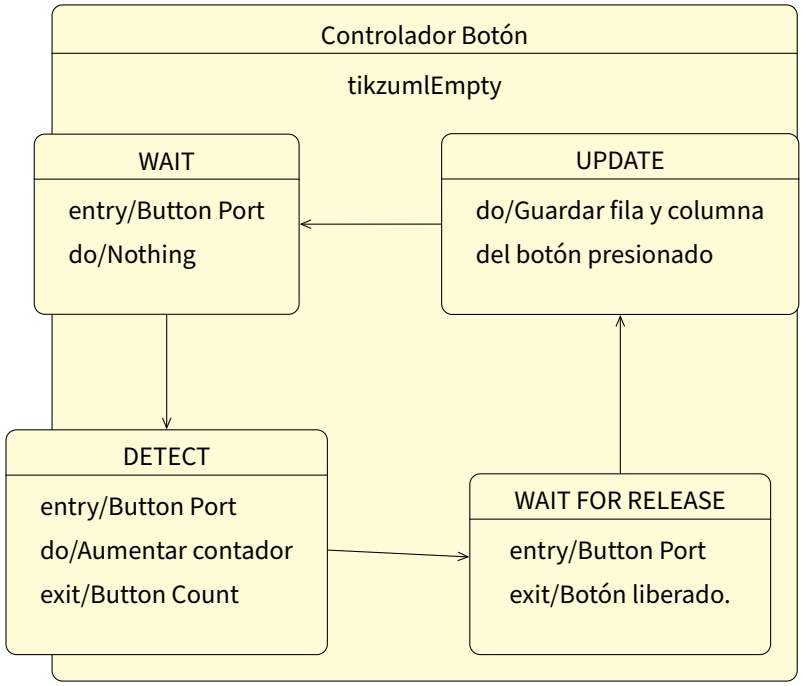
Ejercicio
Safe Lock

Alejandro Ramírez Jaramillo

Universidad Nacional de Colombia sede Manizales Email: alramirezja@unal.edu.co

Problema

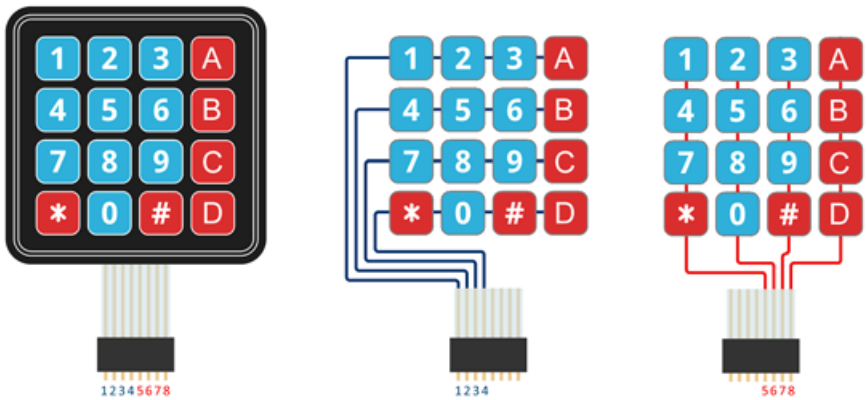
Crear un proyecto que le permita a un usuario ingresar una contraseña que se mostrará en una pantalla, la contraseña ingresada será comparada con una previamente guardada, de ser iguales se "desbloqueará", si no permanecerá "bloqueado", el estado de "bloqueo" se deberá mostrar en la misma pantalla.
Para el proyecto se hará uso de la tarjeta de desarrollo STM32L476RG, un teclado matricial 4x4 y una pantalla OLED, en la guía se explicará como programar el teclado con antirrebote y la pantalla de manera individual, el estudiante deberá usar lo aprendido para completar el proyecto.



1 Teclado 4x4

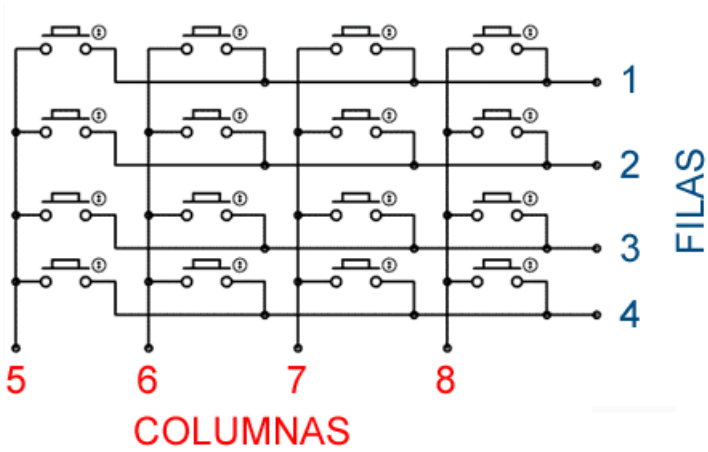
1.1 Funcionamiento del teclado

Se usa un teclado 4x4 de membrana como el que se muestra a continuación, este tiene 8 pines que representan las 4 columnas y 4 filas, usando estos podemos identificar que botón fue presionado. En la siguiente imagen podemos ver cuales pines corresponden a las filas y cuales a las columnas, lo que se usará para identificar que carácter se encuentra en una posición.



Para poder identificar la posición de la tecla presionada tenemos que entender la estructura interna del teclado, que forma una matriz en la que cada elemento corresponde a un botón que conecta fila con columna, si el botón es presionado una fila queda cortocircuitada a una columna, es decir que si conectáramos las filas a una fuente de tensión, al presionar un botón esa tensión aparecería en el pin de la columna correspondiente, así al usar las filas como salidas de tensión y las columnas como entradas, podemos identificar en que columna se encuentra el botón que se presionó.

Una vez identificada la columna, sigue la identificación de la fila, como el voltaje visto en el pin de la columna va a ser el mismo para cualquier botón de dicha columna que sea presionado no se alimentará con voltaje las 4 filas al mismo tiempo, si no que se alimentarán una por una, en cada caso se verificará si algún botón esta siendo presionado, si es así ya sabremos en que fila y columna está dicho botón y podremos identificar el carácter.

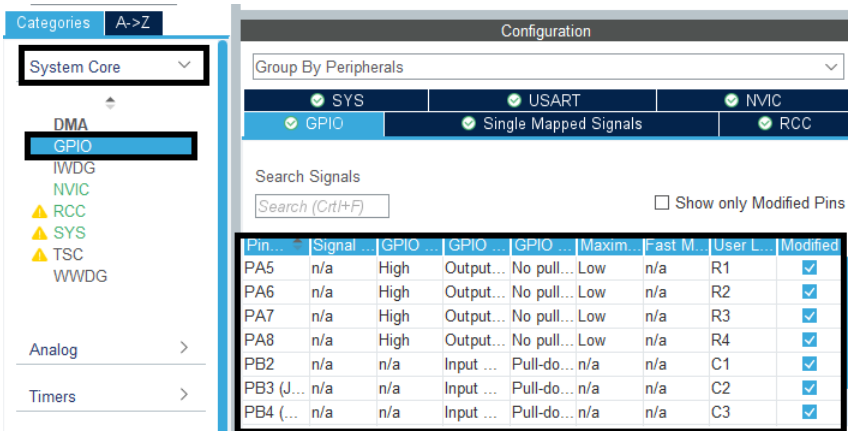


1.2 Configuración de los periféricos

- Pines de salida: Corresponden a los pines que se conectarán a las filas del teclado, son un total de 4 pines que se tendrán que poner en High y Low de manera consecutiva.
- Pines de entrada: Son los pines que se conectarán a las columnas del teclado, son un total de 4 pines que se leerán constantemente para detectar una señal de entrada.
- Timer: Periférico usado para hacer la lectura de los botones cuando pasa una cantidad de tiempo determinada.

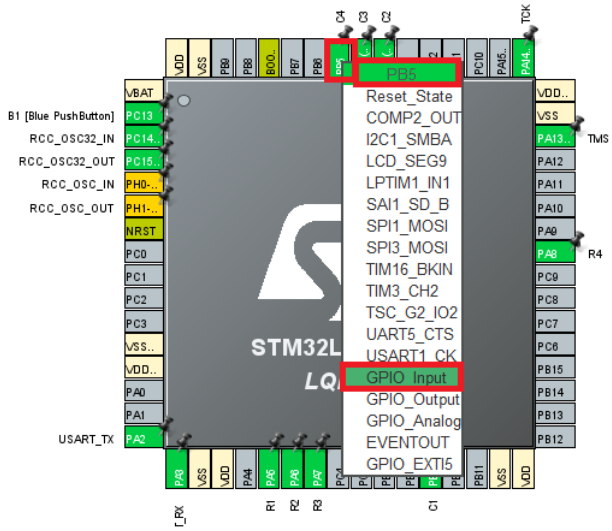
1.2.1 GPIO

Para configurar las entradas y salidas usando el HAL se usará la interfaz del HAL, en la cual podremos seleccionar los pines y configurarlos. Primero vamos a la sección de System Core, donde encontraremos las GPIO, encontraremos ahí una tabla con los pines que se hayan configurado hasta ahora.



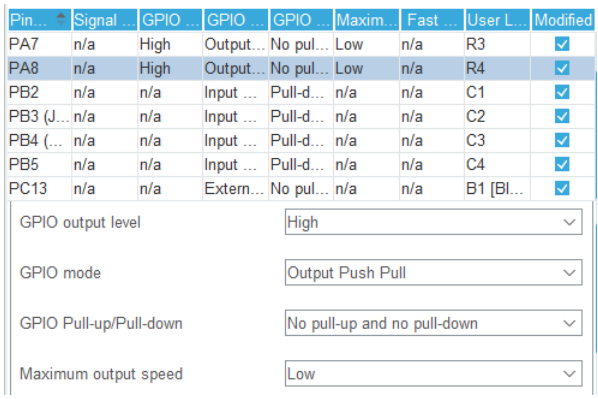
Pin...	Signal ...	GPIO ...	GPIO ...	GPIO ...	Maxim...	Fast M...	User L...	Modified
PA5	n/a	High	Output...	No pull...	Low	n/a	R1	☑
PA6	n/a	High	Output...	No pull...	Low	n/a	R2	☑
PA7	n/a	High	Output...	No pull...	Low	n/a	R3	☑
PA8	n/a	High	Output...	No pull...	Low	n/a	R4	☑
PB2	n/a	n/a	Input ...	Pull-do...	n/a	n/a	C1	☑
PB3 (J...	n/a	n/a	Input ...	Pull-do...	n/a	n/a	C2	☑
PB4 (...	n/a	n/a	Input ...	Pull-do...	n/a	n/a	C3	☑

Entonces usando Pinout view (la ilustración de microcontrolador) podemos configurar un pin al hacer click sobre el, pues se desplegará una lista de las posibles modos de operación, al seleccionar uno la tabla mencionada anteriormente se actualizará, al hacer click en la fila de un pin especifico, se desplegará una lista de características que podremos modificar.

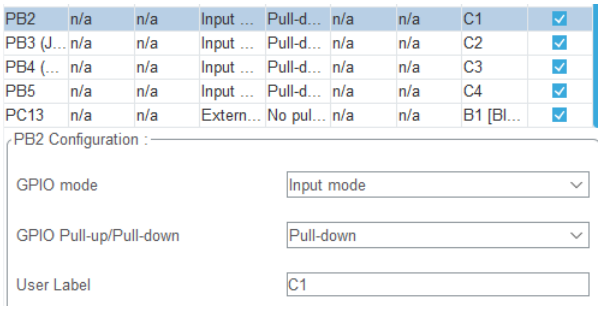


Después de modificar el pin la lista anteriormente mencionada se actualizará, para una configuración

más especifica se hace click al pin en la lista, se abrirá una lista con varias características que podremos modificar, para el caso de las salidas vamos a configurarlas así:

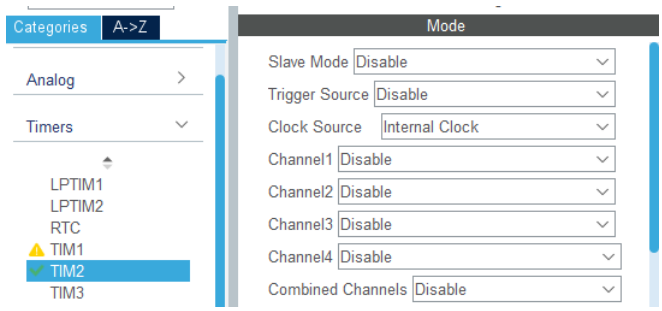


Y las entradas deben configurarse así:

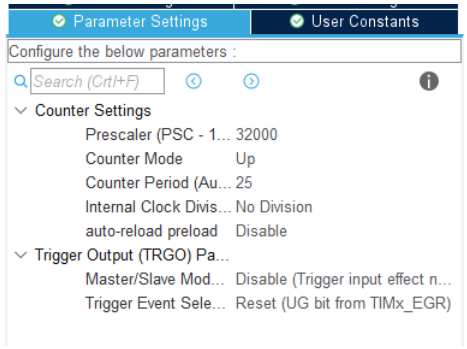


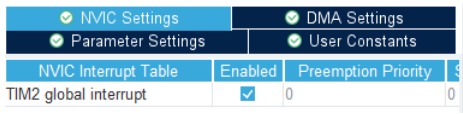
1.2.2 Timer

El Timer se utilizará para revisar periódicamente el estados de los 8 pines, de manera que se podrá identificar cuando un botón haya sido presionado, para evitar que se pase por algo alguna presión, se debe hacer el periodo de muestreo lo suficientemente corto posible. Para comenzar volveremos a la interfaz del HAL y en la sección de timers escogeremos el TIM2, en el menú que se desplegará solo escogeremos la fuente del clock usado por el timer, todo lo demás permanecerá deshabilitado.



Después se escoge el preescalador y el periodo de conteo en Parameter Settings, y habilitamos las interrupciones del timer en NVIC Settings.



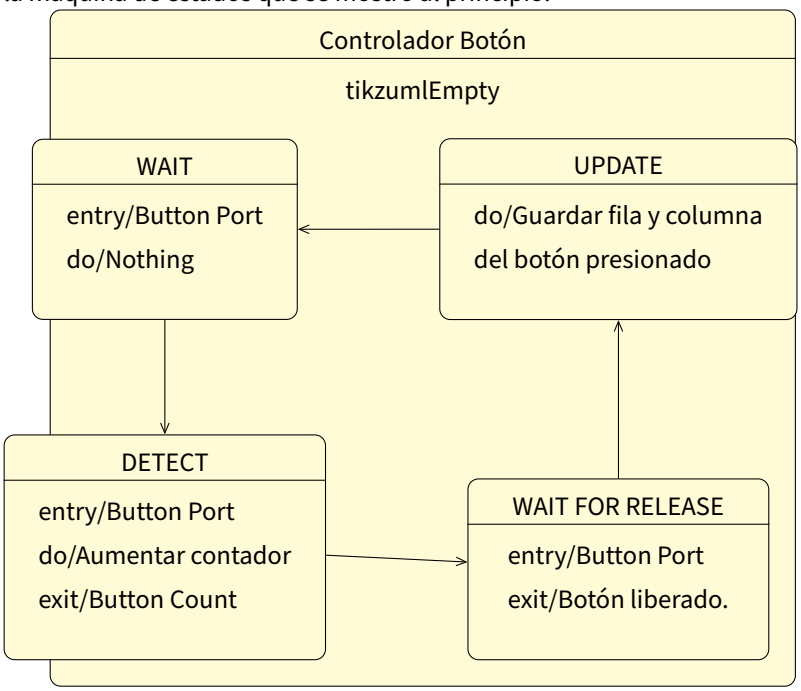


Esto da fin a la configuración de los periféricos del teclado, sigue hacer la lectura de los pines y a través del resultado determinar que botón se presionó.

1.3 Determinación del carácter

La determinación se hace en dos secciones de código distintas, una de ellas se encarga de detectar cuando un botón es presionado, la posición de la fila y columna de dicho botón, además determina que no haya sido un rebote, la segunda sección usa la información de fila y columna para devolver el carácter.

En primer lugar se analizará la detección del botón con antirrebote, para esto usaremos como guía la máquina de estados que se mostró al principio.



- WAIT: Estado de espera, el botón no ha sido presionado, por lo tanto se esta leyendo su estado para detectar cuando pase. Cuando detecte la presión va a guardar la columna (determinado por el pin en el que se detectó la entrada) y la fila (pin en HIGH que se estuvo cambiando periódicamente en el código), reiniciará el contador del botón (para el antirrebote) y pasara al estado Detect.

```
case Waiting:
        if(Button_PORT)   //if any button press detected
        {
                Button_State = Detected;        //change state
                Button_Count = 0;                       //OnEntry/zero the
                    count

                Temp_Press.Full = Button_PORT;  //OnEntry/record the
                    temporary value
                Row_Temp_Press.Row = (Row_PORT & 0x01E0);
        }
        break;
```

- DETECT: El botón ha sido presionado, pero aún no se libera. Cuando se implementa el antirebote se aplica en este estado,aumentando un contador cada vez que se entra al estado y comparándolo con una constante, solo se tiene en cuenta la detección si el contador supera la constante, si es así para al estado de Wait For Realease, si no vuelve a WAIT.

```
case Detected:
        if (Temp_Press.Full == Button_PORT)
        {
                ++Button_Count;    //Guarded state action, if same value
                    increase button count
                if (Button_Count > MIN_BUTTON_COUNT)
                {
                        Button_State = WaitForRelease;   //guarded state
                            transition
                }
        }
        else
        {
                Button_State = Waiting;   //state transition
        }
        break;
```

- WAIT FOR REALEASE: Despúes de verificar que no es un rebote, se espera hasta que el usuario deje de presionar el botón, cuando lo haga se pasa al estado Realeased.

```
case WaitForRelease:
        if (!Button_PORT)
        {
                Button_State = Update;   //state transition when all buttons
                    released
        }
        break;
```

- RELEASED: El botón ha sido liberado, entonces se actualizará el número de la fila y la columna de el botón, se pondrá el contador del antirrebote en cero y volverá al estado Wait.

```
case Update:
        {
                Button_Press = Temp_Press;  //state action         HERE the
                    BUTTON value is updated
                Row_Press = Row_Temp_Press;
                Button_State = Waiting;     //state transition
                Button_Count = 0;           //state action
                Temp_Press.Full=0;          //state action
        }
        break;
```

Pasamos entonces a usar la información de fila y columna para devolver un carácter, para esto en primer lugar si asignan valores numéricos a las columnas y filas.

```
typedef enum {R1=32, R2=64, R3=128, R4=256} rowActive;
typedef enum {C1=4, C2=8, C3=16, C4=32} colActive;
```

Estos equivalen al valor hexadecimal del registro de entrada de los puertos A y B para cada pin, es decir, como la columna 1 corresponde al pin 3 del puerto B, el registro de entrada cuando en ese pin haya un voltaje será 0x4 que equivale al 4 en decimal, lo mismo se aplica para las demás columnas y filas.

La función Find_*Button*_Press nos devolverá la fila y columna, usando un switch case se seleccionará cada caso, por ejemplo, si el botón presionado estuviera en la columna 1:

```
case C1:
        switch(Row_Press.Row)
        {
                case R1:
                        digit = '1';
                break;
                case R2:
                        digit = '4';
                break;
                case R3:
                        digit = '7';
                break;
                case R4:
                        digit = '*';
                break;
        }
        Button_Press.Full=0x00;
```

Entonces según la fila que esté en HIGH en dicho momento se selecciona el carácter. El mismo procedimiento se aplica para las 4 columnas.

2 Oled Screen

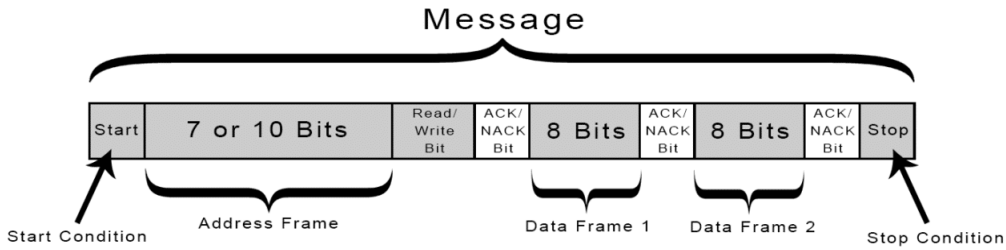
La Oled screen nos permitirá mostrar texto e imágenes simples para transmitir información al usuario de una manera más clara, esto se hace a través del I2C que comunicará el microcontrolador con la pantalla, en esta sección se explicará como funciona el I2C, como debe conectarse a la tarjeta y se explicarán una serie de librerías para la pantalla.

2.1 I2C

Es un puerto y protocolo de comunicación serial que permite la comunicación entre uno o más dispositivos usando solo 2 puertos: SDA (Seria Data) y SCL (Serial Clock), más los puertos de alimentación. Los dispositivos que se comunican se clasifican en Maestro y Esclavo.

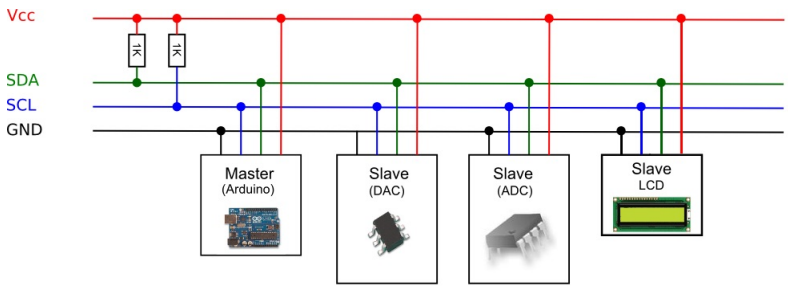
- Maestro: Es el que controla la señal de reloj que se transmite por SCL, la cual se usará para que todos los esclavos interpreten la información con el mismo reloj, además marca el inicio y final a la comunicación usando unos bits específicos en la señal enviada por SDA, puede trabajar como transmisor o receptor.
- Esclavo: Puede funcionar como transmisor o receptor, pero no genera la señal del reloj en SCL, el esclavo también tiene la tarea de enviar una señal de confirmación cuando ha recibido un mensaje.

Un mensaje transmitido por I2C estará compuesto de muchos bits que según su posición tendrán una función diferente, estos se dividirán así:



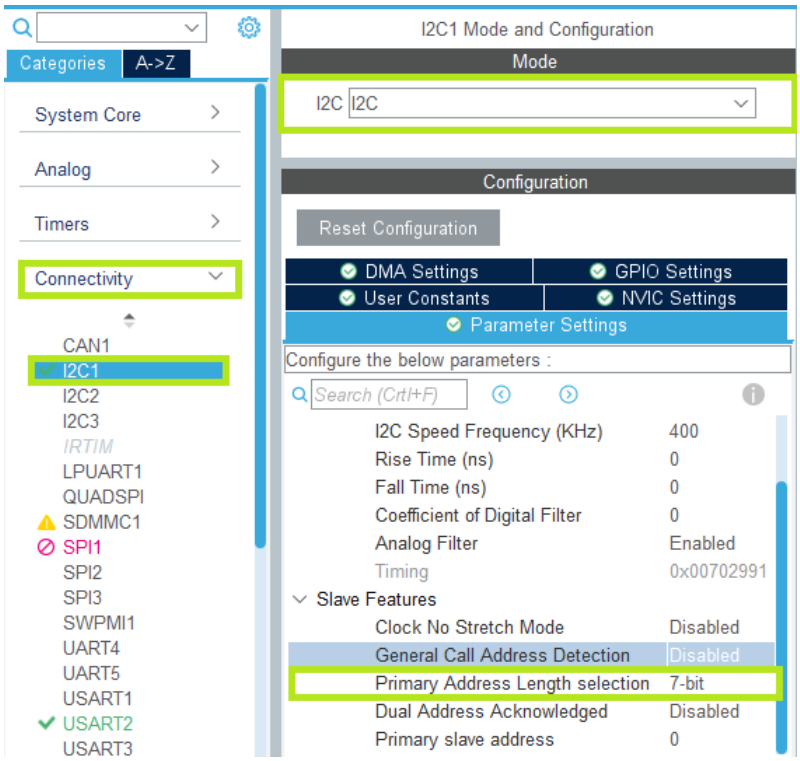
- Start: Bit que señala el inicio de la comunicación.
- Adress: La dirección del dispositivo al que se le enviará en mensaje, puede ser de 7 o 10 bits, en el ejercicio se usará una dirección de 7 bits.
- Read/Write: Bit que determina si se va a leer o escribir la información.
- ACK/NACK: Acknowledge y Not Acknoeledge, bits usados para determinar si se reconoció o no la información.
- Data: Información transmitida en el mensaje, despúes de enviarla se debe recibir el reconocimiento, si no se reconoce la transmisión no esta funcionando.
- Stop: Bit de parada, da por terminada la transmisón de datos.

La conexión entre dispositivos se hace de la siguiente manera:

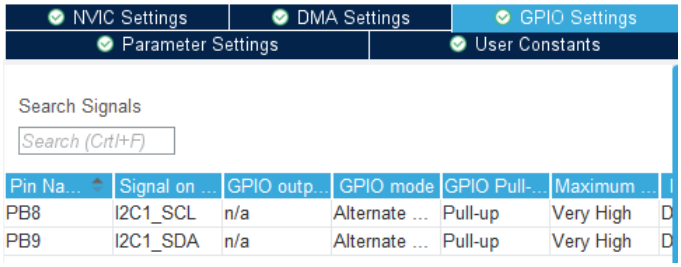


2.1.1 Configuración

Para configurar los puertos para usar la comunicación I2C desde el HAL se debe acceder a la interfaz y en Connectivity seleccionamos I2C1 (podría hacerse lo mismo con I2C2 e I2C3, son iguales pero con distintos pines), en el menú que se despliegue pondremos el modo en I2C y estableceremos en tamaño de la dirección en 7 bits.



En la pestaña de GPIO Settngs encontraremos los pines que se usarán para la comunicación (PB8 y PB9).



2.1.2 Librería SSD1306 y fuentes

Ya establecida la comunicación entre el microcontrolador y la pantalla procederemos a escribir en ella, sin embargo se debe tener en cuenta que esto se hace a través del envión de valores binarios a una posición específica de la pantalla, es decir, ordenamos que un píxel se vea blanco o negro, esto no es suficiente para escribir una letra así que se debe encender píxel por píxel o recurrir a una librería y una fuente.

En primer lugar estudiaremos la librería llamada SSD1306 esta contiene funciones que nos permiten enviar caracteres, hacer formas y más en la pantalla Oled, así nos podremos ahorrar escribir píxel por píxel, aunque hay muchas funciones explicaremos las más importantes:

- SSD1306_Init(): Inicializa la pantalla Oled, esta configuración fue hecha para esta pantalla y siempre debe hacerse cuando se vaya a usar. No hay ningún problema con usar esta versión ya hecha siempre y cuando sea para una pantalla de la misma refrencia.

```
uint8_t SSD1306_Init(void) {

        /* Init I2C */
        ssd1306_I2C_Init();

        /* Check if LCD connected to I2C */
        if (HAL_I2C_IsDeviceReady(&hi2c1, SSD1306_I2C_ADDR, 1, 20000) != HAL_OK) {
                /* Return false */
                return 0;
        }

        /* A little delay */
        uint32_t p = 2500;
        while(p>0)
                p--;

        /* Init LCD */
        SSD1306_WRITECOMMAND(0xAE); //display off
        SSD1306_WRITECOMMAND(0x20); //Set Memory Addressing Mode
        SSD1306_WRITECOMMAND(0x10); //00,Horizontal Addressing Mode;01,Vertical Addressing
             Mode;10,Page Addressing Mode (RESET);11,Invalid
        SSD1306_WRITECOMMAND(0xB0); //Set Page Start Address for Page Addressing Mode,0-7
        SSD1306_WRITECOMMAND(0xC8); //Set COM Output Scan Direction
        SSD1306_WRITECOMMAND(0x00); //---set low column address
        SSD1306_WRITECOMMAND(0x10); //---set high column address
        SSD1306_WRITECOMMAND(0x40); //--set start line address
        SSD1306_WRITECOMMAND(0x81); //--set contrast control register
        SSD1306_WRITECOMMAND(0xFF);
        SSD1306_WRITECOMMAND(0xA1); //--set segment re-map 0 to 127
        SSD1306_WRITECOMMAND(0xA6); //--set normal display
        SSD1306_WRITECOMMAND(0xA8); //--set multiplex ratio(1 to 64)
        SSD1306_WRITECOMMAND(0x3F); //
        SSD1306_WRITECOMMAND(0xA4); //0xa4,Output follows RAM content;0xa5,Output ignores RAM
              content
        SSD1306_WRITECOMMAND(0xD3); //-set display offset
        SSD1306_WRITECOMMAND(0x00); //-not offset
        SSD1306_WRITECOMMAND(0xD5); //--set display clock divide ratio/oscillator frequency
```

```
        SSD1306_WRITECOMMAND(0xF0); //--set divide ratio
        SSD1306_WRITECOMMAND(0xD9); //--set pre-charge period
        SSD1306_WRITECOMMAND(0x22); //
        SSD1306_WRITECOMMAND(0xDA); //--set com pins hardware configuration
        SSD1306_WRITECOMMAND(0x12);
        SSD1306_WRITECOMMAND(0xDB); //--set vcomh
        SSD1306_WRITECOMMAND(0x20); //0x20,0.77xVcc
        SSD1306_WRITECOMMAND(0x8D); //--set DC-DC enable
        SSD1306_WRITECOMMAND(0x14); //
        SSD1306_WRITECOMMAND(0xAF); //--turn on SSD1306 panel


        SSD1306_WRITECOMMAND(SSD1306_DEACTIVATE_SCROLL);

        /* Clear screen */
        SSD1306_Fill(SSD1306_COLOR_BLACK);

        /* Update screen */
        SSD1306_UpdateScreen();

        /* Set default values */
        SSD1306.CurrentX = 0;
        SSD1306.CurrentY = 0;

        /* Initialized OK */
        SSD1306.Initialized = 1;

        /* Return OK */
        return 1;
}
```

- SSD1306_GotoXY(x,y): Posiciona el puntero en unas coordenadas, a partir de donde se encuentre el puntero se comenzará a escribir.

```
void SSD1306_GotoXY(uint16_t x, uint16_t y) {
        /* Set write pointers */
        SSD1306.CurrentX = x;
        SSD1306.CurrentY = y;
}
```

- SSD1306_Puts("Mensaje", Fuente, color): Imprime un string en la pantalla, usando una fuente determinada, el color de la letra es blanco si color es 1 y es negro si color es 0, para imprimir hace uso de otra función que envía carácter por carácter.

```
char SSD1306_Puts(char* str, FontDef_t* Font, SSD1306_COLOR_t color) {
        /* Write characters */
        while (*str) {
                /* Write character by character */
                if (SSD1306_Putc(*str, Font, color) != *str) {
                        /* Return error */
                        return *str;
                }

                /* Increase string pointer */
                str++;
        }

        /* Everything OK, zero should be returned */
        return *str;
}
```

- SSD1306_Putc(char,Fuente, Color): Imprime un carácter en la pantalla, al recibir un carácter de entrada usa la infromación de la fuente y el puntero para poner en blanco los píxeles necesarios para escribir la letra.

```
char SSD1306_Putc(char ch, FontDef_t* Font, SSD1306_COLOR_t color) {
        uint32_t i, b, j;

        /* Check available space in LCD */
        if (
                SSD1306_WIDTH <= (SSD1306.CurrentX + Font->FontWidth) ||
                SSD1306_HEIGHT <= (SSD1306.CurrentY + Font->FontHeight)
```

```
        ) {
                /* Error */
                return 0;
        }

        /* Go through font */
        for (i = 0; i < Font->FontHeight; i++) {
                b = Font->data[(ch - 32) * Font->FontHeight + i];
                for (j = 0; j < Font->FontWidth; j++) {
                        if ((b << j) & 0x8000) {
                                SSD1306_DrawPixel(SSD1306.CurrentX + j, (SSD1306.CurrentY + i
                                    ), (SSD1306_COLOR_t) color);
                        } else {
                                SSD1306_DrawPixel(SSD1306.CurrentX + j, (SSD1306.CurrentY + i
                                    ), (SSD1306_COLOR_t)!color);
                        }
                }
        }

        /* Increase pointer */
        SSD1306.CurrentX += Font->FontWidth;

        /* Return character written */
        return ch;
}
```

- SSD1306_UpdateScreen(): Actualiza la pantalla para que se visualice el último mensaje recibido.

```
void SSD1306_UpdateScreen(void) {
        uint8_t m;

        for (m = 0; m < 8; m++) {
                SSD1306_WRITECOMMAND(0xB0 + m);
                SSD1306_WRITECOMMAND(0x00);
                SSD1306_WRITECOMMAND(0x10);

                /* Write multi data */
                ssd1306_I2C_WriteMulti(SSD1306_I2C_ADDR, 0x40, &SSD1306_Buffer[SSD1306_WIDTH
                    * m], SSD1306_WIDTH);
        }
}
```

3 Referencias

+ PM0214 Programming manual, STM32 Cortex-M4 MCUs and MPUs programming manual.
+ RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs.
+ STM32l476rg datasheet.
+Time-optimal Real-Time Test Case Generation using UPPAAL
+Código en Github: fsm_hal