|  |
| --- |
| ABEBASHOP.COM |
| LIBRERIA LENGUAJE C PARA PIC |
|  |
| **Author: Federico** |

|  |
| --- |
|  |

Contents

[TIMER (CHAR/INT/SHORTLONG/LONG) 3](#_Toc458628379)

[NOTA 3](#_Toc458628380)

[SETEOS QUE EL USUARIO DEBE HACER 3](#_Toc458628381)

[ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA 3](#_Toc458628382)

[USO 3](#_Toc458628383)

[ALTERAR TIEMPOS DINAMICAMENTE 4](#_Toc458628384)

[PULSADOR 5](#_Toc458628385)

[NOTAS 5](#_Toc458628386)

[SETEOS QUE EL USUARIO DEBE HACER 5](#_Toc458628387)

[ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA 5](#_Toc458628388)

[USO 5](#_Toc458628389)

[PULSADORPIN 6](#_Toc458628390)

[NOTAS 6](#_Toc458628391)

[SETEOS QUE EL USUARIO DEBE HACER 6](#_Toc458628392)

[ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA 6](#_Toc458628393)

[USO 6](#_Toc458628394)

[XPORT 8](#_Toc458628395)

[FSM 9](#_Toc458628396)

[SETEOS QUE EL USUARIO DEBE HACER 9](#_Toc458628397)

[ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA 9](#_Toc458628398)

[USO 9](#_Toc458628399)

[INICIO DE LA FSM 9](#_Toc458628400)

[LCD 10](#_Toc458628401)

[LCDX 11](#_Toc458628402)

[MX\_NO\_MEM\_4X4 12](#_Toc458628403)

[SETEOS QUE EL USUARIO DEBE HACER 12](#_Toc458628404)

[ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA 12](#_Toc458628405)

[USO 12](#_Toc458628406)

# TIMER (CHAR/INT/SHORTLONG/LONG)

## NOTA

Hay 3 librerías "timer": timerChar, timerInt y timerLong. Funcionan exactamente igual pero difieren en el tamaño de las variables de memoria que utilizan, permitiendo la "char" contar hasta 2^8 periodos, la "int" contar hasta 2^16 periodos y la "long" contar hasta 2^32 periodos. En la explicación siguiente, se utilizara la librería "char" para ejemplificar, y luego se puede reemplazar "char" por "int" o por "long" donde corresponda para utilizar la otra librería deseada. En las macros, no se diferencian macros para una u otra librería, ya que la macro es la misma para las 3 librerías.

## SETEOS QUE EL USUARIO DEBE HACER

Se requieren 2 encabezados:

* timerChar\_aliases.h: establece un nombre para cada timer que se utilizará. La cantidad de nombres que se escriban en la lista, ya determina automáticamente la cantidad de temporizadores.
* timerChar\_delays.h: establecer un tiempo para cada timer. Los tiempos deben ser mayores a 0. Un tiempo igual a 0, hará que el timer nunca se accione.

## ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA

* timerChar\_lib.h

## USO

* timerCharDriver(TMR\_SAMPLE). La ejecución periódica de esta función, establece la base de tiempo para el timer TMR\_SAMPLE.
* TMR\_CHAR\_START(TMR\_SAMPLE). Arranca temporizador TMR\_SAMPLE. Cuando se cumpla el tiempo, avisará y se detendrá. El timer no continuará contando, a menos que nuevamente se arranque la temporización. Arranca seguro, y siempre, no importa si estaba pausado.
* TMR\_CHAR\_INIT\_LOOP(TMR\_SAMPLE). Hace lo mismo que TMR\_START(TMR\_SAMPLE), con la diferencia de que cuando se cumpla el tiempo, además de avisar, el timer reiniciará. Por lo tanto, no será necesario reiniciarlo manualmente. Es decir que se arranca el timer para que funcione de manera periódica. Arranca seguro, y siempre, no importa si estaba pausado.
* TMR\_CHAR\_TIMEOUT(TMR\_SAMPLE). Chequea el estado del flag de timeout perteneciente a TMR\_CHAR\_SAMPLE (y lo resetea en caso de cumplido el timeout, para que en otra consulta ya no retorne afirmativo). Retorna 1 si se cumplió la temporización del timer, ya sea periódico o no. Retorna cero de otro modo. Consultando esta macro de manera periódica, es la única manera de chequear el estado de los temporizadores.
* TMR\_CHAR\_RUNNING(TMR\_SAMPLE). Indica si esta sin pausa, y el contador aun no produjo timeout.
* TMR\_CHAR\_STOP(TMR\_SAMPLE). Fuerza que el contador detenga el conteo, sin pausarlo, y sin generar timeout.
* TMR\_INT\_PAUSED(TMR\_SAMPLE)). Indica si esta pausado.
* TMR\_INT\_PAUSE(TMR\_SAMPLE)). Genera que el contador pause.
* TMR\_INT\_UNPAUSE(TMR\_SAMPLE). Genera que el contador salga de la pausa.
* TMR\_INT\_STOPPED(TMR\_SAMPLE). Indica si se encuentra detenido ya sea por haber terminado de contar o porque nunca se lo inicio para contar, si se encuentra pausado, considera que aun se encuentra funcionando. Si se consulta justo en el instante cuando se produjo timeout, va a devolver que aun se encuentra funcionando, ya que el timeout aun no fue limpiado.

## ALTERAR TIEMPOS DINAMICAMENTE

La variable "unsigned char timerCharDelayConstant[TMR\_SAMPLE]" almacena la cantidad de bases de tiempo que dura TMR\_SAMPLE.

# PULSADOR

## NOTAS

La librería considera que todos los pines tienen pull-up, y por lo tanto el pulsar un pin, lo que hace es conectarlo a 0V.

Si se necesita detectar flanco ascendente y tambien descendente, se pueden definir 2 pulsadores mapeados al mismo pin de entrada, uno que trabaje al presionar, y el otro que trabaje al liberar.

## SETEOS QUE EL USUARIO DEBE HACER

Se requieren 2 encabezados:

* pulsador\_aliases.h: establece un nombre para cada pulsador que se utilizará. La cantidad de nombres que se escriban en la lista, ya determina automáticamente la cantidad de pulsadores.
* pulsador\_pindef.h: define pines y puerto y seteos para cada pulsador.

## ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA

* pulsador\_lib.h

## USO

* pulsPinDriver(void). Se debe ejecutar periodicamente cada 10ms.
* PULS\_EVENT(). Retorna 1 si ocurrió un evento en cualquiera de los pulsadores. Luego, resetea el flag de aviso, por lo que si vuelve a consultarse, retornará 0.
* PULS\_GET\_ID(). Retorna el número identificatorio del pulsador que fue accionado.
* PULS\_EVENT\_ID(i). Consulta si el pulsador "i" tuvo un evento. Si ese pulsador no tuvo evento, retorna cero y no hace nada. En caso de que el pulsador "i" haya tenido evento, retorna 1 y resetea el flag de aviso.

# PULSADORPIN

para forzar el "clear" del boton pulsado, lo que hay que hacer es simplemente revisar PULSPIN\_EVENT\_ID\_FN(i)

por mas que no se vaya a utilizar el boton que fue pulsado, simplemente ejecutar y no hacer nada.

## NOTAS

Maneja 1 o varios pulsadores. Cada pulsador tiene 1 pin dedicado del microcontrolador, o sea, no hay multiplexacion.

Esta libreria no trabaja con buffer, sino que únicamente se memoriza el ultimo pin tocado.

Puede detectar flanco ascendente o descendente o ambos para un mismo pin.

Si los eventos no se leen y toma accion antes de que llegue otro evento, se pierde el ultimo evento y genera uno nuevo.

## SETEOS QUE EL USUARIO DEBE HACER

Se requieren 2 encabezados:

* pulsadorPin\_aliases.h: establece un nombre para cada pulsador que se utilizará. La cantidad de nombres que se escriban en la lista, ya determina automáticamente la cantidad de pulsadores.

// pulsadorPin\_aliases.h

#ifndef \_\_PULSADOR\_PIN\_ALIASES\_H

#define \_\_PULSADOR\_PIN\_ALIASES\_H

// ALIAS

enum e\_pulsadorPinAlias {PULS\_INPUT\_FP,PULS\_INPUT\_FN,PULS\_PIN\_CANT};

#endif//\_\_PULSADOR\_PIN\_ALIASES\_H

* pulsadorPin\_pindef.h: define pines y puerto y seteos para cada pulsador.
* 22-08-2016: siempre supone que pines usan pullup y al apretar lleva entrada a 0V, por eso, pindef solo debe copiarse tal cual se sugiere y el usuario solo completar los mnemotecnicos especificados.

// pulsadorPin\_pindef.h

#ifndef \_\_PULSADOR\_PIN\_PINDEF\_H

#define \_\_PULSADOR\_PIN\_PINDEF\_H

t\_pulsPin pulsPin[PULS\_PIN\_CANT]={{(char\*)&GPIO,3,1,1,1,0},

{(char\*)&GPIO,5,1,1,0,0}

};

#endif//\_\_PULSADOR\_PIN\_PINDEF\_H

## ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA

* pulsadorPin\_lib.h

## USO

* pulsPinDriver(void). Se debe ejecutar periodicamente cada 10ms.
* PULSPIN\_EVENT(). Retorna 1 si ocurrió un evento en cualquiera de los pulsadores. Luego, resetea el flag de aviso, por lo que si vuelve a consultarse, retornará 0.
* PULSPIN\_GET\_ID(). Retorna el número identificatorio del pulsador que fue accionado.
* PULSPIN\_EVENT\_ID(i). Consulta si el pulsador "i" tuvo un evento. Si ese pulsador no tuvo evento, retorna cero y no hace nada. En caso de que el pulsador "i" haya tenido evento, retorna 1 y resetea el flag de aviso.
* PULSPIN\_EVENT\_ID\_FP(i): consulta y resetea flag
* PULSPIN\_EVENT\_ID\_FP(i): consulta y resetea flag
* pulsPinInit(void). Sirve para inicializar correctamente el estado de los pulsadores si se usan pulsadores o llaves con retencion.

# XPORT

Sirve para expandir las salidas, utilizando shift registers, y utilizar esas salidas de manera transparente a la implementacion del software que controla los shift registers, tal como si fueran puertos propios del microcontrolador.

# FSM

e\_fsmEvent{FSM\_NO\_EVENT

siempre debe estar porque se debe usar como valor por defecto en el retorno de funcion char fsmGetEvent(void)

## SETEOS QUE EL USUARIO DEBE HACER

Se requieren 2 encabezados:

* fsm\_forward.h: xxxxxxxxxxxxxxxxx.
* fsm\_events.h: xxxx.
* fsm\_actions.h: xxxx.
* fsm\_transitions.h: xxxx.

## ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA

* fsm\_lib.h

## USO

* timerDriver(TMR\_SAMPLE). La ejecución periódica de esta función, establece la base de tiempo para el timer TMR\_SAMPLE.
* TMR\_START(TMR\_SAMPLE). Arranca temporizador TMR\_SAMPLE. Cuando se cumpla el tiempo, avisará y se detendrá. El timer no continuará contando, a menos que nuevamente se arranque la temporización.
* TMR\_INIT\_LOOP(TMR\_SAMPLE). Hace lo mismo que TMR\_START(TMR\_SAMPLE), con la diferencia de que cuando se cumpla el tiempo, además de avisar, el timer reiniciará. Por lo tanto, no será necesario reiniciarlo manualmente. Es decir que se arranca el timer para que funcione de manera periódica.
* TMR\_TIMEOUT(TMR\_SAMPLE). Chequea el estado del flag de timeout perteneciente a TMR\_SAMPLE (y lo resetea en caso de cumplido el timeout, para que en otra consulta ya no retorne afirmativo). Retorna 1 si se cumplió la temporización del timer, ya sea periódico o no. Retorna cero de otro modo. Consultando esta macro de manera periódica, es la única manera de chequear el estado de los temporizadores.

## INICIO DE LA FSM

* La maquina de estados siempre debe tener un estado llamado "s\_idle", que sera siempre el estado inicial.

# LCD

Usa pines del microcontrolador para el manejo del LCD.

Se deben definir:

#define LCD\_DAT4 PORTXbits.RXX

#define LCD\_DAT5 PORTXbits.RXX

#define LCD\_DAT6 PORTXbits.RXX

#define LCD\_DAT7 PORTXbits.RXX

#define LCD\_RS PORTXbits.RXX

#define LCD\_RW PORTXbits.RXX

#define LCD\_EN PORTXbits.RXX

# LCDX

Usa librería PORTX como pines para controlar el LCD.

# MX\_NO\_MEM\_4X4

Driver para control basico de teclado matricial. No tiene buffer, no contempla multiples teclas presionadas simultaneamente.

## SETEOS QUE EL USUARIO DEBE HACER

Establecer los nombres de las entradas/salidas y data direction registers segun los nombres requeridos por la libreria.

Establecer los pull-up de las teclas, ya sea por software o por hardware.

## ARCHIVO DE ENCABEZAMIENTO DE LA LIBRERIA

mx\_no\_mem\_4x4.h

## USO

* char kbhit(void)

### Ejemplo

if(kbhit()) getchar();

* char getchar()

Toma un caracter de la entrada de teclado. Si no hay nada en espera, fuerza tomar un caracter.