Taller de Proyecto 1

Trabajo Práctico N°4

CAO AGUSTÍN LEONARDO | 1593/9

UNIVERSIDAD
NACIONAL
DE LA PLATA

Indice

Requerimientos

1. Parte 1: Diseño de Firmware, Simulación y Depuración

1.1 Problema

Se desea realizar un prototipo de consola de videojuegos portátil que cuente con al menos una implementación de alguno de los siguientes juegos clásicos: Tetris, Pong, Arkanoid o Conway's Game of Life.

Dado que los distintos juegos poseen diversos comportamientos, y que se estima que la mayor complejidad estará en hacer funcionar la interfaz visual correspondiente a las matrices, el juego a implementar será elegido de entre las opciones ya listadas una vez que dicha interfaz se encuentre en funcionamiento, o quedará a elección de la cátedra antes de arrancar el proyecto.

Para ello, el sistema utilizarán matrices LED de 8x8 en conjunto con controladores MAX7219, los cuales pueden ser operados mediante SPI. Además, contará con pulsadores que permitirán el control del sistema de acuerdo al juego que sea implementado.

Luego de una entrevista con la cátedra, se optó por la implementación del Conway's Game of Life (Juego de la Vida de Conway), con ejemplos predefinidos a elección del desarrollador.

1.2 Interpretación

Es necesario un sistema que permita visualizar un comportamiento correspondiente a las reglas del Juego de la Vida de Conway. El área de juego estará compuesta por una matriz cuadrada de LEDs, de 8x8. Dando un total de 64 células. A su vez, se deben incluir varias disposiciones de células predefinidas a fin de corroborar el funcionamiento.

2. Parte 2: Diseño de Hardware: Circuito impreso PCB

2.1. Problema

El sistema se implementará con el MCU y las herramientas utilizadas en clases, realizando en primer lugar el diseño y la simulación y posteriormente el esquema eléctrico y el PCB para la fabricación del mismo, utilizando reglas de diseño basadas en las vistas en los trabajos prácticos 2 y 3.

En primer lugar, diseñar el esquema eléctrico completo con todos los componentes y sus conexiones. Tener en cuenta que el PCB contendrá los conectores necesarios para conectar la placa BluePill completa, la cual proveerá de alimentación al resto de los circuitos y los conectores para colocar la matriz LED. Luego diseñar el circuito PCB del sistema implementado según los siguientes requerimientos para la fabricación artesanal:

Simple faz de tamaño máximo: 10x10cm
Pads o vías: 1.8 mm de corona, 0.7mm los agujeros
Ancho de pista y separación: mínimo 0.7mm, preferido 1mm.

Realizar en el informe un resumen de los procedimientos realizados para diseñar el PCB y adjuntar imágenes de la capa superior (capa de componentes), capa inferior (capa de soldadura) y vistas 3D del circuito completo.

2.2. Interpretación

Realizar el diseño correspondiente a la placa impresa del circuito desarrollado. La misma debe estar sujeta a ciertos parámetros mencionados en la sección del Problema.

Resolución

Parte 1: Diseño del Firmware, Simulación y Depuración

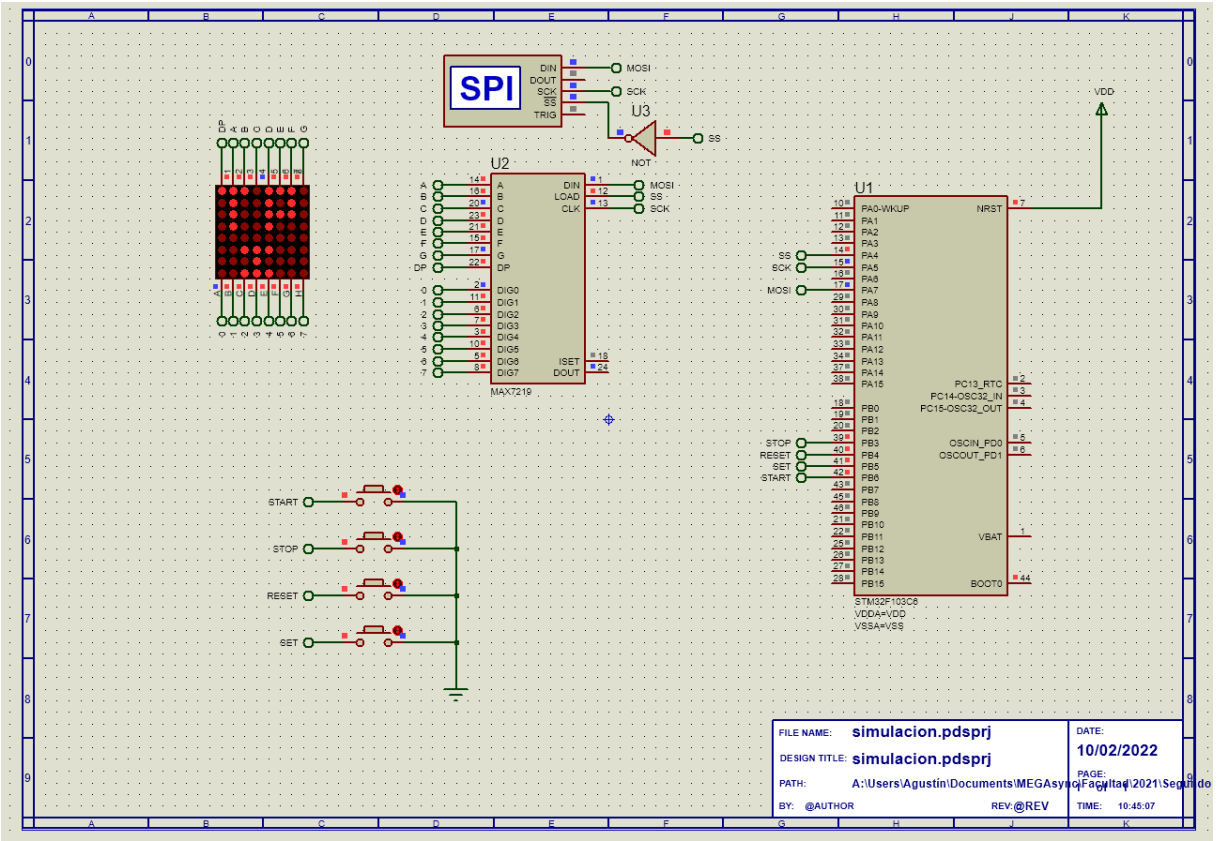

Fig. 1: Esquema de simulación funcionando.

El sistema está controlado por la placa de desarrollo “Blue Pill”, que cuenta con el microcontrolador STM32F103C8T6. La misma comunica mediante SPI la información que debe mostrarse en la matriz de LEDs al controlador MAX7219, encargado del manejo de esta última. A su vez, se utilizan cuatro botones como medio de interacción con el sistema, los cuales están asociados a una tarea especifica cada uno. Por último, la simulación cuenta con un monitor SPI a fin de corroborar los datos transmitidos.

Reglas del Juego de la Vida de Conway

El Juego de la Vida de Conway es un automata celular diseñado por el matemático Jon Horton Conway en el año 1970. Se trata de un juego sin jugadores, lo que significa que su evolución queda determinada por su estado inicial. Uno interactúa con el Juego de la Vida mediante la creación de una configuración inicial y la observación de como este evoluciona. Entre otros datos interesantes, el juego es un sistema Turing Completo, por lo que puede simular una máquina de Turing e incluso crear Constructores Universales de John Von Neumann.

Para la implementación del mismo, cada célula necesita seguir una serie de simples reglas, expresadas en el siguiente pseudo código:

Comprobar el estado de los vecinos.

Si estoy muerta y tengo tres vecinos vivos, paso a estar viva.

Si estoy viva y tengo dos o tres vecinos vivos, sigo viva.

Si tengo más de tres vecinos vivos, muero por sobrepoblación.

Si tengo menos de dos vecinos vivos, muero por soledad.

Este comportamiento es evaluado por cada célula al cumplirse un ciclo entero, y se repite de forma continua hasta que el juego sea detenido.

Administrador de procesos

Para el diseño del sistema se establecieron tres procesos que corren de manera concurrente. Uno controla los botones, otro la matriz de LED y otro es una máquina de estados, diseñada para la lógica del juego.

Para su administración, se diseñó un Sistema Operativo Embebido Simple o SEOS donde el mapeo de los botones se realiza cada 50ms (fig. 2), el refresco de la matriz cada 100ms (fig. 3), y cada ciclo de la máquina de estados se ejecuta cada 200ms (fig.4).

Para controlar la temporización, se utilizó el módulo SysTick propio de los microcontroladores ARM, programado para operar generando interrupciones cada un milisegundo, según una frecuencia de operación del sistema de 8MHz.

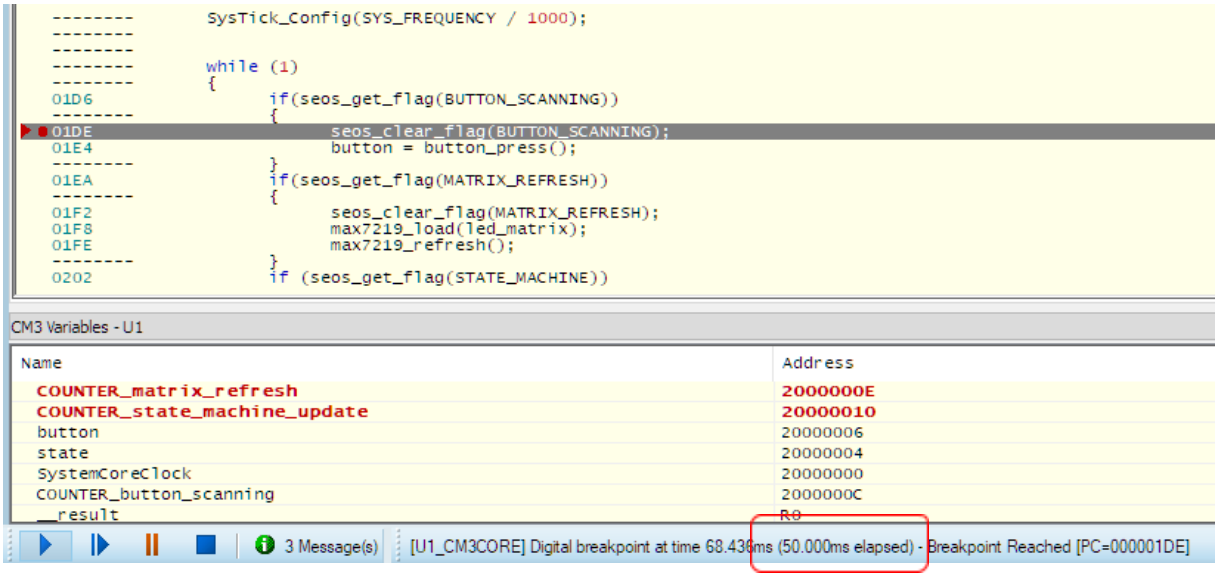

Fig. 2: control de la temporización del escaneo de los botones.

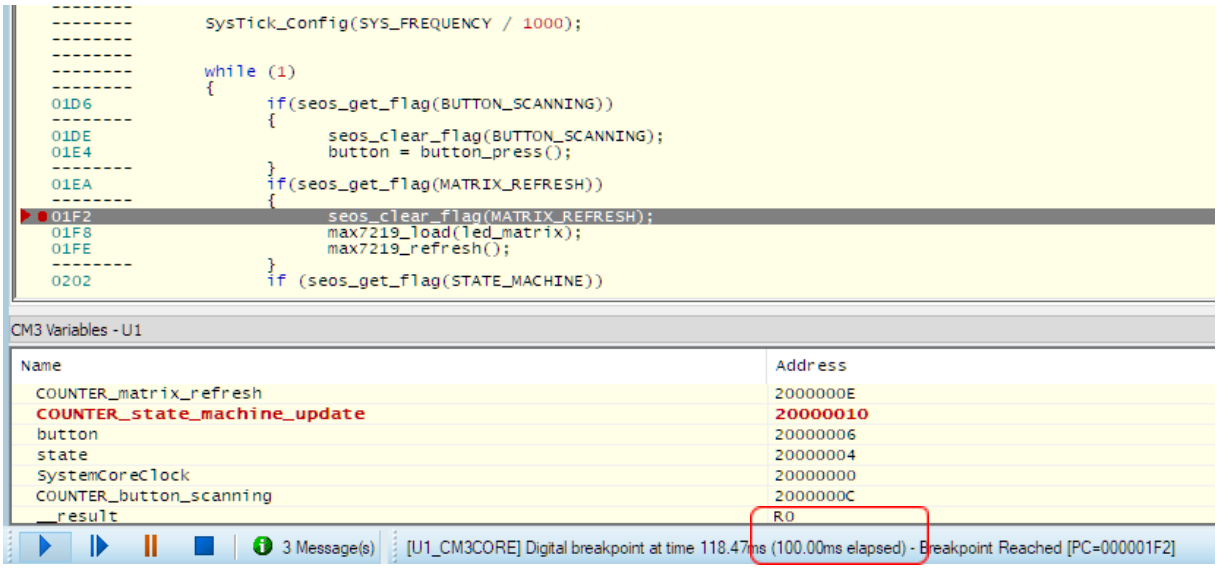

Fig. 3: Control de la temporización del refresco de la matriz LED.

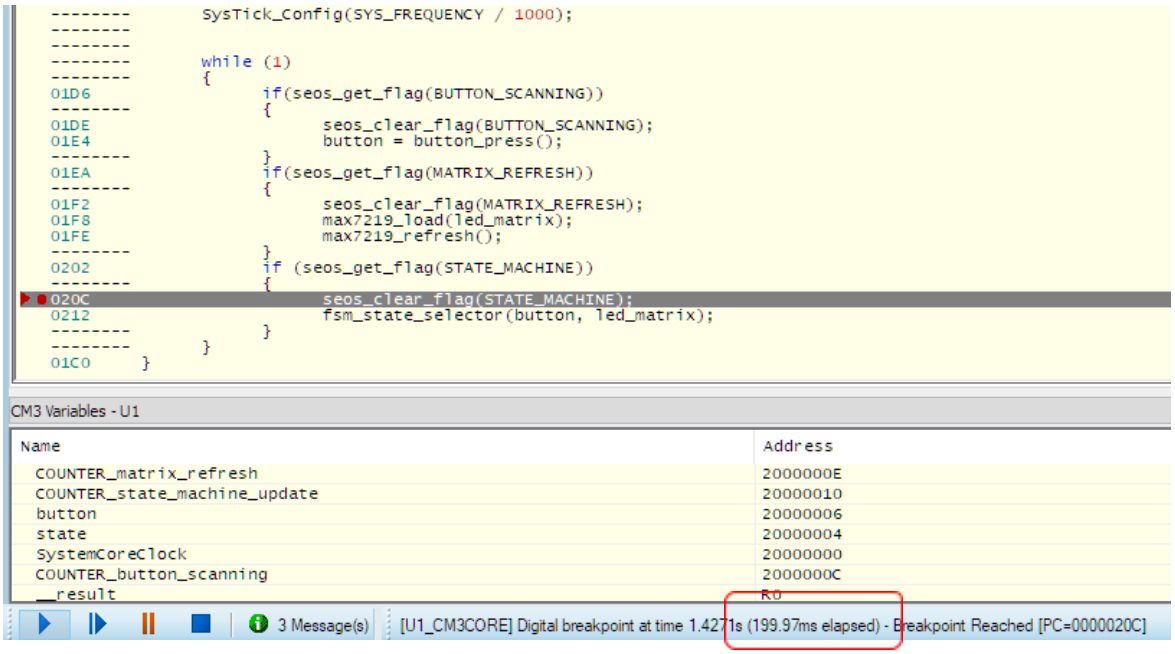

Fig. 4: control de la temporización de los ciclos de la MEF.

Botonera

El control del sistema se realiza mediante cuatro botones (fig. 5), cada uno con una funcionalidad distinta.

- START: Inicia el desarrollo del juego.
- SET: Si el juego se encuentra detenido, cambia entre 8 disposiciones de células distintas.
- RESET: Si el juego se encuentra detenido, devuelve las células a la última disposición que se eligió.
- STOP: Detiene la interacción entre las células.

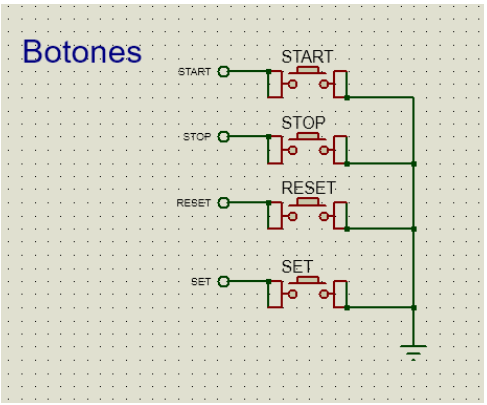

Fig. 5: botones en el esquemático del sistema.

Al realizarse un paneo por los botones, cuya prioridad posee el mismo orden que la lista anterior; en el caso de encontrarse un botón presionado, se guarda en una variable su valor asociado, que luego será interpretado por la MEF. Si ningún botón es presionado, se posee un valor asociado el cual también es interpretado por la MEF en el momento en que sea invocada.

La configuración consiste en conectar los botones a los puertos 3, 4, 5 y 6 (fig. 6) del módulo GPIOB del microcontrolador. Estos están seteados como entradas con resistencias Pull Up. Por lo que se detecta que fueron presionadas si el valor en dicho pin es igual a 0.

```
void buttons_init(void)
{
    RCC->APB2ENR |= (1<<3);
    GPIOB->CRL = 0x48888444;
    GPIOB->ODR |= (1<<3) | (1<<4) | (1<<5) | (1<<6);
}
```

```
uint8_t button_press(void)
{
    if(button_scan(6)) button = START;
    else if(button_scan(5)) button = SET;
    else if(button_scan(4)) button = RESET;
    else if (button_scan(3)) button = STOP;

    return button;
}
```

Fig. 6: Configuración de botones y etiquetas asociadas a ellos.

Matriz LED, MAX7219 y Refresco

Otra parte de la interfaz del sistema es la matriz led, la cual permite visualizar que es lo que ocurre con el juego. El módulo encargado de controlar esta funcionalidad primero obtiene la información producida por la MEF. Luego la prepara de una forma entendible para el módulo MAX7219 (Fig. 7), y por último la transmite al controlador mediante el módulo SPI1, con un baudrate de 31,25KHz.

Frecuencias de transmisión más elevadas fueron probadas sin éxito alguno. Se sospecha que la razón de las fallas se debe al simulador en sí, y no a las configuraciones probadas. Esto se sustenta en el elevado uso de recursos por parte del simulador a la hora de ejecutarse, y la lentitud del mismo, llegando a mostrar décimas de milisegundos en simulación por segundo real. Queda pendiente realizar pruebas de campo que refuten esta teoría.

La configuración de SPI se estableció con la máscara 0x37C (0000 0011 0111 1100), lo cual puede traducirse en:

- Activado el manejo de Slave por Software y configura el Internal Slave Select
- Transmisión con el bit más significativo primero.
- Divisor de frecuencia de baudrate en 256 (7).
- Modo maestro.
- Polaridad de reloj: activo en alto, inactivo en bajo.
- Fase de bits del reloj: Lectura de información en los bordes par de reloj.

Para las demás configuraciones, se activó el módulo SPI1 y los puertos GPIOA. Se inicializaron los pines de MOSI y SCK con la funcionalidad de salida alternativa push pull, y MISO como entrada (aunque no sea utilizada). Como Slave Selector, se optó por usar el pin PA4 como se muestra en las figuras 8 y 9.

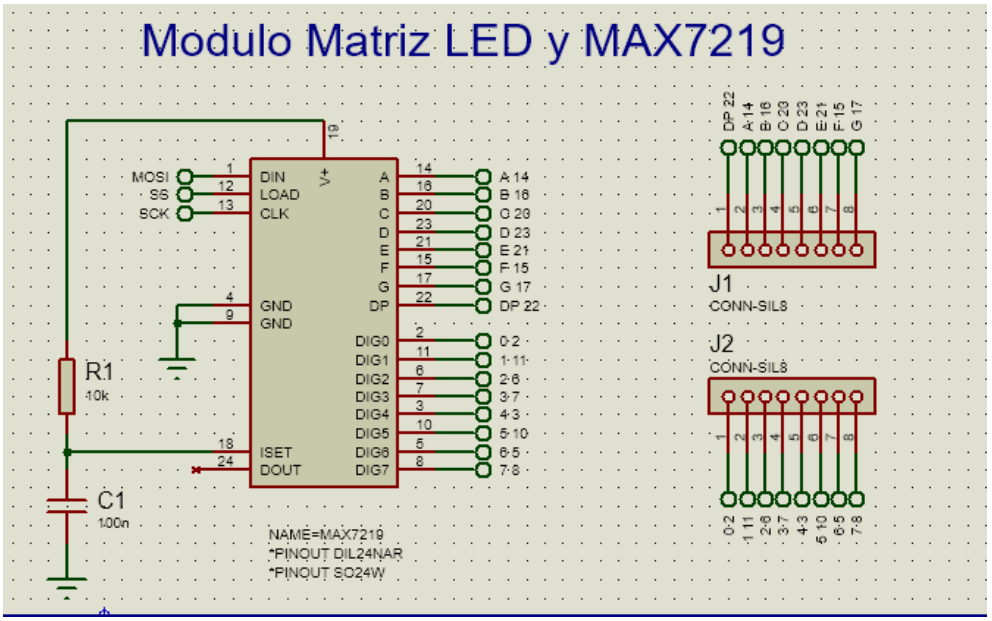

Fig. 7: Módulo Matriz LED y MAX7219

```
void spi_init(void)
{
   RCC->APB2ENR |= (1<<2)|(1<<12);
   GPIOA->CRL = 0xB4B34444;
   SPI1->CR1 = 0x37C;
}
```

Fig. 8: Configuración módulo SPI.

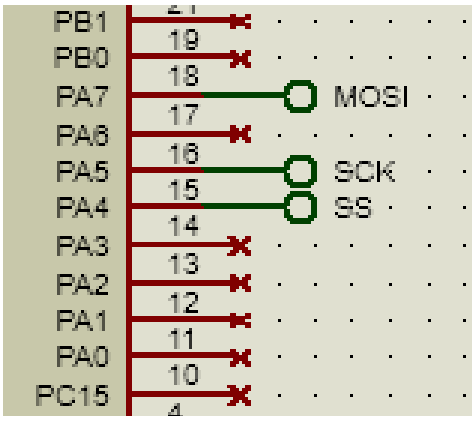

Fig 9: Pines conectados al microcontrolador

El módulo MAX7219 simplemente recibe una palabra (16 bits), de los cuales los primeros 8 bits representan la operación a realizar y los últimos 8 bits un valor asociado a dicha operación.
La tabla 1 muestra el mapeo del registro de direcciones con las cuales uno puede interactuar con el sistema.

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

Tabla 1: Registro de direcciones

El mismo, fue seteado en modo sin decodificación, brillo de los leds controlado por PWM con un ciclo de trabajo de 17/32 partes. Escaneo de líneas de a 8 leds por ciclo, modo de operación normal y modo test desactivado.

Máquina de Estados Finitos

Para el control de menús del sistema y la organización de tareas, se optó por diseñar una máquina de estados finitos. La figura 10 muestra el diagrama resultante de dicho diseño, mientras que la tabla 2 muestra con mayor detalle cuál es la tarea de cada estado, y como ocurren las transiciones entre ellos.

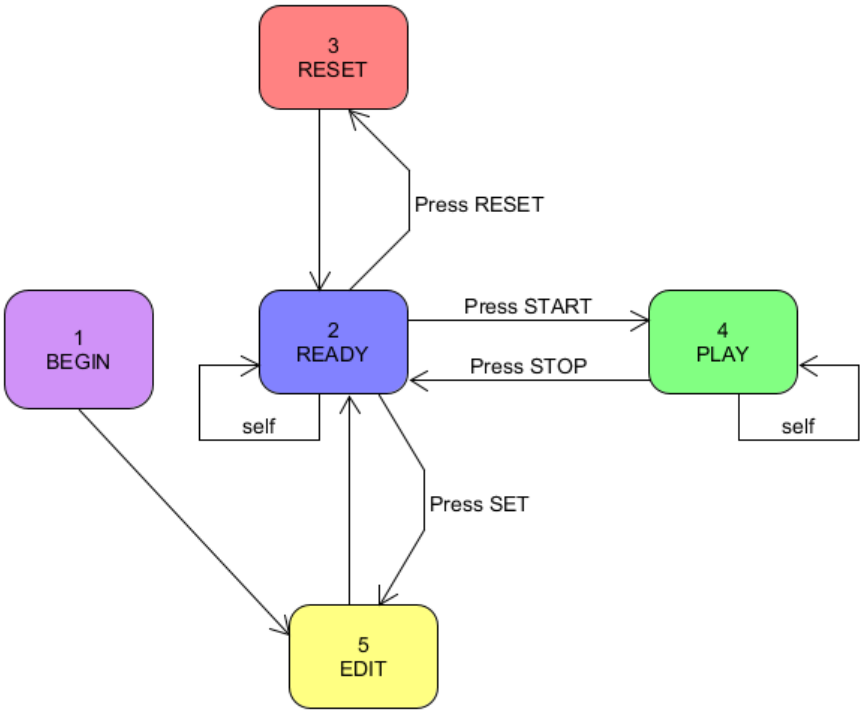

Fig. 10: Máquina de estados finitos.

ID	Nombre	Descripcion	Razon	Destino
1	BEGIN	Se carga el primer estado del juego	Se alimenta el sistema	5
2	READY	Pausar el juego, conjelar las celulas	Estado por defecto. Se presiona STOP en 4.	2, 3, 4, 5
3	RESTART	Recrear el ultimo patron guardado en la sesion	Se presiona el boton RESET en 2	2
4	PLAY	Transcurrir con el juego.	Se presiona el boton START en 2	4, 2
5	EDIT	Se cargan distintos patrones predefinidos en un ciclo (7 en total)	Se presiona el boton SET en 2	2

Tabla 2: Descripción de estados y transiciones.

Parte 2: Diseño de Hardware: Circuito impreso PCB

Se decidió llevar a cabo el diseño en una única hoja, ya que la cantidad de mallas no justifica una mayor cantidad. En la primera hoja (Fig. 11) pueden apreciarse los tres módulos principales del sistema.

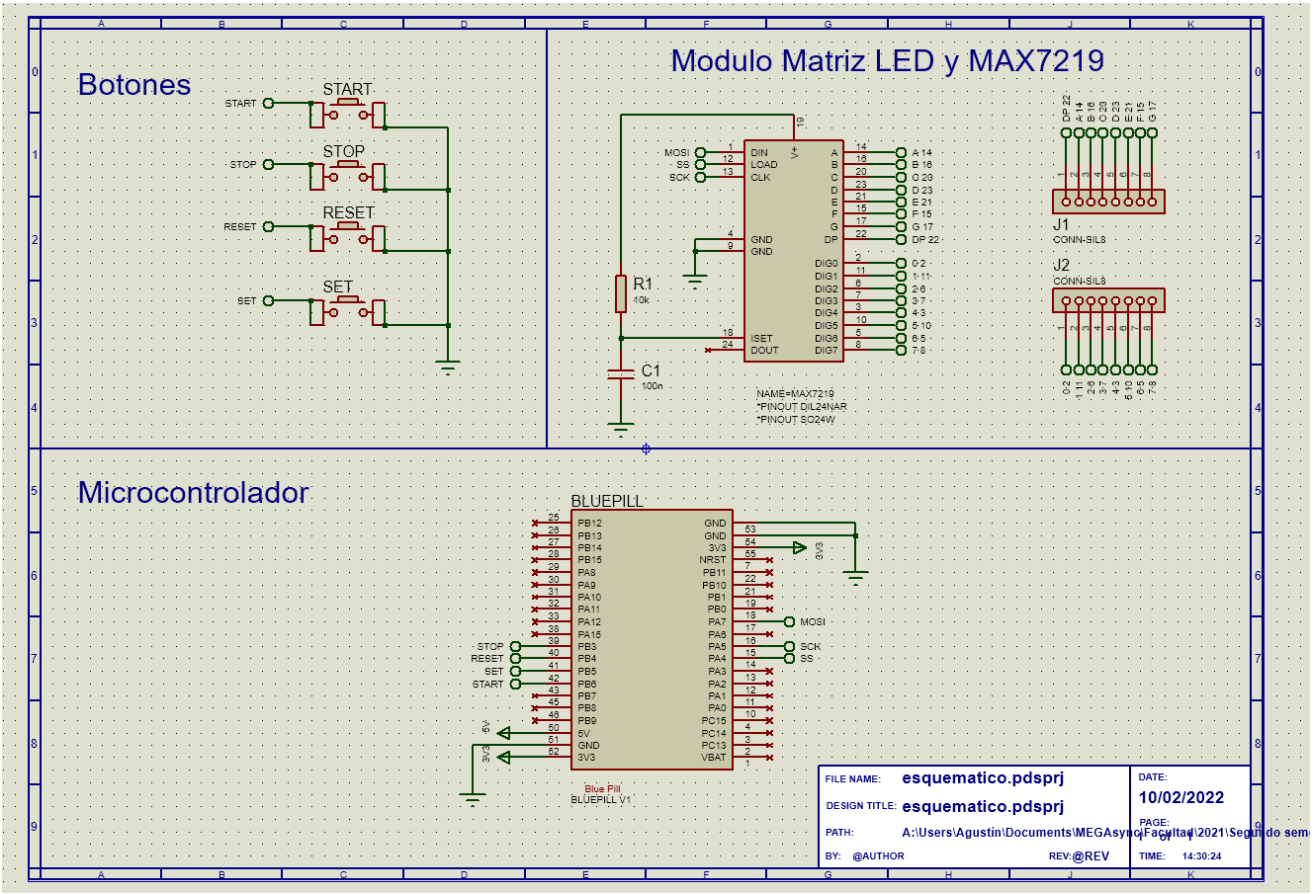

Fig. 1: Hoja 1 del esquemático

Microcontrolador

Para este trabajo, se reutilizó el componente creado en el TP3, diseñado con las especificaciones de la placa de desarrollo BluePill. Las figuras 12, 13 y 14 muestran dicho componente

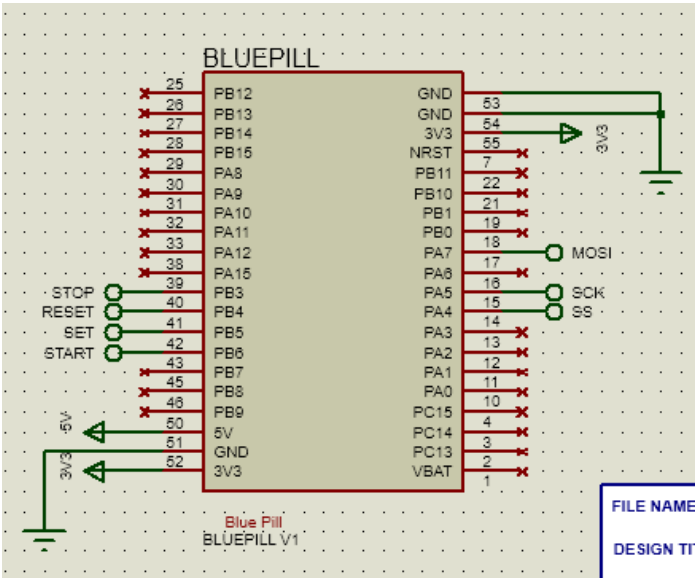

Fig. 12: Diseño de microcontrolador en esquemático

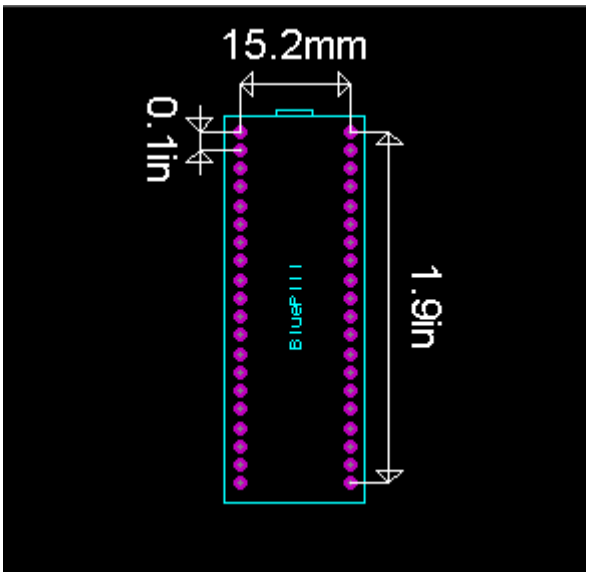

Fig. 13: Diseño PCB

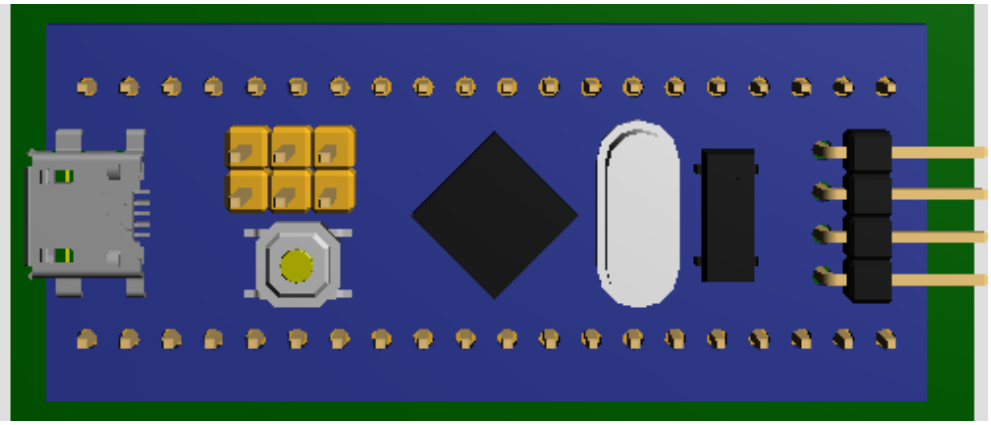

Fig. 14: Modelo 3D

Botones

Para los botones, se optó por la utilización de los clásicos “Tact switches” que suelen venir en diseños de este estilo (Fig. 15). Para ello, fue necesario crear una huella que permita su implementación en la PCB (Fig. 16), y se le agregó un modelo 3D[1] asociado (Fig. 17)

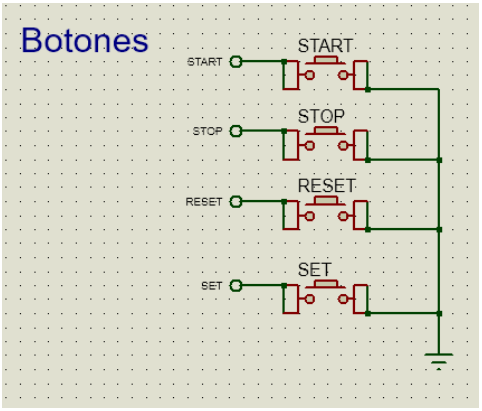

Fig. 15: Diseño en esquemático

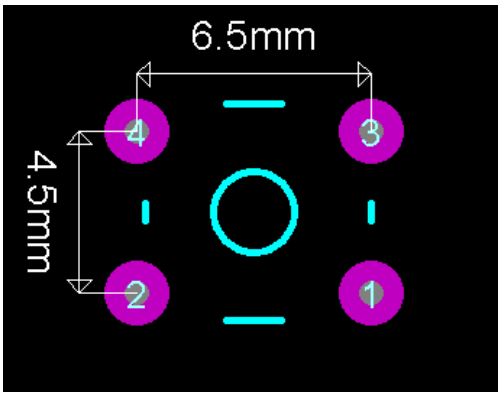

Fig. 16: Diseño PC

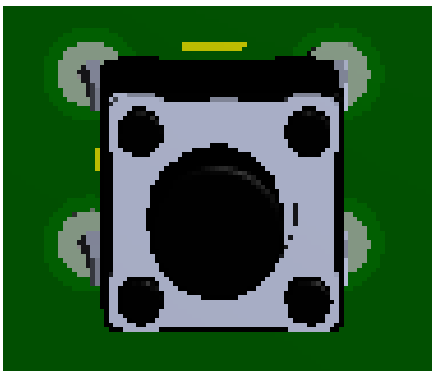

Fig. 17: Modelo 3D

[1] Todos los modelos 3D que no fueron provistos desde la librería de Proteus, fueron descargados de la página web GrabCad. En la sección Enlaces y Fuentes se encuentran los enlaces asociados a la publicación original, y el crédito pertenece a los autores de los mismos.

Matriz LED y MAX7219

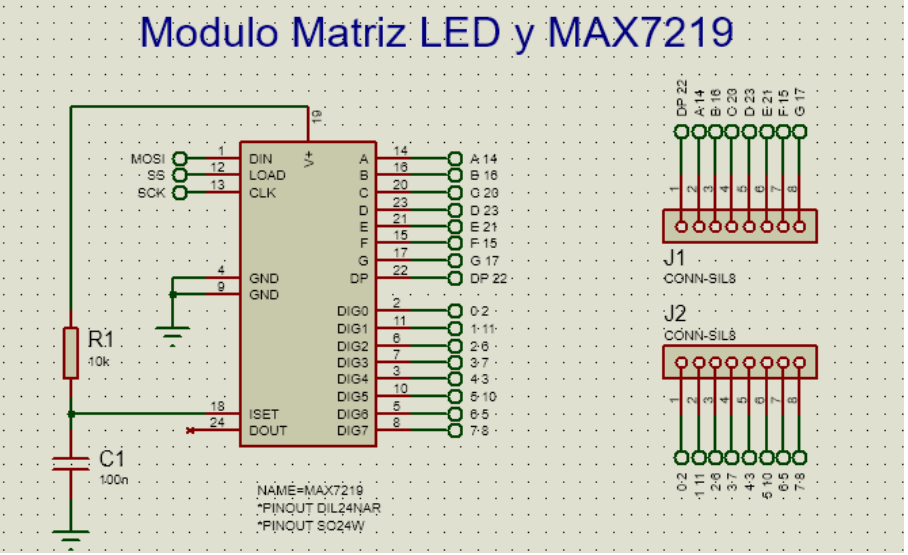

Fig. 18: Conexión de ambos componentes

La conexión del conjunto (Fig. 18) se llevó a cabo como en varios ejemplos realizados por la cátedra y en trabajos prácticos anteriores. Se le agregó una resistencia de 10KΩ para limitar la corriente máxima sobre los LEDs, junto con un capacitor de desacople de 100nF.

Para conectar la matriz LED, se optó por utilizar conectores de pines estándar a fin de simplificar el diseño.

Las figuras 19 y 20 muestran el resultado del diseño PCB y del modelo 3D, respectivamente.

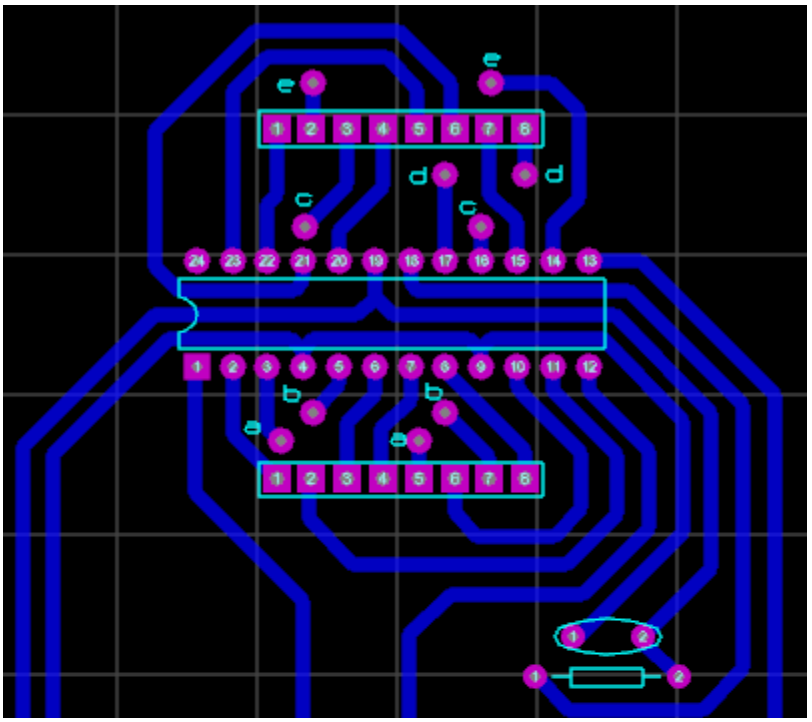

Fig. 19: Diseño PCB

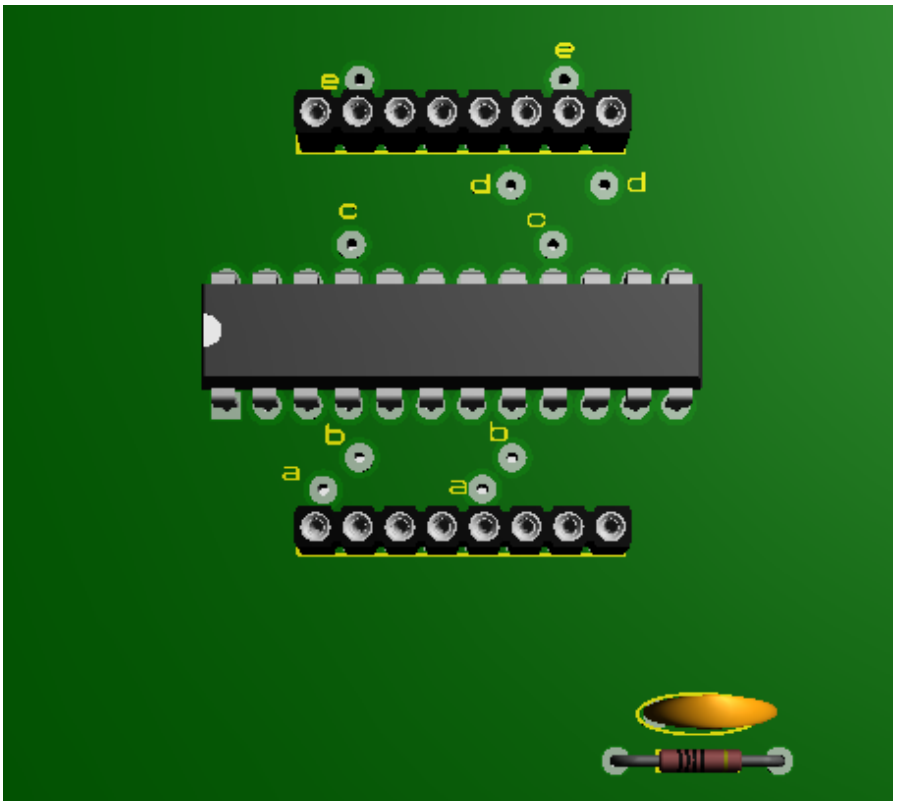

Fig. 20: Modelo 3D

Diseño de la PCB

Para cumplir con los requerimientos, se crearon huellas especiales que coincidan con los mismos. Las figuras 7 y 8 muestran los mismos.

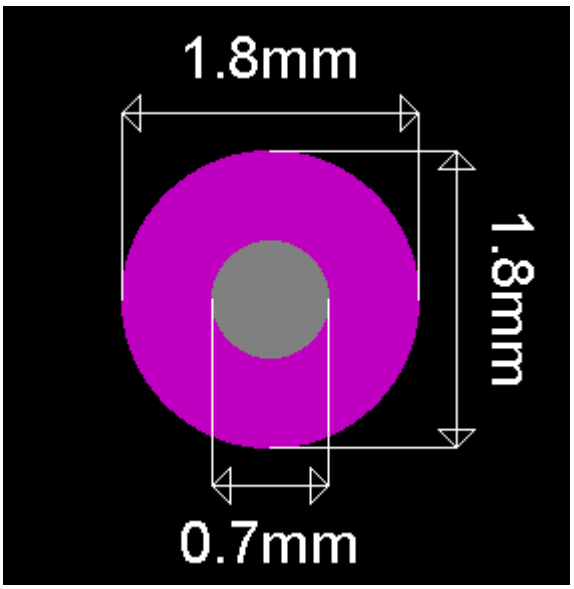

Fig. 21: Versión circular del through hole.

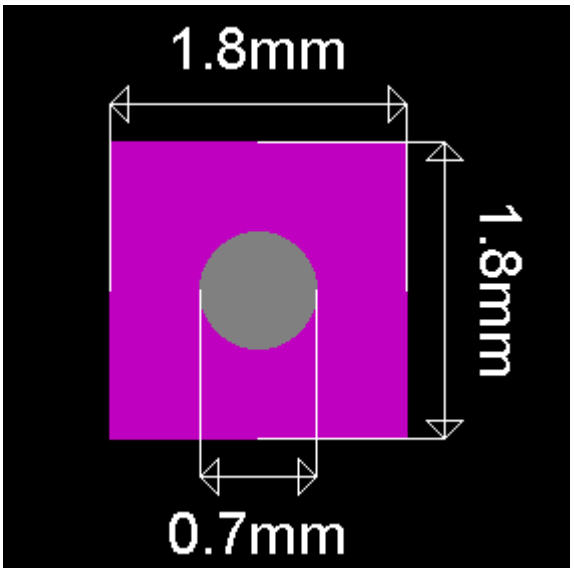

Fig. 22: Versión cuadrada del through hole.

El ancho de las pistas fue también modificado para cumplir con los requerimientos. Esto puede verse en la figura 9:

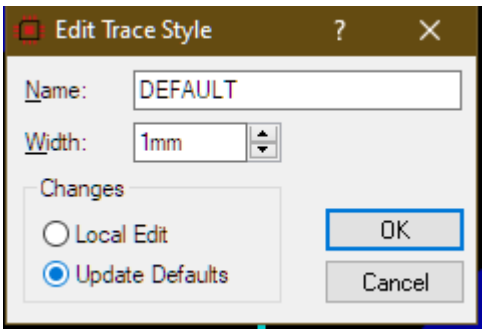

Fig. 23: Modificación del ancho de los trazos.

Se logró que, a pesar de la cantidad de pines a conectar y la mala disposición de los mismos en el empaquetado del MAX7219, solo se necesiten realizar cinco puentes por sobre la PCB para las conexiones de algunas líneas de LEDs. La figura 24 muestra el resultado final de la PCB.

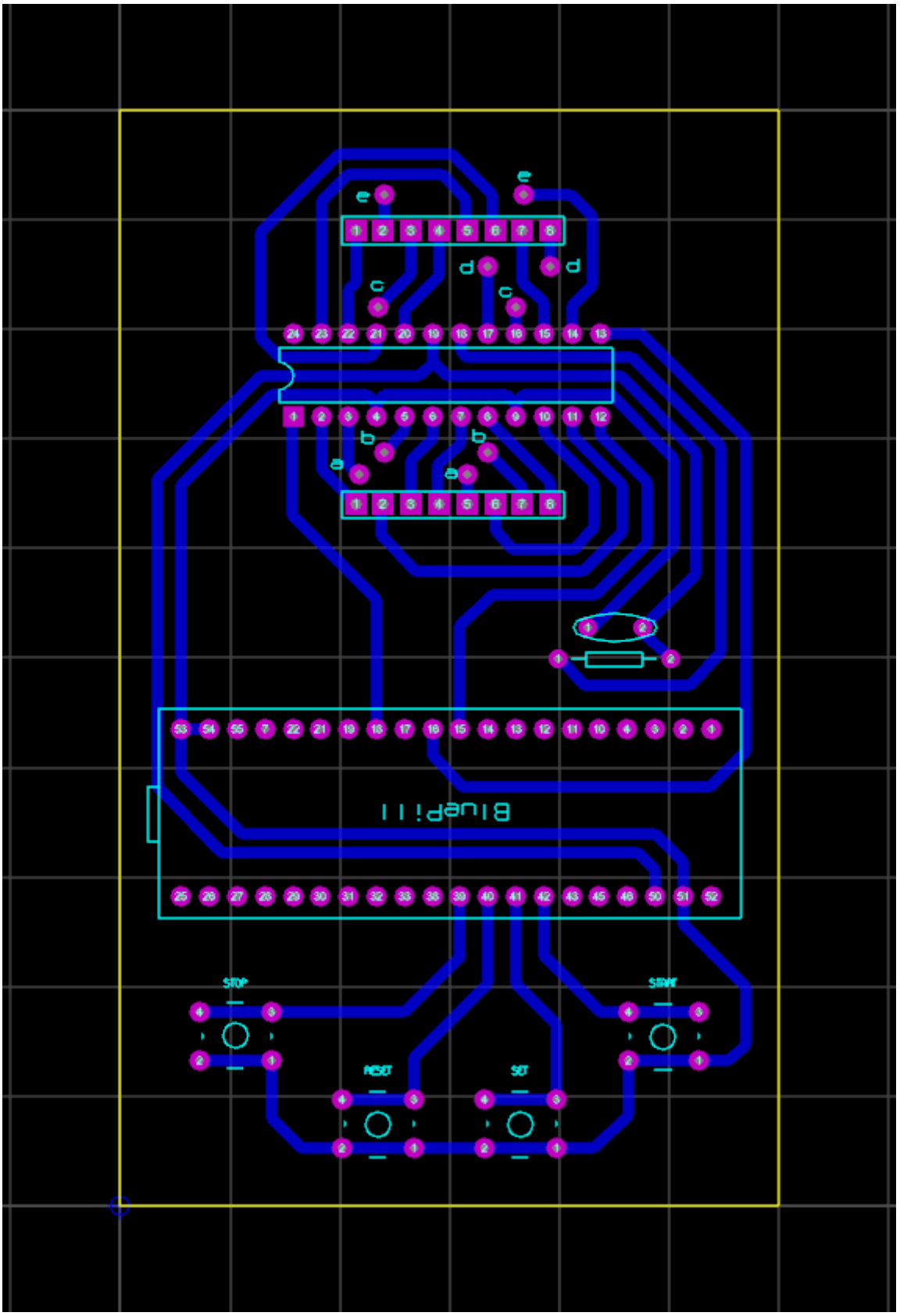

Fig. 24: Diseño de la PCB

Las figuras 25, 26 y 27, muestran los modelos en 3D del circuito final.

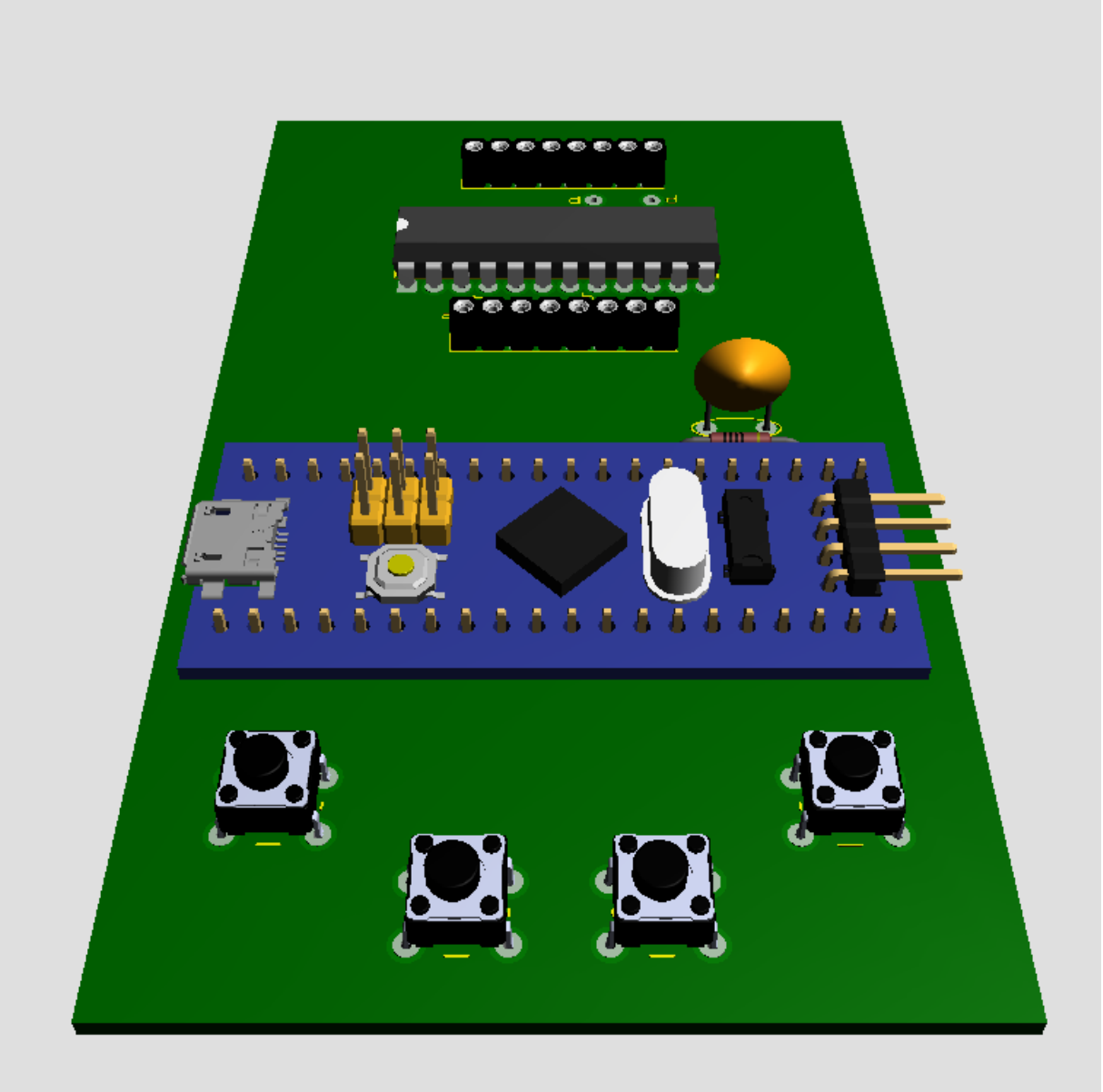

Fig. 25: muestra sesgada del diseño.

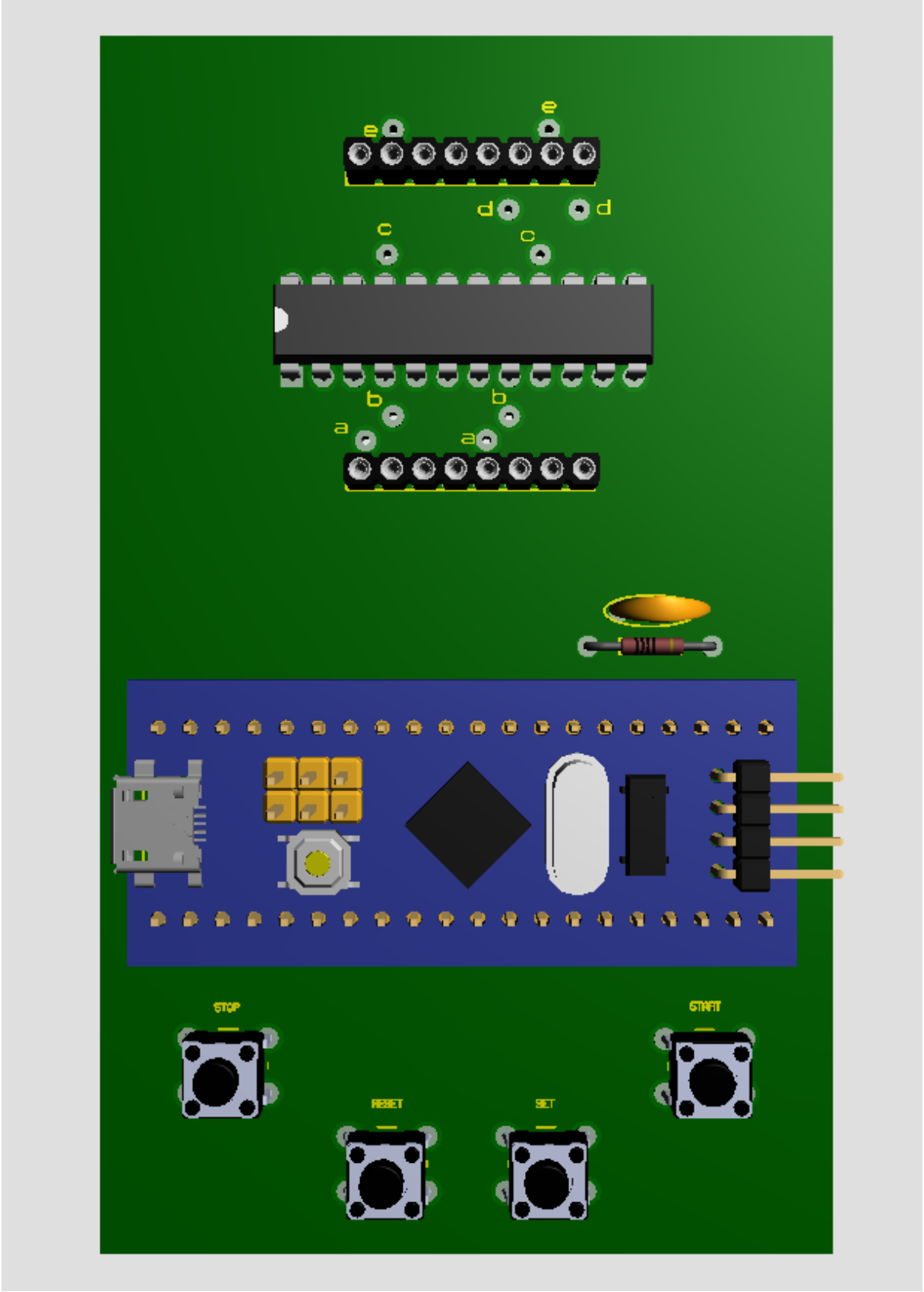

Fig. 26: Vista frontal

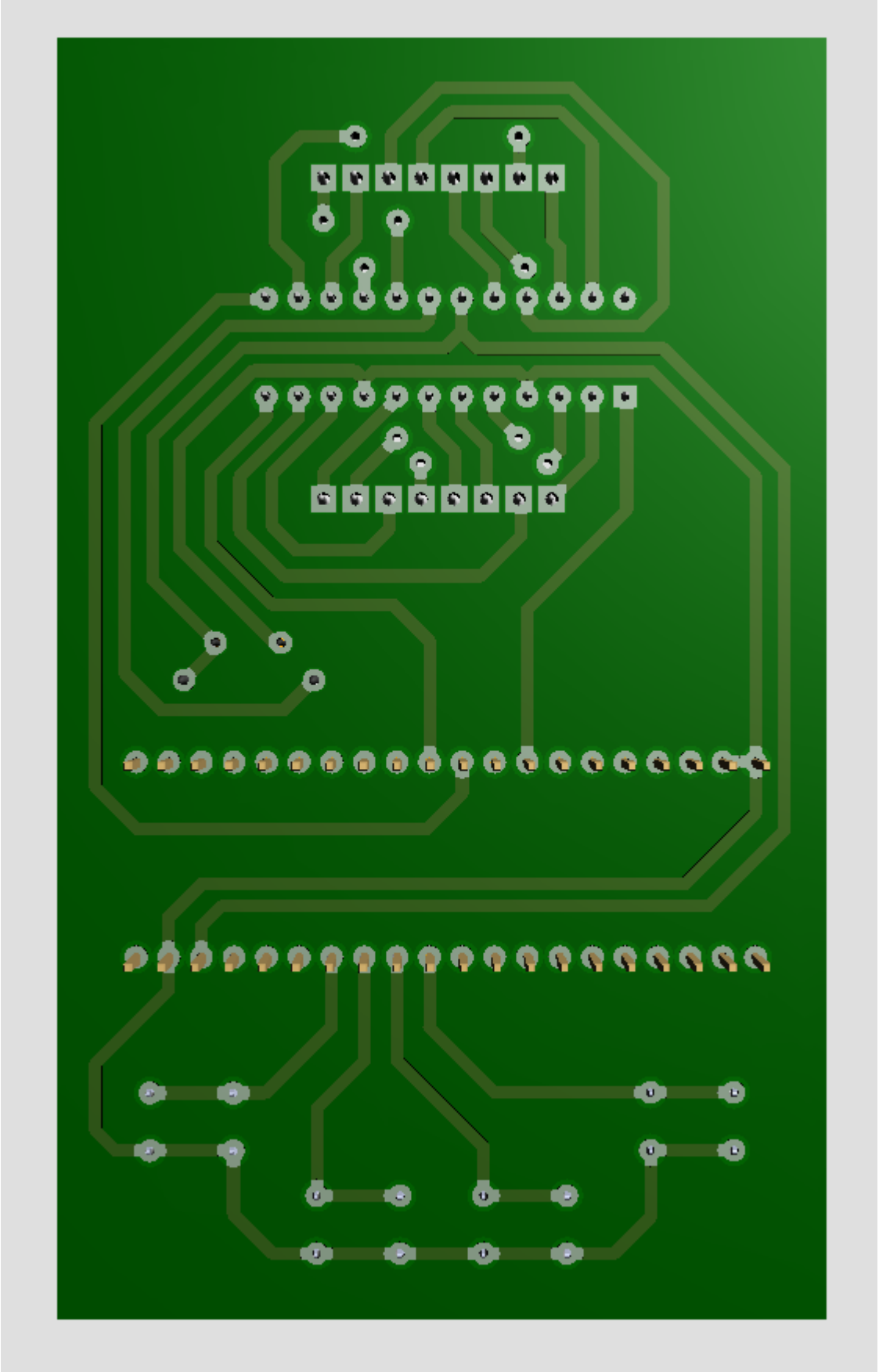

Fig. 27: Vista trasera

Enlaces y fuentes

Conway’s Game of Life: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

MAX 7219: https://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf

Modelo 3D BluePill: https://grabcad.com/library/stm32f103c8t6-blue-pill-board-1

Modelo 3D Tactile Switch: https://grabcad.com/library/tactile-switch-short-1

Repositorio del proyecto en Github: https://github.com/alcaolpg/TP1-4

Demostración de la Simulación: https://youtu.be/U3Y5CLMbZG8

Código

main.c

```
#include <stm32f103x6.h>
#include <spi_config.h>
#include <max7219.h>
#include <stdlib.h>
#include <delay.h>
#include <definitions.h>
#include <fsm.h>
#include <buttons.h>
#include <seos.h>

void SysTick_Handler(void)
{
    seos_task_scheduler();
}

int main (void)
{
// Write your code here

      uint8_t led_matrix[BOARD_SIZE][BOARD_SIZE] = {
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
      };
      int button;

      spi_init();
      max7219_init();
      buttons_init();
      fsm_state_selector(NOP, led_matrix);
      fsm_state_selector(NOP, led_matrix);
      max7219_load(led_matrix);
      max7219_refresh();
```

```
	SysTick_Config(SYS_FREQUENCY / 1000);


	while (1)
	{
		if(seos_get_flag(BUTTON_SCANNING))
		{
			seos_clear_flag(BUTTON_SCANNING);
			button = button_press();
		}
		if(seos_get_flag(MATRIX_REFRESH))
		{
			seos_clear_flag(MATRIX_REFRESH);
			max7219_load(led_matrix);
			max7219_refresh();
		}
		if (seos_get_flag(STATE_MACHINE))
		{
			seos_clear_flag(STATE_MACHINE);
			fsm_state_selector(button, led_matrix);
		}
	}
}
```

spi_config.c

```c
#include <spi_config.h>
#include <stm32f103x6.h>

void spi_init(void);
void spi_send(uint8_t data);

void spi_init(void)
{
   RCC->APB2ENR |= (1<<2)|(1<<12);
   GPIOA->CRL = 0xB4B34444;
   SPI1->CR1 = 0x37C;
}

void spi_send(uint8_t data)
{
   SPI1->DR = data;
   while((SPI1->SR & (1<<0)) == 0);
}
```

```c
#include <stm32f103x6.h>
void spi_init(void);
void spi_send(uint8_t data);
```

max7219.c

```c
#include <max7219.h>
#include <spi_config.h>
#include <stm32f103x6.h>
#include <delay.h>


uint16_t matrix_lines[8] = {0};

void max7219_send(uint8_t cmd, uint8_t data);
void max7219_init(void);


void max7219_send(uint8_t cmd, uint8_t data)
{
    delay_ms(1);
    GPIOA->BRR = (1<<4); //Low signal enables chip selection
    spi_send(cmd);
    spi_send(data);
    GPIOA->BSRR = (1<<4); //High signal disables chip selection
}

void max7219_init(void)
{
    max7219_send(0x09, 0x00); //decode mode: No decode
    max7219_send(0x09, 0x00); //decode mode: No decode
    max7219_send(0x0a, 0x08); //brightness PWM: dutty cycle = 17/32
    max7219_send(0x0b, 0x07); //scan limit: 8 leds
    max7219_send(0x0c, 0x01); //Shutdown mode: Normal operation
    max7219_send(0x0f, 0x00); //test mode: No test
}

void max7219_refresh(void)
{
    volatile int i;
    for(i = 0; i < 8; i++)
    {
        max7219_send(i+1, matrix_lines[i]);
    }
    max7219_send(8, matrix_lines[7]);
}
```

```
void max7219_load(uint8_t (*playboard)[8])
{
    uint8_t i, j;
    for (j = 0; j < 8; j++)
    {
        matrix_lines[j] = 0;
        for (i = 0; i < 8; i++)
        {
            {
                matrix_lines[j] |= (playboard[j][i] << (7 - i));
            }
        }
    }
}
```

```
#include <stm32f103x6.h>
void max7219_send(uint8_t cmd, uint8_t data);
void max7219_init(void);
void max7219_refresh(void);
void max7219_load(uint8_t (*playboard)[8]);
```

delay.c

```c
#include <stm32f103x6.h>
void delay_ms(uint16_t time);

void delay_ms(uint16_t time)
{  //Funcion de retardo
    volatile unsigned long l = 0;
    uint16_t i;

    for(i = 0; i < time; i++)
    {
        for(l = 0; l < 667; l++);
    }
}
```

```c
#include <stm32f103x6.h>
void delay_ms(uint16_t time);
```

fsm.c

```c
#include <stm32f103x6.h>
#include <spi_config.h>
#include <max7219.h>
#include <stdlib.h>
#include <definitions.h>
#include <buttons.h>


typedef struct cells
{
   uint8_t previous_state;
   uint8_t current_state;
   uint8_t edit;
   uint8_t neighbors;
} cell;

cell playboard[BOARD_SIZE][BOARD_SIZE];
cell playboard_copy[BOARD_SIZE][BOARD_SIZE];


uint8_t state = BEGIN;

void fsm_state_selector(int button, uint8_t (*led_matrix)[BOARD_SIZE]);
uint8_t fsm_1_begin(void);
uint8_t fsm_2_ready(int button);
uint8_t fsm_3_reset(void);
uint8_t fsm_4_play(int button);
uint8_t fsm_5_edit(void);
void fsm_clear_playboard(void);
void fsm_copy_playboard(void);
void fsm_to_led_matrix(uint8_t (*led_matrix)[BOARD_SIZE]);
uint8_t fsm_check_neighbors(uint8_t x, uint8_t y);
void fsm_save_copy(void);
void fsm_draw(uint8_t);
void fsm_load_copy(void);




void fsm_state_selector(int button, uint8_t (*led_matrix)[BOARD_SIZE])
```

```
{
    switch (state)
    {
    case BEGIN:
        state = fsm_1_begin();
        break;
    case READY:
        state = fsm_2_ready(button);
        break;
    case RESTART:
        state = fsm_3_reset();
        break;
    case PLAY:
        state = fsm_4_play(button);
        break;
    case EDIT:
        state = fsm_5_edit();
        break;

    default:
        state = BEGIN;
        break;
    }

    button_release();
    fsm_to_led_matrix(led_matrix);

}

uint8_t fsm_1_begin(void)
{
    uint8_t next_state = READY;

    fsm_clear_playboard();
    fsm_5_edit();

    return next_state;
}

uint8_t fsm_2_ready(int button)
{
    uint8_t next_state = READY;
```

```
    switch (button)
    {
    case RESET:
        next_state = RESTART;
        break;
    case START:
        next_state = PLAY;
        break;
    case SET:
        next_state = EDIT;
        break;
    default:
        break;
    }
    return next_state;
}

uint8_t fsm_3_reset(void)
{
    uint8_t next_state = READY;

    fsm_load_copy();

    return next_state;
}

uint8_t fsm_4_play(int button)
{
    uint8_t i, j, next_state = READY;
    if (button != STOP)
    {
        next_state = PLAY;
        for (i = 0; i < BOARD_SIZE; i++)
        {
            for (j = 0; j < BOARD_SIZE; j++)
            {
                                                playboard[i][j].previous_state   =
playboard[i][j].current_state;
            }
        }
```

```
        for (i = 0; i < BOARD_SIZE; i++)
        {
            for (j = 0; j < BOARD_SIZE; j++)
            {
                playboard[i][j].neighbors = fsm_check_neighbors(i, j);

                if (playboard[i][j].current_state == 0)
                {
                    if (playboard[i][j].neighbors == 3)
                    {
                        playboard[i][j].current_state = 1;
                    }
                }
                else
                {
                                    if  (playboard[i][j].neighbors  <  2  ||
playboard[i][j].neighbors > 3)
                    {
                        playboard[i][j].current_state = 0;
                    }
                }
            }
        }

    }
    return next_state;
}

uint8_t fsm_5_edit(void)
{
    static uint8_t x = 0;
    uint8_t next_state = READY;
    fsm_draw(x);
    fsm_save_copy();

    x++;

    if (x > 7)
    {
        x = 0;
    }
```

```
    return next_state;
}


void fsm_clear_playboard(void)
{
    uint8_t i, j;
    for (i = 0; i < BOARD_SIZE; i++)
    {
        for (j = 0; j < BOARD_SIZE; j++)
        {
            playboard[i][j].previous_state = 0;
            playboard[i][j].current_state = 0;
            playboard[i][j].neighbors = 0;
            playboard[i][j].edit = 0;
        }
    }
}

void fsm_save_copy(void)
{
    uint8_t i, j;
    for (i = 0; i < BOARD_SIZE; i++)
    {
        for (j = 0; j < BOARD_SIZE; j++)
        {
                                        playboard_copy[i][j].previous_state    =
playboard[i][j].previous_state;
                                         playboard_copy[i][j].current_state    =
playboard[i][j].current_state;
            playboard_copy[i][j].neighbors = playboard[i][j].neighbors;
            playboard_copy[i][j].edit = 0;
        }
    }
}

void fsm_load_copy(void)
{
    uint8_t i, j;
    for (i = 0; i < BOARD_SIZE; i++)
    {
        for (j = 0; j < BOARD_SIZE; j++)
```

```
        {
                                                playboard[i][j].previous_state    =
playboard_copy[i][j].previous_state;
                                                 playboard[i][j].current_state    =
playboard_copy[i][j].current_state;
            playboard[i][j].neighbors = playboard_copy[i][j].neighbors;
            playboard[i][j].edit = 0;
        }
    }
}

void fsm_to_led_matrix(uint8_t (*led_matrix)[BOARD_SIZE])
{
    uint8_t i, j;
    for (i = 0; i < BOARD_SIZE; i++)
    {
        for (j = 0; j < BOARD_SIZE; j++)
        {
            led_matrix[i][j] = playboard[i][j].current_state;
        }
    }
}

uint8_t fsm_check_neighbors(uint8_t i, uint8_t j)
{
    uint8_t neighbors = 0;

    if(i - 1 >= 0)
    {
        if (playboard[i-1][j].previous_state == 1)
        {
            neighbors++;
        }
    }

    if(i-1 >= 0 && j + 1 < BOARD_SIZE)
    {
        if (playboard[i-1][j+1].previous_state == 1)
        {
            neighbors++;
        }
    }
```

```
if(j + 1 < BOARD_SIZE)
{
    if (playboard[i][j+1].previous_state == 1)
    {
        neighbors++;
    }
}

if(i + 1 < BOARD_SIZE && j + 1 < BOARD_SIZE)
{
    if (playboard[i+1][j+1].previous_state == 1)
    {
        neighbors++;
    }
}

if(i + 1 < BOARD_SIZE)
{
    if (playboard[i+1][j].previous_state == 1)
    {
        neighbors++;
    }
}

if(i + 1 < BOARD_SIZE && j - 1 >= 0)
{
    if (playboard[i+1][j-1].previous_state == 1)
    {
        neighbors++;
    }
}

if(j - 1 >= 0)
{
    if (playboard[i][j-1].previous_state == 1)
    {
        neighbors++;
    }
}

if(i - 1 >= 0 && j - 1 >= 0)
```

```
    {
        if (playboard[i-1][j-1].previous_state == 1)
        {
            neighbors++;
        }
    }

    return neighbors;
}

void fsm_draw(uint8_t x)
{
     uint8_t i;

    fsm_clear_playboard();

    switch (x)
    {
    case 0:
            playboard[0][3].current_state = 1;
            playboard[1][3].current_state = 1;
            playboard[2][3].current_state = 1;
            playboard[4][0].current_state = 1;
            playboard[4][2].current_state = 1;
            playboard[4][3].current_state = 1;
            playboard[4][4].current_state = 1;
            playboard[4][6].current_state = 1;
            playboard[4][7].current_state = 1;
            playboard[6][3].current_state = 1;
            playboard[7][3].current_state = 1;
        break;

    case 1:
            playboard[3][4].current_state = 1;
            playboard[3][5].current_state = 1;
            playboard[3][6].current_state = 1;
        break;

    case 2:
            playboard[0][0].current_state = 1;
            playboard[1][1].current_state = 1;
            playboard[1][2].current_state = 1;
```

```
            playboard[2][0].current_state = 1;
            playboard[2][1].current_state = 1;
        break;

    case 3:
            playboard[2][3].current_state = 1;
            playboard[2][4].current_state = 1;
            playboard[4][2].current_state = 1;
            playboard[4][3].current_state = 1;
            playboard[4][4].current_state = 1;
            playboard[4][5].current_state = 1;
            playboard[6][3].current_state = 1;
            playboard[6][4].current_state = 1;
        break;

    case 4:
            playboard[1][4].current_state = 1;
            playboard[2][3].current_state = 1;
            playboard[2][5].current_state = 1;
            playboard[3][2].current_state = 1;
            playboard[3][6].current_state = 1;
            playboard[4][3].current_state = 1;
            playboard[4][5].current_state = 1;
            playboard[5][4].current_state = 1;
        break;

    case 5:
            playboard[0][0].current_state = 1;
            playboard[0][1].current_state = 1;
            playboard[0][2].current_state = 1;
            playboard[0][4].current_state = 1;
            playboard[0][5].current_state = 1;
            playboard[0][6].current_state = 1;
            playboard[1][1].current_state = 1;
            playboard[1][4].current_state = 1;
            playboard[1][6].current_state = 1;
            playboard[2][1].current_state = 1;
            playboard[2][4].current_state = 1;
            playboard[2][5].current_state = 1;
            playboard[2][6].current_state = 1;
            playboard[3][1].current_state = 1;
            playboard[3][4].current_state = 1;
```

```
            playboard[5][2].current_state = 1;
            playboard[5][3].current_state = 1;
            playboard[5][4].current_state = 1;
            playboard[6][3].current_state = 1;
            playboard[7][2].current_state = 1;
            playboard[7][3].current_state = 1;
            playboard[7][4].current_state = 1;
        break;

    case 6:
            playboard[1][3].current_state = 1;
            playboard[2][2].current_state = 1;
            playboard[2][3].current_state = 1;
            playboard[2][6].current_state = 1;
            playboard[3][1].current_state = 1;
            playboard[3][3].current_state = 1;
            playboard[3][5].current_state = 1;
            playboard[4][0].current_state = 1;
            playboard[4][3].current_state = 1;
            playboard[4][4].current_state = 1;
            playboard[5][3].current_state = 1;
        break;

    case 7:
            for(i = 0; i < BOARD_SIZE; i++)
            {
                playboard[i][0].current_state = 0;
                playboard[i][1].current_state = 0;
                playboard[i][2].current_state = 1;
                playboard[i][3].current_state = 0;
                playboard[i][4].current_state = 0;
                playboard[i][5].current_state = 1;
                playboard[i][6].current_state = 0;
                playboard[i][7].current_state = 0;
            }

    default:
        break;
    }
}
```

```c
#include <stm32f103x6.h>
#include <definitions.h>

void fsm_state_selector(int button, uint8_t (*led_matrix)[BOARD_SIZE]);
```

buttons.c

```c
#include <stm32f103x6.h>
#include <definitions.h>
#include <delay.h>

void buttons_init(void);
uint8_t button_scan(uint8_t);
uint8_t button_press(void);
void button_release(void);

uint8_t button = NOP;

void buttons_init(void)
{
    RCC->APB2ENR |= (1<<3);
    GPIOB->CRL = 0x48888444;
    GPIOB->ODR |= (1<<3) | (1<<4) | (1<<5) | (1<<6);
}

uint8_t button_press(void)
{
    if(button_scan(6)) button = START;
    else if(button_scan(5)) button = SET;
    else if(button_scan(4)) button = RESET;
    else if (button_scan(3)) button = STOP;

    return button;
}

void button_release(void)
{
    button = NOP;
}

uint8_t button_scan(uint8_t pin)
{
    if ((GPIOB->IDR & (1<<pin)) == 0)
    {
        delay_ms(10);
        if ((GPIOB->IDR & (1<<pin)) == 0)
        {
```

```
            return 1;
        }
    }
    return 0;
}
```

```
#include <stm32f103x6.h>

void buttons_init(void);
uint8_t button_press(void);
void button_release(void);
```

seos.c

```c
#include <stm32f103x6.h>
#include <definitions.h>

#define TIME_button_scanning 50
#define TIME_matrix_refresh 100
#define TIME_state_machine_update 200

volatile uint8_t FLAG_button_scanning = 0;
volatile uint8_t FLAG_matrix_refresh = 0;
volatile uint8_t FLAG_state_machine_update = 0;

static uint16_t COUNTER_button_scanning = TIME_button_scanning - 1;
static uint16_t COUNTER_matrix_refresh = TIME_matrix_refresh - 1;
static uint16_t COUNTER_state_machine_update = TIME_state_machine_update - 1;

void seos_init(void);
void seos_task_scheduler(void);
uint8_t seos_get_flag(uint8_t flag);
void seos_clear_flag(uint8_t flag);


void seos_task_scheduler(void)
{
    if(++COUNTER_button_scanning >= TIME_button_scanning)
    {
        COUNTER_button_scanning = 0;
        FLAG_button_scanning = 1;
    }
    if(++COUNTER_matrix_refresh >= TIME_matrix_refresh)
    {
        COUNTER_matrix_refresh = 0;
        FLAG_matrix_refresh = 1;
    }
    if(++COUNTER_state_machine_update >= TIME_state_machine_update)
    {
        COUNTER_state_machine_update = 0;
        FLAG_state_machine_update = 1;
    }
}
```

```c
uint8_t seos_get_flag(uint8_t flag)
{
    switch(flag)
    {
        case BUTTON_SCANNING:
            return FLAG_button_scanning;
        case MATRIX_REFRESH:
            return FLAG_matrix_refresh;
        case STATE_MACHINE:
            return FLAG_state_machine_update;
        default:
            return 0;
    }
}

void seos_clear_flag(uint8_t flag)
{
    switch(flag)
    {
        case BUTTON_SCANNING:
            FLAG_button_scanning = 0;
            break;
        case MATRIX_REFRESH:
            FLAG_matrix_refresh = 0;
            break;
        case STATE_MACHINE:
            FLAG_state_machine_update = 0;
            break;
    }
}
```

```c
#include <stm32f103x6.h>

void seos_init(void);
void seos_task_scheduler(void);
uint8_t seos_get_flag(uint8_t flag);
void seos_clear_flag(uint8_t flag);
```

definitions.h

```
#define SYS_FREQUENCY 8000000

#define START 200
#define SET 201
#define RESET 202
#define STOP 203

#define NOP 0
#define BEGIN 1
#define READY 2
#define RESTART 3
#define PLAY 4
#define EDIT 5
#define CLEAR 6
#define BOARD_SIZE 8

#define BUTTON_SCANNING 101
#define MATRIX_REFRESH 102
#define STATE_MACHINE 103
```