Informe de Práctica Final: Conversor hexadecimal/decimal

Máximo García Aroca

June 5, 2024

1 Introducción

Proyecto creado junto con mi compañero Sergio Maximiliano.

Para este proyecto final se nos dejaba libre albedrío para diseñar alguna funcionalidad real simulado con nuestro microcontrolador STM-L47RG. Se nos dejaba la posibilidad de usar una placa FPGA de testeo con múltiples periféricos.

Hemos decidido usar la mayor cantidad de periféricos posible puesto que el profesor ha indicado que esta debe de ser la mayor prioridad. Así que aprovechando que el teclado matricial tiene simbología de base hexadecimal hemos decidido simular el funcionamiento de una calculadora que es capaz de hacer conversiones de números hexadecimales a decimales.

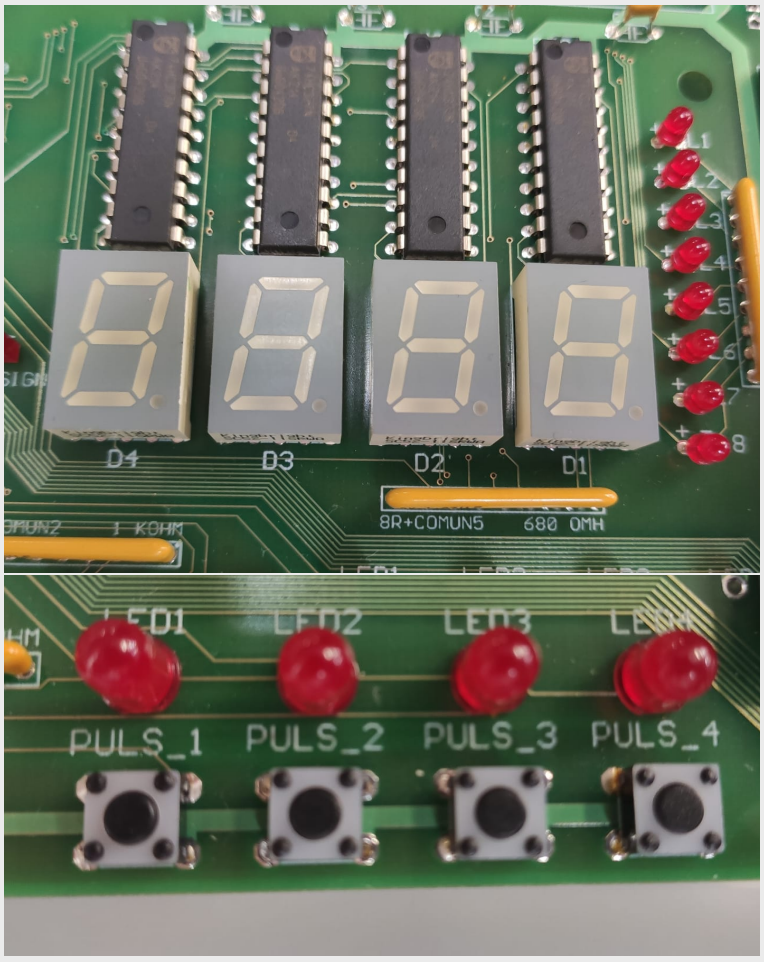
El funcionamiento de nuestro driver debería ser capaz de hacer esta conversión con éxito en estos pasos:

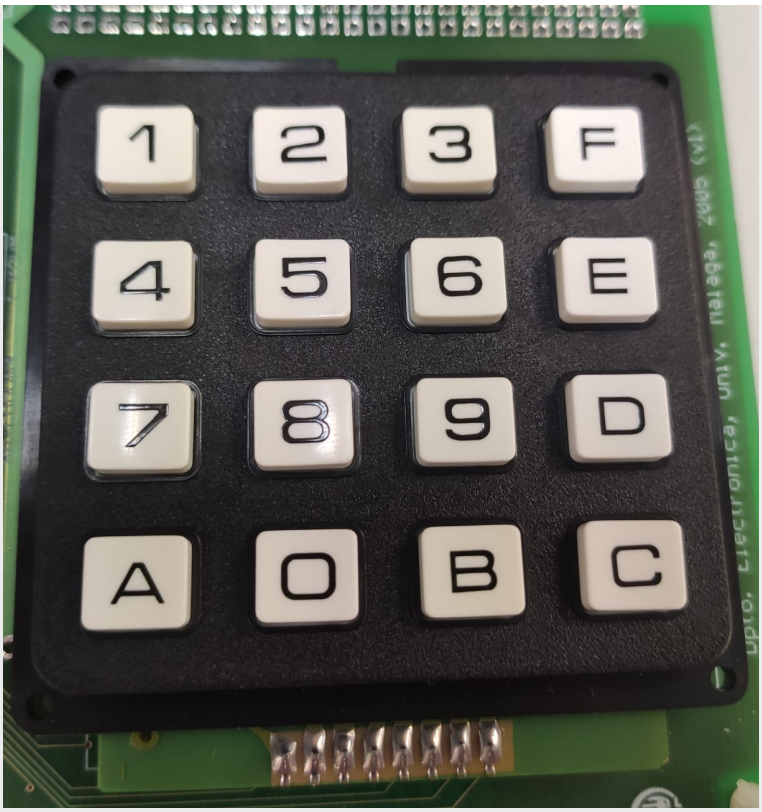
- La calculadora inicializará sus estructuras internas y quedará el display apagado esperando la introducción de dígitos.
- El usuario introducirá dígitos que se colocarán de izquierda a derecha como una calculadora.
- No hay límite de introducción de dígitos. El límite de procesamiento será UINT_MAX = 0xffffffff, equivalente a una conversión **4.294.967.295**. Si se introdujese un número superior (a partir de 0x100000000) se muestra un mensaje de error (Err).
- Cuando el usuario haya introducido el número que desea pulsará el botón de conversión que equivale a un bóton = en una calculadora convencional.
- La calculadora tiene un pulsador que equivaldría al botón de cancelar de una calculadora para resetear y empezar desde el principio. Este botón se usa tanto si se ha equivocado introduciendo las teclas como cuando haga la conversión y quiera introducir otro número posteriormente.
- El usuario podrá visualizar tanto el número introducido como la conversión en decimal en el display 7-segmentos. Este posee 4 displays por lo que para ver los números podrá hacer uso de otros dos pulsadores que simulan un cursor que se mueve hacia la izquierda o hacia la derecha para poder moverse entre los dígitos del número convertido y así poder ver el número en su totalidad.

2 Componentes y diseño físico

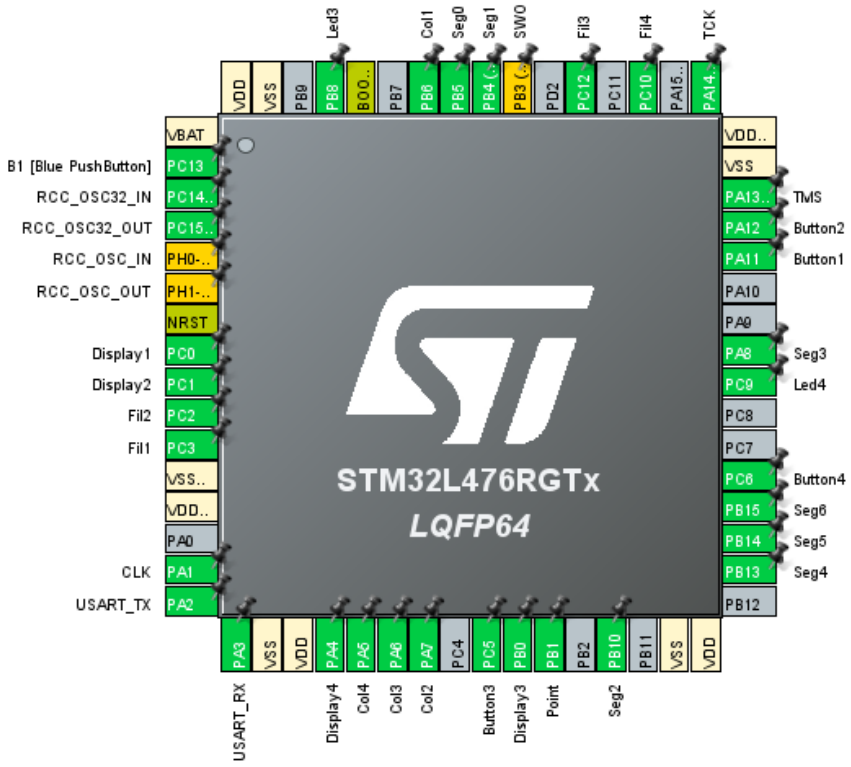
Como he dicho antes usamos la placa de testeo y como microcontrolador el STM-L47RG. De la placa de testeo hemos usado:

- 4 displays 7-segmentos: posee a su vez un punto para marcar los millares.
- 4 pulsadores: del 1 al 4 en orden para resetear, convertir, mover a la izquierda el cursor, mover a la derecha el cursor.
- 2 leds: (L3 y L4) para indizar el fin del número por la izquierda o por la derecha, encima de los pulsadores para que el usuario sepa que no puede mover más el cursor por el límite del número.
- Teclado matricial: Posee 16 teclas con los dígitos del sistema hexadecimal.



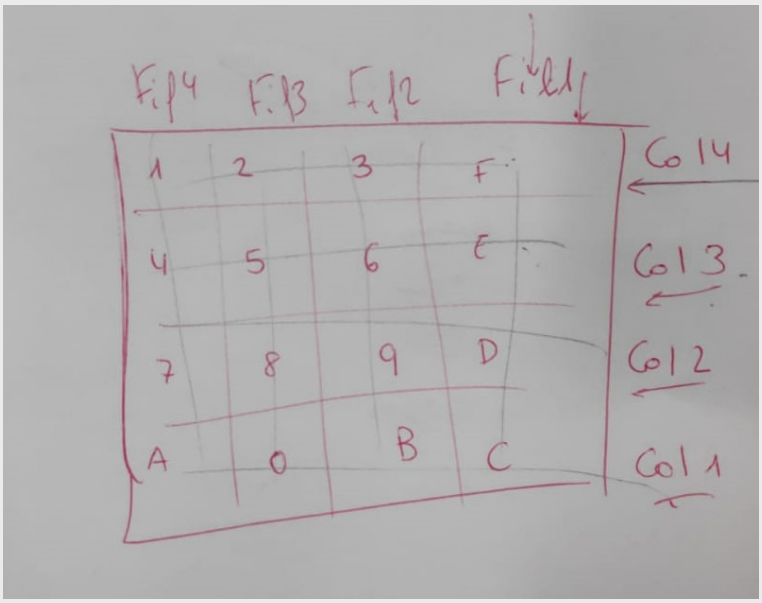


El microcontrolador tendrá el siguiente mapeo de pines para conectarse a la placa de testeo:


STM32L476RGTx
LQFP64
VBAT
PC13
PC14..
PC15..
PH0-..
PH1-..
NRST
PC0
PC1
PC2
PC3
VSS..
VDD..
PA0
PA1
PA2
VDD..
VSS
PA13..
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
PB12
PA3
VSS
VDD
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PB10
PB11
VSS
VDD
B1 [Blue PushButton]
RCC_OSC32_IN
RCC_OSC32_OUT
RCC_OSC_IN
RCC_OSC_OUT
Display1
Display2
Fil2
Fil1
CLK
USART_TX
Led3
Col1
Seg0
Seg1
SWO
Fil3
Fil4
TCK
TMS
Button2
Button1
Seg3
Led4
Button4
Seg6
Seg5
Seg4
USART_RX
Display4
Col4
Col3
Col2
Button3
Display3
Point
Seg2


same as Arduino CN6 header
same as Arduino CN8 header
same as Arduino CN5 header
same as Arduino CN9 header
CN7
CN10
USER_BUTTON

Para el teclado matricial se usan 8 conexiones, de la fila 1 a la 4 y de la columna 1 a la 4 con las conexiones $C19, A20, C20, A21, C21, A22, C22, A23$. Cabe destacar que la documentación está mal redactada. Las filas y columnas tienen un orden y un sentido distinto al que se pudiese esperar, donde las filas son de arriba a abajo y las columnas de izquierda a derecha. Esta es la verdadera distribución:



Los pines de las filas serán de salida y los de las columnas de entrada. El funcionamiento sería poner a 1 el pin de una fila y leer las columnas. De la que se lea un 1 indica que se ha pulsado al tecla y por tanto se puede saber por **polling** el valor que se debe introducir.

Para los pulsadores introducimos pines de entrada para leer y para los leds pines de salida para escribir en los leds e indicar el límite del cursor.

Y para el display 7-segmentos usamos la funcionalidad del registro que posee cada display para no tener que imprimir por *polling*. Solo hace falta introducir el número correspondiente al display, actualizando el reloj e introduciendo los valores de los segmentos del número correspondiente en el canal y el registro almacena el valor para poder poner dígito a dígito, no es necesario utilizar una técnica de multiplexión temporal para que sea visiblen los diferentes dígitos. Le hemos introducido el uso del punto para que sea más fácil a la hora de moverse con el cursor entender mejor cual es la conversión en número decimal. Lógicamente los pines son de salida puesto que hay que escribir en los displays, en el canal de los 7 segmnetos y en el pin del reloj (CLK).

3 Código y funcionamiento del driver

Describiré detalladamente el flujo de ejecución del programa paso por paso:

1. Primero se inicializa algunas de las variables creadas que servirán para la inicialización del *display* y la lectura de teclado. Todo esto está incluido en un bucle *while* infinito que reproduce la conversión tantas veces como se desee.

```
450     while (1) {
451
452         d = 4;
453         hexnum = 0x0;
454         decnum = 0;
455         multiplicador = 1;
456         columna = 0;
457         fila = 0;
458         boton_matricial = 0;
459         tam_array = 0;
460         index_digits = 0;
461         for(int i = 0; i < 10; ++i)
462             dec_digits[i] = -1;
463
464         HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_RESET);
465         HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_RESET);
466         HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_RESET);
467         HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_RESET);
468
469         HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_RESET);
470         HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_RESET);
471         HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_RESET);
472         HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_RESET);
473         HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_RESET);
474         HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_RESET);
475         HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_RESET);
476         HAL_GPIO_WritePin(GPIOB, Point_Pin, GPIO_PIN_RESET);
477
478         HAL_GPIO_WritePin(GPIOC, Led4_Pin, GPIO_PIN_RESET);
479         HAL_GPIO_WritePin(GPIOB, Led3_Pin, GPIO_PIN_RESET);
480
481
482         HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
483         HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
```

   Se inicializa un entero *"d"* que indica el *display* donde escribir el dígito. *hexnum* para almacenar el valor en hexadecimal. *decnum* para almacenar el número en decimal. *multiplicador* que se usará más tarde para la conversión. *columna* y *fila* que almacena la fila y columna que se registra tras pulsar un botón para la muestra por pantalla. *boton_matricial* como booleano para identificar si se sale del bucle correspondiente a la lectura del teclado por pulsar una tecla (1) o por pulsar el Pulsador 1 que corresponde con resetear (0). *tam_array* que indica el tamaño del array que almacena los dígitos del número para facilitar la funcionalidad del movimiento con cursor. *index_digits* como índice del array y el bucle con *dec_digits* para inicializar el array con los dígitos del número en decimal.

2. Tras inicializar las estructuras necesarias viene la lectura del teclado. Se hace mediante *polling* ya que el mostrar los dígitos en los *displays* se hace menos demandante gracias a los registros que posee cada uno de ellos. Se está en bucle mientras no se detecte la pulsación de ninguna tecla o del botón de RESET (PULS1), o del botón de conversión (PULS2).

```
485         // Lectura teclado matricial
486         while (1) {
487             HAL_Delay(500);
488             boton_matricial = 0;
489             while (HAL_GPIO_ReadPin(GPIOA, Button2_Pin) == 0) {
490                 if (HAL_GPIO_ReadPin(GPIOA, Button1_Pin) == 1)
491                     NVIC_SystemReset();
492 
493                 HAL_GPIO_WritePin(GPIOC, Fil2_Pin, GPIO_PIN_RESET);
494                 HAL_GPIO_WritePin(GPIOC, Fil3_Pin, GPIO_PIN_RESET);
495                 HAL_GPIO_WritePin(GPIOC, Fil4_Pin, GPIO_PIN_RESET);
496                 HAL_GPIO_WritePin(GPIOC, Fil1_Pin, GPIO_PIN_SET);
497 
498                 columna = lectura_columnas();
499                 HAL_Delay(50);
500                 if (columna != 0) {
501                     fila = 1;
502                     HAL_GPIO_WritePin(GPIOC, Fil1_Pin, GPIO_PIN_RESET);
503                     boton_matricial = 1;
504                     break;
505                 }
506 
507                 HAL_GPIO_WritePin(GPIOC, Fil1_Pin, GPIO_PIN_RESET);
508                 HAL_GPIO_WritePin(GPIOC, Fil3_Pin, GPIO_PIN_RESET);
509                 HAL_GPIO_WritePin(GPIOC, Fil4_Pin, GPIO_PIN_RESET);
510                 HAL_GPIO_WritePin(GPIOC, Fil2_Pin, GPIO_PIN_SET);
511 
512                 columna = lectura_columnas();
513                 HAL_Delay(50);
514                 if (columna != 0) {
515                     fila = 2;
516                     HAL_GPIO_WritePin(GPIOC, Fil2_Pin, GPIO_PIN_RESET);
517                     boton_matricial = 1;
518                     break;
519                 }
```

La pausa que se hace antes de cada lectura es clave para que una pulsación no cause un error por los rebotes y la velocidad del código en ejecutarse y que desemboque en la mala lectura de varias pulsaciones. Para esto se usa *HAL_Delay(500)*. Si se registra la pulsación del *Button2* significa que debe detenerse la lectura puesto que ya se quiere convertir el número ingresado. Si se registra una pulsación del *Button1* significa que debe detenerse la lectura puesto que se debe de resetear la placa; para esto se usó la función *NVIC_SystemReset();*. Si nada de esto ocurre se estará leyendo pulsaciones del teclado en bucle, escribiendo de la fila 1 a la 4 y leyendo las columnas. Si se obtiene una lectura de la columna con un 1 quiere decir que se pulsó la tecla, por lo que hay un pequeño *delay* para prevenir errores de pulsación, se almacena la fila y columna pulsada para poder distinguir la tecla. Se pone a 0 la entrada de la fila y se pone a 1 el booleano que indica que la detención de lectura es temporal por ser la pulsación de una tecla y no un botón.

Se usa la función *lectura_columnas()* para identificar la columna que se pone a 1. Si ninguna se lee a 1 significa que hay que leer en la siguiente fila.

```
64  int lectura_columnas() {
65      if (HAL_GPIO_ReadPin(GPIOB, Col1_Pin))
66          return (1);
67      else if (HAL_GPIO_ReadPin(GPIOA, Col2_Pin))
68          return (2);
69      else if (HAL_GPIO_ReadPin(GPIOA, Col3_Pin))
70          return (3);
71      else if (HAL_GPIO_ReadPin(GPIOA, Col4_Pin))
72          return (4);
73
74      return (0);
75  }
```

En el procesamiento de la lectura del teclado matricial se distinguen 2 casos. Uno en el que *boton_matricial* es 0 porque se pulsó el botón de conversión por lo que simplemente debe de salir del bucle infinito de lectura de teclado para proseguir con la conversión y otro caso en el que se se registra la pulsación de una tecla y se debe de proceder de dos formas ligeramente distintas. Como se quiere poder introducir números con más dígitos que *displays* disponibles se debe distinguir si estamos con menos de 5 dígitos o más puesto que si estamos con menos simplemente habrá que introducir el dígito al número hexadecimal, escribir en el *display* más a la izquierda que esté sin escribir previamente y registrar en los array de *hex_fil* y *hex_col* la fila y columna correspondiente del dígito para que sea más fácil mostrar y mover los números a posteriori; así como actualizar los valores de *d*, *tam_array*, *fila* y *columna*. Pero si estamos ante un caso con más de 4 dígitos tenemos que empezar a desplazar los dígitos que ya tenemos almacenados hacia la izquierda, del *display* 1 al 2, del 2 al 3, etc, para que haya hueco en el primer *display* para la nueva pulsación y se simule la escritura en una calculadora. Este es el código usado para el procesamiento descrito:

```
549             if (!boton_matricial) {
550                 break;
551             }
552             if (!d) { // Hay más digitos que displays en  el tester
553                 hexnum = hexnum * 0x10 + write_Display(1, fila, columna);
554                 HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
555                 HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
556                 for (int i = 2; i <= 4; i++) {
557                     write_Display(i, hex_fil[tam_array - 1 - (i - 2)],
558                                   hex_col[tam_array - 1 - (i - 2)]);
559                     HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
560                     HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
561                 }
562                 hex_fil[tam_array] = fila;
563                 hex_col[tam_array] = columna;
564                 fila = 0;
565                 columna = 0;
566                 tam_array++;
567
568             } else {
569                 hexnum = hexnum * 0x10 + write_Display(d, fila, columna);
570                 HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
571                 HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
572                 hex_fil[tam_array] = fila;
573                 hex_col[tam_array] = columna;
574                 fila = 0;
575                 columna = 0;
576                 d--;
577                 tam_array++;
578             }
```

Es importante la actualización del *display* donde se quiere mostrar un nuevo valor con la escritura en el reloj del 7-segmentos y también la puesta del valor correspondiente en el canal de 7 segmentos con la función que implementamos *write_Display(display, fila, columna);* al que se le pasa el *display* donde se quiere escribir, y la fila y columna de la tecla escrita que sirve para discernir el caso del número que tiene que representarse.

```
344  int write_Display(int display, int fila, int columna) {
345
346      switch (display) {
347      case (1):
348          HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
349          HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
350          HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
351          HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_RESET);
352          break;
353      case (2):
354          HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
355          HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
356          HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
357          HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_RESET);
358          break;
359      case (3):
360          HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
361          HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
362          HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
363          HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_RESET);
364          break;
365      case (4):
366          HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
367          HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
368          HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
369          HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_RESET);
370          break;
371      }
372
373      return (write_num_display(fila, columna));
374  }
```

```
177   int write_num_display(int fila, int columna) {
178       int num_hex = matriz_teclado[fila - 1][columna - 1];
179       switch (num_hex) {
180       case (0x0):
181           HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_SET);
182           HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_SET);
183           HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_SET);
184           HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_SET);
185           HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_SET);
186           HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_SET);
187           HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_RESET);
188           return (0x0);
189           break;
190       case (0x1):
191           HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_RESET);
192           HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_SET);
193           HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_SET);
194           HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_RESET);
195           HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_RESET);
196           HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_RESET);
197           HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_RESET);
198           return (0x1);
199           break;
200       case (0x2):
```

write_Display distingue el *display* donde escribir y llama a *write_num_display* que según la fila y columna que le llegue sabe qué escribir en los segmentos. Por esto consideramos importante usar arrays que almacenen la fila y columna de los dígitos ordenados descendentemente por los índices, para que así sea sencillo el uso de estas funciones y poder moverlos hacia los lados con una simple actualización y lectura de los arrays.

Para poder conseguir la conversión de un número de fila y columna al dígito hexadecimal introducido se usa la estructura constante previamente declarada *matriz_teclado* que almacena en el verdadero orden las teclas.

```
61  const int matriz_teclado[4][4] = { { 0xC, 0xD, 0xE, 0xF },
62          { 0xB, 0x9, 0x6, 0x3 }, { 0x0, 0x8, 0x5, 0x2 }, { 0xA, 0x7, 0x4, 0x1 } };
```

3. Por último viene la conversión del número. Primero se tiene que comprobar que el número es menor al máximo permitido o si no se muestra un error por pantalla ("Err").

```
598         } else { // Imprimir error
599             err = 1;
600             HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
601             HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
602             HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
603             HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_RESET);
604 
605             HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_SET);
606             HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_RESET);
607             HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_RESET);
608             HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_SET);
609             HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_SET);
610             HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_SET);
611             HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_SET);
612 
613             HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
614             HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
615 
616             HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
617             HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_RESET);
618             HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_RESET);
619             HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
620 
621             HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_RESET);
622             HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_RESET);
623             HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_RESET);
624             HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_RESET);
625             HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_SET);
626             HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_RESET);
627             HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_SET);
628 
629             HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
630             HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
631 
632             HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_RESET);
633             HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
634             HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
635             HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
636 
637             HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_RESET);
638             HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_RESET);
639             HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_RESET);
```

Si el número es válido para la conversión primero se realiza la conversión y se almacena en *decnum*. Del número completo se extraerá cada dígito y se almacenará en el array *dec_digits* para un uso más sencillo de ellos. Con *count_digits* se podrá conocer el número de dígitos que tiene el array para saber dónde están los límites y poder encender los leds.

```
581         if (hexnum <= INT_MAX) {
582             hexnum_pequenyito = (unsigned int) hexnum;
583             err = 0;
584             HAL_Delay(300);
585 
586             while (hexnum_pequenyito != 0) {
587                 int digito = hexnum_pequenyito % 16; // Obtener el dígito hexadecimal menos significativo
588                 decnum += digito * multiplicador; // Agregar el dígito al resultado
589                 multiplicador *= 16; // Actualizar el multiplicador para el siguiente dígito
590                 hexnum_pequenyito /= 16; // Ir al siguiente dígito hexadecimal
591             }
592 
593             tam_dec_digits = count_digits(decnum);
594             // Introduzco los dígitos en el arr;ay
595         for (int i = 0; i <= tam_dec_digits; i++)
596                 dec_digits[i] = (int) ((decnum / (unsigned int)pow(10, i)) % 10);
597 
598         } else { // Imprimir error
```

Por último lo que se hace es mostrar el número usando el array de antes y el *index_digits* que actúa de contador moviendose entre los valores 0 y *tam_dec_digits* para indicar desde el *display* 4 al 1 los valores del array (dígitos) que debe mostrar. Así se simula el movimiento, si se pulsa el botón de moverse a la izquierda, *index_digits* pasa a valer uno más y así muestra del dígito 1 al 4 en vez del 0 al 3.

```
649         leds = 0;
650         limit_right = 1;
651         limit_left = 0;
652         point = 0;
653         index_digits = 0;
654         while (!HAL_GPIO_ReadPin(GPIOA, Button1_Pin)) {
655             if ((HAL_GPIO_ReadPin(GPIOC, Button3_Pin)) && !limit_left) //led izquierda
656             {
657                 HAL_Delay(300);
658                 leds = 0;
659                 index_digits++;
660             }
661             if ((HAL_GPIO_ReadPin(GPIOC, Button4_Pin)) && !limit_right) //led derecha
662             {
663                 HAL_Delay(300);
664                 leds = 0;
665                 index_digits--;
666             }
667             if (!err && !leds) {
668                 leds = 1;
669                 // Imprimo dígitos
670                 for (int display = 1; display <= 4; ++display) {
671                     switch (display) {
672                     case (1):
673                         HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_RESET);
674                         HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
675                         HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
676                         HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
677                         break;
678                     case (2):
679                         HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
680                         HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_RESET);
681                         HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
682                         HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
683                         break;
684                     case (3):
685                         HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
686                         HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
687                         HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_RESET);
688                         HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_SET);
689                         break;
```

```
690                     case (4):
691                         HAL_GPIO_WritePin(GPIOC, Display1_Pin, GPIO_PIN_SET);
692                         HAL_GPIO_WritePin(GPIOC, Display2_Pin, GPIO_PIN_SET);
693                         HAL_GPIO_WritePin(GPIOB, Display3_Pin, GPIO_PIN_SET);
694                         HAL_GPIO_WritePin(GPIOA, Display4_Pin, GPIO_PIN_RESET);
695                         break;
696                     }
697                     if (index_digits % 3 == 0 && index_digits != 0 )
698                         point = 1;
699                     else
700                         point = 0;
701                     write_num_display_dec(dec_digits[index_digits++], point);
702                     HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_SET);
703                     HAL_GPIO_WritePin(GPIOA, CLK_Pin, GPIO_PIN_RESET);
704 
705                 }
706                 if (index_digits == 4) { // Limite derecha
707                     HAL_GPIO_WritePin(GPIOC, Led4_Pin, GPIO_PIN_SET);
708                     limit_right = 1;
709                 } else
710                 {
711                     limit_right = 0;
712                     HAL_GPIO_WritePin(GPIOC, Led4_Pin, GPIO_PIN_RESET);
713                 }
714                 if (index_digits == tam_dec_digits || tam_dec_digits <= 4) { // Limite izquierda
715                     HAL_GPIO_WritePin(GPIOB, Led3_Pin, GPIO_PIN_SET);
716                     limit_left = 1;
717                 } else
718                 {
719                     limit_left = 0;
720                     HAL_GPIO_WritePin(GPIOB, Led3_Pin, GPIO_PIN_RESET);
721                 }
722                 index_digits -= 4;
723                 // Mover digitos izquierda o derecha
724             }
725         }
```

En el bucle se detecta la pulsación del primer botón para salir y resetear la placa, así como la pulsación de los botones 3 y 4 para realizar el desplazamiento del cursor.

Esta vez como no se usa la matriz para imprimir números del sistema hexadecimal, usamos *write_num_display_dec* para pasándole un dígito pueda imprimir en el 7 segmentos el valor. Al final del bucle distinguimos los casos donde se alcanza el límite del array para evitar errores y que se enciendan los leds correspondientes.

Tanto en caso de conversión como de error el programa se queda en bucle hasta que se detecta la pulsación del primer botón que resetea los buffers y variables y empieza de nuevo el funcionamiento.

```
77  void write_num_display_dec(int digit, int point) {
78      switch (digit) {
79      case (0):
80          HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_SET);
81          HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_SET);
82          HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_SET);
83          HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_SET);
84          HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_SET);
85          HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_SET);
86          HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_RESET);
87          break;
88      case (1):
89          HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_RESET);
90          HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_SET);
91          HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_SET);
92          HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_RESET);
93          HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_RESET);
94          HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_RESET);
95          HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_RESET);
96          break;
97      case (2):
98          HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_SET);
99          HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_SET);
100         HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_RESET);
101         HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_SET);
102         HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_SET);
103         HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_RESET);
104         HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_SET);
105         break;
106     case (3):
107         HAL_GPIO_WritePin(GPIOB, Seg0_Pin, GPIO_PIN_SET);
108         HAL_GPIO_WritePin(GPIOB, Seg1_Pin, GPIO_PIN_SET);
109         HAL_GPIO_WritePin(GPIOB, Seg2_Pin, GPIO_PIN_SET);
110         HAL_GPIO_WritePin(GPIOA, Seg3_Pin, GPIO_PIN_SET);
111         HAL_GPIO_WritePin(GPIOB, Seg4_Pin, GPIO_PIN_RESET);
112         HAL_GPIO_WritePin(GPIOB, Seg5_Pin, GPIO_PIN_RESET);
113         HAL_GPIO_WritePin(GPIOB, Seg6_Pin, GPIO_PIN_SET);
114         break;
115     case (4):
```

```
376  int count_digits(unsigned int decnum)
377  {
378      int cnt = 0;
379      while (decnum != 0)
380      {
381          cnt++;
382          decnum /= 10;
383      }
384      return (cnt);
385  }
```

Dejo un ejemplo del uso del driver en vídeo:

`https://drive.google.com/file/d/1oZTbT1iXL4H4zyCZhdEx39H9vNH4oPZq/view?usp=drive_link`