Ingeniería en Computación

2015

Taller de Proyecto 1

Informe Final

"Panel LED+interfaz de usuario"

Integrantes

- Acuña Alejandro (111/6)
- Campagna Cristian (11/2)
- Fuentes Facundo (162/8)
- Labrune Juan Pablo (490/8)

Índice

PAGINA

Objetivo

Crear un poncho que, a través de una interfaz de usuario, muestre mensajes mediante una matriz de leds.

Propuesta

Implementaremos una interfaz compuesta por una matriz de botones y un display, donde el usuario deberá ingresar un mensaje para luego confirmar su visualización en el panel de led.

Tanto la botonera como el display utilizaran los puertos GPIO para comunicarse con la EDU-CIAA, mientras que por SPI podremos manejar un micro-controlador que será el intermediario con el panel LED.

Aprovecharemos el uso del sistema OSEK-OS para gestionar la ejecución de dos tareas, brindando al usuario la sensación de que ambas se realizan “simultáneamente”. Una tendrá como objetivo mostrar el mensaje en los LEDs, mientras que la segunda será responsable de la comunicación del usuario con la interfaz.

Descripción del funcionamiento del sistema

Una vez conectado el poncho a la CIAA y esta a una fuente de tensión, el sistema comienza a iniciar automáticamente. Cuando haya finalizado las funciones de inicialización correspondientes se procederá a la etapa de configuración por parte del usuario, donde se seleccionara:

- Velocidad en que se mostrara el mensaje a través del panel LED. (1-5)
- Intensidad de iluminación de los leds del panel. (1-5)
- Modo de muestreo del mensaje:
    - 1 – “Deletreo”
    - 2 – “Circular”
    - 3 – “Ping Pong”

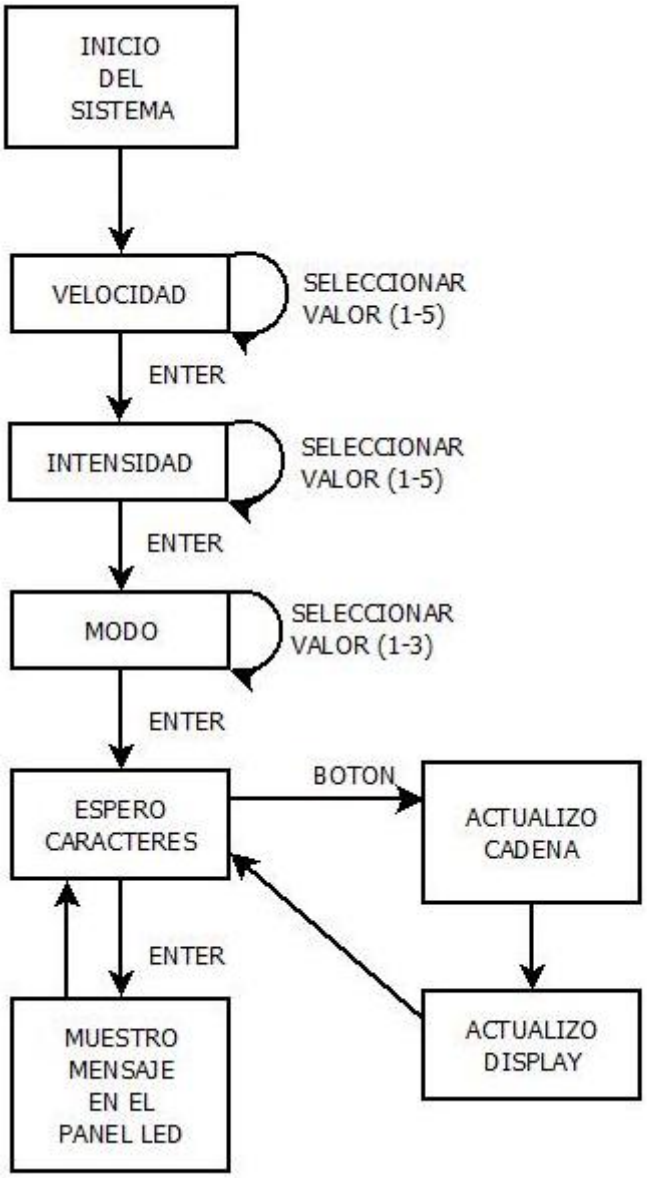
Para la selección de cada nivel se presionara el botón izquierdo de la CIAA, una vez seleccionado el numero deseado se presionara “enter” para pasar a la próxima etapa.

Terminada la etapa de configuración, el sistema espera el ingreso de los caracteres para el armado de la cadena, una vez escrita se presiona “enter” para confirmar su envío hacia el micro-controlador y posteriormente al panel LED.

Mientras la cadena es mostrada a través del panel LED la misma puede ser editada a partir de la botonera y el display. Para actualizar el mensaje nuevamente se requiere que se presione la tecla “enter”.

De esta forma el usuario puede modificar constantemente la cadena mostrada en el display sin afectar al mensaje mostrado por el panel LED hasta que el mismo sea actualizado.

Diagrama de estado de la interfaz del sistema

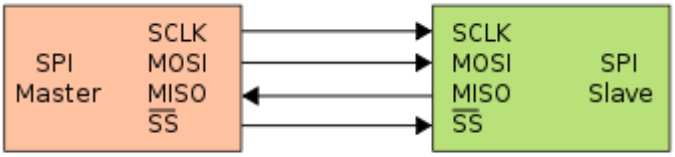


Protocolo de comunicación SPI

SPI es un bus de tres líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada una de estas tres líneas porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es full duplex. Dos de estas líneas trasfieren los datos(una en cada dirección) y la tercer línea es la del reloj.
Algunos dispositivos solo pueden ser transmisores y otros solo receptores, generalmente un dispositivo que tramite datos también puede recibir.
Los dispositivos conectados al bus son definidos como maestros y esclavos. Un maestro es aquel que inicia la transferencia de información sobre el bus y genera las señales de reloj y control. Un esclavo es un dispositivo controlado por el maestro. Cada esclavo es controlado sobre el bus a través de una línea selectora llamada Chip Select o Select Slave, por lo tanto es esclavo es activado solo cuando esta línea es seleccionada. Generalmente una línea de selección es dedicada para cada esclavo.
En un tiempo determinado T1, solo podrá existir un maestro sobre el bus. Cualquier dispositivo esclavo que no esté seleccionado, debe deshabilitarse(ponerlo en alta impedancia) a través de la línea selectora (chip select).
El bus SPI emplea un simple registro de desplazamiento para transmitir la información.

Especificaciones del Bus

Todas las líneas del bus transmiten la información sobre una sola dirección.
La señal sobre la línea de reloj (SCLK) es generada por el maestro y sincroniza la transferencia de datos.
La línea MOSI (Master Out Slave In) transporta los datos del maestro hacia el esclavo y la línea MISO (Master In Slave Out) transporta los datos del esclavo hacia el maestro.



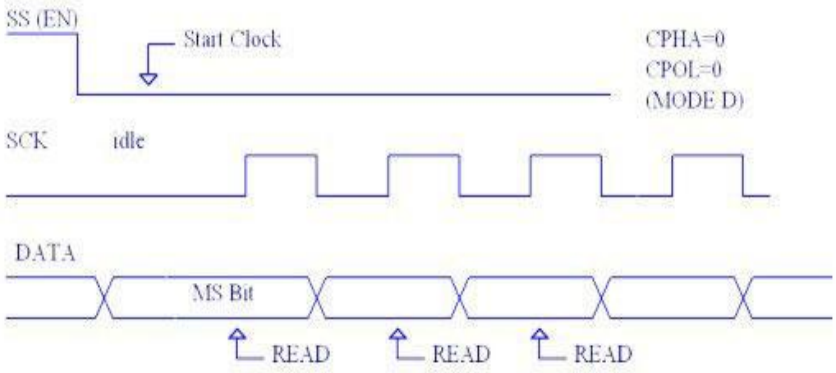
Cada esclavo es seleccionado por un nivel lógico bajo (‘0’) a través de la línea (CS = Chip Select o SS Slave Select). Los datos sobre este bus pueden ser transmitidos a una razón de casi cero bits/segundo hasta 1 Mbits/segundo. Los datos son transferidos en bloques de 8 bits, en donde el bits más significativo(MSB) se transmite primero.

Modos del Reloj

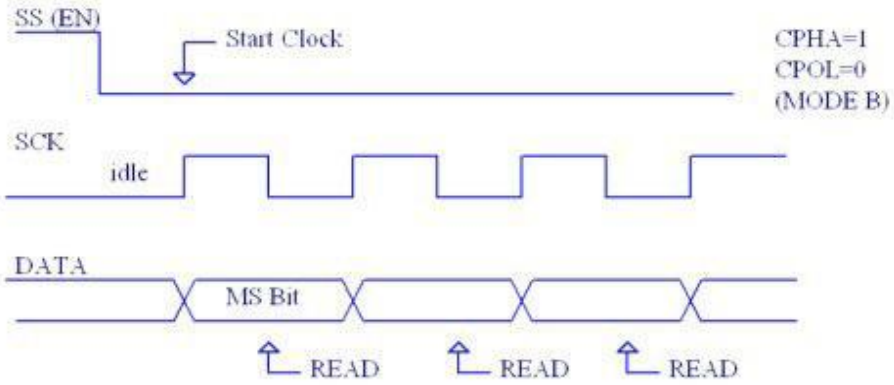
Todos la transferencia de los datos, son sincronizados por la línea de reloj de este bus. Un BIT es transferido por cada ciclo de reloj.
La mayoría de las interfaces SPI tienen 2 bits de configuración, llamados CPOL(Clock Polarity = Polaridad de Reloj) y CPHA(Clock Phase = Reloj de Fase). CPOL determina si el estado Idle de la línea de reloj esta en bajo (CPOL=0) o si se encuentra en un estado alto (CPOL=1). CPHA determina en que flanco de reloj los datos son desplazados hacia dentro o hacia fuera.

Existen cuatro modos en el cual se puede enviar información dependiendo de los parámetros CPOL y CPHA. Al tener dos parámetros donde cada uno puede tomar dos estados se tendrá entonces cuatro modos distintos de poder llevar a cabo el proceso de transmisión y envío de información.

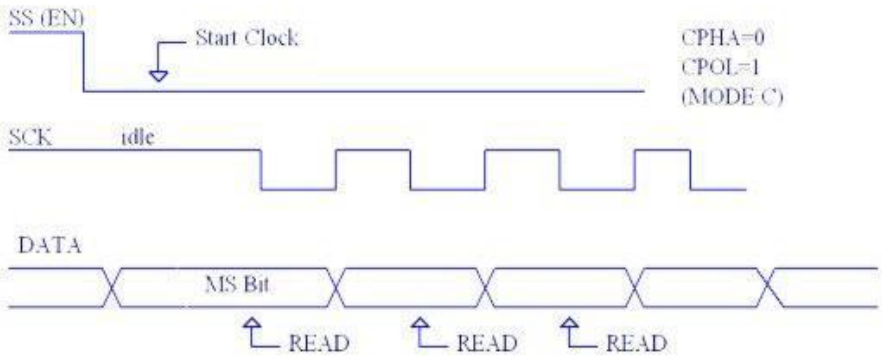
- Modo 0: CPOL = 0 y CPHA = 0. Modo en el cual el estado del reloj permanece en estado lógico bajo y la información se envía en cada transición de bajo a alto, es decir alto activo.



- Modo 1: CPOL = 0 y CPHA = 1. Modo en el cual el estado del reloj permanece en estado lógico bajo y la información se envía en cada transición de alto a bajo, es decir bajo activo.

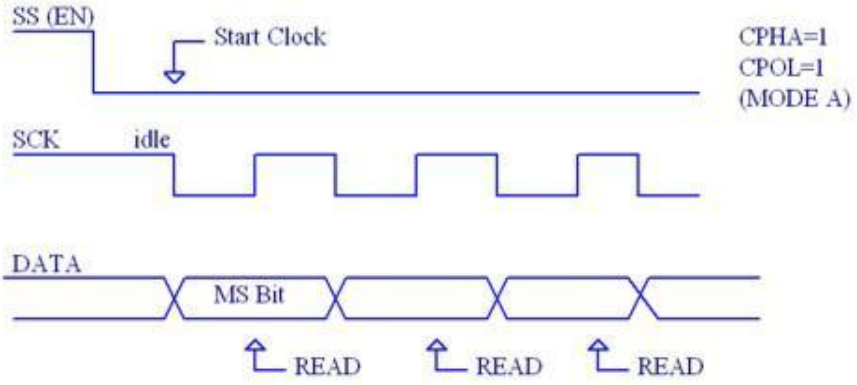


- Modo 2: CPOL = 1 y CPHA = 0. Modo en el cual el estado del reloj permanece en estado lógico alto y la información se envía en cada transición de bajo a alto, es decir alto activo.



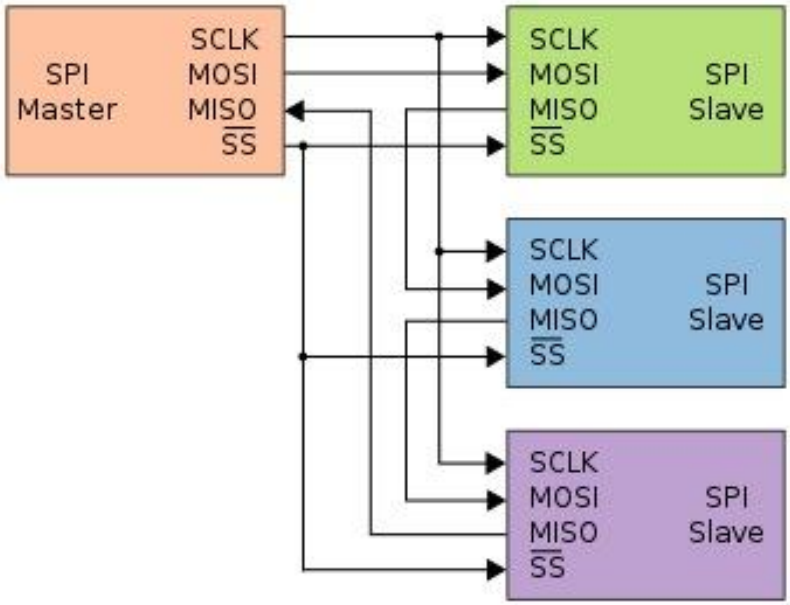
- Modo 3: CPOL = 1 y CPHA = 1. Modo en el cual el estado del reloj permanece en

estado lógico alto y la información se envía en cada transición de alto a bajo, es decir bajo activo.

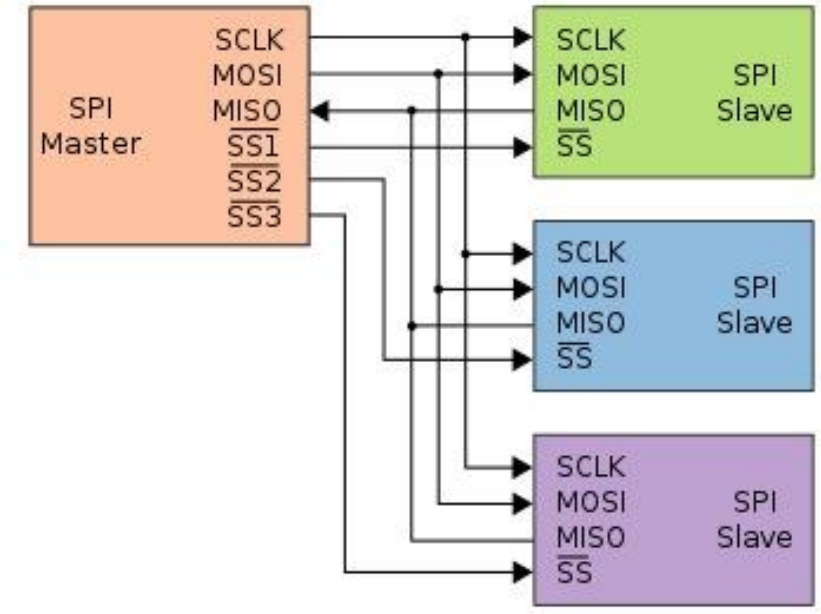


La configuración de modos es independiente para cada esclavo con esto quiere decir que cada esclavo puede tener una configuración de CPOL y CPHA distinta a la de otros esclavos, inclusive una frecuencia de trabajo distinta y entonces para esto, el maestro deberá adaptarse a la configuración de cada esclavo. Por esta razón es recomendable que el sistema trate de trabajar con los mismo parámetros para evitar complicaciones.
En este protocolo se define únicamente un maestro y varios esclavos. La manera en la cual estos dispositivos se conectan pueden ser de dos tipos: encadenado o paralelo. El de tipo encadenado las entrada del MOSI de cada esclavo va conectada con el MOSI del master para el primer caso o de su esclavo anterior para el resto. Además, se utiliza un único de selección de esclavo proveniente del maestro en forma paralela hacia cada esclavo.
Por otro lado, en el tipo paralelo se utiliza un único MOSI proveniente del maestro en forma paralela hacia cada esclavo. Además, se adiciona una línea de selección de esclavo proveniente del maestro por cada esclavo que exista en el sistema.

Tipo encadenado



Tipo paralelo:



Aplicación de SPI

Ya habiendo definido los principios básicos del funcionamiento del protocolo de comunicación procedemos a explicar la forma en que se implemento y utilizo para nuestro proyecto.

A partir del manual de usuario de la Edu-CIAA observamos que el protocolo SPI se implementa a través del SPP, cuyo funcionamiento es similar exceptuando detalles que no afectan a nuestro fin.

Para el uso del periférico abrimos los puertos necesarios y utilizamos el “driver” (Firmware\externals\drivers\cortexM4\lpc43xx\src\ ssp_18xx_43xx) ya implementado bajo la clasificación de “external”, por lo tanto, sin la utilización de POSIX.

El driver por defecto se encuentra configurado para el envío de paquetes de un byte por cada transacción, esto fue modificado ya que, en nuestro caso, debemos enviar 2 bytes para comunicar correctamente el periférico con el esclavo (MAX7219). Dicho cambio se efectuó dentro de la función Chip_SSP_Init(LPC_SSP_T *pSSP) al momento de llamar a Chip_SSP_SetFormat(pSSP, **SSP_BITS_16**, SSP_FRAMEFORMAT_SPI, SSP_CLOCK_CPHA0_CPOL0). De esta manera, cada vez que se realice un envío se transmitirán 16 bits por la señal del MOSI.

La inicialización predefinida en el driver define los siguientes paramentaros de configuración:

- Habilita y obtiene el clock predeterminado.
- Define a la CIAA como el Master.
- Configura el formato del paquete(frame).

- Configura el bit rate en 100,000.

A partir de las funciones ya implementadas en el driver, se desarrollo una función de inicialización para el periférico SPP llamada Init_SPP1(void) dentro del archivo principal del proyecto, la cual realiza la siguiente secuencia:

- Inicialización por “defecto” del SPP con la Dirección de memoria de registros del SPP1.
- Habilito el periférico con la Dirección de memoria de registros del SPP1.
- Apertura de los puertos:
  - P1_4 en función 5 para el MOSI
  - P1_20 en función 1 para el Slave Select
  - PF_4 en función 0 para el Clock

Una vez finalizada la inicialización se puede establecer una comunicación a través de la función predefinida en el driver: Chip_SSP_WriteFrames_Blocking(Dirección de memoria de registros del SPP1, dato, cantidad de paquetes).
Un detalle a tener en cuenta es que la dirección de memoria a registros del SPP1 es utilizada a través del puntero LPC_SSP1, el cual ya se encontraba definido en el archivo Firmware\externals\drivers\cortexM4\lpc43xx\inc\chip_lpc43xx.h del firmware oficial.

Bibliografía

- http://panamahitek.com/como-funciona-el-protocolo-spi/
- Ing. Eric López Pérez , Apunte de Protocolo SPI.

Panel + Max7219

El Max7219 es un integrado que utiliza comunicación serie para manipular distintos tipos de displays de led. En nuestro caso comunicará la EDU-CIAA con un panel led de 8x8. Las características de dichos elementos serán detallados a continuación.

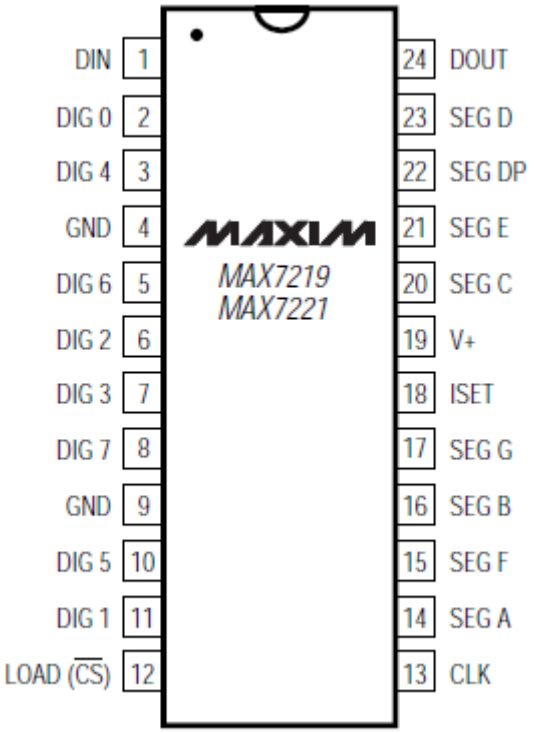
Características del Max7219

- Interfaz serie de 10Mhz.
- Control individual de cada segmento led.
- Selección de modo: decodificador/no decodificador de digito.
- Control de brillo digital o análogo.
- Compatible con protocolo SPI.

Aplicaciones del Max7219

- Display en formato barra.
- Display de 7 segmentos.
- Display matriz de led.

Configuración de pines

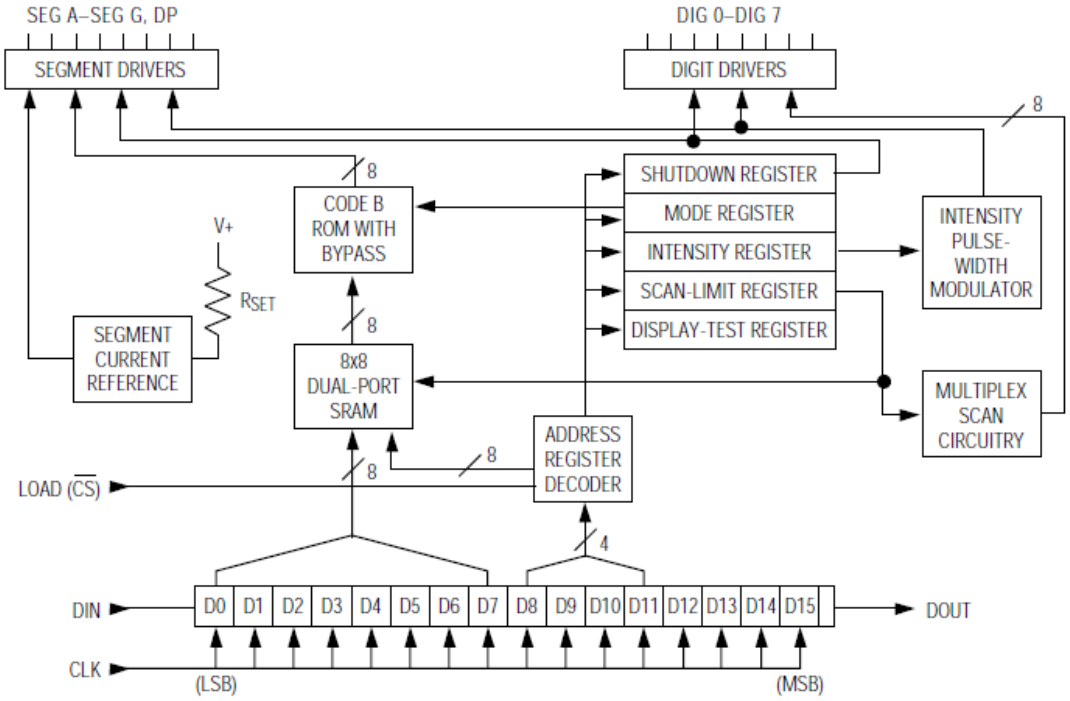


Descripción de pines

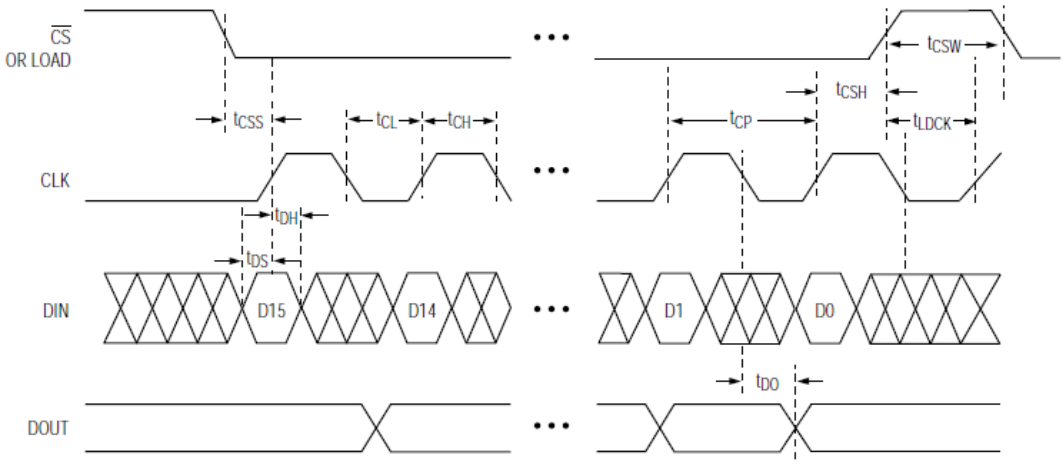
PIN	Nombre	Función
1	DIN	Entrada de datos serie. Los datos son cargados a un registro de desplazamiento interno de 16 bits en cada flanco ascendente de reloj.
2, 3, 5-8, 10, 11	Dig0-Dig7	Líneas para manejar 8 dígitos.
4, 9	GND	Tierra
12	LOAD	Entrada de carga de datos. Los últimos 16 bits del dato serie están sincronizados al flanco ascendente del LOAD.

13	CLK	Entrada de señal de reloj. Máximo 10 Mhz. En cada flanco ascendente del reloj el dato es desplazado dentro del registro interno.
14-17, 20-23	SEG A – SEG G, DP	Administrador de 7 segmentos. Para nuestro caso está conectado a GND.
18	ISET	Conectado a Vdd a través de una resistencia para establecer el pico de corriente.
19	V+	+5V
24	DOUT	Salida de datos serie.

Diagrama funcional



Comunicación serie



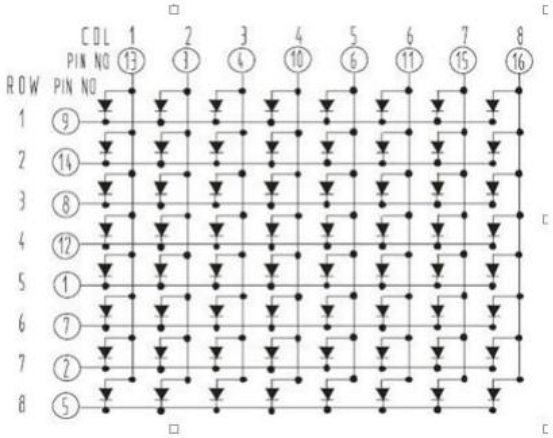
Formato del paquete de 16 bits

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

Descripción del panel:

- Modelo: 1088AS
- Matriz led 8x8 de 3mm.
- Color: rojo.
- Tipo: Columna cátodo – fila ánodo.
- Longitud de onda: 625 - 630nm.
- Tensión: 2.1 – 2.5V.
- Corriente: 20mA.
- Dimensiones: 32mm x 32mm x 8.0mm

Esquema:



Aplicación del MAX7219 y panel LED

Habiendo explicado las características más importantes de los componentes, procedemos a explicar el software desarrollado para su implementación:
Firmware\projects\myproject\src\LedControl.c

Como primer medida creamos una función que nos permita comunicar el integrado con la EDU-CIAA a través del protocolo SPI llamada “sendByte(registro,dato)”, donde el parámetro registro es la dirección (bit 8-11) y dato el valor enviado (bit 0-7). La misma crea un buffer de 16 bits asignado la dirección y el dato, en su ubicación correspondiente, para luego ser transmitido mediante la función Chip_SSP_WriteFrames_Blocking(), la cual fue explicada en el apartado correspondiente al SPI.

El siguiente paso fue realizar la inicialización del integrado a través de la función “Init_MAX7219()” donde se estableció la siguiente configuración:

- Sin test de display (envío 0 al registro 0x15).
- Mostrar los 8 digitos (envío 7 al registro 0x11).
- Utilizo la matriz de led, sin digitos (envío 0 al registro 0x09).

- Realizo un llamado a la función “clearDisplay()”.
- Desactivo modo de apagado (envío 0 al registro 0x12).
- Activo modo de apagado (envío 1 al registro 0x12).
- Selecciono la intensidad (envío valor numérico al registro 0x10).
- Realizo un llamado a la función “clearDisplay()”.

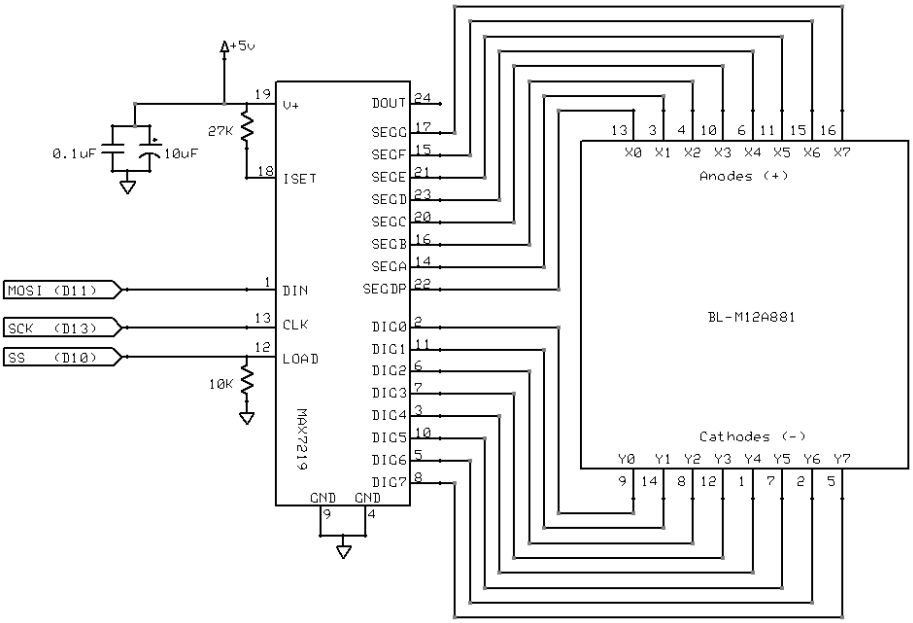
Finalmente desarrollamos la función “msjScroll()” que implementa 3 modos, a elección del usuario, para el pasaje de mensajes a través del panel led:

- Modo 0: El mensaje es deletreado en el panel.
- Modo 1: El mensaje es mostrado en forma de cadena circular.
- Modo 2: El mensaje es mostrado en forma de cadena que “rebota”, primero hacia la izquierda y luego hacia la derecha.

Para llevar a cabo la muestra de mensajes es necesario el llamado de varias funciones complementarias, las mismas son detalladas y explicadas brevemente a continuación:

- printString(posición, cadena): Funcion para mostrar una cadena de caracteres en una posición determinada.
- printChar(posición, letra): Posee el mapeo de los caracteres alfanuméricos que pueden mostrarse a través del panel.
- printStringScroll(posición, cadena, delay, sentido): Es el encargado de realizar el corrimiento de la cadena de caracteres en el modo 1 y 2.
- setRow(columna, valor): Envía un valor a la columna correspondiente.
- clearDisplay(): Limpia el display, apagando todos los leds.

Los parámetros de velocidad, modo e intensidad son cargados por el usuario al momento de iniciar el sistema, mientras que la dirección de los registros del micro-controlador son declaradas en forma de constantes al inicio del archivo.



Bibliografía

- Datasheet Maxim 7219
- Datasheet 1088AS

Puertos GPIO

GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General) es un pin genérico, cuyo comportamiento puede ser definido por el usuario en tiempo de ejecución.

Los pines GPIO no tienen ningún propósito especial definido, y no se utilizan de forma predeterminada. Estos pines pueden ser configurados como entrada o salida, dependiendo de la función que realicen.

Especificaciones de los GPIO en la EDU-CIAA-NXP

La placa utilizada para el proyecto (EDU-CIAA – NPX) posee 2 tiras de 40 pines hembra de 0.1" (2,54mm) lo que nos da un total de 80 pines. Exceptuando los utilizados para la alimentación podemos observar que, si bien cada pin posee un fin específico, todos ellos pueden ser designados y utilizados como GPIO. Aun así, existen 8 pines que están definidos de forma predeterminada para la utilización del protocolo.

Cada pin de la EDU-CIAA posee hasta 7 funcionalidades distintas, por lo cual al momento de su apertura es necesario definir la función requerida. Además será necesario especificar qué tipo de comportamiento físico tendrá, por ejemplo si será utilizado en pull-up o pull-down, o la cancelación del ruido en caso de utilizar una entrada.

Implementación de los GPIO

En el caso del poncho a desarrollar se utilizaran dos módulos de GPIO: el primero consta de un teclado matricial 4 x 3, es decir 4 filas y 3 columnas, que será complementado con la botonera ya implementada en la misma CIAA. El segundo módulo será un display LCD de 2 filas y 16 columnas.

Realizamos una serie de cambios en el driver desarrollado para el uso del protocolo a través del estándar Posix, ubicado en el archivo:

Firmware\modules\drivers\cortexM4\lpc43xx\lpc4337\src\ciaaDriverDio.c

Para el teclado matricial se necesitaran los pines T _FIL0, T_FIL1, T_FIL2, T_FIL3 configurados como salidas, y los pines T_COL0, T_COL1, T_COL2 configurados como entradas:

- T _FIL0: Puerto P4_0 en función 0 para GPIO
- T_FIL1: Puerto P4_1 en función 0 para GPIO
- T_FIL2: Puerto P4_2 en función 0 para GPIO
- T_FIL3: Puerto P4_3 en función 0 para GPIO
- T_COL0: Puerto P1_5 en función 0 para GPIO
- T_COL1: Puerto P7_4 en función 0 para GPIO
- T_COL2: Puerto P7_5 en función 0 para GPIO

Además cada puerto que sea abierto deberá declararse en el vector correspondiente a entrada o salida según corresponda:

Para las entradas → const ciaaDriverDio_dioType ciaaDriverDio_Inputs[] = { {0,4} };

Para las salidas → const ciaaDriverDio_dioType ciaaDriverDio_Outputs[] = { {5,13} };

El número que se ve entre llaves es la designación formal que el fabricante le dio al pin, este número se puede conseguir fácilmente consultando la hoja de datos (DataSheet), o la hoja de asignación de pines (Pin Out).

Luego utilizando el siguiente comando:

Chip_SCU_PinMux(4,0,MD_PUP|MD_EZI,FUNC0);

Determinamos el comportamiento físico y lógico que tendrá dicho puerto. Para este caso estaremos usando de ejemplo el pin T_FIL0 que fue configurado como salida. Para dicha función necesitamso 4 parámetros: En los 2 primeros se deberá indicar el puerto que necesitamos (designación formal del puerto, en este caso los números provienen de P4_0), luego el comportamiento físico del mismo (MD_PUP|MD_EZI indica que estará en pull up y el ZERO enable), y por ultimo cual será el comportamiento lógico eligiendo una de las 7 funciones, la FUNC0 en este caso indica que será utilizado como GPIO. Tomando como ejemplo los botones integrados a la CIAA y su respectiva apertura, a los pines configurados como entrada, se le agrego como propiedad la cancelación de ruido (MD_ZI) para evitar que la señal entre con interferencia.

Por último se utilizan estas 2 funciones complementarias para finalizar la apertura del puerto:

Chip_GPIO_SetDir(LPC_GPIO_PORT, 2,(1<<4)|(1<<5)|(1<<6),1);

Chip_GPIO_ClearValue(LPC_GPIO_PORT, 2,(1<<4)|(1<<5)|(1<<6));

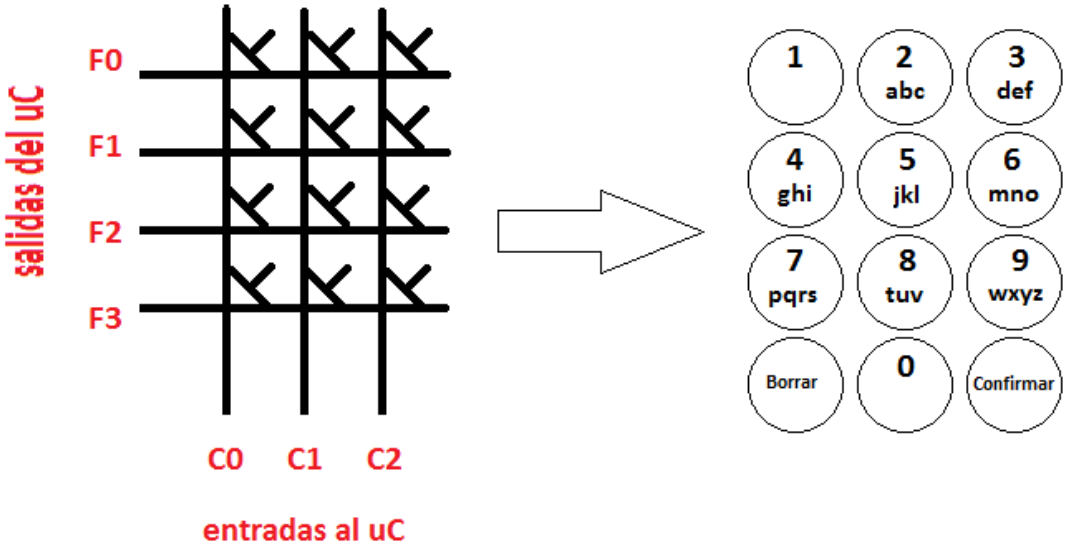
Para el caso del display se debió configurar de la misma manera que se explicó con el teclado matricial, pero para los siguientes pines: LCD1, LCD2, LCD3, LCD4, LCD_RS, LCD_EN, todos ellos configurados como salida y en pull up.

- LCD1: Puerto P4_4 en función 0 para GPIO
- LCD2: Puerto P4_5 en función 0 para GPIO
- LCD3: Puerto P4_6 en función 0 para GPIO
- LCD4: Puerto P4_10 en función 4 para GPIO
- LCD_EN: Puerto P4_9 en función 4 para GPIO
- LCD_RS: Puerto P6_4 en función 0 para GPIO

Teclado Matricial

Para la interfaz de usuario se decidió optar por un teclado matricial de 4 x 3 estilo NumPad por la sencillez en su uso e implementación, dado que la mayoría de los usuarios están familiarizados con este modo de escritura. Además su formato compacto nos da la posibilidad de incluirlo en el poncho junto al resto de los componentes.

Su implementación requiere el uso de 7 pines GPIO, explicados en el apartado anterior y mostrados en la siguiente figura:



Los pines del F0 al F3 estarán marcados como salida, y los del C0 al C2 como entrada, todos configurados en Pull Up. Esto nos dará la posibilidad de "barrer" dichos pines de modo tal que podremos determinar que botón fue apretado cuando veamos un 0 en la entrada del uC, forzando la salida.

Para el análisis del estado de cada puerto necesitaremos una máscara que contendrá el valor actual de todos los pines, en caso de que ninguno se encontrara presionado todos ellos estarán en 1 gracias al pull up. El "barrido" comienza forzando los valores de una fila a la vez (comenzando desde la fila 0) poniéndola en valor bajo '0' y viendo cómo reacciona la entrada, si esta se encuentra también en 0 significa que el botón que corresponde a esa fila (salida) y a esa columna (entrada) fue presionado. En caso de no ver ningún 0 se procede a regresar el valor de la fila a '1' y seguir con la siguiente.

Pseudocodigo del barrido:

```
Input = 0          //variable auxiliar para leer las entradas.
outputs            //variable global, mascara de bits utilizada por todo el sistema
//FILA 1
//forzamos el 0 en la primer fila, asegurándonos de no modificar el resto de los bits de la
máscara
```

```
outputs = outputs & 0xFFBF;                    //xxxx xx11 10xx xxxx
outputs = outputs | 0x0380;                    //xxxx xx11 10xx xxxx
Utilizo POSIX para escribir la salida
Utilizo POSIX para leer la entrada
Si C0 == 0
    chequeo anti rebote
        botón_seleccionado = 1
si C1 == 0
    chequeo anti rebote
        botón_seleccionado = 2
si C2 == 0
    chequeo anti rebote
        botón_seleccionado = 3

//FILA 2
outputs = outputs & 0xFF7F;                    //xxxx xx11 01xx xxxx
outputs = outputs | 0x0340;                    //xxxx xx11 01xx xxxx
//se repite la operación, pero ahora si la entrada esta en 0 tendra otro significado.
Si C0 == 0
    chequeo anti rebote
        botón_seleccionado = 4
si C1 == 0
    chequeo anti rebote
        botón_seleccionado = 5
si C2 == 0
    chequeo anti rebote
        botón_seleccionado = 6
//se repiten las operaciones para las 2 filas siguientes.
```

Para la botonera que se encuentra en la CIAA se optó por dejarla como botones independientes, dado que se utilizaran para el control del mensaje y no para su escritura, además de que serán utilizados en la etapa de inicialización del sistema. Para determinar si algún botón de estos fue presionado tan solo se pregunta a cada uno de ellos si están en 0 de forma individual (se pregunta por 0 ya que también están configurados en pull up), este proceso se realiza 1 vez antes de cada barrido sobre la matriz. Las funciones son SHIFT, IZQUIERDA, DERECHA y ENTER:

Cuando se detecta que un botón es apretado, automáticamente es guardado en una variable que será consultada por el sistema para determinar qué acción tomar. En primera instancia, si

la tecla tocada fue un carácter alfanumérico este se mostrara en el LCD, dándole al usuario una retroalimentación de sus acciones y así poder elegir su acción siguiente.

Dado que se tiene más de un carácter por tecla se optó por precargar una matriz de caracteres que, dependiendo de cuantas veces se ha tocado una misma tecla de forma contigua, muestre el carácter deseado por el usuario. El seguimiento de la matriz se hace mediante un arreglo de contadores que contara cuantas veces se tocó una determinada tecla.

Este conteo es un tanto particular: comenzamos desde la posición 1 e ira sumando de a 2, salteándose un carácter de por medio, esto es así dado que se puede alternar el recorrido de la matriz dependiendo si estamos buscando mayúsculas o minúsculas. Explicaremos mejor este proceso a continuación.

Solo puede haber una posición de ese arreglo con un valor distinto de 0, ya que el sistema solo entenderá que se puede tocar una sola tecla de forma contigua a la vez. Si una posición esta en 0 quiere decir que esa tecla no fue tocada, por otro lado si alguna tecla fue presionada la posición del arreglo de la misma indicara cuantas veces, entonces mediante la posición del arreglo y la cantidad de veces que se tocó la tecla se obtendrán las coordenadas para determinar que carácter mostrar. La matriz de caracteres es la siguiente:

```
uint8_t charmap[10][11] = {
                {' ',' ','?','!','#','$','%','&','*','(',')'},      //0
                {' ','0','1','2','3','4','5','6','7','8','9'},      //1
                {' ','a','A','b','B','c','C',' ',' ',' ',' '},      //2
                {' ','d','D','e','E','f','F',' ',' ',' ',' '},      //3
                {' ','g','G','h','H','i','I',' ',' ',' ',' '},      //4
                {' ','j','J','k','K','l','L',' ',' ',' ',' '},      //5
                {' ','m','M','n','N','o','O',' ',' ',' ',' '},      //6
                {' ','p','P','q','Q','r','R','s','S',' ',' '},      //7
                {' ','t','T','u','U','v','V',' ',' ',' ',' '},      //8
                {' ','w','W','x','X','y','Y','z','Z',' ',' '}       //9
                };
```

El primer carácter esta en blanco porque indica que ese botón no se presionó

El sistema también contempla si la tecla shift fue precionada, la cual solo tendrá efecto sobre los caracteres alfabéticos. Mientras nos encontremos entre la tecla 2 y la 9, cada vez que se haya presionado una de estas teclas, se revisara si no fue presionada la tecla shift anteriormente. Esto realizara que se alterne la manera de cómo se recorra la matriz (si se iba por los casilleros pares, pasara a los impares y viceversa).

Otra característica del sistema es el conteo de auto-confirmación, si luego de apretar una tecla no se presiona ninguna otra (incluyendo las individuales de la CIAA) luego de un determinado tiempo, el sistema interpretara que se confirmó dicha tecla y pasara a esperar por el siguiente carácter o comando.

Cada carácter escrito por el usuario será guardado sobre un buffer y una vez finalizado podrá confirmarse, a través de botón de “enter”, su envío hacia el panel de LED.

Por último se detalla el resto las funcionalidades de las teclas que no corresponden a caracteres alfanuméricos ni a las ya mencionadas:

- Borrar: Borra el carácter en la posición anterior y posiciona al cursor sobre dicha posición.
- Confirmar: Si se llegó a la letra deseada se puede apretar esta tecla para confirmar y pasar a la siguiente posición del mensaje.
- Derecha: Mueve el cursor a la derecha mientras sea posible. Una vez que llego al límite derecho el cursor queda en esa posición (no es cíclico).
- Izquierda: Misma funcionalidad que Derecha, pero para el lado izquierdo.

Bibliografía

- EDU-CIAA-NXP_Pinout_A4_v3r1_ES.pdf

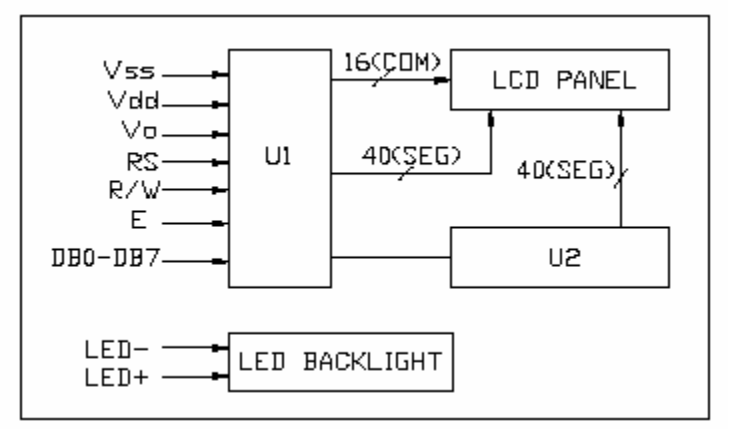
Display

Una de las partes utilizadas para nuestra interfaz de usuario fue un display de 2 filas y 16 columnas, el cual será utilizado para mostrar la cadena escrita por el usuario para su posterior envío.

Modelo: LCD Module CCM1620CSL
Características:

- Tipo: STN de 16 pines.
- 16 caracteres x 2 líneas.
- Entrada de datos: 4 u 8 bits.
- Fuente de 5x8 puntos.
- Modo: amarillo-verde.
- Tensión de funcionamiento: 5.0V.

Diagrama funcional



Descripciones de pines

Numero	Símbolo	Nivel	Función
1	Vss	--	GND(tierra)
2	Vdd	--	5.0V
3	Vo	--	Contraste(preset)
4	RS	Alto/bajo	Alto: Selección de registro Bajo: Instrucción
5	R/W	Alto/bajo	Alto: Lectura – Bajo: Escritura
6	E	Alto, Alto-bajo	Señal que habilita leer o escribir dato.
7	DB0	Alto/bajo	Bus de dato para transferencia de 8 bits.
8	DB1	Alto/bajo	
9	DB2	Alto/bajo	
10	DB3	Alto/bajo	
11	DB4	Alto/bajo	Bus de dato para transferencia de 4 u 8 bits.
12	DB5	Alto/bajo	
13	DB6	Alto/bajo	

14	DB7	Alto/bajo	
15	A	--	Led Backlight (+)
16	K	--	Led Backlight (-)

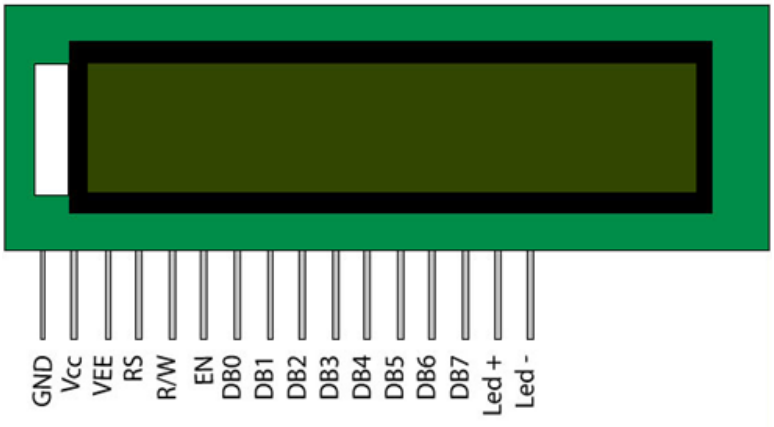
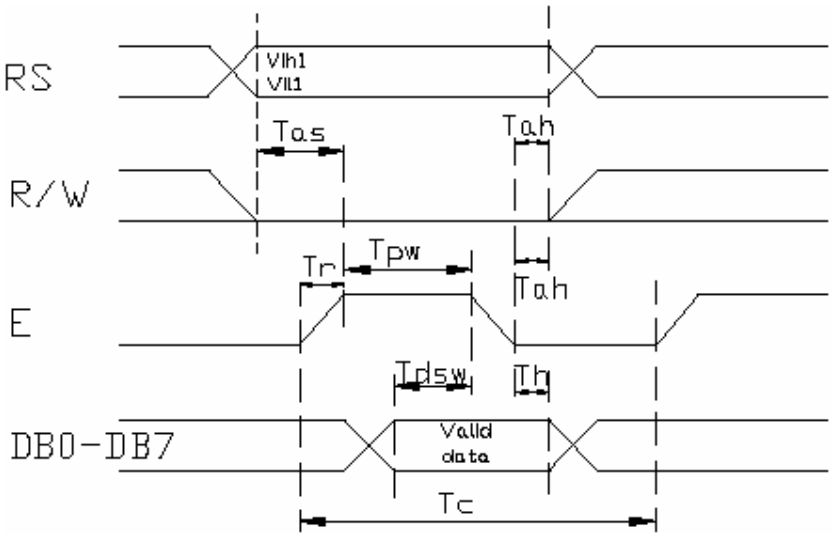


Diagrama de modo escritura



Aplicación del display

Para el uso del display decidimos realizar la comunicación a través de los puertos GPIO, realizando la apertura de los mismos y utilizando la máscara de salida explicada anteriormente en el apartado de dicho protocolo.

La conexión de los puertos del periférico cuenta con la siguiente configuración:

Pin	Valor	Justificación
1	GRND	--
2	5.0V	--
3	Salida del preset	Ajuste de contraste
4	Puerto RS de la CIAA	--
5	GRND	Solo se utilizara para escribir

6	Puerto E de la CIAA	--
7-10	--	No se utilizaran 8bits
11-14	Puertos LCD_1-4 de la CIAA	Comunicacion de 4 bits
15	5.0V	--
16	GRND	--

Habiendo explicado las características físicas del periférico y su respectiva conexión, definiremos el software desarrollado para el mismo: Firmware\projects\myproject\src\lcd.c

Como ya hemos descripto utilizaremos 4bits para escribir sobre el display, por lo tanto creamos una función "LCD_send_nibble(dato)", donde el dato es divido en sus 4 bits para poder cargar cada uno en la máscara de salida. Una vez establecida la salida, envío un pulso alto (1) a través del pin de "Enable" y, luego de una pequeña demora, un pulso bajo(0).
Sin embargo en determinadas situaciones vimos la necesidad de enviar 8 bits, por lo cual se creó la función "LCD_send_byte(dirección, dato)". En caso de que la dirección fuese cero se enviara una señal de igual valor a través del pin RS, caso contrario, se envía una señal de valor uno. Luego se divide el dato en dos parte, enviando a través de la función "LCD_send_nibble(dato)" primero la parte la parte alta y luego la baja.

A partir de esas funciones de envío de los datos se procede a la etapa de inicialización mediante la función "LCD_init(modo1, modo2)", siendo ambos parámetros datos de 8 bits se procede a llevar a cabo la siguiente secuencia, la cual fue utilizada en la asignatura de micro-controladores:

- Se colocan todos los pines del periférico en cero.
- Delay.
- Se realiza tres veces la operación: LCD_send_nibble(3) + delay.
- LCD_send_nibble(2).
- Configuración del periférico:
  - LCD_send_byte(0,0x20 | mode1).
  - LCD_send_byte(0,0x08 | mode2).
- LCD_send_byte(0, 1).
- Delay
- LCD_send_byte(0, 6).

Una vez finalizada la función, el display se encuentra listo para recibir una orden de escritura mediante el llamado a la función update_LCD enviando como parámetros:

- Cadena de caracteres a enviar.
- Numero de línea en la cual escribir (0 o 1)
- Final de la cadena.

Para inicializar el display y escribir dicha cadena fueron utilizadas otras funciones complementarias:

- LCD_pos_xy(x, y): posicionamiento del cursor.
- LCD_write_char(caracter): envío de un carácter.
- LCD_write_string (cadena, fin): envío de una cadena, de a un caracter a la vez.
- LCD_display_on: enciende el display.
- LCD_display_off: apaga el display.

- LCD_cursor_on: enciente el cursor.
- LCD_cursor_off: apaga el cursor.
- LCD_cursor_blink_on: enciende el parpadeo del cursor.
- LCD_cursor_blink_off: apaga el parpadeo del cursor.

Bibliografia

- Datasheet LCD Module CCM1620CSL
- Apuntes y trabajo practico n°4 de Micro-controladores.

Sistema OSEK-OS

Introducción

OSEK-VDX es un comité de estandarización creado en 1994 por las automotrices europeas. OSEKVDX incluye varios estándares que se utilizan en la industria automotriz, entre ellos los más relevantes son:

- OSEK OS
- OSEK COM
- OSEK NM
- OSEK Implementation Language
- OSEK RTI
- OSEK Time Trigger Operating System

El estándar fue creado principalmente para poder reutilizar el SW de un proyecto a otro gracias a la definición de una interfaz estandarizada. Además al proveer un estándar se da la posibilidad a nuevas empresas de proveer SW compatible con el mismo y permitir la implementación de diversos sistemas OSEK compatibles. Esto último permite a la industria automotriz la posibilidad de elegir el proveedor dentro de una lista mayor de proveedores de SW ya que cualquiera puede implementar un OSEK-OS u otro estándar especificado por OSEK-VDX. Hoy en día hay muchas implementaciones de OSEK-OS en el mercado. De ellas algunas son libres con diferentes licencias, así como también hay implementaciones cerradas.

A diferencia de otros sistemas operativos OSEK-OS es un sistema estático. Esto significa que las tareas, sus prioridades, cantidad de memoria asignada, entre otras características, son definidas en tiempo de compilación (generate) y no durante la ejecución. Además esto tiene como ventaja que el sistema se comporta de forma totalmente determinística. Dichas definiciones son declaradas en el archivo OIL.

Configuración de tareas

Debido a lo explicado anteriormente es necesario configurar cada tarea que realiza el sistema con sus respectivas propiedades:

- Nombre
- Prioridad
- Tipo de Scheduler
- Activacion
- Tipo
- AutoStart

Administración de tareas

A diferencia de otros sistemas donde las tareas corren por tiempo indeterminado, en OSEK por lo general realizan su cometido y finalizan. Sin embargo, existen casos en los que se requiere controlar las tareas y su ejecución, para lo cual el sistema ofrece las siguientes interfaces:

- ActivateTask: activa una tarea.
- ChainTask: realiza la combinación de ActiveTask seguido de TerminateTask.

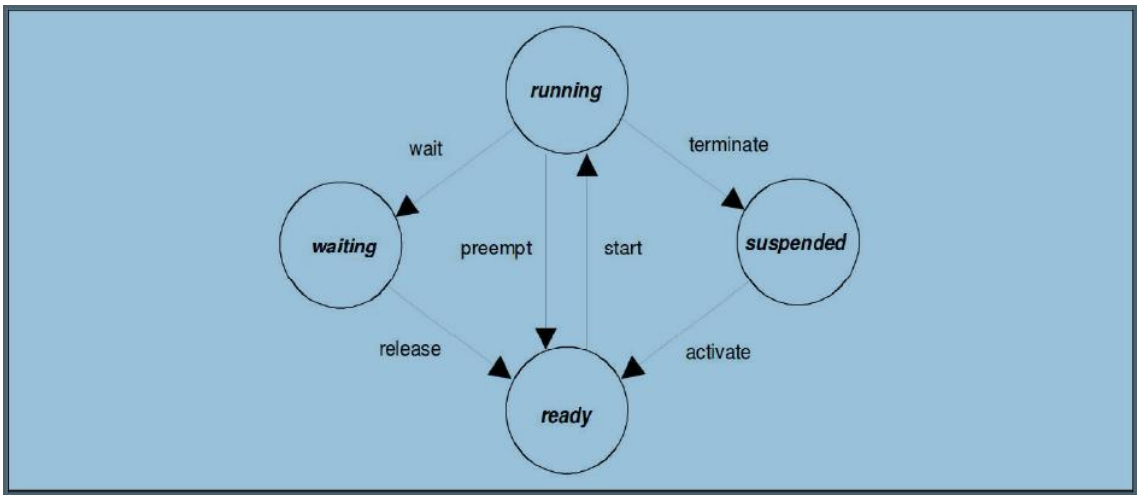
- TerminateTask: termina una tarea.

Estados

Cada tarea en OSEK se encuentra en uno de los siguientes 4 estados:

- Running: la tarea se encuentra corriendo, está utilizando recursos del procesador. En cada momento, solo una tarea puede estar en este estado.
- Ready: aquí se encuentras las tareas esperando los recursos del procesador para poder ejecutarse. No están en running porque una tarea de mayor prioridad esta corriendo.
- Waiting: la tarea estaba corriendo y paso a un estado de "espera".
- Suspended: es el estado por defecto de las tareas, desactivadas.

Diagrama de transición de estados



Explicación del diagrama

Transición	Estado anterior	Estado futuro	Descripción
Actívate	Suspended	Ready	Una nueva tarea es activada y puesta en la lista de ready para correr. Esta transición se puede realizar mediante: - ActivateTask - ChainTask
Start	Ready	Running	Una tarea es llevada a running de forma automática cuando es la tarea de mayor prioridad en la lista de ready.
Wait	Running	Waiting	La tarea es llevada a este estado para esperar la ocurrencia de un evento, mediante: - WaitEvent
Realease	Waiting	Ready	Al ocurrir el evento que una tarea esperaba, esta pasa a ready. Esto ocurre mediante: - SetEvent
Preempt	Running	Ready	Una tarea que estaba corriendo es desactivada, ya

			que una tarea de mayor prioridad se encuentra en ready y la tarea actual tiene un scheduling FULL o se llama a: - Schedule
Terminate	Running	Suspended	La tarea termina su ejecución, esto lo puede llevar a cabo mediante: - ChainTask - TerminateTask

Tipos de tareas

El sistema OSEK-VDX permite dos tipos de tareas, BASIC y EXTENDED. Las tareas básicas no tienen eventos y por ende no pueden permanecer en el estado waiting. A diferencia de las extendidas que pueden tener uno o más eventos y esperar.

Prioridades

Las prioridades definidas en cada tarea en el OIL son un numero entero entre 1 y 255. Mayor sea el numero mayor será su prioridad. Si varias tareas poseen la misma prioridad serán ejecutadas según su orden de llegada (FIFO). Una tarea que esta ejecutándose nunca será interrumpida por una de menor o igual prioridad, estas deberán esperar que finalice o que pase al estado waiting.

Scheduling

OSEK-OS provee dos formas de planificación que se pueden configurar a cada tarea:

- NON PREEMPTIVE
- PREEMPTIVE

Las primeras nunca pueden ser interrumpidas por otra tarea, sin importar la prioridad. Estas se ejecutan sin interrupción hasta que terminan, pasan a waiting esperando un evento o llaman a la función Schedule, la cual es una alternativa para forzar al OS de verificar si hay tareas de mayor prioridad por correr.
Las tareas NON PREEMPTIVE se utilizan por lo general en dos situaciones:

- Tareas de corta ejecución.
- Sistemas determinísticos.

Recursos

OSEK cuenta con interfaces que permiten gestionar los recursos utilizados por las tareas:

- GetResource
- ReleaseResource

Al igual que el resto de los parámetros, estos deben ser declarados en el OIL.

Alarmas

Son utilizadas para realizar acciones luego de un tiempo determinado. Las mismas pueden realizar las siguientes acciones:

- Activar una tarea.
- Setear un evento de una tarea.
- Llamar una callback en C.

Para la implementación de las alarmas el sistema necesita de un contador de hardware el cual es definido en el OIL.
Para manipular las alarmas el sistema brinda las siguientes funciones:

- SetRelAlarm
- SetAbsAlarm
- CancelAlarm

Aplicación del OSEK-OS

Para ayudar a la comprensión del lector, decidimos mostrar en este apartado parte del archivo OIL utilizado en nuestro proyecto junto al manejo de las tareas del programa main. Tenga en cuenta que solo se encuentran detalladas las partes relacionadas al SO, por favor recurra al firmware del proyecto en caso de necesitar leer los archivos.

Archivo OIL

```
OSEK OSEK
{
OS ExampleOS {
	STATUS = EXTENDED;
	ERRORHOOK = TRUE;
	PRETASKHOOK = FALSE;
	POSTTASKHOOK = FALSE;
	STARTUPHOOK = FALSE;
	SHUTDOWNHOOK = FALSE;
	USERESSCHEDULER = FALSE;
	MEMMAP = FALSE;
};

RESOURCE = POSIXR;
EVENT = POSIXE;
APPMODE = AppMode1;

TASK InitTask {
	PRIORITY = 1;
	ACTIVATION = 1;
	AUTOSTART = TRUE
	{
			APPMODE = AppMode1;
	}
	STACK = 512;
	TYPE = EXTENDED;
	SCHEDULE = NON;
	RESOURCE = POSIXR;
	EVENT = POSIXE;
}

TASK SwitchesTask {
	PRIORITY = 2;
	ACTIVATION = 1;
	STACK = 512;
	TYPE = EXTENDED;
	SCHEDULE = FULL;
```

```
        RESOURCE = POSIXR;
        EVENT = POSIXE;
    }

TASK PeriodicTask{
        PRIORITY = 5;
        ACTIVATION = 1;
        STACK = 512;
        TYPE = EXTENDED;
        SCHEDULE = NON;
        RESOURCE = POSIXR;
        EVENT = POSIXE;
 }

ALARM ActivatePeriodicTask{
        COUNTER = HardwareCounter;
        ACTION = ACTIVATETASK
        {
                        TASK = PeriodicTask;
        }
    }

COUNTER HardwareCounter {
         MAXALLOWEDVALUE = 1000;
        TICKSPERBASE = 1;
        MINCYCLE = 1;
        TYPE = HARDWARE;
        COUNTER = HWCOUNTER0;
};
};
```

Como se puede observar se declaran distintos elementos con sus respectivas propiedades:

- Inicializacion del SO.
- Tres Task:
  - InitTask: inicializacion
  - SwitchesTask: Mostrar mensaje por panel LED
  - PeriodicTask: Manejo del teclado y display
- Una alarma: `ActivatePeriodicTask`
- Un contador hardware.
- Un recurso PosixR
- Un evento PosixE
- Selecciona modo APPMODE1.

Parte de Archivo main del proyecto

```
Int main(void)
{
        StartOS(AppMode1);
        Return 0;
}

TASK(InitTask)
{
        // Funcion de inicio de la CIAA
        // Operaciones de inicialización de periféricos
        SetRelAlarm(ActivatePeriodicTask, 350, 100);
        TerminateTask();
```

```
}

TASK(SwitchesTask)
{
	// Funciones de mensaje
	TerminateTask();
}

TASK(PeriodicTask)
{
	// Funcion de maquina_estados();
	TerminateTask();
}
```

Explicación

La tarea InitTask será la primera en ejecutarse debido a que se declara como AutoStart. Es la primera y única en ese momento en el estado ready, por lo cual pasa a ejecutarse.
Dicha tarea realiza la inicialización de la EDU-CIAA y finalmente activa una alarma a través de la función `SetRelAlarm(ActivatePeriodicTask, 350, 100)`, la misma se activara a partir de los 350 milisegundos de ejecución y se reactivará cada 100 milisegundos. Por lo tanto, al momento de la alarma la única tarea que se ejecutara será la PeriodicTask, debido a ser la única en estado ready.
Dentro de PeriodicTask se ejecuta una maquina de estados, la cual se encarga del proceso de inicialización de los parámetros ingresados por el usuario y los periféricos correspondientes. Una vez que el usuario ingresa un mensaje y confirma su envío al panel LED se ejecuta la siguiente instrucción por única vez: `ActivateTask(SwitchesTask)`.
A partir de este momento serán dos las tareas que pasan a competir por el uso de los recursos, por lo tanto las mismas fueron configuradas para poder hacer un uso correcto de los mismos.
Siendo PeriodicTask una tarea con un scheduler NON PREEMPTIVE y prioridad 5, al momento en que la alarma se active y compita con SwitchesTask (prioridad 4 y FULL PREEMPTIVE) por el uso de los recursos, la primera pasara a ejecutarse de forma obligatoria, debido a tener mayor prioridad y que la segunda posee un scheduler que permite su interrupción (FULL PREEMPTIVE).
Queda claro entonces que la tarea PeriodicTask se ejecutará de forma obligatoria al momento de solicitar la unidad de procesamiento. Una vez que la misma finalice, tomara control de la misma la otra tarea, hasta que nuevamente se active la alarma.

De esta manera primero se ejecuta PeriodicTask, se inicializa el sistema, y una vez enviado el primer mensaje ambas tareas compiten. Logrando que mientras el mensaje correspondiente se muestre por el panel LED esta tarea pueda ser interrumpida por el ingreso de caracteres para modificar y reenviar otro mensaje, generando que la interfaz sea amigable para el usuario debido a la sensación de que ambas tareas se ejecutan “simultáneamente”.

Bibliografía

- Apunte OSEK-VDK Gustavo Cerdeiro.

Diseño de hardware

Metodología de trabajo

Al momento de comenzar el proyecto se realizó un primer esquemático para lograr una idea de cómo sería el diseño del poncho.

Si bien se podría haber comenzado con el desarrollo del hardware en ese momento del proyecto, se opto por realizar un prototipo utilizando protoboards. De ese modo, resulta mucho más fácil realizar los cambios necesarios a medida que se avanza con el desarrollo del trabajo.

Una vez que se llego a un prototipo que cumplía con las especificaciones requeridas, se comenzó el diseño de la placa de pcb y su posterior fabricación.

Diseño

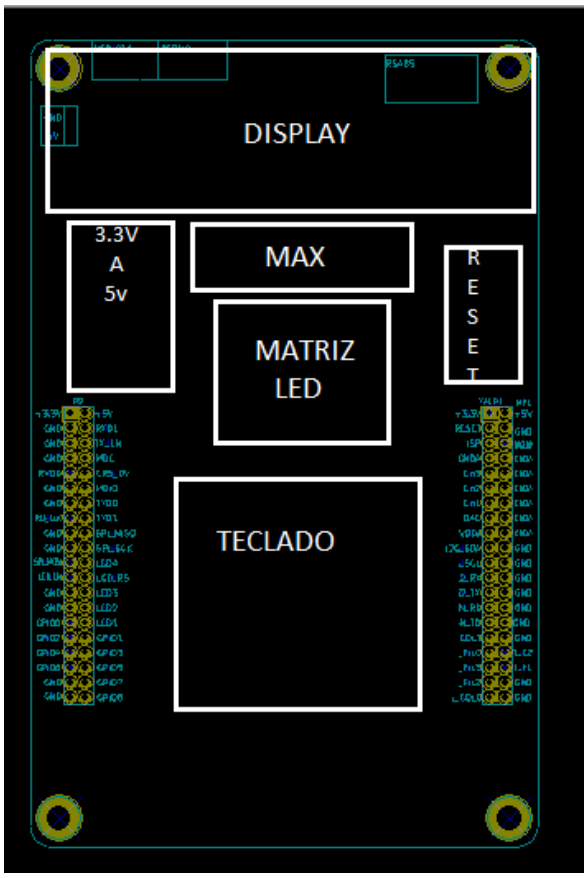
Para llevar a cabo el diseño por sugerencia de la cátedra se utilizo el programa Kicad 4.0.0.

El primer paso es realizar el diseño del esquemático, donde se incluyen todos los elementos.

En él se pueden observar los siguientes “módulos” que luego se deberán disponer físicamente en la placa de PCB de la mejor manera

- La botonera
- El display
- El max7219+matriz de led(8X8)
- El botón de reset
- Modulo conversor de 3,3v a 5v

La orientación de dichos módulos en el poncho será aproximadamente según se representa en la siguiente imagen



Los primeros ítems ya fueron explicados anteriormente en este informe, pero la inclusión de los dos restantes surgió al momento de realizar el prototipo, por los siguientes motivos:

-Botón de reset

Es necesaria su inclusión, dado que al conectar el poncho a la placa EDU-CIAA, el botón de reset que trae la misma, queda inaccesible.

-Modulo conversor 3.3 a 5v

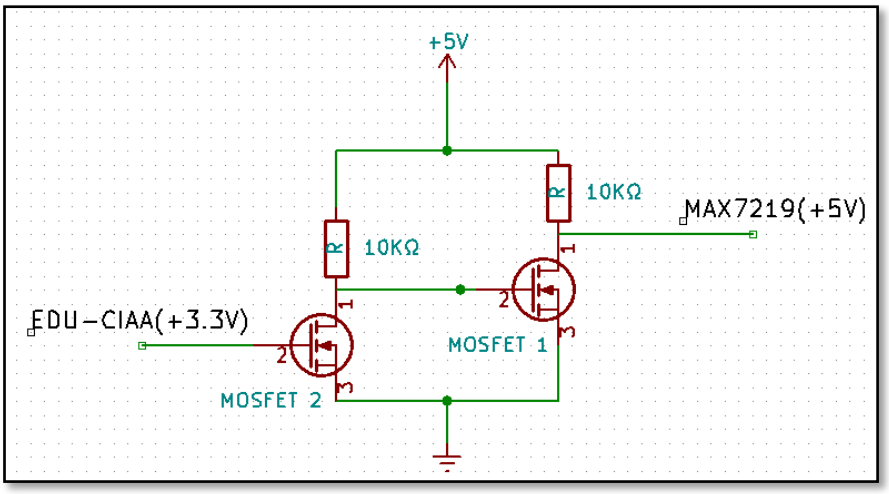
El max7219 trabaja con una tensión de 5v, y el mínimo nivel para detectar el nivel lógico es 3,5v.

Si bien la EDU-CIAA dispone de pines de alimentación tanto de 5v como de 3.3v, las líneas que conectan el integrado MAX7219 con la placa, son las correspondientes al SPI, y éstas sólo entregan 3,3 v.

Surge entonces la necesidad de convertir el nivel lógico de 3,3v a un nivel que el max7219 pueda detectar (mayor a 3,5v)

Para ello se decidió utilizar transistores mosfet 2n7000 para realizar la conversión, pero su circuito estándar, además de convertir la señal recibida, la invertía. Entonces la solución definitiva incluyo el uso de un par de MOSFET conectados en serie para negar dos veces la señal y obtener la conversión perfecta para cada pin del MAX7219.

Para cada una de las 3 líneas del SPI desde la placa EDU-CIAA hasta el MAX se realizó la conversión como se indica en la siguiente figura:



Otros aspectos importantes a tener en cuenta

De la realización del prototipo, también se desprendió un detalle muy importante a tener en cuenta para el diseño final del poncho.

Se observó que si se conectaba el backlight del display(5v), al mismo pin de la EDU-CIAA donde se conectaban los otros modulos que se alimentan con 5v , el backlight del display fallaba y parpadeaba, dificultando así la lectura del mensaje ingresado por el usuario.

Realizamos las medidas de tensión y corrientes mediante un multímetro y pudimos determinar que cada pin de alimentación de la CIAA provee 200 mA aproximadamente, mientras que nuestro sistema requiere alrededor de 250 mA.

Debido a esto, se optó entonces por utilizar el pin de 5v de la pinera 1 de la EDU-CIAA exclusivamente para alimentar el backlight del display.

Todos los otros elementos que requieran 5v por ende se conectara a la pinera 2.

Elección del modelo de poncho

Debido a la cantidad de componentes electrónicos de los que está compuesto nuestro poncho se optó por elegir el modelo de Poncho Grande provisto en los repositorios Github del Proyecto CIAA.

Si bien es importante aclarar que al momento de realizar el diseño se debió agrandar levemente las medidas de la plantilla original del poncho.

De este modo el tamaño final del Poncho_LED es de 87,6mm x 137,6 mm

Componentes a utilizar

El poncho contará con los siguientes elementos

- 13 pulsadores
- 1 display 16x2
- 10 resistencias 10k
- 1 resistencia 27k
- 1 preset 10k
- 1 capacitor electrolítico
- 1 capacitor
- 6 transistores 2n7000
- 1 matriz led 8x8
- 1 integrado MAX7219

Fabricación de la placa

Una vez realizados los esquemáticos y el diseño de la placa de pcb (ver archivos adjuntos) se procede la etapa final que es la fabricación del poncho.

Para ello se siguieron los siguientes pasos:

1- Imprimir el diseño de la placa PCB en papel ilustración.

2- Transferir el diseño utilizando agua a la placa.

3- Introducir la placa en cloruro férrico hasta que solo queden las pistas del circuito

4- Agujerear

5- Soldar los componentes

Bibliografica

- Datasheet Matriz led
  http://pdf.datasheetbank.com/datasheet-download/848836/0/ETC/1088AS
- Datasheet Max 7219
  https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf
- Datasheet Transistor 2n7000 MOSFET
  http://pdf.datasheetcatalog.com/datasheets/320/73031_DS.pdf
- http://www.proyecto-ciaa.com.ar/