**INFORME DEL PARCIAL 1**

Sergio Andrés Alarcón López.

CC:1038866637

Alejandro García Sosa

CC:1001456356

**Análisis problemas**

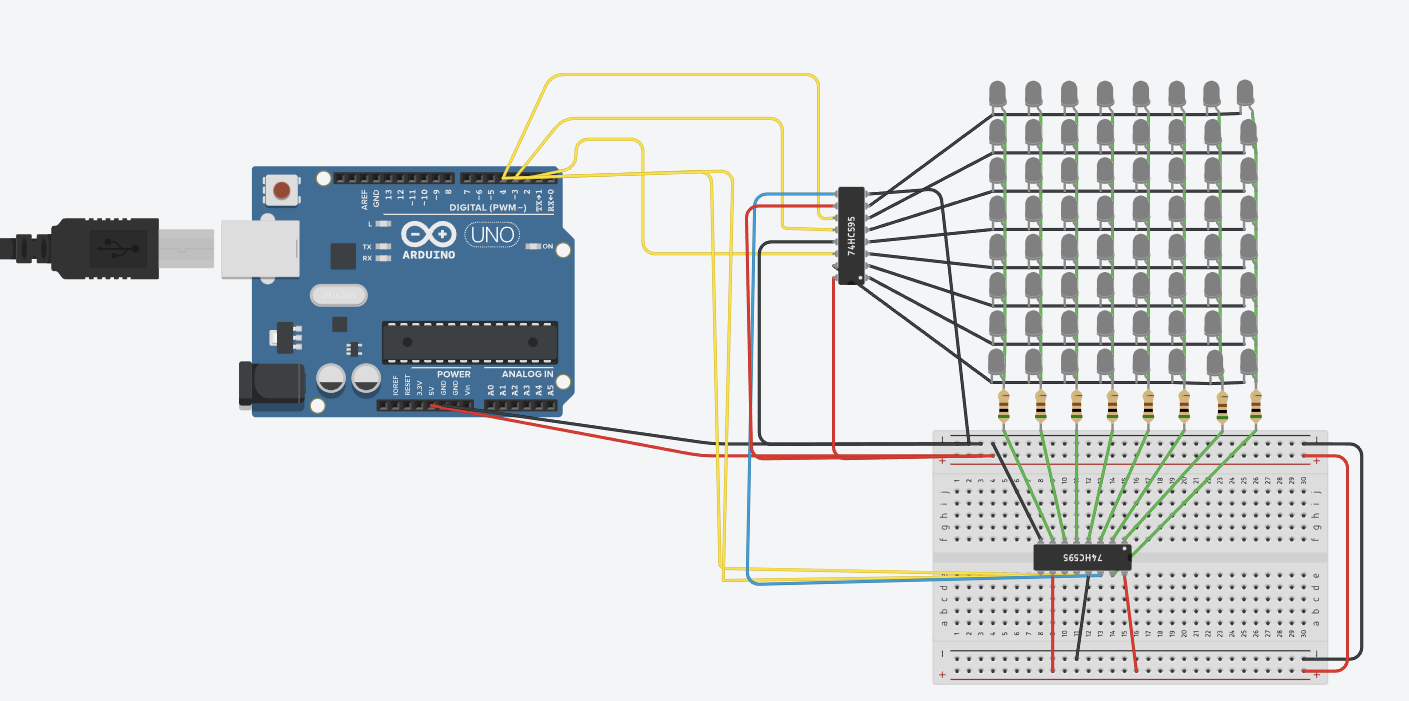
El planteamiento del problema al principio tuvo 2 tipos de análisis diferentes.

El primero tuvo como argumento el desarrollo de matrices dinámicas, utilizando punteros, el cual nos brinda el lenguaje C++ para poder así desarrollar una matriz cuadrada llena de ceros los cuales vamos cambiando según corresponde a un 1 para determinar si un led en esa posición se prende o no, de esta forma solo iteramos sobre la matriz y sobre el arreglo de leds y si encontramos un 1 lo pasamos al led y ese se prende

El otro punto de vista que tuvimos (el cual fue implementado) se desarrolló con el integrado 74HC595, el cual fue propuesto como alternativa para la implementación del parcial. Gracias a la investigación y consulta optamos por elegir dos de estos 74HC595 como base para nuestro trabajo. La implementación fue un tanto compleja debido al poco conocimiento que tenemos con el manejo del hardware, por lo tanto, tuvimos que dedicar mucho tiempo a la realización del circuito y su entendimiento. Nuestra lógica parte de manejar la matriz con las filas y columnas de esta, para eso utilizamos dos integrados 74HC595 los cuales los cargábamos con números binarios, y estos eran representados por la matriz como un encendido (1) o apagado (2). A la hora de realizar los patrones realizamos los siguientes algoritmos:

1. Imprimimos el patrón en dos partes diferentes la primera será para la parte superior entonces iteramos desde cero hasta la mitad del tamaño de la matriz y determinamos los algoritmos para posicionar unos o ceros. Para la segunda mitad vamos desde la mitad hasta el tamaño de la matriz y determinamos los algoritmos para posicionar unos o ceros
2. La nueva versión cambia de usar una matriz a modificar directamente los bytes que permiten encender y apagar tanto las filas como las columnas, entonces empezamos a imprimir desde arriba hasta abajo, empezamos con B00011000 para prender las columnas en las posiciones (3 y 4) y también iniciamos con B00000000 lo que significa que todas las filas van a estar disponibles para ver luz y luego vamos cambiando las columnas cambiando la posición B00011000 por la derecha y por la izquierda hasta llegar a B00100100 y el dato de las filas va de B00000000 a B10000001, de esta manera podemos hacer el patrón
   * 1. Para imprimir el patrón dos vemos que debemos cambiar los valores de la matriz en las posiciones donde (i == j) para obtener la diagonal principal y en las posiciones (i, tamaño matriz – 1 - j) para la diagonal secundaria
     2. Usamos los mismos algoritmos para determinar las posiciones, iterando con un for anidado para las filas y las columnas, para imprimir la diagonal principal empezamos en B10000000 para las columnas y B01111111 para las columnas, luego si estamos en la posición donde i==j prendemos led y modificamos las columnas a B01000000 y las filas a sacamos el complemento B10000000 luego B01000000 y por último el complemento B10111111. Luego para la segunda diagonal empezamos en B10000000 para las columnas y B11111110 para las filas ya que empezamos desde la esquina inferior izquierda, luego si estamos en la posición donde i+j==size-1 modificamos las columnas a B01000000 y las filas sacamos complemento B00000001 luego B00000010 y volvemos a sacar el complemento B11111101.
   1. Solo logramos encontrar una solución a nivel de bytes porque a la hora de encontrar la matriz encontramos problemas para definir cada cuanto realizar los saltos en los patrones. Empezamos a iterar desde 0 hasta 2 porque vamos a imprimir el patrón se alterna cada dos filas, empezamos las columnas en B11011011 y las filas en B00110011 luego modificamos las columnas a B01101101 y las filas a B11001100
3. Dividimos el patrón en dos partes donde imprimimos dos mitades, a la hora de iterar sobre las filas por cada iteración creamos una variable auxiliar que determina el largo del patrón a imprimir, en la primera mitad vemos que cuando (i == j) debemos posicionar un uno, luego si la variable auxiliar es mayor que uno y si las columnas son mayores que las filas imprimimos uno y decrementamos la variable auxiliar Para la segunda mitad, cuando la suma de la columna y la fila actual es igual al tamaño de la matriz menos uno imprimimos uno, luego si la variable auxiliar es mayor que uno y si la columna es mayor que otra variable que es igual a la mitad de la matriz menos uno imprimimos uno y decrementamos la otra variable
4. Empezamos dividiendo el problema en dos partes, para imprimir la mitad de arriba y luego la mitad de abajo, para las columnas empezamos en B11110000 y para las columnas empezamos en B01111111, empezamos a iterar y vamos moviendo las posiciones, la de columnas a B01111000 y las filas a B10111111. Para la segunda mitad empezamos con las columnas en B11110000 y las filas en B11111110 lo que significa que empezamos imprimir desde abajo hasta la mitad, luego modificamos las columnas a B01111000 y las filas a B11111101

**Montaje** **del circuito**



En la imagen se modela el circuito implementado, con sus respectivas conexiones y funcionalidades

**Algoritmos y tareas planteadas**

En este proyecto tuvimos la oportunidad de utilizar nuevas funciones y métodos a la hora del desarrollo de este, entre los cuales encontramos los siguientes.

* Encontramos nuevas palabras con funcionalidades diferentes, entre estas se destaca “shiftOut”. Esta palabra se ve implementada a la hora de enviarle información a los integrados, como parámetros se incluye la entrada del integrado, el reloj de desplazamiento, su forma de la lectura del binario y como parámetro final el numero binario que se quiere cargar
* Tuvimos problemas a la hora de controlar los tiempos para las pausas de los patrones de las matrices, para esto utilizamos la función millis(), esta palabra nos permite contar el tiempo en milisegundos hasta 49 días guardándolos en una variable unsigned long. Su implementación fue un poco compleja, pero sin embargo pudimos solucionar este problema. En su ejecución la inicializábamos antes de hacer cualquier función que implicara contar el tiempo de algo, hacíamos comparaciones con el tiempo que pasaba y el tiempo actual, y así podíamos tener el control de nuestro programa
* Generamos un conocimiento base sobre cómo utilizar más a fondo la implementación de los circuitos y su programación, lo cual es un poco más diferente a lo que llevábamos aprendiendo en otros cursos ya vistos
* Los algoritmos se basaron en determinar posiciones a nivel de bit y realizar desplazamientos según corresponda para encontrar la posición objetivo para que se prendan los leds en ese momento y generar los patrones
* Luego algunos patrones decidimos dividirlos en dos partes para que sean más fáciles de implementar los algoritmos, encendiendo primero una mitad y luego la otra mitad faltante
* También aprendimos la ventaja de trabajar directamente con operaciones nivel bite, ya que manipulamos un byte y moviendo bits en este podemos encontrar patrones eficientes y más fáciles de implementar en cuanto a líneas de código

**Problemas del desarrollo**

* Uno de los principales problemas a la hora de desarrollar el proyecto fue que camino escoger para abordar toda su ejecución, siento que uno de los factores claves a la hora de enfrentarse a este tipo de retos es saber que hacer, como hacerlo, y si está bien hacerlo.
* Otro problema fue la elaboración del circuito, nosotros no tenemos bases muy fuertes de este aspecto, esto nos llevó mucho tiempo a la hora de asimilarlo.
* También tuvimos ciertas molestias con algunas palabras de la sintaxis del lenguaje del Arduino, es un poco diferente la programación orientada a tareas aritméticas, como lo que llevábamos haciendo en cursos anteriores, que la programación a hardware, sin embargo, con la debida dedicación pudimos brindar soluciones a los problemas que se presentaban a la hora de pensar el problema
* Encontramos problemas a la hora de implementar los algoritmos porque estos fueron inicialmente enfocados a trabajar sobre matrices de números enteros y cambiar unos y ceros, pero la matriz de leds implementada era más fácil de manejar con bytes por lo que decidimos cambiar a operaciones a realizar desplazamientos a nivel de bit y así simplificar la lógica porque una vez entendido que hacen los bytes que necesitamos podemos solucionar más fácil los problemas
* Realizar el problema 3 fue muy complicado ya que estábamos iterando sobre cada posición de la matriz para que el usuario decida si ese led se prende o no, entonces almacenábamos esto, pero luego no sabíamos como transformar estos datos a bytes para lograr encender los leds, por lo que decidimos crear más patrones para que el usuario decida que quiere hacer, por ejemplo, creamos un patrón de nubes, otro de cambio, otro de parpadeo, y uno de ajedrez, estas funciones hacen lo siguiente:

1. El primer patrón es uno de tablero de ajedrez que empieza las columnas en B10101010 y las filas en B01010101 y luego las va alternando de la siguiente forma columnas a B01010101 y las filas a B10101010 y de esta manera podemos alternar el patrón
2. Este patrón es de parpadeo que va a empezar en columnas B11111111 y las filas en B00000000 y luego las modificamos a B00000000 las columnas y las filas a B11111111
3. Este patrón es de cambio de posición que va a ir alternando desde la izquierda a la derecha omitiendo mostrar la diagonal secundaria, entonces empezamos desde la esquina superior derecha hasta la esquina inferior izquierda con las columnas en B00000001 y las filas en B11111110 y las modificamos a B00000010 las columnas y las filas a B01111111
4. Este patrón va a ser un patrón de nubes moviéndose de derecha a izquierda, empezamos las columnas en B11001100 y las filas en B00110011 y luego vamos modificando a B01100110 las columnas y las filas a B01100110

**Funciones del programa**

**Verificación()**

Función void sin parámetros que muestra la matriz de leds totalmente encendida durante 2 repeticiones (una vez prendida y otra apagada seria 1 repetición) cada repetición de 1 segundo de duración

**Mostrador()**

Función void que recibe un puntero a una función como argumento, su finalidad es mostrar cualquier patrón de matriz en intervalos de tiempo de encendido y apagado, y también controlar sus repeticiones, las cuales son ingresadas por pantalla por el usuario

**Publik()**

Función void sin parámetros que muestra el menú de interación con el usuario, facilitando el uso del programa

**create\_first\_pattern()**

Función void sin parámetros que se encarga de crear el patrón uno mediante operaciones nivel bit

**create\_second\_pattern()**

Función void sin parámetros que se encarga de crear el patrón dos mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**create\_third\_pattern()**

Función void sin parámetros que se encarga de crear el patrón tres mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**create\_fourth\_pattern()**

Función void sin parámetros que se encarga de crear el patrón cuatro mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**create\_chess\_pattern()**

Función void sin parámetros que se encargar de crear el patrón de ajedrez mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**create\_cloud\_pattern()**

Función void sin parámetros que se encargar de crear el patrón de ajedrez mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**create\_blink\_pattern()**

Función void sin parámetros que se encargar de crear el patrón de ajedrez mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**create\_sweep\_pattern()**

Función void sin parámetros que se encargar de crear el patrón de ajedrez mediante operaciones nivel bit e iteraciones sobre el tamaño de una matriz

**clean\_pattern()**

Función void sin parámetros que se encargar de limpiar los patrones cambiando los bytes a su nivel más simple 0

**delay\_millis()**

Función void sin parámetros que se encarga de mostrar los patrones en orden pidiendo un tiempo de demora entre los patrones, la cantidad de veces que se muestra cada patrón y el tiempo que tiene el patrón en pantalla

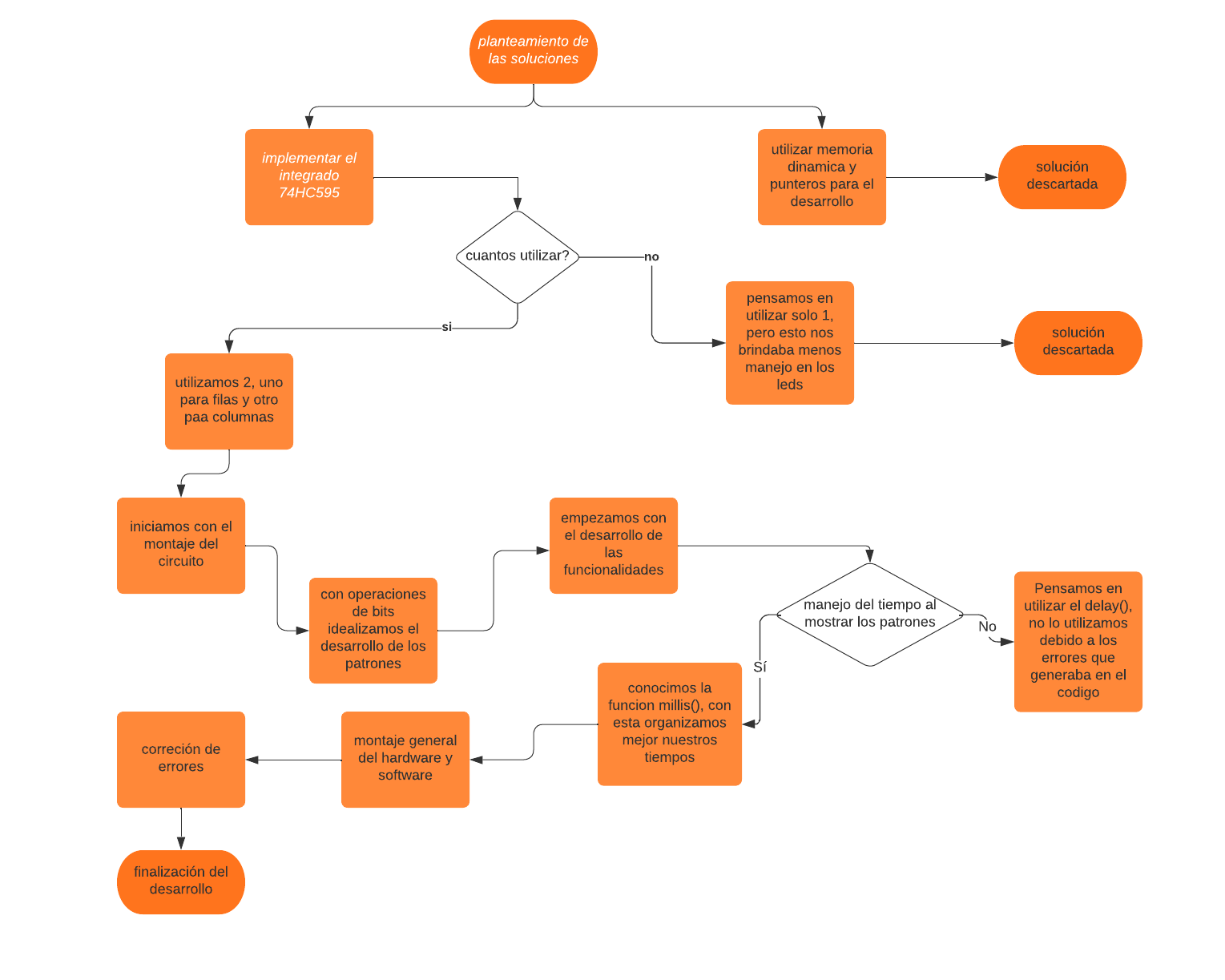
**user\_patterns()**

Función void sin parámetros que se encargar de mostrar un menú para que el usuario pueda decidir que patrones customizados quiere ver, ya sea, nubes, ajedrez, parpadeo o cambio

**local\_pattern()**

Función void sin parámetros que se encargar de mostrar un menú para que el usuario pueda decidir que patrones del programa quiere ver dentro de los cuatro patrones por defecto

**Diagrama del desarrollo**

****