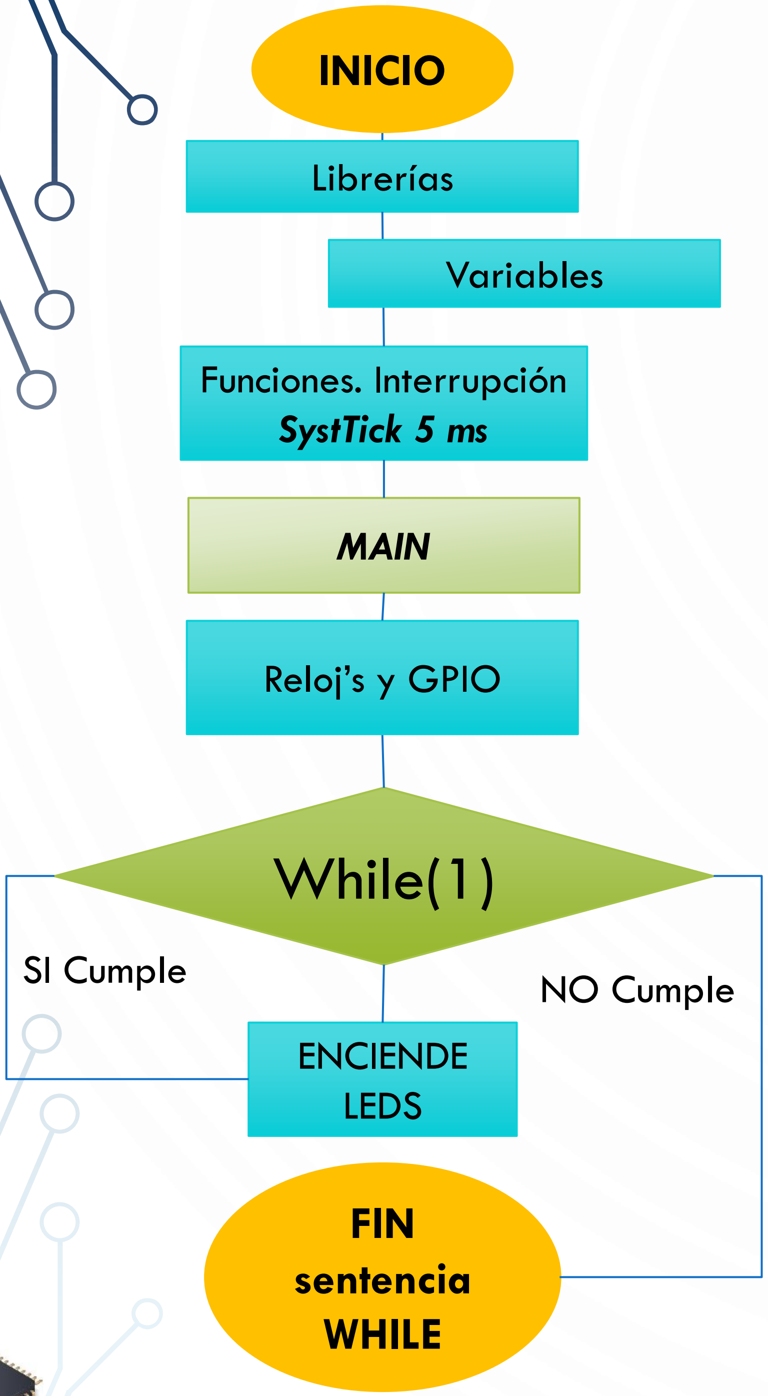
MICROS 32 BITS
STM – VD/GPIO

ROBINSON JIMENEZ MORENO

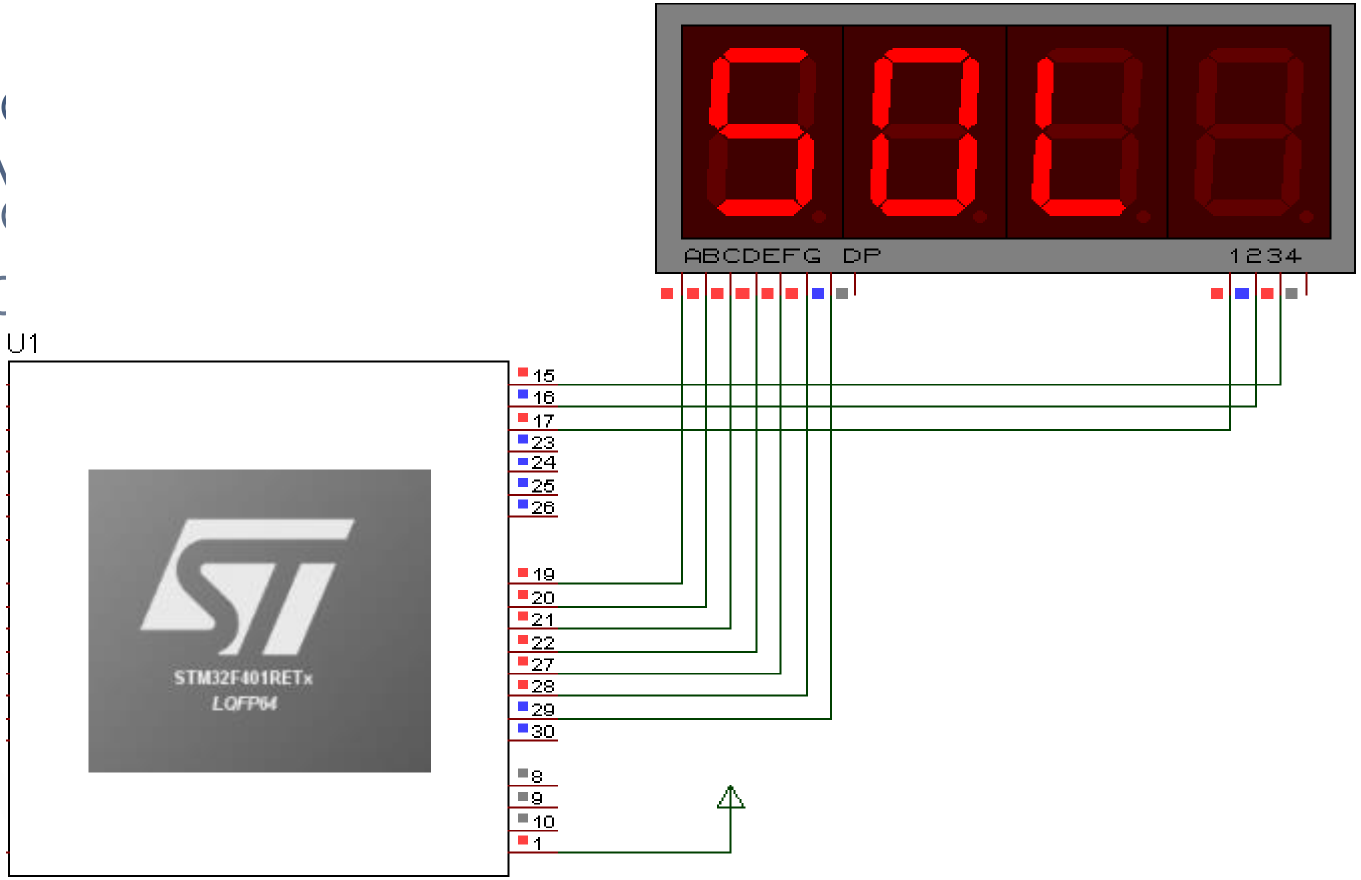
SCIENTIÆ · PATRIÆ · FAMILIÆ

UNIVERSIDAD MILITAR
NUEVA GRANADA

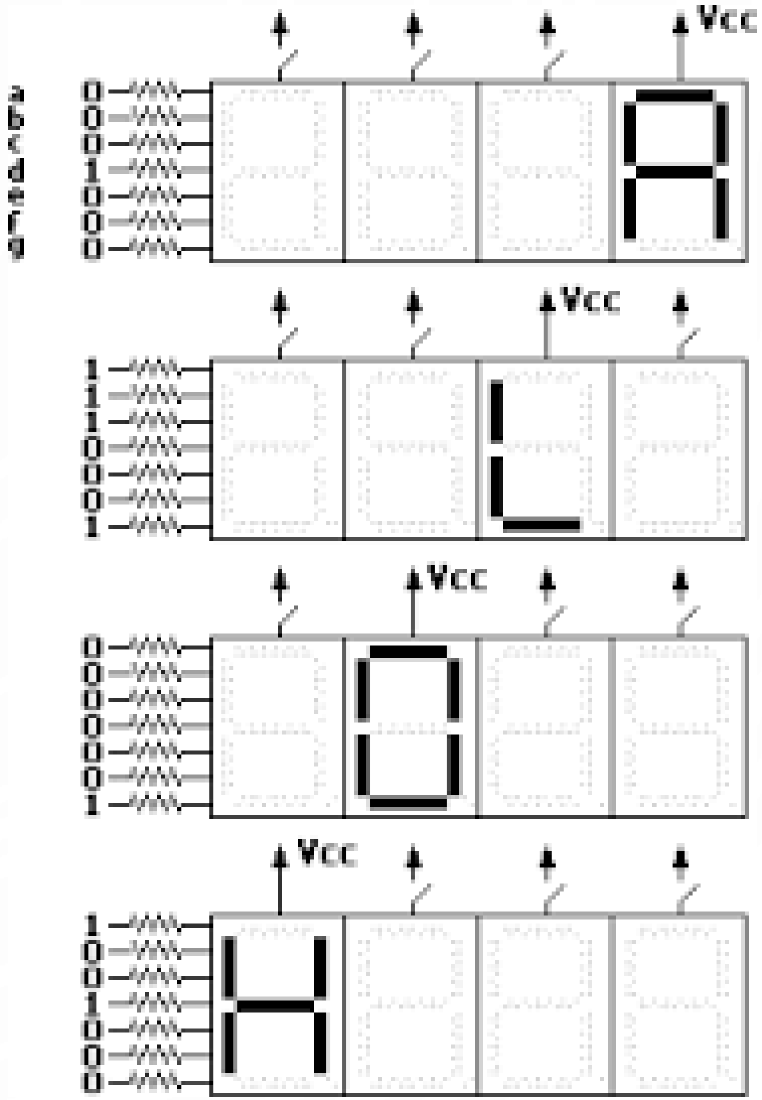
INGENIERÍA
MECATRÓNICA
UMNG




```c
#include <stm32f4xx.h>

char BCD [14] = {0XC0,0xF9,0XA4,0XB0,0X99,0X92,0X83,0XF8,0X80,0X98,0xBF,0x7F};

int main(void){
  //CONFIGURACION "CLOCK"
  RCC->AHB1ENR =0xF;
  //CONFIGURACION DE PINES
  GPIOB->MODER    = 0x55555555;
  GPIOB->ODR=0;

  while(true){          //bucle infinito
  for(int d=0;d<10;d++){ //FOR decenas
    for(int u=0;u<10;u++){ //FOR unidades
      for(int r=0;r<10;r++){ //FOR repeticion

        GPIOB->ODR=BCD[u];           GPIOB->ODR=(1UL<<7);
        for(long i=0;i<500000;i++);GPIOB->ODR=(0UL<<7);

        GPIOB->ODR=BCD[u];           GPIOB->ODR=(1UL<<8);
        for(long i=0;i<500000;i++);GPIOB->ODR=(1UL<<8);
} //FIN FOR 1
}
} }
}
```

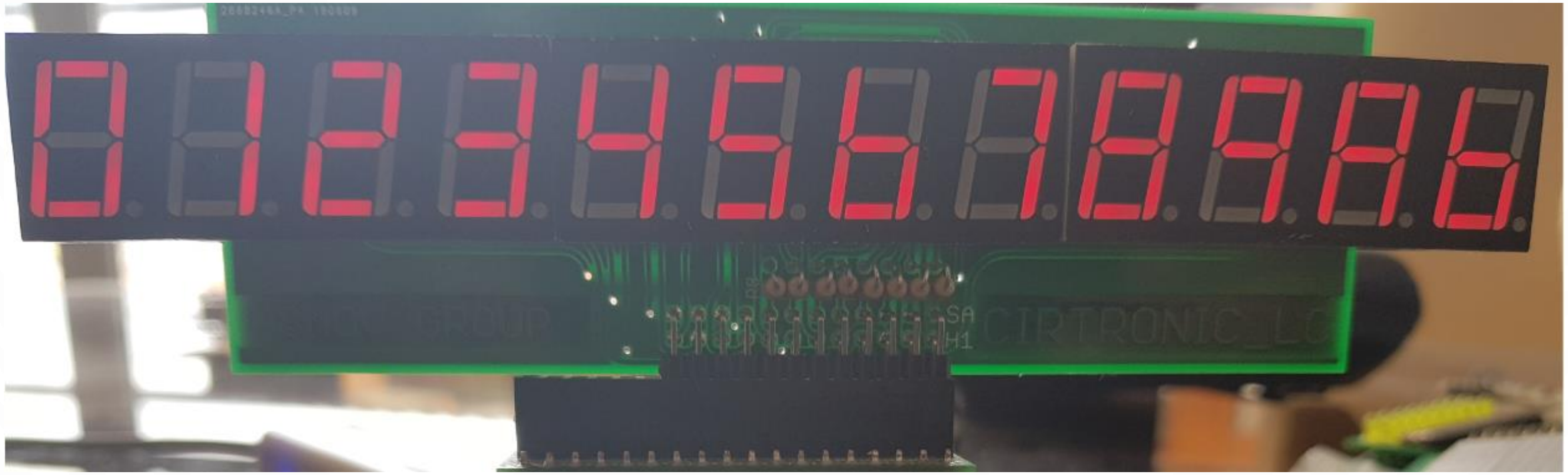


U1
STM32F401RETx
LQFP64
ABCDEFG DP
1234
15
16
17
23
24
25
26
19
20
21
22
27
28
29
30
8
9
10
1

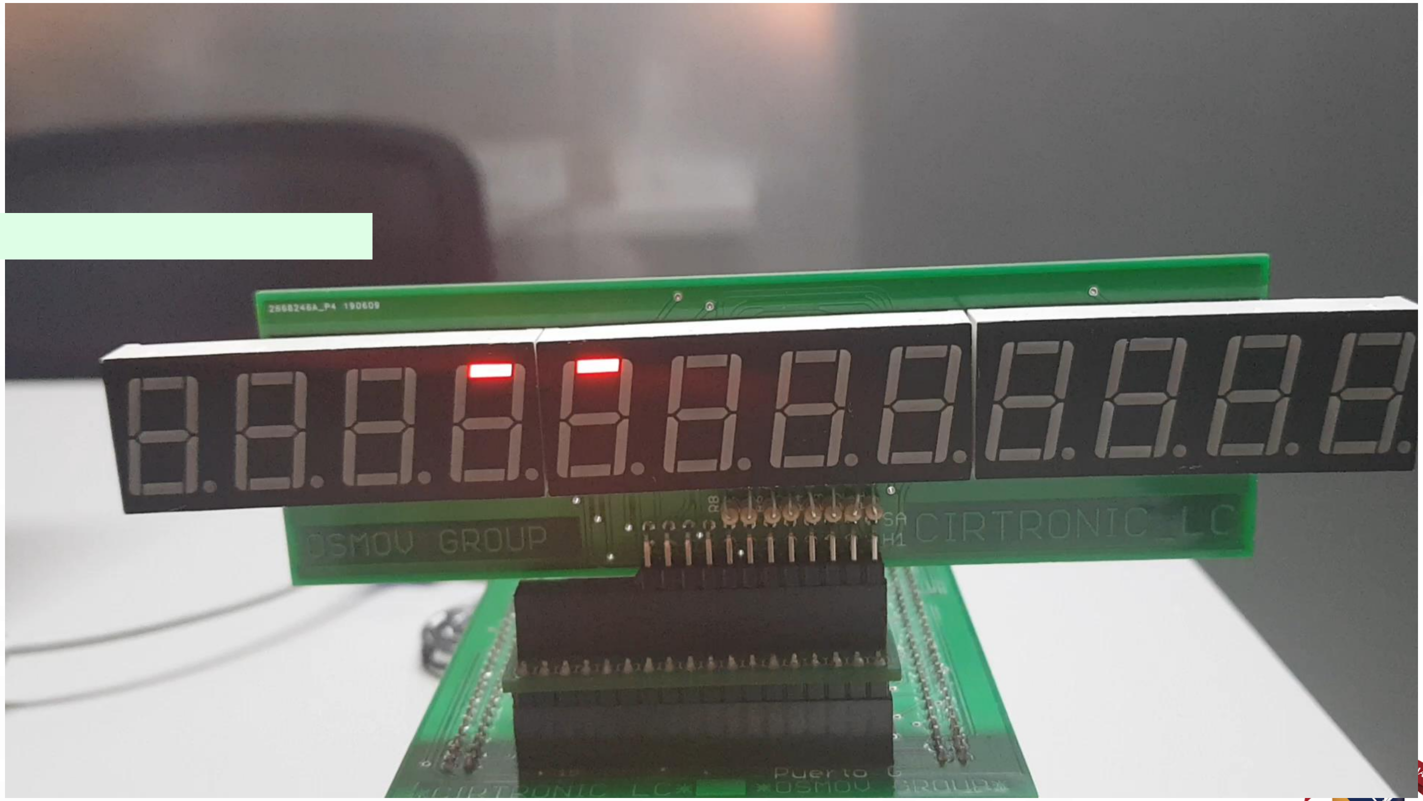


Vcc
a
b
c
d
e
f
g
0 0 0 1 0 0 0
1 1 1 0 0 0 1
0 0 0 0 0 0 1
1 0 0 1 0 0 0

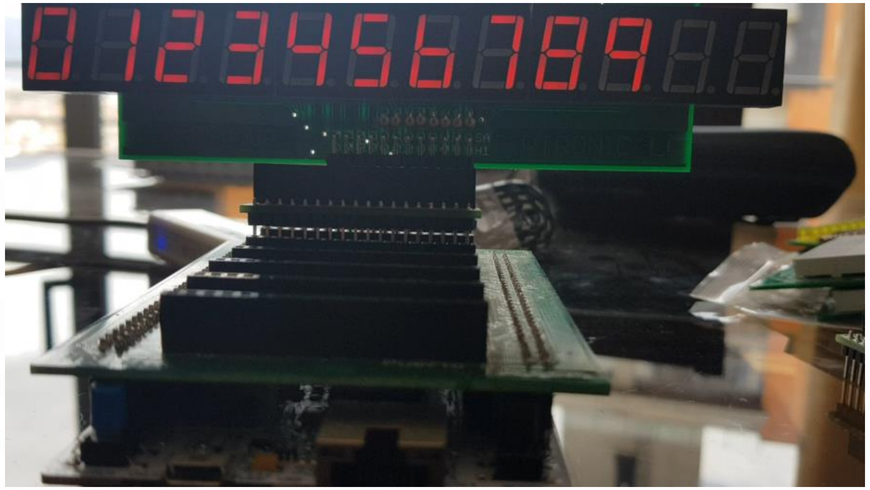
```c
///////////////VD numeros
#include "STM32F7xx.h"
char BCD [12] = {0XC0,0xF9,0XA4,0XB0,0X99,0X92,0X83,0XF8,0X80,0X98,0X88,0X83};

int main(void){
  RCC->AHB1ENR=0xFFFF;
  GPIOB->MODER=0x55555555;
  GPIOC->MODER=0x55555555;
  GPIOB->ODR=0X0;
  GPIOC->ODR=0X0;

  while(true){
for(int cont=0;cont<12;cont++){
    GPIOC->ODR=BCD[cont];
    GPIOB->ODR |= 1UL<<cont;
    for(int i=0;i<100;i++);
    GPIOB->ODR = 0;
} } }
```

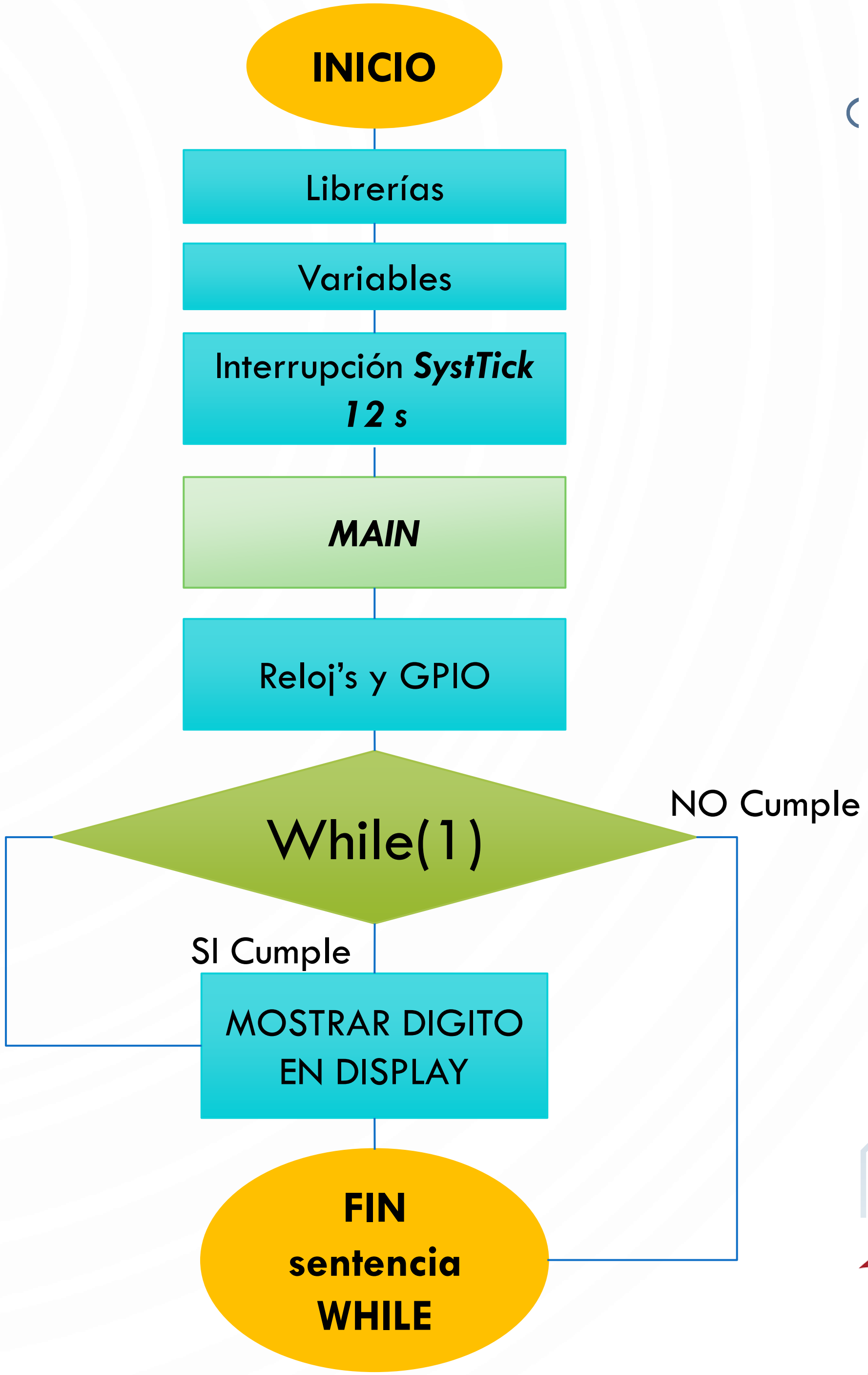


```c
#include "STM32F7xx.h"

int time=100000,cont=0;

char BCD [6] = {0XFE,0XFD,0XFB,0XF7,0XEF,0XDF};

int main(void)
{ int i=0;
  //CONFIGURACION "CLOCK"
  RCC->AHB1ENR =0xFFFF;
  //CONFIGURACION DE PINES
  GPIOF->MODER   = 0x55555555;
  GPIOF->PUPDR   = 0x55555555;
  GPIOG->MODER   = 0x55555555;
  GPIOF->ODR=2;
  GPIOG->ODR=0;

  while(true){         //bucle infinito
  for(cont=0;cont<6;cont++){ //FOR2
    GPIOG->ODR=BCD[cont];
  for(i=0;i<time;i++);
} //FIN FOR 1
}}
```

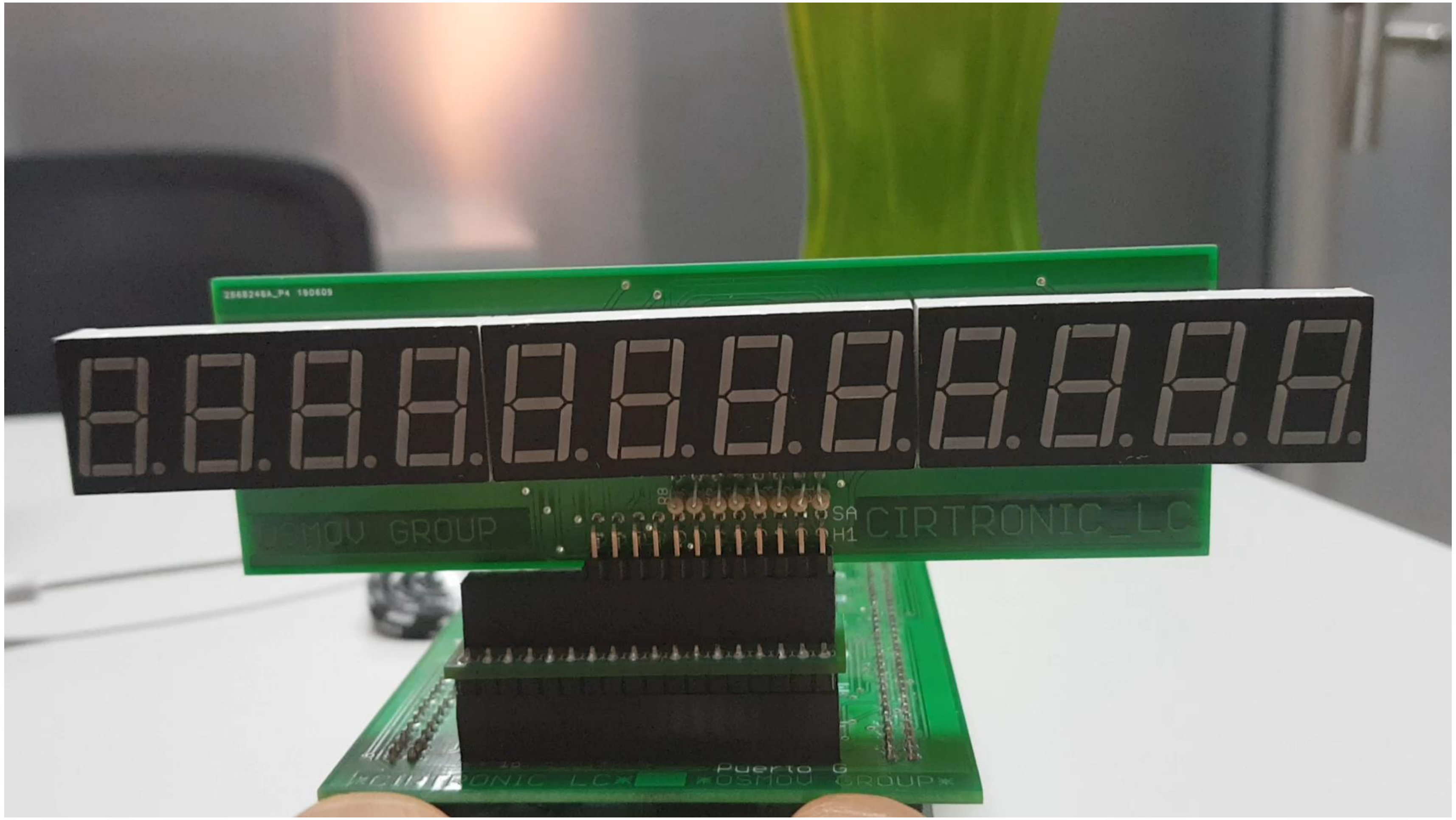


```
///////////////VD numeros desplazamiento
#include "STM32F7xx.h"
char BCD [14] = {0XC0,0xF9,0XA4,0XB0,0X99,0X92,
0X83,0XF8,0X80,0X98,0XFF,0XFF,0XFF,0XFF};

int main(void){
  RCC->AHB1ENR=0xFFFF;
  GPIOB->MODER=0x55555555;
  GPIOC->MODER=0x55555555;
  GPIOB->ODR=0X0;
  GPIOC->ODR=0X0;

  while(true){
for(int b=0;b<10;b++){
for(int a=0;a<10000;a++){
for(int cont=0;cont<10;cont++){
    GPIOC->ODR=BCD[cont+b];
    GPIOB->ODR |= 1UL<<cont;
    for(int i=0;i<100;i++);
    GPIOB->ODR = 0;
} } } } }
```

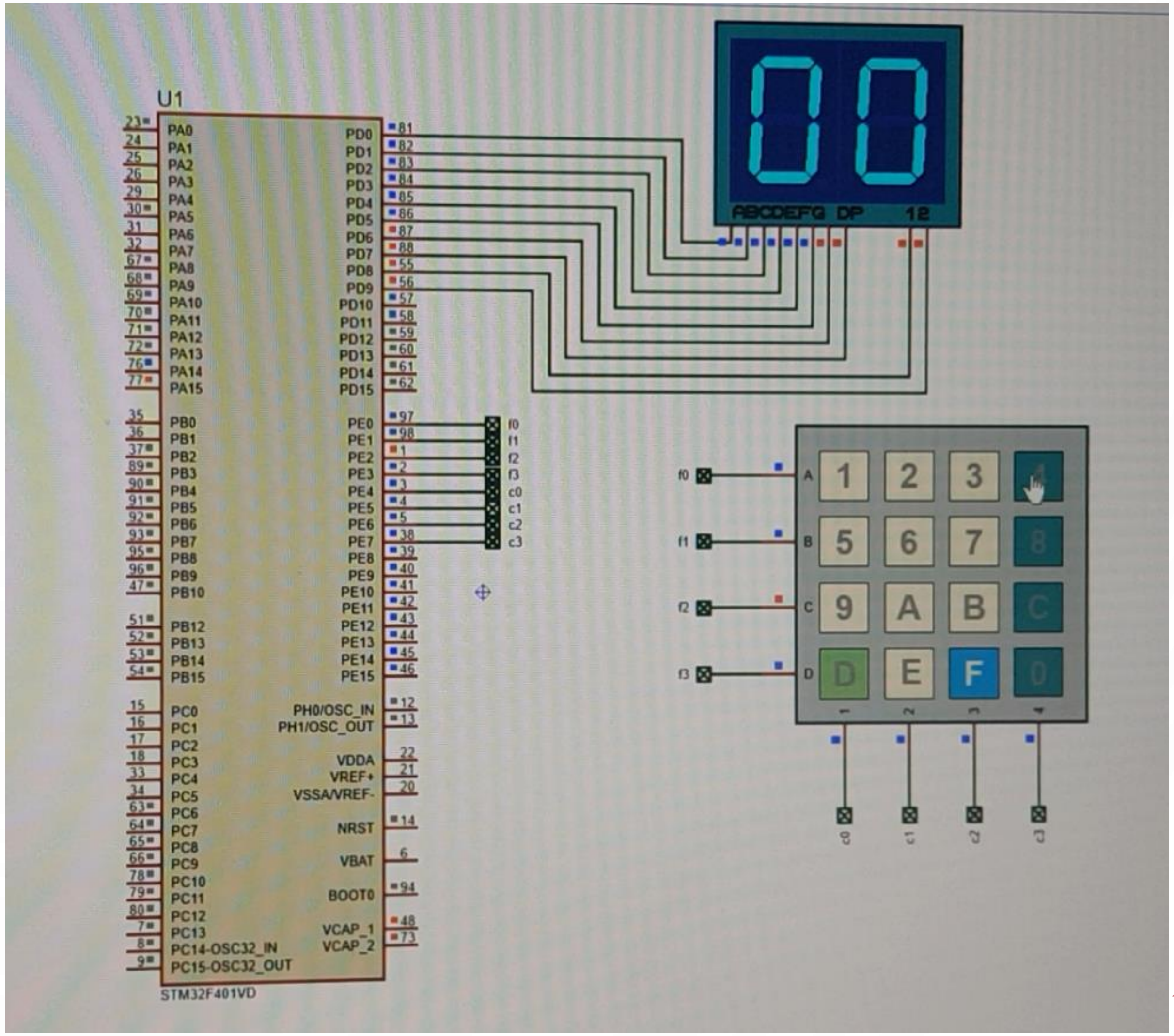




EJERCICIO DE CLASE



```c
#include "math.h"
#define TECLA GPIOE->IDR & 0x0f0
int cont,tc,a=0; int teclado();
int dat[2]={0,0};
int time=100000;
double hypo,l1,l2;
char BCD [14] = {0XC0,0xF9,0XA4,0XB0,0X99,0X92,0X83,0XF8,0X80,0X98,0xBF,0x7F};
void dinamica(int d, int u){
  GPIOD->ODR=BCD[u]|(1UL<<9);
      for(int i=0;i<time;i++);
      GPIOD->ODR=BCD[d]|(1UL<<8);
    for(int i=0;i<time;i++);
}
int teclado(void){
  while(true){
  int cl=0; int num=0;
  for(int f=0;f<4;f++){ GPIOE->ODR=(1UL<<f);
      cl=(GPIOE->IDR & 0x0f0)>>4;
      for(int c=0;c<4;c++){
      num++;
      if(cl==(1UL<<c)){return num;}
      } } }}
int main(void){
  RCC -> APB2ENR |=(1UL << 14);
  RCC->AHB1ENR=0x7F;
  GPIOD -> MODER = 0X00555555;
  GPIOE -> MODER = 0X00000055;
  GPIOE -> PUPDR = 0XAAAAAA00;
while(true){
    if(a>0)a=0; GPIOD->ODR=BCD[a]|(3UL<<8);
    dat[a]=teclado();GPIOD->ODR=BCD[dat[a]]|(1UL<<8);a++;
    for(int tp=0;tp<80000;tp++);while(TECLA){};
    dat[a]=teclado();GPIOD->ODR=BCD[dat[a]]|(1UL<<9);a++;
    for(int tp=0;tp<80000;tp++);while(TECLA){};
    l1=dat[0];if(l1>9)l1=0;
    l2=dat[1];if(l2>9)l2=0;
    hypo=sqrt(pow(l1,2)+pow(l2,2));
    for(cont=0;cont<25;cont++){
    tc=hypo; if(hypo<10){dinamica(0,tc);}
    else {dinamica(tc/10,tc%10);  }
  }
  } }
```



PUNTO FLOTANTE

Los números de coma flotante decimales normalmente se expresan en notación científica con un punto explícito siempre entre el primer y el segundo dígitos. El exponente o bien se escribe explícitamente incluyendo la base, o se usa una **e** para separarlo de la mantisa.

Mantisa	Exponente	Notación científica	Valor en punto fijo
1.5	4	$1.5 \cdot 10^{4}$	15000
-2.001	2	$-2.001 \cdot 10^{2}$	-200.1
5	-3	$5 \cdot 10^{-3}$	0.005
6.667	-11	6.667e-11	0.0000000000667

```
Data Type          Bits  Bytes    Value Range
-------------------------------------------------------------
signed int         16    2        -32768 to 32767
unsigned int       16    2        0 to 65535
signed long        32    4        -2147483648 to +2147483647
unsigned long      32    4        0 to 4294967295
float (IEEE-754)   32    4        +/-1.175494E-38 to +/-3.402823E+38
```

El estándar

Casi todo el hardware y lenguajes de programación utilizan números de punto flotante en los mismos formatos binarios, que están definidos en el estándar IEEE 754. Los formatos más comunes son de 32 o 64 bits de longitud total:

Formato	Bits totales	Bits significativos	Bits del exponente	Número más pequeño	Número más grande
Precisión sencilla	32	23 + 1 signo	8	$\sim 1.2 \cdot 10^{-38}$	$\sim 3.4 \cdot 10^{38}$
Precisión doble	64	52 + 1 signo	11	$\sim 5.0 \cdot 10^{-324}$	$\sim 1.8 \cdot 10^{308}$

Hay algunas peculiaridades:

- La *secuencia de bits* es primero el bit del signo, seguido del exponente y finalmente los bits significativos.
- El exponente no tiene signo; en su lugar se le resta un **desplazamiento** (127 para sencilla y 1023 para doble precisión). Esto, junto con la secuencia de bits, permite que los números de punto flotante se puedan comparar y ordenar correctamente incluso cuando se interpretan como enteros.
- Se asume que el bit más significativo de la mantisa es 1 y se omite, excepto para casos especiales.

http://puntoflotante.org/formats/fp/

PUNTO FLOTANTE

The ANSI/IEEE Std. 754 defines a set of formats for representing floating-point numbers. The main formats described in the original (1985) version of the specification are:

- 32-bit numbers – single-precision
- 64-bit numbers – double-precision

More recent versions of the specification add several further formats, including 16-bit (half-precision) which we'll look at in more detail later in this chapter.

Single-precision – Figure 2.1 shows how the 32 bits in single-precision format are used.

Figure 2.1: Single-precision floating-point number format

- Bit 31 is the sign bit (0 for a positive number, 1 for a negative number)
- Bits 30:23 are the exponent
- Bits 22:0 are the mantissa

PUNTO FLOTANTE

The 8-bit exponent field is used to store a value between −127 and 128, using offset binary. In other words, the value stored in the 8-bits has 127 subtracted from it. For example, an exponent value of 0 is stored as 0111 1111 (127).

The mantissa is formed from the 23 bits as a binary fraction. Floating-point numbers are normalized, so that there is only one non-zero digit to the left of the binary point. In other words, we always have an implicit binary "1." in front of the mantissa value.

The real value represented by the 32-bit binary data is therefore:

$(-1)^{\text{Sign}} \times 2^{\text{Exponent-127}} \times 1.\text{Mantissa}$

The IEEE specification uses certain bit patterns to represent some special cases:

- 0 is defined as each of the mantissa and exponent bits being zero.
- A group of very small "de-normalized" numbers is obtained by removing the requirement that the leading digit in the mantissa is a one. Denormal numbers are a special case. If you set the exponent bits to zero, you can represent very small numbers other than zero by setting mantissa bits. Because normal values have an implied leading 1, the closest value to zero you can represent

	Address+0	Address+1	Address+2	Address+3
Contents	SEEE EEEE	EMMM MMMM	MMMM MMMM	MMMM MMMM

Where

S represents the sign bit where 1 is negative and 0 is positive.

E is the exponent with an offset of 127.

M is the 24-bit mantissa (stored in 23 bits).

Zero is a special value denoted with an exponent field of 0 and a mantissa of 0.

Using the above format, the floating-point number -12.5 is stored as a hexadecimal value of 0xC1480000. In memory, this value appears as follows:

	Address+0	Address+1	Address+2	Address+3
Contents	0xC1	0x48	0x00	0x00

It is fairly simple to convert floating-point numbers to and from their hexadecimal storage equivalents. The following example demonstrates how this is done for the value -12.5 shown above.

The floating-point storage representation is not an intuitive format. To convert this to a floating-point number, the bits must be separated as specified in the floating-point number storage format table shown above. For example:

	Address+0	Address+1	Address+2	Address+3
Format	SEEEEEEE	EMMMMMMM	MMMMMMMM	MMMMMMMM
Binary	11000001	01001000	00000000	00000000
Hex	C1	48	00	00

- The sign bit is **1**, indicating a negative number.
- The exponent value is **10000010** binary or 130 decimal. Subtracting 127 from 130 leaves 3, which is the actual exponent.
- The mantissa appears as the following binary number:

```
10010000000000000000000
```

There is an understood binary point at the left of the mantissa that is always preceded by a **1**. This digit is omitted from the stored form of the floating-point number. Adding **1** and the binary point to the beginning of the mantissa gives the following value:

```
1.10010000000000000000000
```

To adjust the mantissa for the exponent, move the decimal point to the left for negative exponent values or right for positive exponent values. Since the exponent is three, the mantissa is adjusted as follows:

```
1100.10000000000000000000
```

The sum of these values is **12.5**. Because the sign bit was set, this number should be negative.

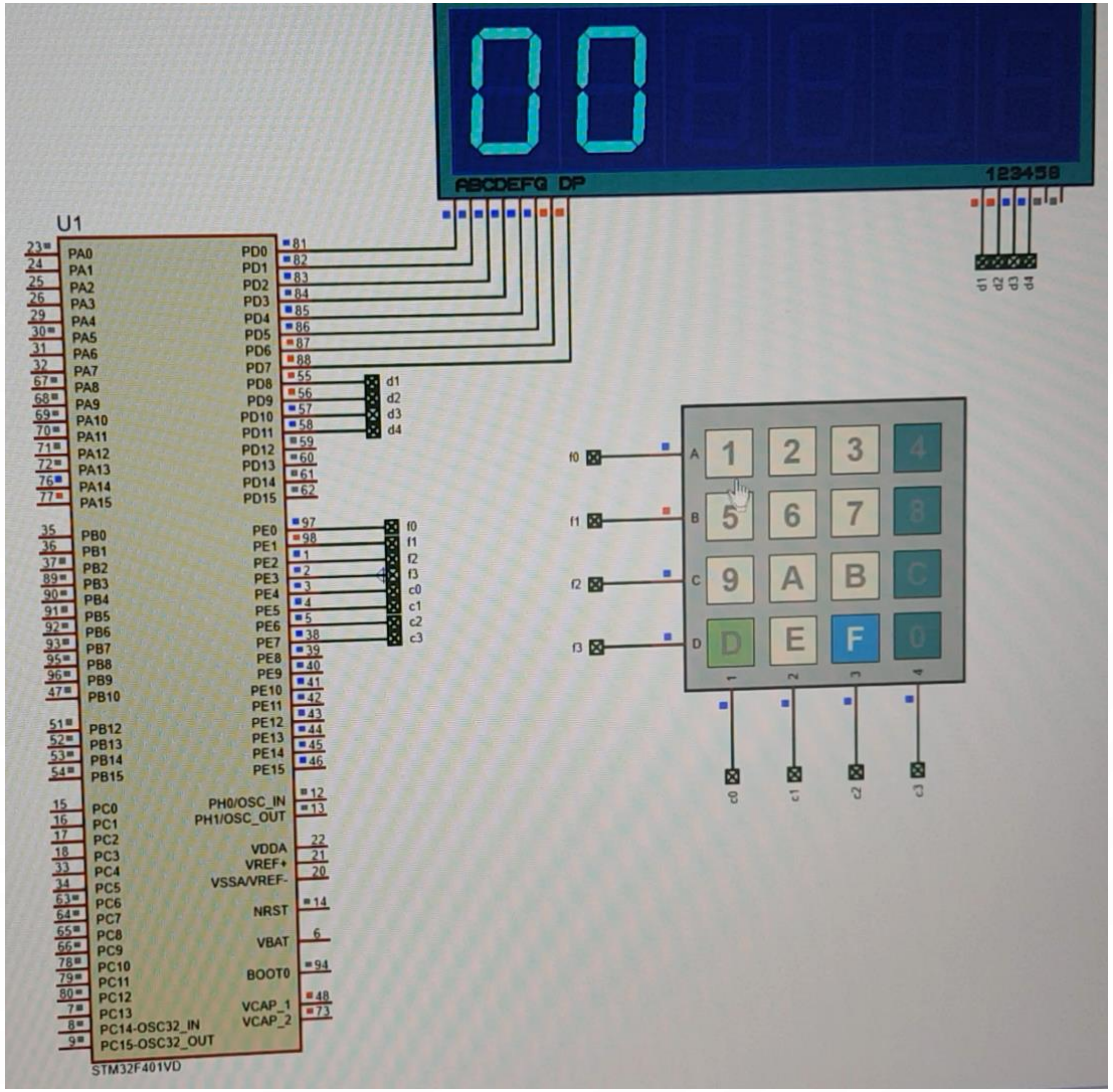
So, the hexadecimal value **0xC1480000** is **-12.5**.

https://www.keil.com/support/man/docs/c51/c51_ap_floatingpt.htm

PUNTO FLOTANTE

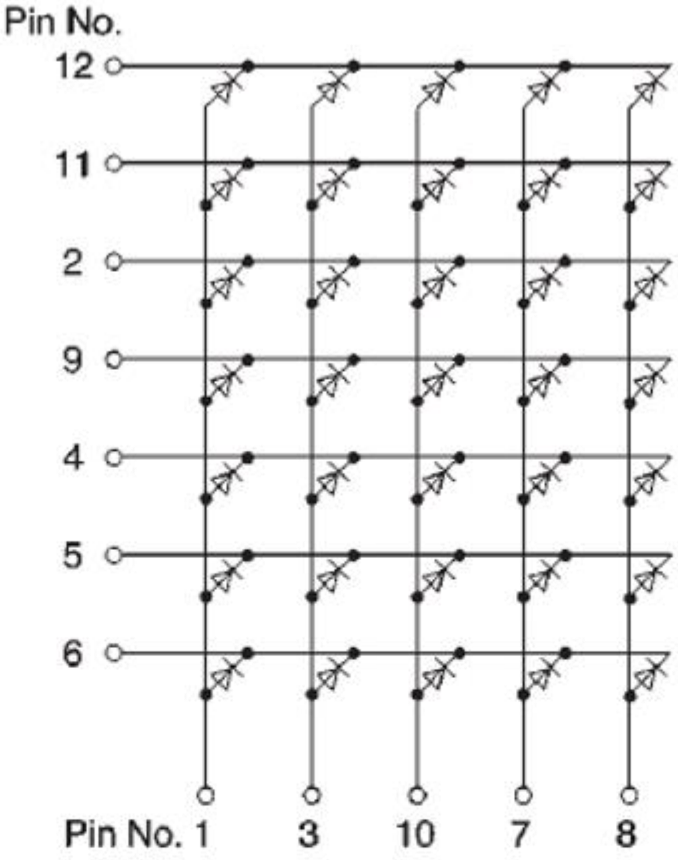
EJERCICIO: AJUSTAR PARA MOSTRAR LAS PRIMERAS 2 CIFRAS DECIMALES

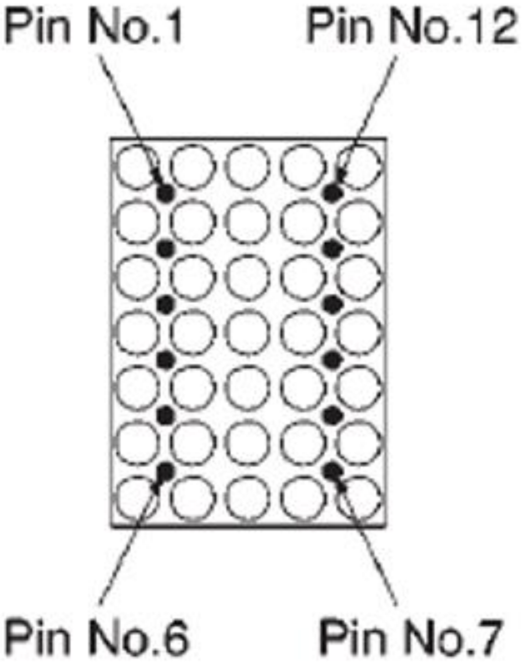
```c
#include "math.h"
#define TECLA GPIOE->IDR & 0x0f0
int cont,tc,a=0; int teclado();
int dat[2]={0,0};
int time=100000;
double hypo,l1,l2;
char BCD [14] = {0XC0,0xF9,0XA4,0XB0,0X99,0X92,0X83,0XF8,0X80,0X98,0xBF,0x7F};
void dinamica(int d, int u){
  GPIOD->ODR=BCD[u]|(1UL<<9);
      for(int i=0;i<time;i++);
      GPIOD->ODR=BCD[d]|(1UL<<8);
    for(int i=0;i<time;i++);
}
int teclado(void){
  while(true){
  int cl=0; int num=0;
  for(int f=0;f<4;f++){ GPIOE->ODR=(1UL<<f);
      cl=(GPIOE->IDR & 0x0f0)>>4;
      for(int c=0;c<4;c++){
      num++;
      if(cl==(1UL<<c)){return num;}
      } } }}
int main(void){
  RCC -> APB2ENR |=(1UL << 14);
  RCC->AHB1ENR=0x7F;
  GPIOD -> MODER = 0X00555555;
  GPIOE -> MODER = 0X00000055;
  GPIOE -> PUPDR = 0XAAAAAA00;
while(true){
    if(a>0)a=0; GPIOD->ODR=BCD[a]|(3UL<<8);
    dat[a]=teclado();GPIOD->ODR=BCD[dat[a]]|(1UL<<8);a++;
    for(int tp=0;tp<80000;tp++);while(TECLA){};
    dat[a]=teclado();GPIOD->ODR=BCD[dat[a]]|(1UL<<9);a++;
    for(int tp=0;tp<80000;tp++);while(TECLA){};
    l1=dat[0];if(l1>9)l1=0;
    l2=dat[1];if(l2>9)l2=0;
    hypo=sqrt(pow(l1,2)+pow(l2,2));
    for(cont=0;cont<25;cont++){
    tc=hypo; if(hypo<10){dinamica(0,tc);}
    else {dinamica(tc/10,tc%10);  }
  }
  } }
```



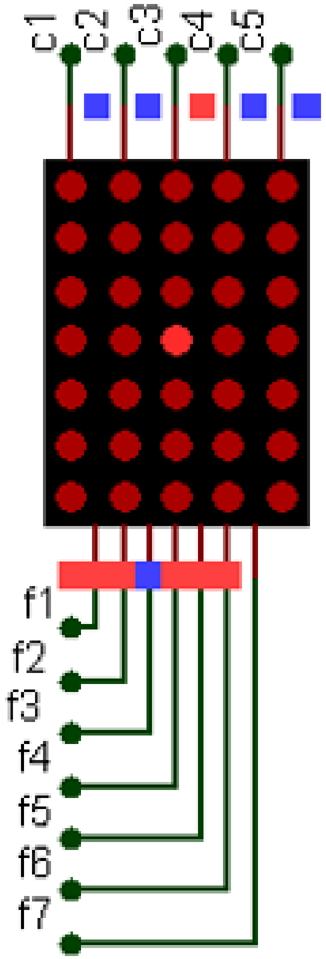
https://www.keil.com/support/man/docs/c51/c51_stdio_h.htm

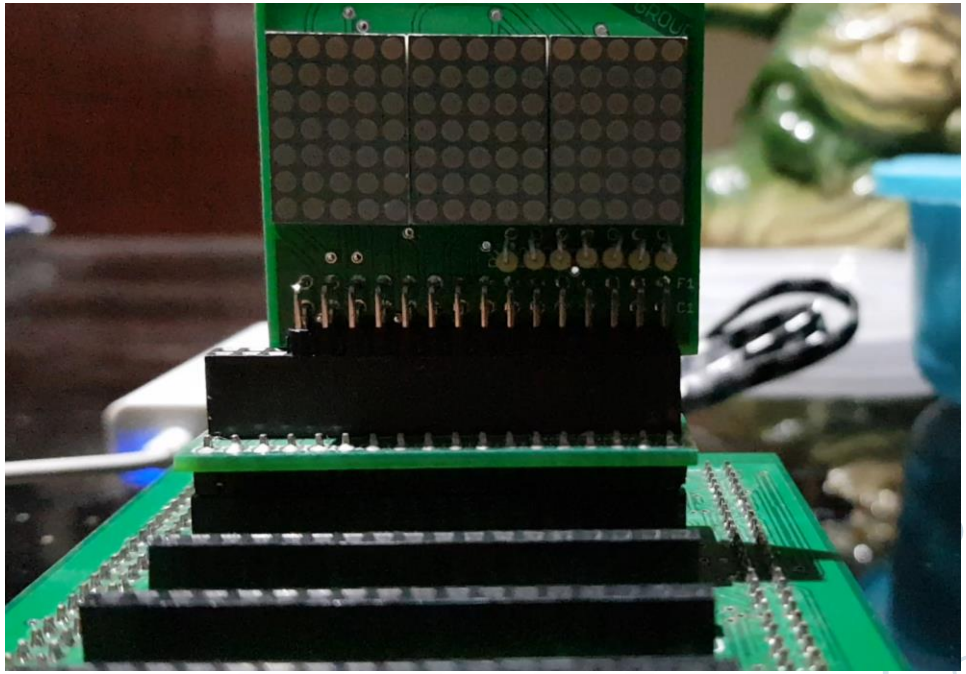
MANEJO DE MATRICES DE LED



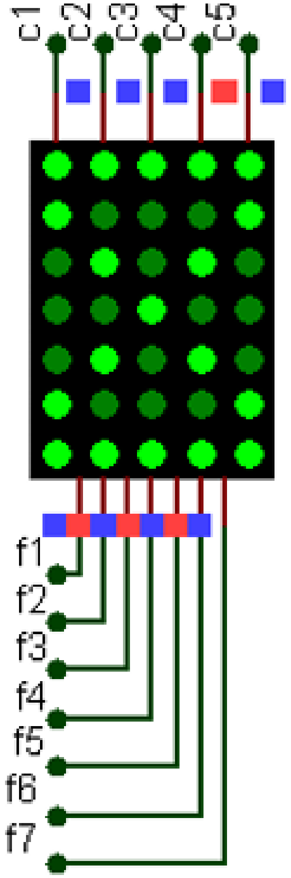


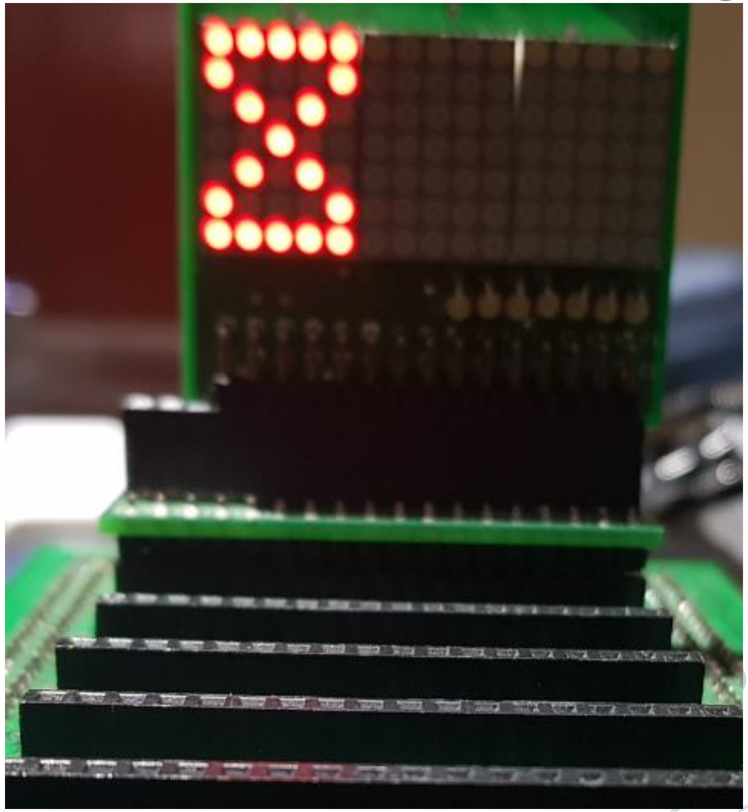
```c
#include "STM32F7xx.h"
int main(void){ int i=0;
  RCC->AHB1ENR |=0xF;
  GPIOB->MODER   = 0x55555555;
  GPIOC->MODER   = 0x55555555;
  while(true){
    for(int fil=0;fil<7;fil++){
    for(int col=0;col<5;col++){
    GPIOB->ODR=1UL<<col;
    GPIOC->ODR  = ~(1UL<<fil);
    for(int i=0;i<500000;i++);
      }
  }
}}
```



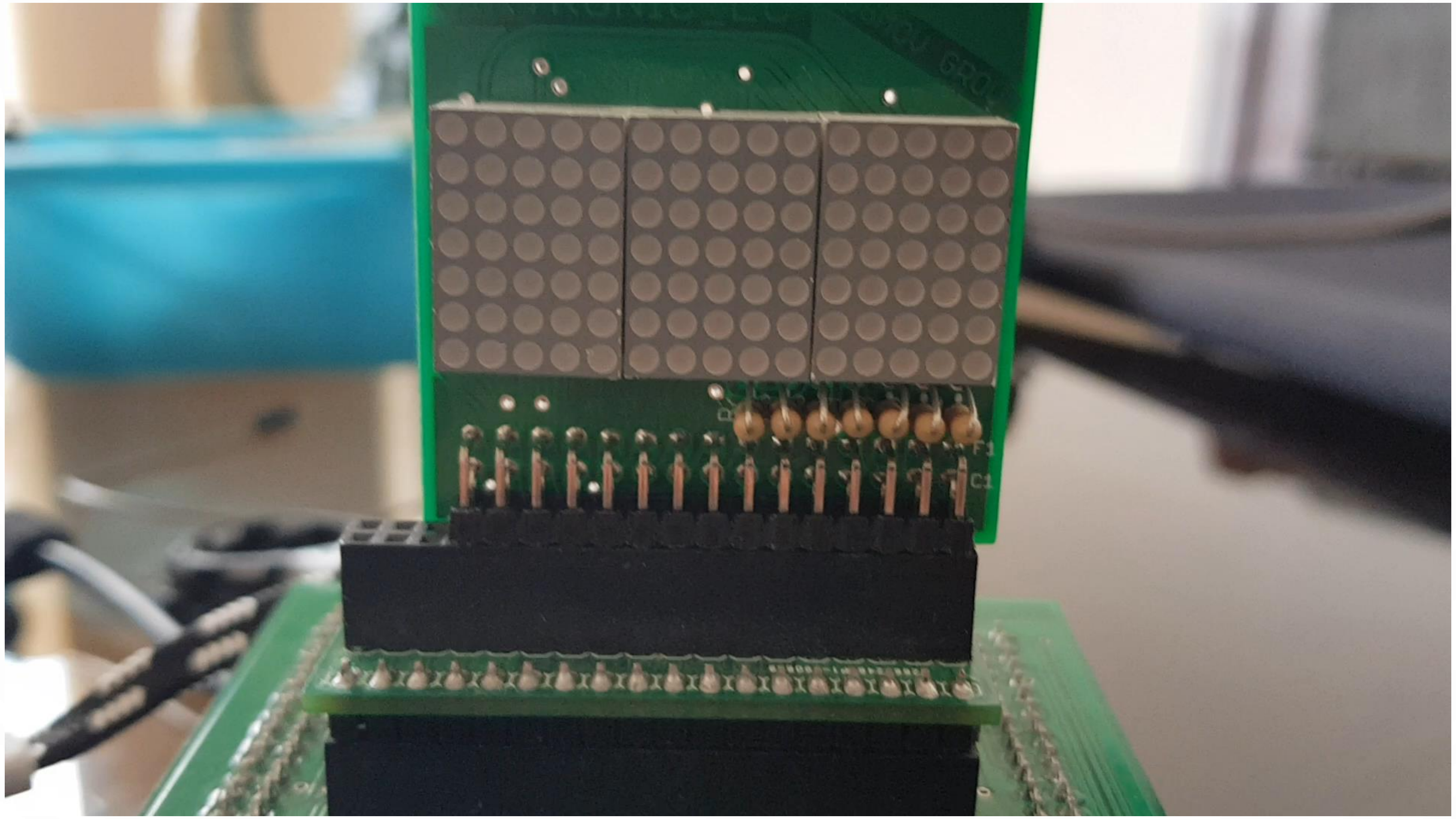


```c
//////////////////////////reloj arena/////

#include "STM32F7xx.h"
int rel[5]={0x63,0x55,0x49,0x55,0x63};
int main(void){ int i=0;
  RCC->AHB1ENR |=0xF;
  GPIOB->MODER   = 0x55555555;
  GPIOC->MODER   = 0x55555555;
  while(true){
    for(int col=0;col<5;col++){
    GPIOB->ODR= (1UL<<col);
    GPIOC->ODR  = ~(rel[col]);
    for(int i=0;i<10000;i++);
      }
}}
```

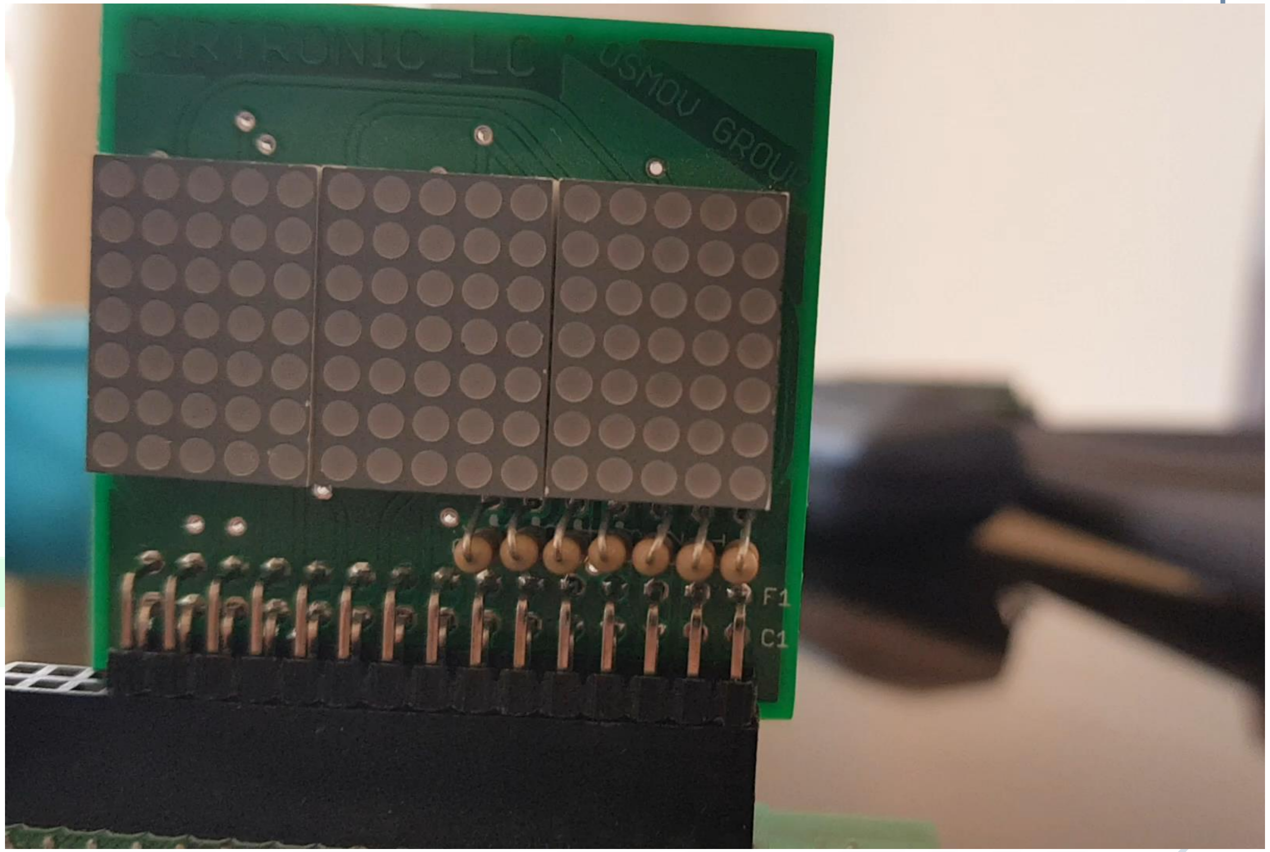


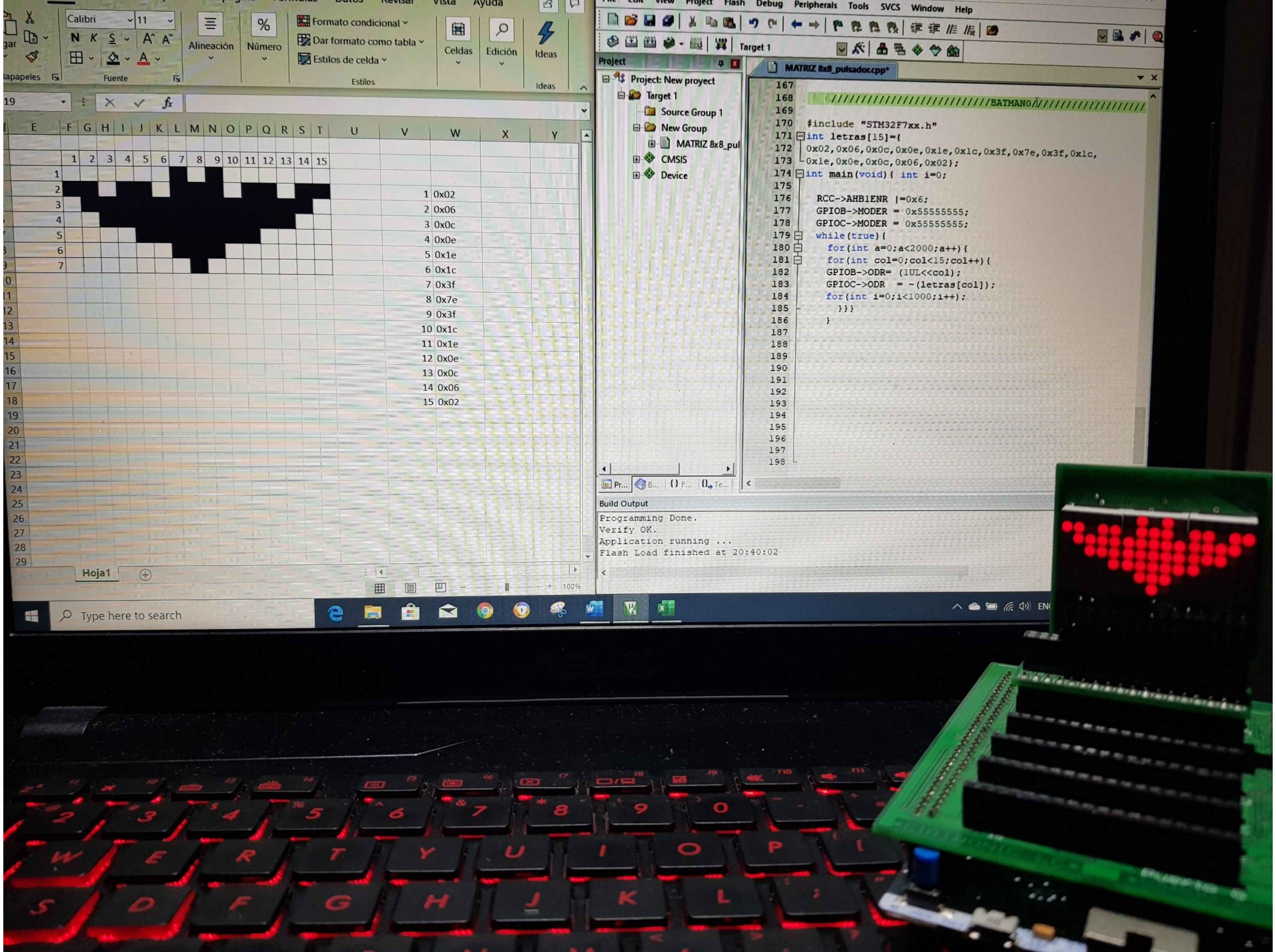


EJERCICIO: DIBUJAR EL RELOJ DE ARENA ANTERIOR EN LAS TRES MATRICES DE LED CON EFECTO DE LLENADO DE ARRIBA HACIA ABAJO



```
///////////////////////////////letras 2/////

#include "STM32F7xx.h"
int letras[5][5]={
{0x7e,0x11,0x11,0x11,0x7e},
{0xff,0x49,0x49,0x49,0x36},
{0x3e,0x41,0x41,0x41,0x0},
{0xff,0x41,0x41,0x22,0x1c},
{0xff,0x49,0x49,0x49,0x00}
};
int main(void){ int i=0;

  RCC->AHB1ENR |=0x6;
  GPIOB->MODER = 0x55555555;
  GPIOC->MODER |= 0x55555555;
  while(true){
    for(int fil=0;fil<5;fil++){
    for(int a=0;a<2000;a++){
    for(int col=0;col<5;col++){
    GPIOB->ODR= (1UL<<col);
    GPIOC->ODR  = ~(letras[fil][col]);
    for(int i=0;i<1000;i++);
      }}}
    }}
```





INGENIERÍA MECATRÓNICA UMNG

UNIVERSIDAD MILITAR NUEVA GRANADA

Tarea: Generar 2 animaciones en tres matrices de led 5x7, una de estas debe ser la ilustrada en el video, conmute de animación por un pulsador.