"MANEJO DE COMUNICACIÓN SERIAL UART y SPI EN ARDUINO"

Trabajo Preparatorio N°8
Laboratorio de Sistemas Embebidos

Melanny Cecibel Dávila Pazmiño
Ingeniería en Telecomunicaciones
Facultad de Eléctrica y Electrónica
Quito, Ecuador
melanny.davila@epn.edu.ec

***Abstract*—En el siguiente documento, se presenta el fundamento teórico acerca de la comunicación serial UART y SPI en Arduino, donde se busca enfatizar la numeración de los pines de la placa Arduino UNO, su modo de comunicación y las conexiones existentes.**

***Index Terms*—Arduino, comunicación serial, UART, SPI.**

I. Introducción

La comunicación serial entre dos dispositivos se realiza a través del intercambio de una secuencia de bits, donde se transmite bit a bit, uno por vez, donde, aunque es lenta la comunicación, tiene la ventaja de poder ser transmitida a mayores distancias y utilizar menos líneas de comunicación.

La comunicación serial entre dos dispositivos únicamente utiliza 3 líneas las cuales son:

- Línea de recepción de datos (RX)
- Línea de transmisión de datos (TX)
- Línea común (GND)

Donde GND, RX y TX del microcontrolador Arduino son fácilmente identificables en la placa. El TX y RX de Arduino son los dos pines que emplea el dispositivo para realizar la comunicación por medio del protocolo serial. Los datos, por lo tanto son transmitidos en la línea o pin TX y son recibidos por la línea o pin RX [1].

II. Objetivos

- Relacionar al estudiante con el uso y manejo de comunicación serial y SPI de la placa de desarrollo Arduino Uno.
- Identificar diferencias entre comunicación serial UART y SPI en Arduino [2].

III. Cuestionario

A. Consultar las características relacionadas al uso de Comunicación Serial UART en la placa Arduino, enfatizar los modos de comunicación, tipos de conexiones y tramas de comunicación.

UART o USART es el puerto serie hardware que todos los microcontroladores tienen al menos uno y la comunicación serie es la base de casi cualquiera de las comunicaciones de los microcontroladores [2].

La comunicación serial también se da mediante variaciones de voltaje, donde dispositivos TTL (transistor transistor logic) realizan la comunicación mediante variaciones de señal entre 0 y 5v o entre 0 y 3.3v [2].

Tipo de conexión

UART (universally asynchronous receiver/transmitter) significa recepción y transmisión asíncronas universales y es un protocolo de comunicación simple que permite que Arduino se comunique con dispositivos serie. El sistema UART usa los pines digitales 0 (RX) y 1 (TX) y con otro PC a través del puerto USB [1].

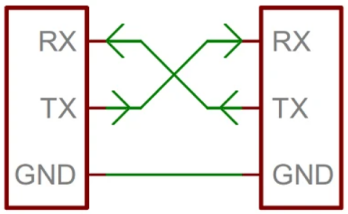

Fig. 1. Sistema UART

Este periférico, que se encuentra en todas las placas Arduino, permite que Arduino se comunique directamente con un PC gracias al hecho de que Arduino tiene un convertidor de USB a serie incorporado (ATmega16U2). Las placas de Arduino poseen unidades UART que operan a nivel TTL 0 / 5v, lo que las vuelve compatibles con la conexión USB [3].

Tipo de comunicación

La comunicación puede ser establecida de tres formas:

- Simplex: un solo dispositivo envía información durante la comunicación.
- Half-duplex: todos los dispositivos pueden enviar información, pero solo uno a la vez.
- Full-duplex: todos los dispositivos pueden enviar información a la vez.

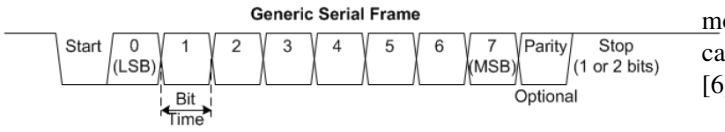

Fig. 2. Trama UART genérica

Trama UART

- Bit de inicio : Cada trama de datos comienza con un bit de inicio. Esto le permite al dispositivo receptor saber que los datos est´an a punto de enviarse.
- Información: El bloque de datos se envía inmediatamente después del bit de inicio. Por lo general, se envía un byte completo de 8 bits en cada trama. Sin embargo, en realidad tiene la opción de enviar entre 5 y 8 bits de datos en el bloque.
- Bit de paridad: El bit de paridad se utiliza para comprobar los errores del paquete cuando se recibe. Hay dos tipos de verificación de paridad; par e impar.
- Bit de parada: El bit de parada es el final del paquete.

La mayoría de las veces sólo se envía un bit de parada y se indica como la línea que regresa al estado alto inactivo. Se puede tener hasta dos bits de parada [4].

B. Consultar las características del Protocolo de Comunicación sincrónica SPI en la placa Arduino, enfatizar la distribución de pines, modos de comunicación, y tipos de conexiones.

La Interfaz de periféricos en serie (SPI) es un protocolo de datos en serie síncrono utilizado por microcontroladores para comunicarse con uno o más dispositivos periféricos rápidamente en distancias cortas. También se puede utilizar para la comunicación entre dos microcontroladores [5].

El puerto SPI permite un solo dispositivo maestro con un máximo de cuatro dispositivos esclavos.

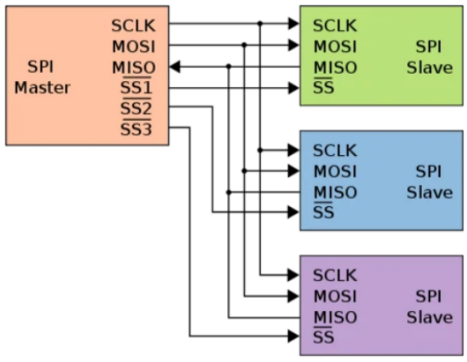

Fig. 3. Sistema SPI

SPI es mucho más rápido que I2C debido al protocolo simple y, aunque las líneas de datos/reloj se comparten entre dispositivos, cada dispositivo requiere un cable de dirección único. SPI se encuentra comúnmente en lugares donde la velocidad es importante, como con las tarjetas SD y los módulos de pantalla, o cuando la información se actualiza y cambia rápidamente, como con los sensores de temperatura [6].

Tipo de comunicación:

Normalmente, hay tres líneas comunes a todos los dispositivos:

- MISO (Master In Slave Out): línea esclava para enviar datos al maestro [6].
- MOSI (Master Out Slave In): línea maestra para enviar datos a los periféricos.
- SCK (Reloj en serie): los pulsos de reloj que sincronizan la transmisión de datos generada por el maestro.
- SS (Selección de esclavo): el pin en cada dispositivo que el maestro puede usar para habilitar y deshabilitar dispositivos específicos (conexión punto-multipunto) [6]. Cuando el pin de selección de esclavo de un dispositivo está bajo, se comunica con el maestro. Cuando está alto, ignora al maestro. Esto le permite tener varios dispositivos SPI que comparten las mismas líneas MISO, MOSI y CLK [7].

Pines de conexión de la placa Arduino

TABLA I
PINES DE LA PLACA ARDUINO UNO

Descripción	Número de Pin
MOSI	11 o ICSP-4
MISO	12 o ICSP-1
SCK	13 o ICSP-3
SS (esclavo)	10

C. Investigar el uso de la librería SPI.h, comandos, objetos de configuración y aplicaciones.

- SPISettings: El objeto SPISettings se utiliza para configurar el puerto SPI para su dispositivo SPI. Los 3 parámetros se combinan en un solo objeto SPISettings, que se proporciona a SPI.beginTransaction (). Sus parámetros son: speedMaximum que establece la velocidad de la comunicación, speedMaximum que establece si se envía el bit mas significativo o el menos significado antes dependiendo de su configuración y dataMode que establece el modo de envío de la comunicación [7].
- begin(): Inicializa el bus SPI configurando SCK, MOSI y SS en salidas, SCK y MOSI en bajo y SS en alto.
- usingInterrupt(): Si el programa realizará transacciones SPI dentro de una interrupción, llame a esta función para registrar el número o nombre de la interrupción en la biblioteca SPI. Esto permite que SPI.beginTransaction() evite conflictos de uso. Se debe tener en cuenta que la interrupción especificada en la llamada a usingInterrupt() se desactivará en una llamada a beginTransaction() y se volverá a activar en endTransaction() [8].
- beginTransaction(): Inicializa el bus SPI usando los SPISettings definidos.
- endTransaction(): Deja de usar el bus SPI. Normalmente, esto se llama después de anular la afirmación de la

selección de chip, para permitir que otras bibliotecas usen el bus SPI [7].

- transfer(), transfer16(): La transferencia SPI se basa en un envío y una recepción simultáneos: los datos recibidos se devuelven en selectedVal (o selectedVal16). En el caso de transferencias de buffer, los datos recibidos se almacenan en el buffer (los datos antiguos se reemplazan por los datos recibidos).
- setClockDriver(): Es una alternativa para establecer la velocidad de transferencia de datos (la velocidad del reloj, que es el que marca los pulsos, en realidad) [8].

D. Investigar tres ejemplos de dispositivos o módulos de comunicación inalámbrica relacionados con la Industria 4.0 que operan en conjunto con los diferentes modelos de placas Arduino, mediante los tipos de comunicación serial UART y SPI. Detallar la aplicación del conjunto, procesos de conexión y configuración.

- RFM69HCW: Es un módulo transceptor capaz de funcionar en un amplio rango de frecuencias, incluidas las bandas de frecuencia ISM (Industry Scientific and Medical) sin licencia de 315,433,868 y 915MHz. Todos los principales parámetros de comunicación de RF son programables y la mayoría de ellos se pueden configurar dinámicamente [9]. El RFM69HCW ofrece la ventaja única de los modos de comunicación programables de banda estrecha y banda ancha. Trabaja con modulaciones FSK, GFSK, MSK, GMSK y OOK. Aplicaciones del módulo transceptor RF RFM69HCW:
  - Lectura de medidor automatizada.
  - Redes de sensores inalámbricos.
  - Automatización de viviendas y edificios.
  - Sistemas de seguridad y alarma inalámbricos.
  - Monitoreo y control industrial.
  - M-BUS inalámbrico [9].
- MKR ZERO: tiene un conector SD integrado con interfaces SPI dedicadas (SPI1) que le permite almacenar datos, reproducir archivos de audio en una memoria microSD y conectar una batería LiPo para un proyecto inalámbrico sin hardware adicional [10].
- MKR WAN 1400: Aprovecha la red celular como medio de comunicación. La red GSM/3G es la que cubre el mayor porcentaje de la superficie mundial, lo que hace que esta opción de conectividad sea muy atractiva cuando no existen otras opciones de conectividad. El procesador principal de la placa es un SAMD21 de 32 bits de bajo consumo, como en las otras placas de la familia Arduino MKR. La conectividad GSM/3G se realiza con un módulo de u-blox, el SARA-U201, un chipset de baja potencia que opera en las diferentes bandas del rango celular (GSM 850 MHz, E-GSM 1900 MHz, DCS 1800 MHz, PCS 1900 MHz). Además de eso, la comunicación segura está garantizada a través del chip criptográfico Microchip ECC508. Además de eso, puede encontrar un cargador de batería y un conector para una antena externa [11].

E. Consultar el funcionamiento del LCD de 16x2, pines de configuración, librerías y códigos a utilizar para la visualización de información.

El LCD(Liquid Crystal Dysplay) o pantalla de cristal líquido es un dispositivo empleado para la visualización de contenidos o información de una forma gráfica, mediante caracteres, símbolos o pequeños dibujos dependiendo del modelo. Está gobernado por un microcontrolador el cual dirige todo su funcionamiento [12].

Pasos para el uso de un LCD con Arduino:

- Conectar la pantalla LCD 16×2 a la alimentación de 5 volts, incluyendo la alimentación de la iluminación LED.
- Colocar un potenciómetro para el ajuste de contraste.
- Conectar los pines de datos a la pantalla (modo de 4 bits o modo de 8 bits).
- Conectar los pines de control RS y EN (de manera opcional el pin RW) [12].

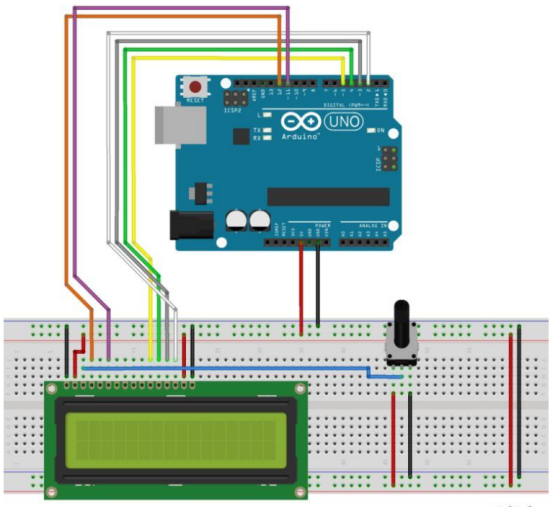

Fig. 4. Esquematización de un LCD

Los pines del LCD son los siguientes:

- Pin 1 – Vss: GND o tierra.
- Pin 2 – Vdd: Alimentación Vcc o +5V.
- Pin 3 – V0: Control del contraste del display, conectamos este pin al terminal variable de un potenciómetro conectado a Vcc y Masa en sus terminales extremos.
- Pin 4 – RS: Selección de Registro.
  - 0 lógico: Registro de comandos (escritura),
  - 1 lógico: Registro de datos (escritura, lectura)
- Pin 5 – R/W:
  - 0 lógico: Escritura del LCD.
  - 1 lógico: Lectura del LCD.
- Pin 6 – Enable: Un 1 lógico señala el inicio de escritura o lectura del LCD, un 0 lógico, desactiva todas las funciones.
- Pin 7-10 – D0/D3: Pines correspondientes al bus de datos. D0 corresponde al bit menos significativo. Estos pines no

se utilizan si se realizan operaciones sobre el LCD de 4 bits.

- Pin 11-14 – D4/D7: Pines correspondientes al bus de datos. D7 corresponde al bit más significativo y puede utilizarse como “Busy Flag”. Un 1 lógico indicará que el LCD se encuentra ocupado, no permitiendo realizar ninguna operación hasta que se deshabilite.
- Pin 15 – Ánodo de la retroiluminación : R + 5V.
- Pin 16 – Cátodo de la retroiluminación: GND [13].

Para poder visualizar los caracteres o símbolos en el LCD es necesario que en el programa de código fuente a emplear, se incluya la librería ”lcd.c”. Siempre que se utilice esta librería se debe analizarla para saber cuales son los pines de control y los pines para el Bus de datos, en este caso podemos observar que están definidos al comienzo de la misma [13].

$\#define\ LCD\_ENABLE\_PIN\ PIN\_D0$
$\#define\ LCD\_RS\_PIN\ PIN\_D1$
$\#define\ LCD\_RW\_PIN\ PIN\_D2$
$\#define\ LCD\_DATA4\ PIN\_D4$
$\#define\ LCD\_DATA5\ PIN\_D5$
$\#define\ LCD\_DATA6\ PIN\_D6$
$\#define\ LCD\_DATA7\ PIN\_D7$

En el resto de la librería se puede encontrar todas las estructuras necesarias así como las funciones que nos permiten utilizar nuestro LCD [12].

Presenta funciones como :

- lcd_init : inicializa el lcd.
- lcd_gotoxy : establece la posicion de escritura en el lcd.
- lcd_putc : nos muestra un dato en la siguiente posición del lcd, podemos emplear funciones como \f para limpiar el display, \n cambio a la segunda línea, mueve una posición atrás.
- $lcd\_getc(x, y)$: devuelve caracteres a la posición x,y.
- Otras funciones: lcd_send_nibble(BYTE n), lcd_send_byte(BYTE address, BYTE n) [13].

F. Realizar un programa en Tinkercad que permita simular un sistema de alarma casero, para ello se deberá emplear 2 placas de desarrollo Arduino Uno interconectadas mediante el protocolo de comunicación UART. En la primera placa Arduino Uno se utilizarán los pines analógicos y digitales para el uso de barrido de teclado y barrido de display, los mismos que permitirán ingresar y mostrar al usuario la clave ingresada. En la segunda placa Arduino Uno se incorporarán tres sensores de movimiento PIR que simularán el monitoreo de tres habitaciones una vez que el sistema se encuentre armado. Si se detecta movimiento en cualquiera de las habitaciones se activará una alarma sonora y solicitará el ingreso por teclado de la clave correcta, luego de tres intentos erróneos el sistema se bloqueará por 30 segundos.

A continuación, se presenta el esquemático realizado en el software de simulación Tinkercad junto con el código fuente que permite implementar lo solicitado.

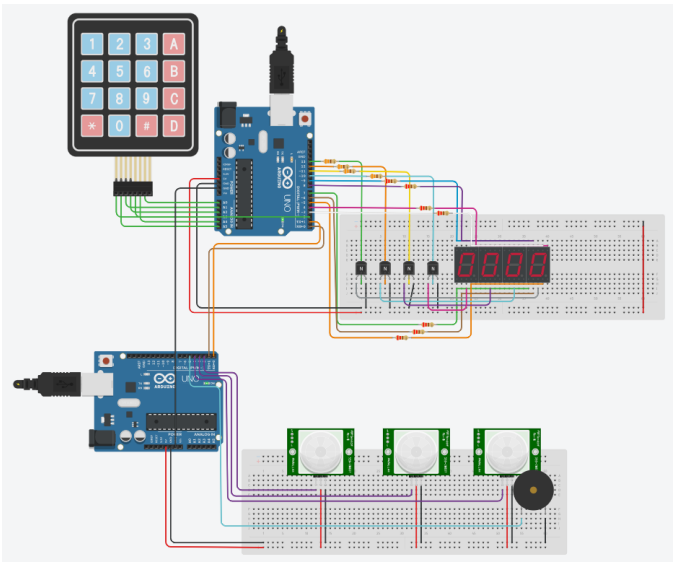

Fig. 5. Esquematización del sistema de alarma casero

```
/*Se implementa el control de una alarma casera, antes se debe
configurar la contrasena, la misma que permitira desactivar la
alarma en el caso que se detecte movimiento mediante un sensor
PIR. Sila contrasena se ingresa 3 veces de manera incorrecta se
bloqueara el teclado durante 2 segundos y se podra ingresar
nuevamente la contrasena ya sea correcta o incorrecta*/

//Libreria para controlar el teclado
#include <Keypad.h>

//Declaracion para el teclado
const byte ROWS = 4; //filas
const byte COLS = 3; //columnas
int aux = 0;//variable auxiliar
int numint = 3; //se define el numero de intentos
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[ROWS] = {2, A5, A1, A2};
byte colPins[COLS] = {A3, A4,A0};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
//Seccion para la declaraci n de variables que se
//van a usar para el barrido de display
int display7c[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};
byte a=3;
byte b=4;
byte c=5;
byte d=6;
byte e=7;
byte f=8;
byte g=9;
//Declaraci n pines de Tbjs
byte t1=10;   //mil
```

```
byte t2=11;  //centena
byte t3=12;  //decena
byte t4=13;  //unidad
//Contadores para hacer el temporizador
long tempor=0,contret=0;
int ans1;
int contador;
//Funci n que coloca en el puerto de salida los bits comenzando
//desde el pin de ini hasta el pin fin

void puerto(int bits, int ini,int fin)
{
  for(int i=ini;i<=fin;i++)
  {
    digitalWrite(i,bitRead(bits,i-ini));
  }
 }
//Funcion encargada de la multiplexaci n de los displays
void mostrar()
{
  int dig[4];
  //digito de mil
  dig[0]=tempor/1000;
  //digito de centena
  dig[1]=(tempor-dig[0]*1000)/100;
  //digito de decena
  dig[2]=(tempor-dig[0]*1000-dig[1]*100)/10;
  //digito de unidad
  dig[3]=(tempor-dig[0]*1000-dig[1]*100-dig[2]*10);

  //Rutina de multiplexacion
  for(int i=t1;i<=t4;i++){
    puerto(display7c[dig[i-t1]],a,g);
    digitalWrite(i,HIGH);  //Encende el display de unidades
    delay(1);
    digitalWrite(i,LOW);   //Apaga el display de unidades
  }
}

void temporizacion()    //lazo de temporizaci n de retardo
{
  //Valor de contret bajo
  contret=2;
  while (contret>0)
  {
    mostrar();
    contret--;
  }
}

//Variables de control del programa

char total[4] = {'0','0','0','0'}; //vector de contrasena
byte index=0;//indice del vector de la contrasena
bool contraseniaNueva = 1;//Control para ingresar una contrasena nueva
bool suenaAlarma = 1;//Control para habilitar el teclado para ingresar una contrasena
int contra;//variable que contiene la contrasena
int numInt=0;//numero de intentos realizados
void setup()
{
  //Inicio del temporizador
  tempor = atoi(total);
  //Configuracion de 8 pines digitales como salida
  for(int i=a;i<=t4;i++){
    pinMode(i,OUTPUT);
  }
  Serial.begin(9600);
  Serial.setTimeout(1);//Tiempo de espera para la recepcion de
  //datos (1 ms)
}
//Verificacion de contrasena
void verificar(int password)
{
  //En el caso de que la contrenia sea correcta se pide a la
  //otra placa que se apague la alarma
   if (password==contra)
   {
     int aux = 0; //variable usada para hacer un flujo de
     //datos y asegurar que la informacion llegue a la otra placa
      while (aux <20)
      {
       Serial.println(7);
         aux++;
      }
   }
   else
   {
     //En el caso de que la contrasenia no sea
     //correcta se incrementa en 1 el conteo de intentos
      //y se verifica si no son 3 intentos
       numInt++;
       if (numInt>2)
       {
         /*En el caso de ser 3 intentos incorrectos se bloquea el
         teclado durante 2 segundos y se reinicia el conteo de
         intentos*/
         delay(2000);
         numInt = 0;
       }

   }
}
//Funcion para recibir datos
void recibirSenial()
{
  //Lectura del buffer de datos, los bits ingresan al lazo if
  int incomingByte = Serial.parseInt();
  if (Serial.available() > 0)
  {
    //Lectura byte de entrada
    int incomingByte = Serial.parseInt();
  if (incomingByte>= 50 && incomingByte<=100)
    {
      /*Si el sensor PIR envia un numero distinto entre 50 y 100
      cuando se enciende, esto es detectado y se habilita la
      opcion para ingresar la contrasena*/
       suenaAlarma = 1;
    }
  }
}
void recibirDatos()
{
  //Llamada a la funcion temporizacion para que los display
  //esten constantemente refrescandose
  temporizacion();
  char key = keypad.getKey();//captura de la
```

```
    tecla presionada
  if (key != NO_KEY && suenaAlarma)
  {
    //suenaAlarma esta en verdadero al inicio
    para poder ingresar la
    //contrasenia
     total[index]= key; //se guardan los valores
     en el vector
     tempor = atoi(total);
     index++;
     if (contraseniaNueva && index==4)
     {
       suenaAlarma = 0;//Una vez se ingresa la
    contrasenia se bloquea
       //el teclado
       int aux = 0;//Variable usada para mostrar
     el numero ingresado
       while (aux<20)
       {
         temporizacion();
         aux++;
       }
       contra = tempor;//Se guarda el valor de
    la contrasenia
       contraseniaNueva = 0;//Se bloquea el
    teclado
 //Se reinician las variables que controlan el
    vector y el display
       index = 0;
       tempor =0;

       for (int i=0;i<4;i++)
       {
        total[i] = '0'; //Designacion de '0000'
    para mostrar
       }
     }
//Ingreso de contrasenas para ser verificadas
     else if (!contraseniaNueva && index==4  &&
    suenaAlarma)
     {
       aux = 0;//Se muestra el valor ingresado
       while (aux<20)
       {
         temporizacion();
         aux++;
       }
      int password = tempor; //Se guarda en una
    variable el valor
       //ingresado
        verificar(password);//Se verifica la
    contasenia
   //Reinicio de las variables que controlan el
    vector y el display
       index = 0;
        tempor =0;
       for (int i=0;i<4;i++)
       {
        total[i] = '0'; //Designacion de '0000'
    para mostrar
       }
     }
  }
}
void loop() //funcion principal
{
  //Llamado de las funciones
  recibirDatos();
  recibirSenial();
}
```

Código 1: Implementación del alarma Arduino 1

-
```
/*Se implementa el control de una alarma casera,
    antes se debe
configurar la contrasena, la misma que permitira
    desactivar la
alarma en el caso que se detecte movimiento
    mediante un sensor
PIR. Sila contrasena se ingresa 3 veces de
    manera incorrecta se
bloqueara el teclado durante 2 segundos y se
    podra ingresar
nuevamente la contrasena ya sea correcta o
    incorrecta*/


//Se encarga de controlar los sensores PIR
bool alarma = 0;//La alarma se encuentra apagada
void setup()
{
  //Se coloca en modo de entrada los pines del
    sensor PIR
  pinMode(4, INPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  Serial.begin(9600);
  Serial.setTimeout(10);//Tiempo de espera
}
void recibirInfo()
{
  int incomingByte = Serial.parseInt();
  if (Serial.available() > 0)
  {
    // Se leen los datos de entrada
    int incomingByte = Serial.parseInt();
    Serial.println(incomingByte);
  if (incomingByte>0)
    {
       alarma = 0;//La funcion verificar de la
    otra placa envia una
      //senial para poder apagar la alarma
    }
  }
}
void loop()
{
  //Cada sensor PIR envia un codigo distinto al
    detectar movimiento
  recibirInfo();
  if (digitalRead(4)==HIGH)
  {
     Serial.println(100);
     alarma = 1;//Se da la senal para sonar la
    alarma
  }
  if (digitalRead(2)==HIGH)
  {
     Serial.println(750);
     alarma = 1;//Se da la senal para sonar la
    alarma
  }
  if (digitalRead(3)==HIGH)
  {
     Serial.println(500);
     alarma = 1;//Se da la senal para sonar la
    alarma
  }
  if (alarma)
  {
    tone(5, 399, 5); //Suena la alarma
  }
}
```

Código 2: Implementación del alarma Arduino 2

REFERENCES

[1] "Comunicación Serial con Arduino - [enero, 2021]", Control Automático Educación, jun. 15, 2019.

https://controlautomaticoeducacion.com/arduino/comunicacion-serial-con-arduino/ (accedido ene. 25, 2021).
[2] E. Espinosa, E. Tatayo, “MANEJO DE COMUNICACIÓN SERIAL UART y SPI EN ARDUINO”. C.P. SISTEMAS EMBEBIDOS, Accedido: ene. 28, 2021. [En línea].
[3] “UART y USB en Arduino”, Aprendiendo Arduino, nov. 09, 2016. https://aprendiendoarduino.wordpress.com/2016/11/09/uart-y-usb-en-arduino/ (accedido ene. 25, 2021).
[4] “Comunicaciones serie en Arduino: UART, I2C y SPI”, kolwidi. https://kolwidi.com/blogs/blog-kolwidi/comunicaciones-serie-en-arduino-uart-i2c-y-spi (accedido ene. 25, 2021).
[5] “Como usar un Arduino UNO/MEGA como programador ISP — Sysadmins de Cuba”. https://www.sysadminsdecuba.com/2020/08/como-usar-un-arduino-uno-mega-como-programador-isp/ (accedido ene. 25, 2021).
[6] “Arduino - SPI”. https://www.arduino.cc/en/Reference/SPI (accedido ene. 25, 2021).
[7] “El bus SPI en Arduino”, Luis Llamas. https://www.luisllamas.es/arduino-spi/ (accedido ene. 25, 2021).
[8] P. Stoffregen “SPI”, GitHub. https://github.com/PaulStoffregen/SPI (accedido ene. 25, 2021).
[9] “RFM69HC 20dBm Programmable 315-915Mhz RF Transceiver Module Sub 1G Programmable 315-915Mhz RF Transceiver Module — HOPERF”. https://www.hoperf.com/modules/rf-transceiver/RFM69HCW.html (accedido ene. 25, 2021).
[10] “Arduino MKR ZERO — Arduino Official Store”. https://store.arduino.cc/usa/arduino-mkrzero (accedido ene. 25, 2021).
[11] “Arduino MKR GSM 1400 — Arduino Official Store”. https://store.arduino.cc/usa/mkr-gsm-1400 (accedido ene. 25, 2021).
[12] W. Fox, “TodoElectrodo: Lcd 16x2”, TodoElectrodo, feb. 10, 2013. https://todoelectrodo.blogspot.com/2013/02/lcd-16x2.html (accedido ene. 25, 2021).
[13] K. Casimiro, “LCD funcionamiento del display de 16x2”, Accedido: ene. 25, 2021. [En línea]. Disponible en: https://www.academia.edu/10937574/LCD-funcionamiento-del-display-de-16x2.