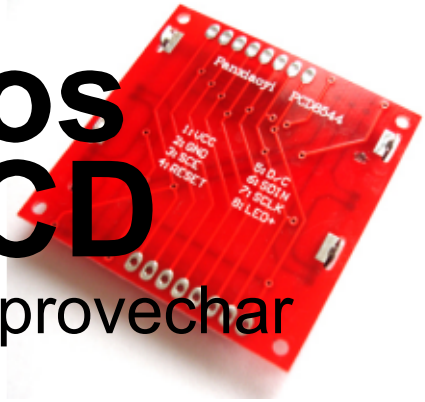
// DESCARGA GRATUITA //

"int año = 2 , número 6" /* SEPTIEMBRE 2009 */

µCONTROL

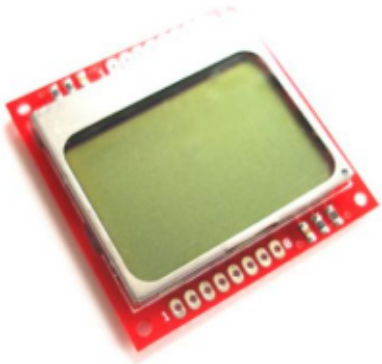
Electrónica en General Pics en Particular

Librería de gráficos para displays GLCD

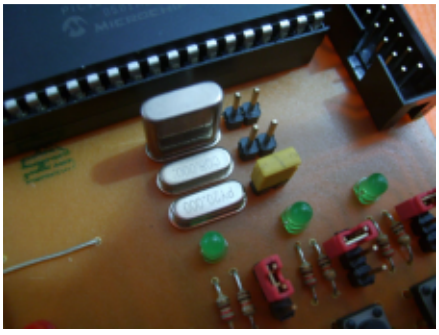


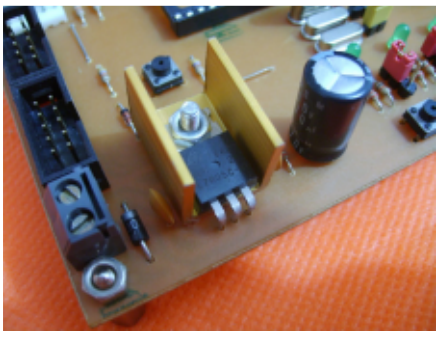
desarrollo de una librería en C para aprovechar todo el potencial de estos displays

¡Un PIC en tu TV!



una biblioteca PAL para PIC18 más un puñado de componentes alcanzan para generar señales de TV en blanco y negro





- » Teclados matriciales en PIC Simulator IDE
- » Programando PICs en assembler
- » Módulo "PIC Trainer 28"
- » Brújula electrónica

Presentación de la "7ma Competencia Nacional de Robótica"

7º Competencia Nacional de Robotica

Organiza:

GRS

Grupo de Robotica y Simulacion

Dto. de Ing. Electrica

Universidad Tecnologica Nacional

Facultad Regional Bahia Blanca

Auspicia:

freescale semiconductor

ED

www.edudevices.com.ar

Categorias:

MINI SUMO

SUMO LIBRE

VELOCISTAS

SUMO POLIMODAL "LIGA DE CAMPEONES"

SUMO POLIMODAL "LIGA INICIAL"

MEK

NOVIEMBRE 2009

WWW.GRSBAHIABLANCA.com.ar

.staff

número = 6; año = 2;

Dirección, Redacción y Corrección:
Ariel Palazzesi
Argentina
arielpalazzesi@gmail.com

Diseño y Diagramación:
Lucas Martín Treser
Argentina
lmtreser@gmail.com

Consejo Editorial:
Mario Sacco
Argentina
service.servisystem@gmail.com

Alejandro Casanova
Argentina
inf.pic.suky@live.com.ar

Pablo
España
quickbasic@terra.es

Bruno Gavando
Francia
bruno.gavand@ad-valorem.fr

.indice

.editorial

Dos meses han pasado desde la salida del número cinco de la **Revista uControl**, y tal como prometimos, tienes en tu pantalla una nueva edición. Esta vez hemos sumado colaboradores y temas, como para que cada vez más lectores encuentran alguna articulo que les resulte de utilidad.

Sin dudas, el "acontecimiento" más importante que tuvo lugar en los meses que pasaron desde la ultima vez que escribí una editorial para la revista fue la caída del servidor que aloja a uControl y el foro. Son cosas que pasan: el hardware puede fallar, ¡y vaya si lo hace!. Durante una semana estuvimos fuera de linea. **Maximiliano Simonazzi**, de Electrónica y Seguridad Digital, nuestro webmaster de cabecera, hizo malabares para convertir una serie de desprolijos archivos de respaldo nuevamente en una página web funcional. Estar fuera de linea una semana implica ser dado de baja de los índices de Google, entre otras cosas. Pasamos de las 2600 visitas diarias que teníamos antes del crash, a unas 100 en los dos o tres días posteriores a la restauración del backup. Sin embargo, en tres semanas volvimos a tener el trafico usual. Quiero aprovechar este espacio para agradecer la paciencia y fidelidad de nuestros visitantes.

Seguimos sumando proyectos y cursos. Como siempre, hay variedad de contenidos como para que todos encuentren algo de utilidad. El que recién comienza a experimentar con la electrónica y programación de microcontroladores en lenguaje C seguramente se sentirá mas cómodos con los displays GLCD (Graphic LCD) luego de leer el articulo que le dedicamos en este número. Para quienes gustan de programar en BASIC, hemos preparado un articulo sobre como gestionar teclados matriciales desde el BASIC del PIC Simulator IDE.

Seguimos con el excelente curso de programación de PICs en assembler, especialmente enfocado a uno de los modelos mas exitosos de Microchip: el 16F628A. Si estás siguiendo la serie de artículos dedicados a la construcción del entrenador para PICs, te gustará saber que ya tenemos listo el módulo para PICs de 28 pines. ¡Es el complemento ideal para el curso de assembler!

Por ultimo, si aún no conoces nuestro foro aprovechamos para recomendarte que pases por él. Esta publicación solo recoge algunos de los temas que allá se tratan, y el formato PDF no permite ni la colaboración ni el intercambio de opiniones que tiene lugar en el foro. Seguramente podrás aprender mucho más visitándolo periódicamente y, sobre todo, ayudarnos a mejorar la revista. ¡Hasta el próximo número!

Foro uControl : http://www.ucontrol.com.ar/forosmf/index.php
Electrónica y Seguridad Digital: http://www.maxisimonazzi.com.ar

uControl recuperandose paulatinamente.

módulo PIC TRAINER 28

Este es tercer módulo de nuestro entrenador. Su función es la de servir como soporte a los microcontroladores PIC de 28 pines, como el 16F874A. Al igual que sus hermanos, emplea un PCB de una sola cara, por lo que su montaje debería ser posible aún para aquellos estudiantes o aficionados que están comenzando transitar este apasionante camino.

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

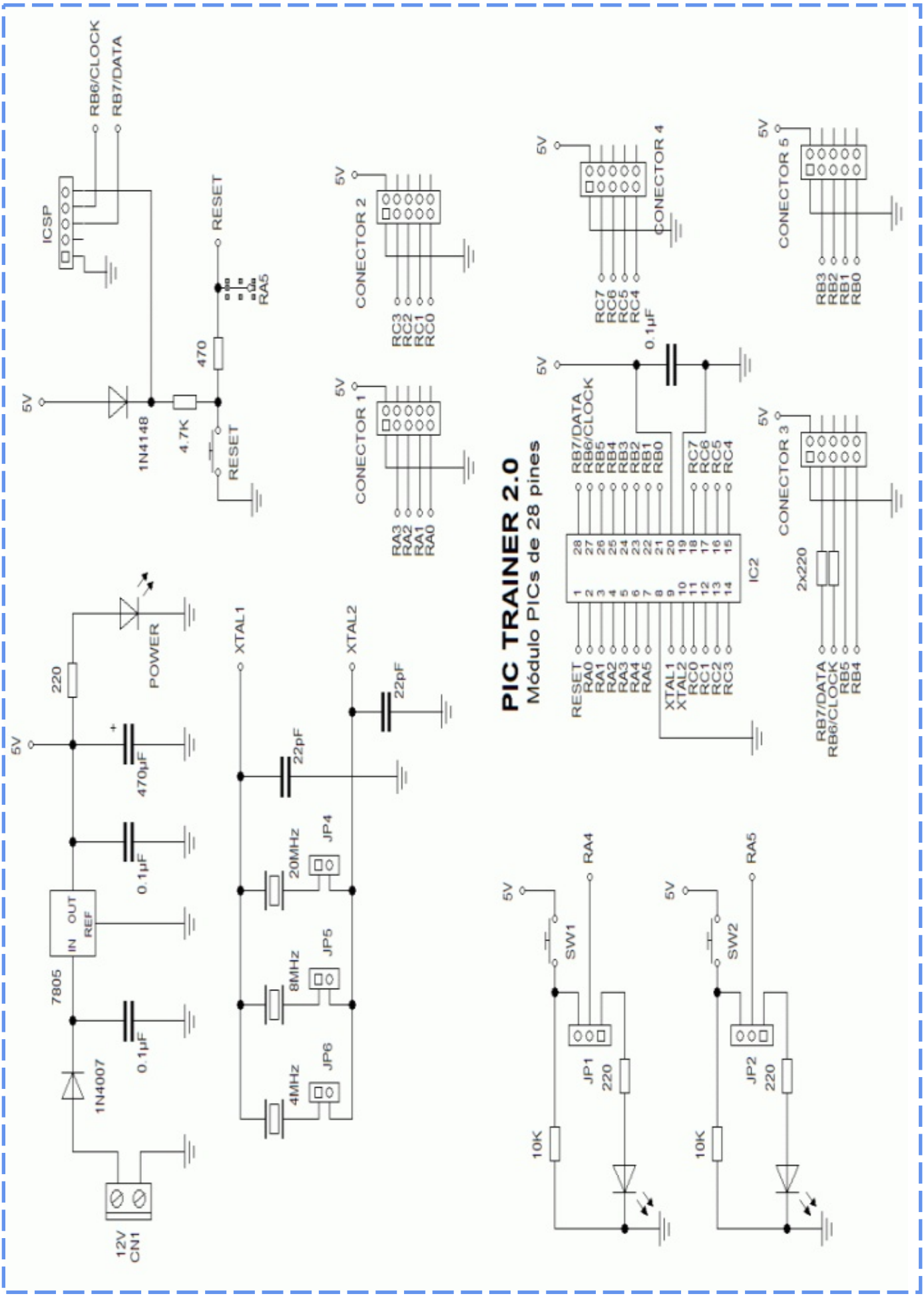
"Veremos como construir el módulo que permite realizar experimentos con PICs de 28 pines"

Siguiendo con la serie de placas destinadas a formar parte de nuestro entrenador modular PIC TRAINER, vamos a ver como construir el módulo que permite realizar experimentos con PICs de 28 pines. Dado que el diseño del circuito y la placa de circuito impreso se asemeja mucho a las dos anteriores (PIC TRAINER 40 y PIC TRAINER 18), solo veremos sus puntos más relevantes, pudiendo el lector consultar los otros artículos de la serie para conocer más detalles interesantes sobre el proyecto. Al igual que antes, tenemos que recomendar fuertemente la consulta de las hojas de datos de los chips que queramos pinchar en esta placa, ya que a pesar de que Microchip distribuye la función de cada pin casi siempre de la misma forma, esto no se garantiza en el 100% de los modelos.

.El circuito

El módulo consiste básicamente en un zócalo encargado de alojar al microcontrolador bajo prueba, un mecanismo de RESET, un conector que permite la programación del integrado sin retirarlo de la placa, una serie de conectores IDC encargados de comunicar al módulo con las placas de ampliación previstas (o con las que el lector desarrolle), una etapa de alimentación y un par de LEDs y pulsadores destinados a proveer una mínima capacidad de entrada y salida onboard.

La alimentación del módulo se ha resuelto mediante uno de los populares reguladores de voltaje integrados de la serie LM78xx. Se trata de un LM7805, que puede proporcionar 5V perfectamente estables a partir de una fuente de corriente continua con una tensión de entre 7.5 y 15V. Una bor-


PIC TRAINER 2.0
Módulo PICs de 28 pines
ICSP
RB6/CLOCK
RB7/DATA
RESET
RA5
1N4148
4.7K
470
5V
CONECTOR 1
CONECTOR 2
CONECTOR 3
CONECTOR 4
CONECTOR 5
RA3 RA2 RA1 RA0
RC3 RC2 RC1 RC0
RB7/DATA RB6/CLOCK RB5 RB4
2x220
RC7 RC6 RC5 RC4
RB3 RB2 RB1 RB0
0.1µF
IC2
RESET RA0 RA1 RA2 RA3 RA4 RA5
XTAL1 XTAL2 RC0 RC1 RC2 RC3
RB7/DATA RB6/CLOCK RB5 RB4
RB7/DATA RB6/CLOCK RB5 RB4 RB3 RB2 RB1 RB0
RC7 RC6 RC5 RC4
XTAL1
XTAL2
22pF
20MHz
8MHz
4MHz
JP4
JP5
JP6
SW1
SW2
JP1
JP2
220
10K
RA4
RA5
POWER
470µF
0.1µF
7805
IN OUT REF
1N4007
12V CN1

nera de dos tornillos permite alimentar a la placa, y un diodo 1N4007 protege al circuito de una conexión con la polaridad equivocada. Tal como recomienda la hoja de datos del regulador de voltaje, hemos colocado los dos condensadores de 0.1 uF de rigor. Además, un condensador electrolítico de 470uF/16V filtra el poco ripple que pueda haber escapado al filtro de la fuente externa, y un diodo LED, en serie con un resistor de 220V, se enciende para indicarnos que el circuito está alimentado.

Tal como ocurría con las placas para microcontroladores de 18 o 40 pines, en lugar de utilizar un cristal como oscilador del PIC hemos colocado 3 de ellos, tambien seleccionables mediante una serie de jumpers (identificados como JP4, JP5 y JP6). De esta manera podremos probar nuestros programas o microcontroladores a diferentes frecuencias de trabajo. Los dos condensadores de 22pF completan esta parte del circuito. En caso de utilizar PICs que funcionen a más de 20 MHz, es posible que haya que cambiarlos por condensadores de 15pF o algo menos. Recordemos que si se coloca más de un jumper a la vez el microcontrolador no funcionará.

En cuanto a los cristales, hemos elegido (como puede verse en el esquema eléctrico) valores de 4MHz, 8MHz y 20MHz, pero nada impide utilizar otros. El lector

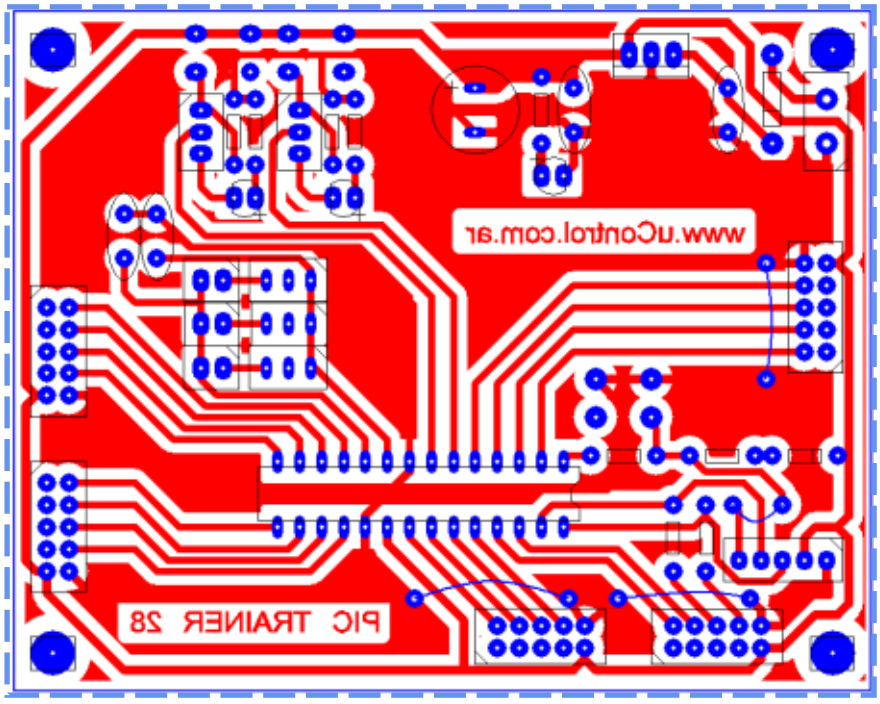


Figura 1. Se trata de un sencillo PCB, de una sola cara.

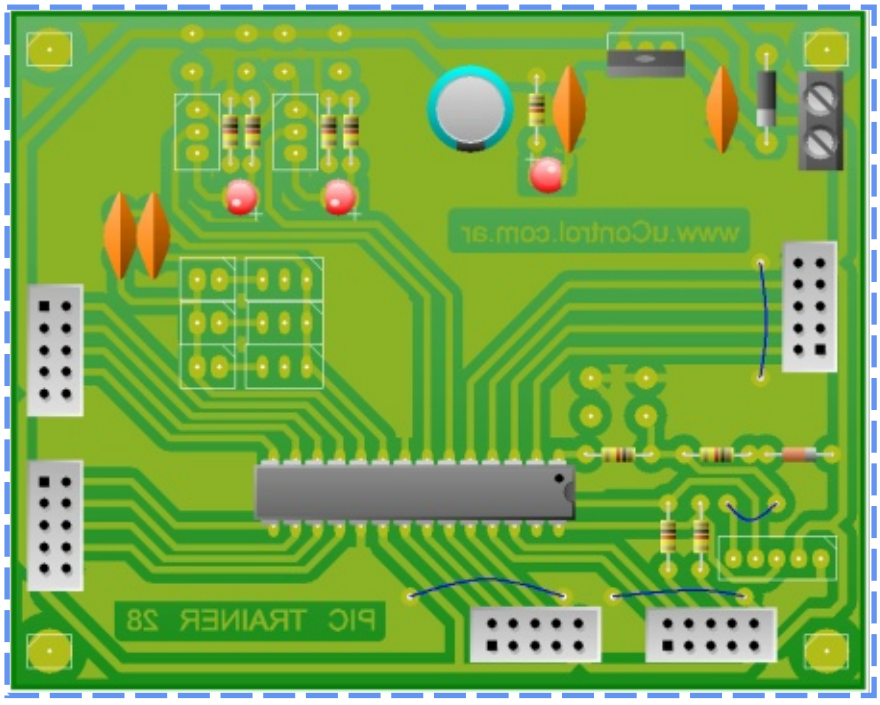


Figura 2. Este es el aspecto que tendrá el módulo terminado.

puede cambiarlos a gusto.

En el caso utilizar los pulsadores o LEDs existentes en la placa, para ingresar (o representar) datos a (o de) nuestro programa, deberemos mover los jumpers JP1 y JP2, que permiten seleccionar si conectamos al PIC el LED o el pulsador.

Si deseamos seleccionar los pulsadores, debemos recordar que estos ponen el pin correspondiente a 5V cuando son presionados. Mientras que están en reposo, las entradas se mantienen a GND a través de sendos resistores de 10K.

Los conectores siguen

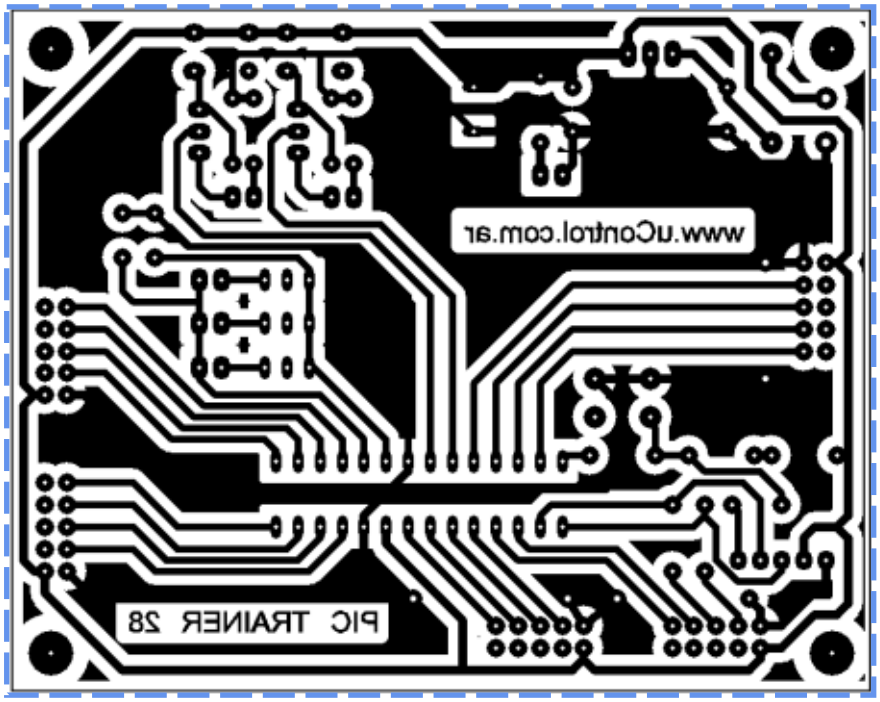


Figura 3. El PCB a utilizar mide solo 98x80 milímetros.

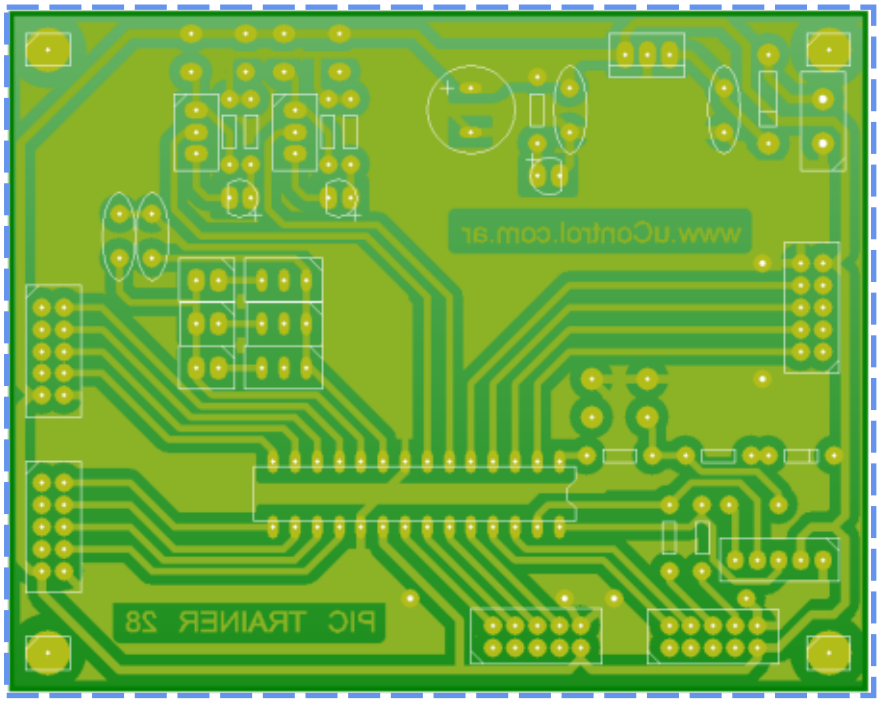


Figura 4. Aspecto de la serigrafía del PCB a construir.

las mismas normas que explicamos en el artículo principal de nuestra revista número 4, así que no deberías tener problemas a la hora de determinar la función de cada pin. Como regla general, recuerda que de los pines exteriores de cada conector solo se emplea uno (+V) y los otros cuatro están sin conectar. De lo cinco interior, uno corresponde a GND y los otros 4 a datos. Deberias tener a mano el grafico con la función de cada pin a mano cuando decidas hacer algún programa para cargar en el entrenador.

La única excepción, o desvío de lo normal, que puedes ver en el diagrama de los conectores es en los pines correspondientes a RB6 (CLOCK) y RB7 (DATA), ya que poseen un resistor de 220 ohms en serie. Cumplen con la función de permitir programar el PIC mediante el conector ICSP sin necesidad de retirar el cable que conecta el entrenador con el modulo de turno. Por supuesto, si lo deseas puedes reemplazar esos dos resistores por sendos puentes. Solo deberás quitar el cable plano a la hora de reprogramar el PIC.

El pin 1, que corresponde al RESET en los microcontroladores PIC de 28 pines compatibles con este entrenador, esta unido a un pulsador a través de un resistor de 470 ohms y a +V mediante otro de 4.7K y un diodo 1N4148. Al presionar el pulsador, el microcontrolador se resetea. Durante el funcionamiento normal del programa, el pin esta a +V. El diodo impide los problemas que podrían surgir entre las alimentaciones del módulo y del programador al utilizar el conector ICSP.

.Construcción

Si ya has montado alguno de nuestros proyectos, estarás al tanto de que basta con descargar el archivo PDF correspondiente al PCB desde nuestra Web, y mediante la forma que más te guste (puedes usar el

"método de la plancha" explicado en la revista numero 1) debes transferir el diseño a un trozo de PCB virgen. Luego, lo sumerges en un baño de cloruro férrico; y por ultimo, luego de una buena limpieza, le haces los agujeros.

A la hora de soldar los componentes ten en cuenta que por lo general resulta más sencillo si primero colocas los que son más bajos, como los puentes, diodos, zócalos y resistores; y dejas para el final los conectores, regulador de voltaje y condensadores. Asegúrate de que, involuntariamente, no haces un puente entre dos puntos del circuito.

Presta especial atención a la hora de soldar los componentes que tienen "polaridad", como los diodos, LEDs y condensadores electrolíticos. También es importante que coloques el zócalo destinado al PIC en la dirección correcta, ya que de hacerlo mal puedes confundirte cuando insertes el microcontrolador, dañándolo.

Una vez montado todo, sin colocar el PIC en su lugar, alimenta el circuito con una tensión de entre 7.5 y 12V. El LED "Power" debería encenderse. Si es así, verifica con un multímetro que la tensión entre los pines 8 (GND) y 20 (Vcc) del zócalo del microcontrolador sea de 5V. También puedes verificar que en los conectores de expansión esté presente esa tensión. Si todo esta bien, ya tienes listo tu entrenador. Caso contrario, repasa las soldaduras y posición de los componentes.

.Conclusión

Ya podemos comenzar a experimentar con PICs de 28 pines. Hay mucho para aprender con ellos y, si decides encarar la construcción de alguno de los módulos accesorios (como el módulo de 8 E/S publicado en el número anterior de la revista uControl), podrás multiplicar por 100 las posibilidades de este entrenador.

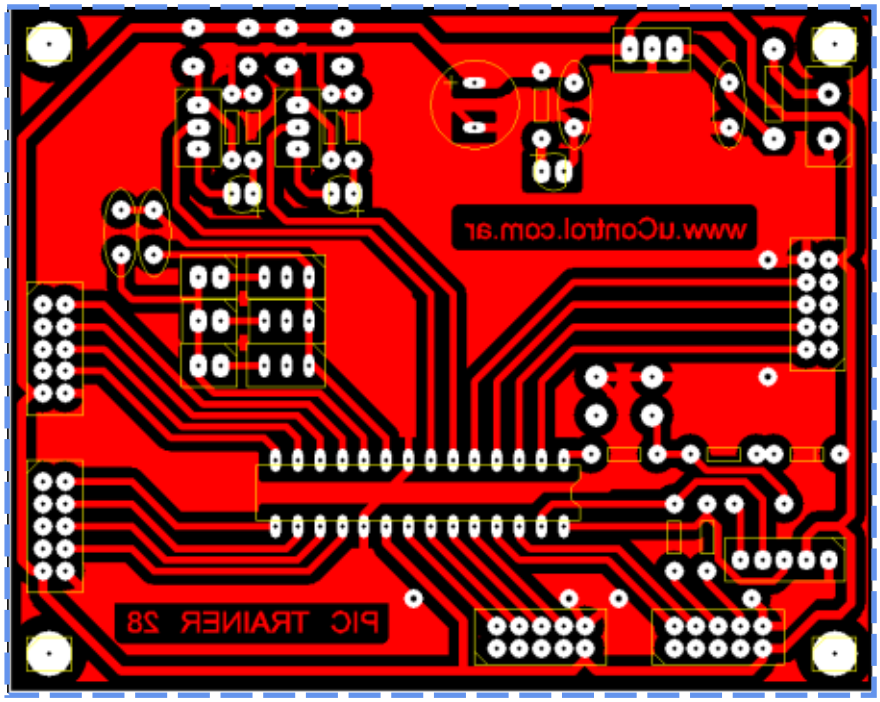
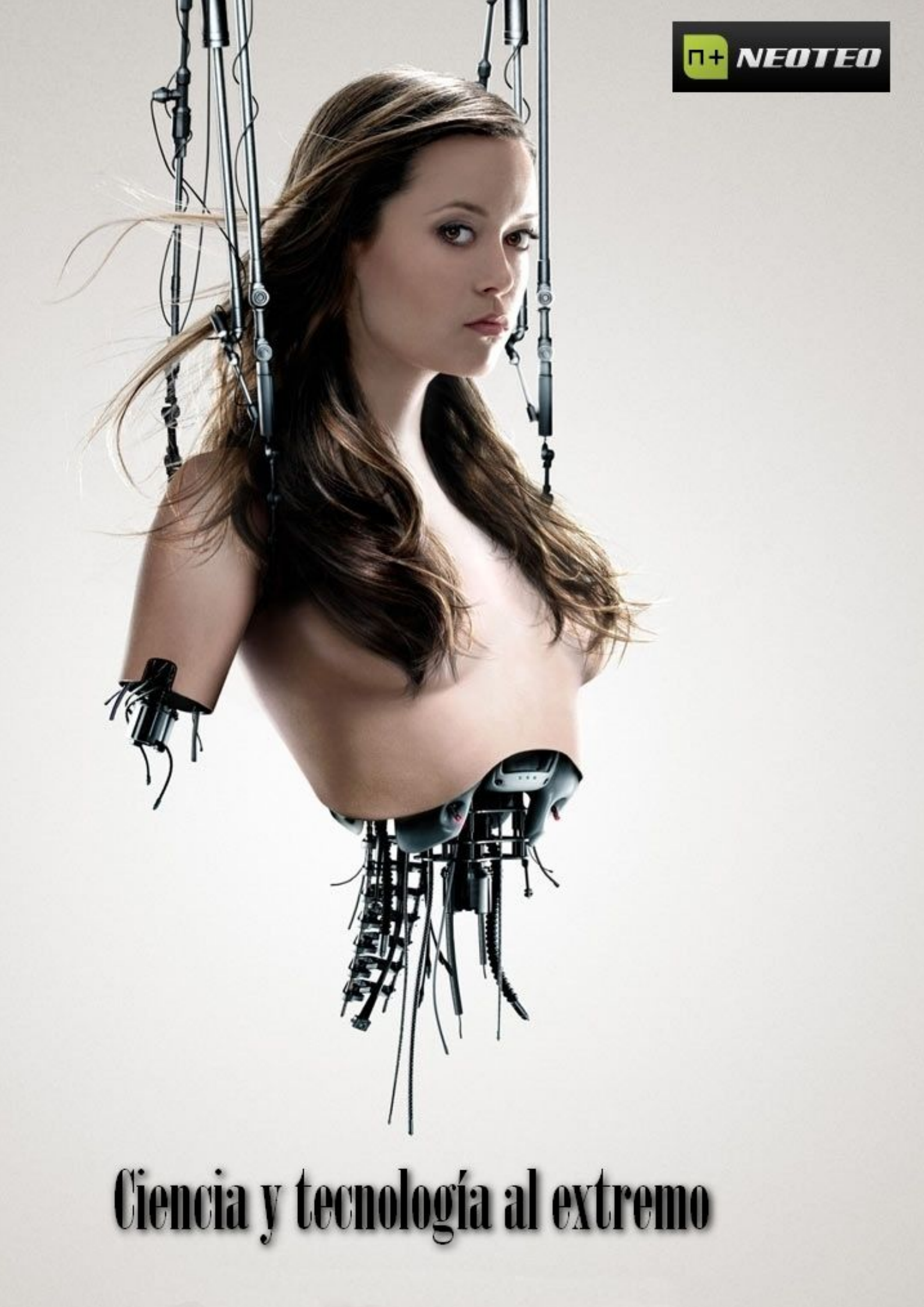


Figura 5. Otra vista del diseño del PCB.


NEOTEO
Ciencia y tecnología al extremo

PIC16F628A en assembler

segunda parte

En esta segunda parte del tutorial de programación en lenguaje assembler nos introduciremos en el manejo del direccionamiento indirecto y en el mundo de las interrupciones del microcontrolador. Explicaremos qué son, para qué sirven, de cuántas disponemos y su implementación en nuestros programas. ¡Manos a la obra!

// por: Alejandro Casanova //
inf.pic.suky@live.com.ar

En la programación de los microcontroladores PIC la mayoría de las instrucciones emplean direccionamiento directo, pero también es posible que operen en un modo de direccionamiento indirecto. Para este modo se emplean dos registros especiales: el FSR y el INDF (este último no es un registro físico).

"En el *modo de direccionamiento indirecto se emplean dos registros especiales: el FSR y el INDF*"

El registro FSR se emplea para "señalar o apuntar" a una dirección de la memoria RAM cuyo contenido puede ser leído o escrito de forma indirecta empleando cualquier instrucción que use como operando al registro INDF. Esta forma de direccionamiento es particularmente útil cuando se manejan tablas o arreglos de datos.

Utilizaremos el direccionamiento Indirecto para crear la tabla de control del Display de 7 segmentos y en este caso no utilizaremos el pulsador, solo se hará el contador automático de 0 a 9. Al iniciar el microcontrolador cargaremos la tabla para controlar el display de 7 segmentos en la memoria de datos (GPR) con direccionamiento indirecto.

Luego, al realizar el conteo leeremos el código correspondiente almacenado y lo enviaremos al PORTB.

Aquí utilizamos el registro STATUS nuevamente, pero para control de las operaciones aritméticas. Se guardará el código de 7 Segmentos del 0 al 9, en los registros 0x30 a 0x39. Si nuestro contador nos direcciona el registro ubicado en 0x3A, que seria el "10", lo reseteamos y direccionamos el "0", ósea registro 0x30. Esto lo hacemos realizando la resta del registro seleccionado y 0x3A, FSR – 0x3A, y si el resultado es cero, reseteamos.

El bit Z (Zero) del registro STATUS, este indica si una operación lógica o aritmética realizada da como resultado cero. También tenemos el bit C

```
; DIRECTO:
; Definimos registro en la memoria de datos.-
MiRegistro    equ    0x20          ; Ubicado en 0x20.-

; Cargamos dato en el registro.-
        movlw 0x8A                  ;
        movwf MiRegistro            ; MiRegistro = 0x8a.-
; Leemos dato del registro.-
        movfw MiRegistro            ; Movemos el valor que tenga MiRegistro a W.-
        movwf PORTB                 ; Por ejemplo, lo cargamos en PORTB.-

; INDIRECTO:
; Cargamos dato en el registro.-
        movlw 0x20                  ;
        movwf FSR                   ; Direccionamos Registro de datos ubicado en 0x20.-
        movlw 0x8A                  ;
        movwf INDF                  ; Cargamos registro direccionado con el valor 0x8A.-

; Leemos dato en el registro.-
        movlw 0x20                  ;
        movwf FSR                   ; Direccionamos Registro de datos ubicado en 0x20.-
        movfw INDF                  ; Movemos el valor que tenga el registro seleccionado a W.-
        movwf PORTB                 ; Por ejemplo, lo cargamos en PORTB.-
```

Direccionamiento directo vs. indirecto.

(Carry) (0), que en instrucciones aritméticas se activa cuando se presenta un acarreo desde el bit mas significativo del resultado, el bit DC (Digit Carry), que en operaciones aritméticas se activa si ocurre acarreo entre el bit 3 y bit 4.

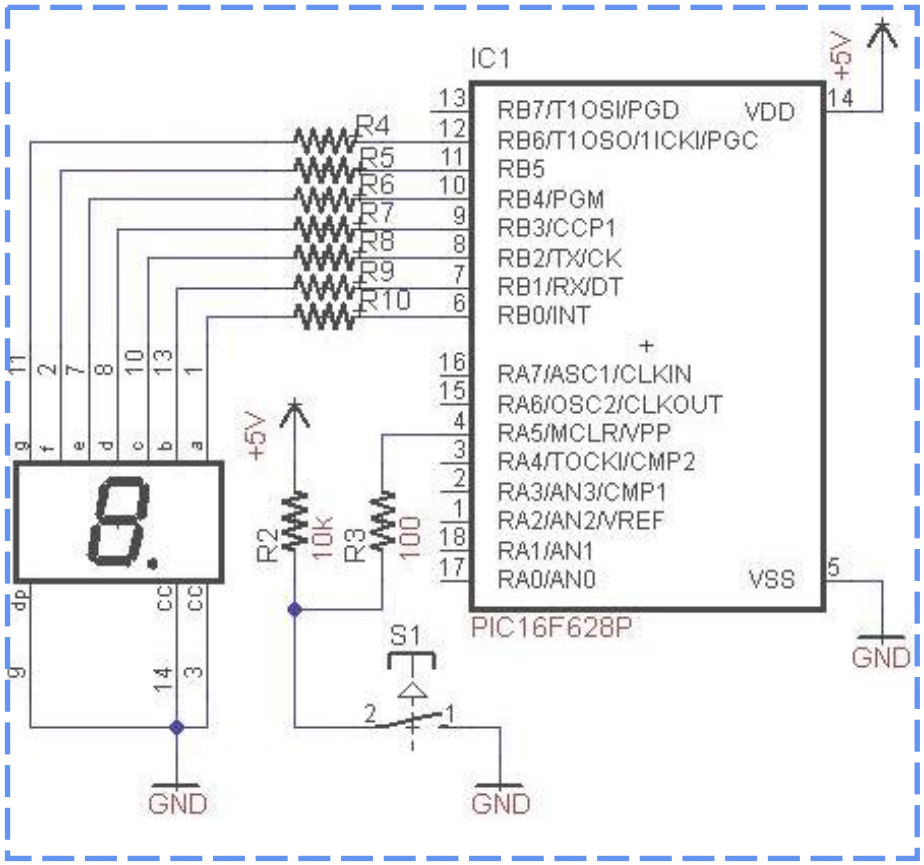


Figura #1. Control de display de 7 segmentos.

```
; **** Encabezado ****
List p=16f628A                  ; list directive to define processor
#include <p16f628A.inc>   ; processor specific variable definitions
  __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_ON
;**** Definición de variables ****
Contador    equ         0x20   ; Seleccionamos posición en la  RAM (GPR) para guardar
Contador1   equ         0x21   ; Registro utilizado en demora.-
Contador2   equ         0x22   ; Registro utilizado en demora.-

Reset
        org     0x00            ; Aquí comienza el micro.-
        goto    Inicio          ; Salto a inicio de mi programa.-
;**** Programa principal ****
;**** Configuración de puertos ****
        org     0x05
Inicio
        bsf     STATUS,RP0      ; Pasamos de Banco 0 a Banco 1.-
        clrf    TRISB           ; PORTB como salida.-
        bcf     STATUS,RP0      ; Paso del Banco 1 al Banco 0
        call    Config_Tabla ; Cargamos registros con Código de 7 segmentos.-
        movfw   INDF            ; Leemos código de 7 Segmentos para el CERO.-
        movwf   PORTB           ; Mostramos el CERO.-
;**** Bucle de visualización ****
Bucle
        call    Demora_190ms    ; Demora para visualizar Display
        incf    FSR,1           ; Incrementamos Puntero.-
        movlw   0x3A            ; Consultamos si se pide código para mostrar "10",
        subwf   FSR,0           ; si es así reseteamos FSR, apunta a 0x30--> "0".-
        btfss   STATUS,Z        ; Si Z=1 --> 0x3A - FSR = 0.-
        goto    Muestro_Display ; No, muestro display.-
        movlw   0x30            ; Si reseteo puntero.-
        movwf   FSR             ;
Muestro_Display
        movfw   INDF            ; Leo Registro que apunta FSR.-
        movwf   PORTB                   ; Lo cargo en PORTB.-
        goto    Bucle           ; Continuo conteo.-
;**** Demora ****
Demora_190ms
        movlw   0xFF            ;
        movwf   Contador1       ; Iniciamos contador1.-
Repeticion1
        movlw   0xFF            ;
        movwf   Contador2       ; Iniciamos contador2
Repeticion2
        decfsz  Contador2,1     ; Decrementa Contador2 y si es 0 sale.-
        goto    Repeticion2     ; Si no es 0 repetimos ciclo.-
        decfsz  Contador1,1     ; Decrementa Contador1.-
        goto    Repeticion1     ; Si no es cero repetimos ciclo.-
        return                  ; Regresa de la subrutina.-
```

```
;**** Cargamos tabla en memoria ****
Config_Tabla
        movlw 0x30   ;
        movwf FSR    ; Direccionamos el registro 0x30 de la memoria RAM (GPR).-
        movlw 0x3F   ; Cargamos el código para mostrar el CERO.-
        movwf INDF   ; Lo guardamos donde apunta FSR --> 0x30.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x31.-
        movlw 0x06   ; Cargamos código para UNO.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x32.-
        movlw 0x5B   ; Cargamos código para DOS.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x33.-
        movlw 0x4F   ; Cargamos código para TRES.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x34.-
        movlw 0x66   ; Cargamos código para CUATRO.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x35.-
        movlw 0x6D   ; Cargamos código para CINCO.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x36.-
        movlw 0x7D   ; Cargamos código para SEIS.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x37.-
        movlw 0x07   ; Cargamos código para SIETE.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x38.-
        movlw 0xFF   ; Cargamos código para OCHO.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        incf    FSR,1 ; Incrementamos FSR, ahora apunta a 0x39.-
        movlw 0x6F   ; Cargamos código para NUEVE.-
        movwf INDF   ; Lo guardamos donde apunta FSR.-
;.....................
        movlw 0x30   ;
        movwf FSR    ; Direccionamos Registro del CERO.-
        return       ; Cargado los valores, retornamos.-
;.....................................................................
        end
```

.Interrupciones

Una de las características más importante de los microcontroladores es que tienen la posibilidad de manejar interrupciones. Se trata de un acontecimiento que hace que el micro deje de lado lo que se encuentra realizando, atienda ese suceso, regrese y continúe con lo suyo.

Hay dos tipos de interrupciones posibles, una es mediante una acción externa (es decir por la activación de uno de sus pines), la otra es interna (por ejemplo cuando ocurre el desbordamiento de uno de sus registros)

En el 16F628A hay 10 fuentes de interrupción:

- Flanco ascendente o descendente del pin RB0/INT, que regresa al PIC del modo SLEEP.
- Por los pines RB4 a RB7, configurados como entrada y en caso de que alguno de ellos cambie de estado.
- Por desbordamiento del registro TMR0, cuando este registro pasa de 255 a 0 en decimal.
- Por desbordamiento del registro Timer1, cuando pasa de 65535 a 0 en decimal.
- Por desbordamiento del registro TMR2, cuando este registro supera el valor del registro PR2.
- Al completar la escritura de la EEPROM de datos.
- Cuando se recibe un dato por USART.
- Cuando se completa el envío de un dato por USART.
- Ocurre un cambio en los comparadores analógicos.
- Interrupción provocada por el módulo CCP en modo comparación o captura.

Cada fuente de interrupción esta controlada por 2 bits. Un bit local de interrupciones (terminado en E) de permiso o prohibición de ejecución. Si esta en 0 bloqueará la solicitud de interrupción, y si esta en 1 permitirá la ejecución. Un bit que actúa como señalizador (terminado en F) el cual es activado (puesto a 1) si se ha producido la interrupción.

Además existen 2 bits de control global, el bit GIE (INTCON <7>) el cual si esta desactivado bloquea todas las solicitudes de interrupción y el bit PEIE (INTCON <6>) que sería como un segundo bit de control global de interrupciones exceptuando a desbordamiento del Timer0, detección de flanco en RB0 y cambio de estado de RB4-RB7.

Lo anterior descrito puede entenderse observando el diagrama lógico de la figura #2.

Nota: Todos estos bits al resetearse o iniciarse el micro se encuentran en 0.

"Hay dos tipos posibles de interrupciones: internas y externas. El PIC16F628A cuenta con 10 de ellas en total."

El bit GIE se borra automáticamente cuando se reconoce una interrupción para evitar que se produzca otra mientras se está atendiendo a la primera y al retornar de la interrupción con la instrucción RETFIE, el bit GIE se vuelve a activar poniéndose a 1. En cambio los bits señalizadores o banderas de interrupción deben ser puestos a cero por el tratamiento de la interrupción realizada por el usuario (programador).

Cuando una interrupción está habilitada (su bit local de habilitación está activado, el bit GIE está activado y/o dependiendo del caso el bit PEIE está activado) y ocurre el evento que la activa, el valor de PC se guarda en la PILA y en éste se carga el 0x04 (único vector de interrupción). Es a partir de esta dirección que se debe colocar el tratamiento de la interrupción, detec-

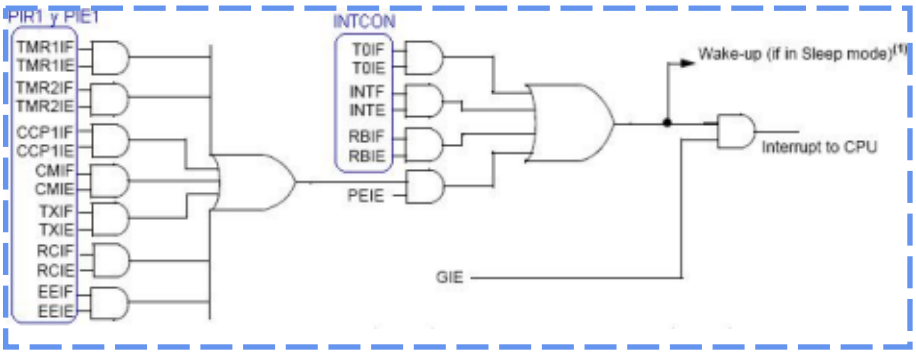


Figura #2. Lógica de interrupciones.

tando por medio de los bits banderas cual de los eventos ha ocurrido y actuar según sea el caso.

Nota: el único registro que se salva en la PILA es PC, para preservar algún otro registro debe ser el propio programa de atención a la interrupción el que se encargue de salvar su estado al inicio de la rutina y de devolverlos al final del mismo.

.Resumiendo

• Ocurre un evento que solicita la interrupción, el bit bandera se activa.
• Si el bit local de interrupciones esta activado, el bit GIE activado y/o PEIE activado se produce la interrupción.
• Se desactiva GIE.
• El valor del PC se guarda en la PILA, y PC se carga con 0x04.
• Se ejecuta la rutina de atención de interrupción creada por el usuario, guardando los registros necesarios y realizando la bifurcación correspondiente para la atención del evento por medio del bit bandera.
• Al finalizar el tratamiento del evento, se devuelven los valores de los registros salvados y se borran por software los bits banderas.
• La rutina de atención deberá terminar con una instrucción RETFIE, la cual activa nuevamente el bit GIE (GIE=1) y lee la PILA para restablecer PC y continuar la ejecución del programa que fue interrumpido en la siguiente instrucción.

.Registros utilizados

INTCON. Registro de lectura y escritura que contiene varios bits de señalización y habilitación para el desbor-damiento del TMR0, cambio sobre el puerto RB e interrup-ción externa en la patilla RB0/INT.

0. RBIF: Indicador de interrupción por cambio de estado RB4-RB7.
1. INTF: Indicador de interrupción externa.
2. T0IF: Indicador de interrupción por desbordamiento de Timer 0.
3. RBIE: Habilitación de interrupción por cambio de estado RB4-RB7.
4. INTE: Habilitación de interrupción externa.
5. T0IE: Habilitación de interrupción por desbordamiento de Timer 0.
6. PEIE: Habilitación de interrupción de periféricos.
7. GIE: Habilitación general de interrupciones.

PIR1. El registro PIR1 contiene los bits de señalización individual de las interrupciones de periféricos.

0. TMR1IF: Indicador de interrupción por desbordamiento de Timer 1.
1. TMR2IF: Indicador de interrupción por desbordamiento de Timer 2.
2. CCP1IF: Indicador de interrupción del módulo de Captura / Comparación.
a) Modo Comparador: Coincidencia entre TMR1 y CCP1.
b) Modo Captura: Ha ocurrido una captura de TMR1
3. No Implementado.
4. TXIF: Indicador de interrupción de fin de transmisión USART.
5. RCIF: Indicador de interrupción de llegada de datos USART.
6. CMIF: Indicador de interrupción por cambio de estado de alguna de las salidas de los comparadores.
7. EEIF: Indicador de interrupción de fin de escritura eeprom interna.

PIE1. Registro que posee los bits de habilitación individual para las interrupciones de periféricos. El bit PEIE del registro INTCON debe ser 1 para permitir la habilitación de cualquier interrupción de periférico.

0. TMR1IE: Habilitación de interrupción por desbordamiento de Timer 1.
1. TMR2IE: Habilitación de interrupción por desbordamiento de Timer 2.
2. CCP1IE: Habilitación de interrupción del módulo de Captura/Comparación/PWM.
3. No Implementado.
4. TXIE: Habilitación de interrupción de fin de transmisión USART.
5. RCIE: Habilitación de interrupción de llegada de datos USART.
6. CMIE: Habilitación de interrupción por cambio de estado de alguna de las salidas de los comparadores.
7. EEIE: Habilitación de interrupción de fin de escritura eeprom interna.

.Rutina de servicio de interrupciones

Primero se debe guardar el contenido del registro W y STATUS. El problema de mover W a otro registro (haciendo uso de movf) es que esta instrucción corrompe la bandera Z, modificando el registro de STATUS. Según la hoja de datos otorgada por Microchip, en uno de sus apartados recomienda una secuencia de código que permite guardar y restaurar los registros sin modificarlos.

```
;**** Rutina de servicio de Interrupción ****
;  Guardado de registro W y STATUS.-
Inicio_ISR
        movwf W_Temp            ; Copiamos W a un registro
                                ; Temporario.-
        swapf STATUS, W         ; Invertimos los nibles del
                                ; registro STATUS.-
        movwf STATUS_Temp       ; Guardamos STATUS en un
                                ; registro temporal.-
ISR
; Atendemos la interrupción.-
; Restauramos los valores de W y STATUS.-
Fin_ISR
        swapf STATUS_Temp,W     ; Invertimos lo nibles de
                                ; STATUS_Temp.-
        movwf STATUS
        swapf W_Temp, f         ; Invertimos los nibles y lo
                                ; guardamos en el mismo
                                ; registro.-
        swapf W_Temp,W          ; Invertimos los nibles
                                ; nuevamente y lo
                                ; guardamos en W.-
        retfie                  ; Salimos de interrupción.-

; Los registros W_Temp y STATUS_Temp son registros
; alternativos para guardar temporariamente sus valores
; correspondientes.
```

.Interrupción externa, RB0/INT

Para el control de la interrupción externa se necesita un bit adicional INTEDG (OPTION_REG<6>) que selecciona el flanco que generará la interrupción. Si está en 1 se genera por flanco ascendente y en 0 por flanco descendente.

Para mostrar su uso haremos un ejemplo sencillo que muestre como se configura, el cual al presionar un pulsador conectado a RB0 cambiará el estado de un LED conectado a RB1, para ello configuramos que la interrupción de genere por flanco ascendente. En esta caso vamos a realizar el guardado de los registro W y STATUS

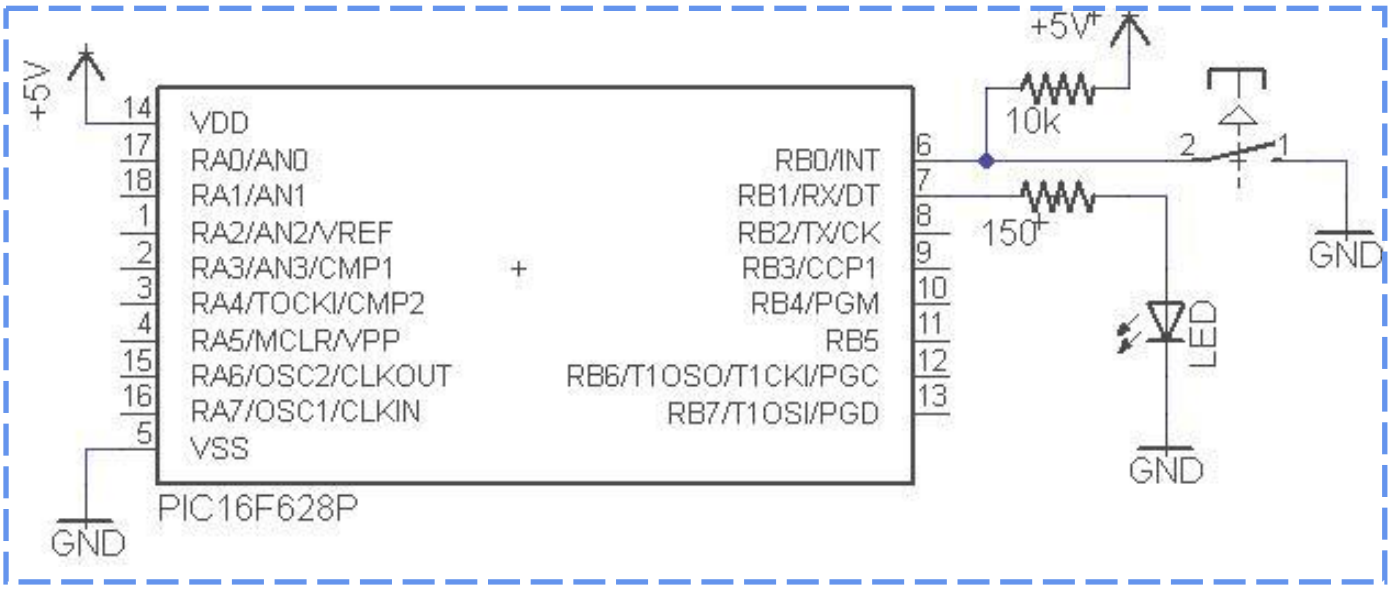


Figura #3. Hardware para interrupción externa RB0/INT.

```
; **** Encabezado ****
list            p=16f628A    ; list directive to define processor
#include       <p16f628A.inc>      ; processor specific variable definitions
  __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_OFF
;**** Definición de variables ****
Contador1    equ            0x20
Contador2    equ            0x21

Pulsador      equ            0              ; pin RB0
Led             equ            1              ; pin RB1

;**** Inicio del Micro ****
Reset
        org     0x00            ; Aquí comienza el micro.-
        goto    Inicio                  ; Salto a inicio de mi programa.-
;**** Vector de Interrupción ****
        org     0x04            ; Atiendo Interrupción.-
        goto    ISR
; **** Programa Principal ****
;**** Configuración de puertos ***
        org     0x05            ; Origen del código de programa.-
Inicio
        bsf     STATUS,RP0          ; Pasamos de Banco 0 a Banco 1.-
        movlw b'11111101'   ; RB0 como entrada y RB1 como salida.-
        movwf TRISB
        bsf     OPTION_REG,INTEDG     ; Config. Por flanco Ascendente.-
        bcf     STATUS,RP0          ; Paso del Banco 1 al Banco 0
        bcf     PORTB,Led   ; El Led comienza apagado.-
        movlw b'10010000'  ; Habilitamos GIE y INTE (interrupción por RB0)
        movwf INTCON
;**** Bucle infinito ****
Bucle
        nop                     ;
        goto    Bucle           ;
;.....
;**** Rutina de servicio de Interrupción ****
;**** Interrupción por RB0 ****
ISR
        btfss   INTCON,INTF                 ; Consultamos si es por RB0.-
        retfie                          ; No, Salimos de interrupción.-
        call    Demora_20ms                 ; Comprueba si es rebote.-
        btfss   PORTB,Pulsador
        goto    Fin_ISR                     ; Es rebote, entonces salimos.-
        btfss   PORTB,Led           ; Si esta prendido, lo apagamos.-
        goto    Prender_Led
        bcf             PORTB,Led           ; Apagamos Led
        goto    Fin_ISR
Prender_Led
        bsf             PORTB,Led           ; Encendemos Led
```

```
Fin_ISR
        bcf             INTCON,INTF             ; Limpiamos bandera.-
        retfie                                  ; Salimos de interrupción.-
;.....................................
;**** Demora ****
Demora_20ms
        movlw 0xFF              ;
        movwf Contador1         ; Iniciamos contador1.-
Repeticion1
        movlw 0x19              ;
        movwf Contador2         ; Iniciamos contador2
Repeticion2
        decfsz Contador2,1      ; Decrementa Contador2 y si es 0 sale.-
        goto   Repeticion2      ; Si no es 0 repetimos ciclo.-
        decfsz Contador1,1      ; Decrementa Contador1.-
        goto   Repeticion1      ; Si no es cero repetimos ciclo.-
        return                  ; Regresa de la subrutina.-
        end
```

para ejemplificar el método pero no es necesario ya que el microcontrolador solo espera a que ocurra la interrupción.

.Interrupción por cambio de estado RB4-RB7

Como el titulo lo indica esta interrupción se genera cuando se presenta un cambio de nivel en cualquiera de los pines RB4 a RB7 configurados como entradas.

Aprovecharemos esta interrupción para detectar cuando se ha presionado una tecla de un teclado matricial. Un teclado matricial es un simple arreglo de botones conectados en filas y columnas, de modo que se pueden leer varios botones con el mínimo número de pines requeridos. Un teclado matricial 4×3 solamente ocupa 4 líneas de un puerto para las filas y otras 3 líneas para las columnas, de este modo se pueden leer 12 teclas utilizando solamente 7 líneas de un microcontrolador.

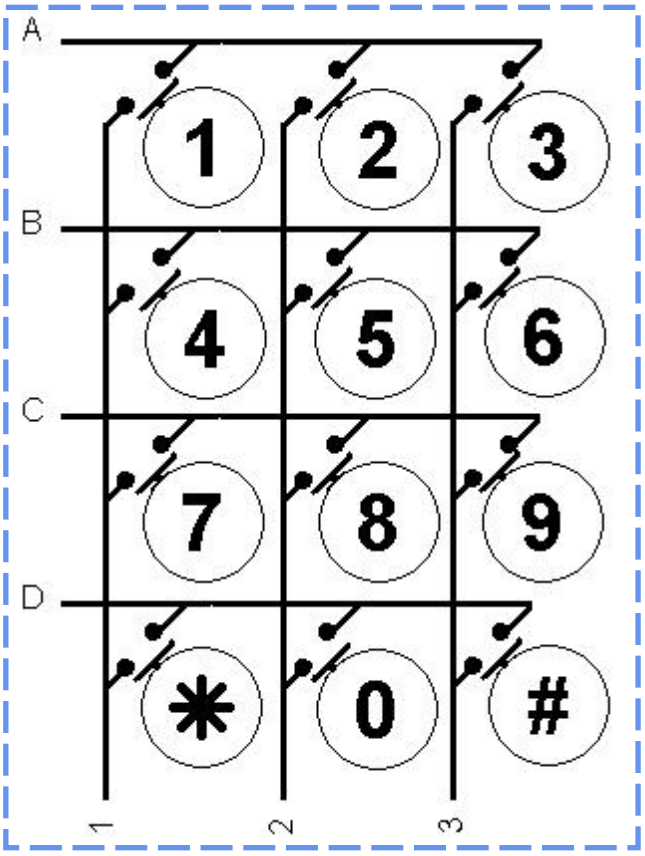


Figura #4. Estructura de un teclado matricial.

Configuraremos RB0 a RB3 como salida y las colocaremos a nivel bajo. RB4 y RB7 configuradas como entradas y habilitaremos las resistencias pull-up internas del puerto B, RPBU (OPTION_REG<7>) a 0, entonces en estado normal (sin presión de teclas) estarán a nivel alto. Al presionar una tecla se conecta una fila con una columna, se produce un cambio de nivel en alguna de

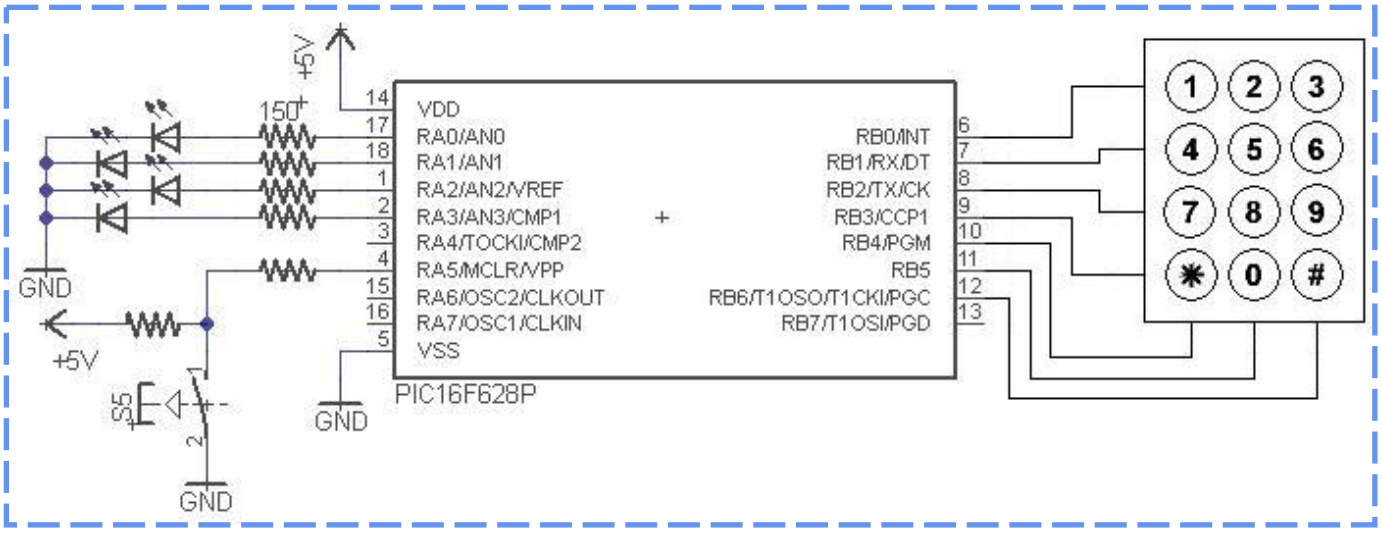


Figura #5. Hardware para control de un teclado matricial.

las columnas (de nivel alto a bajo), y se genera la interrupción. Para detectar que tecla se ha presionado, se colocan RB0 a RB3 a nivel alto, y se pasan a nivel bajo de a una por vez, detectando si se produce algún cambio en las columnas. Se utiliza una variable que se incrementa con la cuenta de las teclas revisadas, de este modo al detectar una pulsación el valor de la cuenta será el valor de la tecla presionada. Si al final no se presionó ninguna tecla la variable se pone a cero y la cuenta vuelve a comenzar.

Algoritmo de detección de tecla pulsada: al ocurrir una interrupción se puede determinar que columna ha cambiado de estado, pero no la fila. Entonces para detectarla iremos colocando a nivel bajo solo una fila por vez, y testearemos las columnas, al detectar un nivel bajo en alguna determinaremos mediante la cuenta de teclas testeadas exactamente cual ha sido.

En nuestro ejemplo representaremos la tecla presionada en forma binaria con LEDs conectados al puerto A.

.El código

Tener en cuenta que también con la variable NTecla (Numero de Tecla presionada) se puede utilizar como entrada a una tabla para codificar en ASCKII la tecla presionada.

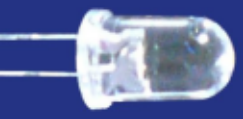
Nota: La bandera RBIF debe ponerse a cero por software pero además debe realizarse previamente una operación de lectura (o escritura) del Puerto B para que deje de darse la discrepancia entre el valor actual y el valor leído, y no genere una nueva interrupción.

```
        movfw NTecla
        call    Tabla_TMatricial
;Codificación de Tecla presionada:
Tabla_TMatricial
        addwf PCL,1
        DT      "1","2", "3", "4", "5", "6", "7", "8", "9", "*", "0", "#"
```

"En el mercado se consiguen todo tipo de teclados matriciales adaptables a nuestros proyectos"

```
; **** Encabezado ****
 list p=16f628A        ; list directive to define processor
 #include <p16f628A.inc>   ; processor specific variable definitions
 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_ON
;**** Definición de variables ****
NTecla                  equ     0x20            ; Seleccionamos posición en la memoria RAM (GPR)
para guardar Numero de Tecla presionada

;**** Inicio del Micro ****
Reset
        org     0x00                    ; Aquí comienza el micro.-
        goto    Inicio                  ; Salto a inicio de mi programa.-
;**** Vector de Interrupción ****
        org     0x04                    ; Atiendo Interrupción.-
        goto    ISR
;**** Programa principal ****
;**** Configuración de puertos ****
        org     0x05
Inicio
        movlw 0x07                      ; Deshabilitamos Comparadores analógicos
movwf CMCON
        bsf     STATUS,RP0                      ; Pasamos de Banco 0 a Banco 1.-
        clrf    TRISA                   ; PORTA como Salida.-
        movlw b'11110000'               ; Nible bajo como Salida y Nible alto como Entrada.-
        movwf TRISB
        bcf     OPTION_REG,RBPU         ; Habilitamos resistencias Pull-Up.
        bcf     STATUS,RP0                      ; Paso del Banco 1 al Banco 0
        clrf    PORTB                   ; El puerto quedará 11110000.-
        clrf    PORTA
        bcf     INTCON,RBIF                     ; Borramos bandera de Interrupción.-
        movlw b'10001000'               ; Habilitamos GIE y RBIE (interrupción RB4 a RB7)
        movwf INTCON
        clrf    NTecla
;**** Bucle ****
Bucle
        nop
        goto    Bucle
;**** Rutina de servicio de Interrupción ****
;**** Interrupción por TMR0 ****
ISR
        btfss   INTCON,RBIF                     ; Consultamos si es por RB4 a RB7.-
        retfie                          ; No, entonces salimos de interrupción.-
        call    Tecla_Presionada        ; Se detecta que tecla fue presionada
        movfw NTecla                            ; Tecla_Presionada la devuelve en NTecla.-
        movwf PORTA                             ; Mostramos en display tecla Presionada.-
        clrf    PORTB                           ; Dejamos Puerto para recibir otra tecla.-
        bcf     INTCON,RBIF                     ; borramos bandera.-
        retfie                          ; Salimos de interrupción.-
;**** Rutinas *****
```

```
; Rastreamos Tecla presionada.-
Tecla_Presionada
        clrf    NTecla          ; Borra Numero de Tecla y…
        incf    NTecla,1                ; …prepara NTecla para primer código.
        movlw   b'00001110'             ; Saca 0 a la primera fila…
        movwf   PORTB                   ; …de la Puerta B
        nop                             ; Para estabilización de señal.
Test_Columnas
        btfss   PORTB,4                 ; Primera columna = 0?
        goto    Suelta_tecla            ; Sale si se ha pulsado tecla.
        incf    NTecla,1                ; No es , incrementa nº tecla.
        btfss   PORTB,5                 ; Segunda columna = 0?
        goto    Suelta_tecla            ; Sale si se ha pulsado tecla.
        incf    NTecla,1        ; No es, incrementa nº tecla.
        btfss   PORTB,6                 ; Tercera columna = 0?
        goto    Suelta_tecla            ; Sale si se ha pulsado tecla.
        incf    NTecla,1        ; No es, incrementa nº tecla.
;       En este caso no se Usa teclado 3x4.-
;       btfss   PORTB,7                 ; Cuarta columna = 0
;       goto    Suelta_tecla            ; Sale si se ha pulsado tecla.
;       incf    NTecla,1        ; No es, incrementa nº Tecla.

Ultima_tecla
        btfss   PORTB,3     ; Ya se revisaron todas las filas?
        goto    Null_tecla      ; Si, Falsa alarma, no se ha presionado ninguna.-
        bsf     STATUS,C                ; No, seguimos con la siguiente. Pone a 1 Bit C…
        rlf     PORTB,1     ; …así la Fila 1 pasa a 1 con la rotación a izqda.
        goto    Test_Columnas       ; Seguimos testeando.

Null_tecla
        clrf    NTecla          ; Coloca variable Tecla a 0 (Ninguna)
        return                  ; regresa.

Suelta_tecla
 ; Ahora se espera a que la tecla sea soltada para evitar rebotes y reactivaciones de tecla.
Espera1
        btfss   PORTB,4                 ; Si no se suelta la tecla FILA 1…
        goto    Espera1         ; …vuelve a esperar.
Espera2
        btfss   PORTB,5                 ; Si no se suelta la tecla FILA 2…
        goto    Espera2                 ; …vuelve a esperar.
Espera3
        btfss   PORTB,6                 ; Si no se suelta la tecla FILA 3…
        goto    Espera3         ; …vuelve a esperar.
Espera4
        btfss   PORTB,7                 ; Si no se suelta la tecla FILA 4…
        goto    Espera4                 ; …vuelve a esperar.
        return                  ; vuelve al programa que hizo la llamada.
;.....................................
        end
```



¿Buscás LEDs?

D+ LED

Todos los LEDs, un solo lugar.

www.dled.com.ar

Soler 4949 Timbre "1" | Capital Federal | Argentina | Teléfono +54 11 4770 9366 y líneas rotativas.

librería de gráficos para GLCD en C

Cuando necesitamos utilizar un display LCD gráfico (GLCD) nos encontramos que no siempre tenemos a mano las funciones necesarias para dibujar formas o texto sobre él. Algunos compiladores disponen de librerias al efecto, pero muchas veces tienen restricciones legales sobre el código generado. Hoy te mostramos como escribir tus propias rutinas, y a adaptarlas a tus necesidades.

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

"Debido a las restricciones legales que impone CCS hemos decidido crear nuestra propia librería para el manejo de GLCDs"

El compilador CCS proporciona una librería capaz de dibujar primitivas sobre varios modelos de displays LCD gráficos o GLCD (por Graphic Liquid Cristal Display). Hay versiones de esta librería para pantallas con diferentes controladores embebidos, como el Samsung KS0108 o el Toshiba T6963.

Pero a pesar de que pueden distribuirse libremente los trabajos que hagamos con ellas, no pueden compartirse los programas que las contengan, a menos que la persona que los recibe también sea un usuario registrado de CCS. Esto limita mucho su uso con fines educativos. De hecho, si quisiésemos publicar en la revista uControl un programa que trafique algo en un GLCD, estaríamos violando la licencia, ya que es muy posible que muchos de los nuestros lectores no hayan comprado el compilador.

Es por ello que nos hemos decidido a escribir una librería propia, que usaremos de ahora en más para nuestros proyectos. Las funciones que posee esta librería son las siguientes:

- *GLCD_limpiar(color)*
- *GLCD_inicializa(modo)*
- *GLCD_punto(x, y, color)*
- *GLCD_linea(x1, y1, x2, y2, color)*
- *GLCD_rectangulo(x1, y1, x2, y2, color)*
- *GLCD_caja(x1, y1, x2, y2, color)*
- *GLCD_circulo(x1, y1, radio, color)*

Se asume que las conexiones entre el PIC y el GLCD son las siguientes:

PIN	LCD	PIC
1	CS1	B1
2	CS1	B0
6	R/S	B2
7	R/W	B3
8	E	B4
9...16	D0...D7	D0...D7
17	RESET	B5

Por supuesto, los pines utilizados pueden cambiarse modificando las sentencias DEFINE que se encuentran al principio de la librería.

```
//Pines a usar
#define GLCD_CS1
PIN_E2
#define GLCD_CS2
PIN_E1
#define GLCD_DI
PIN_C3
#define GLCD_RW
PIN_C2
#define GLCD_E
PIN_C1
#define GLCD_RESET
PIN_E0
```

Comencemos a analizar cada una de las funciones implementadas:

GLCD_inicializa(modo)
Esta es la primer función de la librería que debe llamar nuestro programa. Se encarga de inicializar el GLCD, y el parámetro "modo" determina si estará encendido (si recibe un "1") o apagado (si recibe un "0"). Es importante saber que esta función además de inicializar el GLCD borra la pantalla.

```
//-----------------------------------------------------------------------
// GLCD_inicializa(modo) - Esta función inicializa el LCD.
//-----------------------------------------------------------------------
void GLCD_inicializa(int1 modo)
{
  // Pone los pines de control en el estado correcto.
  output_high(GLCD_RESET);
  output_low(GLCD_E);
  output_low(GLCD_CS1);
  output_low(GLCD_CS2);
  output_low(GLCD_DI);   // Modo instrucción

  // Envio datos de inicialización ----------------------
  GLCD_enviaBYTE(GLCD_lado_CS1, 0xC0);
  GLCD_enviaBYTE(GLCD_lado_CS2, 0xC0);
  GLCD_enviaBYTE(GLCD_lado_CS1, 0x40);
  GLCD_enviaBYTE(GLCD_lado_CS2, 0x40);
  GLCD_enviaBYTE(GLCD_lado_CS1, 0xB8);
  GLCD_enviaBYTE(GLCD_lado_CS2, 0xB8);

  // Si modo = 1 inicializa encendido. Sino, apagado.
  if(modo == 1)
  {  GLCD_enviaBYTE(GLCD_lado_CS1,  0x3F);
 // Enciendo el GLCD
     GLCD_enviaBYTE(GLCD_lado_CS2, 0x3F); }
  else {
     GLCD_enviaBYTE(GLCD_lado_CS1,  0x3E);
  // Apago el GLCD
     GLCD_enviaBYTE(GLCD_lado_CS2, 0x3E); }

  // Borro la pantalla
  GLCD_limpiar(0);
}
```

GLCD_limpiar(color)
Esta es la función que "pinta" toda la pantalla con uno u otro color. Si recibe como parámetro un "1", la pintará completamente de negro. Si recibe un "0", la limpiará por completo. Su funcionamiento también es muy sencillo, y se "apoya" en GLCD_envia BYTE() para escribir en el GLCD. Recorre ambas mitades del GLCD, página por página, de arriba hacia abajo, escribiendo "0x00" o "0xFF" según se haya elegido pintar o borrar.

"Con tan solo 7 funciones podremos mostrar gráficos en nuestros displays GLCDs"

```
//-------------------------------------------------------------------------
// GLCD_limpiar(color) - Limpia el GLCD (pinta la pantalla)
//-------------------------------------------------------------------------
void GLCD_limpiar(int1 color)
{
  int8 i, j;
  // Recorre las 8 paginas (vertical)
  for(i = 0; i < 8; ++i)
  {
    output_low(GLCD_DI);   // Modo instrucción

    //Comienzo, en cada página, desde la dirección 0
    GLCD_enviaBYTE(GLCD_lado_CS1, 0b01000000);
    GLCD_enviaBYTE(GLCD_lado_CS2, 0b01000000);

    //Selecciono la direccion dentro de la pagina
    GLCD_enviaBYTE(GLCD_lado_CS1, i | 0b10111000);
    GLCD_enviaBYTE(GLCD_lado_CS2, i | 0b10111000);

    output_high(GLCD_DI);   // Modo datos

    // Recorre las dos mitades (horizontales)
    for(j = 0; j < 64; ++j)
    {  GLCD_enviaBYTE(GLCD_lado_CS1, 0xFF * color);
   // Enciende/apaga píxeles
       GLCD_enviaBYTE(GLCD_lado_CS2, 0xFF * color);
   // Enciende/apaga píxeles
    }
  }
}
```

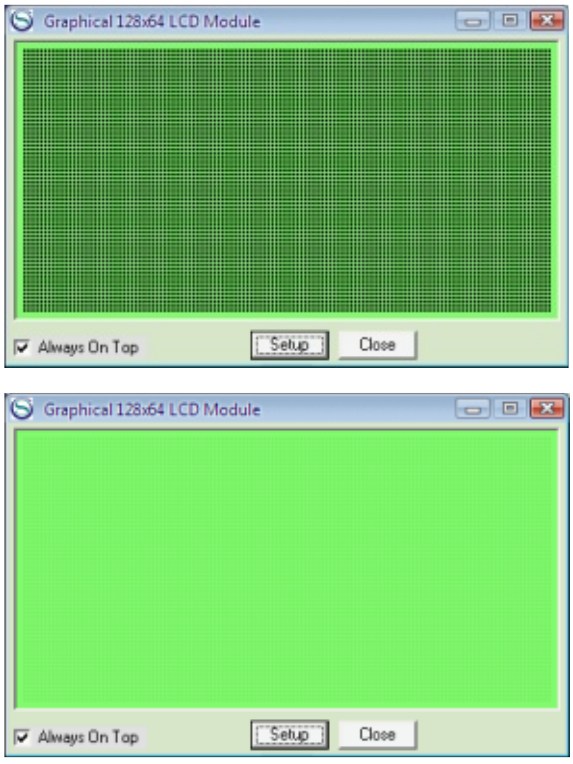
GLCD_punto(x, y, color)
Esta es la "primitiva gráfica" indispensable. A partir de GLCD_punto(x, y, color) escribiremos todas las funciones restantes. Los parámetros que recibe GLCD_punto(x, y, color) son:

x: un byte, es la coordenada "x" (horizontal), con valores válidos de 0 a 127 (izquierda a derecha).

y: un byte, es la coordenada "y" (vertical), con valores válidos de 0 a 63 (arriba a abajo).

color: un bit, "0" = apagado, "1" = encendido.

Abajo a la derecha: ejemplo de uso de la función GLCD_limpiar(color).
Abajo a la izquierda: efecto observado en el display.




```
#include <16F877A.h>
#device *=16
#include <stdlib.h>
#fuses  HS,NOWDT,NOLVP

#USE DELAY (CLOCK=20000000)
#include <GLCD_K0108.C>

//-------Comienza el programa ----------------------
void main()
{
    //Inicializo el GLCD, encendido.
    GLCD_inicializa(1);

    GLCD_limpiar(1);  //Lo pinto completamente
de negro
    delay_ms(2000);   //Espero dos segundos y...
    GLCD_limpiar(0);  //...lo limpio.
}
```

GLCD_linea(x1, y1, x2, y2, color)
La linea también resulta indispensable a la hora de dibujar un gráfico. Los parámetros que recibe GLCD_linea-(x1, y1, x2, y2, color) son:

x1: un byte, es la coordenada "x" (horizontal) del primer extremo de la linea, con valores válidos de 0 a 127 (izquierda a derecha).
y1: un byte, es la coordenada "y" (vertical) del primer extremo de la linea, con valores válidos de 0 a 63 (arriba a abajo).
x2: un byte, es la coordenada "x" (horizontal) del segundo extremo de la linea, con valores válidos de 0 a 127 (izquierda a derecha).
y2: un byte, es la coordenada "y" (vertical) del segundo extremo de la linea, con valores válidos de 0 a 63 (arriba a abajo).
color: un bit, "0" = linea en blanco, "1" = linea en negro.

El trazado de lineas se basa en el Algoritmo de Bresenham.

```
//-----------------------------------------------------------------------
// GLCD_punto(x, y, color) - Dibuja un pixel
//-----------------------------------------------------------------------
void GLCD_punto(int8 x, int8 y, int1 color)
{
  BYTE dato;
  int1 lado = GLCD_lado_CS1;  // Lado en que voy a dibujar.

  if(x > 63)                  // Veo si cae del otro lado
  { x -= 64;
    lado = GLCD_lado_CS2;}

  output_low(GLCD_DI);    // Modo instrucción
  bit_clear(x,7);         // Limpio bit MSB...
  bit_set(x,6);           // ...y pongo el bit 6 en 1
  GLCD_enviaBYTE(lado, x);
  // Envio la dirección de la coordenada X

  // Calculo en que pagina de las 8 cae...
  GLCD_enviaBYTE(lado, (y/8 & 0xBF) | 0xB8);
  output_high(GLCD_DI);     // ...y paso a Modo datos

  // Se necesitan dos lecturas para que devuelva el dato en la
 // nueva dirección
  GLCD_leeBYTE(lado);
  dato = GLCD_leeBYTE(lado);

  // De acuerdo al valor de color...
  if(color == 1) bit_set(dato, y%8);    // Enciendo el píxel
       else      bit_clear(dato, y%8);  // apago el píxel

  output_low(GLCD_DI);          // Modo instrucción
  GLCD_enviaBYTE(lado, x);      // Fijo el lado a escribir,
  output_high(GLCD_DI);         // pongo en Modo Datos y....
  GLCD_enviaBYTE(lado, dato);   // dibujo el píxel
}
```

El trazado de lineas se basa en el Algoritmo de Bresenham.

Ejemplo de uso de la función GLCD_punto(x, y, color).

```
#include <16F877A.h>
#device *=16
#include <stdlib.h>
#fuses  HS,NOWDT,NOLVP

#USE DELAY (CLOCK=20000000)
#include <GLCD_K0108.C>

//-------Comienza el programa ------------------
void main()
{
    int i,j;
    GLCD_inicializa(1);     //Inicializo el
GLCD, encendido.
    GLCD_limpiar(0);        //Limpio la pantalla

    // Pinto puntos en el GLCD
    for(i=0; i<=127; i=i+8){
       for(j=0; j<=63; j=j+8){
          GLCD_punto(i,j,1); //Pinto el punto
       }
    }
}
```

GLCD_rectangulo(x1, y1, x2, y2, color)
Los rectángulos de dibujan (internamente) mediante cuatro llamadas a la función GLCD_linea. Los parámetros que recibe GLCD_rectangulo(x1, y1, x2, y2, color) son:

x1: un byte, es la coordenada "x" (horizontal) de la esquina superior izquierda del rectángulo, con valores válidos de 0 a 127 (izquierda a derecha).
y1: un byte, es la coordenada "y" (vertical) de la esquina superior izquierda del rectángulo, con valores válidos de 0 a 63 (arriba a abajo).
x2: un byte, es la coordenada "x" (horizontal) de la esquina inferior derecha del rectángulo, con valores válidos de 0 a 127 (izquierda a derecha).
y2: un byte, es la coordenada "y" (vertical) de la esquina inferior derecha del rectángulo, con valores válidos de 0 a 63 (arriba a abajo).
color: un bit, "0" = rectángulo en blanco, "1" = rectángulo en negro.

GLCD_caja(x1, y1, x2, y2, color)
Las "cajas" son rectángulos pintados en su interior con el mismo color que el borde exterior. También se dibujan (internamente) mediante llamadas a la función GLCD_linea. Los parámetros que recibe GLCD_caja(x1, y1, x2, y2, color) son:

x1: un byte, es la coordenada "x" (horizontal) de la es-

```
//-------------------------------------------------------------------------
// Dibuja una linea desde (x1,y1) a (x2,y2) de color (0 o 1)
//-------------------------------------------------------------------------
void GLCD_linea(int x1, int y1, int x2, int y2, int1 color)
{
  //Declaro variables-------------------
  signed int  x, y, incremento_x, incremento_y, distancia_x,
distancia_y;
  signed long P;
  int i;

  //Calculo las diferencias entre las coordenadas de origen y
destino
  distancia_x = abs((signed int)(x2 - x1));
  distancia_y = abs((signed int)(y2 - y1));

  //Inicializo x e y con las coordenadas de origen
  x = x1;
  y = y1;

  //Calculo el sentido de los incrementos (positivos o
negativos)
  //en función de la posición del origen y el destino
  if(x1 > x2) incremento_x = -1; else incremento_x = 1;
  if(y1 > y2) incremento_y = -1; else incremento_y = 1;

  //Si la distancia horizontal es mayor a la vertical...
  if(distancia_x >= distancia_y)
  { P = 2 * distancia_y - distancia_x;
    for(i=0; i<=distancia_x; ++i)
    {
      GLCD_punto(x, y, color);

      if(P < 0)
      { P += 2 * distancia_y;
        x += incremento_x; }
      else
      { P += 2*distancia_y - 2*distancia_x;
        x += incremento_x;
        y += incremento_y;}
    }
  }

  //Si la distancia vertical es mayor a la horizontal...
  else
  { P = 2 * distancia_x - distancia_y;
    for(i=0; i<=distancia_y; ++i)
    { GLCD_punto(x, y, color);
      if(P < 0)
      {  P += 2 * distancia_x;
```

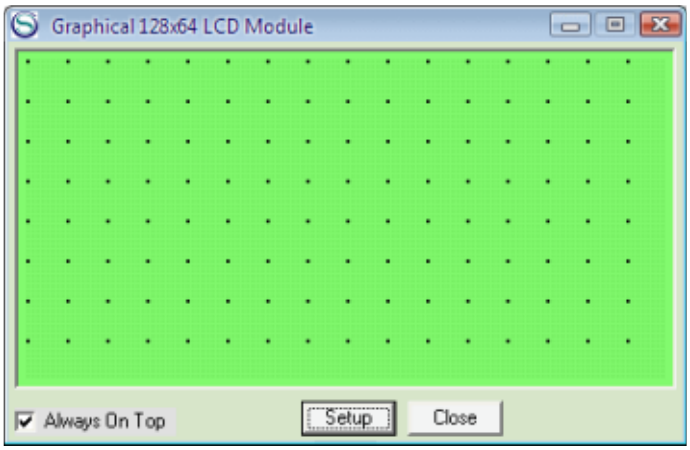
quina superior izquierda del rectángulo, con valores válidos de 0 a 127 (izquierda a derecha).
y1: un byte, es la coordenada "y" (vertical) de la esquina superior izquierda del rectángulo, con valores válidos de 0 a 63 (arriba a abajo).
x2: un byte, es la coordenada "x" (horizontal) de la esquina inferior derecha del rectángulo, con valores válidos de 0 a 127 (izquierda a derecha).
y2: un byte, es la coordenada "y" (vertical) de la esquina inferior derecha del rectángulo, con valores válidos de 0 a 63 (arriba a abajo).
color: un bit, "0" = caja en blanco, "1" = caja en negro.

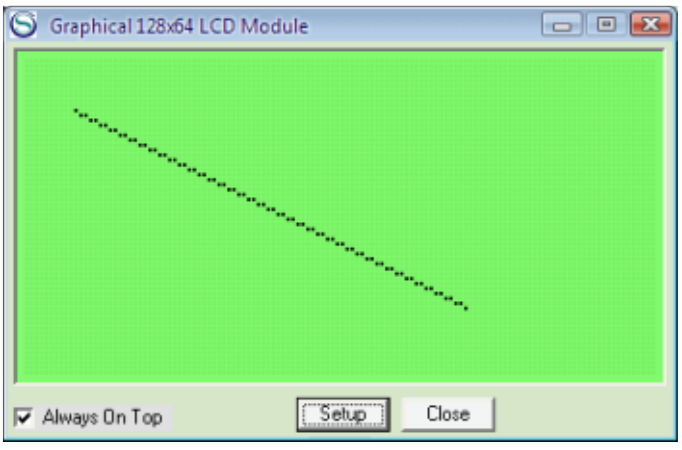
GLCD_circulo(x1, y1, radio, color)
Esta es la función que dibuja un circulo. El interior del circulo permanece del color del fondo. Estrictamente hablando, se dibuja solo la circunferencia. Los parámetros que recibe GLCD_circulo(x1, y1, radio, color) son:

x1: un byte, es la coordenada "x" (horizontal) del centro del circulo, con valores váli-

```
   y += incremento_y; }
       else
       {  P += 2 * distancia_x - 2 * distancia_y;
          x += incremento_x;
          y += incremento_y; }
     }
  }
}
```

```
// Ejemplo de uso de GLCD_linea(x1, y1, x2, y2, color)
#include <16F877A.h>
#device *=16
#include <stdlib.h>
#fuses  HS,NOWDT,NOLVP

#USE DELAY (CLOCK=20000000)
#include <GLCD_K0108.C>

//-------Comienza el programa ----------------------
void main()
{
    int i;

    GLCD_inicializa(1);     //Inicializo el GLCD, encendido.
    GLCD_limpiar(0);        //Limpio la pantalla

    // Dibujo eje "X"
    GLCD_linea(0, 32, 127, 32, 1);
    for(i=0; i<=127; i=i+8){ GLCD_linea(i,31,i,33,1); }

    // Dibujo eje "Y"
    GLCD_linea(64, 0, 64, 64, 1);
    for(i=0; i<=63; i=i+8){ GLCD_linea(63,i,65,i,1); }

    // Dibujo la "grafica"
    GLCD_linea(0,63,127,0,1);
}
```



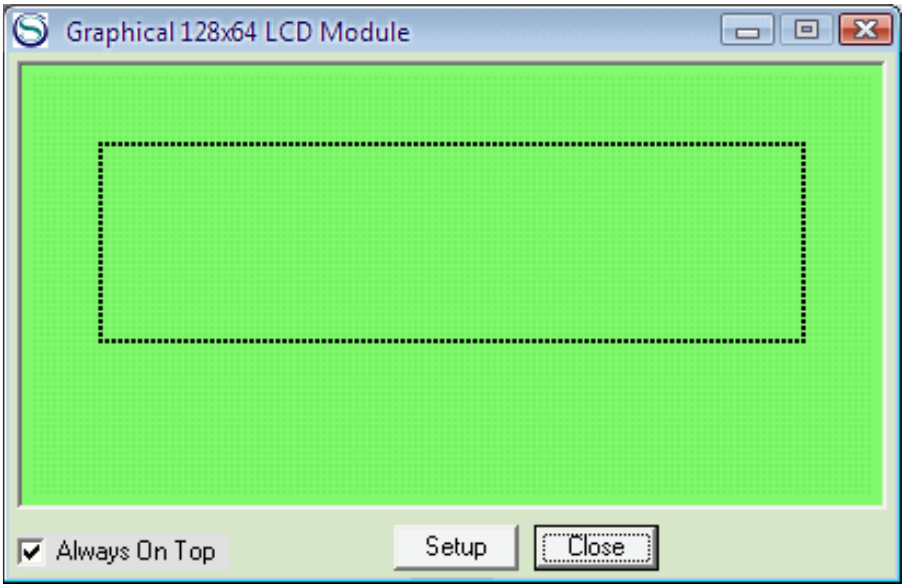


dos de 0 a 127 (izquierda a derecha).
y1: un byte, es la coordenada "y" (vertical) del centro del circulo, con valores válidos de 0 a 63 (arriba a abajo).
radio: un byte, es el radio de la circunferencia (en pixeles).
color: un bit, "0" = circulo en blanco, "1" = circulo en negro.

Las circunferencias se han resuelto mediante el "algoritmo del punto medio", que divide la circunferencia en 8 partes simétricas, evitando utilizar funciones como seno, coseno o potencias, que volverían muy lenta la tarea del trazado.

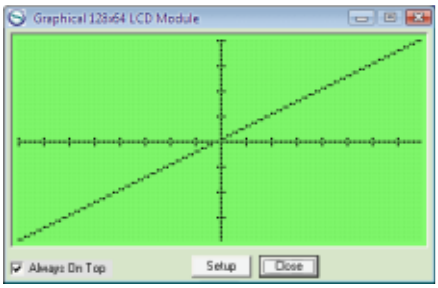
```
//-------------------------------------------------------------------------
// Dibuja un rectángulo desde (x1,y1) a (x2,y2)
// de color (0 ó 1).
//-------------------------------------------------------------------------
void GLCD_rectangulo(int x1, int y1, int x2, int y2, int1 color)
{
  GLCD_linea(x1,y1,x2,y1,color);
  GLCD_linea(x1,y1,x1,y2,color);
  GLCD_linea(x1,y2,x2,y2,color);
  GLCD_linea(x2,y1,x2,y2,color);
}
```

```
// Ejemplo de GLCD_rectangulo(x1, y1, x2, y2, color)

#include <16F877A.h>
#device *=16
#include <stdlib.h>
#fuses  HS,NOWDT,NOLVP

#USE DELAY (CLOCK=20000000)
#include <GLCD_K0108.C>

//-------Comienza el programa ----------------------
void main()
{
    int i;

    GLCD_inicializa(1);     //Inicializo el GLCD, encendido.
    GLCD_limpiar(0);        //Limpio la pantalla

    // Dibujo un rectángulo
    GLCD_rectangulo(10,10,117,40,1);
}
```

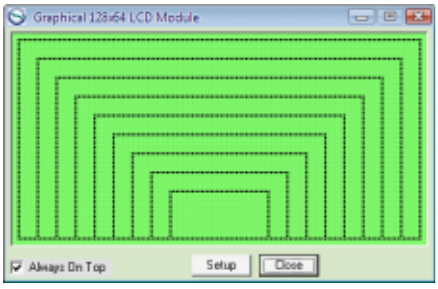


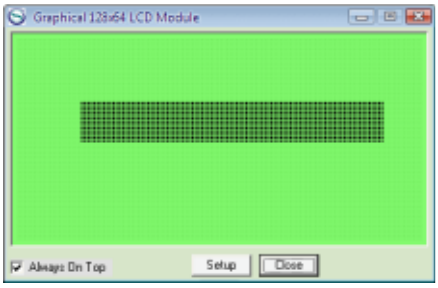
"Para evitar el uso de funciones trigonométricas que restan rendimiento al sistema se utilizo el algoritmo del punto medio a la hora de dibujar un circulo"

```
//-----------------------------------------------------------------------
// Dibuja un rectángulo PINTADO desde (x1,y1) a (x2,y2)
// de color (0 o 1)
//-----------------------------------------------------------------------
void GLCD_caja(int x1, int y1, int x2, int y2, int1 color)
{
  //Declaro variables-------------------
  int i;

  for(i=y1;i<=y2;i++) {
    GLCD_linea(x1,i,x2,i,color); }
}
```

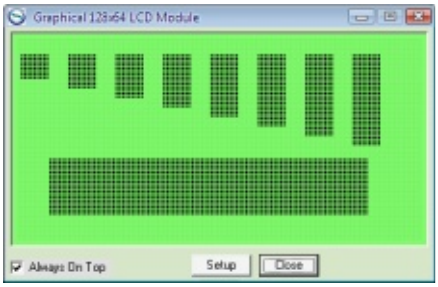


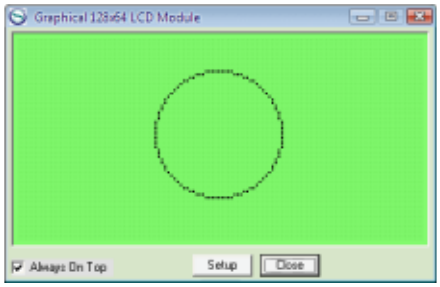

```
// Dibujando una caja
#include <16F877A.h>
#device *=16
#include <stdlib.h>
#fuses  HS,NOWDT,NOLVP

#USE DELAY (CLOCK=20000000)
#include <GLCD_K0108.C>

//-------Comienza el programa ----------------------
void main()
{
    int i;

    GLCD_inicializa(1);     //Inicializo el GLCD, encendido.
    GLCD_limpiar(0);        //Limpio la pantalla

    // Dibujo una caja
    GLCD_caja(20,20,115,32,1);
}
```



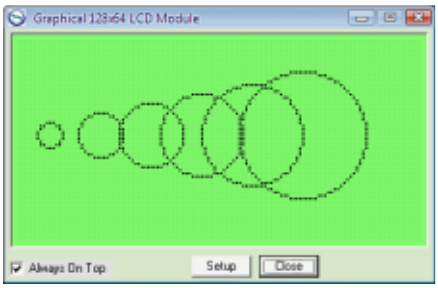


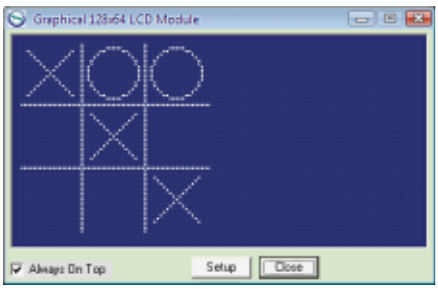

```
//-----------------------------------------------------------------------
// Dibuja un circulo con centro en (x1,y1), radio
// y color (0 ó 1)
//-----------------------------------------------------------------------
void GLCD_circulo(int x1, int y1, int radio, int1 color)
 {
   signed int d, x, y;

   //Inicializo las variables.
   d = 1 -  radio;
   x = 0;
   y = radio;

   // Dibujo los cuatro píxeles que "caen" sobre los ejes
   // cartesianos.
   GLCD_punto(x1, y1 + radio, color);
```



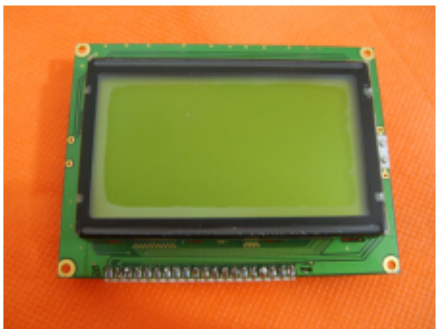


```
   GLCD_punto(x1, y1 - radio, color);
   GLCD_punto(x1 + radio, y1, color);
   GLCD_punto(x1 - radio, y1, color);

   //Este es el bucle que pinta los octavos de la circunferencia.
   while(x < y) {
     if(d < 0) {d = d + 2 * x + 3;}
          else  {d = d + 2 * (x - y ) + 5;
                 y = y - 1 ;}
     x = x + 1;

     //Pone el punto en cada uno de los "octantes".
     GLCD_punto(x1 + x, y1 + y, color);
     GLCD_punto(x1 - x, y1 + y, color);
     GLCD_punto(x1 + x, y1 - y, color);
     GLCD_punto(x1 - x, y1 - y, color);
     GLCD_punto(x1 + y, y1 + x, color);
     GLCD_punto(x1 - y, y1 + x, color);
     GLCD_punto(x1 + y, y1 - x, color);
     GLCD_punto(x1 - y, y1 - x, color);
     }
 }
```






```
// Dibujando un circulo
#include <16F877A.h>
#device *=16
#include <stdlib.h>
#fuses  HS,NOWDT,NOLVP

#USE DELAY (CLOCK=20000000)
#include <GLCD_K0108.C>

//-------Comienza el programa ----------------------
void main()
{
    int i;

    GLCD_inicializa(1);     //Inicializo el GLCD, encendido.
    GLCD_limpiar(0);        //Limpio la pantalla

    // Dibujo varios circulos
    for(i=10; i<=100; i=i+16){
       GLCD_circulo(i,30,i/5+2,1); }
}
```



Hay dos funciones accesorias en las que se basa el funcionamiento de toda la librería. Efectivamente, de forma indirecta o indirecta todas las funciones implementadas hacen uso de GLCD_leeBYTE(int1 lado) y GLCD_enviaBYTE(int1 lado, BYTE dato) para leer o escribii bytes en el display. A continuación, el código fuente de cada una.

Por último puedes descargar la libreria GLCD_K0108.C haciendo click aquí.

```
//-----------------------------------------------------------------------
// Escribe un byte en una de las mitades de la pantalla
// (lado=0:izq Lado=1:der)
//-----------------------------------------------------------------------
void GLCD_enviaBYTE(int1 lado, BYTE dato)
{
  if(lado) output_high(GLCD_CS2);  // Selecciono la mitad
correspondiente
     else  output_high(GLCD_CS1);

  output_low(GLCD_RW);       // Modo escritura
  output_d(dato);            // Coloco el dato en el puerto y...
  delay_us(1);          // ...espero.
  output_high(GLCD_E);       // Pongo el bit Enable en alto y...
  delay_us(2);          // ...espero.
  output_low(GLCD_E);        // Pongo el bit Enable en bajo.

  output_low(GLCD_CS1);      // Libero la linea CS1 y...
  output_low(GLCD_CS2);      // CS2.
}
```

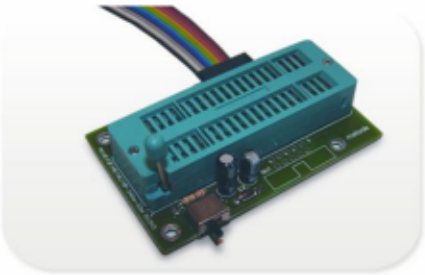
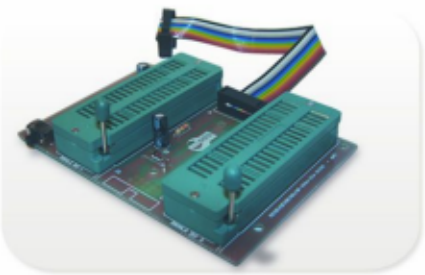
```
//-----------------------------------------------------------------------
// Lee un byte de una de las dos mitades de la pantalla
//-----------------------------------------------------------------------

BYTE GLCD_leeBYTE(int1 lado)
{
  BYTE dato;
  set_tris_d(0xFF);                    // Puerto D como entrada
  output_high(GLCD_RW);                // GLCD en Modo lectura

                                       // Selecciono la mitad del display a leer.
  if(lado) output_high(GLCD_CS2);
     else output_high(GLCD_CS1);

  delay_us(1);                         // Espero...
  output_high(GLCD_E);                 // Pongo en alto el pin enable y...
  delay_us(2);                         // ...espero.
  dato = input_d();                    // Guardo en "dato" el valor devuelto y...
  output_low(GLCD_E);                  // ...pongo en bajo el pin enable.

                                       // Vuelvo a poner en bajo las lineas CS1 y CS2.
  output_low(GLCD_CS1);
  output_low(GLCD_CS2);

  return dato;
}
```

GTP-USB+

Grabador de Microcontroladores y Memorias por puerto USB 2.0
Hardware oficial del software Winpic800.

- **SOLO CONECTAR Y USAR.** El puerto USB provee la alimentación y el HID facilita la instalación. Ideal para uso con notebook.
- **ALTA VELOCIDAD DE TRANSFERENCIA DE DATOS.** Con USB 2.0 Full Speed (12 Mb/s), el conjunto GTP-USB+ /Winpic800, puede ser utilizado en equipos con USB 1.1.
- **AMPLIO LISTADO DE DISPOSITIVOS SOPORTADOS.** Soporta numerosos microcontroladores de Microchip (PIC, dsPIC, rfPIC), Atmel (en modo serie, y paralelo con un adaptador) y EEPROM (24Xxx, I2C y 93Xxx, Microwire).
- **IDENTIFICACION AUTOMÁTICA.** El soft Winpic800 identifica automáticamente el dispositivo conectado.
- **ACTUALIZABLE.** El firmware del GTP-USB+ se actualiza con cada nueva versión del Winpic800.
- **GRABACIÓN DIRECTA O EN CIRCUITO.** ICSP, ISP, I2C, SPI.

Módulos ZIF disponibles para todas las familias


mstools
Electronic Development
www.mstools.com.ar
ventas@mstools.com.ar
011-4764-2324 15-5158-7306

leer un teclado matricial con PIC Simulator IDE

Una de las maneras más usuales de ingresar datos a un microcontrolador es a través de un teclado matricial. En este tutorial explicaremos como realizar una rutina de lectura en el lenguaje Basic del entorno PIC Simulator IDE para utilizarla en nuestros proyectos.

// por: Lucas Martín Treser //
lmtreser@gmail.com

Un teclado matricial no es más que un arreglo de pulsadores, tanto normalmente cerrados como abiertos, dispuestos en filas y columnas y compartiendo algunos de sus pines, logrando una notable reducción de estos. Por ejemplo un teclado de 4 filas y 4 columnas, llamado generalmente 4×4, posee sólo 8 pines de conexión y 16 teclas a nuestra disposición.

Ahora bien, a la hora de implementar un teclado de este tipo en alguno de nuestros proyectos debemos realizar una rutina de lectura pues al compartir pines en su estructura interna, el microcontrolador no puede dilucidar por si solo cual es la tecla presionada.

.Funcionamiento

Nuestra rutina como ya dijimos está escrita en el lenguaje Basic de PIC Simulator IDE, aunque es fácilmente portable a otra variante del lenguaje.

Lo que debemos hacer es verificar el estado de los pulsadores uno a uno para saber si hubo un cambio de estado. Para lograrlo debemos conectar las cuatro columnas del teclado a cuatro pines de salida del PIC, y las cuatro filas a otros cuatros pines de entrada del microcontrolador.

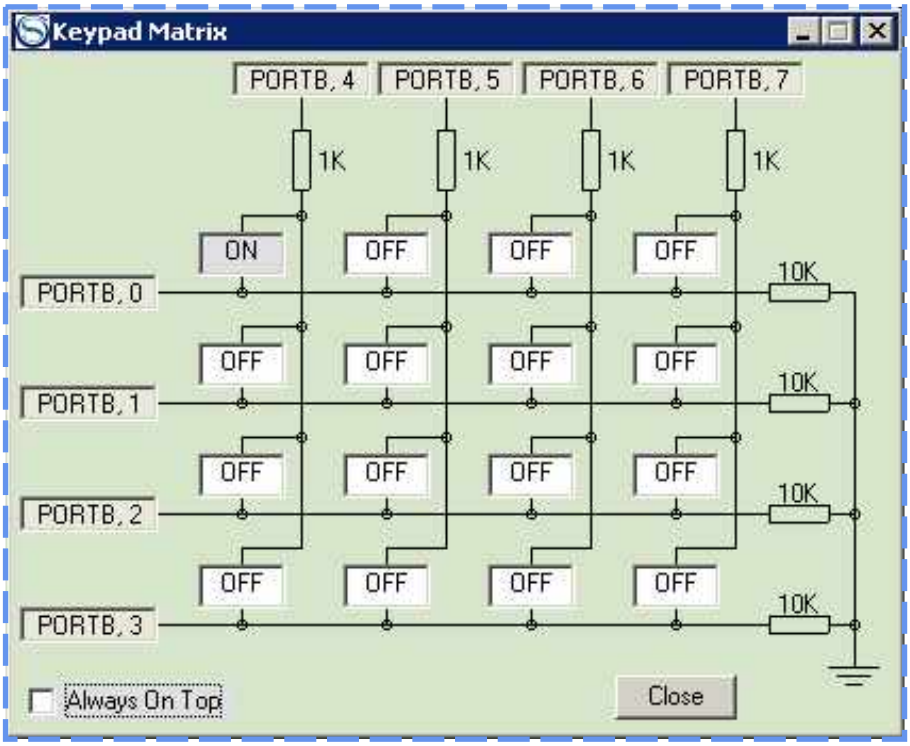
La secuencia a seguir es la siguiente:

- Activar la primera columna.
- Verificar fila a fila en busca de un uno (1) lógico.
- Apagar la primera columna.
- Repetir el procedimiento con las columnas siguientes.

```
AllDigital

'Nombro los pines que
'conforman las filas.
Symbol fila1 = RB0
Symbol fila2 = RB1
Symbol fila3 = RB2
Symbol fila4 = RB3

'Nombro los pines que
'conforman las columnas.
Symbol col1 = RB4
Symbol col2 = RB5
Symbol col3 = RB6
Symbol col4 = RB7

'Definición de puertos.
TRISA = %00000000
TRISB = %00001111

'Definición de variables.
Dim boton As Byte

'Comienza bloque principal.
loop:
   Gosub get_button
   PORTA = boton
   Goto loop
End
```

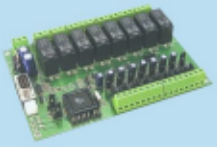
```
'Subrutina de lectura de un teclado matricial 4x4.

get_button:
boton = 0
col1 = 1                    'Activo la primera columna.
If fila1 = 1 Then boton = 1
If fila2 = 1 Then boton = 5
If fila3 = 1 Then boton = 9
If fila4 = 1 Then boton = 13
col1 = 0
col2 = 1                    'Activo la segunda columna.
If fila1 = 1 Then boton = 2
If fila2 = 1 Then boton = 6
If fila3 = 1 Then boton = 10
If fila4 = 1 Then boton = 14
col2 = 0
col3 = 1                    'Activo la tercera columna.
If fila1 = 1 Then boton = 3
If fila2 = 1 Then boton = 7
If fila3 = 1 Then boton = 11
If fila4 = 1 Then boton = 15
col3 = 0
col4 = 1                    'Activo la cuarta columna.
If fila1 = 1 Then boton = 4
If fila2 = 1 Then boton = 8
If fila3 = 1 Then boton = 12
If fila4 = 1 Then boton = 16
col4 = 0
Return                      'Fin y regreso de subrutina
```

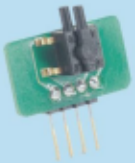
En nuestro código de ejemplo simplemente escaneamos todas las teclas, si se encuentra alguna presionada guarda un valor numérico en la variable ***"boton"*** y luego traslada el contenido de dicha variable directamente al puerto de salida PORTA. Obviamente la finalidad del código es ver la mecánica de lectura de un teclado matricial 4x4 para comprenderla, modificarla y adaptarla a un programa práctico sin mayores dificultades.



Estructura interna y conexionado del teclado.





automatismos

MAR DEL PLATA

* noticias
* artículos
* proyectos
* ideas de diseño
* recursos didácticos
* y mucho, mucho más...

http://www.automatismos-mdq.com.ar/blog

séptima competencia nacional de robótica

El Grupo de Robótica y Simulación (GRS) perteneciente al Departamento de Ingeniería Eléctrica de la Universidad Tecnológica Nacional Facultad Regional Bahía Blanca organiza nuevamente uno de los eventos más importantes del país: la "Séptima Competencia Nacional de Robotica".

// por: Grupo de Robótica y Simulación //
competenciarobotica@frbb.utn.edu.ar

La "Séptima Competencia Nacional de Robótica" se realizará el 14 de noviembre del 2009 en la ciudad de Bahia Blanca, provincia de Buenos Aires (Argentina).

"Este año la fecha elegida para la misma será el 14 de noviembre en la ciudad de Bahía Blanca"

En la categoría Sumo existen tres niveles. Una es la llamada polimodal en cual participan alumnos del nivel medio o secundario. La otra es la llamada libre donde participan alumnos universitarios, ingenieros o publico en general y por ultimo el Mini Sumo donde pueden participar todas las edades siendo los robots de un tamaño menor al Sumo Clásico.

Por tercer año consecutivo desarrollaremos la modalidad llamada velocista. La misma consiste en una carrera de dos robot seguidores de linea en un circuito cerrado. El primer robot que de la cantidad de vuelta asignada gana la carrera.

Los nuevos reglamentos, fotos y vídeos de la competencia estarán a su disposición en nuestra nueva pagina web cuya dirección es: www.grsbahiablanca.com.ar.

Con respecto a los reglamentos hemos tratado de simplificarlo al máximo dando libertad casi absoluta a los participante en la construcción de los robots y también durante la competencia pidiendo solamente que se cumpla con el peso y medidas estipuladas por la norma por lo tanto no habrá tolerancia en los mismos este año.

La inscripción al certamen cerrará el día 6 de noviembre. Aquellos participantes que tengan dudas sobre su participación en el evento se le pedirá que se inscriban igualmente ante de la fecha tope porque de otra manera no podrán competir. La inscripción sera a través de nuestra pagina.

La competencia dura un día solamente y el año pasado se presentaron 71 robots en todas las categorías siendo la distribución mas o menos la

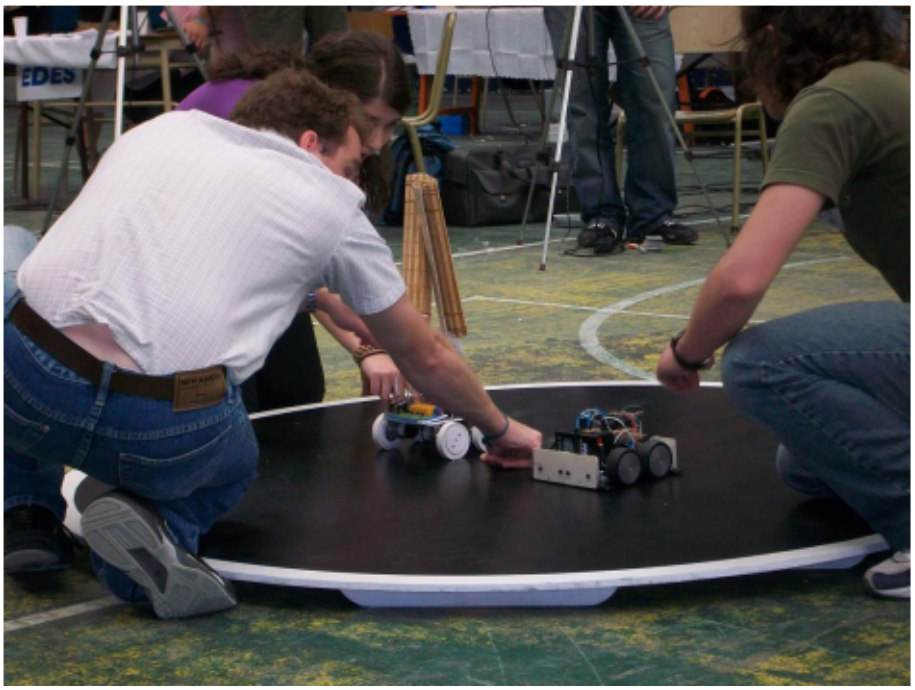
siguiente:

35 Sumo Polimodal
20 Sumo Libre
16 Velocista

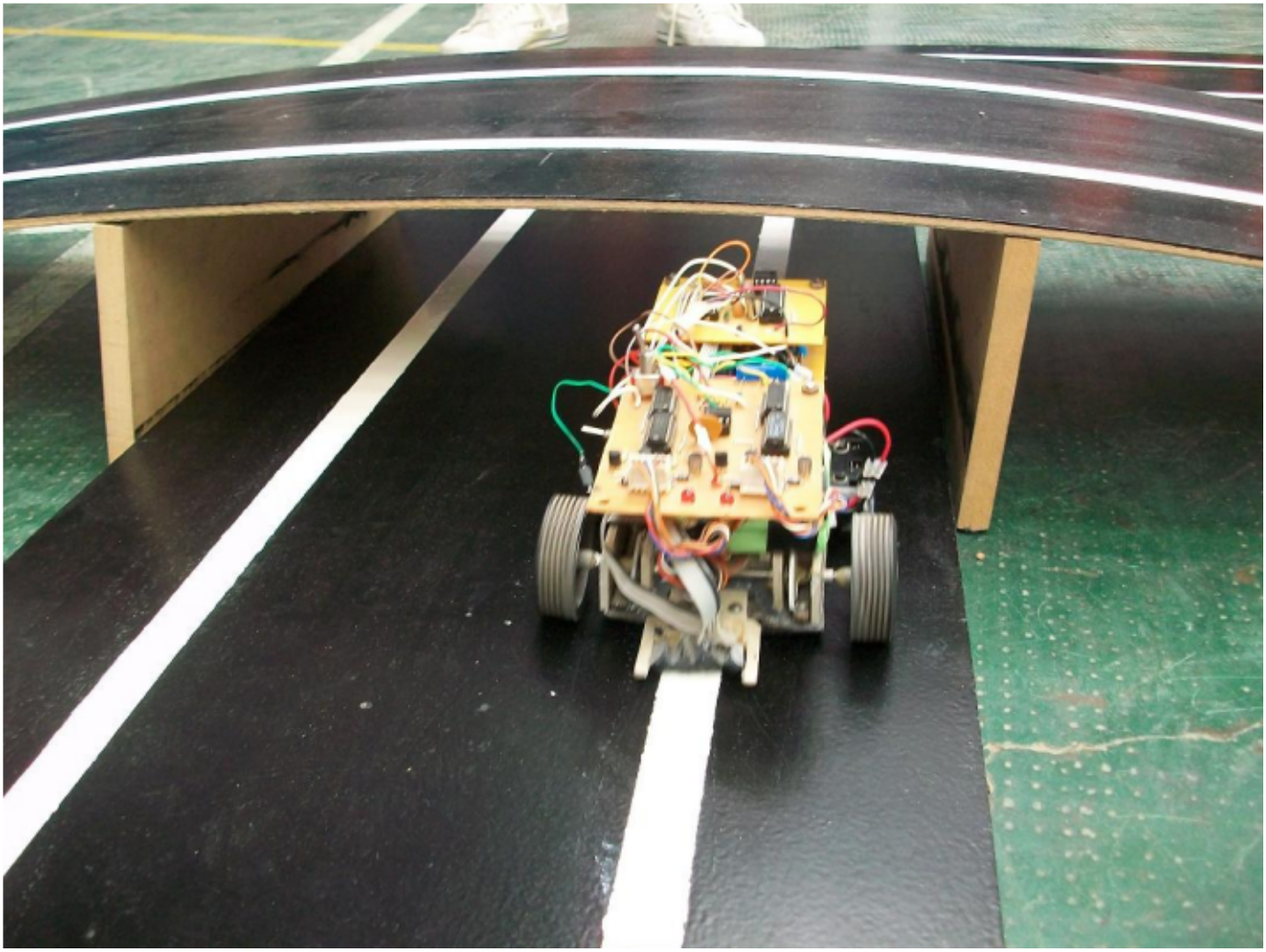
La inscripción no tiene costo hasta ahora. Lo único que se pide es pagar un bono para la cena de camaradería que se hace terminada la competencia.

El desarrollo del robot puede ser a través de una institución o en forma particular.

Cualquier inquietud que tengan no duden en escribir a los organizadores, a la siguiente dirección de correo electrónico: competenciarobotica@frbb.utn.edu.ar



Fotos de la competencia modalidad "Velocista" que se desarrollo el año pasado.




Toda la Television
el Audio y el Video
en un solo sitio web
SERVISYSTEM
Tutoriales, circuitos
reparaciones, software
tips, samples, consejos,
foros, blog, PICs ...
www.servisystem.com.ar

brújula digital de precisión con sensor de efecto hall

El físico Edwin Hall descubrió (por casualidad) el efecto que lleva su nombre en el año1879. Desde entonces han pasado muchos años y se han desarrollado múltiples sensores que aprovechan este fenómeno. Haciendo uso de ellos podremos construir una brújula digital de gran precisión.

// por: Pablo //
quickbasic@terra.es

"Utilizaremos un sensor 1321 de Allegro, que nos entrega una variación máxima entre los polos norte y sur de tan sólo 2.5mV"

El efecto Hall se manifiesta en la tensión transversal que aparece en un conductor cuando está sometido a un campo magnético. Por ejemplo, si tenemos un cable conduciendo corriente y le ponemos un voltímetro sensible entre dos puntos transversales, al acercar un imán se podría medir una pequeñísima tensión. Esto es debido a que los electrones que pasan por el cable se verán desplazados hacia un lado en presencia del imán. Entonces aparece una diferencia de tensión entre esos dos puntos del cable. Al separar el imán, la tensión transversal desaparece.

Para poder medir esta tensión transversal es necesario amplificarla, porque su valor es muy reducido. En el mercado hay sensores de efecto hall lineales que integran un amplificador y alcanzan sensibilidades de 5mV por Gauss. Esto sigue siendo muy poca cosa para nuestro propósito, ya que el campo magnético de la tierra, que es el imán que queremos medir, es tan débil como aproximadamente 0.5 Gauss.

En nuestro caso utilizaremos un sensor 1321 de la compañía Allegro, que a su salida nos entrega una variación máxima entre los polos norte y sur de tan sólo 2.5mV.

.Amplificación de la señal

Es evidente que si queremos convertir la tensión analógica que ofrece el sensor a una señal digital hay que efectuar un proceso de amplificación.

Como ya dijimos anteriormente, el sensor 1321 entrega por su pin "OUT" 2,5V cuando está en estado de reposo, es decir sin la presencia de un campo magnético en sus alrededores. Pero cuando el polo norte de un imán es acercado a él, la salida variará proporcional-

mente de 2,5V hacia la tensión de fuente y si damos vuelta el imán, enfrentándolo al polo sur, la salida bajará de 2,5V a nivel de masa.

Aprovechando esto y para aumentar la sensibilidad al doble montaremos dos sensores opuestos, uno enfrentado a otro.

Si le acercamos al montaje el polo norte del imán, un de los sensores subirá 1V por encima del nivel de reposo (2,5V) y el otro bajará 1V por debajo de dicho nivel. Por lo tanto tenemos un sensor que entrega 3,5V y el otro 1,5. Si entonces conectamos un voltímetro entre sus dos salidas "OUT" tendremos una diferencia sumada de 2V (ver figura 1).

Con este arreglo logramos multiplicar la sensibilidad por 2: ahora 1 Gauss serán 10mV. Pero como el campo magnético de la Tierra produce 5mV, todavía resulta insuficiente para atacar un conversor analógico / digital. Tendremos que amplificar esta señal mucho más.

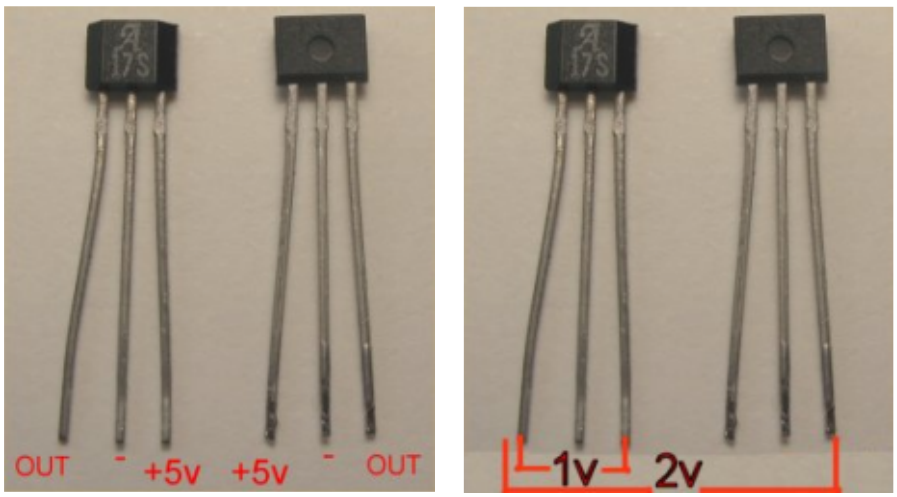


Figura 1. Pinout y encapsulado de los sensores de efecto hall.

.El amplificador de instrumentación

Para este tipo de aplicaciones los amplificadores operacionales (tipo LM-358) resultan ineficaces, introducen demasiado ruido y no son *"rail to rail"*, que quiere decir que no entregan en su salida una tensión máxima de 5V y mínima de 0, sino que su rango dinámico es bastante menor, lo que nos traerá problemas a la hora de convertir la señal de analógica a digital. Por eso usamos el amplificador de instrumentación INA122, de muy bajo ruido y "rail to rail". Este chip de precisión se suele usar para amplificar las mínimas tensiones corporales para equipos de electrocardiogramas y encefalogramas. Es muy fácil de montar ya que sólo necesita una resistencia externa para decirle que ganancia necesitamos.

Si miramos la hoja de datos encontraremos una formula para calcular la resistencia que configura la ganancia del amplificador:

$$(200.000\ /R) + 5 = \text{Ganancia}$$

Por ejemplo con una resistencia de 360 ohms se consigue una amplificación de 560 veces la tensión entrante. Por lo que los 5mV del campo magnético terrestre sacarían 2'8 voltios por el pin "OUT" del INA122. Con estas ganancias el ruido es considerable y hay que filtrarlo con capacitores para lograr algo legible. El origen de este ruido captado por los sensores hall es un tema muy interesante, pero para otra ocasión.

.La conversión analógica a digital

En un conversor A/D de 10 bits (1023 puntos de resolu-

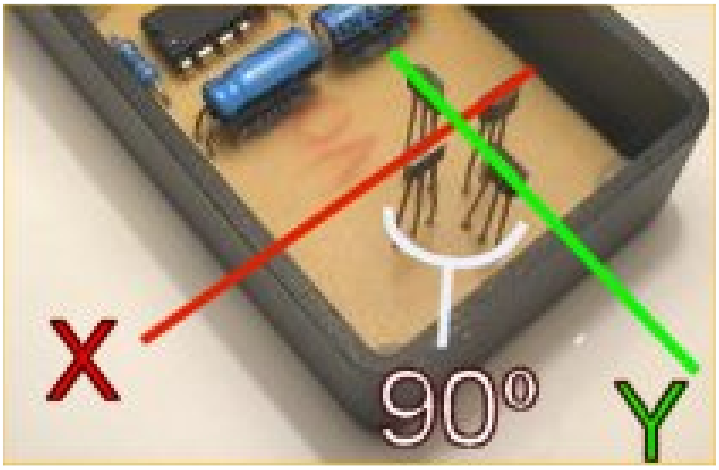


Figura 2. Disposición de los sensores hall.

ción) con una tensión de referencia de 5V, la salida de 2,8V del INA122 equivale a una variación de 572 puntos.

Al girar 360º varias veces el montaje se obtiene una gráfica como la de la figura 3. Se trata de una función SENO. Con un sensor solamente tenemos una coordenada: la X, o el seno. Tenemos que utilizar otro sensor cruzado a 90 grados para obtener el eje Y, el coseno (ver figura 2).

Ahora bien, con dos sensores cruzados y si giramos el montaje varias vueltas completas de 360 grados, veremos dos señales como las de la figura 4.

El desfase entre ambas gráficas coincide con los 90º. Interpolando ambas medidas X e Y, y girando el montaje 360 grados, obtendremos una bonita gráfica de puntos como esta la de la figura 5. Se trata de un círculo perfecto, función del seno y el coseno generados por el campo magnético terrestre. Cabe aclarar que las imperfecciones que se ven son debidas a movimientos indebidos a la hora de los ensayos con el circuito real.

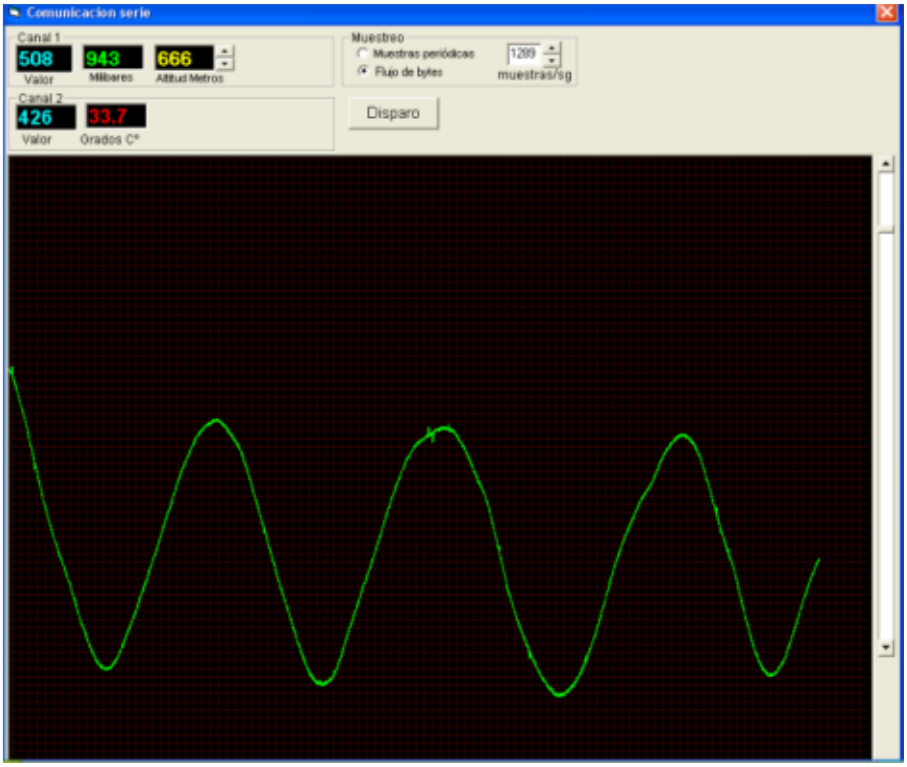


Figura 3. Función seno obtenida.

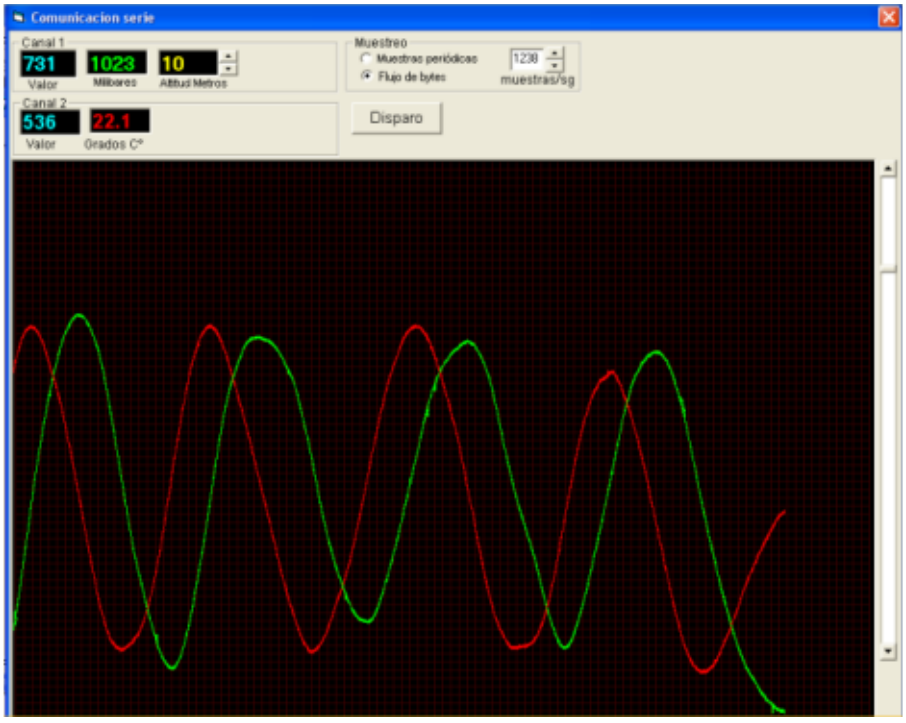


Figura 4. Desfase entre las dos señales obtenidas.

.Calculo de la definición

Las señales obtenidas por el circuito tienen una amplitud de 572 puntos, con esto podemos calcular la precisión que tendrá nuestra brújula así:

572 x 3.1416 = 1796 puntos de circunferencia total del círculo

360 grados /1796 puntos = 0,2 grados de resolución

.Ecuaciones para obtener el ángulo

Ahora unas ecuaciones con función arco-tangente para obtener de esta gráfica circular un angulo entre 0 y 360 grados:

```
' ------(volts es el eje X , volts1 es el eje Y)----

If volts = 0 And volts1 > 0 Then angulo = 270
If volts = 0 And volts1 < 0 Then angulo = 90
If volts > 0 And volts1 > 0 Then angulo = 360 - (Atn(volts1 / volts) * (180 / 3.1416))
If volts > 0 And volts1 < 0 Then angulo = -(Atn(volts1 / volts) * (180 / 3.1416))
If volts < 0 Then angulo = 180 - (Atn(volts1 / volts) * (180 / 3.1416))

print angulo

'-----------------------------------------------------------------
```

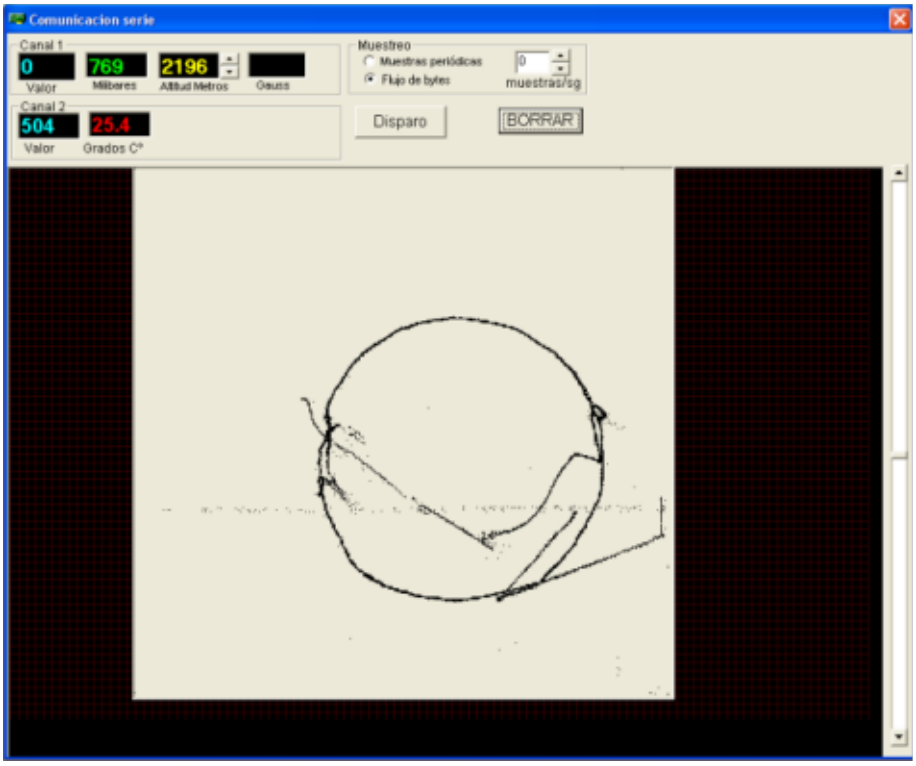


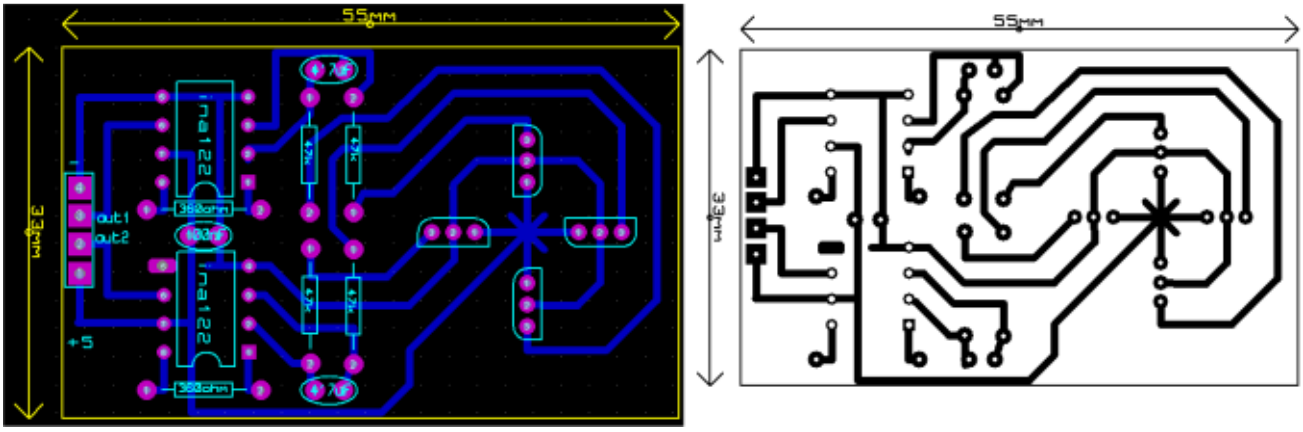
Figura 5. Circulo "perfecto" obtenido con el montaje terminado.

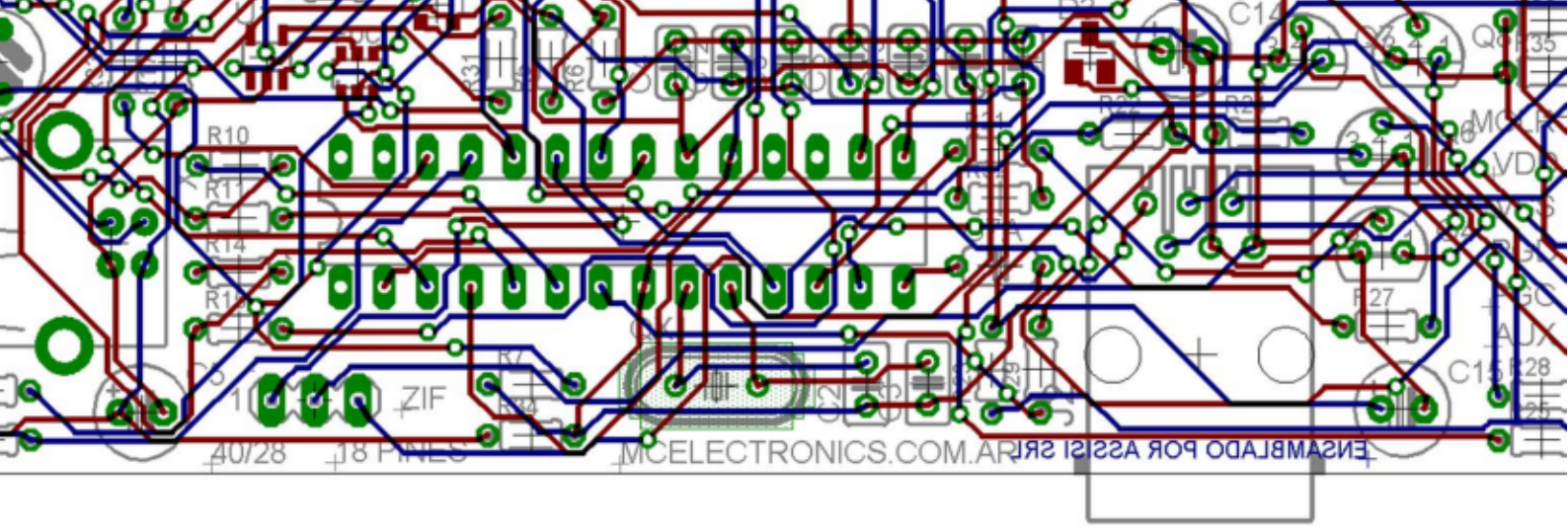
Atención: para que esto funcione, hay que ajustar el centro del "círculo" en las coordenadas 0,0. Esto puedes hacerlo simplemente restando o sumando el valor adecuado a X e Y en el programa. Con esto ya tendremos una bonita lectura digital en grados, con una precisión mayor de 1 grado. Solo resta armar alguna interface atractiva en Visual Basic o similar para usar con nuestro hardware.

.Lista de materiales

Sensor hall Allegro 1321 (4)
Circuito integrado INA122 (2)
Resistencia 47 kohms (4)
Resistencia 360 ohms (2)
Capacitor 4,7 uF (2)
Capacitor 100 nF (1)

Figura 6. Diseño del circuito impreso y ubicación de los componentes.

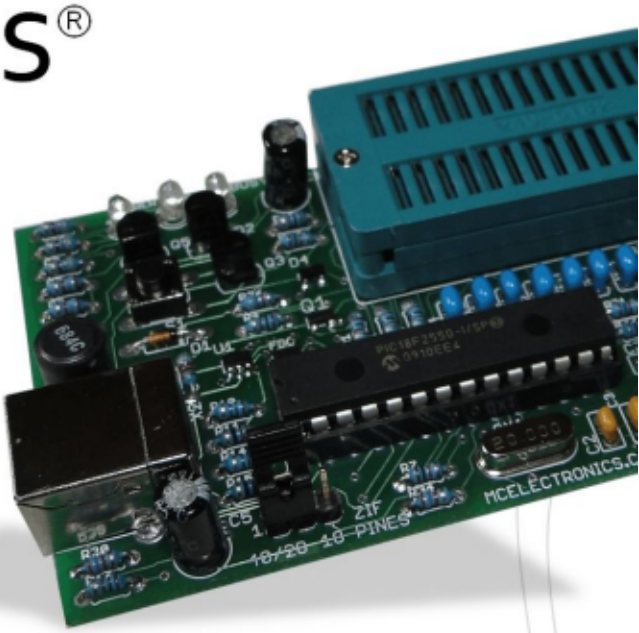




mc electronics®

Programador y Debugger Express

USB para PIC y dsPIC



El MCE PDX USB se puede utilizar como Programadador y Debugger Express. Además es un Analizador lógico de 3 canales. Incluye zócalo ZIF, conector RJ11 y EasyJack de 6 pines compatible con PICKit2 de Microchip.

$ 149,00 +IVA

USD 38,90

Incluye envío sin cargo a todo el país !

MCELECTRONICS
Austria 1760 - OF 8
Ciudad de Buenos Aires (1425)
BA. Argentina.

(011) 6091-4922/4581
www.mcelectronics.com.ar
info@mcelectronics.com.ar

PIN 796948766332890345678046587411

¡un PIC en tu TV!

¿Alguna vez, como parte de alguno de tus proyectos, has necesitado mostrar alguna información en la pantalla de un televisor? Es posible que si. Pero el hardware y el software necesarios para este tipo de proyecto no es fácil de desarrollar. Sin embargo, ahora podrás hacerlo. Este artículo te mostrará cómo convertir un PIC18 en un generador PAL de graficos y textos (en blanco y negro) con un número muy reducido de componentes de bajo costo, con un mínimo de esfuerzo de programación.

// por: Bruno Gavando //
bruno.gavand@ad-valorem.fr

"Utilizaremos un sensor 1321 de Allegro, que nos entrega una variación máxima entre los polos norte y sur de tan sólo 2.5mV"

¿Por que una biblioteca PAL para PICs? Si estás utilizando PICs, posiblemente algun día intentaste construir un dispositivo procesador de vídeo, ya sea para divertirse o como parte de un proyecto más grande. Si intentaste generar señales de vídeo, posiblemente hayas visto alguna de estas páginas web: Rickard Gunees PIC PONG y Eric Smith vídeo reloj.

Estos proyectos son divertidos, pero dificiles de utilizar como un generador de video de carácter general, que permita utilizar un TV como si fuese una pantalla GLCD.

He tenido la idea de utilizar una escalera de resistores como una forma rápida y barata cde crear un convertidor digital a analógico, y comenzado a trabajar en el software.

.La librería PIC PAL

Como la pantalla de vídeo que se ha mapeado en memoria, sólo un PIC con suficiente memoria RAM puede utilizar esta librería. Es por ello que la ***PIC PAL Video Library*** sólo funciona con la familia PIC18.

El PIC debe funcionar a 32MHz, con un cristal de 8MHz, para poder obtener los 64µs necesarios para la sincronización horizontal del sistema PAL.

La librería genera una señal PAL de 625 líneas interlazadas, y puede mostrar hasta 248 líneas verticales de 128 pixeles. Cualquier dispositivo con una entrada PAL de video compuesto puede emplearse para mostrar las imagenes generadas por el PIC.

Dado que el manejo de los tiempos es crítico, el software se ha escrito en C con algunas rutinas en ensamblador. El C utilizado es el mikroC, y puedes descargar el proyecto completo (incluido el código fuente).

La librería contiene muchas funciones asi que explica-

remos el uso de cada una de ellas.

.Inicialización

Esta función prepara la librería PIC PAL. Cuanto mayor sea la cantidad de lineas verticales que desees mostrar en el TV, menor será la cantidad de memoria y de recursos que el PIC tenga para el resto del programa. Esta librería toma el control del TIMER0 y sus interrupciones asociadas. También utiliza el PORTD.

.Control de video

Esta función controla la generación de la señal de video. Cuando comienza la sincronización PAL, la variable PAL_frameCtr (unsigned long global) se incrementa 25 veces por segundo.

.Pintar pantalla

Esta función llena la pantalla con un patrón determinado. Utiliza 0x00 para limpiar la pantalla y 0xFF para pintarla completamente de blanco.

.Seleccionar el color del borde

Esta función cambia el color del borde que rodea la imagen.

.Dibujar puntos

Esta es la función que permite dibujar un punto en una posición determinada de la pantalla.

INICIALIZACION	
Prototipo:	void PAL_init(unsigned char y)
Parámetros:	y : número de lineas verticales, hasta 128.
Devuelve:	Nada
Requiere:	El fichero PAL_library.h debe ser incluido en el codigo fuente del usuario. El PIC18 debe funcionar a 32MHz.
Ejemplo:	PAL_init(128);

CONTROL DE VIDEO	
Prototipo:	void PAL_control(unsigned char st, unsigned char rd)
Parámetros:	st : Control de la sincronización PAL. PAL_CNTL_START : Comienza sincronización PAL. PAL_CNTL_STOP : Detiene la sincronización PAL (libera los recursos del PIC). rd : Control de render PAL_CNTL_BLANK: Solo se muestran los bordes (libera parte de los recursos del PIC). PAL_CNTL_RENDER : Muestra todo el video, con bordes e imagen (consume más recursos del PIC).
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_control(PAL_CNTL_START, PAL_CNTL_RENDER);

PINTAR PANTALLA	
Prototipo:	void PAL_fill(unsigned char c)
Parámetros:	c : Patrón de relleno
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PALL_fill(0);

	SELECCIONAR EL COLOR DEL BORDE
Prototipo:	void PAL_setBorder(unsigned char border)
Parámetros:	border : PAL_COLOR_BLACK o PAL_COLOR_WHITE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_border(PAL_COLOR_BLACK);

	DIBUJAR PUNTOS
Prototipo:	void PAL_setPixel(char x, char y, unsigned char mode)
Parámetros:	x : columna del pixel, de 0 a 127 y : Fila del pixel, de 0 a numero de lineas -1 mode : color del pixel, PAL_COLOR_BLACK, PAL_COLOR_WHITE o PAL_COLOR_REVERSE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_setPixel(10, 20, PAL_COLOR_REVERSE);

	DIBUJAR LINEAS
Prototipo:	void PAL_line(char x0, char y0, char x1, char y1, unsigned char mode)
Parámetros:	x0, y0 : Coordenadas (fila y columna) del comienzo de la linea. x1, y1 : Coordenadas (fila y columna) del final de la linea. mode : color del pixel, PAL_COLOR_BLACK, PAL_COLOR_WHITE o PAL_COLOR_REVERSE.
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_line(0, 0, 127, 127, PAL_COLOR_WHITE);

.Dibujar lineas

Esta función dibuja una linea desde (x0, y0) a (x1, y1).

.Dibujar circulos

Esta función dibuja un circulo con centro en (x,y) y radio z.

.Dibujar rectángulos rellenos

Esta función dibuja (y pinta) un rectángulo.

.Dibujar rectángulos vacíos

Esta función dibuja un rectángulo vacío.

.Escribir un caracter de texto

Esta función escribe un caracter de texto en la fila y columna deseada. Puedes usar PAL_box() para escribir en video invertido.

.Escribir una cadena de texto

Esta función escribe una cadena de texto a partir de la fila y columna deseada.

.Escribir una cadena de texto almacenada en la ROM

Esta función escribe una cadena de texto a partir de la fila y columna deseada. Es igual a la funcion anterior, pero para textos almacenados en ROM.

	DIBUJAR CIRCULOS
Prototipo:	void PAL_circle(char x, char y, char r, unsigned char mode)
Parámetros:	x : Columna del centro del circulo. y : Fila del centro del circulo. r : Radio del circulo. mode : color del pixel, PAL_COLOR_BLACK, PAL_COLOR_WHITE o PAL_COLOR_REVERSE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_circle(30, 30, 5, PAL_COLOR_WHITE);

	DIBUJAR RECTANGULOS RELLENOS
Prototipo:	void PAL_box(char x0, char y0, char x1, char y1, unsigned char mode)
Parámetros:	x0, y0 :Coordenadas de la esquina superior izquierda. x1, y1 :Coordenadas de la esquina inferior derecha. mode : color del pixel, PAL_COLOR_BLACK, PAL_COLOR_WHITE o PAL_COLOR_REVERSE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_box(10, 10, 30, 30, PAL_COLOR_WHITE);

	DIBUJAR RECTANGULOS VACIOS
Prototipo:	void PAL_rectangle(char x0, char y0, char x1, char y1, unsigned char mode)
Parámetros:	x0, y0 :Coordenadas de la esquina superior izquierda. x1, y1 :Coordenadas de la esquina inferior derecha. mode : color del pixel, PAL_COLOR_BLACK, PAL_COLOR_WHITE o PAL_COLOR_REVERSE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_rectangle(10, 10, 30, 30, PAL_COLOR_WHITE);

.Mostrar un bitmap almacenado en ROM

Dibuja una imagen (BitMap) previamente almacenado en la ROM y apuntado por bm, en la posición (x, y). La imagen será monocroma, y puede utilizarse la herramienta para generar mapas de bits para GLCD T6963 de mikroElektronika para convertir las imagenes.

.Rutina de video

Esta función no puede ser llamada directamente por el usuario, perodebe ser colocada dentro de la función interrupt(). Atención: si están habilitadas, otras interrupciones puede ocasionar problemas de sincronismo en el video.

ESCRIBIR UN CARACTER DE TEXTO

Prototipo:	void PAL_char(unsigned char x, unsigned char y, unsigned char c, unsigned char size)
Parámetros:	x : Columna del pixel superior izquierdo del caracter, de 0 to 127 y : Fila del caracter, de 0 a numero de lineas -1 c : Código ASCII del caracter. size : El nibble alto es el multiplicador de la altura, el bajo multiplica el ancho. Tamaños predefinidos: PAL_CHAR_STANDARD, PAL_CHAR_DWIDTH, PAL_CHAR_DHEIGHT, PAL_CHAR_DSIZE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_char(3, 5, 'A', PAL_CHAR_DSIZE);

ESCRIBIR UNA CADENA DE TEXTO

Prototipo:	void PAL_write(unsigned char lig, unsigned char col, unsigned char *s, unsigned char size)
Parámetros:	lig : Linea del texto col : Columna del texto s : Puntero a la cadena de texto (terminado en NULL) size : El nibble alto es el multiplicador de la altura, el bajo multiplica el ancho. Tamaños predefinidos: PAL_CHAR_STANDARD, PAL_CHAR_DWIDTH, PAL_CHAR_DHEIGHT, PAL_CHAR_DSIZE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_write(0, 5, myString, PAL_CHAR_STANDARD);

ESCRIBIR UNA CADENA DE TEXTO ALMACENADA EN LA ROM

Prototipo:	void PAL_constWrite(unsigned char lig, unsigned char col, const unsigned char *s, unsigned char size)
Parámetros:	lig : Linea del texto col : Columna del texto s : Puntero a la cadena de texto (terminado en NULL) size : El nibble alto es el multiplicador de la altura, el bajo multiplica el ancho. Tamaños predefinidos: PAL_CHAR_STANDARD, PAL_CHAR_DWIDTH, PAL_CHAR_DHEIGHT, PAL_CHAR_DSIZE
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_write(0, 5, myConstantString, PAL_CHAR_STANDARD);

MOSTRAR UN BITMAP ALMACENADO EN LA ROM

Prototipo:	void PAL_picture(unsigned char x, unsigned char y, const unsigned char *bm, unsigned char sx, unsigned char sy)
Parámetros:	x : columna del pixel superior izquierdo de la imagen. y : fila del pixel superior izquierdo de la imagen. bm : Puntero al bitmap en ROM sx : Ancho de la imagen. sy : Alto de la imagen.
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();
Ejemplo:	PAL_picture(0, 0, pict, 128, 128);

.El hardware

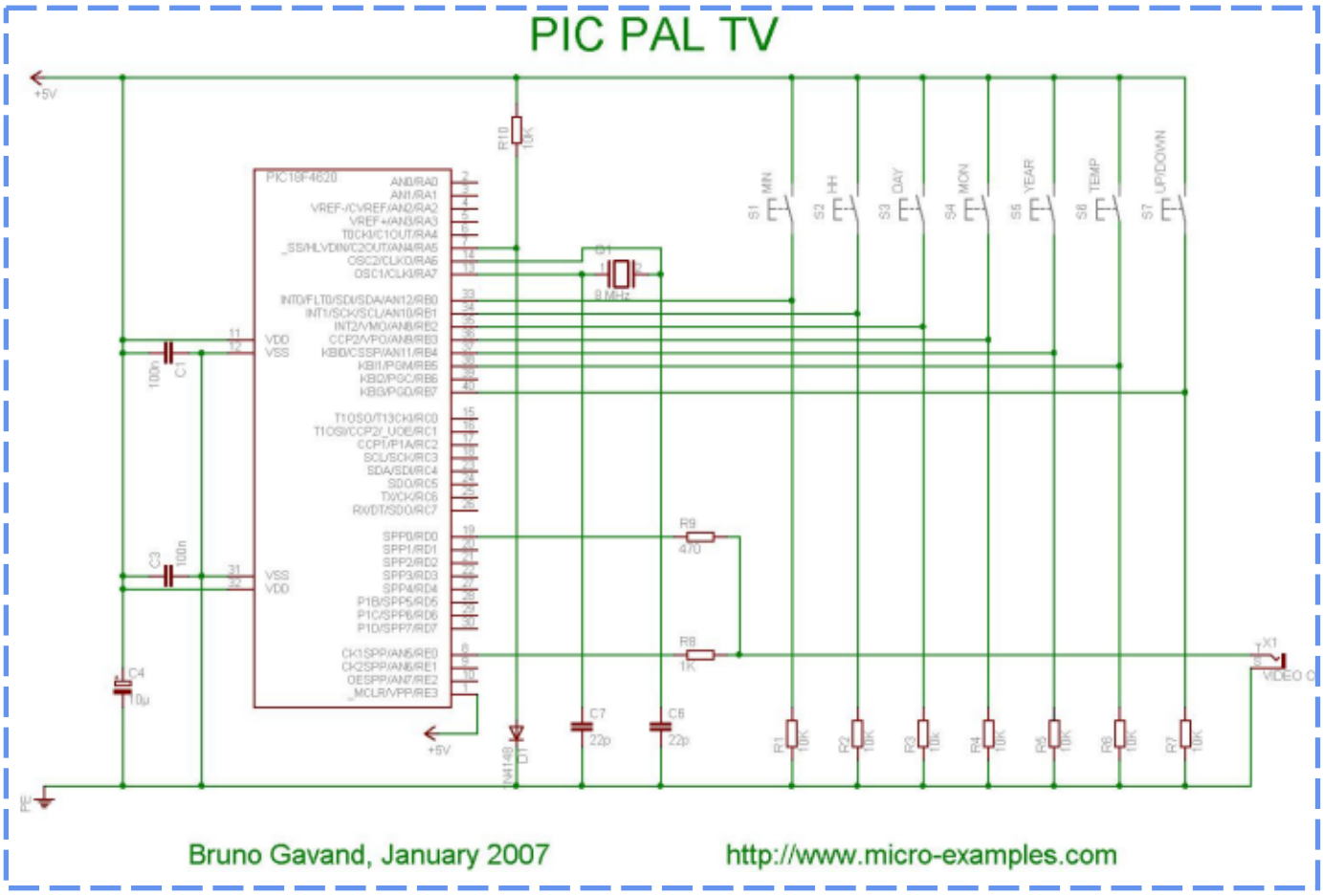
La imagen siguiente muestra el esquema electrico del hardware propuesto.

El corazón del circuito es un PIC18F4620. C1, C3 y C4 son condensadores de desacople. El PIC funciona con un cristal de 8 MHz. D1 se utiliza como un sensor de temperatura conectado al

RUTINA DE VIDEO

Prototipo:	void PAL_ISR()
Parámetros:	
Devuelve:	Nada
Requiere:	La ejecución previa de PAL_init();

Ejemplo:	void interrupt(void)
	{
	PAL_ISR();
	}



conversor analogico digital del PIC. La señal de video se obtiene de los resistores R8 y R9. Los pulsadores se emplean para configurar el soft.

La señal de salida puede aplicarse directamente a la entrada de video compuesto de un aparato de TV.

.El software

El siguiente programa sirve de ejemplo de como puedes utilizar las funciones incluidas en la PIC PAL Library.

En un PIC18F4620 utiliza solamente el 25% de la ROM y el 55% de la RAM!

La primer pantalla del programa muestra una imagen de 128x128 pixeles, con un borde parpadeante. El programa espera a que se pulse la tecla conectada a RB7 para mostrar la segunda pantalla.

La segunda pantalla muestra un reloj-calendario y la temperatura. La hora se muestra simultaneamente en forma digital y analógica.

RB0 cambia los minutos, RB1 las horas, RB2 el día, RB3 el mes y RB4 el año. RB5 permite ajustar la temperatura. Presionando RB7 junto a las anteriores opciones, los valores decrementan en lugar de incrementarse.

.Descargas

Puedes descargar el proyecto completo, incluido el codigo fuente desde aquí.

Primer pantalla del ejemplo.

Segunda pantalla del ejemplo.

El archivo zip contiene:

PAL_library.c, 37 Kb : Código fuente de la libreria
PAL_library.h, 2 Kb : Definiciones de la libreria
PALdemo.c, 15 Kb : Código del ejemplo
PALdemo.eed, 1 Kb : Definición de la EEPROM
PALdemo.hex, 46 Kb : Archivo HEX fpara el PIC18F4620
PALdemo.ppc, 2 Kb : mikroC project
pictures.h, 9 Kb : bitmap de ejemplo

```
/*
 * file        : PALdemo.c
 * project     : PIC PAL SOFTWARE VIDEO GENERATOR DEMO
 * author      : Bruno Gavand
 * compiler    : mikroC V6.2
 * date        : January 17, 2006
 *
 * description  :
 *       This program displays a clock, a calendar and the temperature on a TV screen
 *     and shows how to use the PIC PAL library.
 *           press RB7 to skip the welcome screen
 *           to adjust clock and calendar, press :
 *                 RB0 to adjust minute
 *                 RB1 to adjust hour
 *                 RB2 to adjust day
 *                 RB3 to adjust month
 *                 RB4 to adjust year
 *                 RB5 to adjust temperature
 *           press RB7 at the same time to decrement.
 *
 * target device :
 *     PIC18F4620 @ 32 Mhz (8 Mhz crystal + HS PLL)
 *
 * Licence :
 *     Feel free to use this source code at your own risks.
 *
 * history :
 *     created january 2007
 *
 * see more details on http://www.micro-examples.com/
 */

#include       "PAL_Library.h"
#include       "pictures.h"

/*************
 * DEFINITIONS
 *************/

/*
 * graphic clock
 */
#define CLK_CENTER_X    90      // center
#define CLK_CENTER_Y    60
#define CLK_RADIUS_PSS  28      // clock radius
#define CLK_RADIUS_SS   25      // seconds
#define CLK_RADIUS_MN   20      // minutes
#define CLK_RADIUS_HH   15      // hours

#define DEG_NBHISTO     16      // number of temperature samples
```

```
/*
 * number of vertical pixels
 * from 1 to 128 included
 * the more pixels you have :
 * - the less RAM you have
 * - the less MCU time you have
 */
#define PAL_Y      128

/*
 * simple time structure definition
 */
typedef struct
        {
        unsigned char   ss ;    // seconds
        unsigned char   mn ;    // minutes
        unsigned char   hh ;    // hours
        unsigned char   md ;    // day in month, from 1 to 31
        unsigned char   wd ;    // day in week, monday=0, tuesday=1, .... sunday=6
        unsigned char   mo ;    // month number, from 1 to 12
                               //(and not from 0 to 11 as with unix C time !)
        unsigned int    yy ;    // year Y2K compliant, from 1892 to 2038
        } TimeStruct ;

/********************
 * ROM CONSTANTS
 ********************/

/*
 * month names
 */
const unsigned char monthStr[13][4] =
          {
          "???", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
          } ;

/*
 * day of week names
 */
const unsigned char wDaystr[7][4] =
          {
          "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
          } ;

/********************
 * RAM VARIABLES
 ********************/

/*
 * screen memory map
```

```
 * do not change this line !
 */
unsigned char   PAL_screen[PAL_X * PAL_Y / 8] ;

/*
 * general purpose string
 */
unsigned char   str[20] ;

char    degRef ;                    // DAC temperature reference
char    degHisto[DEG_NBHISTO] ;     // temperature samples buffer
char    tIdx = 0 ;                  // temperature samples index

unsigned long   secOffset = 0 ;     // reference timestamp
unsigned long   oldCtr = 0 ;        // frame counter backup
TimeStruct      ts ;                // time struct

/*************************
 * FUNCTIONS
 *************************/

/*
 * adjust time struct member
 */
void    adjust(unsigned char *v, unsigned char min, unsigned char max)
        {
        if(PORTB.F7)
                {
                if(*v == min) *v = max ;
                else (*v)-- ;
                }
        else
                {
                if(*v == max) *v = min ;
                else (*v)++ ;
                }
        }

/*
 * convert value v into string pointed to by p, leading zero blanks if blk is set
 */
void    char2str(unsigned char *p, unsigned char v, unsigned char blk)
      {
      *p = v / 10 + '0' ;
      if(blk && (*p == '0'))
              {
              *p = ' ' ;
              }
      p++ ;
      *p = v % 10 + '0' ;
```

```
 p++ ;
      *p = 0 ;
      }

/*
 * draw screen with decoration if full is set, using video mode mode
 */
void    drawScreen(unsigned char full, unsigned char mode)
      {
      static unsigned char    osx = CLK_CENTER_X, osy = CLK_CENTER_Y,
                  omx = CLK_CENTER_X, omy = CLK_CENTER_Y,
                  ohx = CLK_CENTER_X, ohy = CLK_CENTER_Y ;
      unsigned int    i ;
        int             t ;
      unsigned char   ss ;
      unsigned char   sx, sy, mx, my, hx, hy ;

      PAL_control(PAL_CNTL_START, mode) ;

      if(full)            // draw full screen with decoration
            {
            PAL_fill(0) ;
            PAL_constWrite( 0,  0, "\xC9\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\
                  xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xBB", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 1,  0, "\xBA PAL LIBRARY DEMO  \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 2,  0, "\xC8\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\
                  xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xBC", PAL_CHAR_STANDARD) ;

            PAL_box(0, 0, 127, 21, PAL_COLOR_REVERSE) ;

            PAL_constWrite( 3,  0, "\xC9\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCB\xCD\xCD\
                  xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xBB", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 4,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 5,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 6,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 7,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 8,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite( 9,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite(10,  0, "\xBA       \xBA       \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite(11,  0, "\xCC\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCA\xCD\xCD\
                  xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xB9", PAL_CHAR_STANDARD) ;
            PAL_constWrite(12,  0, "\xBA                \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite(13,  0, "\xBA                \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite(14,  0, "\xBA                \xBA", PAL_CHAR_STANDARD) ;
            PAL_constWrite(15,  0, "\xC8\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\
                  xCD\xCD\xCD\xCD\xCD\xCD\xCD\xCD\xBC", PAL_CHAR_STANDARD) ;

            PAL_write(4, 3, "H  :", PAL_CHAR_DHEIGHT) ;

            PAL_constWrite(12, 3, "TEMP. : ", 0x31) ;
```

```
        PAL_constWrite(12, 17, "\xf8C", 0x31) ;

            for(ss = 0 ; ss < 60 ; ss++)
                {
                sx = CLK_CENTER_X - (cosE3(90 + 6 * ss) * CLK_RADIUS_PSS) / 1000 ;
                sy = CLK_CENTER_Y - (sinE3(90 + 6 * ss) * CLK_RADIUS_PSS) / 1000 ;

                PAL_setPixel(sx, sy, PAL_COLOR_WHITE) ;

                if((ss % 5) == 0)
                    {
                    PAL_setPixel(sx + 1, sy, PAL_COLOR_WHITE) ;
                    PAL_setPixel(sx - 1, sy, PAL_COLOR_WHITE) ;
                    PAL_setPixel(sx, sy + 1, PAL_COLOR_WHITE) ;
                    PAL_setPixel(sx, sy - 1, PAL_COLOR_WHITE) ;
                    }
                }
            }

        if(PAL_frameCtr > OldCtr)       // it's time to update the clock & calendar
                {
                unsigned char   h ;

              oldCtr = PAL_frameCtr + 24 ;    // prepare oldCtr for next update time

            // convert timestamp to date and time
                Time_EpochToDate(secOffset + PAL_frameCtr / 25, &ts) ;


                /*
                 * draw analog clock
                 */
              sx = CLK_CENTER_X - (cosE3(90 + 6 * ts.ss) * CLK_RADIUS_SS) / 1000 ;
              sy = CLK_CENTER_Y - (sinE3(90 + 6 * ts.ss) * CLK_RADIUS_SS) / 1000 ;

              mx = CLK_CENTER_X - (cosE3(90 + 6 * ts.mn) * CLK_RADIUS_MN) / 1000 ;
              my = CLK_CENTER_Y - (sinE3(90 + 6 * ts.mn) * CLK_RADIUS_MN) / 1000 ;

                h = (ts.hh % 12) * 5 + (ts.mn / 8) ;
              hx = CLK_CENTER_X - (cosE3(90 + 6 * h) * CLK_RADIUS_HH) / 1000 ;
              hy = CLK_CENTER_Y - (sinE3(90 + 6 * h) * CLK_RADIUS_HH) / 1000 ;

              if((hx != ohx) || (hy != ohy))
                    {
                    PAL_line(CLK_CENTER_X, CLK_CENTER_Y, ohx, ohy,
PAL_COLOR_BLACK) ;
                    }
              if((mx != omx) || (my != omy))
                    {
```

```
PAL_line(CLK_CENTER_X, CLK_CENTER_Y, omx, omy, PAL_COLOR_BLACK) ;
                    }
                if((sx != osx) || (sy != osy))
                    {
                    PAL_line(CLK_CENTER_X, CLK_CENTER_Y, osx, osy,
PAL_COLOR_BLACK) ;
                    }
                PAL_line(CLK_CENTER_X, CLK_CENTER_Y, hx, hy, PAL_COLOR_WHITE) ;
                PAL_line(CLK_CENTER_X, CLK_CENTER_Y, mx, my, PAL_COLOR_WHITE) ;
                PAL_line(CLK_CENTER_X, CLK_CENTER_Y, sx, sy, PAL_COLOR_WHITE) ;

                  /*
                   * print date and time
                   */
                char2str(str, ts.ss, 0) ;
                PAL_write(4, 7, str, PAL_CHAR_DHEIGHT) ;

                char2str(str, ts.mn, 0) ;
                PAL_write(4, 4, str, PAL_CHAR_DHEIGHT) ;

                char2str(str, ts.hh, 1) ;
                PAL_write(4, 1, str, PAL_CHAR_DHEIGHT) ;

                PAL_constWrite(6, 2, wdayStr[ts.wd], PAL_CHAR_STANDARD) ;
                PAL_constWrite(7, 2, monthStr[ts.mo], PAL_CHAR_DHEIGHT) ;

                char2str(str, ts.md, 1) ;
                PAL_write(6, 5, str, 0x32) ;

                wordToStr(ts.yy, str) ;
                PAL_write(9, 1, str + 1, PAL_CHAR_DSIZE) ;

                  /*
                   * save old value for fast analog clock cleaning at next update
                   */
                osx = sx ;
                osy = sy ;
                omx = mx ;
                omy = my ;
                ohx = hx ;
                ohy = hy ;
 t = degRef - Adc_Read(4) ;      // read temperature sensor

              t *= 221 ;          // temperature coefficient of the silicon junction
              t /= 102 ;
              t = 25 + t ;      // get the result in celcius

                  /*
                   * adjust limits
                   */
```

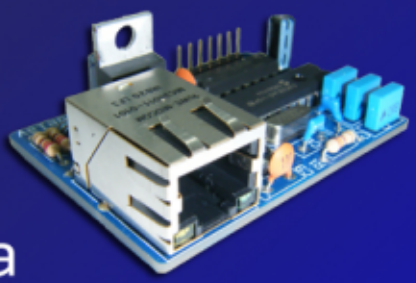
```
if(t < -99)
                    {
                    t = -99 ;
                    }
                if(t > 99)
                        {
                        t = 99 ;
                        }

                /*
                 * average values
                 */
            degHisto[tIdx] = t ;
            tIdx++ ;
            if(tIdx == DEG_NBHISTO)
                    {
                    tIdx = 0 ;
                    }
            t = 0 ;
            for(i = 0 ; i < DEG_NBHISTO ; i++)
                    {
                    t += degHisto[i] ;
                    }
                t /= DEG_NBHISTO ;

                /*
                 * print temperature
                 */
              if(t < 0)
                        {
                        i = -t ;
                        PAL_constWrite(12, 11, "-", 0x31) ;
                        }
              else
                        {
                        i = t ;
                        PAL_constWrite(12, 11, " ", 0x31) ;
                        }
              char2str(str, i, 1) ;
              PAL_write(12, 12, str, 0x32) ;
              }
        // restore video rendering if it was stopped
        PAL_control(PAL_CNTL_START, PAL_CNTL_RENDER) ;
        }
/*
 * interrupt service routine
 */
void    interrupt(void)
        {
```

```
/*
        * do PAL stuff
        */
       PAL_ISR() ;             // library call
       }

/*
 * main program
 */
void main(void)
       {
       unsigned char i ;

         /*
          * I/O configuration
          */
       ADCON1 = 0x0f ;
       TRISA = 0xff ;
       PORTA = 0 ;

       TRISB = 0xff ;
       PORTB = 0 ;

       TRISC = 0xff ;
       PORTC = 0 ;

       TRISD = 0 ;
       PORTD = 0 ;

       TRISE = 0 ;
       PORTE = 0 ;

       degRef = EEPROM_read(0) ;  // get temperature calibration from EEPROM

         /*
          * default time and date
          */
       ts.ss = 0 ;
       ts.mn = 0 ;
       ts.hh = 12 ;
       ts.md = 1 ;
       ts.mo = 1 ;
       ts.yy = 2007 ;

       secOffset = Time_dateToEpoch(&ts) ;

         /*
          * start video and display first screen
          */
```

```
PAL_init(PAL_Y) ;       // init PAL library
        PAL_fill(0) ;           // clear screen
        PAL_picture(0, 0, logo_bmp, 128, 128) ;         // paint picture
        PAL_control(PAL_CNTL_START, PAL_CNTL_RENDER) ;  // start video and rendering

          i = 0 ;
        while(PORTB == 0)    // wait for a key to be pressed
                {
                /*
                        * change border color two times per second
                        */
                if(PAL_frameCtr > 12)
                        {
                                PAL_setBorder(i) ;
                                i = !i ;
                                PAL_frameCtr = 0 ;
                                }
                        }
          PAL_setBorder(PAL_COLOR_BLACK) ;// clear border

        drawScreen(1, PAL_CNTL_BLANK) ;// draw full screen in blank mode (faster)

        for(;;)
                {
                if(PORTB & 0b1111111)   // a key is pressed
                        {
                                Time_EpochToDate(secOffset + PAL_frameCtr / 25, &ts) ;

                                /*
                                 * calendar settings
                                 */
                        if(PORTB.F0)
                                {
                                adjust(&ts.mn, 0, 59) ;
                                        ts.ss = 0 ;
                                }
                        if(PORTB.F1)
                                {
                                adjust(&ts.hh, 0, 59) ;
                                        ts.ss = 0 ;
                                }
                        if(PORTB.F2)
                                {
                                adjust(&ts.md, 1, 31) ;
                                }
                        if(PORTB.F3)
                                {
                                adjust(&ts.mo, 1, 12) ;
                                }
                        if(PORTB.F4)
```

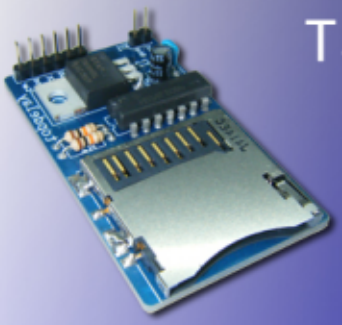
```
{
                    if(PORTB.F7) ts.yy-- ; else ts.yy++ ;
                    }

                secOffset = Time_dateToEpoch(&ts) ;   // new timestamp

                    /*
                     * temperature calibration
                     */
                if(PORTB.F5)
                    {
                    if(PORTB.F7)
                        {
                          degRef-- ;
                          EEPROM_write(0, degref) ;
                        }
                          else
                              {
                          degRef++ ;
                          EEPROM_write(0, degref) ;
                          }
                    }

                    while(PORTB & 0b1111111) ; // wait for the key to be released

                PAL_frameCtr = 0 ;      // reset counters
                  oldCtr = 0 ;
              }

          drawScreen(0, PAL_CNTL_RENDER) ;  // update screen
          }
    }
```


topdelay



Soluciones a medida

Dispositivos y sistemas electrónicos diseñados a petición del cliente



Tarjetas entrenadoras

Microcontroladores, Bluetooth, RS232, Ethernet, Infrarrojos, Keypads, RS485, BusCan, SmartCard, USB, SDCard, RF, ...

Servicio I+D+i

Desarrollos innovadores para empresas, fabricación y diseño de prototipos

www.topdelay.com

contacto@topdelay.com


µCONTROL
Electrónica en General Pics en Particular