Informe Desarrollo Parcial I

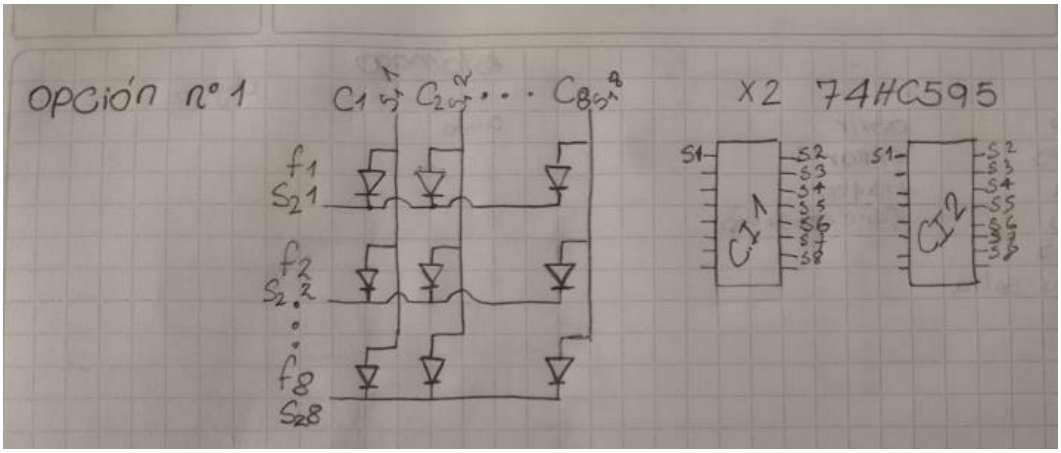
Juan Diego Carrera Quintero
Manuel Esteban Orjuela Montealegre

Universidad De Antioquia
Departamento de Ingeniería y Telecomunicaciones
Informática II
25 de septiembre 2023

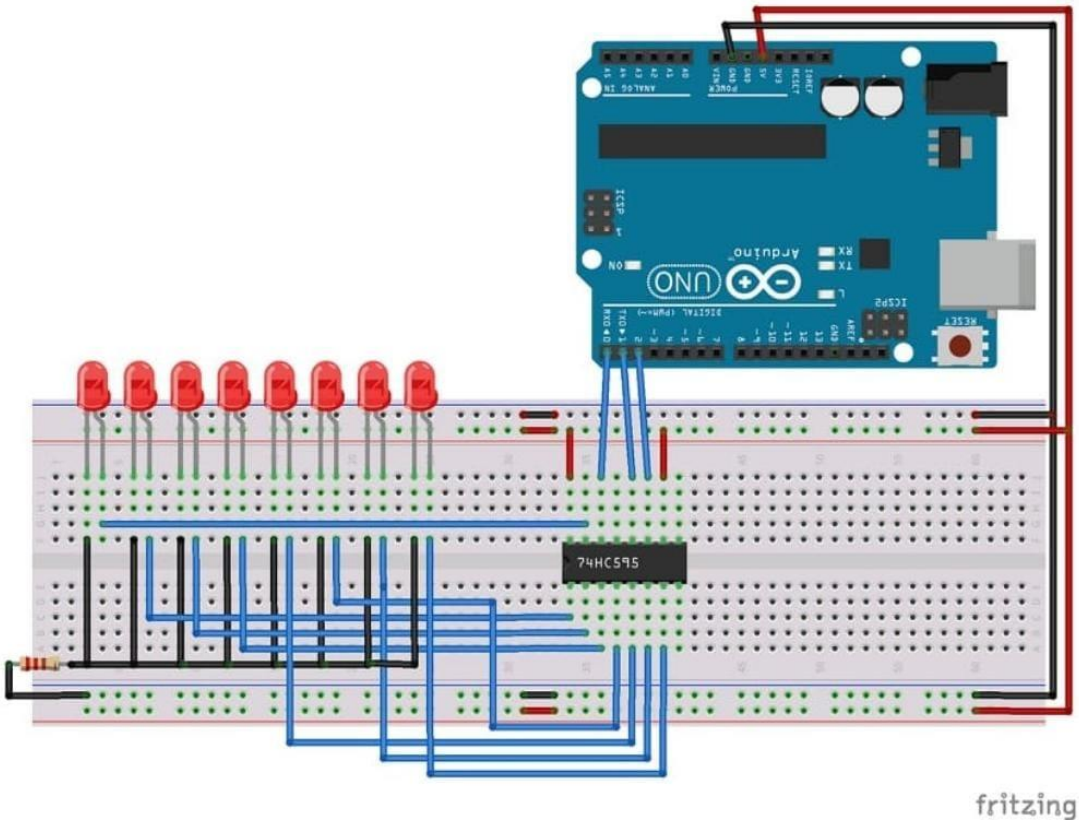
Análisis Del Problema Y Consideraciones Para La Alternativa De Solución Propuesta

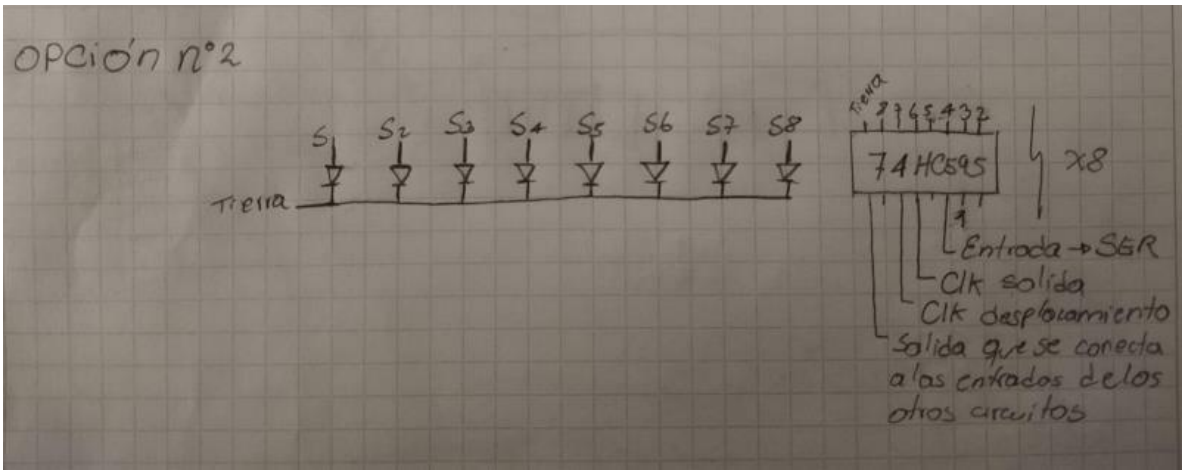
Diseño Circuito.

En un principio teníamos una idea de cómo implementar la matriz de led junto al circuito 74hc595, la idea inicial era conectar en serie los ánodos de 8 leds por fila y los cátodos de 8 leds por columnas, como se muestra en la imagen.



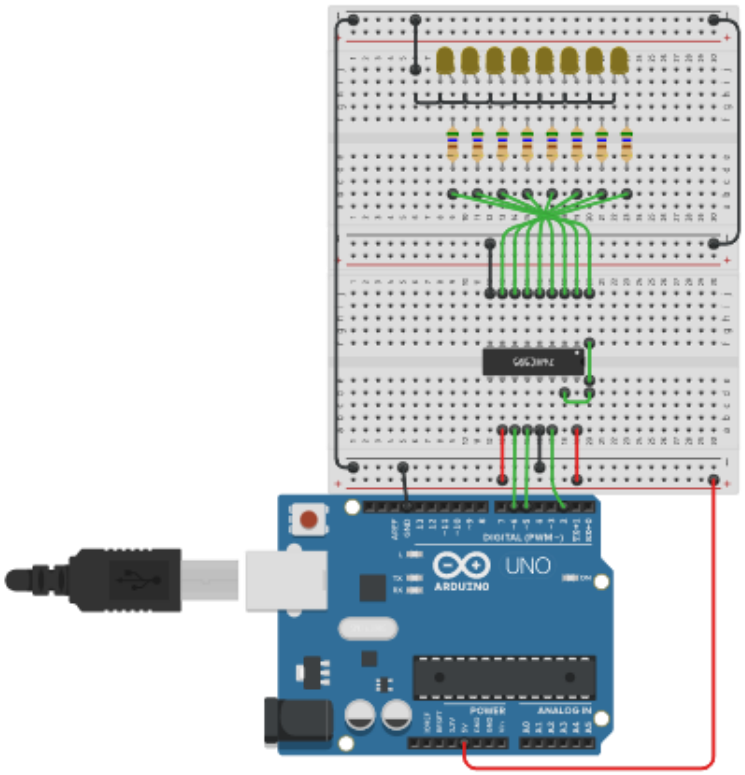
Pero luego de buscar en internet cómo funcionaba el circuito integrado 74HC595, y ver un ejemplo (ver imagen) de cómo usaban el integrado para encender 8 leds de forma secuencial, decidimos hacer lo mismo, pero usando 64 leds y 8 integrados.



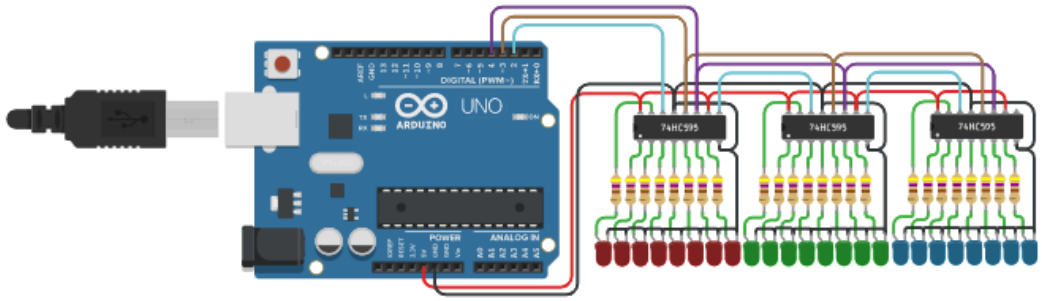


Ahora lo que resta es conectar las 8 filas de leds a cada circuito integrado para posteriormente conectarlos entre sí. La idea base es hacer una fila larga de 64 bits, pero conectarlos en grupos de 8 a cada integrado y finalmente acomodarlos de tal forma que se vea como una matriz 8x8.

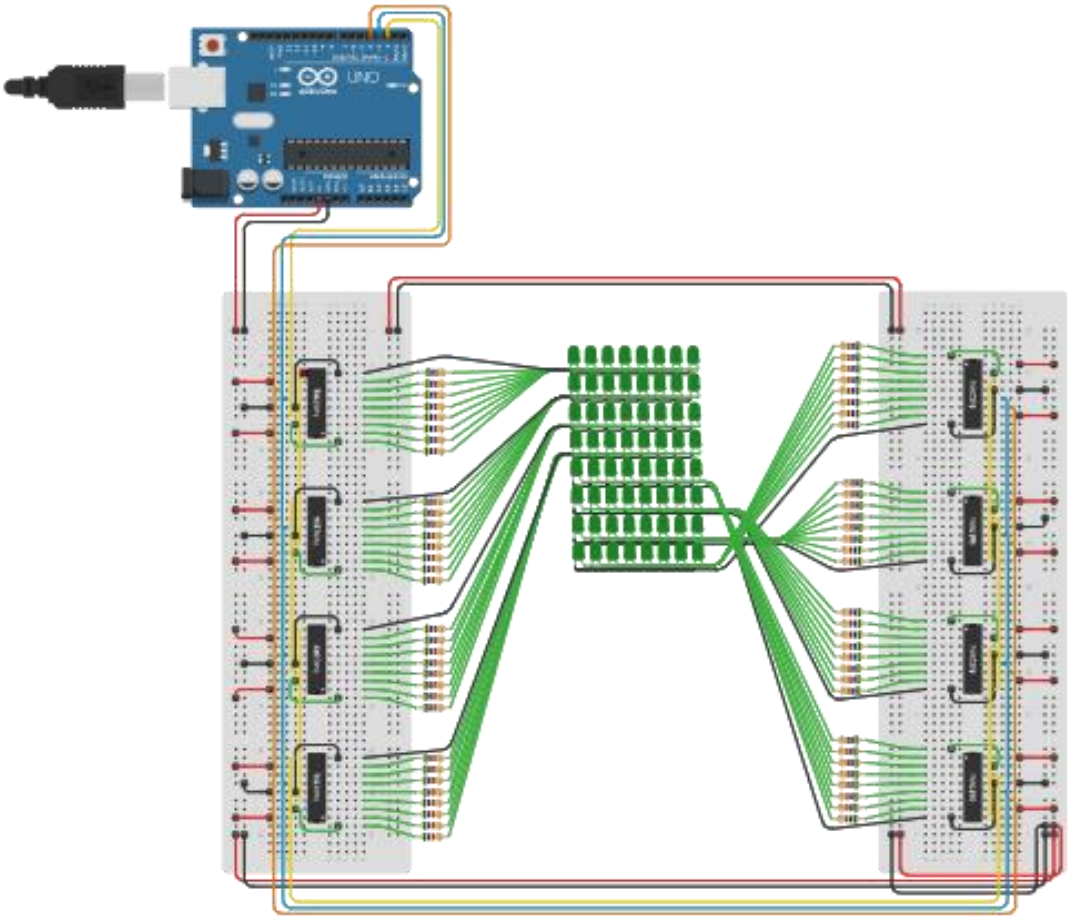
Una vez teniendo la idea del diseño pasamos a replicarla en tinkercad para ver el funcionamiento, por suerte el ejemplo en el cuál nos basamos tenía el código en Arduino del funcionamiento, así que lo copiamos para ver si funcionaba y analizarlo para así después, basándonos en ese código hacer el de la matriz 8x8



Una vez entendido el código y el funcionamiento de este decidimos hacer una prueba más, pero agregando más leds y más circuitos, extendiendo el código para los otros leds y verificar que sí habíamos entendido bien cómo funcionaba y ya poder adentrarnos a la primera parte del problema.



Como esta segunda prueba funcionó, decidimos empezar y montar el circuito con los 64 leds y los 8 integrados, quedando finalmente así:



Con el circuito montado y aparentemente funcionando de manera correcto solo faltaba implementar las funciones verificación, imagen y publik.

Problema – Especificaciones.

Idea Principal: Basado en la construcción de la matriz led 8x8, hay que realizar una implementación de código para la simulación de una pantalla LED.

Objetivo

. Generar patrones de visualización para la pantalla de manera general (Asignados por el parcial) y personalizados junto con pruebas de funcionamiento.

Consideraciones que dificultad el desarrollo del problema

. Escribir el código de manera inmediata en TINKERCARD, para nosotros resulta poco amigable debido a su sintaxis.

Segundo anexo: Tomamos la decisión de primero realizar todo en la plataforma QT, algoritmos de figuras, función Publik, función imagen y verificación.

Tercera Idea: Utilizar una matriz booleana, que retorne valores 0 y 1 para así programar el encendido y apagado de la matriz led.

Cuarta Idea: Usar el mismo ingreso binario para los demás puntos donde se personaliza la visualización, así se facilita el trabajo.

Quinta Idea: Para la función Publik, y todo lo relacionado a encendido y apagado por periodos, dependen del ingreso del usuario junto la alternancia de patrones que iremos trabajando como se mencionó anteriormente.

Conclusión: En conclusión, para avanzar en este proyecto de simulación de una pantalla LED 8x8, comenzar el desarrollo en QT es más recomendable, centrándonos en algoritmos de figuras y funciones de visualización. La utilización de una matriz booleana para el encendido y apagado simplificará la programación, manteniendo un flujo de ingreso binario coherente para la personalización de patrones y la interacción con el usuario. En resumen, este enfoque estructurado nos permitirá lograr una simulación de pantalla LED efectiva y versátil para el conocimiento que hemos desarrollado durante el semestre.

Esquema De Las Tareas Definidas Para El Desarrollo De Algoritmos

Para el desarrollo del programa se definieron las siguientes funciones de la forma.

```
void DefinedFunctions(bool**, int);
```

Función Publik. Unificación de las funciones previamente hechas (funcionamiento total del programa, sirve como menú).

Función Verificación. Encendido y apagado de los leds.

Función Imagen. Pide al usuario que ingrese un patrón personalizado.

Función Squence. Muestra la secuencia de las figuras 1 – 4 un numero de N veces con tiempos de encendido y apagado.

Función Figura 1. Construye el patrón 1 del documento.

Función Figura 2. Construye el patrón 2 del documento.

Función Figura 3. Construye el patrón 3 del documento.

Función Figura 4. Construye el patrón 4 del documento.

Función PrintMatriz. Imprime la representación matricial de los leds, es utilizada en cada una de funciones anteriormente definidas.

Desarrollo de Algoritmos

Punto 1. Función Verificación.

El algoritmo implementado en C++ simula la verificación del encendido y apagado de la matriz led en un ARDUINO.

Desarrollo.

```
        for (int i=0; i<size; i++) {
            for (int j=0; j<size; j++) {
                matrix[i][j] = true;
            }
        }
```

Se asigna a todos los elementos de la matriz booleana el valor de *true* que representaría la entrada binaria 1, es decir, encendido. Para hacer esto, se utilizan dos ciclos, uno que recorra las filas de la matriz y otro que se sitúa en cada una de las columnas.

```
    for (int i=0; i<size; i++) {
        for (int j=0; j<size; j++) {
            matrix[i][j] = false;
        }
    }
```

Con esta última iteración, convertimos todos los elementos del arreglo matricial booleano en false con el valor 0, simbolizando el apagado de los leds.

Punto 2. Función Imagen.

La solución propuesta muestra una imagen de prueba personalizada ingresada por el usuario.

Desarrollo.

```
    for(int i=0; i<size; i++) {
        cout << "Fila No " << i+1 << ": ";
        cin >> fila;
        for (int j=0; j<size; j++) {
            aux_led = fila%10;
            fila /= 10;
            if (aux_led) matriz[i][size-1-j] = true;
            else matriz[i][size-1-j] = false;
        }
    }
}
```

La iteración al inicio del algoritmo busca recorrer las filas la matriz, siendo *size* una variable entera que contenga el tamaño de esta. Ahora bien, lo que se asigna a la variable entera fila son todas las condiciones de una fila de leds (Encendido, apagado – 1,0). Luego, se crea un ciclo que tendrá la siguiente funcionalidad:

j	fila	aux_led
0	10101010	0
1	1010101	1
2	101010	0
3	10101	1
4	1010	0
5	101	1
6	10	0
7	1	1

Si para una fila n se tiene la entrada: 1010101010, podemos ver cómo la variable aux_led toma el último elemento de fila, siendo esta ‘’reducida” de 1 en 1. Luego se utiliza un condicional para evaluar si el contenido aux_led corresponde a 1 o un 0 para así determinar el valor de la matriz booleana.

La entrada en la matriz booleana será la posición donde se ubicaba aux_led en la variable fila original.

Función 3. Patrones.

Algoritmos implementados en C++ para los patrones asignados.

Desarrollo.

Todas las figuras fueron pensadas en una función vacía de la siguiente forma.

```
void FigureN(bool **matriz, int dim)
```

El parámetro bool** hacen referencia a un arreglo doble que corresponde a la matriz y la variable entera representa el tamaño o dimensión de esta.

Figura 1:

```
    int mid = dim/2;
    for(int i=0; i<dim; i++) {
        for(int j=0; j<dim; j++) {
            if (j<=(mid-1)) {
                if (i<=(mid-1) && j>=(mid-1)-i) matriz[i][j] = true;
                else if(i>(mid-1) && j>=(i-mid)) matriz[i][j] = true;
                else matriz[i][j] = false;
            }
            else {
                if (i<=(mid-1) && i>=(j-mid)) matriz[i][j] = true;
                else if (i>mid-1 && (i+j)<=(mid*3)-1) matriz[i][j] = true;
                else matriz[i][j] = false;
            }
        }
    }
}
```

1. Calculamos la mitad de la dimensión de la matriz y lo guardamos en la variable entera dim,
2. Luego utilizamos dos bulces for que están anidados para recorrer todas las celdas de la matriz bidimensional. Siendo el ciclo exterior quien recorrer las filas (‘i’) y el interior las columnas (´j´).
3. Dentro del bucle columna (‘j’), se verifica si esta está en la mitad izquierda de la matriz (menor o igual a mid – 1) o en la mitad derecha (mayor que mid -1).
4. Si j se encuentra en la mitad izquierda:
   . Si i está en la mitad inferior de la matriz y j es mayor o igual a la mitad – 1 – i, entonces la celda corresponde a un valor en true.
   . Si i está en la mitad superior de la matriz, y j es mayor o igual a (i – mid), entonces en el espacio será asignado un true.
   . En los demás casos, se rellenará las posiciones matrices [i][j] con un false.
5. Si j está en la mitad derecha de la matriz:
   . Si i está en la mitad inferior de la matriz e i es mayor o igual a (j – mid), entonces se establece la celda correspondiente en la matriz[i][j] en true.
   . Si i está en la mitad superior de la matriz, y la suma de i y j es menor o igual a (mid * 3) -1, entonces se establece la celda con un true.
   . En los demás casos se asigna un false.

Figura 2:

```
void Figure2(bool **matriz, int dim) {
    for(int i=0; i<dim; i++) {
        for(int j=0; j<dim; j++) {
            if (i==j || (i+j) == (dim-1)) matriz[i][j] = true;
            else matriz[i][j] = false;
        }
    }
}
```

1. **Calculamos** la mitad de la dimensión de la matriz y lo guardamos en la variable entera dim,
2. Luego utilizamos dos bulces for que están anidados para recorrer todas las celdas de la matriz bidimensional. Siendo el ciclo exterior quien recorrer las filas (‘i’) y el interior las columnas (´j´).
3. Planteamos la condición para la parte izquierda la cual determina si i y j son iguales. Para la parte derecha declaramos que si la suma de i + j es igual a la dimensión – 1. Si alguna se cumple se asigna el valor de 1 formando la X.
4. Para los demás casos, se asigna un 0.

Figura 3:

```
void Figure3(bool **matriz, int dim){
    for (int i = 0; i < dim; i++){
        if ((i/2) % 2 == 0){
            int pos = 2;
            for (int k = 0; k < dim; k++){
                if ((i-i)+k == pos){
                    matriz[i][k] = false;
                    pos += 3;
                }else{
                    matriz[i][k] = true;
                }
            }
        }else{
            for (int k = 0; k < dim; k++) {
                if (i - i + k % 3 == 0) {
                    matriz[i][k] = false;
                } else {
                    matriz[i][k] = true;
                }
            }
        }
    }
}
```

1. Dividimos la figura en dos partes, cada una correspondiente al tipo de fila 11011011 y 01101101.
2. Utilizamos la misma lógica de fila – columna para cada caso.
3. Tenemos entonces un condicional que hará que se asigne a dos filas seguidas de la matriz el mismo patrón.
4. Para el caso 1, inicializamos y declaramos la variable pos y recorremos las columnas con k.
5. La idea de estas filas es que las ubicaciones de los 0 se encuentran separadas de 3 en 3, pero inicia en la posición 2. Evaluamos con el condicional si (i-i) + k = 2, por lo que, si es verdad, estaría correspondiendo a la columna del 0, por lo que se asigna un valor false. Incrementamos pos de 3 en 3 y para los casos contrarios agregamos valores true.

6. Para el caso 2, tenemos otro ciclo que recorre las columnas y evalúa si el modulo de la columna es igual a 0, si así es, se ingresa un false en el espacio, para casos contrarios, el valor es true.

Figura 4:

```
void Figure4(bool **matriz, int dim){
    for (int i = 0; i < dim; i++){
        for (int k = 0; k < dim; k++){
            if (i < dim /2){
                if (k - i < 0 or k - i > dim/2 - 1){
                    matriz[i][k] = false;
                }else{
                    matriz[i][k] = true;
                }
            }else{
                if (k+i < dim - 1 or k + i > dim + (dim/2) - 2){
                    matriz[i][k] = false;
                }else
                    matriz[i][k] = true;
            }
        }
    }
}
```

1. Utilizamos dos bulces for que están anidados para recorrer todas las celdas de la matriz bidimensional. Siendo el ciclo exterior quien recorrer las filas (‘i’) y el interior las columnas (´j´).
2. La idea que se tuvo en cuenta en este patrón de la señal fue que la cantidad de 1 por cada fila corresponde a la mitad de la dimensión de la matriz.
3. Se dividió el patrón en dos partes ya que tendrían comportamientos diferentes, uno superior y uno inferior.
4. Para el superior. Nótese que la ubicación de los 1 se va desplazando una columna a la derecha a medida que las filas se van recorriendo, por lo que, la ubicación de estos se encuentra de manera específica. Por eso se establece un rango para posicionarlos, el cual corresponde a que si $k - 1 < 0$ or $k - i > dim/2 - 1$ se asigna un valor false, y los que no cumplan esa condición se ubicarían los números 1, que sería su rango de espaciamiento.
5. Para la parte inferior. Se utiliza la misma lógica, pero ahora se desplaza hacia la izquierda. Se establece la condición para su posición y usando la misma idea del rango, se establece que, si las operaciones entre i y j están fuera de lo establecido, se asigna un false, en el otro caso que se cumpla, un true.

Función Publik.

```
    int opcion;
    cout << "Menu:\nEstas son las funciones que puedes realizar." << endl;
    cout << "1. Ejecutar la verificacion de los leds." << endl;
    cout << "2. Un patron personalizado o imagen de prueba." << endl;
    cout << "3. Mostrar de forma altermada 4 patrones ya predeterminados." << endl;
    cout << "Elija una opcion: ";
    cin >> opcion;
    while (opcion < 1 || opcion > 3) {
        cout << "La entrada " << opcion << " no es valida, intente de nuevo: ";
        cin >> opcion;
    }

    switch (opcion) {

    case 1:
        Verificacion(matriz, dim);
        break;

    case 2:
        Imagen(matriz, dim);
        PrintMatrix(matriz, dim);
        break;

    case 3:
        Sequence(matriz, dim);
        break;
    }
}
```

Utiliza una entrada para verificar la selección del usuario.

Funciones Auxliares.

Función PrintMatrix.

```
void PrintMatrix(bool **matrix, int size){
    for(int i=0; i<size; i++) {
        for(int j=0; j<size; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}
```

Imprime la matriz.

Algoritmo de repetición.

```
int Repe, time;
cout << "Acontinuacion se van a mostrar 4 patrones ya declarados" << endl;
cout << "Ingrese la cantida N veces que quiere que se repita el patron: N=";
cin >> Repe;
cout << "Ingrese el tiempo T que se visualizara cada patron: T= ";
cin >> time;

//Ciclo para repetir los patrones
for (int i=0; i<Repe; i++) {
```

Lo utilizamos para simular las veces que se ejecutará cada función, con time_ 1 y time_2 para conocer el encendido y apagado de leds.

Problemas De Desarrollo Que Se Afrontaron

En un inicio no tuvimos problemas en el diseño de nuestra solución, el código "esqueleto" que usamos para la solución final fue hecho en el lenguaje C++ en la herramienta Qt Creator, el cuál resultó relativamente sencillo de hacer, el único problema que tuvimos (que no fue un problema como tal) fue hacer las funciones que generaran los patrones, ya que se requería ingeniar una solución y no recurrir a la implementación de la función explícita, pero pensándolo bien y haciendo las debidas pruebas se llegó a un resultado bastante óptimo.

Otro problema con el que nos pudimos haber enfrentado fue el montaje del circuito, pero por suerte en internet había suficiente información sobre cómo con un circuito integrado hacer una secuencia de 8 leds, haciendo que se encendieran uno tras otro, con un poquito de ingenio y mente, pudimos utilizar el mismo principio para 64 leds.

Luego de tener el circuito montado en tinkercad llegaba la hora de, antes de implementar la solución hecha previamente en Qt, hacer que se enciendan los leds y ahí fue el primer gran problema, en nuestra solución había una matriz 8x8 que contenía únicamente 1 y 0 haciendo referencia a led encendido y apagado respectivamente, pero la función shiftOut que era la encargada de encender cada fila de leds recibía o un número en binario o un número entero, entonces tras pensarlo un rato nos dimos cuenta que como la matriz contenía 1 y 0 y habían 8 por fila, cada fila se podía ver como un número binario de 8 bits, así que creamos un arreglo auxiliar de dimensión 8 que guardaría el número entero correspondiente a cada fila de led, y una función que convertiría los 8 "bits" de cada fila a un número entero, finalmente ese arreglo era el que entraría a la función que sería la encargada de encender no solo la fila de leds sino toda la matriz en general.

Ya con eso correctamente quedaba implementar la solución y aquí tuvimos bastantes problemas, pero todos eran de semántica o sintaxis, alguna función que recibía algo como parámetro y al momento de ser llamada no le entraba el parámetro correspondiente, o que tenía una variable definida en las entradas, pero en la función había otra que no lo estaba o el típico problema de algún punto y coma o paréntesis faltante.

Finalmente, después de hacer muchas "prueba y error" y arreglar errores tontos dimos con la finalización de la programación del Arduino, hicimos un testeo y funcionaba correctamente.

Lo único malo es que al ser una simulación algunas cosas tardaban bastantes pero un poquito demorado y todo, el ejercicio estaba por fin terminado.

Enlaces:

Video: https://youtu.be/maIJ8j6U1k8

Tinkercad:
https://www.tinkercad.com/things/aUkw2D6RFm8?sharecode=UoCtXq90t935J7Fv99R8Ptsp62HlEB7LW5UFn4gLCd8