h	95.69 kHz
10 MHz	10 MHz	20 MHz	B8h	95.24 kHz
10 MHz	10 MHz	20 MHz	B9h	94.79 kHz
10 MHz	10 MHz	20 MHz	BAh	94.34 kHz
10 MHz	10 MHz	20 MHz	BBh	93.90 kHz
10 MHz	10 MHz	20 MHz	BCh	93.46 kHz
10 MHz	10 MHz	20 MHz	BDh	93.02 kHz
10 MHz	10 MHz	20 MHz	BEh	92.59 kHz
10 MHz	10 MHz	20 MHz	BFh	92.17 kHz
10 MHz	10 MHz	20 MHz	C0h	91.74 kHz
10 MHz	10 MHz	20 MHz	C1h	91.32 kHz
10 MHz	10 MHz	20 MHz	C2h	90.91 kHz
10 MHz	10 MHz	20 MHz	C3h	90.50 kHz
10 MHz	10 MHz	20 MHz	C4h	90.09 kHz
10 MHz	10 MHz	20 MHz	C5h	89.69 kHz
10 MHz	10 MHz	20 MHz	C6h	89.29 kHz
10 MHz	10 MHz	20 MHz	C7h	88.89 kHz
10 MHz	10 MHz	20 MHz	C8h	88.50 kHz
10 MHz	10 MHz	20 MHz	C9h	88.11 kHz
10 MHz	10 MHz	20 MHz	CAh	87.72 kHz
10 MHz	10 MHz	20 MHz	CBh	87.34 kHz
10 MHz	10 MHz	20 MHz	CCh	86.96 kHz
10 MHz	10 MHz	20 MHz	CDh	86.58 kHz
10 MHz	10 MHz	20 MHz	CEh	86.21 kHz
10 MHz	10 MHz	20 MHz	CFh	85.84 kHz
10 MHz	10 MHz	20 MHz	D0h	85.47 kHz
10 MHz	10 MHz	20 MHz	D1h	85.11 kHz
10 MHz	10 MHz	20 MHz	D2h	84.75 kHz
10 MHz	10 MHz	20 MHz	D3h	84.39 kHz
10 MHz	10 MHz	20 MHz	D4h	84.03 kHz
10 MHz	10 MHz	20 MHz	D5h	83.68 kHz
10 MHz	10 MHz	20 MHz	D6h	83.33 kHz
10 MHz	10 MHz	20 MHz	D7h	82.99 kHz
10 MHz	10 MHz	20 MHz	D8h	82.64 kHz
10 MHz	10 MHz	20 MHz	D9h	82.30 kHz
10 MHz	10 MHz	20 MHz	DAh	81.97 kHz
10 MHz	10 MHz	20 MHz	DBh	81.63 kHz
10 MHz	10 MHz	20 MHz	DCh	81.30 kHz
10 MHz	10 MHz	20 MHz	DDh	80.97 kHz
10 MHz	10 MHz	20 MHz	DEh	80.65 kHz
10 MHz	10 MHz	20 MHz	DFh	80.32 kHz
10 MHz	10 MHz	20 MHz	E0h	80.00 kHz
10 MHz	10 MHz	20 MHz	E1h	79.68 kHz
10 MHz	10 MHz	20 MHz	E2h	79.37 kHz
10 MHz	10 MHz	20 MHz	E3h	79.05 kHz
10 MHz	10 MHz	20 MHz	E4h	78.74 kHz
10 MHz	10 MHz	20 MHz	E5h	78.43 kHz
10 MHz	10 MHz	20 MHz	E6h	78.13 kHz
10 MHz	10 MHz	20 MHz	E7h	77.82 kHz
10 MHz	10 MHz	20 MHz	E8h	77.52 kHz
10 MHz	10 MHz	20 MHz	E9h	77.22 kHz
10 MHz	10 MHz	20 MHz	EAh	76.92 kHz
10 MHz	10 MHz	20 MHz	EBh	76.63 kHz
10 MHz	10 MHz	20 MHz	ECh	76.34 kHz
10 MHz	10 MHz	20 MHz	EDh	76.04 kHz
10 MHz	10 MHz	20 MHz	EEh	75.76 kHz
10 MHz	10 MHz	20 MHz	EFh	75.47 kHz
10 MHz	10 MHz	20 MHz	F0h	75.19 kHz
10 MHz	10 MHz	20 MHz	F1h	74.91 kHz
10 MHz	10 MHz	20 MHz	F2h	74.63 kHz
10 MHz	10 MHz	20 MHz	F3h	74.35 kHz
10 MHz	10 MHz	20 MHz	F4h	74.07 kHz
10 MHz	10 MHz	20 MHz	F5h	73.80 kHz
10 MHz	10 MHz	20 MHz	F6h	73.53 kHz
10 MHz	10 MHz	20 MHz	F7h	73.26 kHz
10 MHz	10 MHz	20 MHz	F8h	72.99 kHz
10 MHz	10 MHz	20 MHz	F9h	72.73 kHz
10 MHz	10 MHz	20 MHz	FAh	72.46 kHz
10 MHz	10 MHz	20 MHz	FBh	72.20 kHz
10 MHz	10 MHz	20 MHz	FCh	71.94 kHz
10 MHz	10 MHz	20 MHz	FDh	71.68 kHz
10 MHz	10 MHz	20 MHz	FEh	71.43 kHz
10 MHz	10 MHz	20 MHz	FFh	71.17 kHz

Clock = Fosc / 4 * (SSPADD+1)

↳ the equation of the clock

Clock speed Value

↳ this table to slecte the clock speed at the Master Mode I2C Module.



---

note → I2C Firmware Controlled Master Mode

↳ Master Mode functionality by Sw.

→ I2C Master Mode, Clock = Fosc / 4 * (SSPADD+1)

↳ Master Mode functionality by Hw.

note the Pull up resistors Must be at the I2C bus to operate the Module