ESCUELA DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería Informática

PERIFÉRICOS E INTERFACES

PRÁCTICAS DE LABORATORIO

Módulo 4 – Práctica 4

Máquina de Juego

José María Amusquívar Poppe
2018-2019 | Grupo 44 | 11/05/19

Índice

Contenido

Primera parte: Realización práctica básica

Objetivos

En esta cuarta práctica se desarrollará la implementación de un juego usando todas las herramientas vistas en prácticas anteriores, asimismo, esta práctica irá anidada a la calculadora desarrollada en la práctica 3, consiguiendo finalmente un programa multifuncional capaz de alternar entre dos modos, el juego o la calculadora.

La realización de la práctica por parte de los estudiantes se orienta a complementar la consecución de la competencia de título CII01 del Plan de Estudios de la Ingeniería Informática y que los capacita para: diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos asegurando su fiabilidad, seguridad y calidad conforme a principios éticos y a la legislación y normativa vigente. En base a ello, con la realización de esta práctica se pretende que los estudiantes alcancen satisfactoriamente las siguientes capacidades:

1. Capacidad para entender e interrelacionar los diferentes componentes hardware y software que integran un interfaz sencillo de un sistema basado en microcontrolador.
2. Capacidad para diseñar, implementar y verificar el correcto funcionamiento de dispositivos externos sencillos para ser conectados a través de interfaces seriales y paralelos.
3. Capacidad para el desarrollo de programas que permitan el control básico del dispositivo externo haciendo uso de un lenguaje de programación.
4. Capacidad para desarrollar programas que doten al sistema de una determinada funcionalidad.
5. Capacidad para aprender y aplicar nuevos conceptos de forma autónoma e interdisciplinar.
6. Capacidad para emplear la creatividad en la resolución de los problemas.

El logro de las citadas y pretendidas capacidades pasa por el planteamiento de un conjunto de intenciones y metas que orientan el proceso de aprendizaje de los estudiantes y que constituyen los objetivos de la práctica. De esta forma, los objetivos propuestos para esta práctica se concretan en:

1. Profundizar en el conocimiento de la estructura interna de un periférico sencillo dotándolo del software de control e interconexión con el usuario haciendo uso de la plataforma Arduino.
2. Conocer las particularidades de las diferentes señales físicas que aparecen en los conectores de la placa a efectos de poder conectar dispositivos correctamente.
3. Conocer el funcionamiento de los componentes electrónicos básicos utilizados y saber integrarlos en el sistema para lograr la funcionalidad pretendida del periférico.
4. Saber diseñar el software de control del periférico básico propuesto para su correcto funcionamiento de acuerdo a la funcionalidad exigida incluido la comunicación con el usuario y con el sistema computador si fuese necesario.

Recursos utilizados

Para la realización de la práctica básica se ha usado una gran variedad de recursos, tanto hardware como software.

Entre los recursos software cabe destacar el uso de la función random(N), donde N es el número máximo posible generado menos uno, es decir, generará números entre 0 y N-1, dicha función va acompañada de su semilla randomSeed(analogRead(0)), esta última asegura que el número generado por random() sea realmente aleatorio entre varias ejecuciones del programa.
También cabe destacar el uso de funciones como atof() o dtostrf(), la primera convierte un array de char en un float, la segunda realiza el proceso inverso. Se hizo uso de estas funciones puesto que la calculadora es capaz de operar con números racionales (estas funciones tienen su equivalente para números enteros atoi() e itoa()). Además, se empleó una interrupción por el Timer que está programado para que interrumpa cada 10 ms. Además, al imprimir por la pantalla LCD de la placa de expansión fue necesario usar unas instrucciones con unos parámetros propios. Y, finalmente, otra gran cantidad de funciones y elementos básicos como bucles, condiciones, printf(), enteros, booleanos, etc.

Entre los recursos hardware cabe destacar el uso la pantalla LCD 20x4, que tiene un protocolo de comunicación Serial TTL con el Arduino, por lo que emplea los pines TX/RX. Y, por consiguiente, usa instrucciones de este tipo como Serial3.write(). También se empleó los 7-segmentos junto con el teclado, los enunció a la vez dado que comparten el mismo puerto de Arduino (PORTL), los 4 pines pertenecientes a los 7-segmentos están configurados como salida, en cambio, los 4 pines del teclado están como entrada. Y también se usó los botones y el speaker, compartiendo el mismo puerto de Arduino (PORTC), en este caso los 5 botones configurados como entrada y el pin del speaker como salida. Finalmente destacar que en el apartado de la calculadora se habilitó un botón adicional conectado a una interrupción, este pulsador se emplea para escribir una coma ‘,’ en el cálculo de número racionales.

Realización de la práctica

Para la realización de esta práctica, en el apartado de la calculadora, se añadió un pulsador adicional conectado a una interrupción, dicho botón se emplea para señalar la coma en los números racionales. Recordar que este añadido ya estaba presente en la entrega de la práctica 3. Ahora se procederá a analizar el código de la práctica:

- **setup():** Se configura la interrupción del Timer para que interrumpa cada 10 ms. También se declara la interrupción del botón adicional (coma), y por último se declaran los puertos a usar como salida o entrada (DDRA, DDRL, DDRC), y se habilita el pullup de los botones con PORTC = 0xF8.

```
129 /*******************Declaración de puertos************************/
130 DDRA = 0xFF; //Puerto leds 7-segmentos.
131 DDRL = 0x0F; //Puerto de activacion digito 7-segmentos. 0B(1111)_XXXX lectura botones.
132 DDRC = 0x01; //Puerto entrada de botones, bit menos significativo a 1, salida speaker.
133 PORTC = 0xF8; //Activacion pull-up botones, menos speaker.
134 /*****************************************************************/
```

- **loop():** Aquí se lee el puerto del teclado con PINL>>4, para obtener cuatro bits, cada uno correspondiente a una fila del teclado, acorde a ello se accede a modo juego si se ha pulsado '*' y '7' (0x0C) y dentro muestra una serie de mensajes por pantalla, cambia el valor de dos booleanos indicadores del modo y reinicia las variables pertenecientes a juego; si se ha pulsado '*' y '1' (0x06) sale de modo juego y entra a modo calculadora imprimiendo una serie de mensajes, cambiando el valor de dos booleanos y reiniciando sus variables.

```
266
267   byte teclaPulsado = 0xFF; //Se lee el puerto del teclado, se desplaza 4
268   teclaPulsado = PINL >> 4;
269
```

- **cuentaDigitos(char* x):** Es un método auxiliar al que se le pasa un char*, del cual se calcula su longitud para devolverlo como un int.

```
378 /***Cuenta la longitud del char que contiene los números***/
379 int cuentaDigitos(char* pos){
380   int digitos = 0;
381   while(*pos != '\0'){
382     digitos++;
383     pos++;
384   }
385   return (digitos-1);
386 }
387 /*****************************************************************/
```

- **error(char* x):** Método auxiliar usado sólo en la calculadora, imprime un mensaje en la esquina inferior derecha, por lo que se le resta esta posición (0x67) menos la longitud del char* a imprimir. Se ayuda del método cuentaDigitos().

```
389 /*Imprime los caracteres pasado como char* en la última posición de la pantalla*/
390 void error(char* mensaje){
391   Serial3.write(0xFE);
392   Serial3.write(0x48);
393
394   Serial3.write(0xFE);
395   Serial3.write(0x45);
396   Serial3.write(0x67 - cuentaDigitos(mensaje));
397   Serial3.print(mensaje);
398 }
399 /*****************************************************************************/
```

- **ISR(TIMER1_COMPA_VECT):** Es la rutina de la interrupción del Timer a la cual accede cada 10 ms. Dentro se cambia de modo comprobando los booleanos alterados en el loop(), acorde a ello saltará a ISR_Calculadora() o, si está en pleno juego, a cuentaJuego() o, si está en espera a elegir un juego, ISR_Juego().

```
401 /*****Rutina del Timer*****/
402 ISR(TIMER1_COMPA_vect){
403   if(modoJuego){ //Modo juego.
404     if(finJuego){
405       ISR_Juego(); //Contador de modo juego, en espera.
406     }else{
407       cuentaJuego(); //Contador de modo juego, mientras está jugando, 60..0.
408     }
409   }else{
410     ISR_Calculadora(); //Contador de modo calculadora.
411   }
412 }
413 /*****************************/
```

Rutinas del juego:

- **ISR_Juego():** Esta rutina es similar a ISR_Calculadora(). Se tiene un vector con el código en hexadecimal de los 4 dígitos de los 7-segmentos (D1, D2, D3, D4) y un contador que va de 0 a 3, acorde a este contador se habilita un determinado dígito. Dentro también salta a la rutina reboteBotonesJuego(), incrementa contadores y si el contador ha llegado a 4, se reinicia a 0. Y si el contador de tiempo es un múltiplo de 200 actualiza los segmentos mostrados en los 7-segmentos. Se eligió 200 porque el Timer está configurado a 200Hz.

```
446  if(cont == 4) cont = 0; //Si se ha llegado al último dígito del 7-segmentos, se reinicia a 0.
447  if(tiempo%200 == 0) contar(); //Se actualiza los valores del 7-segmentos cada 10 ms.
448  /**Ya que el timer está configurado como OCR1A = 77, va a 200 Hz, por lo que para actualizar cada
449   * 10 ms, se debe dividir el tiempo/200, y obtener su módulo, qué será 0 si han pasado 10 ms.
450   */
```

- **reboteBotonesJuego():** Se lee el puerto de los botones con PINC>>3, y se analiza los dos casos posibles. Si el pulsador de arriba ha sido pulsado se inicia el juego, genera el número aleatorio, reinicia variables, altera un booleano y salta a cuentaJuego(). Si se ha pulsado el botón de arriba y abajo a la vez, se reinicia el juego, por tanto, se reinician las variables.

```
455 void reboteBotonesJuego(){
456   byte operacionJuego = 0x00;
457   byte botonPulsado = PINC >> 3; //Se lee el puerto de los botones,
458
```

- **cuentaJuego():** Mismo mecanismo que ISR_Juego(), con excepción que a los dígitos de las centenas y millar se les pasa un 0x00, ya que no en necesario encenderlos al tener un contador de 60 a 00, que tienen sus propios punteros.

```
572   PORTL = segmentos[contJuego]; //Se habilita el dígito correspondiente.
573   //Se imprimen los valores en cada dígito activado.
574   switch(contJuego){
575     case 0:
576           PORTA = num[unidJuego];
577           break;
578     case 1:
579           PORTA = num[decJuego];
580           break;
581     case 2:
582           PORTA = 0x00; //No muestra nada, pero sí requerido para leer el teclado.
583           break;
584     case 3:
585           PORTA = 0x00; //No muestra nada, pero sí requerido para leer el teclado.
586           break;
587   }
```

- **Contarjuego():** Contador para el juego, va desde 60 a 00, si se llega a 00 se imprime un mensaje de fallo y se altera un booleano.

```
626   //Se emplea un contador con 60 segundos y un minutero.
627   if(unidJuego == 0){
628     if(decJuego == 0){
629       //Si dec es igual a cero, el tiempo ha finalizado, se imprime un mensaje.
```

- **compruebaNumero(int a, int b):** Método que comprueba si el número introducido 'a' es igual, mayor o menor que el generado por la máquina, además también se encarga de imprimirlo por pantalla. Si es igual genera unos pitidos y modifica un booleano, si no lo es, comprueba si es mayor o igual, pero antes se comprueba si hay espacio en la fila actual de la pantalla LCD sumando en una variable el tamaño del número 'b', si esta variable supera 20, se debe limpiar la fila siguiente con limpiaFila(byte fila) y situar el cursor a su inicio. Por esta razón, se imprime el número introducido después de enviar el carácter verificación '#'. Una vez hecho eso se comprueba si es mayor o menor y se imprime por pantalla.

```
675    //LCD de 20 posiciones y 4 filas. 1 posición añadida por el símbolo "< o >".
676    posicion += tamano+1;
677    if(posicion > 20){ //Si sobrepasa los 20, se limpia la fila siguiente.
678      if(fila == 0){
679        limpiaFila(0x40);
680        fila = 1;
681      }else if(fila == 1){
682        limpiaFila(0x14);
683        fila = 2;
684      }else if(fila == 2){
685        limpiaFila(0x54);
686        fila = 3;
687      }else if(fila == 3){
688        limpiaFila(0x00);
689        fila = 0;
690      }
691      posicion = tamano+2; //Si se ha limpiado la fila, se reinicia posicion a 2 más el tamaño del dígito.
```

- **limpiaFila(byte fila):** Método auxiliar de compruebaNumero() que limpia la fila siguiente a una cuando ésta está llena. Limpia la fila imprimiendo un espacio en las 20 posiciones de esa fila. Posteriormente, sitúa el cursor a su inicio.

```
711 //Método que limpia la fila pasada por parámetro, y sitúa el cursor a su inicio.
712 void limpiaFila(byte fila){
713   //Se limpia cada fila, imprimiendo un espacio en cada posición.
714   for(int i = 0; i < 20; i++){
715     Serial3.write(0xFE);
716     Serial3.write(0x45);
717     Serial3.write(fila + i);
718     Serial3.write(' ');
719   }
720   Serial3.write(0xFE);
721   Serial3.write(0x45);
722   Serial3.write(fila);
723 }
```

- **reboteTecladoJuego():** Método que recoge las pulsaciones del teclado. Se lee el puerto del teclado con PINL>>4 quedándose así sólo con los bits de las 4 filas, verifico qué fila ha sido pulsada, y dentro de cada uno de los 4 casos verificó qué columna fue pulsada comprobando el valor del contador que recorre los cuatro dígitos de los 7-segmentos. En cada uno de los casos se le asigna su respectivo valor. Y finalmente, mientras no sea el carácter verificación '#', se acumula el número en un char[] avanzando su respectivo puntero. Una vez se pulsa el carácter verificación '#', entonces se transforma el char[] en un entero con atoi() y se manda a comprobar el número con compruebaNumero(num, tamNum).

```
796   byte entero = 0x00; //Se almacena el valor pulsado.
797   byte teclaPulsado = 0x00; //Se lee el puerto del teclado,
798   teclaPulsado = PINL >> 4;
```

Rutinas de la calculadora:

- **ISR_Calculadora():** Rutina muy similar a ISR_Juego(), con la única excepción que en este caso también se comprueba el teclado con reboteTeclado().

```
890  reboteTeclado(); //Rutina de lectura de los botones.
891  reboteBotones(); //Rutina de lectura del teclado.
892  cont++;
893  tiempo++;
```

- **operar():** Método que realiza el cálculo de los número introducidos, se escoge la operación según el carácter guardado en una variable global y se realiza dicha operación (se contempla también el divisor 0). Si el divisor no fue cero, entonces se transforma el float resultado en un char[] y se imprime en pantalla con error().

```
927    if(!divisorCero){ //Se convierte el resultado double a char[], con 2 decimales.
928      dtostrf(res, sizeof(res), 2, aux);
929      calculadoraFin = true;
930      error((char*) aux);
931    }
```

- **coma():** Rutina de la interrupción del botón adicional utilizado para operar con números decimales. Si se ha introducido el signo, entonces el punto se añadirá a la segunda expresión, si no, se añade a la primera. Se añade simplemente colocando su valor ASCII (0x2E) en el char[] de los números.

```
940    //Se actualiza los arrays de char y sus apuntadores.
941    if(opera == 0x00){
942      primeroC[avanzaP] = 0x2E;
943      avanzaP++;
944    }else{
945      segundoC[avanzaS] = 0x2E;
946      avanzaS++;
947    }
```

- **contar():** Método que se encarga de actualizar el tiempo, que va de 0000 a 9959, esto es debido a que cuenta como un reloj, los dos dígitos de la izquierda son los minutos, y los dos de la derecha son los segundos, que van desde 00 a 59.

```
954  //Se emplea un contador con 60 segundos y un minutero.
955  if(unid == 9){
956    unid = 0;
957    if(dec == 5){
958      dec = 0;
959      if(cent == 9){
960        cent = 0;
961        if(mil == 9){
962          mil = 0;
```

- **reboteBotones():** Método que lee el puerto de los botones con PINL>>4, compara cual se ha pulsado, y según ello le asigna algún operando "+-*/". Se analiza cada caso, como si se ha introducido un '-' o '+' al inicio lo cuente como el símbolo del número siguiente. O si no se han introducido datos, o se han introducido pocos datos, o un formato erróneo. Suponiendo que respeta el formato, si se ha pulsado algún botón de operación, convierte el primer número a float. Y si se ha pulsado el botón central '=', convierte el segundo número a float y procede a operarlos en operar().

```
980  byte operacion = 0x00; //Se recoge en una variable el operando pulsado.
981  byte botonPulsado = PINC >> 3; //Se lee el puerto de los botones, y se
```

- **reboteTeclado():** Se recoge y se asigna valores del mismo modo que en reboteTecladoJuego. Si el carácter recogido no es '*' o '#', pues almacena dicho carácter en sus respectivo char[]. Si se ha pulsado '#', borra un elemento, es decir, hace una backSpace(). Para ello comprueba si se ha pulsado el carácter operación o no, de este modo se sabe si borrar el primer número, el segundo o simplemente el carácter operación. Si se ha pulsado '*', limpia toda la pantalla llamando al método reinicioCalculadora().

```
1052  byte entero = 0x00; //Se almacena el valor pulsado.
1053  byte teclaPulsado = 0x00; //Se lee el puerto del teclado,
1054  teclaPulsado = PINL >> 4;
```

- **reinicioCalculadora():** Reinicia todos y cada una de las variables de la calculadora, también limpia la pantalla LCD y sitúa el cursor al inicio.

```
1158      calculadoraFin = false;
1159      finJuego = true;
1160      pausa = false;
```

- **reinicioJuego():** Reinicia todos y cada una de las variables del juego, también limpia la pantalla LCD y sitúa el cursor a inicio.

```
1180  numeroIn[0] = '\0';
1181  unidJuego = 0;
1182  decJuego = 6;
1183  posicion = 0;
```

Para cualquier duda, aparte de este documento, siempre estará el código ".ino" con algunos comentarios para explicar su funcionamiento.

Conclusiones

Con la realización de la parte básica de esta práctica se han alcanzado todos los objetivos descritos en el primer apartado de este informe, ya que he aprendido a interrelacionar los distintos periféricos con las distintas herramientas software, también he aprendido cómo está montado este circuito, todas las conexiones realizadas entre la placa de expansión y el propio Arduino, también a imaginar distintas soluciones para un mismo problema, entre otras cosas más. La realización de esta práctica no fue dificultosa ya que la mitad de la práctica ya estaba resuelta en la práctica 3, sólo hubo que montar un sistema para poder permutar el modo de funcionamiento y desarrollar la implementación del juego. Podría afirmar que, a partir de ahora, seré capaz de ver con mayor facilidad la resolución de algún problema que lleve partes interrelacionadas, así como montar mis propios circuitos electrónicos y aplicarle también un desarrollo software.

Conclusiones finales

La realización de esta práctica ha implicado demostrar todos nuestros conocimientos adquiridos en las prácticas anteriores, así como en las clases de teoría. Pues en esta práctica y en sus respectivas mejoras, qué se explicarán a continuación, se ha necesitado todos los conocimientos aprendidos de los 7-segmentos, el Monitor Serie y el uso del speaker (práctica 1), los conocimientos sobre la escritura y lectura de los módulos de memoria aplicando el protocolo I2C (práctica 2), los conocimientos sobre la sincronización entre interrupción del Timer y las distintas funciones, también sobre la conexión Serial TTL con la pantalla LCD, la lectura del teclado y los botones (práctica 3). Algo que dificultó realmente esta práctica no fue la sincronización entre las constantes interrupciones del Timer y las distintas funciones, sino que fue la escritura adecuada de datos en la pantalla LCD, pues esta pantalla tiene una distribución algo compleja, saltando de la primera fila a la tercera, luego a la segunda, luego a cuarta y otra vez a la primera. No sigue un orden, por tanto, para imprimir correctamente los valores en pantalla sin necesidad de partir los datos entre filas, se tuvo que ingeniar una solución que evitara esto, esta solución queda reflejada en el método compruebaNumero() y limpiaFila().

Segunda parte: Realización práctica mejorada

Objetivos

Lo objetivos de esta parte son básicamente los mismos que los especificados en la primera parte con algún añadido extra, por ejemplo:

- Aprender el correcto uso de los distintos módulos de memoria, y, en este caso, de su correspondiente protocolo de comunicación, el bus I2C. Esto implica saber cómo funcionan las señales de reloj (SCL) y datos (SDA), con el fin de ser capaz de escribir en memoria y también leer.
- Aprender a sincronizar el Monitor Serie con los distintos periféricos, así como saber enviar información recibida por este mecanismo a la pantalla LCD de la placa de expansión.
- Estudiar el protocolo de conexión que usan los mandos infrarrojos, saber cómo integrar y montar el circuito del receptor infrarrojo al Arduino, reconocer el funcionamiento y tratamiento de los comandos que se reciben por el receptor infrarrojo que, en esta práctica, está conectado a un nivel de interrupción.
- Aprender cómo incluir librerías externas al proyecto para proporcionar herramientas adicionales.

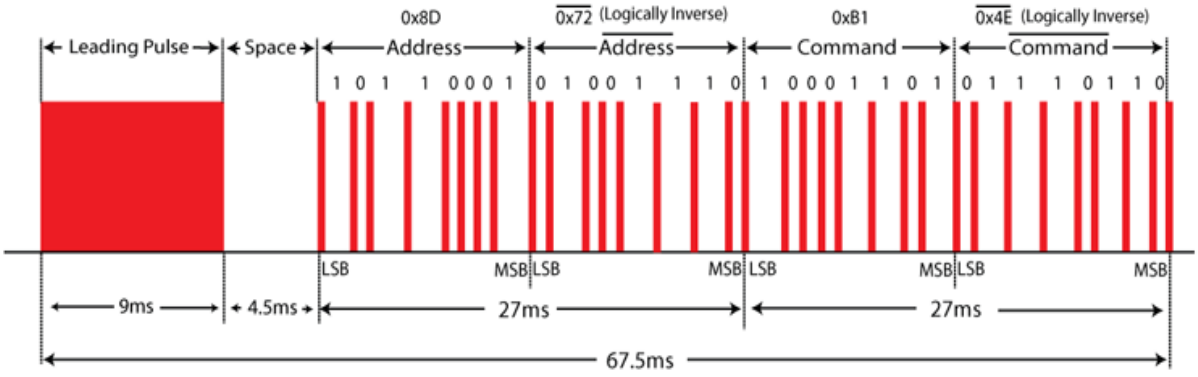
Recursos utilizados

El recurso utilizado más importante en las mejoras es el mando infrarrojo, pues para conectar este mando es necesario incluir una librería externa al fichero (IRLremote.h), también se requiere un receptor infrarrojo que decodifica la señal recibida, señal que es enviada, en este caso, siguiendo el protocolo NEC, que emplea una onda portadora de 38 KHz y una modulación de distancia por pulsos(PDM)., y también el propio mando infrarrojo.

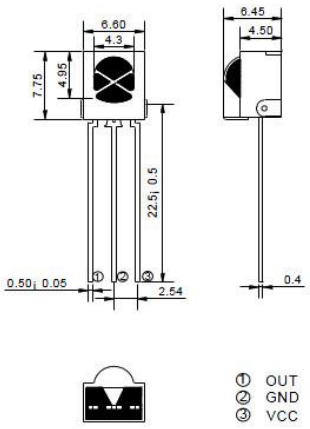
```
1 #include "IRLremote.h"
2
```

El mando infrarrojo tiene una dirección asignada que lo identifica, por tanto, al apretar un botón del mando infrarrojo, lo que se está haciendo es enviar su dirección que ocupa 8 bits junto al comando asignado a ese botón que ocupa otros 8 bits, esto significa que puede controlar hasta 256 dispositivos sin interferencias entre ellos, y también manejar un repertorio de 256 comandos.

La señal que envía el mando se divide en 3 partes, la primera es una señal de comienzo de 9ms, seguido de un espacio de 4,5ms; a continuación, se envía la segunda parte perteneciente a los 8 bits de la dirección del mando seguido de esta misma dirección, pero negada, esta segunda parte dura 27ms; y por último se envía la tercera parte que corresponde con los 8 bits del comando seguido de este mismo comando, pero negado, esta última parte dura también 27ms. Algo característico de este protocolo es que envía sus respectivos negados como un sistema para prevenir errores.



En el otro extremo de esta comunicación le espera el decodificador infrarrojo, que está compuesto de tres terminales, que consisten en un sensor infrarrojo que incluye un demodulador de la banda de 36-38 KHz, un filtro PCM (Pulse Code Modulation), y una preamplificación rechazo de luz ambiental. Este dispositivo está compuesto de tres pines, la señal (SIGNAL), tierra (GND) y voltaje (VCC). Con esto se consigue que la señal que llega al Arduino a través del pin SIGNAL esté ya desmodulada.



También se incluye otra mejora como la escritura y lectura del módulo de memoria mediante el bus I2C (práctica 2), el módulo usado es de 8 bits, por lo que permite escribir información sólo hasta 0xFF (255), por tanto, para escribir un dato superior a este se debe almacenar en dos bytes o crear algún sistema de almacenamiento; y este módulo sólo permite escribir en 128 posiciones ce memoria.
En esta práctica se emplea este módulo para guardar el mayor puntaje obtenido de cada dificultad (hay 10 dificultades), acompañado del nombre de quién lo consiguió. Para lograr esto se desarrolló un sistema de almacenamiento segmentos, es decir, de las posiciones 0 a la 9 se escribe el puntaje restante de unas sucesivas restas entre 255; de las posiciones 10 a la 19 se escribe las veces que se le restó 255 al puntaje; y de las posiciones 20 a la 119 se escribe el nombre de quien lo consiguió. Al haber 10 dificultades, el puntaje restante se almacena en un byte, el número de veces que se le restó 255 en otro byte, y el nombre del usuario (10 caracteres máximo) en 10 bytes.

Tanto para aplicar estas mejoras, como para aplicar la mejora de dificultad se emplearon, por ejemplo, unas herramientas software específicas (el caso del mando IR), y otras herramientas de uso básico utilizados también en la práctica básica.

Realización de la práctica

Para emplear los recursos anteriormente hizo falta la implementación de varios métodos y la declaración de una gran cantidad de variables, entre los métodos usado, se pueden encontrar:

- **setup():** Se inicializa la conexión con el mando infrarrojo, qué utiliza el protocolo NEC, conectado al pin 19 (nivel de interrupción 4). Se inicializan también el monitor serie, y la pantalla LCD. Y finalmente, también se inicializan los distintos pines usados para la conexión con el bus I2C.

```
122
123 Serial3.begin(9600); //Conexión pantalla LCD(TTL).
124 Serial.begin(9600); //Conexión Monitor Serial.
125
126 randomSeed(analogRead(0)); //Semilla para la función random(), asegura números aleatorios distintos.
127 IRLbegin<IR_NEC>(interruptIR); //Mando infrarrojo con protocolo NEC.
```

- **IREvent():** Rutina propia para reasignar los parámetros característicos de la conexión por infrarrojo, es decir, recoge el protocolo usado, la dirección del mando, y también el comando desmodulado y recibido del receptor infrarrojo.

```
151 //Rutina característica de la conexión por infrarrojo.
152 void IREvent(uint8_t protocol, uint16_t address, uint32_t command) {
153   IRProtocol = protocol;
154   IRAddress = address;
155   IRCommand = command;
156 }
```

- **reinicioMem():** Este método es una mejora de memoria. Elimina todos los registros de memoria escribiendo un 0x00 en sus 128 posiciones de memoria.

```
162     for(byte i = 0x00; i <= 0x7F; i++){
163       escribirDatoDir(0x00, i);
164     }
```

- **remoteIRL():** Rutina que registra cada pulsación del mando y lo asigna a una variable según sea el caso, explicado con mayor detalle en el fichero ".ino".

```
203      switch(IRCommand){ //Cada valor corresponde a su botón en el mando.
204        case 0x6897:
205                    numeroIn[avanzaIn] = '0';
206                    avanzaIn++;
```

- **loop():** Además de lo correspondiente a la práctica básica, dentro de esta rutina se llama también a remoteIRL(), se verifica si se ha pulsado todas estas teclas a la vez "*741", de ser así, se salta a reinicioMem(). Además, dentro de la condición juego, se recoge por el Monitor Serie el nombre del jugador y lo imprime por pantalla, so puede omitirlo pulsando el botón central. Además, también se verifica si se ha cambiado o no de modo con el los botones del mando.

```
315      //Se queda en bucle hasta que se introduzca un valor, o se pulse el botón central.
316      while(Serial.available() <= 0){
317        if(PINC>>3 == 0x1D){
318          jugador = "Sin nombre";
319          sinNombre = true;
320          break;
321        }
322      }
```

- **ISR_Juego():** Mientras está a la espera del inicio de juego, si se ha pulsado en el mando el botón >||, se inicia el juego saltando a la rutina cuentaJuego().

```
434  if(inicia){ //Controla el pulsador >|| del mando, para empezar el juego.
435    reinicioJuego(); //Reinicia los valores.
436    finJuego = false; //Deshabilita el flag.
437    numeroSecreto = random(100*dificultad); //Genera un número aleatorio.
438    inicia = false; //Reinicia la variable correspondiente al botón del mando a false.
439    cuentaJuego(); //Salta directamente a la rutina de juego en curso.
440  }
```

- **reboteBotonesJuego():** En este método se recoge la elección de dificultad por parte de los botones del mando o los botones de la placa de expansión. Además, también se verifica si se ha pulsado el botón de abajo para activar la pausa.

```
504  }else if(operacionJuego == 0x2B && dificultad <= 9 && finJuego || (
505    difSup = false;
506    dificultad++;
507    elegirDificultad();
```

- **elegirDificultad():** Método que imprime en pantalla la dificultad elegida actualmente, existen 10 niveles de dificultad, ampliando únicamente el rango de valores en los que el número aleatorio se genera, puede ir desde 0->100, 100->200… 900->1000. Para cada nivel de dificultad se lee desde memoria su respectivo récord junto al nombre del usuario que lo consiguió, ambos se imprimen también en pantalla.

```
539    /**Se lee el nombre, caracter por caracter. Los nombres ocupan 10 bytes,
540      *y se almacenan a partir de la dirección 20..119, un récord por dificultad.
541      **/
542    for(int i = 0; i < 10; i++){
543      aux = (char) leerDir((byte) ((dificultad-1)*10)+20+i); //Dif=1, i= 0 -> (((1-1)*10)+20+0) = 20...29
544      if(aux >= 0x20 && aux <= 0x7E){
545        jugadorRecord += aux; //Recupera sólo estos caracteres válidos.
546      }
547    }
```

- **cuentaJuego():** Mientras el juego está ejecutándose se puede recoger desde el mando las pulsaciones de pausa, en la que detiene el tiempo y pone el cursor intermitente, o la pulsación de reinicio, que reinicia el juego por completo.

```
602  if(pausa){ //Variable obtenida del mando, pausa el juego, detiene contadores.
603    Serial3.write(0xFE);
604    Serial3.write(0x4B);
605    reboteBotonesJuego();
606    contJuego++;
607    if(contJuego == 4) contJuego = 0; //Si se ha llegado al último dígito del 7-segmentos, se reinicia a 0.
```

- **guardaPuntaje():** Método que escribe en memoria el puntaje récord junto a su usuario. Primero lee desde memoria el puntaje récord actual, y lo compara con el puntaje obtenido en el juego actualmente, si es mayor guarda el nuevo puntaje récord en memoria junto a su usuario e imprime un mensaje avisando de nuevo récord, si no lo es, simplemente imprime el mensaje de victoria.

```
762    //Entonces, se guarda el puntaje en memoria.
763    while(puntaje > 255){
764      puntaje = puntaje - 255;
765      dosCinco++; //Se guarda las veces que se le resta 255.
766    }
767    escribirDatoDir((byte) dosCinco, (byte) dificultad-1+10); //Se guarda nVeces le resta 255, en posiciones 10..19.
768    escribirDatoDir((byte) puntaje, (byte) dificultad-1); //Y el puntaje restante, en posiciones 0..9.
769
770    //Se guarda también, el nombre del jugador, sólo 10 caracteres.
771    for(int i = 0; i < 10; i++){
772      escribirDatoDir((byte) jugador.charAt(i), (byte) ((dificultad-1)*10)+20+i); //En posiciones 20..119.
773    }
```

Como siempre, para cualquier duda, el fichero ".ino" contiene comentarios que explican su funcionamiento.

Conclusiones

Las mejoras realizadas en esta práctica mejoraron mi comprensión de los distintos protocolos de conexión por infrarrojo, así como montar circuitos electrónicos, manejar los distintos periféricos mediante interrupciones, desarrollar sistemas y algoritmos para el almacenamiento de datos en memoria y para la implementación de un programa multifuncional como este.
Asimismo, agradecer todas las explicaciones y dedicación aportados por el profesorado que sirvieron de base para la correcta implementación de esta práctica.

Para finalizar, se ha adjuntado la librería usada en el proyecto, IRLremote.zip, junto al PDF y el fichero ".ino".