v0.1

Unai Martinez Corral

umartinez012@ikasle.ehu.es

Empleo del Ordenador Personal en la Instrumentación de Panel

EUITI de Bilbao

2010-2011

Unai Martinez Corral
umartinez012@ikasle.ehu.es

Índice

Unai Martinez Corral
umartinez012@ikasle.ehu.es

Índice de figuras

Índice de tablas

Unai Martinez Corral
umartinez012@ikasle.ehu.es

1. Sobre Acher

El objetivo de este proyecto es diseñar y construir un sistema para el control de carteles de texto formados por matrices de LEDs de forma dinámica y gestionar a través de un ordenador la información que en ellas se muestra. Lo que en un principio surgió como un reto personal, se ha enmarcado en el contexto de un trabajo monográfico para la asignatura Empleo del Ordenador Personal en la Instrumentación de Panel, impartida en la Escuela Universitaria de Ingeniería Técnica Industrial de Bilbao, centro perteneciente a la Universidad del País Vasco / Euskal Herriko Unibertsitatea.

El proyecto se ha estructurado en tres partes claramente diferenciadas: adquisición, transmisión y muestra de información. Todas ellas son en realidad independientes y podrían ser sustituidas por otras de idéntica funcionalidad, sin que el funcionamiento de las otras se viera afectado. Por ello, analizaremos cada una de las partes por separado, obviando la existencia del resto, aunque especificando las condiciones necesarias para que el proyecto funcione al unirlas.

1.1. Wishlist

A continuación se listan aquellas funcionalidades que por falta de tiempo no se han implementado en esta revisión:

- Completar el contenido de CHAR_ROM para la totalidad de los caracteres ASCII (ahora sólo están las mayúsculas y algunos caracteres sueltos).
- Diseñar uno o varios PCBs para la fabricación de un sistema real, y no sólo el prototipo.
- Diseñar e implementar la lógica necesaria para permitir la programación en circuito, evitando así la necesidad de un programador externo.
- Averiguar cómo trabajar con rutas y referencias relativas en LabVIEW (variables del tipo $self o $thispath).
- Ofrecer al usuario la opción de ver en la propia aplicación lo que debería mostrarse en la matriz de LEDs.
- Implementar órdenes mediante sentencias de escape para que el usuario pueda seleccionar diferentes formas de visualización en la matriz. Para lo cual también es necesario programar diferentes rutinas, a atender por la interrupción de columna, en el microcontrolador.
- Implementar órdenes mediante sentencias de escape para que el usuario pueda seleccionar la velocidad de desplazamiento del texto.
- Probar diferentes programas libres y multiplataforma, tanto para el desarrollo de la aplicación de usuario final como para el desarrollo y depurado del programa del microcontrolador.

Se espera puedan ser añadidas en un futuro por el propio desarrollador o por terceras personas, en virtud de la licencia utilizada para la distribución del proyecto y del código fuente.

1.2. Atribuciones y agradecimientos

Para la programación del microcontrolador utilizado en el proyecto y para el diseño del circuito electrónico se ha utilizado como referencia el artículo *Funcionamiento de una matriz de LEDS*[1] publicado por Ariel Palazzesi en la revista electrónica uControl.

LabVIEW[2] es el nombre de un producto de la empresa *National Instruments Corporation*[3]. Todas las figuras de la sección Adquisición se han obtenido de *LabVIEW* o de la ejecución de programas desarrollados con *LabVIEW*.

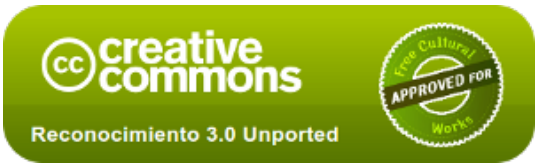
Unai Martinez Corral
umartinez012@ikasle.ehu.es

El desarrollador quiere y debe agradecer el inestimable apoyo de Iñigo Oleagordia, profesor en la EUITI de Bilbao (UPV/EHU), al haber facilitado todos los recursos, tanto materiales como bibliográficos, solicitados.

Toda la documentación del programa (este documento), a excepción de aquellas figuras capturadas directamente de *LabVIEW*, se ha creado y modificado con software libre. A continuación se citan los programas utilizados:

- LaTeX
- Gedit
- Notepad++
- Inkscape
- Dia
- Gimp
- Qucs
- KiCAD

1.3. Licencia



Este documento, a excepción hecha del código fuente, se distribuye bajo licencia Creative Commons By 3.0 (CC-by-3.0). Están permitidas la copia, distribución, y comunicación pública de la obra, así como su modificación y adaptación, siempre y cuando se reconozca la autoría mencionando a Unai Martínez Corral (pero no de una manera que sugiera que tiene su apoyo o apoya el uso que hace de su obra).

El texto legal completo está disponible en la página de la organización Creative Commons:

http://creativecommons.org/licenses/by/3.0/legalcode

El código fuente del programa se distribuye bajo licencia GPL, tal como lo indican las cabeceras de los ficheros que lo contienen. En el fichero LICENSE.txt puede encontrarse el texto legal completo. También está disponible en la página del proyecto GNU.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

2. Adquisición

De las tres partes principales que componen el proyecto, ésta es la única en la que el desarrollador no ha tenido opción de decisión. Debido a su voluntad de presentarlo como trabajo monográfico para una asignatura, se ha visto en la obligación de adecuarlo para poder relacionarlo con ésta. Por ello, se ha optado por el desarrollo de la interfaz de usuario y la gestión del envío a través de *LabVIEW*[2], software desarrollado por *National Instruments*[3] y centrado en la instrumentación electrónica.

Sin duda, y como podrá comprobarse a continuación, el consumo de recursos derivado de la adopción de esta solución resulta excesivo para la aplicación deseada. Si bien se trata de un software con gran utilidad en otros ámbitos, en éste en concreto y siempre bajo el punto de vista del autor, su uso podría definirse con la expresión "matar moscas a cañonazos". La extensa lista de herramientas, tanto gratuitas como libres y además multiplataforma, que podrían haberse utilizado resulta difícil de reproducir. Por poner algunos ejemplos, podrían mencionarse las bibliotecas GTK+[4] y Qt[5], en conjunto con cualquiera de los muchos lenguajes de programación que las soportan (C++, Java, Ruby, D, Perl, Python, PHP...). Como ya se expondrá en la sección 3, correspondiente a la transmisión, la interfaz y protocolo utilizados están ampliamente extendidos y se dispone de librerías en prácticamente la totalidad de los lenguajes.

Tras esta introducción, se presenta la aplicación en sí, creada como un VI y posteriormente empaquetada en un ejecutable, que finalmente se ha utilizado para la generación de un instalador. Se perciben dos partes muy diferenciadas[1] en ésta y así se presentarán: por un lado la aplicación base (aquella encargada de adquirir y enviar los datos) y por otra todos los aspectos estéticos que contribuyen a la apariencia del programa pero resultan inútiles desde un punto de vista operativo.

2.1. Aplicación base

La base de esta parte del proyecto es la adquisición de datos y el envío de éstos a través de la interfaz seleccionada. Como puede verse en la figura 1, todo ello se consigue mediante tres bloques específicamente desarrollados para transmitir cadenas de caracteres a través del puerto de serie. El primero de ellos, obtenido con el nombre *VISA Configure Serial Port* en el grupo *VISA* dentro del apartado *Instrument I/O* de la paleta de funciones, recibe los parámetros de configuración para definir el funcionamiento del protocolo, que más adelante se tratará en el apartado 3.2.

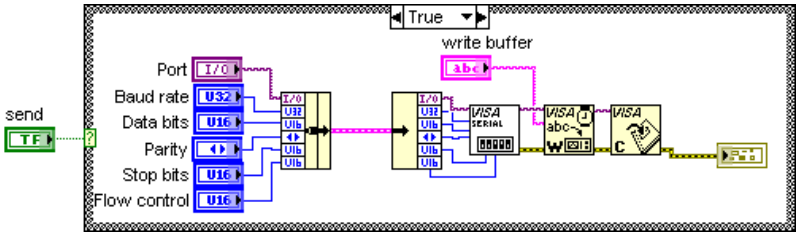


Figura 1: *Envío de datos con LabVIEW.*

Con el objetivo de hacer la aplicación lo más versátil posible, dentro de las limitaciones que su naturaleza establece, se han creado controladores para que el usuario tenga la posibilidad de configurar el tipo de comunicación, aunque están fijados por defecto los valores utilizados en este proyecto.

Para mejorar la legibilidad del código, se han unido todas las señales de los controladores en un cluster que después se ha separado a la entrada del bloque. En este caso en concreto, no supone una gran ventaja, pero es un avance de cara a futuras modificaciones.

El segundo bloque, *VISA Write*, recibe como parámetro el string a enviar. Tal como se ha diseñado, envía los caracteres del string uno a uno, de forma secuencial y una sola vez. El bloque

[1] El diagrama completo se encuentra en el Anexo A

Unai Martinez Corral
umartinez012@ikasle.ehu.es

permite la especificación de un pequeño retardo entre caracteres. Se han realizado pruebas y se ha constatado que para la aplicación que se ha desarrollado no es necesaria la inclusión de dicha funcionalidad. El string a enviar se ha definido como un controlador, pues el objetivo es que el usuario final pueda decidir qué mostrar en la matriz de LEDs, y no que se envíe siempre el mismo contenido.

El último de los bloques, *VISA Close*, cierra y termina la comunicación. La figura 1 muestra cómo se ha pasado el apuntador que indica el puerto a utilizar de un bloque a otro. Lo mismo se ha hecho con el cluster de error. Que finalmente enlaza con un *Simple Error Handler*, para que el usuario pueda tener constancia e información cuando suceda algún error en la comunicación.

2.2. Apariencia

2.2.1. Interfaz

Una vez definida la funcionalidad base de la aplicación y para mejorar la apariencia de ésta, haciendo más fácil y visual su uso, se han realizado múltiples modificaciones mediante el uso de *Property nodes* principalmente, aunque también se han utilizado otros recursos.

Se ha creado un indicador personalizado[6] para mostrar una imagen en formato PNG con transparencia. Esta imagen corresponde al logotipo del proyecto que puede verse en la figura 2. No es ésta la forma más elegante de mostrar una imagen estática, pues para eso se dispone de otros recursos específicos, pero ha resultado la más fácil y rápida desde el punto de vista del desarrollador. Se ha modificado un indicador de tipo booleano para que muestre la misma imagen independientemente del valor que se le asigne. Este indicador puede encontrarse con el nombre *logo_acher.ctl* entre las fuentes del proyecto.

El controlador de tipo string que define el contenido a enviar por la aplicación se ha configurado mediante *Property nodes* para que el texto aparezca siempre centrado y el tamaño sea mayor que el que aparece por defecto. Estos parámetros son constantes, por lo que el usuario no puede modificarlos durante la ejecución.

Figura 2: *Panel frontal: vista general.*

Se han creado cuatro botones (controladores booleanos) para que el usuario pueda interactuar con la aplicación. Éstos son *Send*, *Configure*, *Error* y *Quit*.

- **Send**:

  La función de este botón es enviar, al pulsar sobre él, el contenido del controlador de tipo string:

  Todo el programa se encuentra dentro de un bucle *while*. El valor del botón *Send* controla una estructura *case* dentro del *while*. Cuando es verdadera, se ejecuta la aplicación base

Unai Martinez Corral
umartinez012@ikasle.ehu.es

analizada en la sección anterior. Mientras sea falsa, no se ejecuta acción ninguna.

Mediante los *Property nodes* se han modificado el tamaño y color del texto y el color de fondo del botón. Este último, además, cambia cuando el botón está pulsado. El texto se encuentra en todo momento centrado tanto vertical como horizontalmente.

- **Configure**:

  Este botón muestra y oculta los controladores que definen los parámetros a utilizar por el bloque *VISA Configure Serial Port*. Para ello, actúa sobre la propiedad de visibilidad de éstos, además de sobre la misma propiedad del indicador utilizado para mostrar el logotipo y el *Simple Error Handler*. También afecta al parámetro *Disabled* del controlador string, del botón *Send* y del botón *Error*.

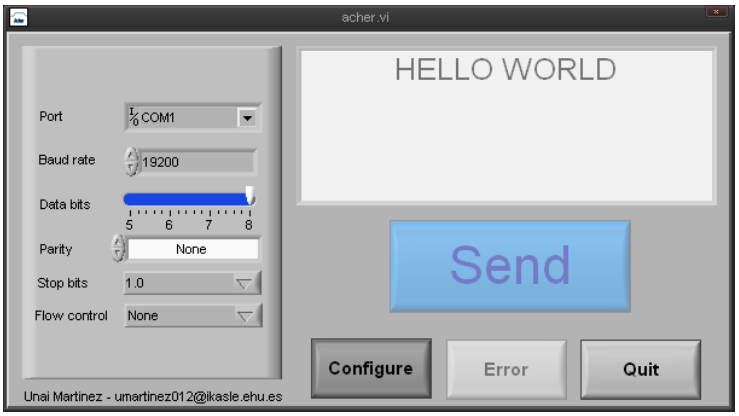


Figura 3: *Panel frontal: configure.*

De esta manera, cuando el botón se encuentra pulsado y por lo tanto su valor es verdadero (Figura 3), se muestran únicamente los controladores de configuración, deshabilitando el controlador string y los botones *Send* y *Error*. El logo y el gestor de error se ocultan. Una vez el usuario ha realizado las modificaciones oportunas, la desactivación del botón *Configure* devuelve la aplicación al estado mostrado en la vista general (Figura 2).

El tamaño del texto del botón y su alineación tanto horizontal como vertical se encuentran definidas mediante constantes.

- **Error**:

Figura 4: *Panel frontal: no error.*

El funcionamiento de este botón, muy parecido al de *Configure*, simplemente oculta el logo y muestra el gestor de errores cuando se encuentra pulsado (Figura 4). Se debe tener en cuenta que *Configure* tiene prioridad sobre *Error*, por lo que el primero deberá estar desactivado para poder ver el gestor de errores.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

El tamaño del texto del botón y su alineación tanto horizontal como vertical se encuentran definidas mediante constantes y mantienen su valor durante toda la ejecución. No así el color de fondo. Cuando se detecta algún error en el envío, éste cambia a rojo (Figura 5), indicando así que no se ha efectuado correctamente.

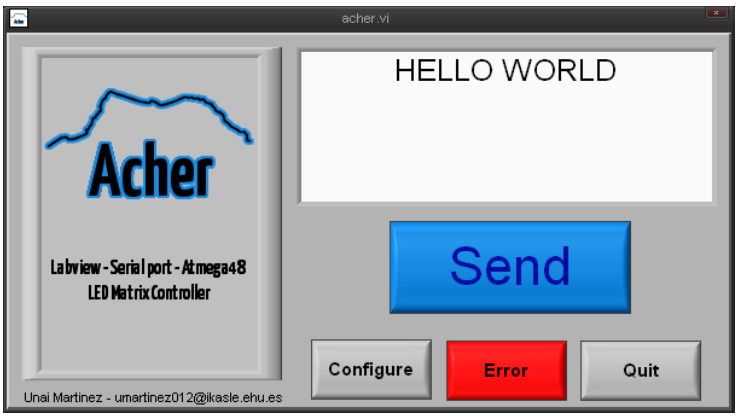


Figura 5: *Panel frontal: error.*

Pulsando sobre el botón, se puede ver cuál es el error (Figura 6). Para diferenciar cuándo se encuentra pulsado y cuándo no, en la nueva definición de color se han especificado dos constantes diferentes, siendo el segundo ligeramente más oscuro. Un simple *select* controlado por el estado del botón booleando *Error* decide cuál utilizar.

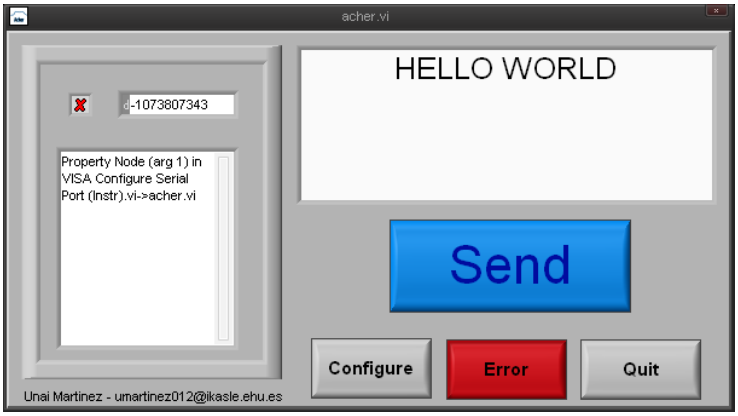


Figura 6: *Panel frontal: view error.*

Al realizar un nuevo envío correcto, y por lo tanto no haber ningún error en el gestor de errores, el botón adquiere de nuevo el color original.

- **Quit**:

  Para evitar que el usuario salga de la aplicación por error, la pulsación del botón *Quit* controla una estructura *case*. Cuando el valor es falso, éste se pasa directamente al controlador que termina el bucle *while*, permitiendo que la aplicación siga ejecutándose. Cuando es verdadero, se lanza una ventana que pide confirmación al usuario (Figura 7).

Figura 7: *Ventana de salida.*

Unai Martinez Corral
umartinez012@ikasle.ehu.es

El lanzamiento de esta ventana se efectúa mediante el bloque *Three Button Dialog*. Éste recibe como parámetros el contenido de los tres botones, el título de la ventana, el texto a mostrar y la alineación (Figura 8). Un bloque de comparación hace que la salida sólo sea verdadera cuando el usuario pulse sobre el botón de la izquierda (*Quit*). Cualquiera de las otras opciones (*Return* o *Cancel*), devolverá una salida falsa, permitiendo que se siga ejecutando el bucle principal.

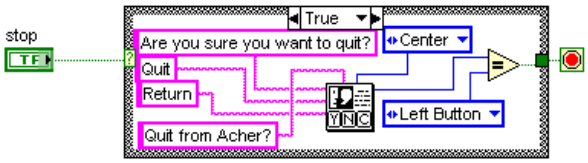


Figura 8: *Lógica de salida.*

2.2.2. Icono, patrón y ventana

El icono y el patrón del VI se han personalizado para adecuarlos al proyecto. El icono reproduce el logotipo[2], mientas el patrón indica que no hay ninguna salida ni entrada, pues se trata de un VI que trabaja de forma autónoma.

Por otra parte, en las propiedades del VI se ha configurado como de tipo *ventana de diálogo* para que no aparezcan los controles propios de *LabVIEW* durante a ejecución. Además, el tamaño de ésta se ha adecuado al del panel frontal.

2.3. Ejecutable e instalador

Con el fin de facilitar el uso de la aplicación y para favorecer su portabilidad (entre equipos, no entre plataformas), se ha creado un ejecutable a partir del VI desarrollado. Para realizar este proceso, simplemente se ha creado un proyecto[3] y, tras realizar las selecciones pertinentes (icono, nombres, rutas, etc.), se ha construido mediante el *Application Builder* del propio *LabVIEW*. Una vez obtenido el ejecutable, y haciendo uso de la misma herramienta[7], se ha construido un instalador. En éste, además del ejecutable, el icono y otras opciones de personalización, se han añadido *NI LabVIEW Run-Time Engine 2009* y *NI-VISA Runtime 4.5*, un conjunto de librerías y drivers que permiten la instalación y ejecución del programa en aquellos equipos donde no esté instalado *LabVIEW*.

El resultado final es un instalador de aproximadamente 150MB que nos garantiza la posibilidad de ejecutar el programa en cualquier equipo con un sistema operativo compatible y un puerto serie. Las condiciones de distribución y utilización se rigen por la licencia de *LabVIEW*, pues es el software sobre el que corre la aplicación.

2.3.1. Cierre del VI

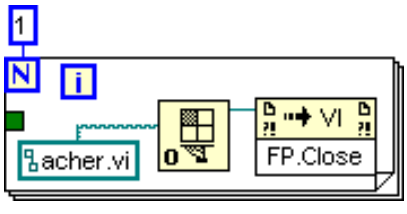


Figura 9: *Lógica de cierre del VI.*

[2]El icono está disponible en las fuentes con el nombre *acher_icon.ico*
[3]El fichero del proyecto está disponible en las fuentes con el nombre *acher.lvproj*

Unai Martinez Corral
umartinez012@ikasle.ehu.es

Debido a las características de ejecución de *LabVIEW*, una vez terminado el bucle *while*, es decir, cuando el usuario haya pulsado el botón *Quit* y confirmado la acción en la ventana emergente, el VI se mantiene abierto, aunque parado. Aunque ésta es una característica muy útil (prácticamente imprescindible) en el proceso de desarrollo, resulta inadecuada en la aplicación final. Por un lado porque el usuario se ve obligado a cerrar una ventana más, a pesar de haber solicitado y confirmado el cierre de la aplicación. Por otro, porque una vez ha terminado la ejecución, e independientemente del tipo de configuración que se haya seleccionado en las propiedades del VI, se muestran menús propios de *LabVIEW*. Éstos pueden producir confusión en el usuario, al desconocer su funcionalidad y la razón por la que aparecen. Además, en principio, no es de su interés el software utilizado para el desarrollo de la aplicación.

Para forzar el cierre del VI[8] una vez se ha confirmado por parte del usuario, se ha añadido un bucle *for* de una sola iteración (que actúa como un *frame* secuencial) a continuación del *while* principal (Figura 9). Dentro de éste encontramos un bloque del tipo *Open VI Reference* al que se pasa como argumento el nombre del propio VI y un bloque *Invoke Node* con el método *Close FP* seleccionado que recibe el apuntador creado por el anterior.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

3. Transmisión

Como interfaz de comunicación entre el ordenador personal y el microcontrolador encargado de controlar la matriz de LEDs, se ha optado por el puerto serie. La elección se ha basado en la sencillez y extensión de uso de éste. Dado que la velocidad de transmisión no resulta crítica en el sistema, no se ha considerado necesaria la implementación de otras interfaces más rápidas que conllevan el uso de conectores más grandes o de protocolos más complejos en su gestión.

Se trata de un puerto compatible con el estándar *RS232* o *EIA232 Standard*, cuyo uso con el conector DB-9 es un estándar de facto en las placas de desarrollo con microcontroladores, debido a que la gran mayoría de éstos integran periféricos específicos para la transmisión/recepción serie asíncrona.

Pese a que cada vez menos equipos de sobremesa incorporan puertos serie externos y que prácticamente ningún portátil los tiene, no es difícil encontrar adaptadores y extensores disponibles en el mercado. El autor considera que este contratiempo es un mal menor en comparación con las ventajas que esta interfaz ofrece.

Señal		**DB-9**
Common Ground	G	5
Transmitted Data	TD	3
Received Data	RD	2
Data Terminal Ready	DTR	4
Data Set Ready	DSR	6
Request To Send	RTS	7
Clear To Send	CTS	8
Carrier Detect	DCD	1
Ring Indicator	RI	9

Tabla 1: *Conexiones DB-9 según el estándar RS232, desde la perspectiva del DTE.*

De los nueve contactos que define el estándar para el conector (Tabla 1), sólo se han utilizado el contacto de masa común (5) y el de transmisión de datos (3), pues la comunicación será unidireccional. Si se quisiera que el microcontrolador enviara información al ordenador personal, se utilizaría también el contacto 2, recepción de datos.

3.1. Interfaz física

Las especificaciones de la norma en lo que a los niveles lógicos se refiere indican que la tensión para un cero lógico debe estar entre +3v y +15v; entre -3v y -15v para un uno lógico. De estos valores se deduce que los datos se transmiten con lógica negativa, al contrario que los microcontroladores habitualmente.

Por otra parte, los puertos de los microcontroladores no admiten ese rango de tensiones, pues están preparados para trabajar con señales TTL compatibles.

Por lo tanto, es necesario adecuar la tensión a valores TTL compatibles e invertir el valor lógico de los datos. Esto último podría omitirse si se implementara el protocolo a mano en el microcontrolador, o si el periférico dedicado a la comunicación serie de éste permitiera trabajar con lógica negativa. Para garantizar la compatibilidad con el mayor número de dispositivos posible, se obviará la omisión.

En el mercado se dispone de muchos circuitos que facilitan la conversión RS232-TTL y viceversa. Destaca especialmente el MAX232[9] de *MAXIM*, que realiza las conversiones alimentándose únicamente de una fuente de +5v y utilizando cuatro condensadores externos para generar las tensiones necesarias.

Este chip pone a disposición del diseñador cuatro drivers, dos RS232-TTL y dos TTL-RS232.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

Aunque para este sistema en concreto sólo se ha utilizado un driver RS232-TTL, el uso de este integrado, y su correcto cableado, permitirá reutilizar el montaje para futura aplicaciones.

De acuerdo con las especificaciones disponibles en la hoja de características del circuito integrado, además de los cuatro condensadores para generar las tensiones RS232 compatibles, se ha necesitado un quinto para estabilizar la alimentación. Teniendo en cuenta la aplicación que se está tratando, todos ellos tienen una capacidad de 1μF.

3.1.1. Esquema electrónico

Asumiendo que sólo se ha utilizado uno de los drivers, no se ha prestado atención a los contactos asociados a los otros tres, pues su funcionamiento no afecta al sistema. El resto de contactos se han cableado según las indicaciones de la hoja de características, tal como ilustra la figura 10.

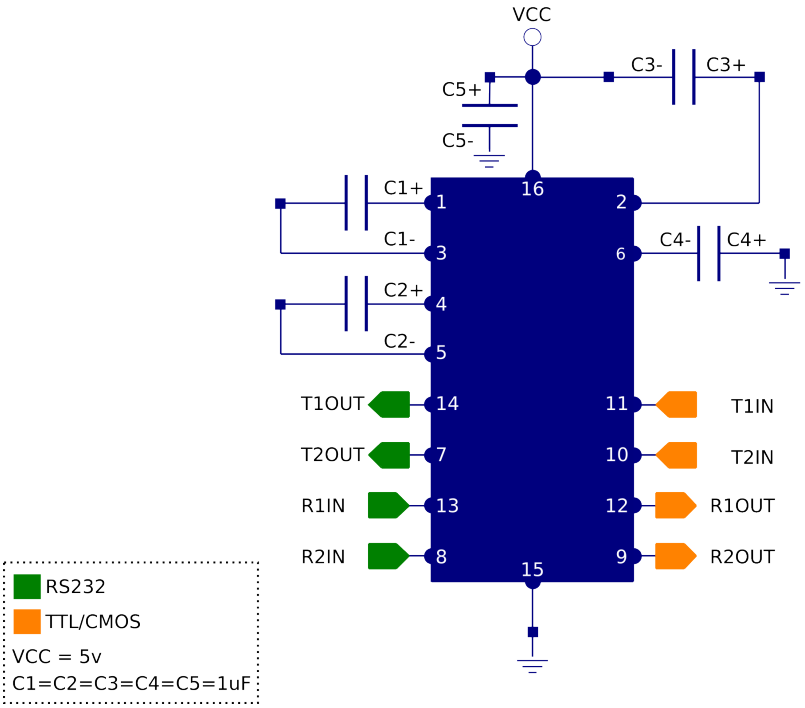


Figura 10: *Esquema de conexionado del circuito integrado MAX232.*

3.2. Formato de la trama

La norma RS232 comprende varios protocolos de entre los cuales cabe destacar la diferenciación entre *síncronos* y *asíncronos*. Los primeros requieren el uso de una señal de reloj que, gestionada por el maestro en la comunicación, sirve para sincronizar el envío de datos entre los componentes. En el caso de los asíncronos la transmisión se basa en el determinación de la temporización a ambos lados, de forma que se puede utilizar una sola línea modificando los datos con una frecuencia acordada. Debido a que el número de conectores suele ser un bien preciado en los microcontroladores, se ha optado por utilizar una comunicación asíncrona, reduciendo de tres a dos el número de contactos necesarios para una comunicación unidireccional.

Atendiendo al protocolo establecido por la norma RS232 para las comunicaciones asíncronas sin tiempo de inicio preestablecido, la información se envía, como ilustra la figura 11, estructurada en cuatro partes:

- START: bit de inicio o arranque. Inicia la comunicación cuando la línea, que en estado de reposo se encuentra en un uno lógico, pasa a un cero lógico. A partir de este flanco, el receptor

Unai Martinez Corral
umartinez012@ikasle.ehu.es

comienza a leer las señales con una frecuencia preestablecida.

- DATA: bits de datos. Se envía el número de bits preestablecido, empezando por el de menor peso, LSB (*Least Significant Bit*), y terminando por el de mayor peso, MSB (*Most Significant Bit*).
- PARITY: bit de paridad. Si se ha preestablecido la transmisión de un bit de paridad para comprobar la integridad de los datos, se enviará un uno o un cero lógico en función del tipo de paridad (par o impar) y la palabra enviada.
- STOP: bit de parada. La línea se mantiene en un uno lógico cuando termina la transmisión. Este bit, que puede mantenerse el equivalente a 1, 1.5 o 2 bits, indica la finalización de la transmisión.

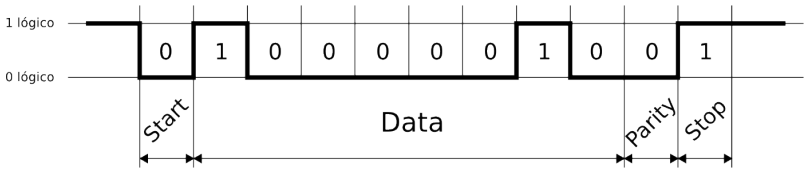


Figura 11: *Ejemplo de envío de una trama según RS232. Código ASCCI de "A" (01000001), 8 bits de dato con paridad par y 1 bit de stop.*

Dado que los datos a transmitir serán de tipo char y la integridad de los datos no resulta crítica, en este proyecto se ha optado por implementar una comunicación con un bit de start, ocho bits de dato, ninguno de paridad y un bit de stop. Éstos serán los datos a tener en cuenta por la aplicación que transmita los datos desde el ordenador, y la configuración del receptor del microcontrolador.

3.3. Velocidad de transmisión

La unidad más utilizada para expresar la velocidad de transmisión, la cantidad de información enviada, es el *baudio* [10]. Proporcional a los bits por segundo (bps), expresa el número de bits de información. La velocidad a la que pueden trabajar tanto los puertos serie de un ordenador como los microcontroladores está normalizada a 75, 150, 300, 600, 1200, 2400, 4800, 9600 baudios, etc.[11].

Para la aplicación que nos ocupa, se ha escogido una velocidad de transmisión de 19200 baudios. Si bien podrían utilizarse velocidades mayores, nuestro sistema no requiere una alta velocidad de transmisión y este valor facilita los cálculos para la configuración del microcontrolador, como se analizará en la sección correspondiente.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

4. Muestra

Para la gestión de la matriz de LEDs se ha optado por la utilización de un sistema basado en un microcontrolador, junto con diversos drivers que faciliten la programación y la adecuación de las señales. Aunque podría haberse gestionado toda la lógica desde el ordenador, esto habría vuelto el sistema directamente dependiente de éste, lo cual impediría su funcionamiento cuando estuviera apagado. La externalización de la lógica nos permite utilizar el ordenador únicamente para enviar el contenido a mostrar en la matriz, obteniendo un sistema autónomo una vez recibida la información. Si quisiéramos mostrar información preprogramada, en lugar de definir el contenido dinámicamente, el microcontrolador nos da la opción de cargar la información durante el diseño.

Asimilando las ventajas de implementar la lógica de forma externa al ordenador, el microcontrolador se muestra como una de las opciones, entre las cuales también debemos valorar el uso de FPGAs, CPLDs o VLSIs. El diseño con VLSIs y CPLDs resulta excesivamente complejo, debido a la necesidad de efectuar operaciones aritméticas de cierta complejidad y el uso de diversos registros. El diseño se complicaría lo suficiente como para no resultar rentable por el tamaño final del circuito y el número de integrados necesarios. Las FPGAs suplen a la perfección estas necesidades, pero su alimentación resulta más compleja que la de un microcontrolador, y el hecho de disponer de memoria volátil complica la implementación física, al ser necesaria la adición de elementos de memoria para evitar tener que programarlas en cada arranque. Además, ninguna de las opciones anteriores incorpora periféricos específicos para la temporización o la comunicación serie. Estos recursos deberían ser desarrollados por el diseñador. Por estas razones, un sistema microcontrolador se muestra como la mejor opción, al aunar la simplicidad de diseño del circuito y la complejidad de recursos necesaria para esta aplicación.

De entre los muchos microcontroladores disponibles en el mercado, se ha utilizado el modelo Atmega48 de Atmel[12], con arquitectura AVR[13]. La elección de este modelo responde a la facilidad de acceso a los recursos para trabajar con él, pues el desarrollador ha tenido a su disposición un kit de desarrollo STK500[14] del mismo fabricante. El modelo citado dispone de recursos suficientes para el sistema a desarrollar y permite la evolución de este de cara a añadir funcionalidades en futuras versiones. Además, la arquitectura AVR se diseñó específicamente para la ejecución eficiente de código C compilado, lenguaje utilizado por el desarrollador.

En cualquier caso, y dado que no se hace uso de recursos sólo disponibles en la arquitectura AVR, con la debida adecuación del código podría utilizarse cualquier otro microcontrolador para el cual se disponga de un compilador de C, ya sean los muy extendidos PIC[15] de Microchip, aquellos compatibles con el 8051[16] de Intel, u otros muchos disponibles en el mercado.

4.1. Elección de componentes y esquema electrónico

Antes de abordar la elección de componentes para el circuito, se ha efectuado una división basada en la funcionalidad de éstos[4]. A saber:

- Circuito base: agrupa los componentes básicos para el funcionamiento del sistema basado en un microcontrolador, independientemente de su uso
- Recepción: agrupa los componentes necesarios para la recepción de la información según el estándar RS232, cuyo uso se ha justificado en la sección 3.
- Muestra: agrupa los componentes necesarios para mostrar la información.

4.1.1. Circuito base

Como resulta evidente, el componente principal necesario es el microcontrolador. De entre los muchos empaquetados disponibles[12], se ha escogido el PDIP de 28 contactos. Esta elección se

[4] El esquema electrónico completo se encuentra en el Anexo B

Unai Martinez Corral
umartinez012@ikasle.ehu.es

justifica por la facilidad de montaje. Debido a la naturaleza didáctica de este proyecto y puesto que es la primera vez que el autor utiliza muchos de los recursos, este empaquetado facilita el montaje de pequeños circuitos de prueba, el cambio de dichos circuitos al kit de desarrollo y viceversa, etc.

Para el correcto funcionamiento en esta aplicación en concreto, basta con alimentar el dispositivo con una tensión de aproximadamente 5v. Para ello, conectaremos una fuente al contacto 7 (Vcc) y la masa a los contactos 8 y 22 (GND). También deberemos conectar el contacto 20 (AVCC) a la tensión de alimentación, pues a pesar de no utilizar el conversor A/D, resulta necesario para un funcionamiento adecuado del circuito.

El modelo utilizado dispone de un oscilador RC interno que genera la señal de reloj (de hasta 8Mhz) necesaria para el funcionamiento del dispositivo. Aunque éste podría ser suficiente para muchas aplicaciones, la precisión no es todo lo buena que desearíamos para garantizar una comunicación serie fiable. Al tratarse de una comunicación asíncrona, las temporización cobra una importancia vital y por lo tanto deberemos disponer de una señal de reloj más estable.

Debido a que el microcontrolador dispone de los componentes y los contactos[12] (9:XTAL1 y 10:XTAL2) necesarios para amplificar y utilizarlo, y puesto que su uso está muy extendido, emplearemos un cristal oscilador externo. La frecuencia de éste, 11,0592 MHz, se ha escogido teniendo en cuenta los criterios de programación para la comunicación serie asíncrona. Pueden consultarse los detalles en la sección 4.2.

El uso de un oscilador externo de cristal exige la utilización de dos condensadores cerámicos de idéntica capacidad, tal como puede verse en la figura 12. El valor de éstos, atendiendo a la información contenida en la hoja de características del microcontrolador[12], se ha definido como 22pF.

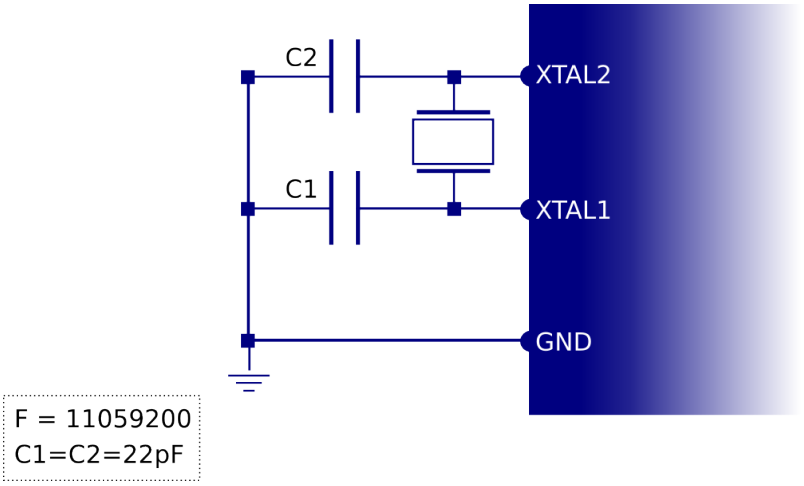


Figura 12: *Esquema de conexión del oscilador externo.*

4.1.2. Recepción

Como el microcontrolador dispone de un periférico específico para la implementación de comunicaciones serie que permite el modo de trabajo asíncrono con dos contactos asociados en el empaquetado, siempre y cuando la señales sean TTL compatibles, no es necesaria la adición de ningún componente externo.

Tal como se ha descrito en la sección 3.1, las señales provenientes del ordenador se adecuan mediante el circuito integrado MAX232[9] a niveles apropiados. Simplemente se debe conectar la salida adecuada (en este caso R2OUT, el contacto 9 del integrado) a la entrada asociada en el microcontrolador, el contacto 2 (RXD).

Unai Martinez Corral
umartinez012@ikasle.ehu.es

4.1.3. Muestra

Una matriz de LEDs no es más que una serie de LEDs dispuestos de forma matricial y garantizando que todos aquellos situados en una fila tengan un contacto en común (en el diseño desarrollado, el ánodo), y aquellos situados en una columna el otro (el cátodo). La diferencia de tensión adecuada entre el ánodo (número de fila) y el cátodo (número de columna), encenderá el LED apropiado. Gestionando la tensión en todas las filas y todas las columnas se puede controlar qué LEDs se encienden y cuáles se mantienen apagados.

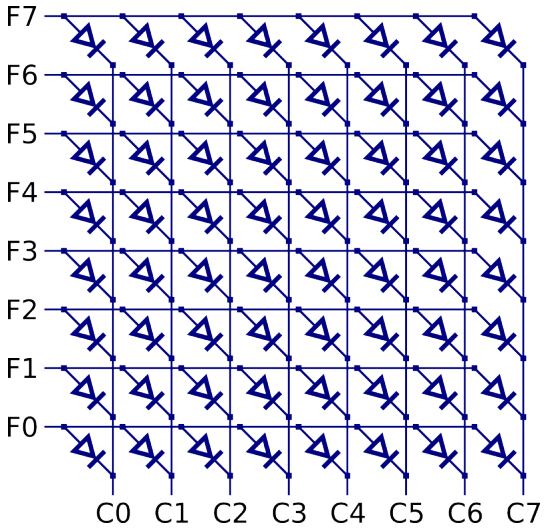


Figura 13: *Esquema de conexión de una matriz de LEDs 8x8.*

La matriz puede fabricarse mediante la correcta colocación y conexionado de LEDs discretos, tal como ilustra la figura 13. Sin embargo, el número de contactos a conectar resulta lo bastante elevado como para resultar una labor tediosa: para una matriz de 8x8 se deben conectar 8x8x2=128 contactos. Existen en el mercado matrices ya empaquetadas con sólo el número de contactos correspondiente al número de filas y el número de columnas. Para una matriz de 8x8 se tendrían 8+8=16 contactos. La comodidad y el ahorro tanto en complejidad como en tiempo sin duda obligan a valorar muy positivamente esta posibilidad.

Para el desarrollo del prototipo de este proyecto, se ha utilizado un modelo comercial de 8x8[17] del tipo columna-ánodo/fila-cátodo, debido a que era el modelo más accesible. Sin embargo, dado que la lógica se había considerado para una arquitectura donde las filas estuvieran conectadas a los ánodos, se ha optado por conectar los contactos descritos en la hoja de características como de columna a las filas, y viceversa.

La solución más rápida para gestionar una matriz de 8x8 mediante el microcontrolador consistiría en conectar cada uno de los contactos de ésta a un contacto del microcontrolador. Esto no resulta un problema para una matriz del tamaño descrito, pues disponemos de más 16 de contactos en el microcontrolador. Sin embargo, el objetivo de este proyecto es poder escalar el número de columnas para poder controlar carteles del tamaño deseado. Si se quisieran controlar 30 columnas, serían necesarios 38 contactos, de los cuales no se dispone. Es por esta razón por la que se ha optado por controlar la matriz de forma dinámica, haciendo uso de la multiplexación de las filas y utilizando registros de desplazamiento[18].

Aprovechando que el microcontrolador funciona a una velocidad mucho mayor que la que el ojo humano es capaz de apreciar, se activarán las filas una por una, en lugar de encender todas ellas al mismo tiempo. Suponiendo que se trabaja con ocho filas, cada una de ellas estará encendida 1/8 parte del ciclo total de actualización. Como el ciclo total será menor que el tiempo mínimo apreciable, la sensación será la de estar viendo todas las filas iluminadas al mismo tiempo. Asumiendo que cada fila se encenderá por separado y nunca habrá dos de ellas activas al mismo tiempo, los LEDs que se encenderán vendrán determinados por el valor que tengan los contactos de las columnas en

Unai Martinez Corral
umartinez012@ikasle.ehu.es

el momento de activación de una fila concreta. Ahí es donde cumplen su función los registros de desplazamiento: almacenarán los datos de las columnas durante el tiempo en que ésta esté activa. La comunicación entre el microcontrolador y un registro de desplazamiento es serie síncrona. Como es unidireccional, sólo necesitamos dos contactos, el de dato y el de reloj (que serán controlados por el microcontrolador), además de la masa.

De entre los registros de desplazamiento disponibles en el mercado, se ha optado por el 74HC164 [19] en empaquetado PDIP de 14 contactos. Se trata de un modelo con dos contactos de entrada serie (1:A y 2:B), a las cuales se les realiza una operación lógica AND, y salida en paralelo de ocho bits. Puesto que no se necesitan las dos líneas de datos, se han conectado ambas entradas entre sí y al contacto de salida que se ha escogido en el microcontrolador para esta función (15:PB1). Las ocho salidas se han conectado a las respectivas columnas por medio de sendas resistencias [5], tal como describe la tabla 2.

47HC164		**Matriz**
Q0	3	C0
Q1	4	C1
Q2	5	C2
Q3	6	C3
Q4	10	C4
Q5	11	C5
Q6	12	C6
Q7	13	C7

Tabla 2: *Conexiones entre el registro de desplazamiento y la matriz de LEDs.*

Los contactos de alimentación (14:VCC y 7:GND) se han conectado directamente a la fuente de alimentación. El reset global (9:$\overline{MR}$) se ha deshabilitado permanentemente dejándolo conectado a Vcc (el borrado se realizará por software al ejecutar la rutina de inicialización). El último contacto disponible, el de la señal de reloj (8:CP) se ha conectado al contacto escogido en el microcontrolador para dicha función (16:PB2).

Las salidas del microcontrolador pueden dar corriente suficiente para alimentar un reducido número de LEDs. Sin embargo, resulta claramente insuficiente a lo hora de trabajar con carteles de treinta, cuarenta o cincuenta columnas. En estos casos se debe utilizar un driver intermedio que actúe como etapa de potencia. Aunque el cartel del prototipo tiene pocas columnas, y a pesar de que en esas condiciones el microcontrolador podría alimentar los LEDs directamente, siempre es recomendable poner un driver para proteger el integrado de posible errores de montaje.

Atmega48		**Matriz**
PC0	23	F0
PC1	24	F1
PC2	25	F2
PC3	26	F3
PC4	27	F4
PC5	28	F5
PD2	4	F6
PD3	5	F7

Tabla 3: *Conexiones entre el microcontrolador y la matriz de LEDs.*

En este proyecto las salidas del microcontrolador que deberán llevar driver son aquellas conectadas a las filas de la matriz de LEDs. Para conseguir la ganancia de potencia, se han utilizado transistores bipolares PNP[20] que soportan una corriente de emisor de hasta 1.5A, lo cual permite conectar entre 75 y 100 columnas (en función de la corriente que consuman los LEDs). Al

[5] La función de las resistencias es limitar la tensión que caerá en los LEDs. El valor de éstas dependerá de la fuente con que se alimenten, como se verá al analizar la conexión de las filas.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

ser de tipo PNP, los transistores conducirán cuando la tensión en su base sea cero. Se debe tener en cuenta este hecho a la hora de realizar la programación de las rutinas del microcontrolador. Para desactivar una fila, por lo tanto, debería haber una tensión positiva en la base, lo cual se conseguirá activando la salida adecuada. Como sólo hace falta una pequeña corriente, se ha puesto una resistencia de 1K5 entre la salida y la base. El emisor de todos los transistores está conectado a la tensión de alimentación, mientras los colectores se conectan a las filas correspondientes de la matriz, de acuerdo con lo establecido por la tabla 3.

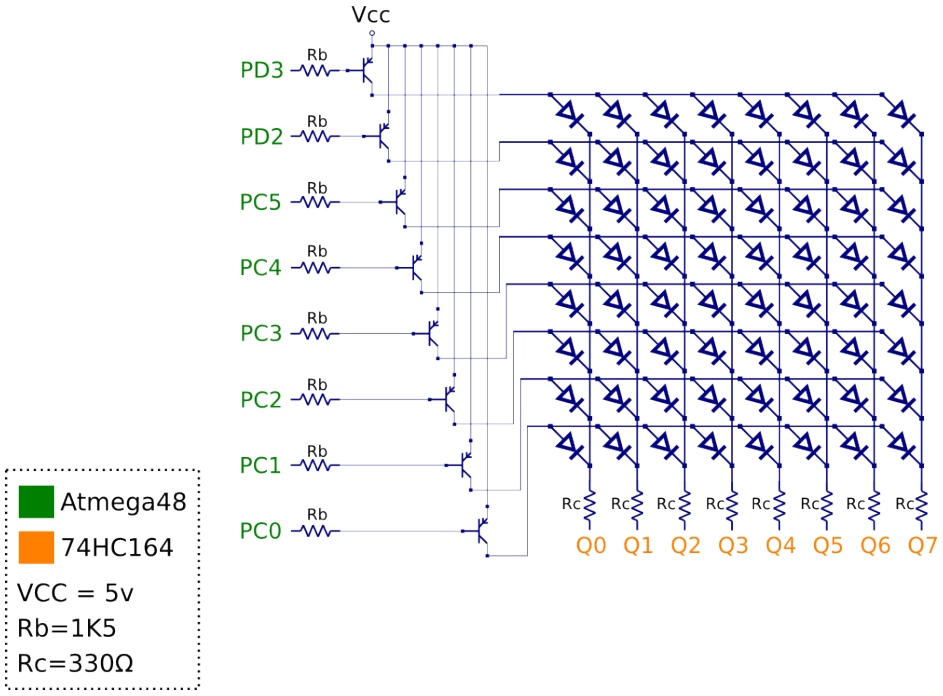


Figura 14: *Esquema de conexión de la matriz al microcontrolador y al registro de desplazamiento.*

Con la arquitectura descrita, cuando un transistor determinado conduzca y al menos una de las salidas del registro de desplazamiento esté a nivel bajo, habrá un LED que encuentre cero voltios (masa) en su cátodo y la tensión de alimentación en su ánodo. Lo más probable es que en estas condiciones el LED se queme, pues la alimentación será de aproximadamente cinco voltios. Para evitar que esto suceda, se podría sustituir la tensión en los emisores por una menor. Esto requiere dimensionar un divisor/reductor de tensión para soportar la corriente de todos los transistores. Desde un punto de vista práctico, resulta más fácil mantener ese mismo valor da alimentación e introducir resistencias entre los cátodos y las salidas del registro de desplazamiento, tal como ilustra la figura 14. Para el prototipo se han utilizado resistencias de 330Ω.

4.2. Programación

Al ser el juego de instrucciones de la familia AVR desconocido para el desarrollador y no haber tenido nunca contacto con este tipo de microcontroladores, decidió para un primer acercamiento programar utilizando el lenguaje C, más concretamente el estándar ANSI. La arquitectura AVR está soportada por el compilador libre GCC[21] y existe una implementación, también software libre, que permite utilizarlo bajo sistemas Windows, distribuida como WinAVR[22] y que incluye compilador, programador y debugger.

Pese a la existencia de herramientas libres y multiplataforma para efectuar el desarrollo completo, se ha utilizado el AVR Studio 4[23] como IDE. La elección de esta herramienta se justifica sólo por la seguridad de saber que se están utilizando recursos del propio fabricante que garantizan una alta compatibilidad con el kit de desarrollo. En cualquier caso, el uso de las herramientas de depurado y programación se escapan al objetivo de este documento, por lo que no se profundizará en

Unai Martinez Corral
umartinez012@ikasle.ehu.es

su uso.

4.2.1. Notas del desarrollador

Si bien no son necesarias para la utilización del sistema, el desarrollador considera que las siguientes anotaciones pueden resultar de interés para quién quiera entender el desarrollo completo y qué decisiones se han tomado durante el mismo.

- **Estructura de ficheros**: la programación del microcontrolador se ha dividido en tres ficheros[6]:
  - **acher_main.c**: contiene la función *main* y las rutinas de vectorización de las interrupciones.
  - **acher_config.c**: contiene las rutinas de inicialización de variables y configuración de puertos y periféricos.
  - **acher_main.h**: contiene la declaración de constantes, variables y prototipos de las funciones.
- **Mayúsculas y minúsculas**: respetando la convención de código escrito en C, se han declarado los nombres de todas las constantes en mayúsculas, mientras las variables globales y las funciones aparecen en minúsculas. Todas ellas se encuentran declaradas en el fichero 'acher_main.h'.
- **Estructura operativa de memorias**: se han utilizado tres matrices (arrays unidimensionales de tipo char/uint_8t) que actúan implícitamente como memorias:
  - **leds**[ ]: su tamaño es proporcional al número de columnas declaradas. Actúa como buffer y contiene la información a mostrar bit a bit.
  - **char_ram**[ ]: su tamaño es proporcional al límite de caracteres definido. Actúa como RAM al almacenar los caracteres recibidos a través del puerto de serie. Los caracteres almacenados en esta memoria son los que irán volcándose a leds[] en un orden establecido.
  - **CHAR_ROM**[ ]: su tamaño es fijo. Contiene la equivalencia bit a bit en grupos de 8x5 píxeles para cada carácter del estándar ASCII. En otras palabras, define la tipografía para cada uno de los caracteres.
- **Operaciones con bits**: debido a que el compilador utilizado no permite la declaración de datos de tipo bit y como el microcontrolador no dispone de direccionamiento del mismo tipo, la modificación de bits determinados de un registro se ha realizado mediante máscaras implícitas. Así, se ha utilizado la operación lógica OR para poner a uno y la operación lógica AND para poner a cero. A continuación se muestra un ejemplo:

Instrucción	Contenido de REG tras la ejecución
REG=0;	00000000
REG|=(1<<3);	00001000
REG&=~(1<<3);	00000000

  1<<3 genera una máscara equivalente a desplazar un uno tres posiciones a la izquierda, es decir, *00001000*. El operador ~ complementa el contenido, luego, se obtiene *11110111*. Las expresiones utilizadas son equivalentes a los valores binarios ahora descritos, o a sus equivalentes en cualquier otra base, por ejemplo, hexadecimal: 0x08 y 0xF7.

  Más aún, la función *_BV(x)* actúa como *1<<x*, generando una máscara con un uno desplazado x posiciones a la izquierda y el resto ceros. Por lo tanto, *_BV(5) == 1<<5 == 0b00100000*.

[6] El diagrama de flujo del programa completo se encuentra en el Anexo C

Unai Martinez Corral
umartinez012@ikasle.ehu.es

- <**avr/io.h**>: al cargar la librería *io.h* el compilador reconoce no sólo los nombres de los registros de función especial del microcontrolador, sino los nombres de los bits que éstos contienen. Aunque no está permitida la modificación de los bits directamente, sí pueden usarse sus nombres junto con las operaciones descritas en el apartado anterior. Así, para activar los bits RXEN0 y RXCIE del registro UCSR0B, se puede ejecutar la siguiente instrucción:

  UCSR0B|=(1<<RXEN0)|(1<<RXCIE0);

  Equivalente a:

  UCSR0B|=(1<<RXEN0);
  UCSR0B|=(1<<RXCIE0);

  Se trata de una práctica que se ha reproducido a lo largo de todo el programa no sólo porque permite ver más claramente qué se está haciendo, sino también porque facilita la detección de errores (es más difícil confundirse en un nombre que en un número) y vuelve el código ligeramente más portable (algunos bits cambian de posición dentro del mismo registro entre los diferentes modelos de la familia AVR).

4.2.2. acher_main.c

- **Librerías y cabeceras**:

```
1 #include <avr/io.h>
2 #include <avr/interrupt.h>
3 #include <avr/pgmspace.h>
4 #include <compat/ina90.h>
5 #include <util/delay.h>
6
7 #include "acher_main.h"
8 #include "acher_config.c"
```

  En primer lugar se han cargado las librerías necesarias para reconocer funciones y variables específicas.

  - **io.h**: reconoce el modelo de microcontrolador y permite utilizar nombres de registros específicos.
  - **interrupt.h**: reconoce el sistema de interrupciones, declara funciones y vectores de interrupción específicos.
  - **pgmspace.h**: declara funciones para permitir la declaración de constantes y variables en zonas de memoria concretas.
  - **ina90.h**: declara las funciones específicas, entre ellas _SLEEP() para entrar en modos de bajo consumo.
  - **delay.h**: declara funciones para generar retardos mediante ticks.
  - **acher_main.h**: declara todas las variables y prototipos de funciones a utilizar.
  - **acher_config.c**: se definen las funciones y rutinas de inicialización y configuración.

- **Cuerpo del programa**:

```
10 int main(void){
11 config();
12
13 while(1) {
14   SMCR|=(1<<SE);
15   _SLEEP();
16   SMCR&=~(1<<SE);
17 }
18 }
```

  La ejecución del programa sólo llama la rutina de configuración, *config()* y entra continuamente en modo de bajo consumo. Sólo "despierta" para atender a las interrupciones. Después, vuelve a "dormir".

Unai Martinez Corral
umartinez012@ikasle.ehu.es

- **Rutina de tratamiento de la interrupción por recepción de la USART**:

```
20 ISR(USART_RX_vect) {
21 if(show_lim<CHAR_NUM){ char_ram[show_lim++]=UDR0; }
22 if(char_ram[show_lim-1]==BORRAR){ show_lim=0; }
23 if(show_lim==0x00){ timer_int(0); }else{ timer_int(1); }
24 }
```

Comprueba si hay espacio libre, si lo hay guarda el dato. Si el dato es una orden de borrado, reinicia el índice y desactiva las interrupciones de los temporizadores. Paso a Paso:

- Si el índice de próximo carácter a escribir (*show_lim*) es menor que el máximo reservado (*CHAR_NUM*), es decir, si hay sitio libre en *char_ram*, guarda el dato recibido en el registro al que apunta *show_lim* dentro de *char_ram* e incrementa el índice.
- Si el dato recibido es el declarado como *BORRAR*, se reinicia el índice.
- Mientras el índice sea cero, cuando la ram esté vacía, se desactivan las interrupciones de los Timers.

- **Rutina de tratamiento de la interrupción por comparación del Timer0**:

```
26 ISR(TIMER0_COMPA_vect) {
27 uint8_t x;
28
29 line_write(0);
30
31 if (l==LINES-1) { l=0; }
32 else { l++;}
33
34 line_shadow=_BV(l);
35
36 for(x=0;x<COLUMNS;x++){ shiftr_write(line_shadow&leds[x]); }
37
38 line_write(line_shadow);
39 }
```

Carga el contenido a mostrar para cada fila y mantiene activa la fila hasta la siguiente interrupción. Paso a paso:

- Desactiva todas las filas.
- Si el índice es igual a la última fila, se reinicia. Si no, se incrementa.
- Carga en la máscara *line_shadow* el índice de la fila a mostrar.
- Envía al registro de desplazamiento el bit indicado por la máscara de cada columna almacenada en el buffer (*leds[ ]*).
- Activa la fila.

- **Rutina de tratamiento de la interrupción por comparación del Timer1**:

```
41 ISR(TIMER1_COMPA_vect) {
42 uint8_t i;
43
44 for(i=0;i<COLUMNS;i++){ leds[i]=leds[i+1]; }
45
46 if (show_col==CHAR_WIDTH){
47   leds[COLUMNS-1]=0;
48   show_col=0;
49   show_char++;
50   if(show_char==show_lim){ show_char=0; }
51 } else {
52   leds[COLUMNS-1]=pgm_read_byte(&CHAR_ROM[(char_ram[show_char]*CHAR_WIDTH)+show_col++]);
53 }
54 }
```

Modifica el contenido del buffer de acuerdo con la información presente en *char_ram*. Paso a paso:

- Desplaza todas las columnas almacenadas en el buffer (*leds[ ]*) una posición a la izquierda.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

- Si el índice de columna se sale de rango (es uno más que la última columna del carácter) se escribe null en la última columna del buffer, se reinicia el índice, y se incrementa el índice de carácter. Además, si el índice de carácter se sale de rango (apunta al primer carácter vacío de la ram) se reinicia.
- Si no hay problemas de rango, escribe la columna de *CHAR_ROM* a la que apunta *show_col* dentro del carácter al que apunta *show_char* dentro de *char_ram*.

- **Rutina de envío de dato al registro de desplazamiento**:

```
56 void shiftr_write(uint8_t byte){
57 SHIFT&=~_BV(SHIFT_CLK);
58
59 if(byte!=0){ SHIFT&=~_BV(SHIFT_DATA); }
60 else { SHIFT|=_BV(SHIFT_DATA); }
61
62 cli();
63 _delay_us(1);
64 SHIFT|=_BV(SHIFT_CLK);
65 _delay_us(1);
66 sei();
67 }
```

Envía el bit recibido al registro de desplazamiento activando convenientemente la señal de reloj. Paso a paso:

- Desactiva la señal de reloj.
- Si el dato recibido es diferente de cero, carga un cero en la línea de dato. Si es cero, carga un uno[7].
- Desactiva las interrupciones.
- Espera 1μs[8].
- Activa la señal de reloj, cargando el dato en el registro de desplazamiento.
- Espera 1μs.
- Activa las interrupciones.

- **Función de escritura en los contactos conectados a las filas de la matriz**:

```
69 void line_write(uint8_t byte){
70 PORTC=~byte;
71 PORTD=~((byte>>4)&(_BV(2)|_BV(3)));
72 }
```

Escribe el valor correspondiente a cada contacto en los puertos y bits adecuados en función de las conexiones realizadas.

4.2.3. acher_config.c

- **Rutina de configuración e inicialización**:

```
1 void config(void){
2
3 conf_ports();
4 conf_variables();
5 conf_timers();
6 conf_serial();
7
8 SMCR&=~((1<<SM2)|(1<<SM1)|(1<<SM0));
9 PRR&=~((1<<PRADC)|(1<<PRTIM2)|(1<<PRTWI));
10
11 sei();
12 }
```

[7]Por el diseño realizado en el apartado 4.1.3, para activar un LED de la matriz la salida correspondiente del registro de desplazamiento debe estar a nivel bajo. Por lo tanto, en lo que al registro respecta, estamos trabajando con lógica negativa. De ahí que invirtamos el significado del dato recibido.

[8]De no hacerlo, no podríamos garantizar el correcto funcionamiento del registro de desplazamiento, pues en la hoja de características[19] se especifica el tiempo mínimo que debe durar una transición en la línea de reloj.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

Llama a todas las subrutinas de configuración e inicialización, configura el modo de bajo consumo y activa las interrupciones. Se desactivan ADC, Timer2 y TWI durante el tiempo en que esté "dormido".

- **Configuración de los puertos**:

```
14 void conf_ports(void){
15 DDRB|=_BV(SHIFT_DATA)|_BV(SHIFT_CLK);
16 PORTB=0x00;
17 DDRC=0xFF;
18 PORTC=0xFF;
19 DDRD|=_BV(2)|_BV(3);
20 PORTD=0xFF;
21 }
```

  - Bits *SHIFT_DATA* y *SHIFT_CLK* del puerto B como salida.
  - Las salidas del puerto B desactivadas.
  - Todo el puerto C como salida.
  - Todo el puerto C activado[9].
  - Bits 2 y 3 del puerto D como salida.
  - Las salidas del puerto D activadas.

- **Inicialización de las variables**:

```
23 void conf_variables(void){
24 unsigned char n;
25
26 show_col=show_char=show_lim=line_shadow=0;
27 for(n=0;n<COLUMNS;n++){
28   leds[n]=0;
29   shiftr_write(1);
30 }
31 for(n=0;n<CHAR_NUM;n++){ char_ram[n]=0; }
32 l=LINES-1;
33 }
```

  - Borra los índices de columna, de carácter y de ram, además de la máscara de filas.
  - Borra todo el buffer y carga unos en el registro de desplazamiento.
  - Borra toda la ram.
  - El índice de fila apunta a la última, para reiniciarse en la primera interrupción.

- **Configuración de los Timers**:

```
35 void conf_timers(void){
36 TCNT0=0;
37 TCNT1=0;
38 OCR0A=LINE_LIMIT;
39 OCR1A=COLUMN_LIMIT;
40
41 TCCR0A|=1<<WGM01;
42 TCCR0B|=1<<CS02;
43 TCCR0B|=1<<CS00;
44
45 TCCR1B|=1<<WGM12;
46 TCCR1B|=1<<CS12;
47 TCCR1B|=1<<CS10;
48
49 TIFR0=0;
50 TIFR1=0;
51
52 timer_int(1);
53 }
```

  - Borra el contador de los Timers 0 y 1.

[9] Como el diseño se ha realizado con transistores PNP, éstos conducen cuando la tensión en base es nula. Activamos las salidas para que los transistores estén en corte.

- Define los límites de comparación para el Timer 0 (*LINE_LIMIT*) y el Timer 1 (*COLUMN_LIMIT*).
- Configura el modo y el prescaler para los Timers 0 y 1 (CTC, $clk_{I/O}/1024$).
- Limpia todos los flags de los Timers 0 y 1 (compA, compB, OV).
- Llama a la función *timer_int()* para activar las interrupciones de los Timers 0 y 1.

- **Configuración de la USART**:

```
55 void conf_serial(void){
56 UBRR0H=(UBR_VAL>>8);
57 UBRR0L=UBR_VAL;
58
59 UCSR0A&=~((1<<U2X0)|(1<<RXC0)|(1<<TXC0));
60 UCSR0C|=(1<<UCSZ01)|(1<<UCSZ00);
61 UCSR0C&=~((1<<UMSEL01)|(1<<UMSEL00)|(1<<UPM01)|(1<<UPM00)|(1<<USBS0));
62 UCSR0B|=(1<<RXEN0)|(1<<RXCIE0);
63 UCSR0B&=~((1<<UCSZ02)|(1<<TXCIE0));
64 }
```

Carga el valor calculado para un velocidad de 19200 baudios y configura el periférico para trabajar en modo asíncrono normal con 1 bit start, 8 bits data, no paridad y 1 bit stop. Limpia los flags de recepción y transmisión y habilita la recepción y la interrupción asociada. Deshabilita la transmisión.

- **Des/activación de las interrupciones de los Timers 0 y 1**:

```
66 void timer_int(uint8_t byte){
67 if(byte==0x00){
68   TIMSK0&=~(1<<OCF0A);
69   TIMSK1&=~(1<<OCF1A);
70 } else {
71   TIMSK0|=1<<OCF0A;
72   TIMSK1|=1<<OCF1A;
73 }
74 }
```

Si el dato recibido como argumento es cero, se desactivan las interrupciones de los Timers 0 y 1. Si es diferente de cero, se activan.

4.2.4. acher_main.h

- **Definición de constantes**:

```
1  #define LINES 5
2  #define COLUMNS 8
3  #define CHAR_NUM 40
4  #define CHAR_WIDTH 5
5
6  #define BORRAR 0x7E
7
8  #define SHIFT PORTB
9  #define SHIFT_DATA 1
10 #define SHIFT_CLK 2
11
12 #define LINE_LIMIT 25;
13 #define COLUMN_LIMIT 2700;
14
15 #define FREC 11059200
16 #define BAUDRATE 19200
17 #define UBR_VAL ((FREC/(BAUDRATE*16UL))-1)
```

Se definen:

- LINES: número de líneas de la matriz.
- COLUMNS: número de columnas de la matriz.
- CHAR_NUM: capacidad de la ram.
- CHAR_WIDTH: anchura de cada carácter (en columnas).
- BORRAR: código de borrado para la USART.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

- SHIFT: puerto al que está conectado el registro de desplazamiento.
- SHIFT_DATA: bit del puerto *SHIFT* al que está conectado la entrada de dato del registro de desplazamiento.
- SHIFT_CLK: bit del puerto*SHIFT* al que está conectado la entrada de reloj del registro de desplazamiento.
- FREC: frecuencia de la señal de reloj, a utilizar para calcular el valor para la velocidad de transmisión y para las funciones de delay.
- BAUDRATE: velocidad de transmisión deseada (en baudios).
- UBR_VAL: valor que debemos escribir en el registro de la USART para obtener una velocidad *BAUDRATE* con una frecuencia *FREC*.

- **Declaración de los prototipos de las funciones**:

```
19 void config(void);
20 void conf_ports(void);
21 void conf_variables(void);
22 void conf_timers(void);
23 void conf_serial(void);
24
25 void shiftr_write(uint8_t byte);
26 void line_write(uint8_t byte);
27 void timer_int(uint8_t byte);
```

Se declaran:

- config(): rutina de inicialización y configuración.
- conf_ports(): rutina de configuración e inicialización de puertos.
- conf_variables(): rutina de inicialización de variables.
- conf_timers(): rutina de configuración de los Timers 0 y 1.
- con_serial(): rutina de configuración de la USART.
- shiftr_write(): función para escribir un bit en el registro de desplazamiento.
- line_write(): función para escribir en las salidas conectadas a las filas de la matriz.
- timer_int(): función para des/activar las interrupciones de los Timers 0 y 1.

- **Declaración de variables**:

```
29 uint8_t show_col,show_char,show_lim;
30 unsigned char char_ram[CHAR_NUM];
31 uint8_t leds[COLUMNS];
32 uint8_t line_shadow,l;
33
34 const uint8_t CHAR_ROM[] PROGMEM= {
```

Se declaran:

- show_col: índice de columna dentro del carácter que se está cargando en el buffer.
- show_char: índice de carácter dentro de la ram que se está cargando en el buffer.
- show_lim: índice que apunta al próximo espacio libre en la ram.
- char_ram[ ]: ram donde se almacenan los caracteres recibidos a través de la USART.
- leds[ ]: buffer donde se almacena el contenido que está mostrándose.
- line_shadow: máscara para gestión de las filas.
- l: índice que apunta a la fila activa
- CHAR_ROM[ ]: juego de caracteres almacenado en la zona de programa de la memoria del microcontrolador.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

5. Bibliografía

[1] Ariel Palazzesi. *Funcionamiento de una matriz de LEDs*. Revista uControl.

[2] *LabVIEW*. National Instruments.

[3] *National Instrumens - Company*. National Instruments.

[4] Colaboradores de la Wikipedia. *GTK+*. Wikipedia.

[5] Colaboradores de la Wikipedia. *Qt*. Wikipedia.

[6] Thoric. *Custom Button Control*. National Instruments Comunity.

[7] *Building an Installer (Windows)*. National Instruments LabVIEW 8.2 Help

[8] *How Can I Close the Front Panel of My Executable Programmatically?*. National Instruments Support

[9] *MAX232, +5V-Powered, Multichannel RS-232 Drivers/Receivers Datasheet*. MAXIM.

[10] Colaboradores de la Wikipedia. *Baudio*. Wikipedia.

[11] Enrique Palacios Municio, Fernando Remiro Domínguez, Lucas J. López Pérez. *Microcontrolador PIC16F84. Desarrolo de proyectos*. Ra-Ma Editorial.

[12] *ATmega48/88/168, 8-bit Microcontroller with 4/8/16K Bytes In-System Programmable Flash Datasheet*. Atmel.

[13] Colaboradores de la Wikipedia. *AVR*. Wikipedia.

[14] *AVR STK500 User Guide*. Atmel.

[15] Colaboradores de la Wikipedia. *Microcontrolador PIC*. Wikipedia.

[16] Colaboradores de la Wikipedia. *Intel 8051*. Wikipedia.

[17] *TA23-11, 58mm (2.3 INCH) 8x8 DOT MATRIX DISPLAY Datasheet*. Kingbright.

[18] Colaboradores de la Wikipedia. *Registro de desplazamiento*. Wikipedia.

[19] *74VHC164, 8-Bit Serial-In, Parallel-Out Shift Register Datasheet*. Fairchild Semiconductor.

[20] *BD136/138/140, PNP Epitaxial Silicon Transistor Datasheet*. Fairchild Semiconductor.

[21] Colaboradores de la Wikipedia. *GNU Compiler Collection*. Wikipedia.

[22] *WinAVR*. Sourceforge.

[23] *AVR Studio*. Atmel.

[24] *AVR-GCC Tutorial*.

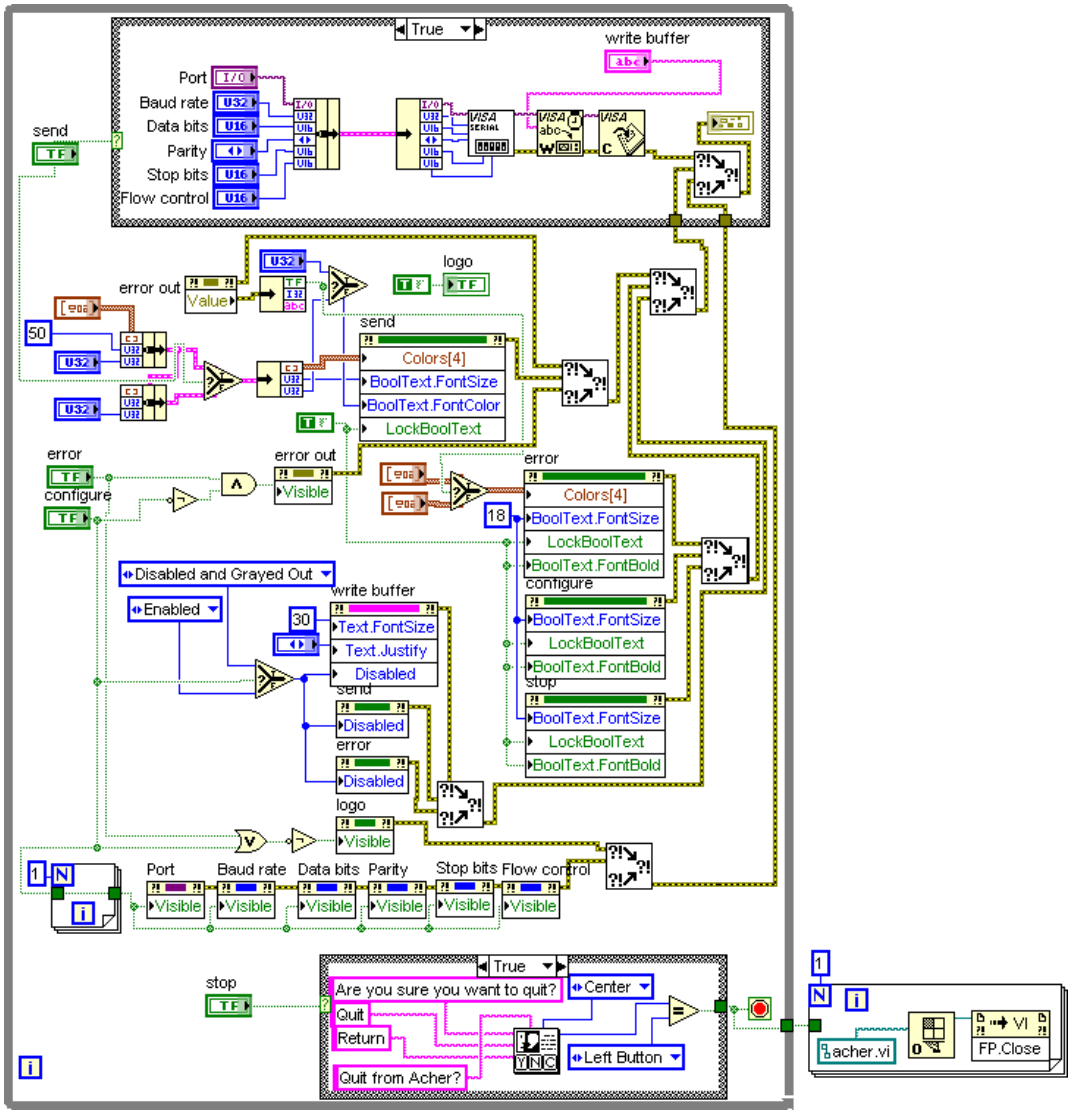
[25] AndersonMicro *Getting Started With Atmel AVR - Part II*. andersonmicro.com.

[26] Brian W. Kernighan, Dennis M. Ritchie. *El lenguaje de programación C*. Prantice Hall.

Unai Martinez Corral
umartinez012@ikasle.ehu.es

Anexo A

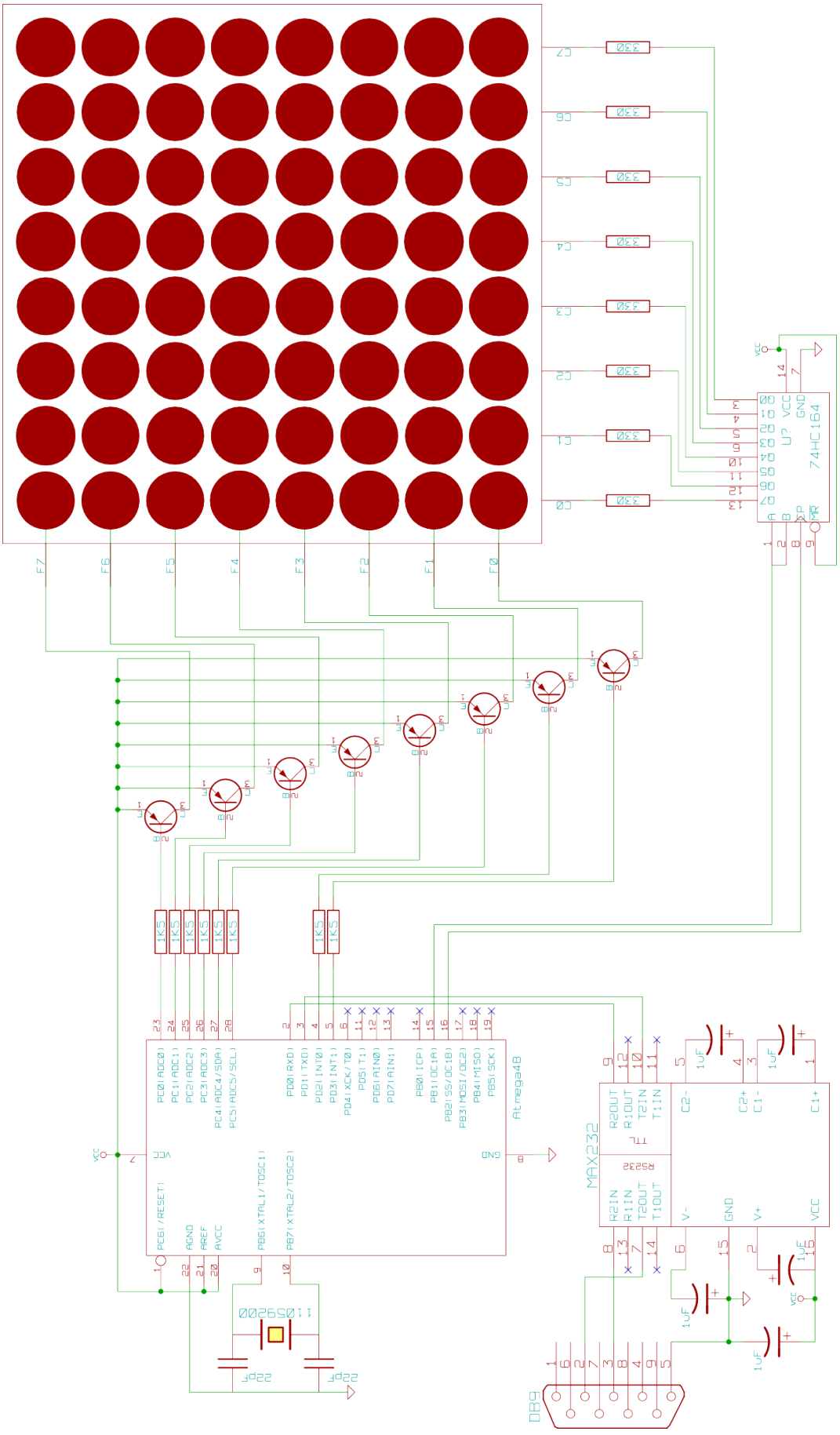
Unai Martinez Corral
umartinez012@ikasle.ehu.es



Unai Martinez Corral
umartinez012@ikasle.ehu.es

Anexo B

Unai Martinez Corral
umartinez012@ikasle.ehu.es



Unai Martinez Corral
umartinez012@ikasle.ehu.es

Anexo C

Unai Martinez Corral
umartinez012@ikasle.ehu.es