Proyecto Final: Lector de Braille

Nombre	Carné	Participación
Ramiro A. Chacón Castañeda	2915963551903	100%
Jeysson E. Godoy Torres	3429393512210	100%
Mario R. Palma Villeda	3384151142007	100%
Erick E. Pineda Palma	2472451942001	100%
Eduardo R. Cruz Sánchez	2472597802001	100%

Universidad de San Carlos de Guatemala

964: Laboratorio de Organización Computacional

Aux. Javier Alejandro Gutiérrez de León

Viernes 28 de junio 2024

Contenido

Introducción

El diseño del lector de Braille se basa en la implementación de diversos componentes electrónicos y mecánicos, incluyendo Flip Flops, contadores, transmisiones seriales, motores paso a paso, y sensores de color. Además, se hace uso de tecnologías como Arduino y Proteus para la creación de una memoria RAM dinámica y la gestión de datos en tiempo real.

La motivación detrás de este proyecto es contribuir a la comunidad de personas con discapacidad visual en Guatemala, quienes enfrentan desafíos significativos debido al deterioro de los libros educativos en Braille.

Este documento presenta una descripción detallada del proyecto, los objetivos específicos, la metodología de trabajo, y los resultados obtenidos.

Descripción del Problema

Prociegos y Sordos de Guatemala con el fin de lograr una verdadera inclusión ha creado muchos libros educativos para personas ciegas que abarcan grados desde primaria hasta diversificado, el problema es que con el pasar del tiempo muchas de las páginas de los libros se han deteriorando y los lectores ya no pueden entender las letras, por lo que usted con los conocimientos adquiridos en el curso de Organización Computacional deberá crear un prototipo de verificación del texto. El prototipo será muy sencillo ya que se requiere saber si usted es capaz de reconocer las letras antes de hacer una inversión más grande para la lectura de páginas completas, por lo que se presentará por medio de tarjetas cada una de las letras.

Lógica del Sistema

El objetivo final de la estructura es desarrollar un lector de tarjetas en braille. Este sistema está diseñado para facilitar la lectura de tarjetas braille mediante el uso de sensores de color y tecnología Arduino.

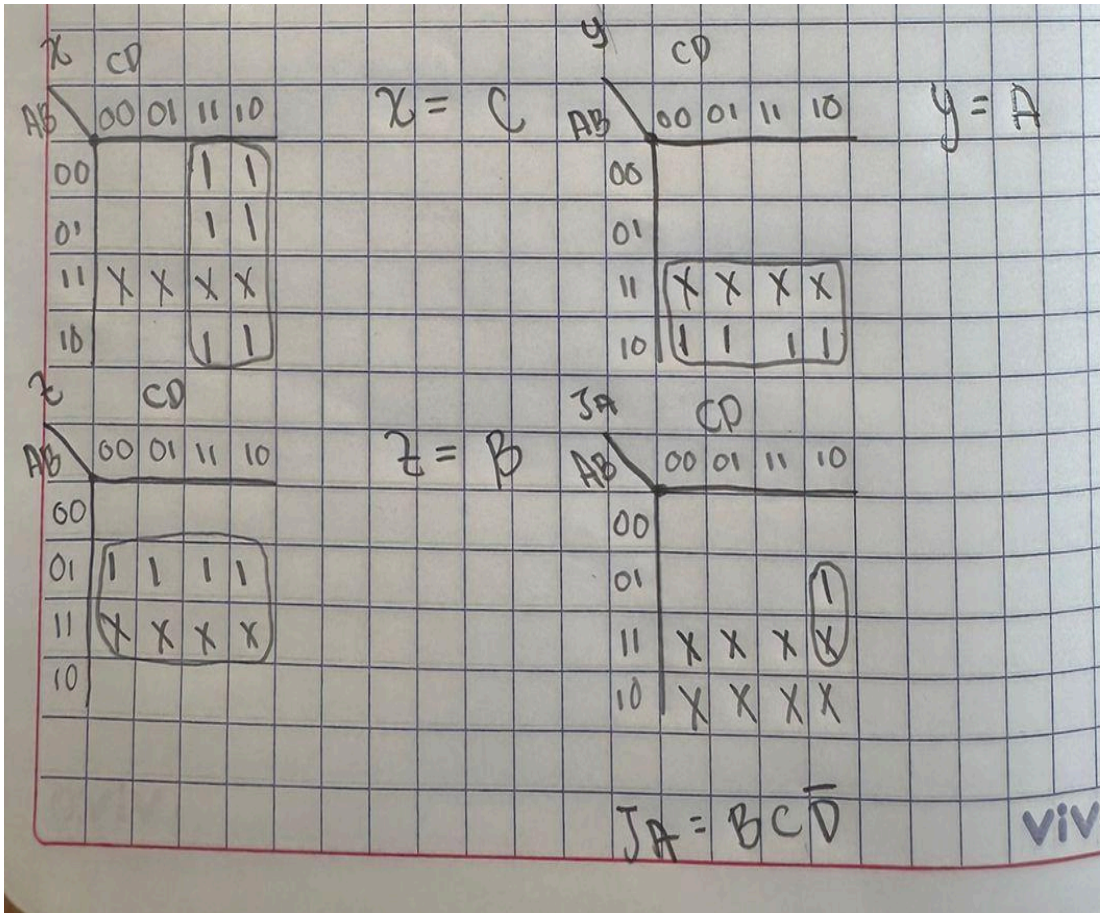
En la parte física del proyecto, se emplearán cinco sensores de color. Cuatro de estos sensores tienen la función de asegurarse de que la tarjeta esté en la posición correcta para la lectura. El quinto sensor es el encargado de leer la información de la tarjeta en sí. El sistema interpretará los colores detectados: el color negro se considerará como un “1” y el color blanco como un “0”. Esta codificación binaria permitirá representar el contenido de la tarjeta en un formato digital comprensible para el sistema.

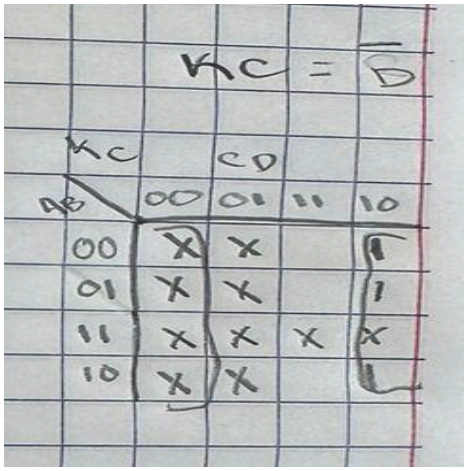
Los datos obtenidos por los sensores de color serán enviados mediante un conjunto de placas Arduino UNO. Estas placas transmitirán la información a un Arduino Mega, que estará simulado en el entorno de Proteus. El Arduino Mega se encargará de procesar y almacenar los datos en una memoria RAM de dimensiones 7 x 8, es decir, una matriz que puede almacenar 8 conjuntos de datos de 7 bits cada uno.

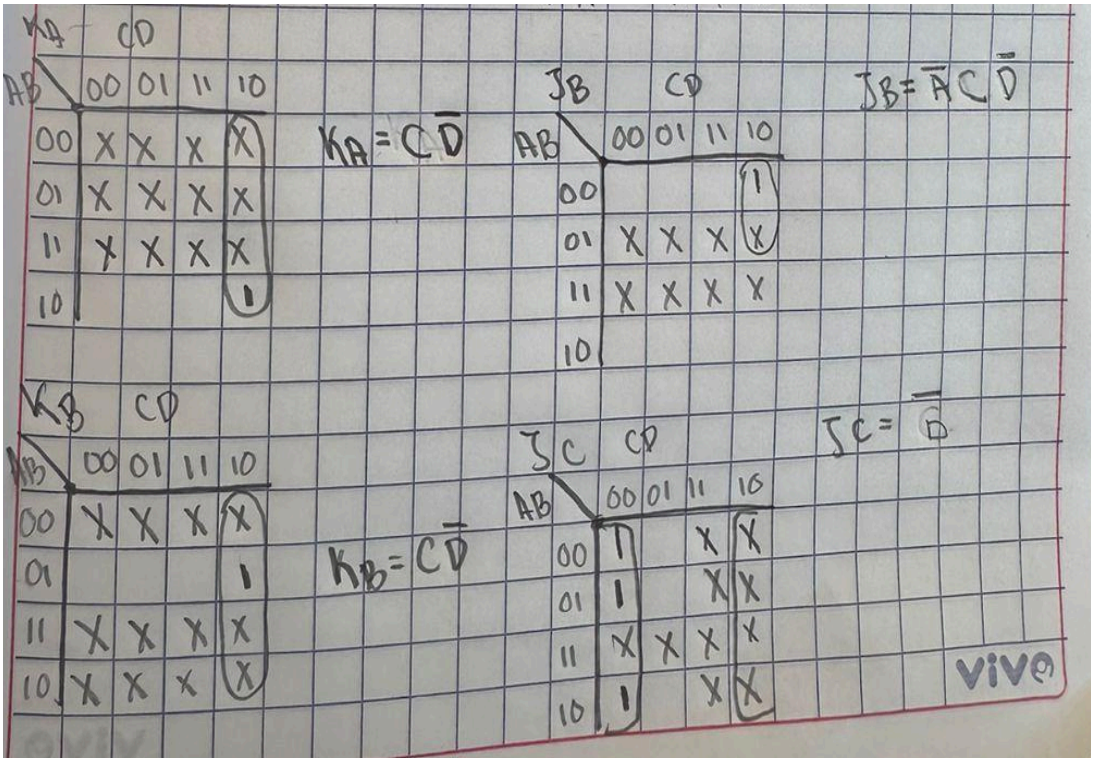
Una vez que se hayan leído las ocho tarjetas, el sistema presentará la información recopilada en una interfaz. Esta interfaz podría estar diseñada para mostrar los datos de manera visual o mediante otra forma accesible, facilitando la interpretación de la información contenida en las tarjetas braille.

Funciones Booleanas y Mapas K

Contador

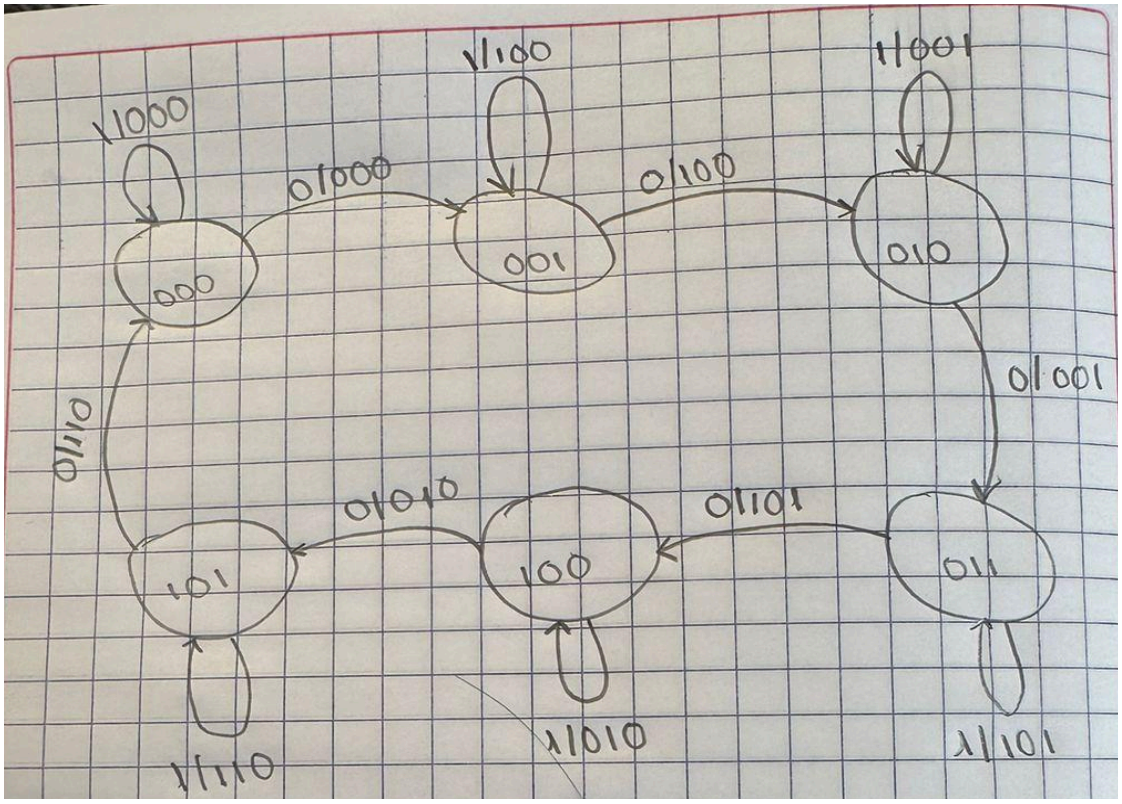


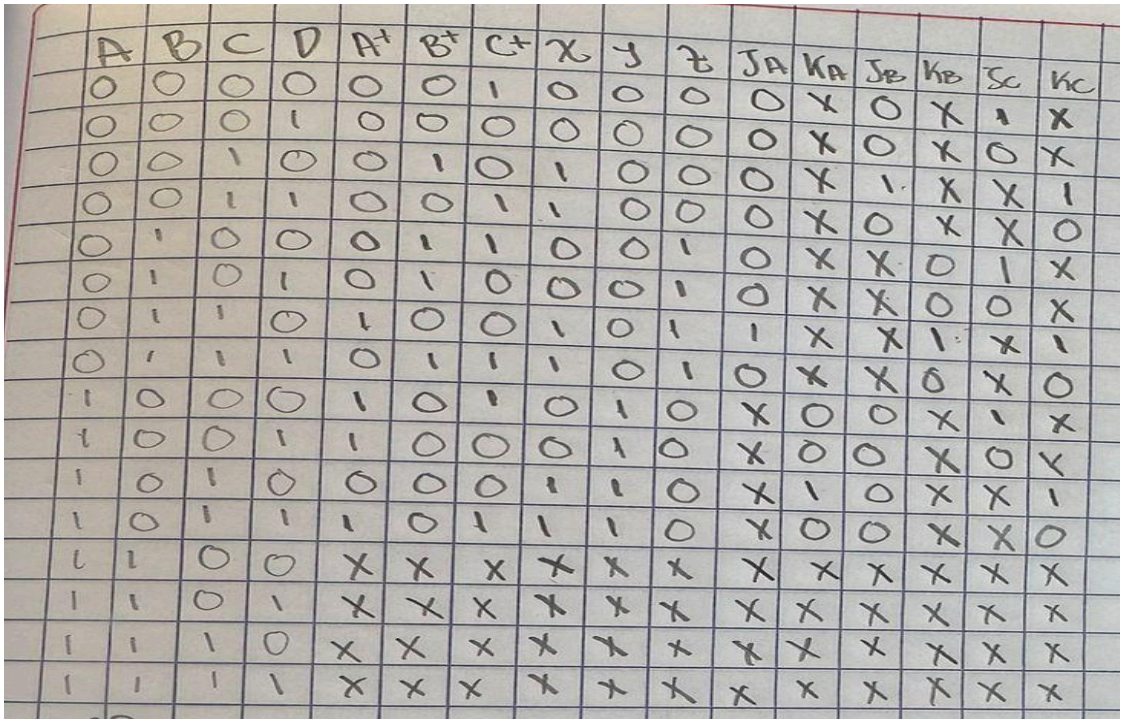



KA CD
AB 00 01 11 10
00 X X X X
01 X X X X
11 X X X X
10 1
KA = C D̄
JB CD
AB 00 01 11 10
00 1
01 X X X X
11 X X X X
10
JB = Ā C D̄
KB CD
AB 00 01 11 10
00 X X X X
01 1
11 X X X X
10 X X X X
KB = C D̄
JC CD
AB 00 01 11 10
00 1 X X
01 1 X X
11 X X X X
10 1 X X
JC = D̄
vivo

Diagramas de Estado

Contador





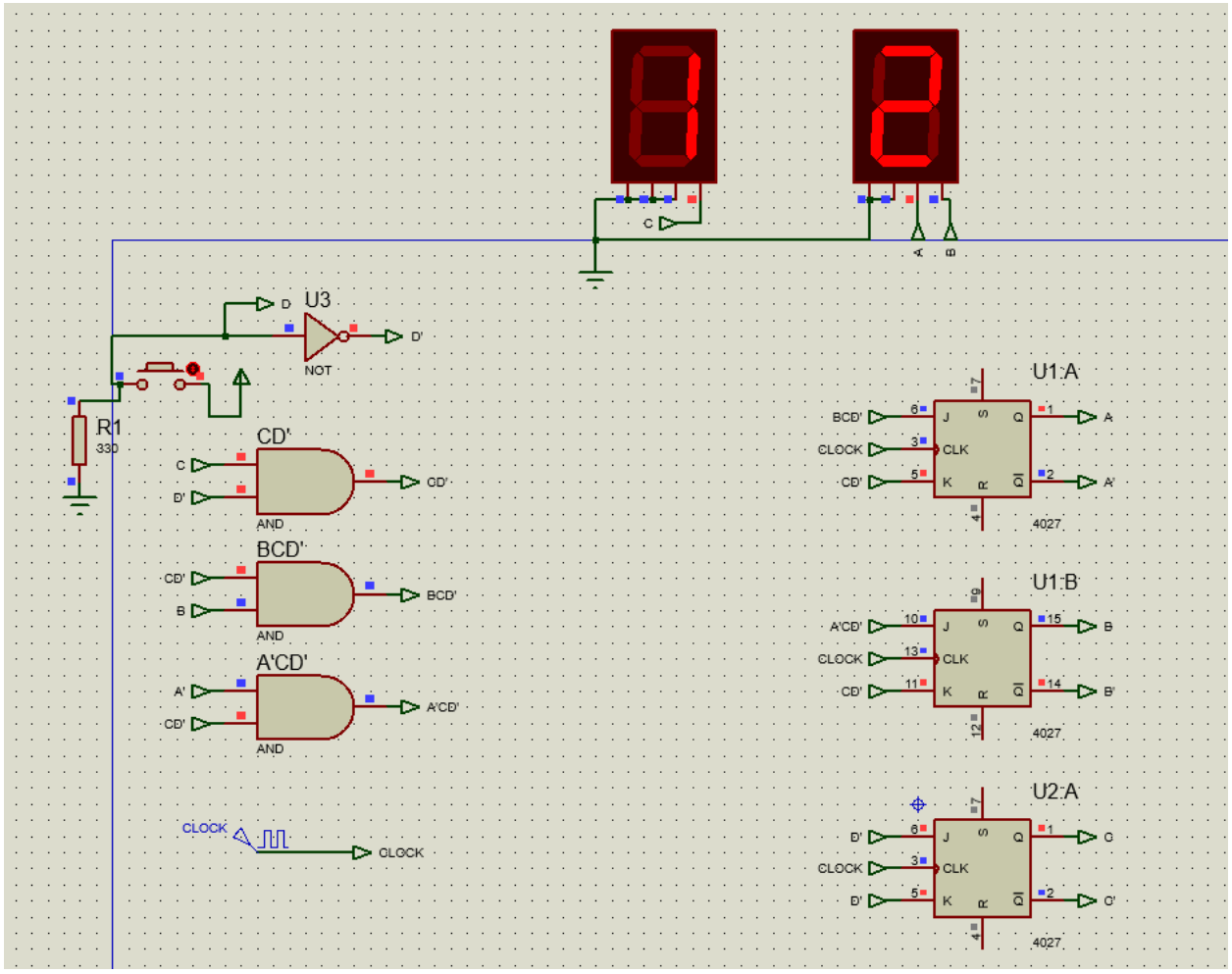
A	B	C	D	A+	B+	C+	X	Y	Z	JA	KA	JB	KB	Sc	Kc
0	0	0	0	0	0	1	0	0	0	0	X	0	X	1	X
0	0	0	1	0	0	0	0	0	0	0	X	0	X	0	X
0	0	1	0	0	1	0	1	0	0	0	X	1	X	X	1
0	0	1	1	0	0	1	1	0	0	0	X	0	X	X	0
0	1	0	0	0	1	1	0	0	1	0	X	X	0	1	X
0	1	0	1	0	1	0	0	0	1	0	X	X	0	0	X
0	1	1	0	1	0	0	1	0	1	1	X	X	1	X	1
0	1	1	1	0	1	1	1	0	1	0	X	X	0	X	0
1	0	0	0	1	0	1	0	1	0	X	0	0	X	1	X
1	0	0	1	1	0	0	0	1	0	X	0	0	X	0	X
1	0	1	0	0	0	0	1	1	0	X	1	0	X	X	1
1	0	1	1	1	0	1	1	1	0	X	0	0	X	X	0
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Para la implementación del contador utilizamos 2 display donde contamos las 6 posiciones que nuestro lector tendrá, el primer display mostrará la posición de la columna, es decir si está en la columna 0 o 1, el segundo display nos mostrará en cuál de las filas estamos situados si en la fila 0, 1 o 2.

La lógica empleada en la máquina de estados finitos es la siguiente, cuando estemos en la posición 0,0, el arduino nos enviará un 1 como señal indicando que nuestro lector no se está moviendo y cuando el lector esté en movimiento el arduino nos enviará un 0 como entrada, para pasar al siguiente estado que sería el 0,1 y cuando ya estemos situados en el estado siguiente el arduino enviará un 1 y es ahí cuando el contador cambia, este procedimiento se realizará para todas las posiciones que tenemos y se repetirá cuantas veces nosotros necesitemos.

Diagramas del Diseño del Circuito

Contador



Código Comentado

Sensores

```
// Definimos los pines de los sensores infrarrojos
const int sensor1Pin = 2;
const int sensor2Pin = 3;
const int sensor3Pin = 4;
const int sensor4Pin = 5;
const int sensorLecturaPin = 6;

// Pin para enviar señal al otro Arduino
const int signalPin = 7;

struct BrailleMapping {
  const char* sequence;
  char asciiChar;
};

// Mapeo de secuencias Braille a caracteres ASCII en minúsculas
BrailleMapping brailleMap[] = {
  {"100000", 'a'},
  {"101000", 'b'},
  {"110000", 'c'},
  {"110100", 'd'},
```

```
{"100100", 'e'},
{"111000", 'f'},
{"111100", 'g'},
{"101100", 'h'},
{"011000", 'i'},
{"011100", 'j'},
{"100010", 'k'},
{"101010", 'l'},
{"110010", 'm'},
{"110110", 'n'},
{"100110", 'o'},
{"111010", 'p'},
{"111110", 'q'},
{"101110", 'r'},
{"011010", 's'},
{"011110", 't'},
{"100011", 'u'},
{"101011", 'v'},
{"011101", 'w'},
{"110011", 'x'},
{"110111", 'y'},
{"100111", 'z'},
{"000000", ' '}, // Espacio
```

```
  {"000001", '.'}, // Punto
  {"000011", ','}, // Coma
  {"000101", ';'}, // Punto y coma
  {"000111", ':'}, // Dos puntos
  {"001001", '?'}, // Signo de interrogación
  {"001011", '!'}, // Signo de exclamación
  {"001101", '('}, // Paréntesis izquierdo
  {"001111", ')'}, // Paréntesis derecho
  {"010000", '1'},
  {"010100", '2'},
  {"010110", '3'},
  {"010010", '4'},
  {"011000", '5'},
  {"011100", '6'},
  {"011010", '7'},
  {"011110", '8'},
  {"100001", '9'},
  {"101001", '0'}
};

char brailleToAscii(String brailleSequence) {
  for (int i = 0; i < sizeof(brailleMap) / sizeof(brailleMap[0]); i++) {
    if (brailleSequence == brailleMap[i].sequence) {
```

```
      return brailleMap[i].asciiChar;
    }
  }
  return '?'; // Carácter por defecto si no se reconoce la secuencia
}

// Convertir un carácter ASCII a su representación en binario
char *charToBinary7Bits(char c) {
  static char bin[8];
  bin[7] = '\0';
  for (int i = 0; i < 7; i++) {
    bin[6 - i] = (c & (1 << i)) ? '1' : '0';
  }
  return bin;
}

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);

  // Configuramos los pines de los sensores como entradas
  pinMode(sensor1Pin, INPUT);
  pinMode(sensor2Pin, INPUT);
```

```
  pinMode(sensor3Pin, INPUT);
  pinMode(sensor4Pin, INPUT);
  pinMode(sensorLecturaPin, INPUT);

  // Configuramos el pin de señal como salida
  pinMode(signalPin, OUTPUT);
  digitalWrite(signalPin, LOW); // Inicialmente, la señal está en bajo
}

void loop() {
  // Leemos el estado de los cuatro sensores
  int sensor1State = digitalRead(sensor1Pin);
  int sensor2State = digitalRead(sensor2Pin);
  int sensor3State = digitalRead(sensor3Pin);
  int sensor4State = digitalRead(sensor4Pin);

  // Verificamos si todos los sensores detectan la tarjeta en posición
  if (sensor1State == HIGH && sensor2State == HIGH && sensor3State == HIGH &&
sensor4State == HIGH) {
    // Si la tarjeta está en posición, leemos el quinto sensor para la secuencia en código Braille
    String brailleSequence = "";
    for (int i = 0; i < 6; i++) {
      int sensorLecturaState = digitalRead(sensorLecturaPin);
```

```
      brailleSequence += String(sensorLecturaState);
      delay(100); // Ajusta el tiempo según sea necesario para leer los datos correctamente
    }

    // Convertir la secuencia a carácter ASCII
    char asciiChar = brailleToAscii(brailleSequence);

    // Enviamos la secuencia binaria al Arduino Mega
    Serial.println( charToBinary7Bits(asciiChar));

    // Enviamos una señal al otro Arduino para mover el motor
    digitalWrite(signalPin, HIGH);
    delay(5000); // Mantenemos la señal alta durante 100 ms
    digitalWrite(signalPin, LOW);
  } else {
    // Si la tarjeta no está en posición, mostramos un mensaje de error
    Serial.println("Tarjeta no en posición");
  }

  // Esperamos un poco antes de la siguiente lectura
  delay(5000);
}
```

Motores

```
#include <AFMotor.h>

// Motores con 200 pasos por revolución (1.8 grados)
AF_Stepper motor1(200, 2); // Motor ahora conectado al puerto #2 (M3 y M4)
AF_Stepper motor2(200, 1); // Motor ahora conectado al puerto #1 (M1 y M2)

// Variables para contar la distancia en micro pasos
long contador_distancia_x = 0;
long contador_distancia_y = 0;

// Distancia máxima para detener los motores
const long distancia_maxima = 130;

// Coordenadas predefinidas para 8 posiciones (en micro pasos)
const long posiciones_x[8] = { 0, 8, 8, 55, 55, 110, 110, 0 };
const long posiciones_y[8] = { 0, 0, 40, 0, 40, 0, 40, 0 };

// Índice de la posición actual
int posicion_actual = 0;

void setup() {
  // Configurar la biblioteca Serial a 9600 bps
```

```
  Serial.begin(9600);
  Serial.println("Stepper test!");

  // Configurar la velocidad de ambos motores a 400 pasos por segundo (50 rpm)
  motor1.setSpeed(400);
  motor2.setSpeed(400);
}

void loop() {
  moverAPosicion(posicion_actual);
  posicion_actual++;
  if (posicion_actual >= 8) {
    posicion_actual = 0; // Reiniciar a la primera posición si se supera la última
  }
  delay(1000); // Retraso de 2 segundos antes de moverse a la siguiente posición
}

void moverAPosicion(int pos) {
  Serial.print("Moviendo a la posición: ");
  Serial.println(pos + 1);

  long distancia_a_mover_x = posiciones_x[pos] - contador_distancia_x;
  long distancia_a_mover_y = posiciones_y[pos] - contador_distancia_y;
```

```
 // Definir direcciones de movimiento
 int direccion_x = distancia_a_mover_x > 0 ? BACKWARD : FORWARD;
 int direccion_y = distancia_a_mover_y > 0 ? FORWARD : BACKWARD;

 // Convertir distancias a valores absolutos
 distancia_a_mover_x = abs(distancia_a_mover_x);
 distancia_a_mover_y = abs(distancia_a_mover_y);

 // Mover ambos motores simultáneamente
 while (distancia_a_mover_x > 0 || distancia_a_mover_y > 0) {
  if (distancia_a_mover_x > 0) {
    motor1.step(1, direccion_x, MICROSTEP);
    distancia_a_mover_x--;
    contador_distancia_x += direccion_x == BACKWARD ? 1 : -1;
  }
  if (distancia_a_mover_y > 0) {
    motor2.step(1, direccion_y, MICROSTEP);
    distancia_a_mover_y--;
    contador_distancia_y += direccion_y == FORWARD ? 1 : -1;
  }
 }
```

```
  Serial.println("Movimiento completado.");
}
```

Equipo Utilizado

- Lectores de CD
- Cinco sensores de color
- Cartón
- Protoboard
- Flip flops JK
- Integrado AND 74LS08
- Integrado NOT 74LS04
- Integrados BCD 74LS47
- Resistencias 220 Ohm
- Protoboard
- Arduinos UNO
- Cable para protoboard
- Jumpers

Diagramas con Explicación

Controles:

- WR (Write Control): Este es el control de escritura. Cuando está en "1", se habilita la escritura de datos en la memoria.
- A, B, C (Memory Addresses): Estos son los selectores de las posiciones de memoria, permitiendo direccionar una de las 8 posiciones de memoria disponibles.
- R (Reset): Este control se usa para resetear todas las celdas de la memoria, es decir, poner todos los datos en "0". Cuando se envía un "1" a esta entrada, todas las celdas de memoria se borran.

Ingreso de datos:

- Los datos ingresados en la memoria son siete bits.
- Estos siete bits se gestionan mediante siete multiplexores (mux).
- Los multiplexores utilizan los selectores de dirección (A, B, C) para determinar en cuál de las 8 posiciones de memoria se deben almacenar los datos.

Escritura de datos:

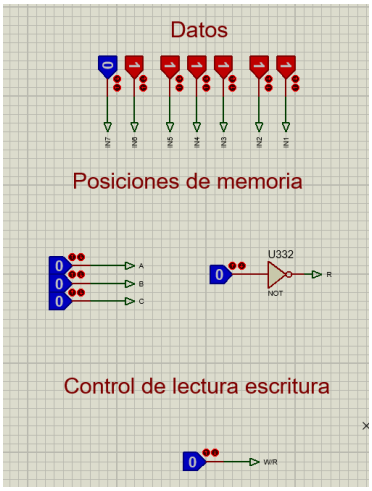
- Cuando el control de escritura (WR) está activo (en "1"), la memoria está lista para recibir y almacenar los datos.
- Los siete bits de datos se almacenan en siete celdas de memoria a través de siete flip-flops tipo D.
- Los flip-flops tipo D se utilizan para almacenar cada bit individualmente y mantener su estado hasta que se cambie explícitamente.
- De esta manera, cada posición de memoria consta de 7 bits, y hay 8 posiciones disponibles en total.

Direccionamiento:

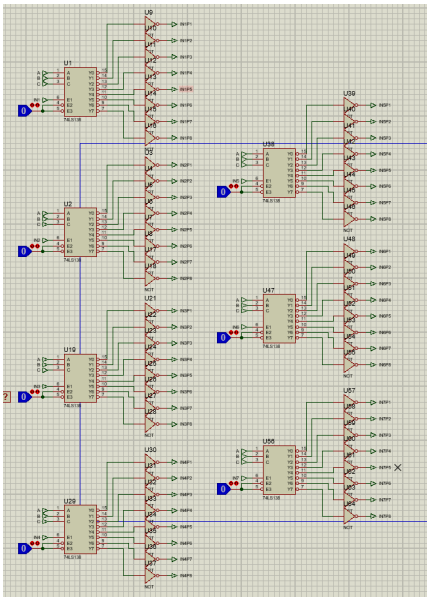
- Los selectores de los multiplexores (A, B, C) permiten seleccionar cuál de las 8 posiciones de memoria se está leyendo o escribiendo en un momento dado.
- Este direccionamiento preciso es crucial para garantizar que los datos se almacenen y recuperen correctamente.

Memoria RAM:

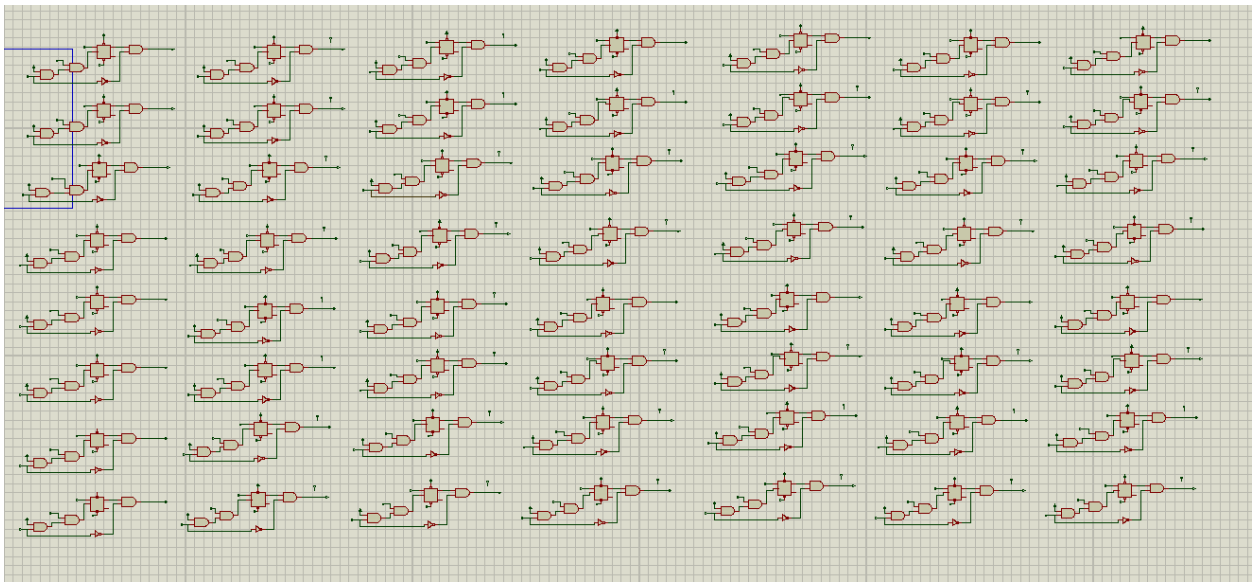
Ingreso:



Multiplexores de direccionamiento:



Celdas de memoria:



Presupuesto

Item	Cantidad	Precio	Total
Tornillos	7	Q4.00	Q28.00
Tuercas	12	Q0.75	Q9.00
Broca	1	Q15.00	Q15.00
Lectores CD	2	Q40.00	Q80.00
Barras de silicón	3	Q1.00	Q3.00
Sensores de color	5	Q16.00	Q80.00
Total			Q215.00

Conclusiones

El desarrollo del prototipo de lector de tarjetas en braille ha demostrado ser una solución efectiva para solucionar el problema presentado por la organización Prociegos y Sordos de Guatemala. La iniciativa busca lograr una verdadera inclusión al permitir que las personas ciegas continúen accediendo a material educativo, a pesar del deterioro físico de los libros con el tiempo.

El prototipo, utilizando tecnología Arduino y sensores de color, ha sido capaz de reconocer con precisión las letras individuales en tarjetas braille. Este sistema sencillo pero eficaz ha logrado leer y procesar las tarjetas, identificando correctamente los caracteres mediante la codificación de los colores (negro para "1" y blanco para "0"). Además, la implementación de una memoria RAM de 7x8 ha permitido almacenar y gestionar los datos de manera eficiente, asegurando que la información leída sea precisa y accesible.

Recomendaciones

1. Utilizar el sensor de color TCS230: El sensor de color TCS230 es conocido por su alta precisión y fiabilidad en la detección de colores. Su integración en el lector de tarjetas braille asegurará una lectura precisa de los caracteres, distinguiendo claramente entre el color negro (representando "1") y el color blanco (representando "0").
2. Fijar bien el sensor de lectura: Asegurar que el sensor de lectura esté firmemente montado en una posición fija y estable. Esto evitará movimientos o vibraciones que puedan afectar la precisión de la lectura de las tarjetas braille.
3. Limpiar y mejorar el código en Arduino: Estructurar el código en Arduino de manera modular, dividiendo las funciones en bloques manejables y reutilizables. Esto no solo mejora la claridad del código sino que también facilita su mantenimiento y actualización.
4. Utilizar un Arduino Mega por la cantidad de pines: El Arduino Mega ofrece un mayor número de pines de entrada y salida (54 pines digitales y 16 pines analógicos), lo cual es esencial para manejar múltiples sensores y otros componentes en el sistema de lectura de tarjetas braille.

Anexos