Aguijón: Introducción Básica

Objetivos del Curso

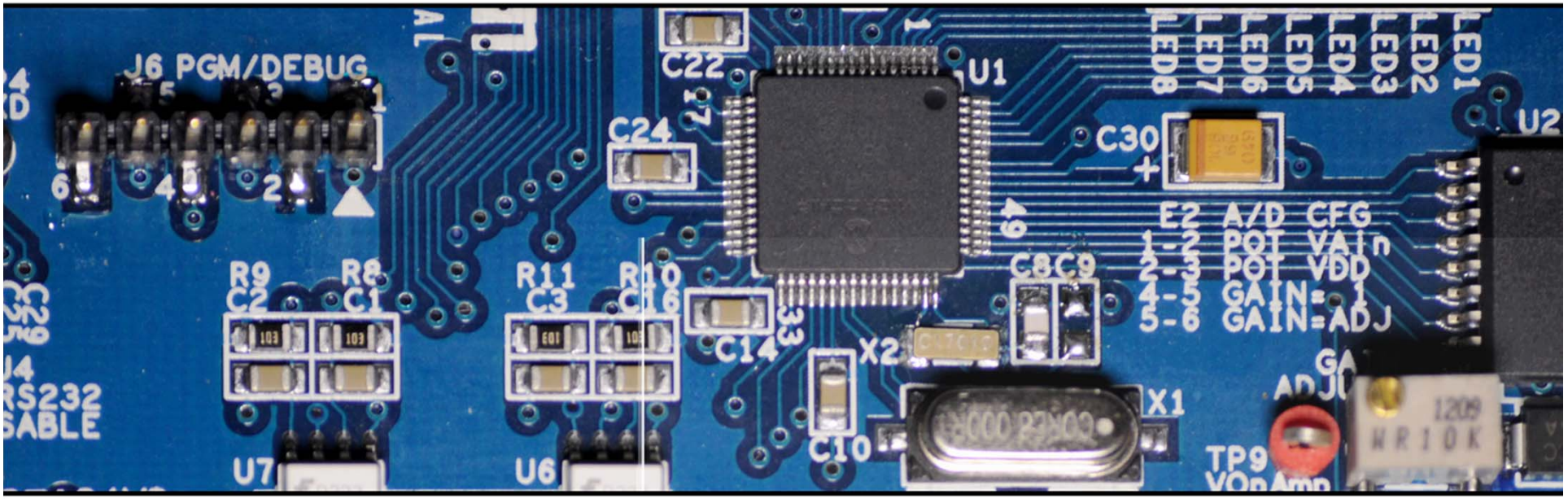
Al finalizar este curso, podrás:

- *Diseñar aplicaciones embebidas utilizando las librerías HammerHead.*
- *Conocer de una mejor manera la arquitectura de hardware en el Aguijón.*
- *Conocer los alcances que implica el utilizar una plataforma de desarrollo profesional de alto desempeño.*

Agenda

- *Diseño de Hardware*
- *Bloques funcionales*
- *Herramientas de desarrollo*
- *Introducción a las librerías HammerHead*
- *Sumario*

Sinagrón DIGITAL

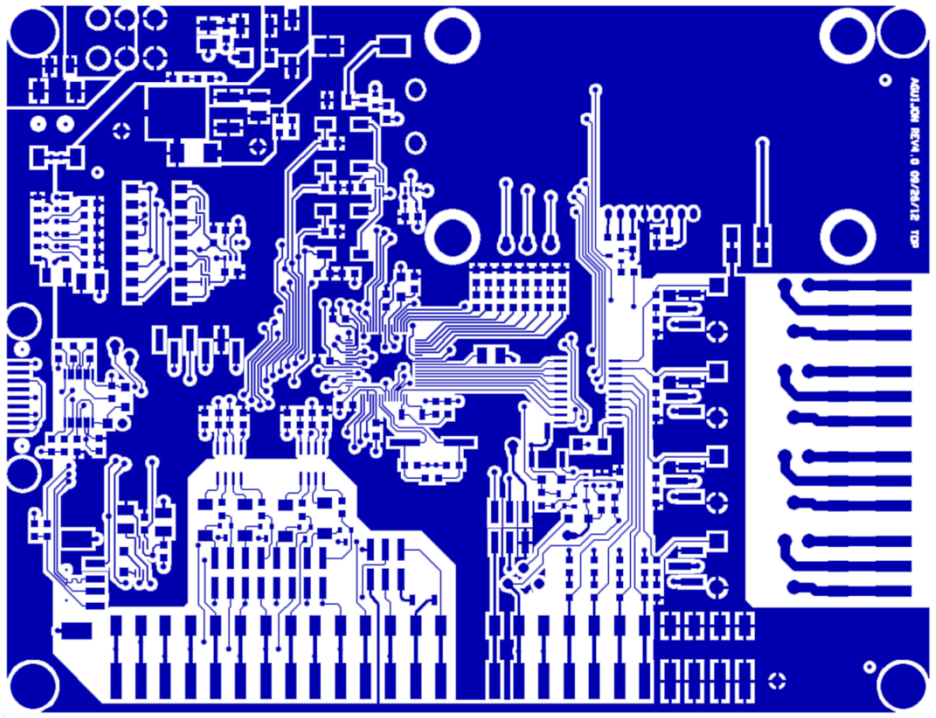


Diseño de hardware

Manufactura

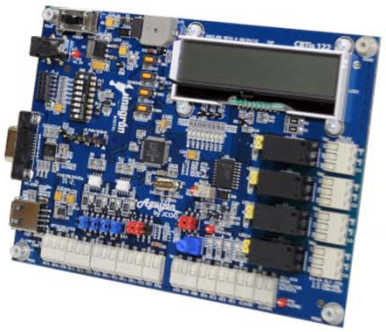
Diseño del circuito impreso:

- *Integridad de señal*
- *Distribución de potencia*
- *Inmunidad al ruido*
- *Componentes SMT*



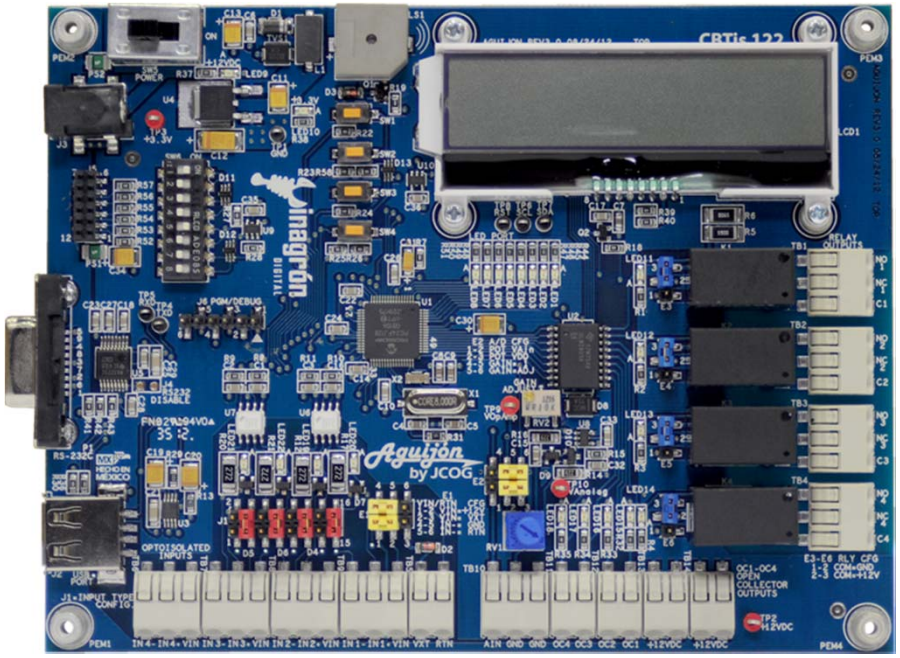
Empaque:

- *Kit con maleta ergonómica, transformador de pared de +12VDC y desarmador reversible incluidos.*



Diseño de Hardware:

- *Diseño compacto, fabricado con tecnología SMT*
- *Cuenta con los módulos necesarios para realizar una diversidad de proyectos.*



Microchip PIC24F

Arquitectura:

- *Procesador de 16 bits*
- *128KB Flash*
- *16KB RAM*
- *5 módulos PWM*
- *6 timers*
- *3 módulos SPI*
- *3 módulos I2C*
- *RTCC*
- *Pines re-asignables*

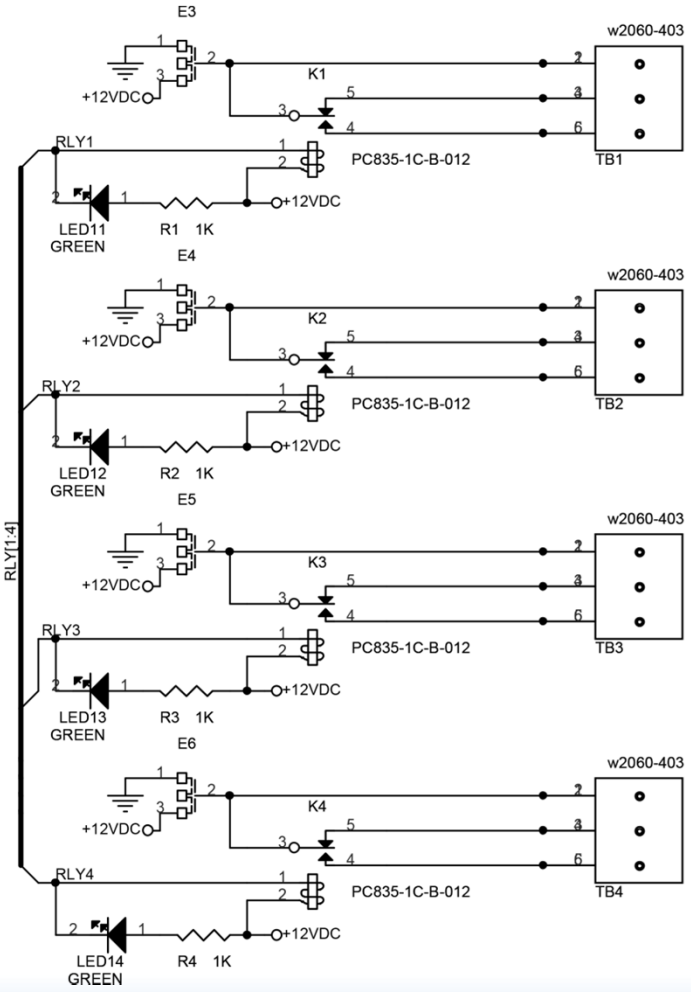
inagrón
DIGITAL

Bloques Funcionales - Salidas

Salidas en Relevador

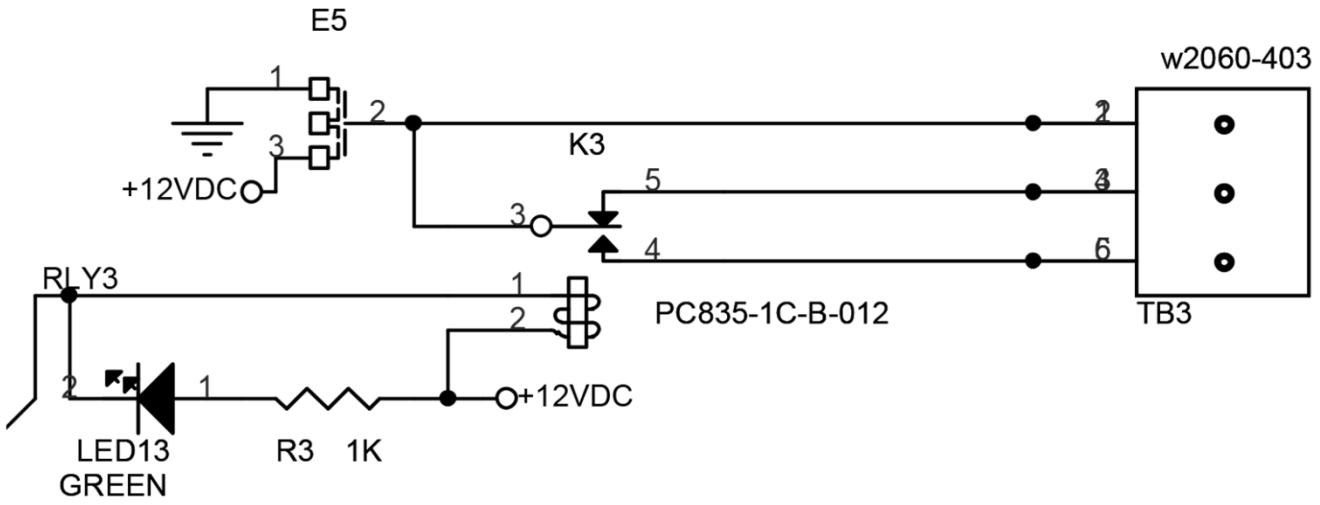
Características:

- *4 Salidas en relevador tipo "C " @10amp*
- *+12VDC ó GND disponibles para conmutar al pin común*
- *LEDs indicadores de estado para cada salida*



Configuración:

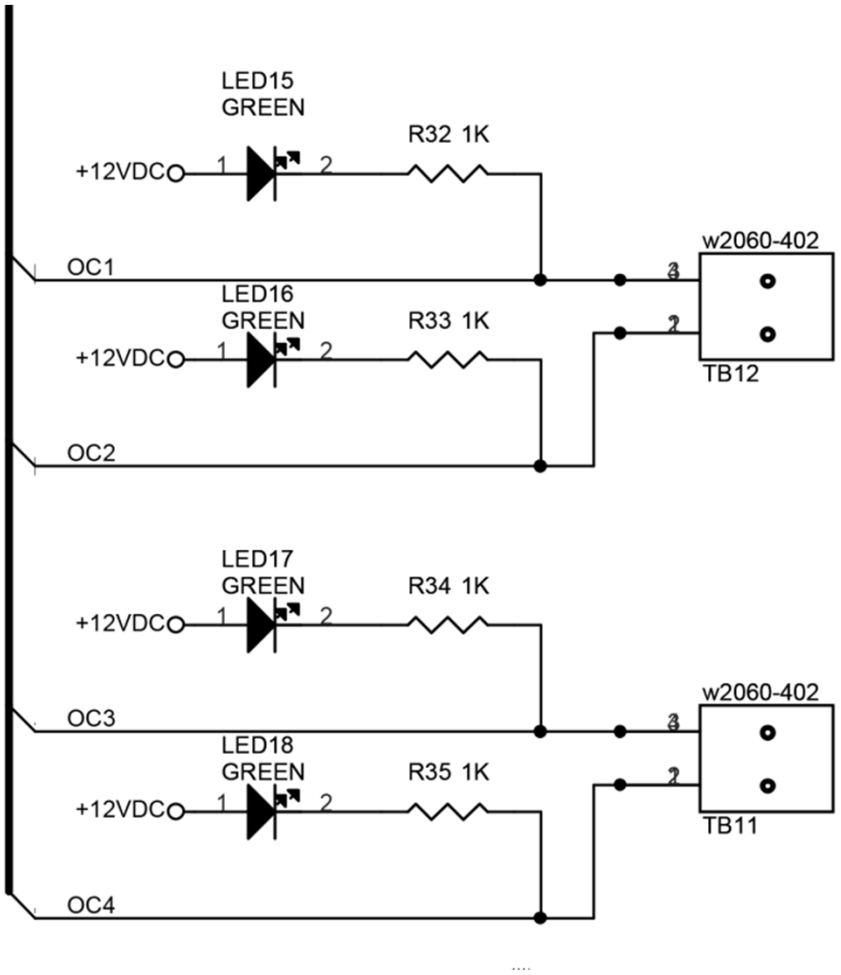
	Jumpers E2-E6	Pin común del relevador:
Configuración de los relevadores	1-2	GND
	2-3	+12VDC
	Abierto	Abierto



Salidas en Colector Abierto

Características:

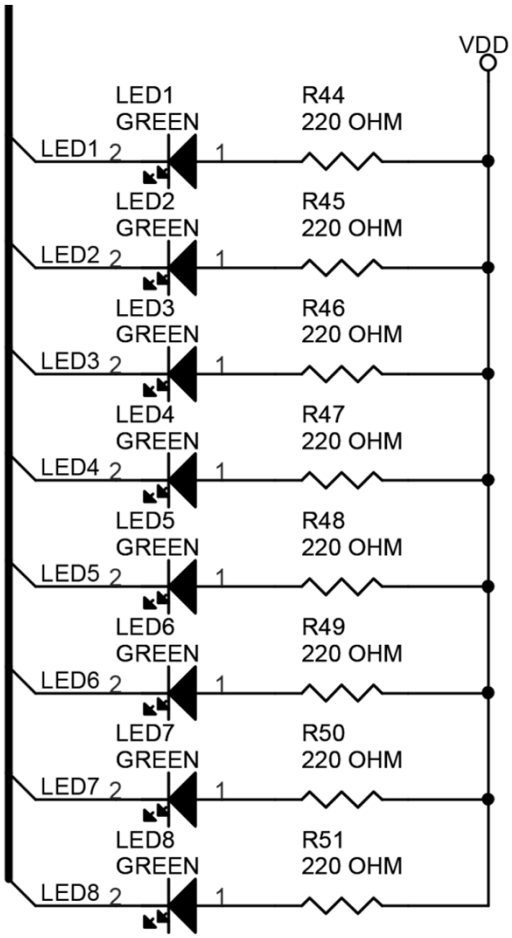
- *4 Salidas con capacidad de 500mA*
- *+12VDC en 4 borneras adicionales*
- *LEDs indicadores de estado para cada salida*



Puerto de LEDs

Características:

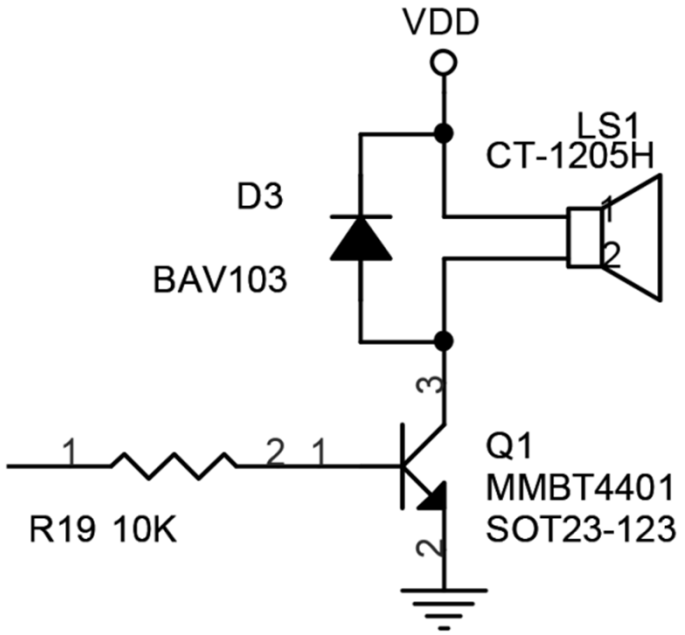
- *Puerto de 8 bits dedicado*

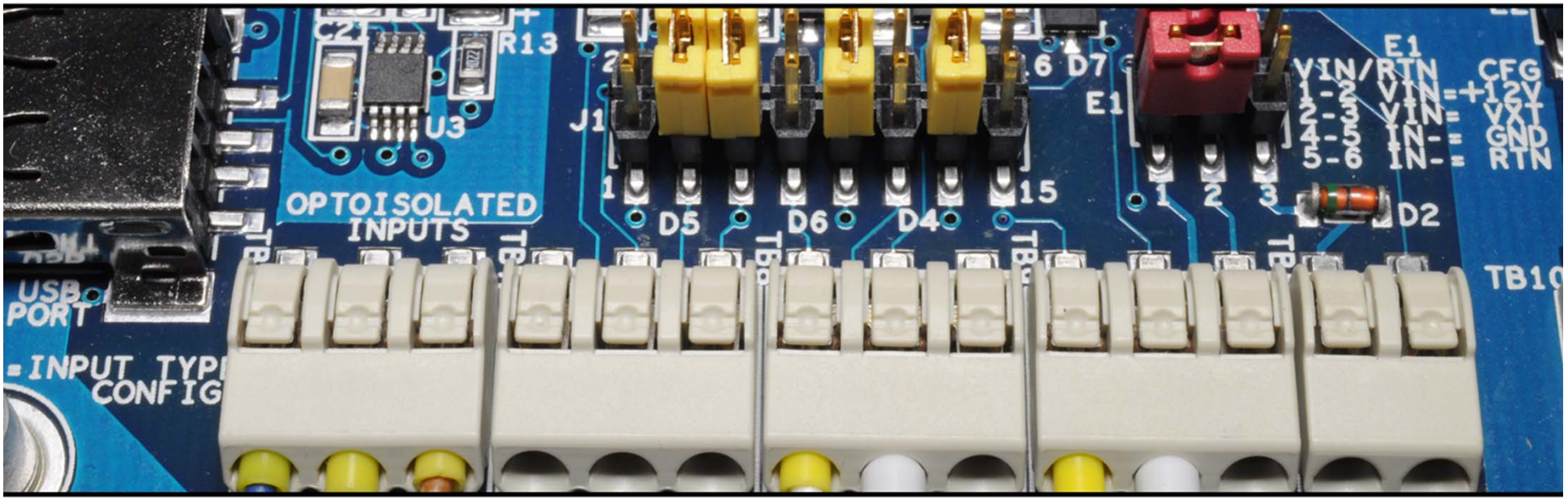


Buzzer

Características:

- *Controlado por PWM*
- *Frecuencia nominal de 2.4kHz*
- *Capaz de generar distintos tonos*





Bloques Funcionales - Entradas

Entradas Opto-aisladas

Características:

- *4 Entradas con modo de operación configurable*
- *LEDs indicadores de estado*
- *Entrada externa de voltaje de hasta 48VDC*

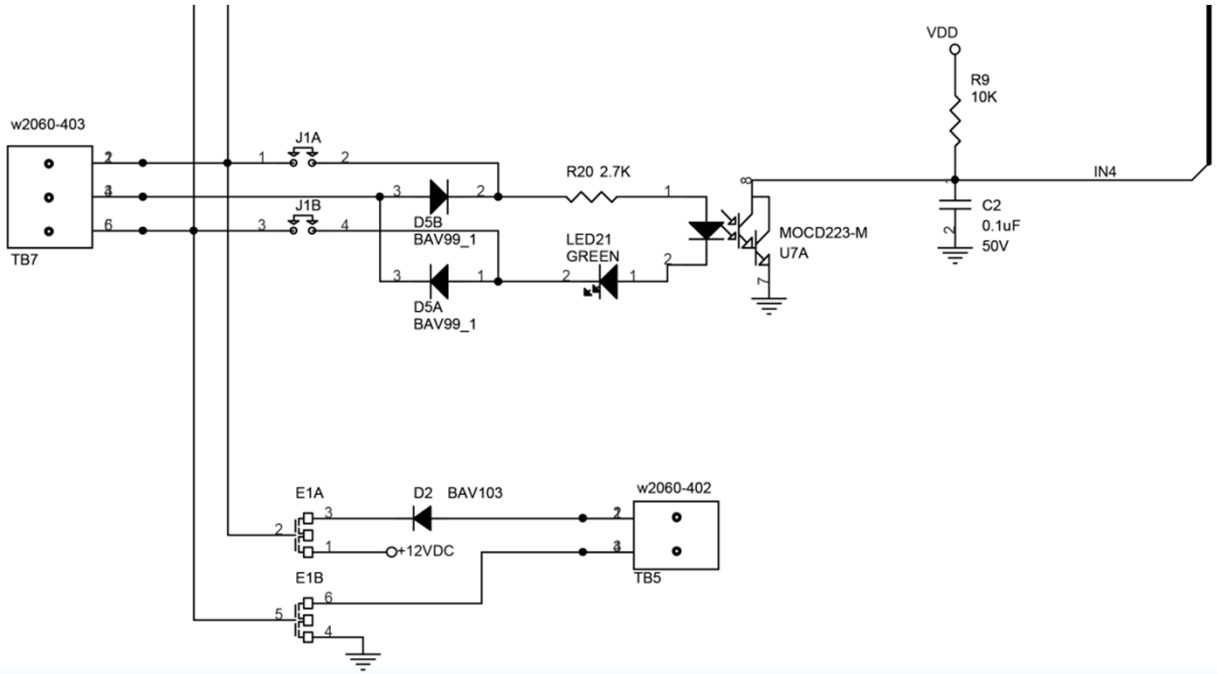


Tabla de configuración:

Positive Voltage Source	***E1A***	***E1B***	***J1C***	***J1D***	***VIN***	***IN+***	***IN-***
	2-3	5-6	NC	7-8	NC	+Source	Return
Current sink to return	***E1A***	***E1B***	***J1C***	***J1D***	***VIN***	***IN+***	***IN-***
	2-3	5-6	5-6	NC	NC	Sink	Return
Externally Wetted Contacts	***E1A***	***E1B***	***J1C***	***J1D***	***VIN***	***IN+***	***IN-***
	NC	NC	NC	7-8	NC	1+	1-
Dry contacts positive common	***E1A***	***E1B***	***J1C***	***J1D***	***VIN***	***IN+***	***IN-***
	2-1	5-4	5-6	NC	NC	IN-	IN+
Dry contacts return common	***E1A***	***E1B***	***J1C***	***J1D***	***VIN***	***IN+***	***IN-***
	2-1	5-4	NC	7-8	IN+	VIN	NC

DIP SW + Teclado

Características:

- *DIP SW y teclas compartidas*
- *Se leen 12 entradas con 6 pines (4 para leer y 2 de control)*
- *DIP switch de 8 posiciones*
- *4 Teclas en tablilla*

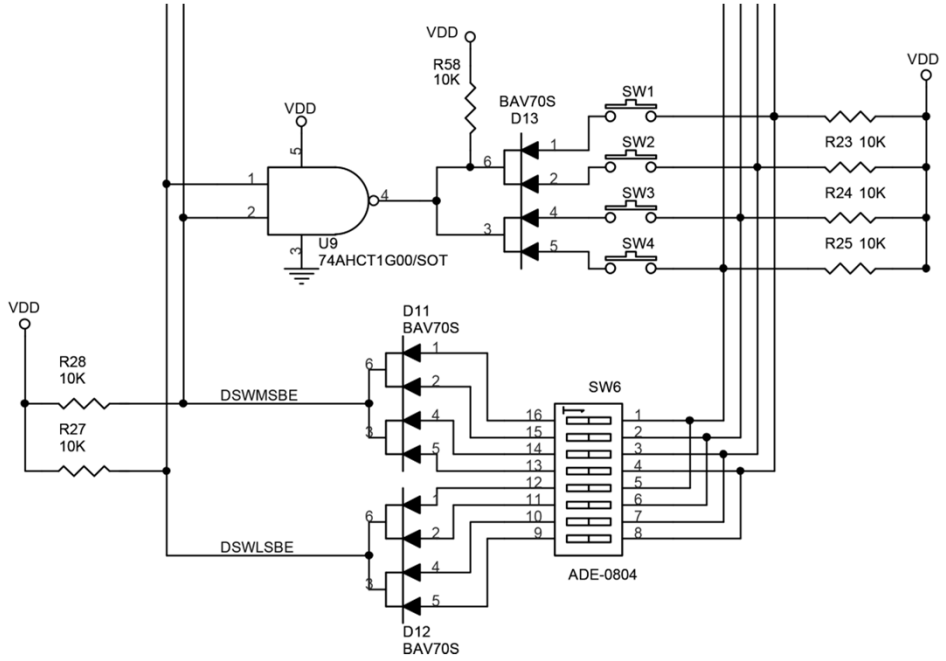


Tabla de verdad:

MSB_CTRL	LSB_CTRL	Estado:
1	1	A
1	0	B
0	1	C
0	0	D

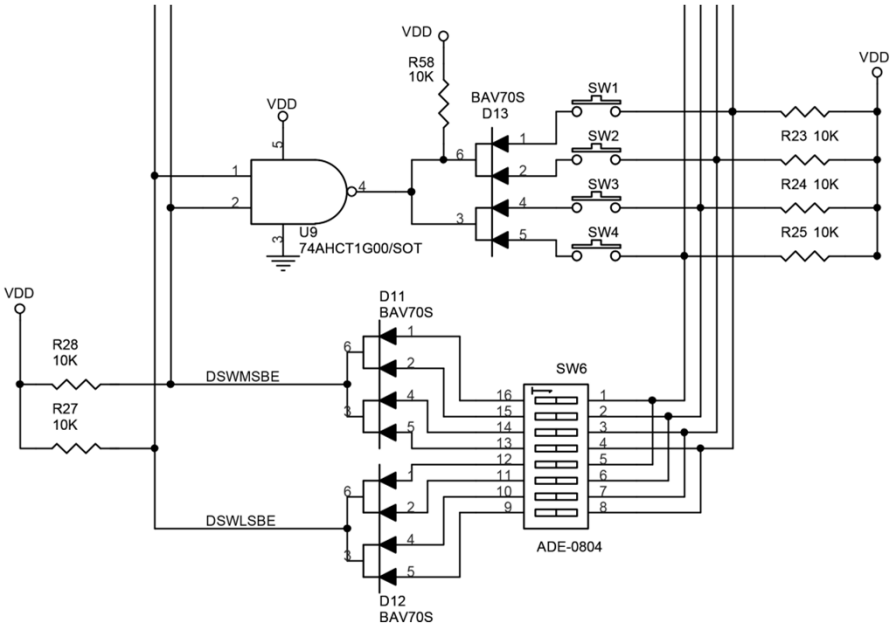


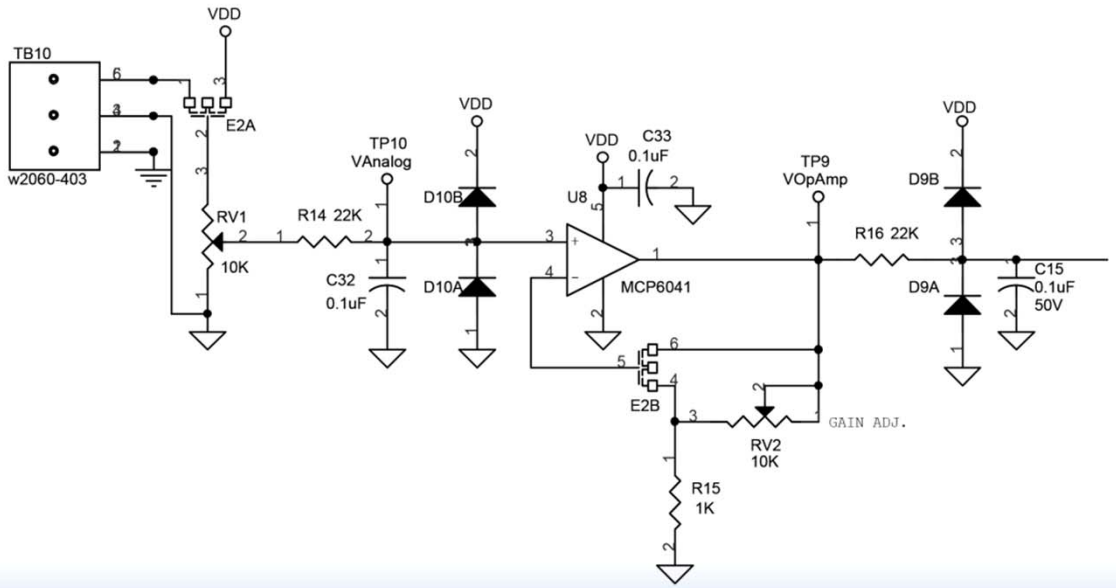
Tabla de Estados:

A = Teclas habilitadas y DIP SW deshabilitado
B = Teclas deshabilitadas y LSB habilitado
C = Teclas deshabilitadas y MSB habilitado
D = Lectura inválida

Entrada Analógica

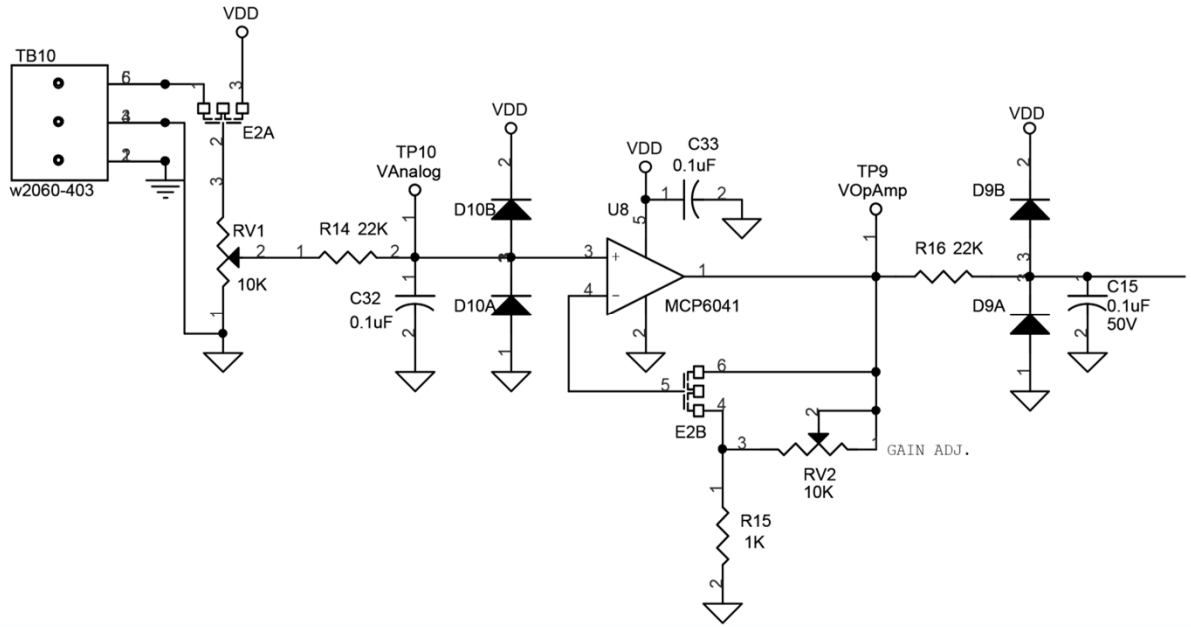
Características:

- *Ganancia unitaria o ajustable*
- *Conversión digital de 10 bits, vref = +3.3V*
- *Protección contra sobre voltaje*
- *Potenciómetro en tablilla para atenuar +3.3V o un voltaje externo*

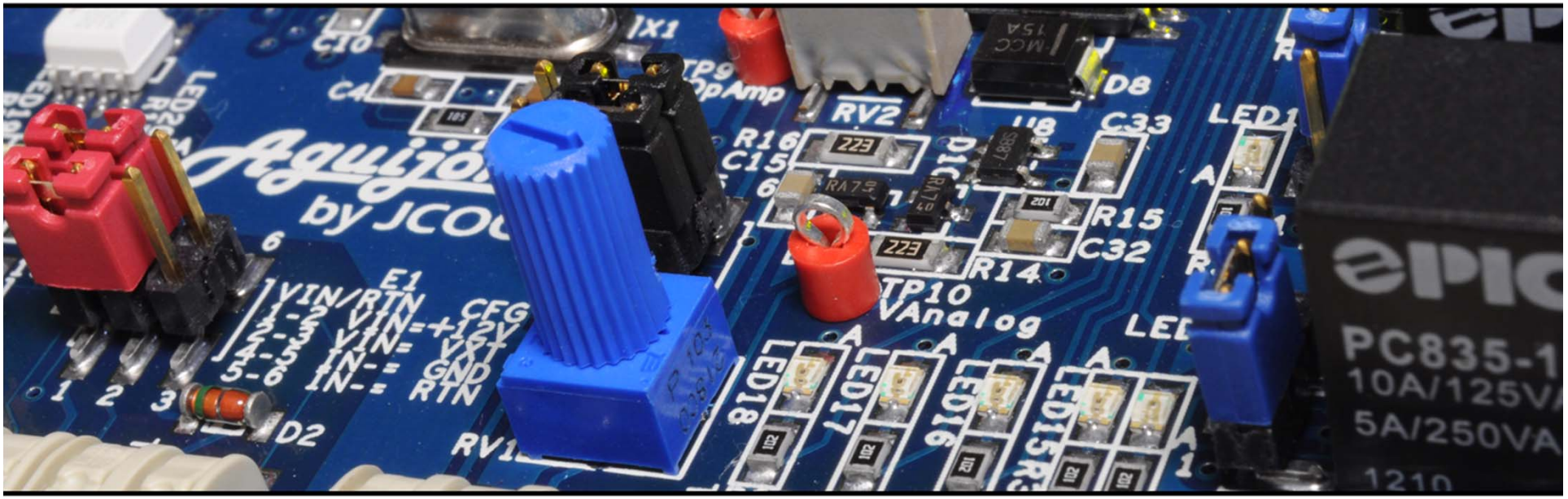


Configuración:

Jumper E2A	Jumper E2B	Ganancia	Entrada analógica	RV1
1-2	5-6	Unitaria	Externa	Atenúa señal externa
2-3	5-6		Interna (+3.3V)	Atenúa +3.3V
1-2	4-5	Ajustable con RV2	Externa	Atenúa señal externa
2-3	4-5		Interna (+3.3V)	Atenúa +3.3V



Vinagrón
DIGITAL

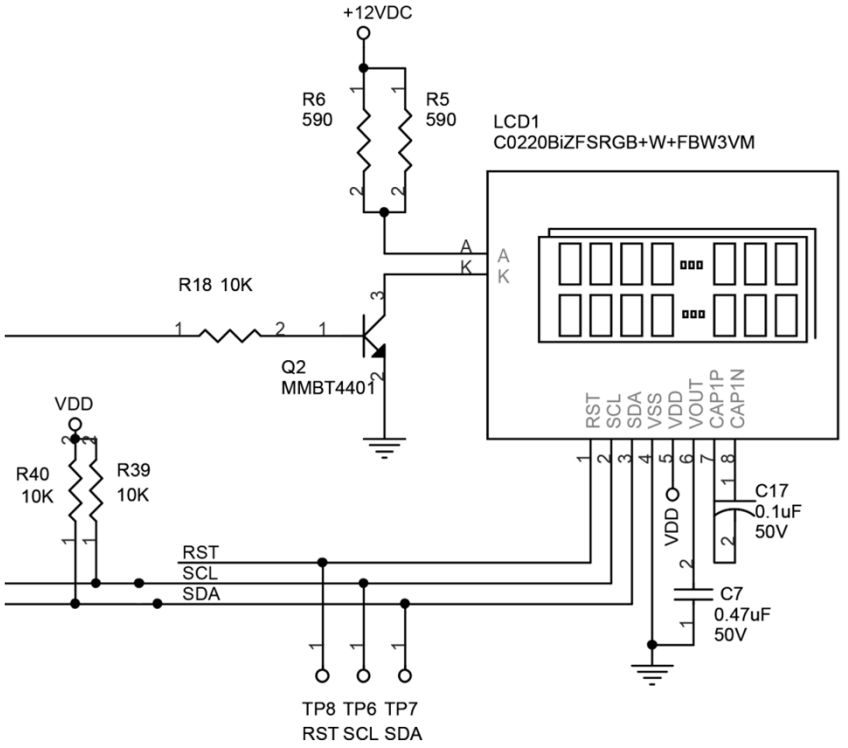


Bloques Funcionales

Display LCD

Características:

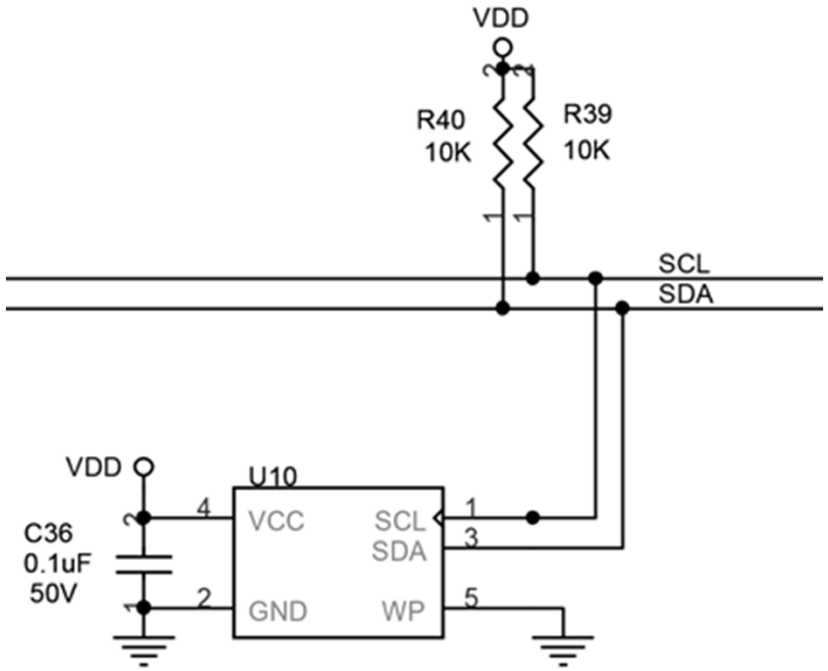
- *LCD - COG con interfaz I2C de hasta 400kHz*
- *2 renglones x 20 columnas*
- *Intensidad de backlight ajustable por PWM*
- *Comparte bus I2C con la EEPROM*



EEPROM

Características:

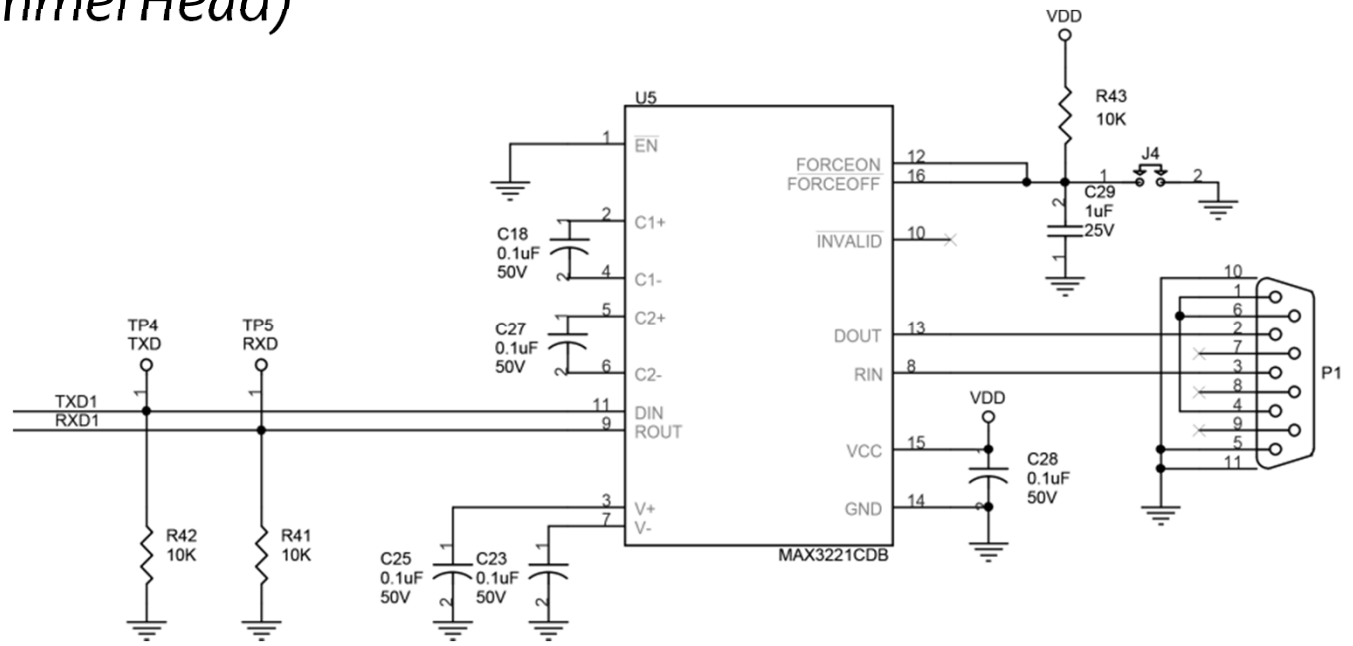
- *Capacidad de 127 Bytes*
- *Comparte bus I2C con la LCD*



RS232C

Características:

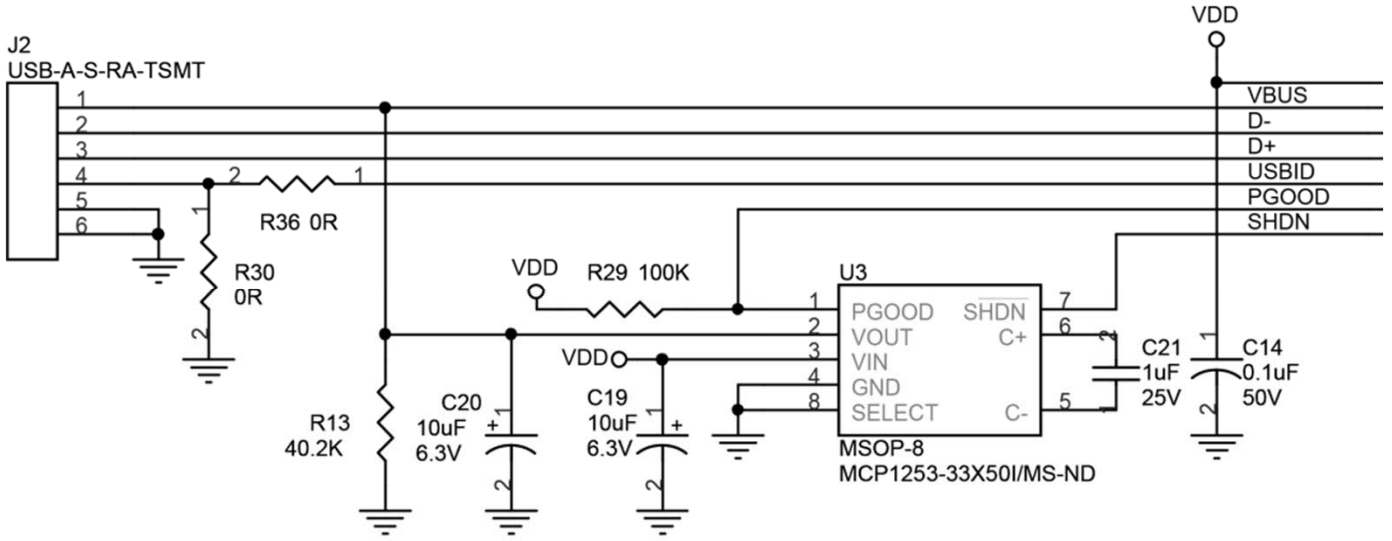
- *Driver: MAX3221*
- *110 – 115200 bps*
- *Recepción de datos controlada por interrupción (usando HammerHead)*



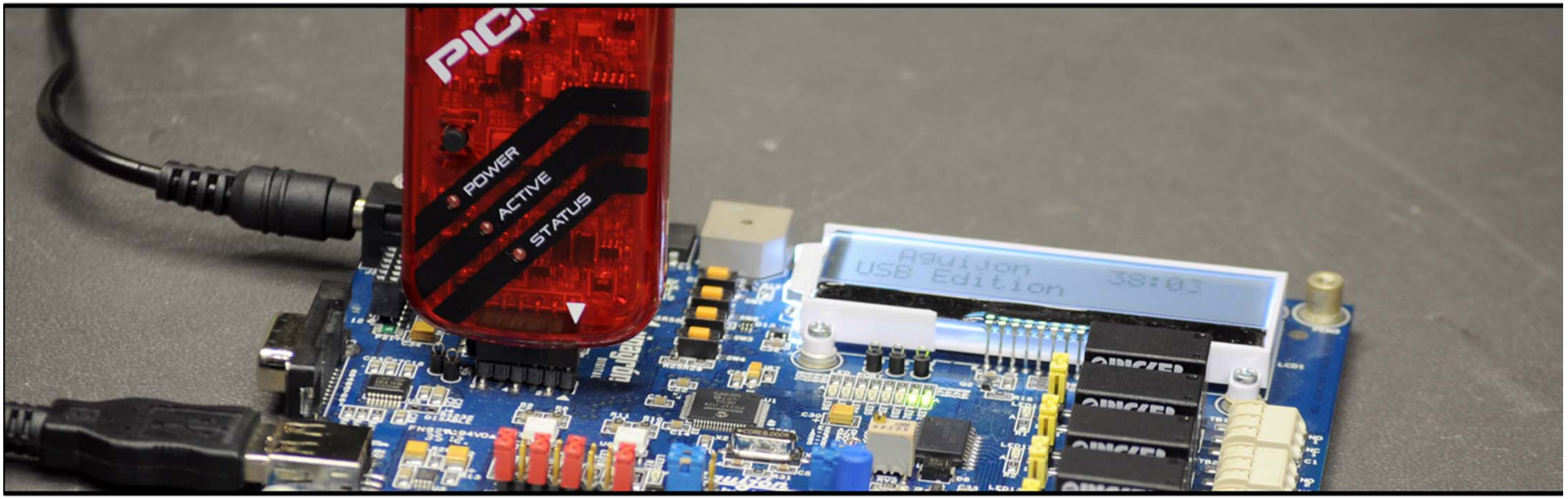
USB

Características:

- *Conector tipo 'A' permite actuar como 'HOST' o 'DEVICE'*
- *Switching Regulator para generar +5VDC de salida al conector (modo HOST)*
- *Puede ser usado como programador (bootloader)*

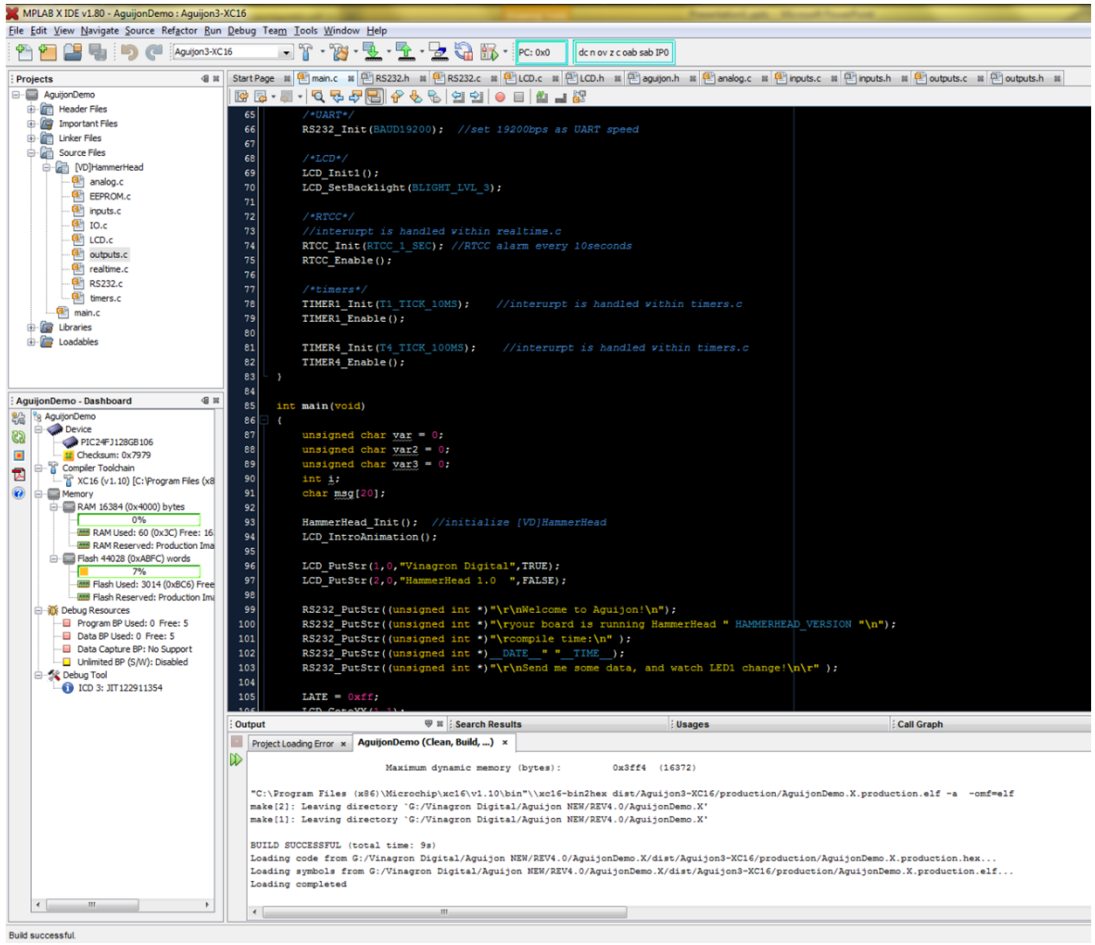


inagrón
DIGITAL

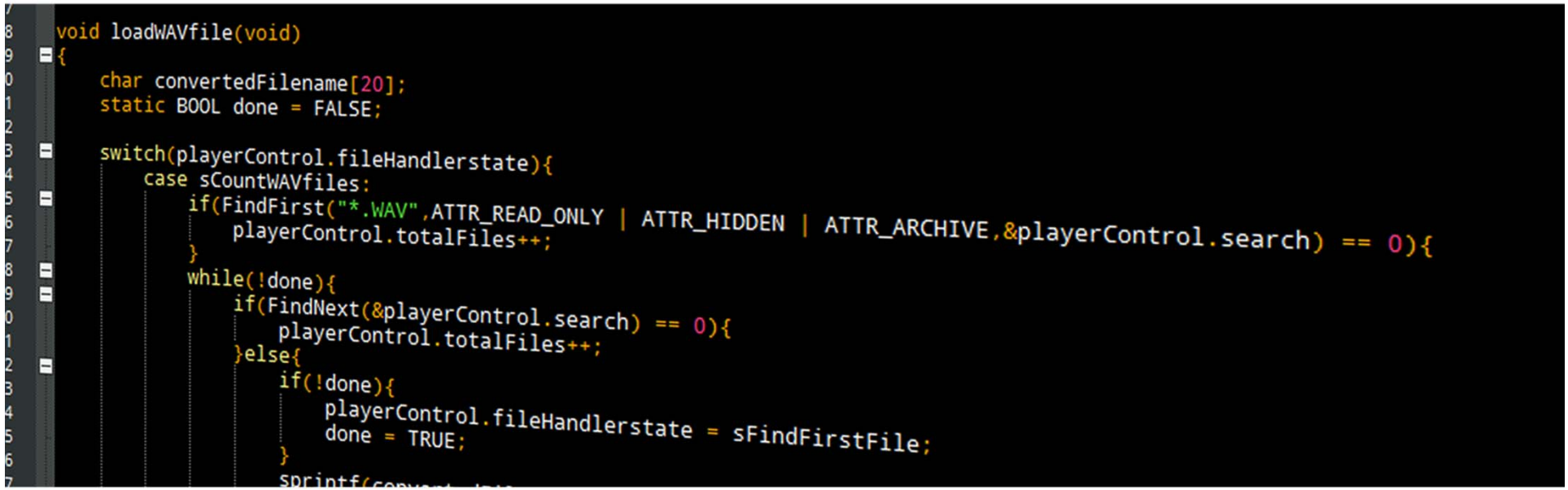


Herramientas de Desarrollo

MPLAB X®


MPLAB X IDE v1.80 - AguijonDemo : Aguijon3-XC16
File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Projects
AguijonDemo
Header Files
Important Files
Linker Files
Source Files
[VD]HammerHead
analog.c
EEPROM.c
inputs.c
IO.c
LCD.c
outputs.c
realtime.c
RS232.c
timers.c
main.c
Libraries
Loadables
AguijonDemo - Dashboard
Device
PIC24FJ128GB106
Compiler Toolchain
Memory
Debug Resources
Debug Tool
ICD 3: JIT122911354
/*UART*/
RS232_Init(BAUD19200); //set 19200bps as UART speed
/*LCD*/
LCD_Init1();
LCD_SetBacklight(BLIGHT_LVL_3);
/*RTCC*/
//interurpt is handled within realtime.c
RTCC_Init(RTCC_1_SEC); //RTCC alarm every 10seconds
RTCC_Enable();
/*timers*/
TIMER1_Init(T1_TICK_10MS); //interurpt is handled within timers.c
TIMER1_Enable();
TIMER4_Init(T4_TICK_100MS); //interurpt is handled within timers.c
TIMER4_Enable();
int main(void)
{
unsigned char var = 0;
unsigned char var2 = 0;
unsigned char var3 = 0;
int i;
char msg[20];
HammerHead_Init(); //initialize [VD]HammerHead
LCD_IntroAnimation();
LCD_PutStr(1,0,"Vinagron Digital",TRUE);
LCD_PutStr(2,0,"HammerHead 1.0 ",FALSE);
RS232_PutStr((unsigned int *)"\r\nWelcome to Aguijon!\n");
RS232_PutStr((unsigned int *)"\ryour board is running HammerHead " HAMMERHEAD_VERSION "\n");
RS232_PutStr((unsigned int *)"\rcompile time:\n");
RS232_PutStr((unsigned int *)__DATE__ " "__TIME__);
RS232_PutStr((unsigned int *)"\r\nSend me some data, and watch LED1 change!\n\r");
LATE = 0xff;
Output
Search Results
Usages
Call Graph
Project Loading Error
AguijonDemo (Clean, Build, ...)
Maximum dynamic memory (bytes): 0x3ff4 (16372)
"C:\Program Files (x86)\Microchip\xc16\v1.10\bin"\\xc16-bin2hex dist/Aguijon3-XC16/production/AguijonDemo.X.production.elf -a -omf=elf
make[2]: Leaving directory `G:/Vinagron Digital/Aguijon NEW/REV4.0/AguijonDemo.X'
make[1]: Leaving directory `G:/Vinagron Digital/Aguijon NEW/REV4.0/AguijonDemo.X'
BUILD SUCCESSFUL (total time: 9s)
Loading code from G:/Vinagron Digital/Aguijon NEW/REV4.0/AguijonDemo.X/dist/Aguijon3-XC16/production/AguijonDemo.X.production.hex...
Loading symbols from G:/Vinagron Digital/Aguijon NEW/REV4.0/AguijonDemo.X/dist/Aguijon3-XC16/production/AguijonDemo.X.production.elf...
Loading completed
Build successful.

ICD3 y PICkit3



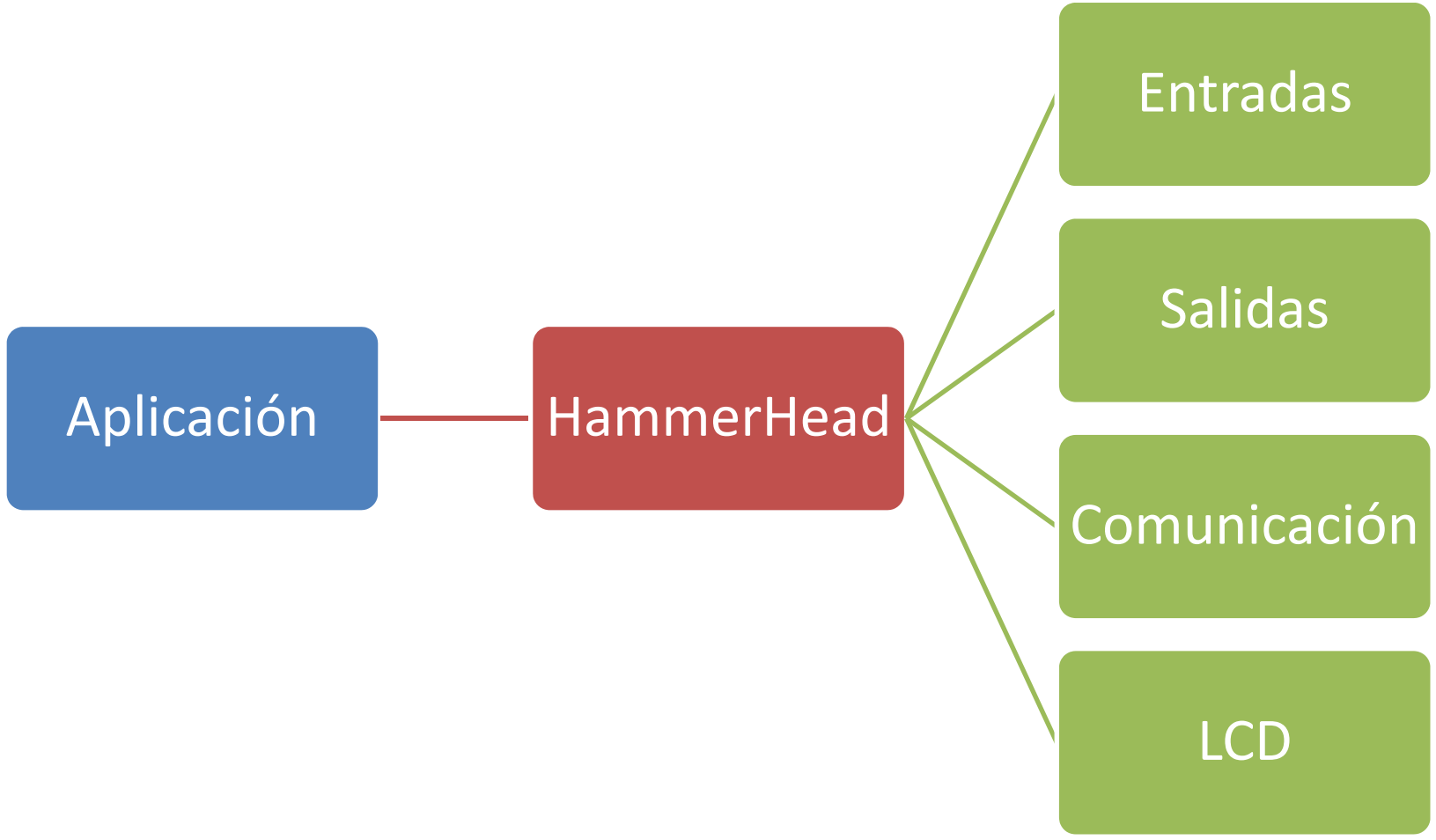
```
void loadWAVfile(void)
{
    char convertedFilename[20];
    static BOOL done = FALSE;

    switch(playerControl.fileHandlerstate){
        case sCountWAVfiles:
            if(FindFirst("*.WAV",ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_ARCHIVE,&playerControl.search) == 0){
                playerControl.totalFiles++;
            }
            while(!done){
                if(FindNext(&playerControl.search) == 0){
                    playerControl.totalFiles++;
                }else{
                    if(!done){
                        playerControl.fileHandlerstate = sFindFirstFile;
                        done = TRUE;
                    }
                    sprintf(convert_data
```

HammerHead

Características:

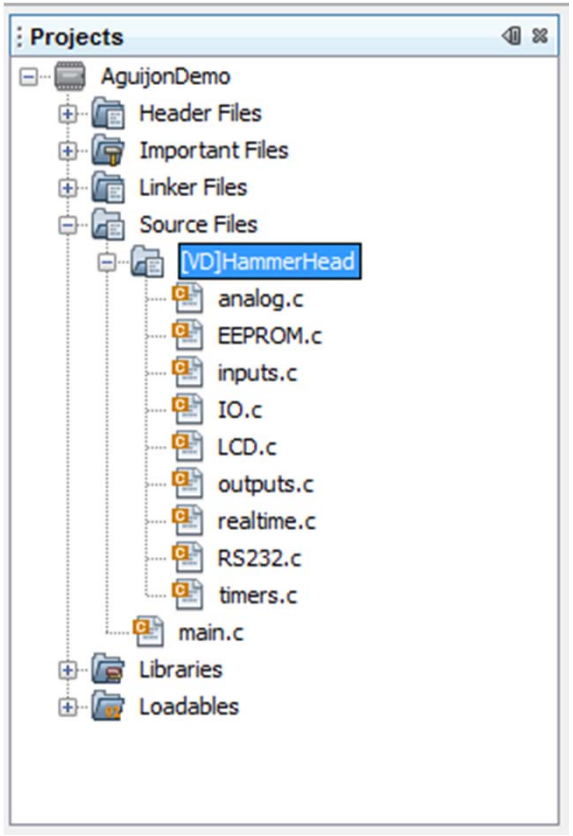
- *Librerías de control de Vinagrón Digital (Open Source)*
- *Tamaño compacto(< 10KB)*
- *Soporte para todos los bloques funcionales*
- *Incluye funciones de inicialización, lectura, escritura y control general*

¿Cómo funciona?



Estructura:

- *Cada módulo cuenta con su propia librería, que consta de dos archivos; un «*.c» y «*.h»*



Más a detalle:

- *Cada archivo '*.c' incluye:*
  - *Inicialización*
  - *Lectura*
  - *Escritura*
  - *Control General*
- *Los archivos '*.h' incluyen:*
  - *Prototipos de funciones*
  - *Definiciones*
  - *Macros*
- *Archivo 'BSP.h' global:*
  - *Incluye definiciones de pines*
- *Archivo 'aguijon.h' global:*
  - *Incluye 'BSP.h'*
  - *Incluye el header del procesador*
  - *Incluye STDIO y STDLIB*
  - *Incluye GenericTypeDefs de Microchip*
  - *Incluye definiciones de delay*

Ejemplo: inputs.h

```
7
8   #ifndef INPUTS_H
9   #define INPUTS_H
10
11  #define DIPSW_ENABLE_MSB_CMD    0   //enable MSB nibble (DIP SW)
12  #define DIPSW_ENABLE_LSB_CMD    1   //enable LSB nibble (DIP SW)
13  #define DIPSW_DISABLE_CMD       2   //disables DIP SW and leaves keys operational
14
15  #define KEYBOARD_MASK           0x3C00
16  #define INPUT_MASK              0x000F
17
18  /*Function prototypes*/
19  unsigned char   DIPSW_Read  (void);
20  void            DIPSW_Config(int );
21
22  //single input functions
23  unsigned char   SW_Read     (void);
24  unsigned char   IN_Read     (void);
25  //port functions
26  unsigned char   SW_ReadPort (void);
27  unsigned char   IN_ReadPort (void);
28
29  #endif  /* INPUTS_H */
30
31
```

Estructura:

- ***Functions:***
  - *PREFIJO_CamelCase* — `void ADC_Init(void);`
- ***Variables:***
  - ***Externas:*** *PREFIJO_CamelCase* — `extern int ADC_Lectura;`
  - ***Globales:*** *CamelCase* — `int ConversionTemp = 0;`
  - ***Locales:*** *mixedCase* — `static int mathVariable = 0;`
- ***Defines:***
  - `#define ADC_MAX_VALUE 0x03F6`
- ***Macro:***
  - `Macro_Call();`
- ***Typedefs:***
  - ***Prefijo_Nombre_t***

analog.h - ejemplo:

```
#ifndef ANALOG_H
#define ANALOG_H

#define ADC_MAX_VALUE    0x03F6
#define ADC_MIN_VALUE    0x00

/* Function prototypes*/
void ADC_Init   (void);
int  ADC_Read   (void);
int  ADC_Range  (int min_val, int max_val);

#endif /* ANALOG_H */
```

analog.h - ejemplo:

```
#ifndef ANALOG_H
#define ANALOG_H

#define ADC_MAX_VALUE    0x03F6
#define ADC_MIN_VALUE    0x00

/* Function prototypes*/
void ADC_Init   (void);
int  ADC_Read   (void);
int  ADC_Range  (int min_val, int max_val);

#endif /* ANALOG_H */
```

Include Guard

analog.h - ejemplo:

```
#ifndef ANALOG_H
#define ANALOG_H

#define ADC_MAX_VALUE    0x03F6
#define ADC_MIN_VALUE    0x00

/* Function prototypes*/
void ADC_Init   (void);
int  ADC_Read   (void);
int  ADC_Range  (int min_val, int max_val);

#endif /* ANALOG_H */
```

Include Guard

Definiciones

analog.h - ejemplo:

```
#ifndef ANALOG_H
#define ANALOG_H

#define ADC_MAX_VALUE    0x03F6
#define ADC_MIN_VALUE    0x00

/* Function prototypes*/
void ADC_Init   (void);
int  ADC_Read   (void);
int  ADC_Range  (int min_val, int max_val);

#endif /* ANALOG_H */
```

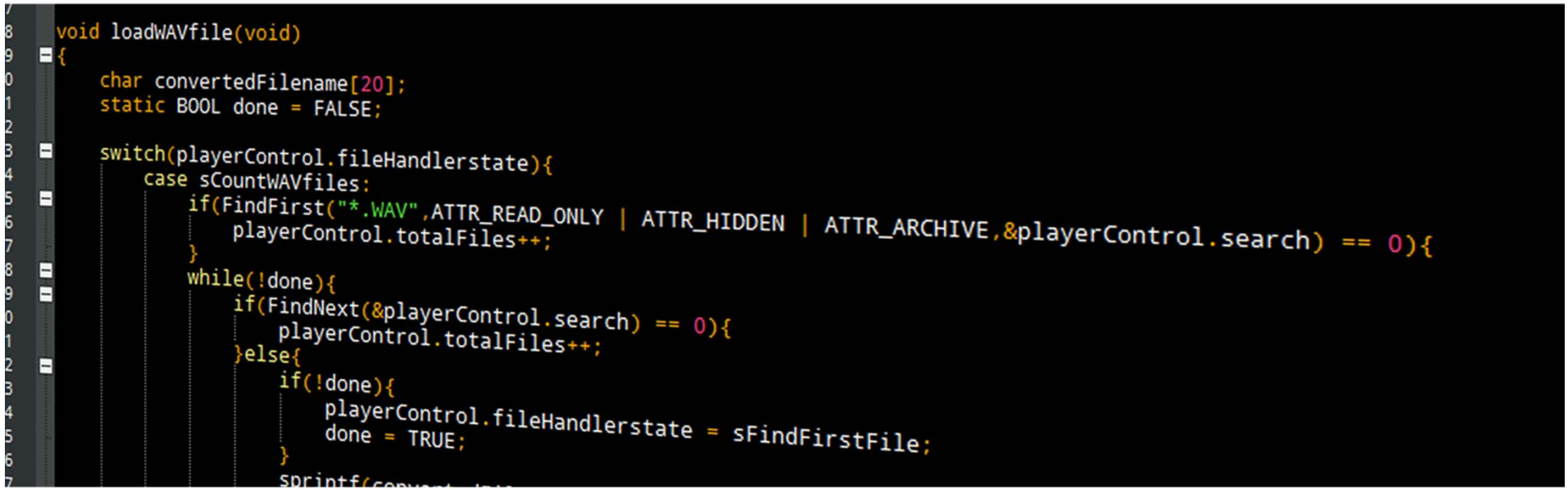
Include Guard

Definiciones

Prototipos de funciones

analog.c - ejemplo:

```
#include "aguijon.h"
#include "analog.h"

////////////////////////////////////////
void ADC_Init(void)
{
    AD1PCFG           = 0x7FFF;   //select RB15(AN15) as analog input
    AD1CON1           = 0x00E0;   //auto start conversion
    AD1CON2           = 0;        // use AVDD, AVSS as reference pins
    AD1CON3           = 0x1F02;   // Tsamp = 32 x Tad; Tad=125ns
    AD1CSSL           = 0;        // no scanning required
    AD1CON1bits.ADON = 1;         // turn on the ADC
}
```



```
void loadWAVfile(void)
{
    char convertedFilename[20];
    static BOOL done = FALSE;

    switch(playerControl.fileHandlerstate){
        case sCountWAVfiles:
            if(FindFirst("*.WAV",ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_ARCHIVE,&playerControl.search) == 0){
                playerControl.totalFiles++;
            }
            while(!done){
                if(FindNext(&playerControl.search) == 0){
                    playerControl.totalFiles++;
                }else{
                    if(!done){
                        playerControl.fileHandlerstate = sFindFirstFile;
                        done = TRUE;
                    }
                    sprintf(convert
```

HammerHead - Funciones

Entradas («inputs.h»)

Librerías HammerHead

```
#ifndef INPUTS_H
#define INPUTS_H

#define DIPSW_ENABLE_MSB_CMD    0   //enable MSB nibble (DIP SW)
#define DIPSW_ENABLE_LSB_CMD    1   //enable LSB nibble (DIP SW)
#define DIPSW_DISABLE_CMD       2   //disables DIP SW and leaves keys operational

#define KEYBOARD_MASK           0x3C00
#define INPUT_MASK              0x000F

/*Function prototypes*/
unsigned char   DIPSW_Read  (void);
void            DIPSW_Config(int );

//single input functions
unsigned char   SW_Read     (void);
unsigned char   IN_Read     (void);
//port functions
unsigned char   SW_ReadPort (void);
unsigned char   IN_ReadPort (void);

#endif  /* INPUTS_H */
```

Entradas («inputs.c»)

DIP Switch - Config

```
void DIPSW_Config(int config){
    switch(config){
        case DIPSW_ENABLE_MSB_CMD:
            DIPSW_ENABLE_MSB_PIN = 0;
            DIPSW_ENABLE_LSB_PIN = 1;
            break;
        case DIPSW_ENABLE_LSB_CMD:
            DIPSW_ENABLE_MSB_PIN = 1;
            DIPSW_ENABLE_LSB_PIN = 0;
            break;
        case DIPSW_DISABLE_CMD:
            DIPSW_ENABLE_LSB_PIN = 1;
            DIPSW_ENABLE_MSB_PIN = 1;
            break;
    }
}
```

DIP Switch - Read

```
58  unsigned char DIPSW_Read(void)
59  {
60      unsigned char rawValue = 0;
61      unsigned char lsbValue = 0;
62      unsigned char msbValue = 0;
63      unsigned char newValue = 0;
64
65      /*...*/
73
74      DIPSW_Config(DIPSW_ENABLE_LSB_CMD); //remember that DIP SW is shared
75      rawValue = (PORTB & KEYBOARD_MASK) >> 10;  //construct LSB nibble
76      lsbValue = DipSWkeyTable[rawValue];
77
78      DIPSW_Config(DIPSW_ENABLE_MSB_CMD);
79      rawValue = (PORTB & KEYBOARD_MASK) >> 10;  //construct MSB nibble
80      msbValue = DipSWkeyTable[rawValue];
81
82      DIPSW_Config(DIPSW_DISABLE_CMD);
83
84      newValue  = (newValue | msbValue) << 4;  //final shifting work
85      newValue |= lsbValue;
86
87      return newValue;    //done!
88  }
```

Implementación

```
if((DIPSW_Read() & 0x01) == 0x01){ LED_Set(1,ON); }

while(DIPSW_Read() > 120){ dato++; }

unsigned char variable = 0;

variable = DIPSW_Read();
```

Operación de lectura (MSB)

PORTB:

1	0	1	0	1	1	1	1	0	0	0	1	0	1	0	0

DIP + Teclas

Aplicamos una máscara

PORTB:

0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0

Y construímos un byte (>>10), al terminar lo guardamos en una variable

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

Operación de lectura (LSB)

PORTB:

1	1	0	1	0	1	0	1	1	0	1	1	0	1	0	1

DIP + Teclas

Aplicamos una máscara

PORTB:

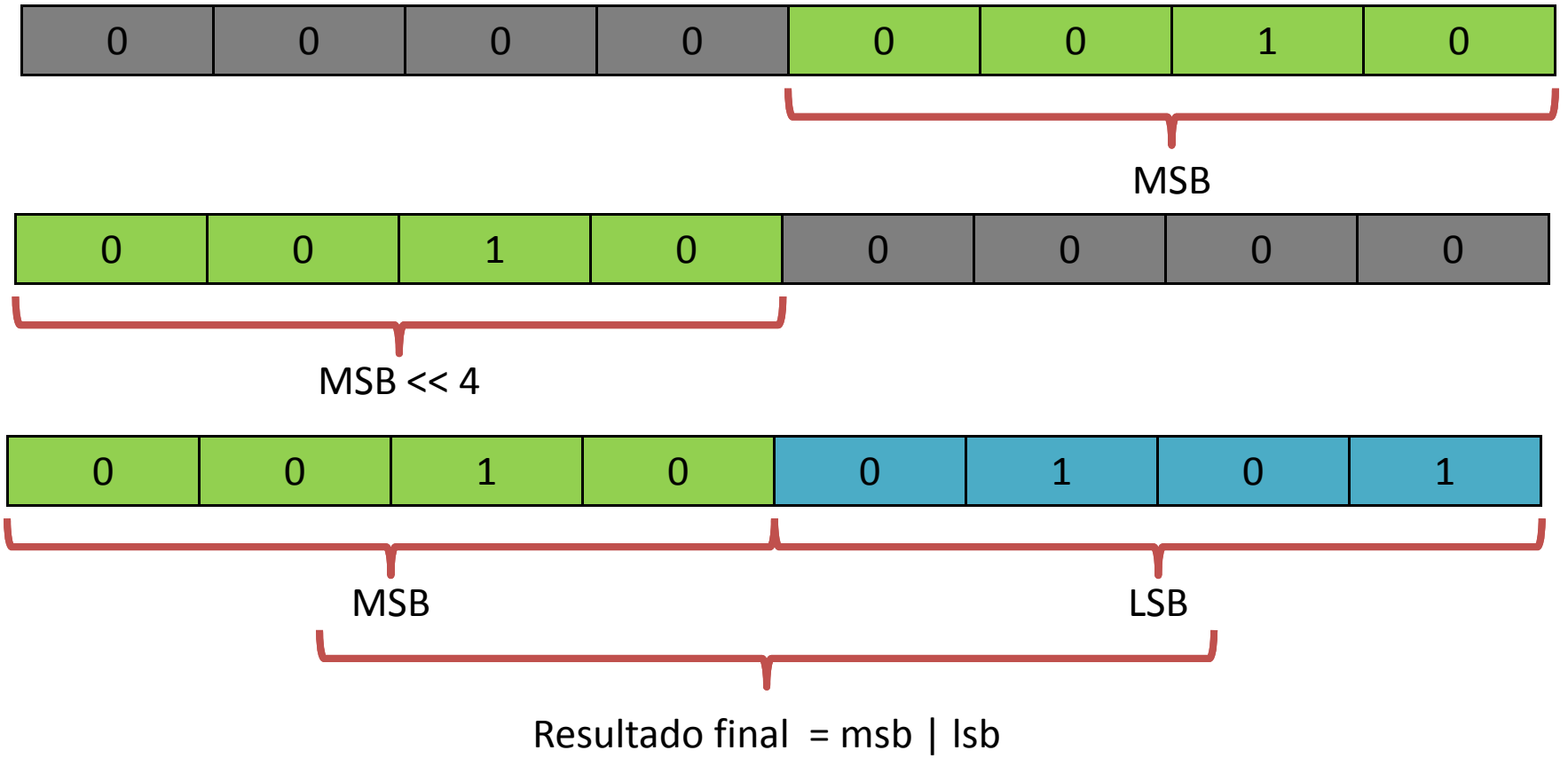
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0

Y construímos un byte (>>10), al terminar lo guardamos en una variable

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Operación final

Return var:



Keyboard - Read

```
135  unsigned char SW_Read(void)
136  {
137      static unsigned char pressedKey  = 0;
138      static unsigned char lastPressed = 0;
139      unsigned char        portData    = 0;
140      int                  i, scannedKey  = 0;
141
142      portData = (PORTB & KEYBOARD_MASK) >> 10; //SWITCH1 will be read first
143
144      for(i=0 ; i<4 ; i++){
145          if(!(portData & (1<<i))){   //check if bit is not set
146              pressedKey = 4-i;       //reverse logic
147              scannedKey = i;
148              break;
149          }
150      }
151
152      if(pressedKey){
153          delayms(1);                                 //debounce delay
154          portData = (PORTB & KEYBOARD_MASK) >> 10;   //read PORT *again*
155          if((portData & (1<<scannedKey))){           //check if bit is set
156              pressedKey = 0;                         //SW's are active low
157          }
158      }
159
160      if(pressedKey){                         //pressed key?
161          if(pressedKey == lastPressed){      //same as before?
162              return 0;                       //not good
163          }else{
164              lastPressed = pressedKey;       //update lastpressed
165          }
166      }else if(lastPressed != 0){
167          lastPressed = 0;
168      }
169
170      return pressedKey;  //done!
171  }
```

Input - Read

```
135  unsigned char IN_Read(void)
136  {
137      static unsigned char activeInput = 0;
138      static unsigned char lastActive  = 0;
139      unsigned char        portData    = 0;
140      int                  i,scannedIN   = 0;
141
142      portData = (PORTB & INPUT_MASK);     //INPUT1 will be read first
143
144      for(i=0 ; i<4 ; i++){
145          if(!(portData & (1<<i))){   //check if bit is not set
146              activeInput = i+1;
147              scannedIN = i;
148              break;
149          }
150      }
151
152      if(activeInput){
153          delayms(1);                      //debounce delay
154          portData = PORTB & INPUT_MASK;  //read PORT *again*
155          if((portData & (1<<scannedIN))){  //check if bit is set
156              activeInput = 0;             //IN's are active low (for this function)
157          }
158      }
159
160      if(activeInput){                     //active input?
161          if(activeInput == lastActive){  //same as before?
162              return 0;                    //not good
163          }else{
164              lastActive = activeInput;   //update lastactive
165          }
166      }else if(lastActive != 0){
167          lastActive = 0;
168      }
169
170      return activeInput;  //done!
171  }
```

Lectura de puertos

```
/*Port functions*/
////////////////////////////////////////////////////////////////////////////////
unsigned char SW_ReadPort(void)
{
    //returns raw port data (ex.0x08)
    //inputs are active 0!
    unsigned char keybPortValue = 0;
    unsigned char portData      = 0;

    portData = (PORTB & 0x3C00) >> 10;
    keybPortValue = DipSWkeyTable[portData];

    return keybPortValue;
}

unsigned char IN_ReadPort(void)
{
    //returns raw port data (ex.0x04)
    //inputs are active 0!
    unsigned char inputPortValue = 0;

    inputPortValue  = PORTB & 0x000F;
    inputPortValue ^= 0xFF; //user should mask when comparing

    return inputPortValue;
}
////////////////////////////////////////////////////////////////////////////////
```

Implementación

```
if((SW_ReadPort() & 0x01) == 0x01){ RS232_Send(0x01); }
if((IN_ReadPort() & 0x0F) == 0x0F){ RS232_Send(0x0F); }

unsigned char tecla   = 0;
unsigned char entrada = 0;

tecla   = SW_Read();
entrada = IN_Read();

if(tecla == 1){
    RLY_Set(1,ON);
}
```

Salidas («outputs.h»)

Librerías HammerHead

```
8  #ifndef OUTPUTS_H
9  #define OUTPUTS_H
10
11 #define PWM_CONV_FACTOR 16
12
13
14 BOOL        RLY_Get              (int num);
15 void        RLY_Set              (int num, BOOL state);
16 BOOL        OC_Get               (int num);
17 void        OC_Set               (int num, BOOL state);
18 void        OC_PWMSet            (int period, int duty_cycle);
19 void        OC_PWMChannelConfig  (int num);
20 BOOL        LED_Get              (int num);
21 void        LED_Set              (int num, BOOL state);
22 void        BUZZER_On            (int period, int duty_cycle);
23 inline void BUZZER_Off           (void);
24
25 #endif  /* OUTPUTS_H */
26
27
```

Salidas («outputs.c»)

Relays – Set() & Get()

```
41  /*Relays*/
42  ////////////////////////////////////////////////////////////////////////////
43  BOOL RLY_Get(int num)
44  {
45      unsigned char portData = 0;
46
47      if(num > 4 || !num){ return FALSE; }
48
49      portData   = (PORTD & (1<<(8-num)));
50      portData >>= (8-num);    //convert to BOOL
51
52      return portData;
53  }
54
55  void RLY_Set(int num, BOOL state)
56  {
57      if(num > 4 || !num){ return; }
58
59      switch(num){
60          case 1: if(state){ RLY1_SET = 1; }
61                  else     { RLY1_SET = 0; }
62                  break;
63          case 2: if(state){ RLY2_SET = 1; }
64                  else     { RLY2_SET = 0; }
65                  break;
66          case 3: if(state){ RLY3_SET = 1; }
67                  else     { RLY3_SET = 0; }
68                  break;
69          case 4: if(state){ RLY4_SET = 1; }
70                  else     { RLY4_SET = 0; }
71                  break;
72      }
73  }
```

OC – Set() & Get()

```
76  /*OpenCollector*/
77  ////////////////////////////////////////////////////////////////////////////
78  BOOL OC_Get(int num)
79  {
80      unsigned char portData = 0;
81
82      if(num > 4 || !num){ return FALSE; }
83
84      portData   = (PORTD & (1<<(4-num)));
85      portData >>= (4-num);   //convert to BOOL
86
87      return portData;
88  }
89
90  void OC_Set(int num, BOOL state)
91  {
92      if(num > 4 || !num){ return; }
93
94      switch(num){
95          case 1: if(state){ OC1_SET = 1; }
96                  else     { OC1_SET = 0; }
97                  break;
98          case 2: if(state){ OC2_SET = 1; }
99                  else     { OC2_SET = 0; }
100                 break;
101         case 3: if(state){ OC3_SET = 1; }
102                 else     { OC3_SET = 0; }
103                 break;
104         case 4: if(state){ OC4_SET = 1; }
105                 else     { OC4_SET = 0; }
106                 break;
107     }
108 }
```

OC –PWMChannelConfig()

```
134  void OC_PWMChannelConfig(int num)
135  {
136      /*  OpenCollector1 is RP22
137       *  OpenCollector2 is RP23
138       *  OpenCollector3 is RP24
139       *  OpenCollector4 is RP11
140       */
141      CloseOC3();
142
143      if(num > 4){ return; } //we only have 4 outputs
144
145      PPSUnLock;       //unlock remappable pins
146      switch(num){
147          case 1: PPSOutput(PPS_RP22, PPS_OC3); break;//remap RP22 for OC3 PWM
148          case 2: PPSOutput(PPS_RP23, PPS_OC3); break;//remap RP23 for OC3 PWM
149          case 3: PPSOutput(PPS_RP24, PPS_OC3); break;//remap RP24 for OC3 PWM
150          case 4: PPSOutput(PPS_RP11, PPS_OC3); break;//remap RP11 for OC3 PWM
151      }
152      CloseOC3();                 //turn it off
153      ConfigIntOC3(OC_INT_OFF);   //no interrupt
154      PPSLock;                    //lock remappable pins
155  }
156  ////////////////////////////////////////////////////////////////////////////
```

OC – PWMSet()

```
110  void OC_PWMSet(int period, int duty_cycle)
111  {
112      //period is given in microSeconds (uS)
113      UINT32 duty        = 0;
114      UINT16 fixedPeriod = 0;
115      UINT16 fixedDuty   = 0;
116
117      if(period < 8000){
118          fixedDuty   = 100 - duty_cycle;
119          fixedPeriod = (period * PWM_CONV_FACTOR);
120      }else{
121          fixedDuty   = 50;
122          fixedPeriod = 1000;
123      }
124
125      CloseOC3();
126      if(fixedDuty && fixedPeriod){
127          duty  = fixedPeriod;
128          duty *= fixedDuty;
129          duty /= 100;
130          OpenOC3(OC_SYSCLK_SRC  | OC_CONTINUE_PULSE, OC_SYNC_TRIG_IN_CURR_OC, fixedPeriod, duty);
131      }
132  }
```

LEDs – Set() & Get()

```
158  /*LEDport*/
159  ////////////////////////////////////////////////////////////////////////////////
160  BOOL LED_Get(int num)
161  {
162      unsigned char portData = 0;
163
164      if(num > 8 || !num){ return FALSE; }
165
166      portData  = (PORTE & (1<<(num-1)));
167      portData >>= num-1;
168
169      /*Since LEDs are sinked, we have to deal with inverse logic.
170      * This XOR is used to compensate that*/
171      portData ^= 1;
172
173      return portData;     //1=LEDx is off, 0=LEDx is on.
174  }
175
176  void LED_Set(int num, BOOL state)
177  {
178      if(num > 8 || !num){ return; }
179
180      switch(num){
181          case 1: if(state){ LED1_SET = 0; }
182                  else     { LED1_SET = 1; }
183                  break;
184          case 2: if(state){ LED2_SET = 0; }
185                  else     { LED2_SET = 1; }
186                  break;
187          case 7: if(state){ LED7_SET = 0; }
188                  else     { LED7_SET = 1; }
189                  break;
190          case 8: if(state){ LED8_SET = 0; }
191                  else     { LED8_SET = 1; }
192                  break;
193      }
194  }
```

Implementación

```
if(LED_Get(1) == OFF){ LED_Set(1,ON);}
if(OC_Get(4)  == ON) { OC_Set(4,OFF);}

for(i=1 ; i<=4 ; i++){
        if(RLY_Get(i) == OFF){
                RLY_Set(i,ON);
        }else{
                RLY_Set(i,OFF);
        }
}

OC_PWMChannelConfig(2);
OC_PWMSet(200,50);    //200uS( 5kHz) period @ 50% duty
OC_PWMSet(100,75);    //100uS(10kHz) period @ 75% duty
```

BUZZER – On() & Off()

```
209  /*Buzzer*/
210  ////////////////////////////////////////////////////////////////////////////
211  void BUZZER_On(int period, int duty_cycle)
212  {
213      //period is given in microSeconds (uS)
214      UINT32 duty = 0;
215      UINT16 fixedPeriod = 0;
216
217      if(period < 8000){
218          fixedPeriod = (period * PWM_CONV_FACTOR);
219      }else{
220          fixedPeriod = 1000;
221      }
222
223      CloseOC2();
224      if(duty_cycle && fixedPeriod){
225          duty  = fixedPeriod;
226          duty *= duty_cycle;
227          duty /= 100;
228          OpenOC2(OC_SYSCLK_SRC  | OC_CONTINUE_PULSE, OC_SYNC_TRIG_IN_CURR_OC, fixedPeriod, duty);
229      }
230  }
231
232  inline void BUZZER_Off(void)
233  {
234      CloseOC3();
235  }
236  ////////////////////////////////////////////////////////////////////////////
```

Implementación

```
BUZZER_On(333,5);      //333uS( 3kHz) period @  5% duty
BUZZER_On(200,50);     //200uS( 5kHz) period @ 50% duty
BUZZER_On(100,75);     //100uS(10kHz) period @ 75% duty



BUZZER_Off();
```

Analógico («analog.h»)

Librerías HammerHead

```
29
30  #ifndef ANALOG_H
31  #define ANALOG_H
32
33  #define ADC_MAX_VALUE    0x03F6
34  #define ADC_MIN_VALUE    0x00
35
36  /* Function prototypes*/
37  void ADC_Init    (void);
38  int  ADC_Read    (void);
39  int  ADC_Range   (int min_val, int max_val);
40
41  #endif  /* ANALOG_H */
42
```

Analógico («analog.c»)

ADC – Init() & Read()

```
35  ////////////////////////////////////
36  void ADC_Init(void)
37  {
38      AD1PCFG           = 0x7FFF;  // select RB15(AN15) as analog input
39      AD1CON1           = 0x00E0;  // internal counter ends sampling and starts conversion
40      AD1CON2           = 0;       // use AVDD, AVSS as reference pins
41      AD1CON3           = 0x1F02;  // Tsamp = 32 x Tad; Tad=125ns
42      AD1CSSL           = 0;       // no scanning required
43      AD1CON1bits.ADON = 1;        // turn on the ADC
44  }
45
46  int ADC_Read(void)
47  {
48      AD1CHS            = 15;      // 1. select analog input channel (15 is POT)
49      AD1CON1bits.SAMP = 1;        // 2. start sampling
50      while (!AD1CON1bits.DONE);   // 3. wait for the conversion to complete
51      return ADC1BUF0;             // 4. read the conversion result
52  }
53
```

ADC – Range()

```
54  int ADC_Range(int min_val, int max_val)
55  {
56      //not very accurate, but still works well
57      float x           = 0.0;
58      float preResult   = 0.0;
59      int   adcReading  = 0;
60      int   finalResult = 0;
61
62      if(min_val > max_val || max_val > ADC_MAX_VALUE){
63          return 0;
64      } //min cannot be > than max and max cannot be > than physical max value
65
66      adcReading = ADC_Read();
67
68      x = ADC_MAX_VALUE / (max_val - min_val);
69
70      preResult  = (adcReading / x);
71      preResult += min_val;
72
73      finalResult = (int)(preResult);
74
75      return finalResult;
76  }
77
```

Implementación

```
ADC_Init();
OC_PWMChannelConfig(1);


int variable = 0;

variable = ADC_Range(5,95);

OC_PWMSet(400,variable);
```

EEPROM(«EEPROM.h»)

Librerías HammerHead

```
7
8   #ifndef EEPROM_H
9   #define EEPROM_H
10
11  //refer to 24LC01B datasheet
12  #define EEPROM_SIZE          0x7F
13  #define EEPROM_WRITE_CMD     0xA0
14  #define EEPROM_READ_CMD      0xA1
15
16  #define ACK_EEPROM           TRUE
17  #define DONT_ACK_EEPROM      FALSE
18
19  /* Function prototypes*/
20  unsigned char    EEPROM_Read (unsigned char address);
21  BOOL             EEPROM_Write(unsigned char address, unsigned char new_data);
22
23  #endif  /* EEPROM_H */
24
```

EEPROM(«EEPROM.c»)

EEPROM – Read()

```
36  ////////////////////////////////////////
37  unsigned char EEPROM_Read(unsigned char address)
38  {
39      unsigned char eeData=0;
40
41      if(address > EEPROM_SIZE){ return -1; }
42
43      IO_I2Cstart();
44      IO_I2CsendByte(EEPROM_WRITE_CMD);
45      IO_I2CsendByte(address);
46      IO_I2Cstop();
47
48      IO_I2Cstart();
49      IO_I2CsendByte(EEPROM_READ_CMD);
50      //let's see what the EEPROM sends us
51      eeData = IO_I2CreadByte(DONT_ACK_EEPROM);
52      //Send NACK
53      IO_I2CsendNACK();
54      IO_I2Cstop();
55
56      return eeData;
57  }
58
```

EEPROM – Write()

```
59  BOOL EEPROM_Write(unsigned char address, unsigned char new_data)
60  {
61      unsigned char verifyWrite = 0;
62
63      //we limit (kind of) the maximum EEPROM address
64      address &= EEPROM_SIZE;
65
66      IO_I2Cstart();
67      IO_I2CsendByte(EEPROM_WRITE_CMD);
68      IO_I2CsendByte(address);
69      IO_I2CsendByte(new_data);
70      IO_I2Cstop();
71
72      delayus(10);    //wait a bit
73
74      //now, lets verify the data we've written
75
76      verifyWrite = EEPROM_Read(address);
77
78      if(verifyWrite == new_data){
79          return TRUE;    //write operation succeeded
80      }else{
81          return FALSE;   //write operation failed
82      }
83  }
```

RS232(«RS232.h»)

Librerías HammerHead

```
8  #ifndef RS232_H
9  #define RS232_H
10
11 //refer to MCU datasheet for these values
12 //or Microchip UART document
13 #define BAUD115200 8
14 #define BAUD56000  17
15 #define BAUD38400  25
16 #define BAUD19200  51
17 #define BAUD9600   103
18
19 void          RS232_Init     (int baud);
20 void          RS232_Send     (unsigned int data);
21 void          RS232_PutStr   (unsigned int *string);
22 unsigned int  RS232_Receive  (void);
23
24 #endif  /* RS232_H */
25
```

RS232(«RS232.c»)

RS232 – Init()

```
39  ////////////////////////////////////////
40  void RS232_Init(int baud)
41  {
42      //low baudrate, 8bit comm, no parity, 1 stop bit
43      PPSUnLock;  //unlock register
44
45      RPINR19bits.U2RXR = 8;  //assign U2RX to pin RP8
46      RPOR4bits.RP9R    = 5;  //assign U2TX to pin RP9
47
48      PPSLock;    //lock register to avoid further writes
49      CloseUART2();
50
51      IFS1bits.U2RXIF  = 0;
52      IEC1bits.U2RXIE  = 1;   //enable only RX interrupt
53      IPC7bits.U2RXIP2 = 0;
54      IPC7bits.U2RXIP1 = 1;
55      IPC7bits.U2RXIP0 = 0;
56
57      U2MODE = 0x8000;
58
59      U2STA  = 0x2400;
60      U2BRG  = baud;
61  }
62
```

RS232 – Send(), PutStr() & Read()

```
63  void RS232_Send(unsigned int data)
64  {
65      WriteUART2(data);
66  }
67
68  void RS232_PutStr(unsigned int *string)
69  {
70      putsUART2((unsigned int*)string);
71  }
72
73  unsigned int RS232_Read(void)
74  {
75      unsigned int inputData = 0;
76
77      inputData = ReadUART2();
78
79      return inputData;
80  }
```

RS232 – _U2RXInterrupt()

```
82  /* RS232 Receive interrupt*/
83  void __attribute__((interrupt,__no_auto_psv__)) _U2RXInterrupt(void)
84  {
85      unsigned char rxData = 0;
86
87      rxData = ReadUART2();       //get received data
88      RS232_Send(rxData);         //echo it
89
90      LED8_SET = !LED8_SET;       //toggle LED1 using the global macro
91
92      IFS1bits.U2RXIF = 0;        //clear the interrupt status of UART2 RX
93  }
```

Implementación

```
RS232_Init();

RS232_Send(0xFF);
RS232_Send(0xAA);
RS232_Send(23);

RS232_PutStr("Vinagron Digital is the best");
RS232_PutStr("Aguijon; introduccion basica");

int variable = 0;

variable = RS232_Read();
```

LCD(«LCD.h»)

Librerías HammerHead

```
#ifndef LCD_H
#define LCD_H


#define LCD_I2C_ADDR    0x78    //LCD's address on the I2C bus
#define LCD_DATA_CMD    0x40    //data write command
#define LCD_CMD_CMD     0x00    //command write command

#define BLIGHT_LVL_ON   0xFF    //no PWM'd backlight
#define BLIGHT_LVL_OFF  0x00    //no PWM'd or full on backlight
#define BLIGHT_LVL_4    2000    //highest intensity 100%
#define BLIGHT_LVL_3    4000    //..                 75%
#define BLIGHT_LVL_2    6000    //..                 50%
#define BLIGHT_LVL_1    7500    //lowest intensity   25%

void LCD_Data           (unsigned char data);
void LCD_StoreCustomChar(unsigned char *rows, unsigned char cgram_loc);
void LCD_Clear          (void);
void LCD_GotoXY         (int x, int y);
void LCD_GotoYX         (int y, int x);
void LCD_SetCursorHome  (void);
void LCD_SetCursorBlink (BOOL enabled);
void LCD_PutStr         (int y, int x, char *msg, BOOL clear);
void LCD_SetBacklight   (int level);
void LCD_Init1          (void);
void LCD_Init2          (void);
void LCD_IntroAnimation (void);


#endif  /* LCD_H */

```

LCD(«LCD.c»)

LCD – Init1()

```
190  void LCD_Init1(void)//initialization for ST7036i (LCD driver)
191  {
192      LCD_RESET_PIN = 1;
193      delayms(80);
194
195      IO_I2Cstart();
196      IO_I2CsendByte(LCD_I2C_ADDR);
197      IO_I2CsendByte(LCD_CMD_CMD);
198      IO_I2CsendByte(0x38);
199      delayms(60);
200      IO_I2CsendByte(0x39);//8bits,line 2,instruction table 01
201      delayms(60);
202      IO_I2CsendByte(0x14);//IS1: 1/5 bias,FX not fixed
203      IO_I2CsendByte(0x70);//IS1: Contrast set for internal follower mode C3:C0 = 0
204      IO_I2CsendByte(0x5E);//IS1: ICON display on, Booster on,Contrast C5,C4=1,0
205      IO_I2CsendByte(0x6D);//IS1: Follower circuit on, follower circuit ratio
206      IO_I2CsendByte(0x0C);//Display on, cursor off,cursor position off
207      IO_I2CsendByte(0x01);//Clear display
208      IO_I2CsendByte(0x06);//Cursor direction, no display shif
209      delayms(60);
210      IO_I2Cstop();
211  }
```

LCD – StoreCustomChar()

```
72  static const char LCDcustomChar[] = {0x09, 0x15, 0x1D, 0x15, 0x15, 0x00, 0x00, 0x00};
73
74  void LCD_StoreCustomChar(unsigned char *rows, unsigned char cgram_loc)
75  {
76      int i;
77
78      if(cgram_loc>=8) return; //we only have 8 spaces at CGRAM
79
80      // Set up the display to write into CGRAM - configure LCD to use func table 0
81      IO_I2Cstart();
82      IO_I2CsendByte(LCD_I2C_ADDR);      //address
83      IO_I2CsendByte(LCD_CMD_CMD);       //command
84      IO_I2CsendByte(0x38);              //set to func table 0
85      delayms(5);                          //wait for the LCD
86      //Set CGRAM position to write
87      IO_I2CsendByte( 0x40 + (cgram_loc*8));
88      IO_I2Cstop();
89
90      IO_I2Cstart();
91      IO_I2CsendByte(0x78);
92      IO_I2CsendByte(0x40);     //RS=1 (write data to ram)
93
94      //send the 8 rows of the character
95      for(i=0;i<8;i++){
96          IO_I2CsendByte(*rows); // write the current pointed row to CGRAM
97          rows++;              //increment the pointer
98      }
99
100     IO_I2Cstop();
101     delayms(1);
102
103     // Leave the LCD as it was - function table 1 DDRAM and set the cursor
104     LCD_Command(0x39);
105
106     delayms(1);
107 }
```

LCD – PutStr()

```
156  void LCD_PutStr(int y, int x, char *msg, BOOL clear)
157  {
158      if(clear){
159          LCD_Clear();
160          delayms(1);
161      }
162
163      LCD_GotoYX(y,x);
164
165      IO_I2Cstart();
166      IO_I2CsendByte(LCD_I2C_ADDR);
167      IO_I2CsendByte(LCD_DATA_CMD);
168
169      while(*msg){
170          IO_I2CsendByte(*msg);
171          msg++;
172      }
173      IO_I2Cstop();
174  }
```

LCD – SetBacklight()

```
176  void LCD_SetBacklight(int level)
177  {
178      if(level != BLIGHT_LVL_OFF && level != BLIGHT_LVL_ON){
179          CloseOC1();
180          if(level){
181              OpenOC1(OC_SYSCLK_SRC  | OC_CONTINUE_PULSE, OC_SYNC_TRIG_IN_CURR_OC, 8000, level);
182          }
183      }else if(level == BLIGHT_LVL_ON){
184          CloseOC1();
185          LCD_BLIGHT_PIN = 1;
186      }else{
187          CloseOC1();
188          LCD_BLIGHT_PIN = 0;
189      }
190  }
```

Implementación

```
IO_I2Cinit();
LCD_Init1();
LCD_SetBacklight(BLIGHT_LVL_4);
LCD_SetBacklight(BLIGHT_LVL_1);

const char VDcustomChar1[] = {0x04, 0x0C, 0x1C, 0x14,
                              0x1C, 0x0C, 0x0E, 0x0F };

LCD_StoreCustomChar(VDcustomChar1,1);

LCD_PutStr(1,0,"Vinagron Digital",TRUE);
LCD_PutStr(2,0,"HammerHead v1.0" ,FALSE);
```

IO(«IO.h»)

Librerías HammerHead

```
8  #ifndef GPIO_H
9  #define GPIO_H
10
11 void IO_Init          (void);
12 void IO_PWMinit       (void);
13 void IO_SPIinit       (void);
14 void IO_SPIsendByte   (BYTE data);
15 BYTE IO_SPIreadByte   (void);
16 void IO_I2Cinit       (void);
17 void IO_I2Cstart      (void);
18 void IO_I2Cstop       (void);
19 void IO_I2CsendByte   (BYTE data);
20 BYTE IO_I2CreadByte   (BOOL ack);
21 void IO_I2CsendNACK   (void);
22
23 #endif  /* GPIO_H */
24
```

IO(«IO.c»)

IO – Init()

```
39  /*GPIO*/
40  ////////////////////////////////////////////////////////////////////////////
41  void IO_Init(void)
42  {
43      TRISB = 0xBDCF;             //configure PORTB
44      LATB  = 0x0000;
45
46      TRISD = 0x0000;             //configure PORTD as outputs
47      LATD  = 0x0000;
48
49      TRISE = 0x0000;             //configure PORTE as outputs
50      LATE  = 0x0000;
51
52      TRISG = 0x0000;             //configure PORTG as outputs
53      LATG  = 0x0000;
54
55      TRISF = 0x0000;             //configure PORTF as outputs
56      LATF  = 0x0000;
57  }
```

BSP(«BSP.h»)

Librerías HammerHead

```
11  #define ON     TRUE
12  #define OFF    FALSE
13
14  //opto isolated inputs
15  #define INPUT1          PORTBbits.RB0
16  #define INPUT2          PORTBbits.RB1
17  #define INPUT3          PORTBbits.RB2
18  #define INPUT4          PORTBbits.RB3
19
20  //On board switches & DIP Switch (shared)
21  #define SWITCH1         PORTBbits.RB13  //SWITCH 1 is shared with DIP 4 & 8
22  #define SWITCH2         PORTBbits.RB12  //SWITCH 2 is shared with DIP 3 & 7
23  #define SWITCH3         PORTBbits.RB11  //SWITCH 3 is shared with DIP 2 & 6
24  #define SWITCH4         PORTBbits.RB10  //SWITCH 4 is shared with DIP 1 & 5
25
26  #define DIPSW_KEY_1_5   PORTBbits.RB10  //DIP SW keys 1 & 5 are shared
27  #define DIPSW_KEY_2_6   PORTBbits.RB11  //DIP SW keys 2 & 6 are shared
28  #define DIPSW_KEY_3_7   PORTBbits.RB12  //DIP SW keys 3 & 7 are shared
29  #define DIPSW_KEY_4_8   PORTBbits.RB13  //DIP SW keys 4 & 8 are shared
30
```

Aguijón («aguijon.h»)

Librerías HammerHead

```
#ifndef AGUIJON_H
#define AGUIJON_H

/* Includes Section*/

//Standard C libraries
#include <stdio.h>              //standard C I/O
#include <stdlib.h>             //standard C lib

//Microchip libraries
#include <p24FJ128GB106.h>      //PIC24F MCU header file
#include <GenericTypeDefs.h>    //support for various data types

#include "BSP.h"                //VD Aguijon Board Support Package

#define FCY 16000000UL              //16MHz is our work frequency
#include <libpic30.h>               //MCHP delay Library
#define delayms(x)   __delay_ms(x); //global delayms function
#define delayus(x)   __delay_us(x); //global delayus function

#define HAMMERHEAD_VERSION "1.0"

#endif  /* AGUIJON_H */
```

Sumario

- *Aguijón:*
  - *Cuenta con los módulos necesarios para una diversidad de proyectos.*
  - *Microcontrolador de 16 bits.*
  - *Reprogramable vía USB.*
  - *Expandible.*
  - *Puntas de prueba facilitan la visualización de protocolos.*
- *Librerías de software:*
  - *Facilitan la implementación de diversos bloques funcionales.*
  - *Open Source.*

¡Gracias!

Hojas de retroalimentación