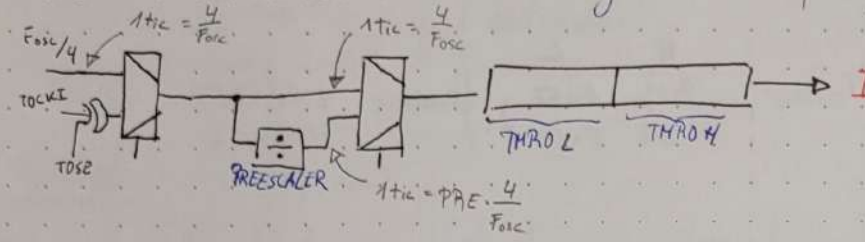
TIMERS

Timer 0

8 bits o 16 bits. Pot ser → Timer → Fa servir Fosc intern (Fosc/4)
→ Counter → Compta periodes externs. (T0CKI)

Hi ha interrupció quan passa de 0xFF..F → 0x00...0

Només té PRE escalos. Pot carregar-se valor previament.



Exemple

Càlcul de Valors

1. Primer hem de saber cada quant volem que hi hagi un tick. # Normalment Fosc/4
2. Ara hem de saber si necessitem PRE o no i com calcular-ho.

2.1. Volem que hi hagi OVF cada 100 ms ⇒ Timer 0 compta desde x fins 0xFF..F ⇒ OVF.
Això sig. que volem saber quants ticks són necessaris per fer 100 ms.

$$100 \times 10^{-3} \div \frac{PRE \cdot 4}{F_{osc}} = \text{ticks necessaris.}$$

2.2. Donat que estarem fent servir 16b, La quantitat de ticks ha de ser $< 2^{16} - 1$.

2.3. Finalment queda anar provant possibles valors de PRE # Està formulari.

▷ PRE = 2 ⇒ $100 \times 10^{-3} \cdot \frac{1}{\frac{2 \cdot 4}{8 \times 10^6}} = 100.000 > 2^{16} - 1$ ✗

▷ **PRE = 4** ⇒ $100 \times 10^{-3} \cdot \frac{1}{\frac{4 \cdot 4}{8 \times 10^6}} = 50.000 < 2^{16} - 1$ ✓

3. Això sig. que si PRE=4, necessitem 50.000 Ticks. ⇒ Com que TMR0 són 16 que és més gran que 50.000 haurem de carregar TMR0 = $(2^{16} - 1) - 50000 =$ **15535**

Configuran Timer 0

1. Configurar TMR0 # No impo. ordre
2. Afegir el valor del TMR0 ▽ IMPO ORDRE
3. Habilitar interrupcions # No impo. ordre

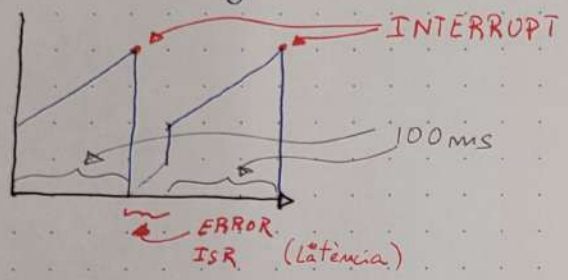
ISR del TMR0

```
void interrupt [low/high] ISR(void){
  if(TMR0IE && TMR0IF){
    TMR0IF=0;
    TMR0 = (2^16-1)-50000;
  }
}
```

OBS = Si fem servir 16b. necessitem 2 cicles per fer lectura del valor.

Consideracions

- Donat que lectura 16 bits = 2 cicles pot ser que canvi el High i llegim malament. PIC18 fa "foto" del High quan llegim Low.
- A l'hora d'escriure primer s'escriu part alta (En el REG "representant") i en el segón cicle en Low.



H L
0x05 0xFF
0x06 0x00
⇒ Podria passar: L = 0xFF, H = 0x06. Cosa falsa.

Timer 1/3/5

Sempre 16 bits. Pot ser Timer / Counter } Igual TMR0. Interrupció quan 0xFFFF ⇒ 0x0000 (2B 2B)

Permeten Complex Gate (Controlar TMR inputs externs) però en CI DESHABILITAT.

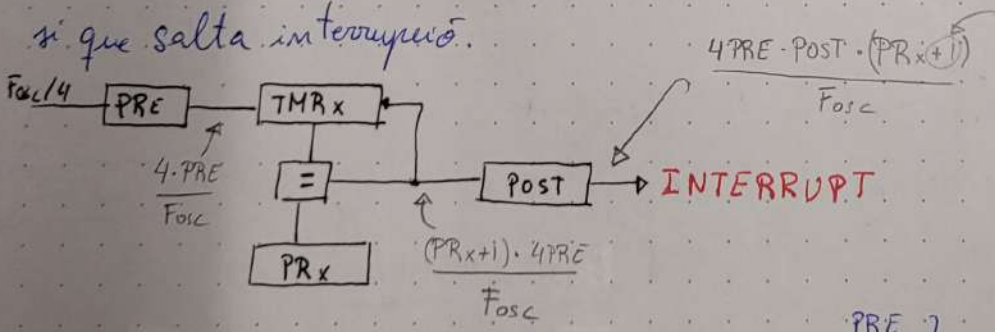
Pot ser usat per Capture / Compare. # Procediment config. igual.

NO OBLIDAR TMRx GE = 0;

Timer 2/4/6

Sempre 8 bits. Sempre Fosc/4. ∇ No té problemes de Latència.

Quan hi ha OVF no hi ha interrupt ⇒ Es compara amb el valor de PRx i quan tenen mateix valor s'envia al POST. Quan hi ho hagut POST vegades sí que salta interrupció.



Per calcular-ho hem d'anar provant valors de PRE, POST, PRx } fins interrupt cada 1 ms.

Ara en la ISR no fa falta tocar TMRx pq es BESET automàtic.

Es fa servir per PWM.

Capture, Compare i PWM (CCP)

Capture TMR1/3/5

Registra valor del TMRx en el moment exacte que succeeix un event extern.
Amb el valor del TMRx_PREV fa una resta i pot saber quant temps hi ha hagut entre events ⇒ Freq senyal entrada.
"TMRx INTERRUPT permeta saber el temps entre CCPx INTERRUPTS":

Exemple

```
void main(void){

  ANSELRC2 = 0; Digital
  TRISRC2 = 1; Input

  CCP1M = 0b0101; Captura
  C1TSEL = 0; TMR1
  CCP1IF = 0; Seg
  CCP1IE = 1;

  T1GE = 0; IMPO
  TMR1CS = 0; Fosc/4
  T1CKPS = 0b00; PRE=1
  TMR1IF = 0; Seg
  TMR1IE = 1;

  PEIE = 1; Perifèric
  GIE = 1; Interrupt

{
```

```
void interrupt high biaISR(void){

  if(TMR1IE && TMR1IF){

    TMR1IF = 0;
    num_ovf++; ← Tractar les vegades que no estem tenint en compte

  {

  if(CCP1IE && CCP1IF){

    valor_captura = CCPR1H<<8 | CCPR1L;

    if(valor_captura_prev > valor_captura){

      num_ticks = (65535 - valor_captura_prev) +
                  valor_captura;

    } else {

      num_ticks = valor_captura - valor_captura_prev;

    {
    num_ticks += (65535) * num_ovf; ← Sumar el que no hem tingut en compte
    num_ovf = CCP1IF = 0;

  {
{
```

RECORDA

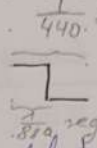
volatile uint16_t valor_captura, valor_captura_prev, num_ticks;
↑ Pq ho tractem en ISR

Compare TMR 1/3/5

Compara el valor del TMRx amb CCPx i quan coincideixen CCPx genera INTERRUPT.
Pot fer-se servir amb una lògica post que canvi l'estat d'un PIN.
Quan hi ha una INTERRUPT (ISR) hem d'augmentar el valor del CCPx pq. següent surgi al mateix temps. No hi ha problema OVF pq TMRx tmb. ← NO el capturem

Exemple

Això sig. que volem un període de $\frac{1}{440}$ seg



Però jo només vull sortida a 1 la meitat del temps $\frac{1}{2} \cdot \frac{1}{440}$

Sabem $F_{osc} = 8 \times 10^6$ Hz i $1\ tic = \frac{4 \cdot PRE}{F_{osc}}$ així que calc nº ticks per fer $\frac{1}{880}$ seg.

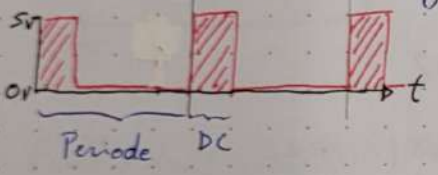
$\frac{1}{880} \div \frac{4\ PRE}{F_{osc}} = \overset{\text{Tanteig}}{[PRE = 9]} \simeq 2273$ ticks.

Per configurar-ho és tot igual amb 2 mod:

- NO fa falta iniciar TMRxIE
- En la ISR, dins CCPxIE, s'ha de fer `CCPx += num_ticks;` (No passa res. pq OVF pq tots dos TMRx, CCPx i faran ovf.)
- Iniciar TMRx a 0 per ser rigurós (Sino simplement fallem la primera).

PWM TMR2/4/6

Permet generar un senyal digital a una freq. constant però variant el període.



- Període: Relacionat amb Freq o temps. PRx
- DC: Percentatge temps que està "ON". CCPRx $\frac{t_{ON}}{T} \cdot 100$

Exemple Ens diuen: $F_{osc} = 8$ MHz, $F_{PWM} = 200$ kHz, DC% = modular

- Càlculs: 1. Primer hem de saber el Període del PWM (Formula al Formulari)

$$\frac{1}{200 \times 10^3} = (PR2 + 1) * 4 * \frac{1}{8 \times 10^6} * TMR2\_PRE \overset{1\ tanteig}{\Rightarrow} PR2 = 9$$

⚠ RECORDEM: PRx de 8b i PRE sempre més petit.

Si 100% 2. Ara que sabem el període, necessitem saber quanta estona ON.

Pulse Width = DCV * Tosc * TMR2_PRE = $1/F_{PWM}$ # Està en form

$DCV = \frac{1}{200 \times 10^3} \cdot \frac{8 \times 10^3}{1 \text{ PRE}} = 40$ Però aquest és el de 100% i el nostre modular.

3. Calculem el DCV amb regla de 3

40 — 100%
DCV — modular %

DCV = (40 * modular) / 100 on modular és un valor de %.

⚠ IMPO: CCPxM = 1100, no 11xx.

OBS: Donat que no podrem dividir entre 2, afegim 2b (Multi per 4) i per això tenim CCPRxL : CCPxCON<5:4>.

```
CCPRxL = (DCV >> 2) & 0xFF;   } IMPO
CCPxCON bits.. DCxB = DCV & 0x03;
```

COMMUNICATIONS

GND Connectat

Pq. tots els disp. de la comunicació tinguin mateixa ref del qual sig. Vss hem de connectar tots els GND. ▽▽ IMPO

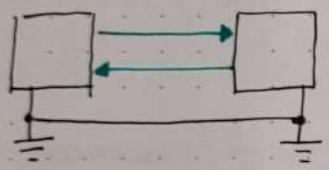
Classificació

Asíncron / Síncron

Asíncron

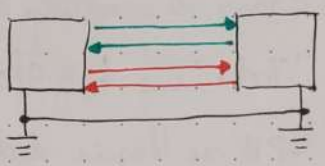
Cada disp. té el seu propi clock.

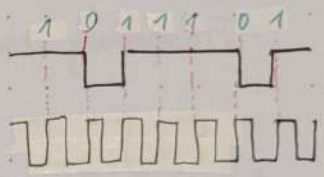
Hem d'estar a mateixa freq per no malinterpretar.



Síncron

Per cada cable de dades hi ha d'haver un cable de clock.

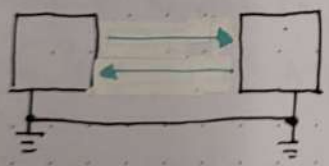




Full / Half Duplex

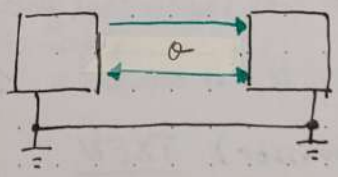
Full Duplex

Dades es poden enviar dos direccions en el mateix moment.



Half Duplex

Dades només es poden en 1 direcció o una direcció cada moment.



Tarden el doble.

Port Sèrie

1 cable per direcció

Només 2 disp

Qualsevol pot iniciar comunicació.

És Serial, Asíncron, Full-Duplex, Point-2-Point, Multimaster.

Idea principal:

- Per defecte estàs Idle ⇒ Emetent 1.
- Algo indica que ja has acabat o comences ⇒ Normalment 0
- Bits s'envien 1 a 1 en un temps determinat. (Prefixat)

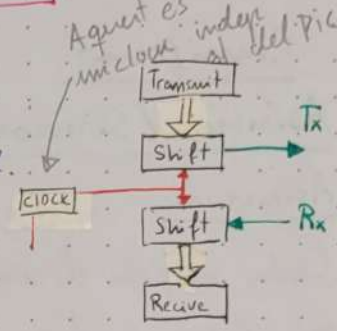
- START Bit: Invers del valor de Idle
- PARITY Bit: (Opcional) Indica quantitat de 1s que hi ha → Parell / Senar
- STOP Bit: Permet agafar dades i guardar-ho.

Total = 1 (START) + n (Data) + 3 0, 1 & (PARITY) + 3 1, 1½, 2 & (STOP)

USART: Dispositiu que permet comunicar-se → Síncron. / Asíncron.

Permet enviar i rebre dades on fem ús del protocol RS-232.

Shift-Register: Passen dades a frame, per enviar-se.



Errors Comuns

- Framming error: Error de sincronització de clocks.
  "Trobo un bit STOP on no deuria".
- Reciver overrun: CPU no llegeix dades que li envien o li envien massa.
  S'omple RCREGx (FIFO) i port sèrie deixa de funcionar.
- Parity Error: No es correcte el bit de paritat amb les dades rebudes.
  #Pot ser degut a soroll electromagnètic

Implementacions

- Bit-Banging: Tot per software (Delay, Shift, ...) Van fent data & 01 i enviant i després data >> 1
- Hardware: Microchip te HW dedicat. 1 bit STOP, 8 bits DATA

REG Tx (Transmissor) TXEN

- TXREGx: Registre on es guarden dades que s'han d'enviar. // Pre càrrega pel TSR
- TXxIF: Indica si TXREGx està disp per ficar dades (1) o no (0) // Sobreescriure
- TSR: Shift Register. Envia dades pel cable.
- TRMT: Indica si s'estan enviant dades (0) o ja ho acabat (1)
- Baud Rate Generator: Indica el clock que s'ha de fer servir
  - BRG16: Indican si necesitem 16 bits per indicar el n (1) o no (0).
  - SPBRG: Indicar el valor de velocitat de Baud. (El n en la fórmula)
  - BRGH: Volem que sigui High (1) o Low (0) speed.
  - SYNC: Seleccionar Asyncron (0) o Syncron (1).

REG Rx (Receptor) RCEN

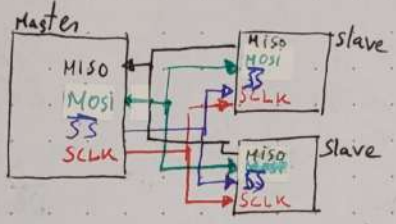
- FERR: Detectar Framering Error
- OERR: Overrun Error. ▽ Tot i que hi hagi OERR si que entram ISR → El podem gestionar aquí dins
- RCREGx: FIFO de 2 pos on entàn dades rebudes. Si arriba tercer ⇒ Overrun.
- RCxIF: Indica que hi ha dada a llegir (1) o no (0).

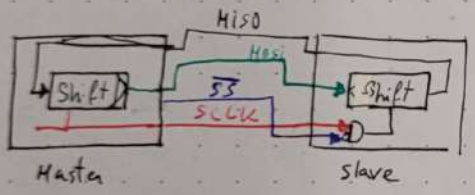
SPI (Serial Peripherial Interface)

És Serial, Síncron (Clock del Master), Full-Duplex, Multipoint, Master/Slave.

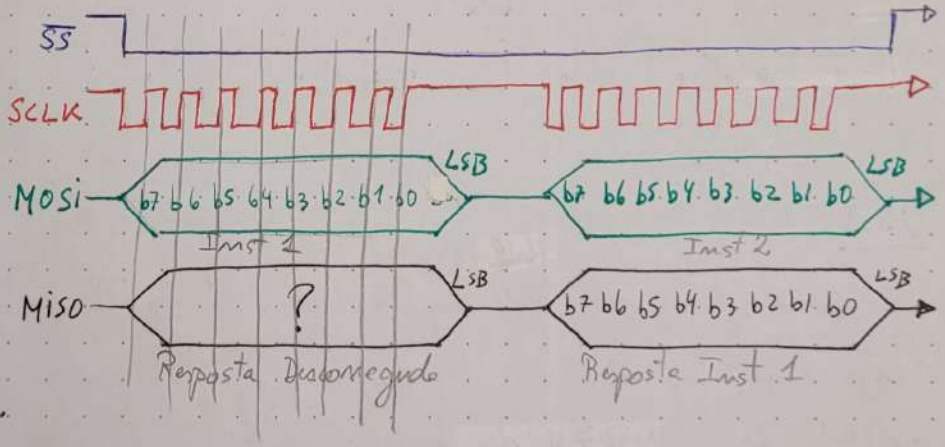
Connexions:

- MISO: Master Input.
- MOSI: Master Output.
- SCLK: Source Clock
- $\overline{SS}$: Slave Select





Hem de llegir el datasheet del slave per veure quina inst. envien pel MOSI i el format de la resposta rebuda pel MISO.



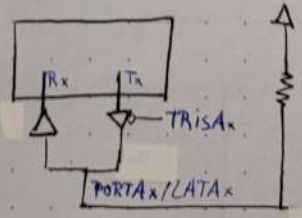
I2C (Inter Integrated Circuit)

És Serie, Síncron (Clock del Master), Half-Duplex, Multipoint, Master/Slave.

Canal SDA (Dades) i SCL (Clock) que estàn connectats en un Pull-Up (Sempre 1)

Procediment:

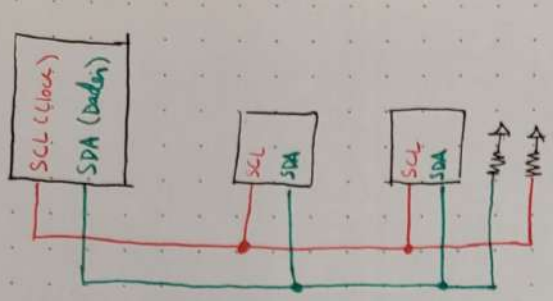
1. S'envia START BIT
2. S'envia @7b (Marcat fabricant)
3. Tipus d'operació R/W (1bit)
4. Resposta ACK del Rx. ACK=0 ⇒ OK
5. Dades s'envien/reben (8b)
6. Resposta ACK del Tx.
7. S'envia STOP BIT.

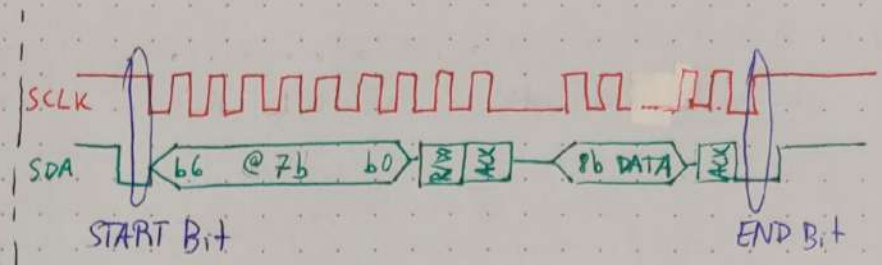


Obs: Quan M ha enviat dades ha de ficar-se a escoltar el ACK del Slave.

Aquí (Slave) Fica a 0 el seu TRISx. i escriu en LATAx un 0 per fer que SDA caigui a 0V. (Resistència fa caigudes)

Master (que estava llegint amb TRISx=1) "reb" del SDA un 0 i sig. que OK.





1-Wire

Serie, Asíncron, Half-Duplex, Multipoint, M/S. Només hi ha un cable i GND.

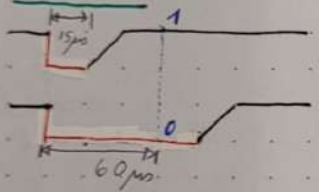
Idea Principal:

1. Host ha de descobrir si hi ha algú.
2. Device avisa de l'existència.
3. Host ho detecta
4. A partir d'ara (Host) pot enviar / rebre dades.

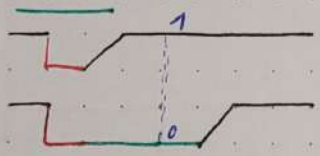
- Cada disp té un ID únic.
- Dins dels disp hi ha un Capacitador que permet "fer foto" del medi.
- Sempre Pull-Up.
- És lent i s'ha de fer Bit-Banging.

Per exemple. Accés a algo.

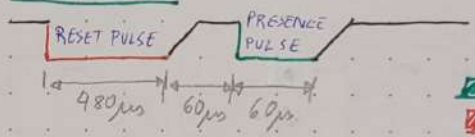
Enviar



Rebre



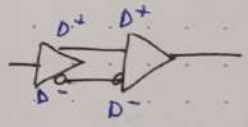
Descobrir



USB

Half-Duplex (1-2)

És Serial, Asíncron, Full-Duplex (3), M/S, Té bus diferencial

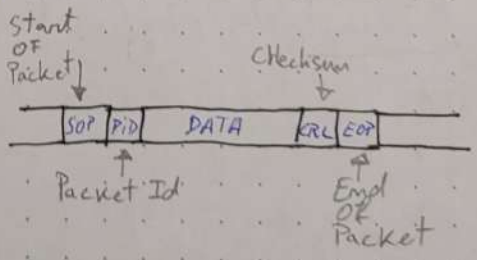


- Signaling: Defineix com representen físicament dades. Interpretacions / Velocitat, ...
- Codification: → NRZI: El 0 fa switch level
  → Stuffed bit: Bit extra (0) que s'afegeix després del 6è 1.

— Start: Per defecte $D^+ = 1 \Rightarrow D^+ = 0$ per iniciar.

— Stop: Ficar $D^+ = D^- = 0$.

- Packing: Dades s'escriuen seguint estàndard



PID	Name	Resum
Token	OUT IN SOF SETUP	Iniciar transmissió de dades
Data	DATA 0 DATA 1	Enviar dades s'ha d'anar intercanviant
Handshake	ACK NAK	Ha rebut OK No està disp (Rx Tx)