UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Informe parcial 1

Angie Paola Jaramillo Ortega
Emmanuel Garcés Agudelo

Universidad de Antioquia
Facultad de Ingeniería
Informática II
Septiembre de 2023

1. Análisis del problema y soluciones:

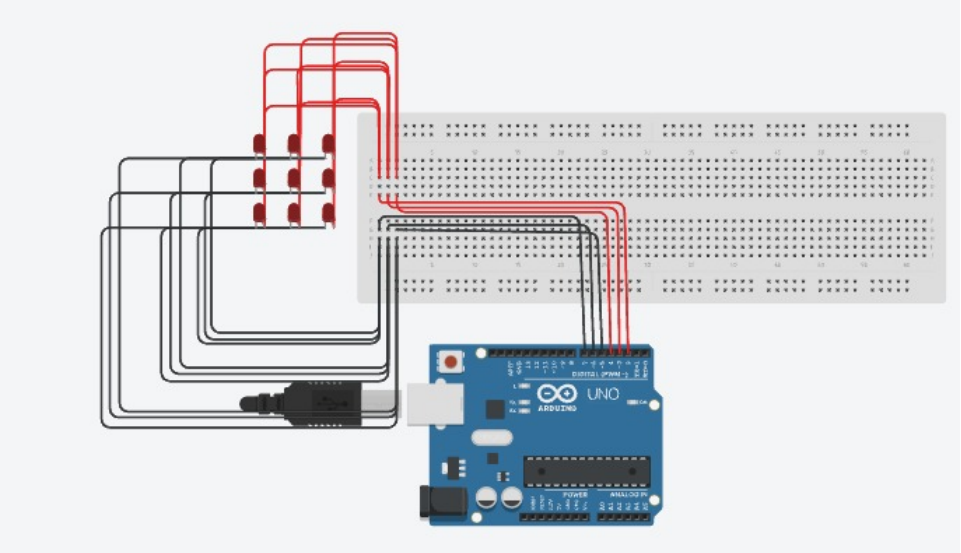


Imagen 1 Matriz de leds 3x3

Primero, se piensa en cómo hacer la matriz de LEDS sin usar más de 7 pines digitales empezando por a mirar con una matriz pequeña cómo se hacen las conexiones correctas, e intentar conectar cada fila a los ánodos y cada columna a los cátodos ayudándonos de la protoboard como se muestra en la Imagen 1, el entendimiento de esta matriz sirvió de base para hacer las conexiones de la matriz 8x8 que se definió. Luego de investigar sobre el 74HC595 , se llega a la conclusión de que al ser un registro de desplazamiento de 8 bits se puede asignar una salida del circuito a cada fila y cada columna usando dos 74HC595, uno para controlar cuál columna se ilumina y otro para controlar qué filas de cada columna se ilumina, así teniendo el control sobre cada led de la matriz.

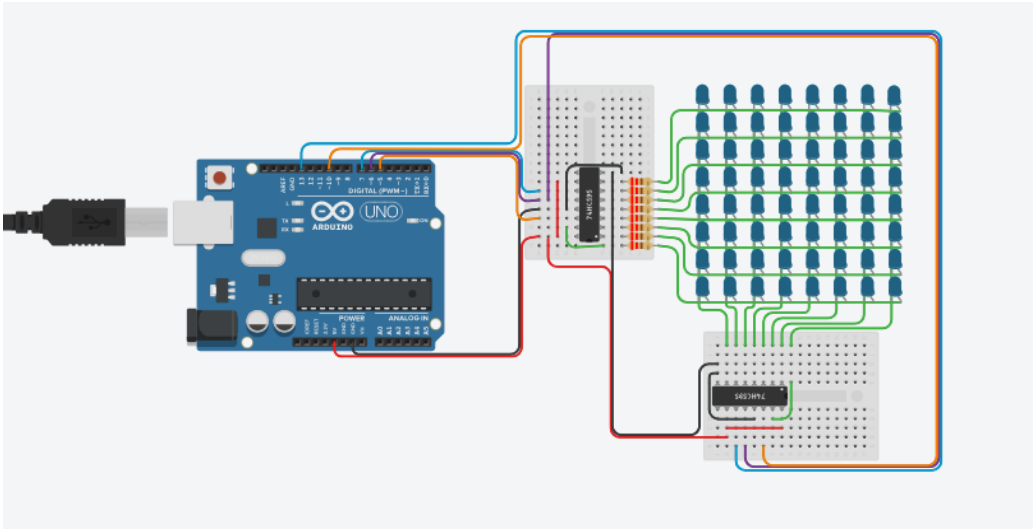
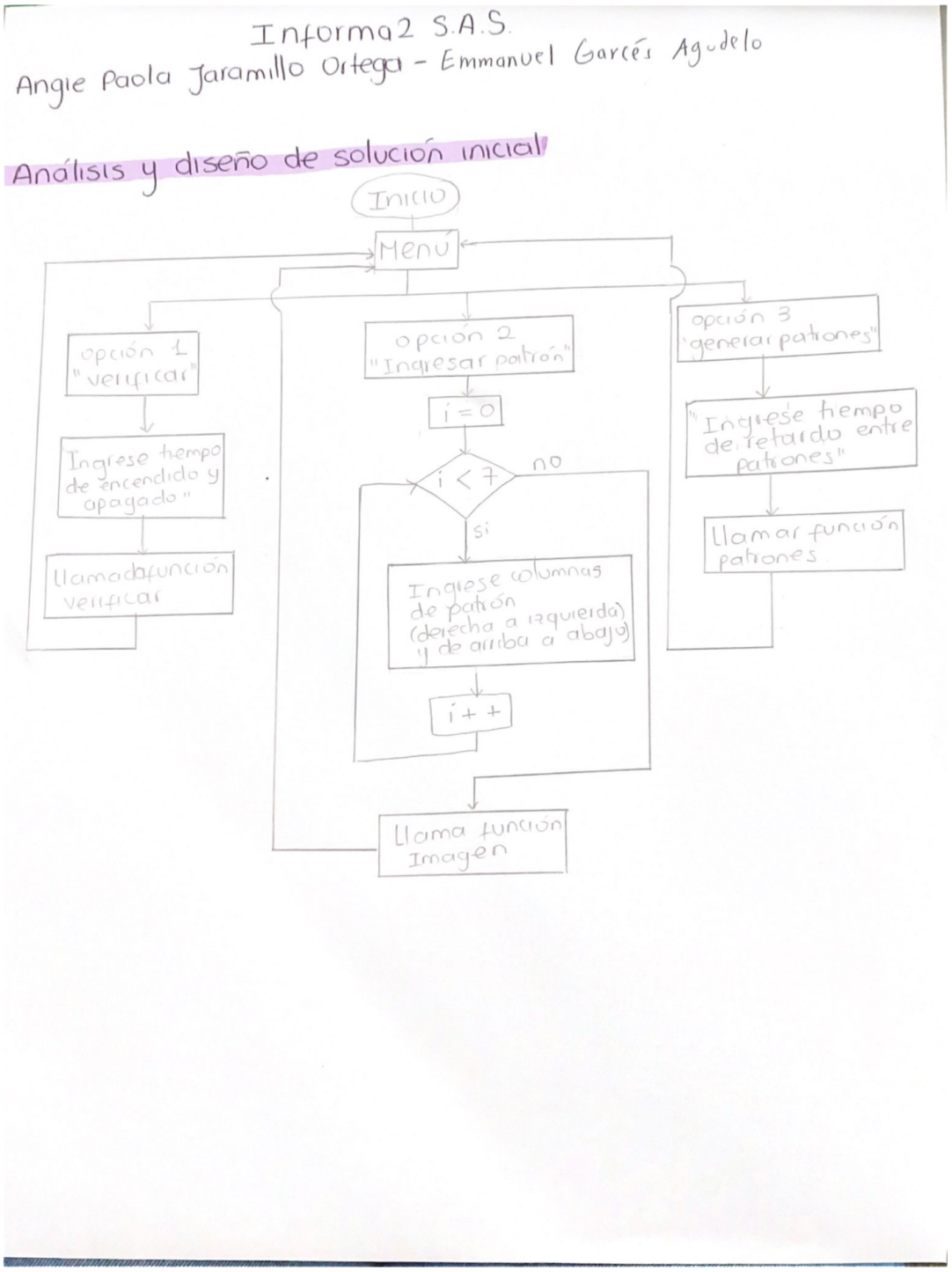


Imagen 2 Matriz de leds 8x8 usando 2 74HC595

2. Esquema de las tareas definidas



3. Algoritmos implementados

Para la solución del problema se realizaron las siguientes funciones:.

- Función WriteY(Control de Filas): esta función es responsable de escribir un byte en el registro de desplazamiento conectado a las filas. Itera a través de los bits y envía los datos a través de digitalWrite y pulse para controlar las filas.
- Función WriteX(Control de Columnas): esta función se encarga de escribir un byte en el registro de desplazamiento conectado a las columnas. Antes de enviar los datos, invierte los bits para ajustarse a la lógica de la matriz de LEDs.
- Verificación: se consideró la utilización de las funciones writeX y writeY para activar todas las filas y columnas y posteriormente desactivarlas. Este proceso se repite varias veces para verificar todos los LEDs, tal como se indica en el enunciado. En este caso, se realizó siete veces (por considerar que siete iteraciones son suficientes para la comprobación). Para lograr esto, se utilizó una variable temporal y un ciclo simple. Dentro de ese ciclo, se implementa otro bucle for para encender todas las filas y columnas, seguido de otro bucle for para apagar todas las filas y columnas. Además, se incluyó un retraso de un segundo entre encendidos y apagados para permitir una visualización adecuada de la activación y desactivación.
- Imagen: crea una matriz bidimensional llamada input que almacena la configuración de la imagen personalizada ingresada por el usuario. Luego, llama a convertToChar(input, Filas) para convertir la matriz bidimensional input en un arreglo unidimensional Filas.

La función convertToChar() procesa cada fila de la matriz input y la convierte en un valor char, donde cada bit en el valor char representa el estado de un LED en esa fila. El valor char resultante se almacena en el arreglo Filas. Después de esta conversión, el bucle principal de Imagen() utiliza el arreglo Filas para encender y apagar los LEDs en la matriz de LEDs controlada por el Arduino de acuerdo con la configuración de la imagen personalizada durante la duración especificada.

- Patrones: Aquí vamos a hacer algoritmos para cada uno de los patrones y usar las funciones writeX y writeY, dependiendo de cada algoritmo, para mostrar filas y columnas deseadas en cada patrón.
    - Patrón 1: Para este patrón, al analizarlo se ve que se repite una secuencia que en cada fila, primero va la mitad de los pines apagados, luego los prendidos y luego la otra mitad de los pines apagados por lo que se crea una variable i=6 que son los pines apagados, se necesita un byte tal que se pueda mover i veces hacia la derecha, luego i/2 veces hacia la izquierda y que en la mitad de estos dos desplazamientos quede en 1 (los prendidos) y así quedaría de la forma en la que se observó la secuencia, esto dentro de un ciclo hasta que i sea 0, y se va restando en 2 porque en cada línea los apagados se disminuyen en 2. El único byte que se puede utilizar para realizar los desplazamientos es el 11111111 (255), al tener todos prendidos, luego de hacer estos 2 desplazamientos, en la mitad siempre quedan los que deben ir prendidos. Ahora, esta lógica solo funciona con la mitad del patrón 1, la otra mitad se hace con la misma lógica, pero al revés. La variable i

(los apagados) empieza en 0 y termina cuando sea igual a 6, esta vez i va incrementando en 2.

Al hacer los desplazamientos del byte 255, en cada iteración queda un byte que representa la fila y se agrega a un arreglo para que luego, dentro de un ciclo, se use las funciones de writeX y writeY con cada elemento del arreglo.

- Patrón 2: Para crear el patrón se utiliza un bucle for para iterar a través de cada fila de la matriz de LEDs, que consta de 8 filas. Para cada fila, se utiliza un carácter de 8 bits (un byte) llamado fila para representar el estado de los LEDs en esa fila.

  El patrón se genera mediante la manipulación de los bits en fila. Se encienden dos LEDs en cada fila a la vez, uno en la posición más a la izquierda y otro en la posición más a la derecha. Luego, el patrón se desplaza hacia la derecha en la próxima iteración del bucle, de modo que los LEDs encendidos anteriormente se apagan y los siguientes en la fila se encienden. Este proceso se repite hasta que se haya barrido toda la fila, creando la ilusión de un movimiento horizontal de encendido de LEDs.

  Una vez que se ha completado un ciclo de barrido en todas las filas, el bucle vuelve a comenzar y se repite indefinidamente hasta que se interrumpa la ejecución del programa.
- Patrón 3: En este patrón, se dividen las filas en dos grupos: aquellas en las que todos los LEDs están encendidos (fila con todos los bits en 1) y aquellas en las que solo algunos LEDs están encendidos (fila con algunos bits en 1 y otros en 0). Esto crea un patrón de fila a fila donde

algunas filas están completamente encendidas y otras tienen solo algunos LEDs encendidos.

El patrón se actualiza en cada iteración del bucle, y se utiliza una variable “desp” para controlar el número de bits encendidos en las filas que tienen LEDs parcialmente encendidos. Los bits encendidos se desplazan hacia la derecha en la fila, creando un efecto de transición gradual de iluminación en las filas.

- Patrón 4: En este patrón se utiliza casi la misma lógica del patrón 1, solo que en este, los apagados van incrementando en 1 por cada fila hasta la mitad del patrón, luego de la mitad van disminuyendo en 1 por cada fila. Para la primera mitad del patrón se utiliza un ciclo que se haga 4 veces (las primeras 4 filas que corresponden a la mitad), y se empieza con el byte que representa la primera fila que es 240. Se usa la variable i para el ciclo y también se usa esta para desplazar las filas, simplemente es que por cada iteración se desplace i veces y esto hace que tome la forma del patrón 1, hasta la mitad. Para la otra mitad es lo mismo solo que la variable i empieza en 3 y se termina cuando i sea -1, esto para que se haga 4 veces pero esta vez al revés.

  Al hacer los desplazamientos del byte 240, en cada iteración queda un byte que representa la fila y se agrega a un arreglo para que luego, dentro de un ciclo, se use las funciones de writeX y writeY con cada elemento del arreglo.

Algoritmo implementado:

```
// Declaracion de variables globales
int LATCH = 6;
int inputy = 5;
int inputx = 10;
int CLKY = 7;
int CLKX = 13;
char opcion = 0;
bool mostrarMenu = true;
unsigned long tiempoInicial;
unsigned long duracionDeseada = 500;
unsigned long duracionImagen;

unsigned char LED[8] = { 1, 2, 4, 8, 16, 32, 64, 128 };

unsigned char *Filas = new unsigned char[8];

// Funcion para enviar un pulso a un pin
void pulse(int pin) {
  // Generar un pulso en el pin especificado
}
```

```
// Funcion para escribir en el eje Y
void writeY(unsigned char byte) {
  // Escribir el valor 'byte' en el eje Y
}

// Funcion para escribir en el eje X
void writeX(unsigned char byte) {
  // Escribir el valor 'byte' en el eje X
}

// Funcion principal para ejecutar una opcion
void publik(char opcion) {
  // Segun la opcion seleccionada, ejecutar la funcion
     correspondiente
}

// Funcion para realizar la verificacion
void verificacion() {
  // Realizar la verificacion de la matriz de LEDs
}

// Funcion para configurar los pines y el puerto serial en el setup
void setup() {
  // Configurar los pines y el puerto serial
}
```

```
// Funcion principal del loop
void loop() {
  // Mostrar el menu si se debe
  // Leer la opcion seleccionada por el usuario
  // Ejecutar la funcion correspondiente a la opcion seleccionada
}

// Funcion para ejecutar el patron 1
void patron1() {
  // Ejecutar el patron 1


}

// Funcion para ejecutar el patron 2
void patron2() {
  // Ejecutar el patron 2
}

// Funcion para ejecutar el patron 3
void patron3() {
  // Ejecutar el patron 3
}

// Funcion para ejecutar el patron 4
void patron4() {
  // Ejecutar el patron 4
}


// Funcion para mostrar una imagen personalizada
void Imagen() {
  // Solicitar al usuario la duracion de la imagen
  // Leer la duracion deseada
  // Solicitar al usuario las columnas de la imagen
  // Leer las columnas de la imagen
  // Mostrar la imagen en la matriz de LEDs durante la duracion
      especificada
}

// Funcion para convertir una matriz de bits en un arreglo de
    caracteres
void convertToChar(unsigned short int** input, unsigned char *Filas
    ) {
  // Convertir la matriz de bits en un arreglo de caracteres
}
```

D. Problemas de desarrollo y solución

- **Problema 1:** Un primer problema surgió cuando se intentó activar todas las filas y todas las columnas simultáneamente en la función de verificación. El intento de encender todas las filas y columnas al mismo tiempo utilizando

writeX(255) y writeY(255) resultaba en la quema del chip 74HC595. Esto se debió a que encender todas las filas y columnas simultáneamente requería una corriente que excedía la capacidad de manejo del 74HC595. Para resolver este problema, se implementó una solución que implica representar las 8 columnas en forma de bytes en un arreglo y utilizar un ciclo para que writeX tome cada byte del arreglo y encienda cada una de las columnas utilizando writeY(255). Este ciclo permite que el usuario vea los 64 LEDs encendidos al mismo tiempo, pero de una manera controlada y sin sobrecargar el chip 74HC595.

- **Problema 2:** En la implementación original, la función Imagen intentaba leer los valores ingresados por el usuario a través del puerto serial en un bucle for anidado que recorre todas las filas y columnas de la matriz. Sin embargo, el problema principal era que el código no esperaba a que se ingresaran los 64 valores antes de continuar. Esto tenía como consecuencia que el usuario no alcanzaba a ingresar todos los valores y, como resultado, la función Imagen avanzaba directamente a imprimir una matriz vacía.

  Para solucionar este problema, se realizó un cambio en la lógica de espera de la función Imagen. En lugar de utilizar if(Serial.available() > 0) para verificar si había datos disponibles en el puerto serial, se cambió a un bucle while (!Serial.available()) {} que bloquea la ejecución hasta que haya datos disponibles en el puerto serial. Esto asegura que la función espere activamente a que el usuario ingrese cada valor antes de continuar al siguiente, garantizando que se ingresen los 64 valores antes de continuar con la conversión y visualización de la imagen.

- **Problema 3:** En la función convertToChar(), se intentó convertir cada fila en el arreglo input de una representación binaria en un valor decimal utilizando la

función pow(2, potencia). La variable potencia comenzaba en 7 y disminuía en 1 a medida que se recorrían los elementos de la fila, acumulando el valor decimal de cada bit en valorChar. Sin embargo, había un problema en la acumulación de valores en valorChar, lo que resultaba en sumas incorrectas de los elementos.

El problema de la acumulación se debía probablemente a la función pow(). Para resolver este problema, se optó por utilizar el desplazamiento de 1 bit a la izquierda (<< 1) para mover los bits en la variable valorChar un lugar hacia la izquierda en cada iteración del bucle de columnas.

El razonamiento detrás del desplazamiento de bits fue el siguiente:

1. Inicialmente, valorChar se establece en 0, lo que significa que todos los bits en valorChar son 0.
2. A medida que recorremos las columnas de la fila en input, queremos agregar los bits uno por uno a valorChar.
3. Usamos el operador << 1 para desplazar todos los bits en valorChar un lugar hacia la izquierda en cada iteración. Esto crea un espacio vacío en el lado derecho del valor valorChar.
4. Luego, utilizamos la operación OR (|) para combinar el bit actual en *ptr con valorChar. Si el bit en *ptr es 1, se establece el bit más a la derecha en charCol en 1; si es 0, el bit se mantiene en 0.
5. Repetimos este proceso para cada bit en la fila, lo que nos permite acumular los bits en valorChar y construir el valor final de valorChar que representa la fila en formato de tipo short int.

Esta corrección garantiza una conversión binaria precisa sin problemas de redondeo y soluciona el problema de la acumulación incorrecta en valorChar.

- **Problema 4:**

  El problema se manifiesta después de seleccionar y ejecutar la opción 3 en el programa, donde la opción 2 deja de funcionar como se esperaba. Aunque la opción 3 en sí misma opera correctamente, después de su ejecución, el sistema parece afectar negativamente la ejecución de la opción 2, lo que resulta en un comportamiento inesperado.

  Después de realizar un análisis del código, confirmamos que todas las variables, incluida la matriz Filas, estaban declaradas adecuadamente a nivel global. Sin embargo, para abordar este problema, decidimos cambiar el ámbito de la matriz Filas declarando la matriz dentro de cada función que la necesita. Después de eliminar la matriz Filas como variable global, evidenciamos el correcto funcionamiento de todas las funciones, lo que nos lleva a atribuir el comportamiento anterior a que la liberación del espacio de la matriz Filas cuando era variable global después de la opción 3 afectaba al funcionamiento de la opción 2. Esta modificación en el alcance de Filas solucionó el problema y permitió que las opciones 2 y 3 funcionaran correctamente sin interferencias mutuas.

E. Consideraciones en la implementación

1. Este programa fue diseñado con la finalidad de permitir a los usuarios ingresar un patrón a mostrar por columnas, en un sentido de derecha a izquierda. Utiliza una representación binaria en la que cada bit corresponde a un LED. En esta representación, el estado "1" indica que el LED está encendido, mientras que el estado "0" indica que está apagado. Como se muestra en el siguiente ejemplo:

   Si el usuario desea ingresar el siguiente patrón:

   ```
   00001000
   00011100
   00101010
   01001001
   00001000
   00001000
   00001000
   00001000
   ```

   Debe ingresar las columnas de la siguiente manera:

   ```
   00010000
   00100000
   01000000
   11111111
   01000000
   00100000
   00010000
   00000000
   ```

2. Una consideración crucial al utilizar este programa se relaciona con la precisión de la medición del tiempo en Tinkercad. Aunque el programa permite a los usuarios especificar la duración de algunas funciones, es esencial tener en cuenta que los segundos en Tinkercad pueden no ser precisos en términos de tiempo real. Esto implica que, al calcular el tiempo de ejecución, el programa podría ser más lento de lo anticipado debido a las limitaciones de esta plataforma.