Proyecto Voyeur

Documentación

Sistemas Embebidos para Tiempo Real
Octubre 2007

Facultad de Ingeniera
U. de la R.
1. Índice

Índice de Temas
1. Índice
2. Introducción........................................................................................................................................... 3
3. Objetivos.................................................................................................................................................. 3
4. Alcance................................................................................................................................................... 3
5. Organización de la documentación........................................................................................................ 3
6. Software.................................................................................................................................................. 4
6.i. Introducción......................................................................................................................................... 4
6.ii. Arquitectura....................................................................................................................................... 4
6.iii. Tareas Implementadas...................................................................................................................... 5
6.iii.a. Tarea CommIN.................................................................................................................................. 5
6.iii.b. Tarea CommOUT.......................................................................................................................... 7
6.iii.c. Tarea Luz....................................................................................................................................... 8
6.iii.d. Tarea Seguridad............................................................................................................................ 8
7. Hardware.................................................................................................................................................. 9
7.i. Sensores Implementados.................................................................................................................... 9
7.i.a. Sensor de Luz................................................................................................................................... 9
7.i.b. Sensor de Temperatura................................................................................................................... 10
7.ii. Comunicación con el PC................................................................................................................ 10
7.iii. Microcontrolador............................................................................................................................ 11
8. Entorno de Desarrollo y Herramientas de Debuggeo.......................................................................... 11
9. Problemas Encontrados...................................................................................................................... 12
10. Conclusiones....................................................................................................................................... 12
Anexo 13
A. Placa Diseñada..................................................................................................................................... 13
B. Esquemático Completo....................................................................................................................... 14
C. Código.................................................................................................................................................. 15
Referencias................................................................................................................................................ 28

Índice de Tablas
Tabla 1 – Destinatarios y Acciones........................................................................................................... 5
Tabla 2 – Comandos al Sistema de Control de Luces............................................................................... 6
Tabla 3 – Comandos al Sistema de Electrodomésticos............................................................................. 6
Tabla 4 – Comandos al Sistema de Temperatura..................................................................................... 6
Tabla 5 – Comandos al Sistema de Seguridad......................................................................................... 6
Tabla 6 – Valores de Componentes.......................................................................................................... 13

Índice de Figuras
Figura 1 – Placa de Desarrollo y Protoboard........................................................................................... 3
Figura 2 – Arquitectura del Software......................................................................................................... 4
Figura 3 – Diagrama de la Tarea CommIN............................................................................................... 7
Figura 4 – Diagrama de la Tarea CommOUT........................................................................................... 8
Figura 5 – Diagrama de la Tarea Luz......................................................................................................... 8
Figura 6 – Diagrama de la Tarea Seguridad............................................................................................ 9
Figura 7 – Circuito Sensor de Luz............................................................................................................. 10
Figura 8 – Circuito Sensor de Temperatura............................................................................................. 10
Figura 9 – Circuito de Comunicaciones con el PC................................................................................ 10
Figura 10 – Microcontrolador y Conexiones........................................................................................... 11
Figura 11 – Placa Diseñada...................................................................................................................... 13
Figura 12 – EsquemáticoCompleto.......................................................................................................... 14
2. Introducción
El presente trabajo fue realizado como proyecto de fin de curso de la materia Sistemas Embebidos para Tiempo Real, durante su dictado en el primer semestre del año 2007.

En la Figura 1 se muestra el sistema diseñado, durante una de las pruebas realizadas. En la parte superior se encuentra la placa de desarrollo AVRDragon, a su derecha la placa de conversión de niveles para la comunicación con el PC, y debajo el protoboard con el ATMEGA32 y el resto del sistema implementado.

![Figura 1 – Placa de Desarrollo y Protoboard.](image)

3. Objetivos
Los objetivos de este proyecto se pueden dividir en dos grupos; por un lado se encuentran los objetivos académicos: utilización de algún sistema operativo en tiempo real para la solución de un problema específico y aprendizaje de conceptos básicos del lenguaje de alto nivel C. Por otro lado los objetivos personales: utilización de un microcontrolador y la realización de un proyecto de domótica.

4. Alcance
En esta primera etapa, se realizó una aproximación a un módulo de domótica, cuyas características principales apuntan a: 1) ahorro de energía; 2) facilitación de tareas rutinarias y 3) seguridad. Este módulo podrá integrarse como parte de un proyecto posterior a un sistema de domótica con mayores prestaciones, dentro de los criterios de diseño se tuvo en cuenta esta proyección.

5. Organización de la documentación
Mostraremos primero la arquitectura del software, indicando el uso y la configuración del sistema operativo en tiempo real utilizado (De aquí en adelante RTOS), luego mostraremos el hardware utilizado, hablaremos de las herramientas que se utilizaron para su desarrollo y finalmente expondremos las conclusiones junto con un breve resumen de los problemas encontrados.
6. Software

6.i. Introducción

Para la implementación de este proyecto se utilizó como RTOS el FreeRTOS, el cual cuenta con todas las funcionalidades que consideramos serían necesarias para la realización de este proyecto. De estas funcionalidades se utilizaron: 1) tareas; 2) colas de mensajes; 3) semáforos; 4) scheduler e 5) interrupciones.

De las funcionalidades con las que cuenta el FreeRTOS, no se utilizaron: 1) co-rutinas; 2) eliminación o creación de tareas en tiempo real y 3) Hooks.

6.ii. Arquitectura

En la Figura 2 se muestra la arquitectura del software implementado.

El sistema se diseñó en capas de abstracción, la primera de las cuales es la capa de abstracción del hardware, la cual se muestra como HW en la Figura 2. Esta capa se incluyó para poder intercambiar con relativa facilidad el hardware de los periféricos, es decir: conversor A/D, RS232 y sensores de temperatura entre otros. Para lograr esto se confeccionaron una serie de bibliotecas de funciones las cuales interactúan con los periféricos.

En segundo lugar se diseñaron las capas A y Mod. En dichas capas se implementaron las tareas que conforman el sistema. En la capa A se encuentran las tareas que actúan sobre los periféricos y en la capa Mod están las tareas de comunicación con el PC.

La comunicación entre tareas se realizó utilizando colas de mensajes, de modo que una tarea escucha una cola hasta que recibe un pedido de acción y mientras tanto se mantiene bloqueada y así se libera el procesador, para que el RTOS pueda disponer de él.

Los dispositivos que realizan tareas asíncronas se implementaron por medio de interrupciones, por ejemplo el conversor A/D o el RS232, de esta manera las tareas no necesitan estar consultando en busca de datos nuevos y se puede devolver el procesador mientras el periférico trabaja por su lado.

\[1\] Polling.
6.iii. Tareas Implementadas

Se implementaron varias tareas para llevar a cabo este proyecto, estas fueron: 1) CommIN; 2) CommOUT; 3) Luz, y 4) Seg.

6.iii.a. Tarea CommIN

La tarea CommIN es la encargada de la comunicación y de la decodificación de los mensajes enviados por el PC. Parte de la decodificación de los mensajes enviados por el PC hacia el sistema consiste en chequear la estructura del mismo en busca del byte de comienzo del mensaje, el Id del módulo y el byte de fin del mensaje.

De aquí en más llamaremos mensajes a las estructuras de datos que son enviadas desde el PC al módulo y viceversa, y comandos a las estructuras de datos que son enviadas entre los distintos bloques del sistema. Los mensajes se diferencian de los comandos en que los primeros son más largos pues contienen datos de arranque, parada y de identificación del módulo destinatario del mismo.

Dentro de esta tarea se implementaron las funciones de medida de la temperatura y el control de electrodomésticos, dado que la complejidad de las funciones a realizar por estas dos últimas, no ameritaba la creación de una tarea específica para cada una. Por este motivo es que los mensajes dirigidos a alguno de estos módulos son interpretados y ejecutados directamente por la tarea CommIN.

Los mensajes tienen la siguiente estructura: Start, ModID, Msj, End, donde Start es un byte de arranque de mensaje; ModID es un byte que identifica al módulo; Msj es el comando a ejecutar; y End es un byte de fin de mensaje.

Los comandos recibidos desde el PC, luego de ser procesados por CommIN, tienen la siguiente estructura: C,Cn,nn donde C denota un carácter y n un valor entre 0 y FF. Cabe aclarar que la coma no debe ir en el comando a enviar al módulo, solo se usa en la documentación de modo que resulte más cómoda la lectura, es decir el comando a enviar a través de la interfase RS232 debe ser CCnnn.

La codificación tanto del destinatario, como de la acción a realizar con el mensaje se muestra en la Tabla 1.

<table>
<thead>
<tr>
<th>Comando</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>C,xx,xx</td>
<td>Cambiar. Comando enviado hacia el módulo.</td>
</tr>
<tr>
<td>R,xx,xx</td>
<td>Leer. Comando enviado hacia el módulo.</td>
</tr>
<tr>
<td>r,xx,xx</td>
<td>Respuesta. Comando enviado hacia el PC.</td>
</tr>
<tr>
<td>a,xx,xx</td>
<td>Alarma. Comando enviado hacia el PC.</td>
</tr>
<tr>
<td>x,Ex,xx</td>
<td>Electrodomésticos. Comando enviado hacia el módulo.</td>
</tr>
<tr>
<td>x,Lx,xx</td>
<td>Luz. Comando enviado hacia el módulo.</td>
</tr>
<tr>
<td>x,Tx,xx</td>
<td>Temperatura. Comando enviado hacia el módulo.</td>
</tr>
<tr>
<td>x,Sx,xx</td>
<td>Seguridad. Comando enviado hacia el módulo.</td>
</tr>
</tbody>
</table>

Tabla 1 – Destinatarios y Acciones

El comando C,xx,xx se utiliza para realizar algún cambio en la configuración del módulo, el comando R,xx,xx se utiliza para realizar pedidos de datos al módulo, el comando r,xx,xx identifica una respuesta por parte del módulo a un pedido realizado desde el PC mediante el comando R,xx,xx, y por ultimo el comando a,xx,xx indica una condición de alarma.

Los comandos válidos enviados al sistema automático de control de luz, se muestran en la Tabla 2. En esta tabla n y d denotan valores entre 00h y FFh, mientras que x denota don’t care y puede ser cualquier valor.
<table>
<thead>
<tr>
<th>Uso</th>
<th>Comando</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sistema automático de control de luz apagado</td>
<td>C,Ln,FF</td>
<td>Enciende la luz n.</td>
</tr>
<tr>
<td></td>
<td>C,Ln,00</td>
<td>Apaga la luz n.</td>
</tr>
<tr>
<td></td>
<td>C,L0F,xx</td>
<td>Prende el sistema automático de control de luz.</td>
</tr>
<tr>
<td></td>
<td>R,Ln,xx</td>
<td>Devuelve el estado de la luz n.</td>
</tr>
<tr>
<td></td>
<td>r,Ln,xd</td>
<td>Respuesta del sistema, en d se encuentra el estado de la luz n.</td>
</tr>
<tr>
<td>Sistema automático de control de luz encendido</td>
<td>C,L0F,nn</td>
<td>Cambia el umbral inferior de nivel de luz a nn.</td>
</tr>
<tr>
<td></td>
<td>C,LFF,nn</td>
<td>Cambia el umbral superior de nivel de luz a nn.</td>
</tr>
<tr>
<td></td>
<td>C,L00,xx</td>
<td>Apaga el sistema automático de control de luz.</td>
</tr>
<tr>
<td></td>
<td>R,Lx,xx</td>
<td>Devuelve el valor actual de nivel de luz.</td>
</tr>
<tr>
<td></td>
<td>r,Lx,dd</td>
<td>Respuesta del sistema, en d se encuentra el nivel de luz medido.</td>
</tr>
</tbody>
</table>

**Tabla 2 – Comandos al Sistema de Control de Luces**

En la respuesta del sistema a un pedido para conocer el estado de la luz n, si d es 0h, la luz se encuentra apagada, mientras que si d es Fh la luz se encuentra encendida. No existen las luces 0h y Fh, estos dos valores se utilizan para prender y apagar el sistema automático de control de la luz.

Los comandos válidos enviados al sistema de electrodomésticos se muestran en la Tabla 3.

<table>
<thead>
<tr>
<th>Comando</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>C,En,00</td>
<td>Apaga el electrodoméstico n.</td>
</tr>
<tr>
<td>C,En,FF</td>
<td>Enciende el electrodoméstico n.</td>
</tr>
<tr>
<td>R,En,xx</td>
<td>Devuelve el estado del electrodoméstico n.</td>
</tr>
<tr>
<td>r,En,xx</td>
<td>Respuesta del sistema, en d se encuentra el estado del electrodoméstico n.</td>
</tr>
</tbody>
</table>

**Tabla 3 – Comandos al Sistema de Electrodomésticos**

En la respuesta del sistema a un pedido para conocer el estado del electrodoméstico n, si d es 0h, el mismo se encuentra apagado, mientras que si d es Fh esta encendido.

Los comandos válidos enviados al sistema de temperatura se muestran en la Tabla 4.

<table>
<thead>
<tr>
<th>Comando</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>R,Tn,xx</td>
<td>Devuelve la temperatura del sensor n.</td>
</tr>
<tr>
<td>r,Tn,dd</td>
<td>En d se encuentra el valor de la temperatura del sensor n.</td>
</tr>
</tbody>
</table>

**Tabla 4 – Comandos al Sistema de Temperatura**

Los comandos válidos enviados al sistema de seguridad se muestran en la Tabla 5.

<table>
<thead>
<tr>
<th>Comando</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>C,S0,xx</td>
<td>Apaga todo el sistema de seguridad.</td>
</tr>
<tr>
<td>C,SF,xx</td>
<td>Prende todo el sistema de seguridad.</td>
</tr>
<tr>
<td>C,Sn,00</td>
<td>Apaga el sensor n.</td>
</tr>
<tr>
<td>C,Sn,FF</td>
<td>Prende el sensor n.</td>
</tr>
<tr>
<td>R,Sn,xx</td>
<td>Devuelve el estado del sensor n.</td>
</tr>
<tr>
<td>r,Sn,xd</td>
<td>Respuesta del sistema, en d se encuentra el estado del sensor n.</td>
</tr>
<tr>
<td>a,Sn,xx</td>
<td>Alarma enviada por el sistema, se produjo una alarma en el sensor n.</td>
</tr>
</tbody>
</table>

**Tabla 5 – Comandos al Sistema de Seguridad**
En la respuesta del sistema a un pedido para conocer el estado del sensor \( n \), si \( d \) es \( 0 \), el sensor se encuentra apagado, mientras que si \( d \) es \( F \) esta encendido. No existen los sensores \( 0 \) y \( F \), estos dos valores se utilizan para prender y apagar el sistema de seguridad.

En la Figura 3 se muestra un diagrama de las colas implementadas y utilizadas por la tarea CommIN, las colas \( xQCommIN \) y \( xQResTemp \) son colas que se crean en esta tarea, mientras que el resto de las colas son creadas en la tarea que es propietaria de ellas.

![Diagrama de la Tarea CommIN](image)

**Figura 3 – Diagrama de la Tarea CommIN**

6.iii.b. Tarea CommOUT

La tarea CommOUT es la encargada de la comunicación hacia el PC, por parte del módulo, el mismo además de manejar la interfase RS232 entre el módulo y el PC, es el encargado de conformar el mensaje a enviar, agregando el byte de comienzo del mensaje, el Id del módulo y el byte de fin de mensaje, al comando a enviar por parte del módulo. Los mensajes válidos enviados desde el módulo al PC pueden consultarse en las tablas 1 a 5.

Con el fin de simplificar el código y la cantidad de espacio ocupado por el programa se dejó para implementar la “interfaz humana” del lado del PC, de esta manera, los mensajes enviados hacia el PC no son legibles.

En la Figura 4 se muestra un diagrama de las colas implementadas y utilizadas por la tarea CommOUT, las colas \( xQCommIN \) y \( xQResTemp \) son colas que se crean en esta tarea, mientras que el resto de las colas son creadas en la tarea que se definió como propietaria de ellas.
6.iii.c. Tarea Luz

La tarea Luz es la encargada de mantener el nivel de luz dentro del rango deseado, el cual es definido mediante dos valores, uno superior y uno inferior. Si el valor de luz es inferior al nivel inferior de luz deseado, se enciende una luz y se repite la medición del nivel de luz, si aun es inferior al nivel bajo entonces se enciende otra luz. De la misma manera se van apagando de a una las luces hasta estar dentro del rango deseado. Para evitar que el sistema quede oscilando se deberán calcular los límites de acción de forma que el apagar o prender una fuente de luz no implique un cambio mayor al tamaño de la ventana.

En la Figura 5 se muestra un diagrama de las colas implementadas y utilizadas por la tarea Luz, las colas xQPedLight y xQResLight son colas que se crean en esta tarea, mientras que el resto de las colas son creadas en la tarea que se definió como propietaria de ellas.

Esta tarea dentro de su bucle infinito, primero realiza un pedido al conversor A/D para conocer el nivel de luz actual, en caso de ser necesario solicita se prendan o apaguen luces. Luego de esto se fija si hay algún pedido para realizar alguna acción solicitada desde el PC.

6.iii.d. Tarea Seguridad

La tarea Seguridad es la encargada de detectar rupturas en las barreras de seguridad, las cuales en este caso se implementaron mediante la utilización de botones tipo normalmente abiertos. Al ser detectada una ruptura, se genera una interrupción y se envía un mensaje de alarma al PC.
En la Figura 6 se muestra un diagrama de las colas implementadas y utilizadas por la tarea Luz, las colas xQPedLght y xQResLght son colas que se crean en esta tarea, mientras que el resto de las colas son creadas en la tarea que se definió como propietaria de ellas.

7. Hardware

7.i. Sensores Implementados

Se implementaron dos sensores, uno de luz y uno de temperatura, ambos armados alrededor de un par de amplificadores operacionales. Para el sensor de luz, se utilizó un fotodiodo mientras que para el sensor de temperatura se utilizó un termistor.

Todos los componentes utilizados durante este proyecto, excepto el microcontrolador, fueron extraídos de circuitos en desuso. Incluso la fuente de alimentación de 5V y ±12V, la cual se encontraba dentro de un router en desuso.

7.i.a. Sensor de Luz

En la Figura 7 se puede ver el circuito utilizado para medir el nivel de luz. El primer amplificador operacional cumple la función de buffer, mientras que el segundo amplifica e invierte la señal obtenida del primero.

Si la variación de la corriente a través del fotodiodo es una función lineal de la cantidad de luz, entonces obtenemos un voltaje a la salida que es una función lineal de la cantidad de luz. La relación entre el voltaje obtenido a la salida y esta corriente es: \[ V(\text{out}) = \frac{R_4}{R_3} \cdot \left( R_\text{f} \cdot I_L \right). \]
7.i.b. Sensor de Temperatura

En la Figura 8 se puede ver el circuito usado para medir la temperatura. El funcionamiento del mismo, es muy similar al sensor de temperatura, por lo que no se entrara en más detalles. Solo diremos que la salida de este circuito es: \[ V(\text{out}) = \frac{R_2}{R_6} \left( 1 + \frac{R(T)}{R_5} \right) \]

7.ii. Comunicación con el PC

En la Figura 9 se puede ver el circuito utilizado para comunicarse con el PC a través del puerto serie. El mismo fue proporcionado por el equipo docente y funciona en torno a un adaptador de niveles MAX232.
7.iii. Microcontrolador

En la Figura 10 se pueden ver las conexiones realizadas en torno al microcontrolador. Tanto el encendido de electrodomésticos, como el encendido de luces se simuló mediante la utilización de led’s. En tanto que la simulaciones de los sensores de seguridad se hizo mediante botones.

Además de todo lo mostrado anteriormente se agrego un capacitor de 100nF próximo al microcontrolador para eliminar posibles ruidos en la fuente de alimentación. Pese a que estos no se esperaban dado que no estábamos realizando operaciones con los puertos del mismo a grandes velocidad fue colocado por precaución. En la Figura 11 se muestra el sistema completo.

8. Entorno de Desarrollo y Herramientas de Debuggeo

Durante la etapa de desarrollo se utilizaron tres niveles de debuggeo.

En el primero se utilizó el IAR para escribir el código en C, este entorno permite además debuggear antes de volcar al microprocesador y de esta forma evitar una gran cantidad de errores, sobre todo en lo que respecta a sintaxis y manejo de datos.

Como segunda etapa se debuggeo la interacción con los periféricos y para ello se utilizó el AVR Dragon. El AVR Dragon es una placa de desarrollo “in circuit debugger”. Con esta herramienta es posible testear el funcionamiento del sistema utilizando directamente los periféricos del microprocesador sin necesidad de “volcar” el código.

Como última etapa del debug se utilizó la comunicación con el PC, es decir, habiendo implementando la comunicación RS232 se incluyeron dentro del código algunas trazas de modo de conocer el contenido de los mensajes y variables entre otros.
9. Problemas Encontrados

La mayor parte de los problemas encontrados fueron debido al software, y en particular al RTOS utilizado.

El FreeRTOS no cuenta con una buena documentación, lo que dificulta mucho su uso. La única ayuda con la que contamos es la página web de dicho RTOS, la cual es más una referencia de funciones implementadas que una documentación de uso del mismo. Los problemas con los que nos enfrentamos durante el uso del FreeRTOS fueron: 1) tuvimos problemas a la hora de asignar prioridades, no siempre la tarea de más alta prioridad era la que era ejecutada por el scheduler; 2) la cantidad de memoria usada por las colas de mensajes implementadas hizo que en un momento no pudiéramos correr el sistema en el microcontrolador; 3) el FreeRTOS usa el Timer0 para la generación de los Tic’s del sistema, lo cual solo queda claro leyendo el código de las rutinas de atención a las interrupciones que el FreeRTOS trae por defecto; 4) el AVRStudio no ve los registros de la uart del microcontrolador durante el debug.

Otros problemas surgieron por las herramientas utilizadas; 1) la licencia del IAR que podíamos usar para compilar este proyecto solo dura 14 días. Dado que el código es mayor a 4kb, no se puede usar la versión lite que permite compilar hasta 4kb, mientras que la otra versión no limitada en cantidad de código generado, tiene una duración de 14 días; 2) tuvimos algunos problemas de origen desconocido a la hora de utilizar la placa AVRDragon, algunas de las posibles causas son, falso contacto en los cables de la interface JTAG, falta de corriente necesaria en el puerto USB, lo que hacía que al tratar de comunicarse con el PC el sistema se bloqueara.

El sistema de control de luz automático oscila, dado que simplemente se implementaron dos niveles de luz y un control de tipo on-off, por lo que en ciertas condiciones el sistema prende y apaga una de las luces en forma intermitente. Esto sería solucionable mediante algún algoritmo de control y verificación de esta oscilación.

Los sistemas de medida de luz y temperatura no pudieron ser calibrados, pero consideramos que en esta etapa no era algo estrictamente necesario. En el caso del sistema de medida de temperatura, la calibración es algo que no sería demasiado complicado, pero implica el control de la temperatura en algún recinto aislado para poder llevarla a cabo, lo cual escapa a los objetivos de este proyecto. En el caso del sistema de medición de luz, ya no es tan sencillo.

10. Conclusiones

En general se cumplió con el diseño proyectado logrando alcanzar todos los objetivos planteados inicialmente.

En lo que respecta a objetivos académicos se trabajó satisfactoriamente con el FreeRTOS. A la hora de diseñar el código del proyecto, se observó que el uso de un RTOS simplifica el trabajo. El dividir la estructura en tareas que colaboran entre sí resulta en un código más limpio y fácil de debuggear. Si bien existe un compromiso de diseño entre la cantidad de líneas de código que agrega el RTOS y la simplicidad final del código el resultado se evaluó como positivo. Tampoco se observó un aumento significativo en el uso de la memoria de programa debido a la programación en C en lugar de hacer directamente en assembler, y el hecho de utilizar un lenguaje de alto nivel simplifica las tareas, tanto de escritura del código como de debuggeo.

Por otra parte en lo que respecta a la depuración del código resultó de gran ayuda contar con una herramienta de “in circuit debugger”. Esta herramienta permite probar cosas que con los simuladores resulta imposible, entre ellas se encuentra el manejo de la USART y por consiguiente la comunicación con el PC.

En cuanto al los objetivos del diseño se logró armar y probar un dispositivo real que funciona según lo esperado por el grupo, en tiempo y forma. En lo que respecta a funcionalidades se pudieron incluir los sensores de temperatura y de luz, los que en principio planteaban la mayor dificultad. En lo que tiene que ver con el diseño del firmware se logró hacer una estructura en capas de abstracción que permitirá sustituir el hardware que actúa sobre el exterior de una manera sencilla.
Anexo

A. Placa Diseñada

Se diseñó también una placa en la que se podría colocar el sistema completo, excepto por el MAX232, la misma se muestra en la Figura 11 y la misma fue diseñada con OrCAD v10.5. En la Tabla 6 se muestra la función de cada uno de los conectores utilizados y los valores de los componentes.

<table>
<thead>
<tr>
<th>Componente</th>
<th>Valor</th>
<th>Comentarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>150kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>R2</td>
<td>1.2kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>R3</td>
<td>1.0kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>R4</td>
<td>100kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>R5</td>
<td>1kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>R6</td>
<td>2.2kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>R7 – R12</td>
<td>10kΩ</td>
<td>1/4W, 5%</td>
</tr>
<tr>
<td>U1, U2</td>
<td>NE4558</td>
<td>Amplificador operacional</td>
</tr>
<tr>
<td>U3</td>
<td>ATMEGA32</td>
<td>Microcontrolador</td>
</tr>
<tr>
<td>PWR2</td>
<td>0 / +12V / -12V</td>
<td>Fuente simétrica para alimentación de los AO.</td>
</tr>
<tr>
<td>PWR</td>
<td>0 / +5V</td>
<td>Fuente de alimentación del microcontrolador</td>
</tr>
<tr>
<td>Q1</td>
<td>Fotodiodo</td>
<td>Tomado de un mouse en desuso</td>
</tr>
<tr>
<td>R(T)</td>
<td>Termistor</td>
<td>Tomado de una placa en desuso</td>
</tr>
<tr>
<td>RS232</td>
<td>Conn</td>
<td>MAX232 para comunicación con el PC</td>
</tr>
<tr>
<td>RST</td>
<td>Conn</td>
<td>Botón de reset del microcontrolador</td>
</tr>
<tr>
<td>SEGR1, SEGR2</td>
<td>Conn</td>
<td>Barreras de seguridad, tipo normalmente abierta</td>
</tr>
</tbody>
</table>

Tabla 6 – Valores de Componentes

Figura 11 – Placa Diseñada
B. Esquemático Completo

En la Figura 12 se muestra el esquemático completo del sistema diseñado.
C. Código

A continuación se muestra el código generado durante la realización de este proyecto.

Archivo “main.c”

```c
//****************************************************************************
// PROYECTO VOYEUR - Archivo main.c
//****************************************************************************
//****************************************************************************
// ARCHIVOS DEL FreeRTOS A INCLUIR
#include "FreeRTOS.h"
#include "task.h"
#include "Tipos.h"
#include "CommIN.h"
#include "CommOUT.h"
#include "Luz.h"
#include "Seg.h"
//****************************************************************************
// DEFINICION DE LAS PRIORIDADES
#define mainPRUE_TASK_PRIORITY ( tskIDLE_PRIORITY + 1 )
//****************************************************************************
// MAIN
portSHORT main( void ){
    xTaskCreate( vCommINTask, "Commin", configMINIMAL_STACK_SIZE, NULL, mainPRUE_TASK_PRIORITY,
                      NULL );
    xTaskCreate( vCommOUTTask, "Commout", configMINIMAL_STACK_SIZE, NULL, mainPRUE_TASK_PRIORITY,
                      NULL );
    xTaskCreate( vLuzTask, "Luz", configMINIMAL_STACK_SIZE, NULL, mainPRUE_TASK_PRIORITY, NULL );
    xTaskCreate( vSegTask, "Seg", configMINIMAL_STACK_SIZE, NULL, mainPRUE_TASK_PRIORITY, NULL );
    SegIni();
    vPortESIni();
    vTaskStartScheduler();
    return 0;
}
```

Archivo “Tipos.h”

```c
//****************************************************************************
// DEFINICIONES DE TIPOS DE DATOS
#ifndef _H_TIPOS
#define _H_TIPOS
//****************************************************************************
// Tamaños de los tipos de datos
#define lengthSTART  ( 8 )
#define lengthMODiD  ( 8 )
#define lengthCMD    ( 8 )
#define lengthCMD2   ( 8 )
#define lengthSENS   ( 8 )
#define lengthVALUE  ( 16 )
#define lengthEND    ( 8 )
#define largoCOMANDOS (8)
//****************************************************************************
// ESTRUCTURA DE LOS MENSAJES
struct Mensaje {
    unsigned portCHAR   Cmd :       (lengthCMD);
    unsigned portCHAR   Cmd2:       (lengthCMD2);
    unsigned portCHAR   Sens:       (lengthSENS);
    unsigned portSHORT  Value:      (lengthVALUE);
};
//****************************************************************************
```
// ESTRUCTURA DE LOS COMANDOS
struct Comando {
    struct FB{
        unsigned portCHAR Start: (lengthSTART);
        unsigned portCHAR ModId: (lengthMODiD);
    } FirstByte;
    struct Mensaje Msj;
    struct LB{
        unsigned portCHAR End: (lengthEND);
    } LastByte;
};
#endif
Archivo “CommIN.c”
#include "CommIN.h"
#include "PuertoES.h"
#include "ConvAD.h"

#include "CommIN.h"
#include "PuertoES.h"
#include "ConvAD.h"

xQueueHandle xQCommIN;
xQueueHandle xQResTemp;
extern xQueueHandle xQPedLght;
extern xQueueHandle xQPedSeg;
extern xQueueHandle xQCommOUT;

static portCHAR msjRs232[largoCOMANDOS+1];
static struct Mensaje MsjComm; // Mensaje que se enviara al resto del sistema
static unsigned portSHORT res; //Resultados de envio o recepcion de colas
static unsigned portSHORT respuesta; //Resultados de envio o recepcion de colas
static unsigned portCHAR cont;

void vCommINTask(void *pvParameters){

    xQCommIN = xQueueCreate(QCommIN_QueueSize, sizeof(struct Mensaje));
xQResTemp = xQueueCreate(xQResTemp_QueueSize, sizeof(portSHORT));

    // Cargo los datos en msjRs232
    cont=0;
    res = xQueueReceive(xQCommIN, &msjRs232[cont]), (portTickType)5);
    while (!res)
        res = xQueueReceive(xQCommIN, &msjRs232[cont]), (portTickType)5);
    cont=1;
    res=0;
    while (cont<(largoCOMANDOS))
        while (!res)
            res = xQueueReceive(xQCommIN, &msjRs232[cont]), (portTickType)5);
        res=0;
        cont++;
    }
    if ((msjRs232[0]==ARRANQUE)&&(msjRs232[largoCOMANDOS-1]==FIN)&&(msjRs232[1]==_MODID))
        // Si no ocurrieron errores en la recepcion
        {
            // Armo el mensaje para enviar al resto
            MsjComm.Cmd=msjRs232[2];
        }
MsjComm.Cmd2=msjRs232[3];
MsjComm.Sens=msjRs232[4];
MsjComm.Value=msjRs232[5];
MsjComm.Value=(MsjComm.Value<<8)+msjRs232[6];

switch (MsjComm.Cmd2)
{
   // Envio el mensaje a la tarea Temperatura
   case 'T':
   {
      vConvvADSnd(0,xQResTemp); //Pido una lectura de temp
      res = 0;
      while(!res){(res=xQueueReceive(xQResTemp, &(respuesta), (portTickType)5));
      res=0;
      MsjComm.Cmd='r';
      MsjComm.Cmd2='T';
      MsjComm.Sens='1';
      MsjComm.Value=respuesta;
      xQueueSend(xQCommOUT, &(MsjComm), (portTickType)10);
      break;
   }
   // Envio el mensaje a la tarea Luz
   case 'L':
   {
      res = xQueueSend(xQPedLght, &(MsjComm), (portTickType)10);
      break;
   }
   // Envio el mensaje a Electrodomesticos
   case 'E':
   {
      switch(MsjComm.Sens)
      {
      // Electrodomestico 1
      case (1):
      {
         switch(MsjComm.Value )
         {
         case 0x00:
            vPortESElecOff(1);
            break;
         case 0xFF:
            vPortESElecOn(1);
            break;
         }
         break;
      }
      case (2):
      {
         switch(MsjComm.Value )
         {
         case 0x00:
            vPortESElecOff(2);
            break;
         case 0xFF:
            vPortESElecOn(2);
            break;
         }
         break;
      }
      break;
      }
      // Envio el mensje a la tarea Seguridad
      case 'S':
      {
         res = xQueueSend(xQPedSeg, (void*) &(MsjComm), (portTickType)10);
         break;
      }
      default :
      res = (0);
      MsjComm.Cmd='e';
      xQueueSend(xQCommOUT, (void*) &(MsjComm), (portTickType)10);
      break;
   } //end-switch
} //end-switch
else
{
  MsjComm.Cmd='e';
  xQueueSend(xQCommOUT, (void*) &(MsjComm), (portTickType)10);
}
} //fin-while infinito

Archivo “CommIN.h”

// ==============================  CommIN
//                          =======
#include "FreeRTOS.h"
#include "queue.h"
#include "Tipos.h"
#ifndef _CommIN
#define _CommIN
#define QCommIN_QueueSize (2)
#define xQResTemp_QueueSize (2)
//Declaracion de funciones
void vCommINTask(void *pvParameters);
// Declaracion de colas propias
extern xQueueHandle xQCommIN;
extern xQueueHandle xQResTemp;
#endif

Archivo “CommOUT.c”

// ==============================  CommOUT
//                               =======
#include "CommOUT.h"
//----------------------- Declaracion de colas ----------------------
xQueueHandle xQCommOUT;
//---------------- Declaracion de variables de archivo --------------
static unsigned portCHAR  msjRs232[lengthSTR];
static unsigned portCHAR alarma["!Alarma "]="!Alarma ";
static unsigned portCHAR mensajeT["Respuesta de Temp"]="Respuesta de Temp";
static unsigned portCHAR mensajeS["Respuesta de Seg"]="Respuesta de Seg";
static unsigned portCHAR mensajeL["Respuesta de Luz"]="Respuesta de Luz";
static unsigned portCHAR mensajeE["Respuesta de Elec"]="Respuesta de Elec";
static unsigned portCHAR mensajeErr["Mensaje Erroneo"]="Mensaje Erroneo";
static struct Mensaje MsjComm; // Mensaje que se enviara al resto del sistema
static unsigned portSHORT res; //Resultados de envio o recepcion de colas
//--------------------- Declaracion de funciones -------------------
void vCommOUTTask(void *pvParameters){
  //--------------- inicializacion de la tarea -----------------------
  vRS232Ini();
  xQCommOUT = xQueueCreate(QCommOUT_QueueSize, sizeof(struct Mensaje));
  //--------------- Bucle principal -------------------------------
  while (1)
  {
  
  }
res = 0;
while (!res) {
    (res = xQueueReceive(xQCommOUT, &MsjComm, (portTickType)5));
    msjRs232[0] = ARRANQUE;
    msjRs232[1] = MODID;
    msjRs232[5] = (MsjComm.Value >> 8);
    msjRs232[7] = FIN;
    msjRs232[8] = 0x00;
    switch (MsjComm.Cmd) {
        case 'r':
            {
                switch (MsjComm.Cmd2)
                {
                    case 'T':
                        vRS232SendSTR(mensajeT);
                        break;
                    case 'S':
                        vRS232SendSTR(mensajeS);
                        break;
                    case 'L':
                        vRS232SendSTR(mensajeL);
                        break;
                    case 'E':
                        vRS232SendSTR(mensajeE);
                        break;
                    } break;
                }
                case 'e':
                        vRS232SendSTR(mensajeErr);
                        break;
                case 'a':
                        vRS232SendSTR(alarma);
                        break;
                }
                vRS232SendCMD(msjRs232);
            }
        }
    }
}

Archivo “CommOUT.h”

//=======================================
// Proyecto Voyeur
// CommOUT.c
//=======================================

#ifndef _CommOUT
#define _CommOUT

#define QCommOUT_QueueSize (2)

#include “FreeRTOS.h”
#include “queue.h”
#include “Tipos.h”
#include “RS232.h”
#include “PuertoES.h”

#ifndef _CommOUT
#define _CommOUT

#define QCommOUT_QueueSize (2)

#include “FreeRTOS.h”
#include “queue.h”
#include “Tipos.h”
#include “RS232.h”
#include “PuertoES.h”

#endif
#endif

Archivo “Luz.c”

//=======================================
// Proyecto Voyeur
//=======================================

19
// Luz.c

#include "Luz.h"

// Declaracion de colas
xQueueHandle xQPedLght;
xQueueHandle xQResLght;

// Declaracion de variables de archivo
portBASE_TYPE haypedido;
unsigned portSHORT LghtUp;  //Nivel superior de luz
unsigned portSHORT LghtDn;  //Nivel inferior de luz
unsigned portSHORT LghtLvl; //Nivel actual de luz
static unsigned portSHORT res; //Resultados de envio o recepcion de colas
static struct Mensaje msgLuz;
static portSHORT respuesta;
static char Luces;
char status; // Si se controla o no

// Declaracion de funciones
void vLuzTask(void *pvParameters){

    // inicializacion de la tarea
    xQPedLght = xQueueCreate(QLuz_QueueSize, sizeof(struct Mensaje));
xQResLght = xQueueCreate(QLuz_QueueSize, sizeof(portSHORT));

    LghtUp=0x090;  //Defino los limites iniciales para el control
    LghtDn=0x070;  //de la luz
    LghtLvl=0x0BF; // inicializo el valor de luz;
    Luces=0;      //bandera de luces
    status=1; // el control arranca apagado
    vPortESIni();
    vConvADini(); //inicio el conversor

    while (1)
    {
        vConvADSnd(1,xQResLght); //Pido una lectura de luz
        res =0;
        while(!res){(res=xQueueReceive(xQResLght, &(respuesta), (portTickType)5));}
        // Seccion de control de luz
        if (status)
        {
            LghtLvl=respuesta; //Cargo el valor acutal de luz en registro
            if ( LghtLvl<LghtDn)
            {
                //Estoy por debajo del nivel bajo
                switch(Luces)
                {
                    case 0x00:
                        vPortESLghtOn(1);
                        Luces=0x01;
                        break;
                    case 0x01:
                        vPortESLghtOn(2);
                        Luces=0x11;
                        break;
                }
            }
            else if (LghtLvl>LghtUp)
            {
                //Estoy por encima del nivel alto
                switch(Luces)
                {
                    case 0x11:
                        vPortESLghtOff(2);
                        Luces=0x01;
                        break;
                    case 0x01:
                        vPortESLghtOff(1);
                }
            }
        }
    }
}
Luces=0x00;
break;
case 0x00:
break;
}
}
// Seccion Comando del sistema de control
haypedido=xQueueReceive( xQPedLght, &msgLuz, ( portTickType )10 );
if(haypedido)
{
if(!status)
{
switch (msgLuz.Cmd)
{
    case 'R':
    {
        vConvADSnd(1,xQResLght);
xQueueReceive( xQResLght, &respuesta, ( portTickType ) 10 );
        msgLuz.Cmd='r';
        msgLuz.Value=respuesta;
xQueueSend( xQCommOUT, &msgLuz, ( portTickType ) 10 );
brea
    }
}
    case 'C':
    {
        switch (msgLuz.Sens)
        {
            case 0x0F:
            {
                status=1;  //Prendo el sistema de luces
                TIMSK=2;
            break;
        }
            default:break;
        }
    }
}
else
{
switch (msgLuz.Cmd)
{
    case 'R':
    {
        msgLuz.Cmd='r';
        msgLuz.Cmd2='L';
        msgLuz.Sens=1;
        msgLuz.Value=LghtLvl;
xQueueSend( xQCommOUT, &msgLuz, ( portTickType ) 10 );
break;
    }
    case 'C':
    {
        switch (msgLuz.Sens)
        {
            case 0x00:
            {
                status=0;  //Apago el sistema de luces
                TIMSK=0;
            break;
        }
            case 0x0F:
            {
                LghtDn=msgLuz.Value;
            break;
        }
            case 0xFF:
            {
                LghtUp=msgLuz.Value;
            break;
        }
    }
    break;
}
Archivo “Luz.h”

```c
#include "FreeRTOS.h"
#include "queue.h"
#include "Tipos.h"
#include "CommOUT.h"
#include "Timer.h"
#include "ConvAD.h"

#ifndef _Luz
#define _Luz
#define QLuz_QueueSize (2)
#define Luz

void vLuzTask(void *pvParameters);

#endif
```

Archivo “Seg.c”

```c
#include "Seg.h"

static portCHAR sensores;
static portCHAR estado;
static struct Mensaje msgSeg;
static unsigned portSHORT res; //Resultados de envio o recepcion de colas

#include "Seg.h"

void SegIni();
char cIsSensOn(char sen);
void vSegTask(void *pvParameters){
    xQueueHandle xQPedSeg;
    static portCHAR sensores;
    static portCHAR estado;
    static struct Mensaje msgSeg;
    static unsigned portSHORT res; //Resultados de envio o recepcion de colas

    void SegIni();
    char cIsSensOn(char sen);
    void vSegTask(void *pvParameters){
        // Espero por un pedido, desde el pc o desde
        while (1)
        {
            // Espero por un pedido, desde el pc o desde
            res = 0;
            while(!res){(res=xQueueReceive(xQPedSeg, &(msgSeg), (portTickType)5));}

            if (msgSeg.Cmd2=='S')
            {
                switch (msgSeg.Cmd)
                {
                }
            }
        }
    }
    }
```
case 'C':
{
    switch (msgSeg.Sens)
    {
    case 0x00: //Apago el sistema de Seguridad
    {
        estado=0;
        break;
    }
    case 0x0F: //Prendo el sistema de Seguridad
    {
        estado=1;
        break;
    }
    default:
    {
        switch (msgSeg.Value)
        {
        case 0x00: // Apago el sensor n
        {
            sensores=(0<<msgSeg.Sens);
            break;
        }
        case 0xFF: // Prendo el sensor n
        {
            sensores=(1<<msgSeg.Sens);
            break;
        }
        break;
        }
    }
    break;
    }
    case 'R': // Devuelvo el estado del sensor n
    {
        msgSeg.Cmd='r';
        msgSeg.Value=sensores&(1<<msgSeg.Sens);
        xQueueSend(xQCommOUT, (void*) &(msgSeg), (portTickType)10);
        break;
    }
    case 'a': // Se produjo una alarma en el sensor n
    {
        VPortESSeg(2);
        if ((estado)&(cIsSensOn(msgSeg.Sens)))
        {
            msgSeg.Cmd='a';
            xQueueSend(xQCommOUT, (void*) &(msgSeg), (portTickType)10);
            break;
        }
    }
    msgSeg.Cmd2=0;
    }
}

void SegIni()
{
    GICR_Bit7=1;   // Habilito int1
    GICR_Bit6=1;   // Habilito int1
    MCUCR_Bit3=0; // Configuro flanco de bajada en INT1
    MCUCR_Bit2=0;
    MCUCR_Bit1=0; // Configuro flanco de bajada en INT0
    MCUCR_Bit0=0;
    estado=1;
    sensores=0xFF;
}

char cIsSensOn(char sen)
{
    char temp;
    temp=(1<<sen);
    temp= sensores && temp;
}
return temp;
}

__interrupt void Seg_INT0 (void)
{
    GIFR_Bit6=0;
    VPortESSeg(0);
    msgSeg.Cmd = 'a';
    msgSeg.Cmd2 = 'S';
    msgSeg.Value = 0x00;
    msgSeg.Sens = 0x1;
    xQueueSendFromISR(xQPedSeg, (void*) &msgSeg,1);
}

__interrupt void Seg_INT1 (void)
{
    GIFR_Bit7=0;
    VPortESSeg(1);
    msgSeg.Cmd = 'a';
    msgSeg.Cmd2 = 'S';
    msgSeg.Value = 0x00;
    msgSeg.Sens = 0x2;
    xQueueSendFromISR(xQPedSeg, (void*) &msgSeg,1);
}

Archivo “Seg.h”

#include "FreeRTOS.h"
#include "queue.h"
#include "Tipos.h"
#include "CommOUT.h"
#ifndef _Seg
#define _Seg
#define QSeg_QueueSize (2)
//Declaracion de funciones
void vSegTask(void *pvParameters);
void SegIni();
// Declaracion de colas propias
extern xQueueHandle xQPedSeg;
#endif

Archivo “ConvAD.c”

#include "ConvAD.h"
#include "semphr.h"
xQueueHandle xQtemporal;
static xSemaphoreHandle xSemaphore;

void vConvADSnd(char CH,xQueueHandle xQResSens)
{
    xSemaphoreTake( xSemaphore, ( portTickType ) 10 );
    xQtemporal=xQResSens;
    CH=CH&7;
    ADMUX=ADMUX|CH;
    if (!ADCSR_Bit4){
        ADCSR_Bit6=1; //Prende el conversor
        if (IADCSR_Bit4){
            ADCSR_Bit5=1; // Inicia la conversion
        }else{
            ADCSR_Bit4=1; // Inicia la conversion
        }
    }
}

CH=CH&7;
ADMUX=ADMUX|CH;
ADCSR_Bit7=1; //Prende el conversor
if (IADCSR_Bit4){
    ADCSR_Bit5=1; // Inicia la conversion
}
void vConvADini(){
    vSemaphoreCreateBinary( xSemaphore );
    ADMUX_Bit7=1;  // Volt de Referencia interna
    ADMUX_Bit6=1;  // Volt de Referencia interna
    ADMUX_Bit5=0;  // Resultado Ajustado a la derecha
    ADMUX_Bit4=0;
    ADMUX_Bit3=0;
    ADCSR_Bit5=0; //Single Conversion mode
    ADCSR_Bit3=1; //Interrupt enable
    ADCSR=ADCSR&248;
}

void vConvADoff(){
    ADCSR_Bit7=1; //Prende el conversor
}

__interrupt void ADC_ISR (void){
    unsigned portSHORT resultado;
    resultado=ADCL;
    resultado=resultado + ADCH*256;
    xQueueSend(xQtemporal, &resultado, (portTickType )10);
    xSemaphoreGiveFromISR( xSemaphore,1);
}

Archivo “ConversorAD.h”

// ARCHIVOS A USAR
#include "FreeRTOS.h"
#include "queue.h"
#include "PuertoES.h"
#include "Task.h"

//FUNCIONES IMPLEMENTADAS
void vConvADSnd(char CH,xQueueHandle xQResSens);
void vConvADini();
void vConvADoff();

//DEFINICIONES
// Cola a la que el conversor AD debe devolver el dato convertido, esta cola
// debe definirse fuera de ConvAD
extern xQueueHandle xQResSens;

Archivo “PuertoES.c”

//IMPLEMENTACION DEL MODULO PUERTOS DE E/S
//ARCHIVOS A INCLUIR
#include "PuertoES.h"

// Puertos de E/S de Seguridad
char vPortESSecRd(char act){
    switch (act){
        case 1:
            return PORTB_Bit0;
        case 2:
            return PORTB_Bit1;
    }
    return 0xFF;
}
Puertos de E/S de Luces

// ON
void vPortESLghtOn(char act){
    switch (act){
    case 1:
        PORTB_Bit2=1;
        break;
    case 2:
        PORTB_Bit3=1;
        break;
    }
}

// OFF
void vPortESLghtOff(char act){
    switch (act){
    case 1:
        PORTB_Bit2=0;
        break;
    case 2:
        PORTB_Bit3=0;
        break;
    }
}

Puertos de E/S de Electrodomésticos

// ON
void vPortESElecOn(char act){
    switch (act){
    case 1:
        PORTB_Bit4=1;
        break;
    case 2:
        PORTB_Bit5=1;
        break;
    }
}

// OFF
void vPortESElecOff(char act){
    switch (act){
    case 1:
        PORTB_Bit4=0;
        break;
    case 2:
        PORTB_Bit5=0;
        break;
    }
}

void VPortESblink(char cc){
    switch (cc){
    case 0:
        PORTB_Bit4=0;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=1;
        break;
    case 1:
        PORTB_Bit4=0;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 2:
        PORTB_Bit4=0;
        PORTB_Bit5=1;
        PORTB_Bit6=1;
        PORTB_Bit7=1;
        break;
    case 3:
        PORTB_Bit4=0;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 4:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=0;
        break;
    case 5:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 6:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=0;
        break;
    case 7:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 8:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=0;
        break;
    case 9:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 10:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=0;
        break;
    case 11:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 12:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=0;
        break;
    case 13:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    case 14:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=0;
        PORTB_Bit7=0;
        break;
    case 15:
        PORTB_Bit4=1;
        PORTB_Bit5=0;
        PORTB_Bit6=1;
        PORTB_Bit7=0;
        break;
    }
}
PORTB_Bit5=1;
PORTB_Bit6=0;
PORTB_Bit7=0;
break;
    case 3:
PORTB_Bit4=1;
PORTB_Bit5=0;
PORTB_Bit6=0;
PORTB_Bit7=0;
break;
    case 4:
PORTB_Bit4=0;
PORTB_Bit5=0;
PORTB_Bit6=0;
PORTB_Bit7=0;
break;
}
}
void VPortESSeg(char cc){
    switch (cc)
    {
        case 0:
PORTB_Bit0=1;
PORTB_Bit1=0;
break;
        case 1:
PORTB_Bit0=0;
PORTB_Bit1=1;
break;
        case 2:
PORTB_Bit0=0;
PORTB_Bit1=0;
break;
    }
//******************************************************************************
// Inicializo los pines 0 y 1 del puerto c como entradas
void vPortESIni(){
    // Configuro los puertos de entrada de la seguridad
    DDRD_Bit2=0;
    DDRD_Bit3=0;
    PORTD_Bit2=1;
    PORTD_Bit3=1;
    DDRB_Bit0=1;
    DDRB_Bit1=1;
    //Configuro los puertos de salida de LUZ
    DDRB_Bit2=1;
    DDRB_Bit3=1;
    //Configuro los puertos de salida de Electrodomesticos
    DDRB_Bit4=1;
    DDRB_Bit5=1;
    DDRB_Bit6=1;
    DDRB_Bit7=1;
}
Archivo “PuertoES.h”
******************************************************************************
// ARCHIVOS A USAR
#include "FreeRTOS.h"
******************************************************************************
// FUNCIONES IMPLEMENTADAS
void vPortESIni();
char vPortESSecRd (char act); // Leer barreras de seguridad
void vPortESLghtOff(char act); // Apagar luces
void vPortESLghtOn(char act); // Prender luces
void vPortESElecOff(char act); // Apagar Electrodomesticos
void vPortESElecOn(char act); // Encender Electrodomesticos
void VPortESblink(char act); //Prende y apaga el led 6
archivo "RS232.c"

// IMPLEMENTACION DEL MÓDULO RS232
// ARCHIVOS A INCLUIR
#include "RS232.h"

// Inicialización de la interface Rs232
void vRS232Ini()
{
    UBRRH = 0x00;
    UBRRL = 0x33;
    UCSRA = 0x02;
    UCSRC = 0x86;
    UCSRB = 0x98;
}

// Envia datos a través de la interface Rs232
void vRS232Send(portCHAR msg)
{
    static portCHAR mensaje;
    mensaje = msg;
    while (!(UCSRA_Bit5)){} // espero a que este libre el buffer de salida
    UDR = mensaje; // mando el mensaje
}

// Envia comando a través de la interface Rs232
void vRS232SendCMD(unsigned portCHAR *msg)
{
    char cont;
    cont = 0;
    while (cont < (largoCOMANDOS))
    {
        vRS232Send(*msg);
        msg++;
        cont++;
    }
}

Referencias
[1] 2503J-AVR-10/06, ATMEGA 32 datasheet, ATMEL.