ALTERNATIVAS ALGORÍTMICAS DE SOFTWARE

Algunas características

- Se basa en un Núcleo ARM Cortex-M4F, con una velocidad de reloj máxima de72 Mhz
- 16KB de RAM de propósito general
- 64 KB de memoria Flash
- Operación a 3,3V pero puede ser alimentada a 5V
- Conectividad de pines similar a las de un Arduino nano.

http://www.st.com/stm32nucleo

https://www.st.com/en/evaluation-tools/nucleo-f303k8.html

Arduino vs STM32

"Todo depende de la aplicación las especificaciones y el proyecto que deseamos desempeñar"

> "Arduino es una plataforma 'open-source' basada en hardware y software fácil de usar (easy-to-use). Está dirigido para artistas, diseñadores, aficionados y cualquier persona que realiza proyectos interactivos".

> Por otro lado, elegir un microcontrolador caro, con un montón de periféricos adicionales que no se utilizarán, simplemente porque tiene uno que realmente se necesita no es una elección sabia.

> Los STM32 son bastante versátiles y altamente configurables, algo bueno y malo debido a que los hace un poco mas complicados de inicializar.

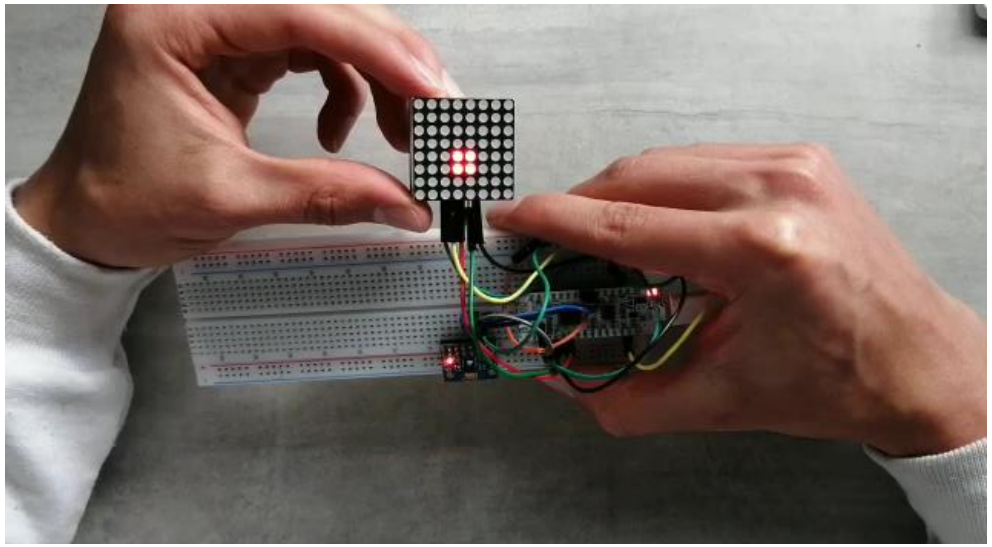
> Las variantes y subfamilias de los microcontroladores y su compatibilidad es algo que también se debe tener en cuenta, para realizar migraciones e proyectos o extensiones.

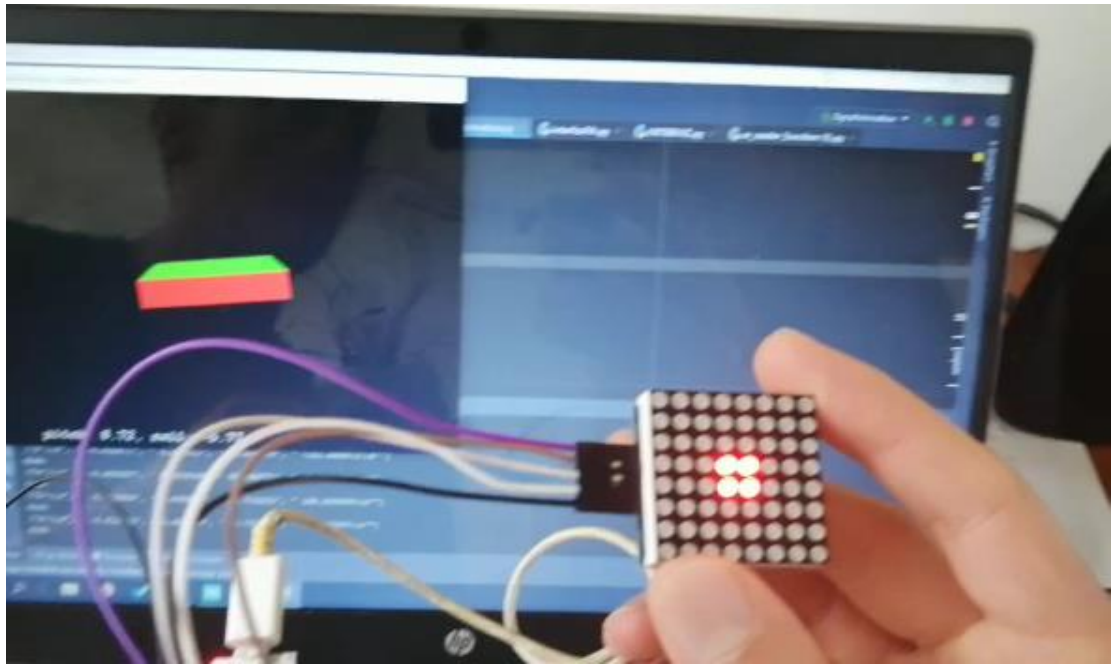
Arduino vs STM32

"Todo depende de la aplicación las especificaciones y el proyecto que deseamos desempeñar"

- Resolución de 10 bits de ADC
- Pasos de 4,9 mV
- PWM de máximo de 8 bits
- Arduino nano/Mega 16Mhz
- Dos pines de interrupción externos

STM32F303K8





ALTERNATIVAS ALGORÍTMICAS DE SOFTWARE

Algunas consideraciones

- Es un lenguaje de programación creado por Brian Kernighan y Dennis Ritchie a mediados de los años 70
- Proporciona una gran flexibilidad de programación y una baja comprobación de errores dejando bajo responsabilidad del programador la mayoría de las acciones y correcciones.

- ✓ No se comprueba el índice de referencia de un vector.
- ✓ No se revisa que no se accedan a memorias que no pertenecen al área de dato del programa.
- ✓ No es rígido en la comprobación del tipo de datos.

```
float a;    /*Declaro una variable para numeros reales*/
int b;      /*Declaro otra variable para numero enteros*/
b=a;        /*Asigno a la variable para entera el numero real*/
```

ANSI C

Estándar de unificación de lenguaje

auto	break	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	sizeof	static	struct	switch	typedef	union
unsigned	void	volatile	while			

Número reducido de palabras reservadas (32)

Estructura

```
<tipo de dato> <nombre de variable> [, nombre de variable];
```

Tipo de dato	Descripción.
char	Carácter o entero pequeño (byte)
int	Entero
float	Punto flotante
double	Punto flotante (mayor rango que *float*)
void	Sin tipo (uso especial)

```
float a;
int b,c;
char caracter,otro_caracter;
```

```
unsigned char a;
long double b;
short int i;
```

Modificador	Tipos de actuación		Descripción
signed	char	int	Con signo (por defecto)
unsigned	char	int	Sin signo
long	int	double	Largo
short	int		Corto

Modificador	Efecto
const	Variable de valor constante
volatile	Variable cuyo valor es modificado externamente

Constantes

Hexadecimal:

Comienzan con 0x

0x34 (52 en decimal)

Octal:

Comienzan con un 0

011 (9 decimal)

01173 (123 decimal)

Conjunto de Caracteres:

“Hello World \n”

Constantes de barra Invertida:

Código	Significado
\b	Retroceso
\f	Alimentación de hoja
\n	Nueva línea
\r	Retorno de carro
\t	Tabulador horizontal
\"	Doble comilla
\'	Simple comilla
\0	Nulo
\\	Barra invertida
\v	Tabulador vertical
\a	Alerta
\o	Constante octal
\x	Constante hexadecimal

Operadores

Operadores Aritméticos

Operador		Operador		Operador	
++	Incremento	--	Decremento		
-	Menos unario				
*	Multiplicación.	/	División	%	Módulo
+	Suma	-	Resta		

```
/* Pone 11 en var2, pues primero incrementa var1,*/
/* y luego asigna su valor a var2 */
int var1=10,var2;
var2=++var1;
```

```
/* Pone 10 en var2, pues primero asigna su valor */
/* a var2, y luego incrementa var1 */
int var1=10,var2;
var2=var1++;
```

Operadores Lógicos

Operador		Operador		Operador		Operador	
!	Not						
>	Mayor que	>=	Mayor o igual que	<	Menor que	<=	Menor o igual que
==	Igual	!=	No igual				
&&	And						
||	Or						

```
10>3+9 es equivalente a escribir 10>(3+9).
```

```
if ((c=a*b)<0)
/* Esta expresión asigna a la variable c el valor de
a*b y devuelve su valor para compararlo con el
valor constante 0. Los paréntesis son necesarios
pues el operador asignación tiene la prioridad mas
baja de todos los operadores */
```

Manejo de datos

Reglas

- Todos los char y short int se convierten a int. Todos los float a double.
- Si uno de los operandos es un long double, el otro se convierte en long double.
- Si uno de los operandos es double, el otro se convierte a double.
- Si uno de los operandos es long, el otro se convierte a long.
- Si uno de los operandos es unsigned, el otro se convierte a unsigned

```
char ch;
int i;
float f;
double d;

( ch / i ) + ( f * d ) - ( f + i );
  char   int      float double    float  int

( ch / i ) + ( f * d ) - ( f + i );
  int    int      double double   double  int
```

Manejo de datos “casts”

```
(tipo)expresión
```

```
int a=3,b=2;
float c;
c=a/b;
```

C = 1

```
c=(float)a/b;
```

C = 1,5

If / Else

```
if (condición
   sentencia;
else
   sentencia;
```

```
int a,b;

if (a>b)
{
   b--;
   a=a+5;
}
else
{
   a++;
   b=b-5;
}
if (b-a!=7)
   b=5;
```

Switch/Case

```
switch(variable
{
   case const1:
      sentencia
      break;
   case const2:
      sentencia
      break;
   ...
   default:
      sentencia
}
```

```
char d,e;
switch(d)
{
   case 'a':
   case 'A': switch(e)
             {
                case '1': d='z';
                          e='+';
                          break;
                case '2': d='Z';
                          e='-';
             }
             break;
   case 'b':
   case 'B': switch(e)
             {
                case '0': d='2';
                default: e='+';
             }
}
```

For

```
for(inicialización,condición,incremento) sentencia;
```

```
int i,suma=0;
for(i=1;i<=100;i++)
   suma=suma+i;

int i,j;
```

```
for(i=0,j=100;j>i;i++,j--)
{
   printf("%d\n",j-i);
   printf("%d\n",i-j);
}
```

```
char d;
for(;;)
{
   d=getc(stdin);
   printf("%c",d);
   if (d=='\x1B')
      break;
}
```

La sentencia break provoca la salida del bucle en el cual se encuentra y la ejecución de la sentencia que se encuentra a continuación del bucle.

La sentencia continue provoca que el programa vaya directamente a comprobar la condición del bucle en los bucles while y do/while, o bien, que ejecute el incremento y después compruebe la condición en el caso del bucle for.

Arrays

```
tipo nombre[tamaño];
```

```
int x[100],i;
for(i=0;i<100;i++)
   x[i]=i;

char letras[256];
int i;
for(i=0;i<256;i++)
   letras[i]=i;
```

```
float a[10];
int i;
for(i=0;i<100;i++) /* Este bucle es incorrecto */
   a[i]=i;
```

```
tipo nombre[tam1][tam2]...[tamN];
```

```
float vector[3]={-3.0,5.7,-7.5};

char cadena[]="Esto es una cadena";
```

Punteros a funciones

```
tipo de dato (*nombre de la variable)(prototipo);
```

Al igual que cualquier otro tipo de dato, una función ocupa una dirección de memoria, y por tanto, puede ser apuntada por un puntero. Generalmente, los punteros a funciones se usan en la programación de bajo nivel, tal como modificación de interrupciones, creación de controladores de dispositivos, etc

Static a funciones

Al igual que en el caso de las variables globales, es posible aplicar delante de una función el modificador de almacenamiento static. Dicho modificador hace que la función sobre la que se aplica sea local al módulo donde se encuentra, y no pueda ser conocida por los restantes módulos del programa, de igual forma a como sucedía con las variables globales.

Y lo ultimo…

typedef

```
typedef tipo nombre;
```

El lenguaje C permite mediante el uso de la palabra reservada typedef definir nuevos nombres para los tipos de datos existentes. La palabra clave typedef permite solo asignarle un nuevo nombre a un tipo de datos ya existente.

```
typedef int entero;

typedef struct{
   unsigned codigo;
   char nombre[40];
   char apellido[40];
}cliente;
```

```
entero a;
cliente b,*c;
```

Preprocesador

En un programa escrito en C, es posible incluir diversas instrucciones para el compilador dentro del código fuente del programa

#if	#ifdef	#ifndef	#else
#elif	#endif	#include	#define
#undef	#line	#error	#pragma

Y lo ultimo…

#define

La directiva define se usa para definir un identificador y una cadena que el compilador sustituirá por el identificador cada vez que se encuentre en el archivo fuente.

```
#define TRUE 1
#define FALSE 0
```

El uso mas común de la directiva #define es la definición de valores constantes en el programa, tamaños de arrays, etc.

```
#define MIN(a,b) (a<b) ? a : b
```

#undef

La directiva #undef permite quitar una definición de "nombre de macro" que se realizo con anterioridad.

```
#define TAM 10
.......
#undef TAM
```

Y lo ultimo…

#include

La directiva #include fuerza al compilador a incluir otro archivo fuente en el archivo que tiene la directiva #include, y a compilarlo. El nombre del archivo fuente a incluir se colocara entre comillas dobles o entre paréntesis

```
#include <stdio.h>
#include "stdio.h"
```

La diferencia existente entre encerrar el archivo entre paréntesis de ángulo o entre comillas dobles, es que, en el primer caso, se busca el archivo en los directorios de la linea de ordenes de compilación, y, después en los directorios standard de C, pero nunca en el directorio de trabajo

Y lo ultimo…

#ifdef #ifndef

La directivas #ifdef y #ifndef se usan también para compilación condicional, solo que no evalúan expresión alguna, solo comprueba si esta definido (#ifdef) o si no esta definido (#ifndef) algún nombre de macro.

```
#ifdef nombre de macro
   secuencia de sentencias
#else
   secuencia de sentencias
#endif
```