Antonio Molina Cortés – Salvador Moreno Gomez

Arpa Laser

Indice

Contenido

1-Hardware

Vamos a ver toda la parte tangible del arpa.

1.1-Electrónica

Dentro de este apartado vamos a estudiar todo lo relacionado con el entorno electrónico del arpa.

1.1.1-Etapas de potencia

Dentro del arpa hay dos etapas de potencia: Etapa de potencia para los fotoemisores y etapa de potencia para todo lo demás

La etapa de potencia para los fotoemisores está hecha con un LM7805 ya que estos normalmente van a 4.5V de tensión. Se podría haber colocado un L200 pero para regularle la intensidad máxima al circuito necesitaría una resistencia de alta potencia, los cual haría que se calentara el interior de la carcasa de la CPU. Plano …..

La etapa de potencia para los fotosensores está hecha con un LM317 ya que estos pueden trabajar a un mínimo de 3.5V y un máximo de 18V. Se intentará ajustar el regulador al mínimo para que el pic diferencie entre entrada en alto o en bajo, y al mismo tiempo conseguir que la tensión en los transistores sea la mínima posible. Plano…..

1.1.2-Etapa de regulación

Esta etapa está diseñada para regular la tensión que le llega a cada patilla del pic dedicada a cada transistor. Se ha hecho con una resistencia de 82Ω 1/4W y un potenciómetro de 470Ω 1/4W en serie, el potenciómetro regula en gran medida la tensión y la resistencia evita que, por mucho que se baje la resistencia del potenciómetro, nunca se llegue a 0Ω en la rama, lo que provocaría la rotura del transistor. Plano…..

1.1.3-Etapa de ensamblaje

Esta etapa se encarga de conectar nuestra placa madre a los fotosensores y a la placa portadora del pic, está compuesta de tiras de pines.

1.1.4-Fotorreceptores

Los fotorreceptores son 12, pero es el mismo circuito replicado: un fotodiodo BPW34 con el cátodo conectado al colector del transistor BC547C y el ánodo conectado a la base del transistor.

Se planteó usar fototransitores comerciales pero encarecían mucho el precio.

1.1.5-Fotoemisores

Los fotoemisores son 12, pero es el mismo circuito replicado: un diodo laser y una resistencia en serie de 68Ω 1/4W en serie para hacer de freno electrónico. Los diodos laser han sido extraidos de punteros laser comerciales de color rojo.

Se planteó el poner laser de color verde, pero para que sean visibles usan más potencia, lo cual los hacen más peligrosos para el uso sin protección ocular.

También se planteó el uso de diodos laser comerciales pero salían muy caros.

1.1.6-Pedal

El pedal se ha construido en una caja de mecanismos de electricidad: la caja hace de chasis del pedal y la tapa, que la hemos articulado con muelles, es la parte móvil. El circuito electrónico consta de una pulsador robusto (tiene que aguantar el maltrato de los pisotones de la gente) y un cable con un jack para conectarlo a la placa.

1.2-Estructura y Pintura

En esta sección vamos a explicar cómo se construyó el chasis del arpa.

El chasis del arpa está costruida con tubos de PVC de 50mm de diámetro esterno en general y codos de 90°.

1.2.1-Soportes de fotorreceptores y fotoemisores

Los dos tubos que constituyen los dos soportes están hechos de la misma manera.

1. Se le traza una línea longitudinal a lo largo del tubo.
2. Se escoge el final que esté mejor acabado para ser el que se vea, el otro irá insertado en un codo, por lo tanto, no se verá.
3. Desde el final “bueno” vamos a trazar puntos cada 4.5cm (consideramos que una mano de adulto media mide entorno a 9cm en la parte más ancha, de esta manera los laseres están separaos entre si a 9cm, por lo cual no aseguramos que solo se corte uno si esa es la voluntad del que está tocando el arpa) .
4. Al tubo que va a alojar los fotoemisores se le practican agujeros en cada marca que le hicimos anteriormente con una broca de 9mm, al de los fotorreceptores se le hace agujeros en las marcas con una broca de 6mm. Hay que procurar que la broca entre lo más perpendicular posible a la superficie del tubo. Estos tubos no se van a pegar en los codos, ya que el arpa está diseñada para que se pueda desmontar.
5. El tubo se dispone de manera vertical tendrá 50cma este se le colocara un codo en cada punta con la misma orientación (este paso tiene que ser muy preciso).

6. Hay que practicarle una apertura de forma oval a uno de los codos, para crear así un punto de acceso para los cables desde los soportes hasta la CPU. Este codo se quedará para la parte inferior del arpa.

1.2.2-CPU

1. Al el tubo que va a estar dedicado a alojar la CPU, que será de 90mm de diámetro, hay que practicarle una apertura de forma oval en algún punto central del tubo.
2. En una de las dos tapas, que se dedicaran a tapar la CPU, la vamos a taladrar para ubicarle los leds de señalización, los jacks de alimentación y pedal y la clavija de USB.
3. Pegamos el tubo de la CPU al codo que taladramos haciendo coincidir los agujeros.

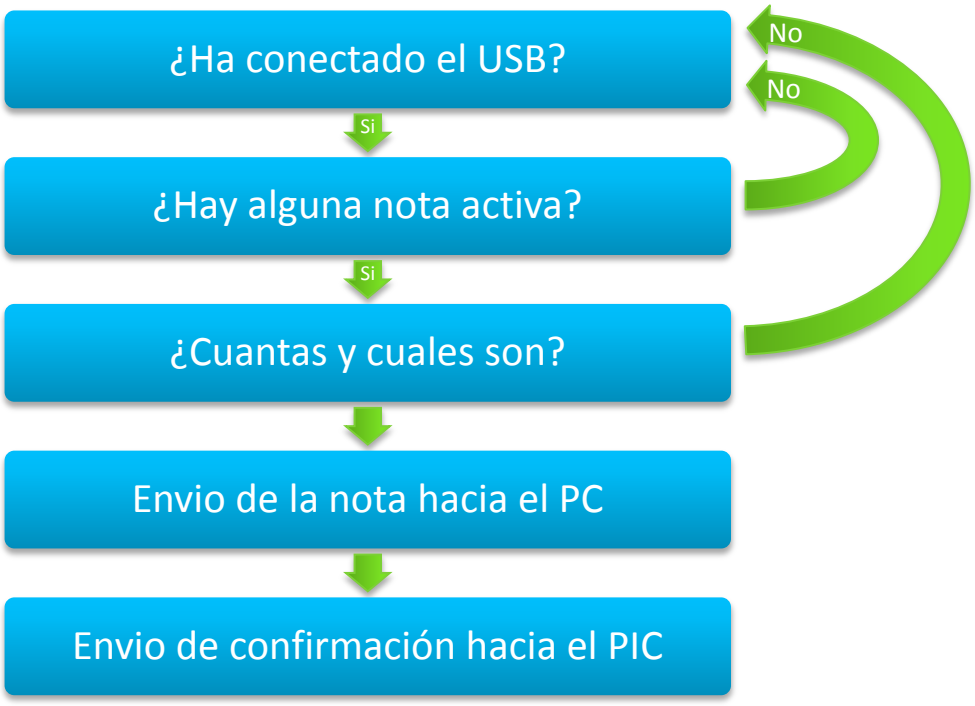
1.2.3-Protección de los fotorreceptores

Los protectores de los fotorreceptores están dedicados a proteger los fotorreceptores

1.2.4-Pintura

2-Software

2.1-Diagrama de flujo



2.2-Código de cabecera

```
////////////////////////////////////////////////////////////////////////
////
//// usb_kbd_HID.h
////
////////////////////////////////////////////////////////////////////////

#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

//////////////////////////////////////////////////////////////////
///
/// HID Report. Nos cuenta como se va a manejar la informacion
/// respecto a los drivers.
///
/// Configuramos los registros para una emulacion de un keyboard.
///
//////////////////////////////////////////////////////////////////

const char USB_CLASS_SPECIFIC_DESC[] = {
0x05, 0x01, //usage page (generic desktop) //0,1
0x09, 0x06, //uso (keyboard) //2,3
0xA1, 0x01, //coleccion (aplicacion) //4,5
0x85, 0x02, //report id (2) //6,7
0x05, 0x07, //usage page (codigo de teclas) //8,9
0x19, 0xE0, //uso min (224) //10,11
0x29, 0xE7, //uso max (231) //12,13
0x15, 0x00, //logica min (0) //14,15
0x25, 0x01, //logica max (1) //16,17
0x75, 0x01, //tamaño de report (1) //18,19
0x95, 0x08, //contador de report (8) //20, 21
0x81, 0x02, //entrada (data, variable, absolute) //22,23
0x95, 0x01, //contador de report (1) //24,25
0x75, 0x08, //tamaño de report (8) //26,27
0x81, 0x01, //entrada (constante) [Byte reservados] //28,29
0x95, 0x05, //contador de report (5) //30,31
0x75, 0x01, //tamaño de report (1) //32,33
0x05, 0x08, //usage page (apartado de leds) //34,35
0x19, 0x01, //uso min (1) //36,37
0x29, 0x05, //uso max (5) //38,39
0x91, 0x02, //salida (data, var, abs) [leds report] //40,41
0x95, 0x01, //contador de report (1) //42,43
```

```
0x75, 0x03, //tamaño de report (3) //44,45
0x91, 0x01, //salida (constante) [leds report padding] //46,47
0x95, 0x05, //contador de report (5) //48,49
0x75, 0x08, //tamaño de report (8) //50,51
0x15, 0x00, //logica min (0) //52,53
0x25, 0x65, //logica max (101) //54,55
0x05, 0x07, //usage page (codigo de teclas) //56,57
0x19, 0x00, //uso min (0) //58,59
0x29, 0x65, //uso max (101) //60,61
0x81, 0x00, //entrada (data, array) //62,63
0xC0 //fin de coleccion //64
};

//Si tienes un descriptor extra especifico que no esta en la configuracion,
// esta tabla de array define donde mirar para encontrar esa constante
// USB_CLASS_SPECIFIC_DESC[].
//El primer elemento es de configuracion (si es que tiene mas de una configuracion).
//el segundo numero es de la interfaz.
//se autoselecciona a 0xFFFF si es combinacion configuracion-interfaz no existe.
const int16 USB_CLASS_SPECIFIC_DESC_LOOKUP[USB_NUM_CONFIGURATIONS][1] =
{
//configuracion 1
//interfaz 0
0
};

//Si tienes un descriptor extra especifico que no esta en la configuracion,
// esta tabla de array define donde mirar para encontrar esa constante
//El primer elemento es de configuracion (si es que tiene mas de una configuracion).
//el segundo numero es de la interfaz.
//se autoselecciona a 0xFFFF si es combinacion configuracion-interfaz no existe.
const int16 USB_CLASS_SPECIFIC_DESC_LOOKUP_SIZE[USB_NUM_CONFIGURATIONS][2] =
{
//configuracion 1
//interfaz 0
sizeof(USB_CLASS_SPECIFIC_DESC)
};

////////////////////////////////////////////////////////////////
///
/// Comenzamos un nuevo descriptor de configuracion.
/// Ahora mismo solo tenemos un descriptor de configuracion.
/// la configuracion, interfaz, clase y endpoint van en este array.
///
////////////////////////////////////////////////////////////////
```

```
#DEFINE USB_TOTAL_CONFIG_LEN 41 //config+interfaz+class+endpoint

const char USB_CONFIG_DESC[] = {
//SEGUN LA CONFIGURACION EN WINDOWS, EL ORDEN DE ESTE ARRAY DEBE SER:
// config(s)
// interfaz(s)
// class(es)
// endpoint(s)

//config_descriptor para la configuracion del index 1
USB_DESC_CONFIG_LEN, //tamaño del descriptor ==0
USB_DESC_CONFIG_TYPE, //constante de CONFIGURACION (CONFIGURACION 0x02) ==1
USB_TOTAL_CONFIG_LEN,0, //tamaño total de toda esta configuracion ==2,3
1, //numero de interfaces que soporta nuestro periferico ==4
0x01, //identificador para esta configuracion. (si es que tenemos mas de una) ==5
0x00, //inicio del string del descriptor de esta configuracion ==6
0xC0, //bit 6=1 si esta autoalimentado por el usb, bit 5=1 si necesitamos una fuente externa,
bits 0-4 sin uso y bit7=1 ==7
0x32, //bus de maxima alimentacion requerida (maximo mA/2) (0x32 = 100mA) //8

//interfaz de descriptor 1 (RATON O MOUSE)
USB_DESC_INTERFACE_LEN, //tamaño del descriptor =9
USB_DESC_INTERFACE_TYPE, //constante INTERFAZ (INTERFAZ 0x04) =10
0x00, //numero que define a esta interfaz (si es que tenemos mas de una) ==11
0x00, //configuracion alternativa ==12
2, //numero de endpoints para esta interfaz //13
0x03, //ccodigo de clase, 03 = HID ==14
0x00, //codigo de subclase //boot ==15
0x00, //codigo de protocolo ==16
0x00, //inicio del string del descriptor de interfaz ==17

//class descriptor 1 (HID)
USB_DESC_CLASS_LEN, //tamaño del descriptor ==18
USB_DESC_CLASS_TYPE, //tipo de descriptor (0x21 == HID) ==19
0x00,0x01, //hid class release number (1.0) (try 1.10) ==20,21
0x00, //codigo del pais (0 = none) ==22
0x01, //numero de la clase de hid del descriptor (1) ==23
0x22, //tipo de report del descriptor (0x22 == HID) ==24
USB_CLASS_SPECIFIC_DESC_LOOKUP_SIZE[0][0], 0x00, //tamaño del report del descriptor
==25,26

//endpoint descriptor 1 ENTRADA
USB_DESC_ENDPOINT_LEN, //tamaño del descriptor ==27
USB_DESC_ENDPOINT_TYPE, //constante ENDPOINT (ENDPOINT 0x05) ==28
0x81, //endpoint numero y direccion (0x81 = EP1 IN) ==29
USB_ENDPOINT_TYPE_INTERRUPT, //tipo de transferencia soportada (0x03 es
interrumpida)==30
```

```
USB_EP1_TX_SIZE,0x00, //tamaño maximo del paquete soportado ==31,32
10 //intervalo de sondeo, en ms. (puede ser menor de 10 para perifericos de baja
velocidad)==33

//endpoint descriptor 1 SALIDA
USB_DESC_ENDPOINT_LEN, //tamaño del descriptor ==34
USB_DESC_ENDPOINT_TYPE, //constante ENDPOINT (ENDPOINT 0x05) ==35
0x01, //endpoint numero y direccion (0x01 = EP1 OUT) ==36
USB_ENDPOINT_TYPE_INTERRUPT, //tipo de transferencia soportada (0x03 es
interrumpida)==37
USB_EP1_RX_SIZE,0x00, //tamaño maximo de paquete soportado ==38,39
10 //intervalo de sondeo, en ms. (puede ser menor de 10 para perifericos de baja
velocidad)==40
};


//****** COMIENZA LA CONFIGURACION DE LA TABLA DE DESCRIPTORES ********
//desde que nosotros no podemos crear punteros a constantes en pics16s, esta es la tabla
para encontrar
// un descriptor especifico.

//NOTA: HAY UNA LIMITACION EN EL CODIGO CCS, TODOS LOS INTERFACES HID DEBEN
EMPEZAR EN 0 Y SER SECUENCIALES
// POR EJEMPLO, SI TU TIENES 2 INTERFACES HID DEBEN SER, LA INTERFAZ 0 Y LA INTERFAZ 1
#define USB_NUM_HID_INTERFACES 1

//el maximo numero de interfaces que hemos visto en la configuración
//por ejemplo, si la configuracion 1 tiene 1 interfaz y la configuracion 2 tiene 2 interfaces, tu
debes definirlo como 2
#define USB_MAX_NUM_INTERFACES 1

//define cuantos interfaces hay por configuracion. [0] es la primera configuracion, etc.
const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

//define donde encontrar la clase de descriptores
//la primera celda es el numero de configuracionfirst dimension is the config number
//la segunda celda especifica cual es la interfaz
//la ultima celda especifica que clase coger, pero debe haber solo una clase por interfaz
//si la clase de descriptor no es valida, lo selecciona a 0xFFFF
const int16 USB_CLASS_DESCRIPTORS[USB_NUM_CONFIGURATIONS][1][1]=
{
//configuracion 1
//interfaz 0
//clase 1
18
};
```

```
//****** FIN DE LA CONFIGURACION DE LA TABLA DE DESCRIPTORES ********

#if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
#error USB_TOTAL_CONFIG_LEN not defined correctly
#endif


////////////////////////////////////////////////////////////////
///
/// start device descriptors
///
////////////////////////////////////////////////////////////////

const char USB_DEVICE_DESC[] = {
//comienzo de la configuracion del periferico. solo es posible 1.
USB_DESC_DEVICE_LEN, //tamaño de este report ==1
0x01, //constante DEVICE (DEVICE 0x01) ==2
0x10,0x01, //usb version en bcd (pic167xx is 1.1) ==3,4
0x00, //codigo de clase ==5
0x00, //codigo de subclase ==6
0x00, //codigo de protocolo ==7
USB_MAX_EP0_PACKET_LENGTH, //maximo tamaño del paquete para endpoint 0. (baja
velocidad especifica 8) ==8
0x61,0x04, //vendor id (0x0461 es para MicroChip)
0x57,0x00, //product id ==11,12 //no usar ffff dice usb-by-example
0x00,0x01, //numero de version del dispositivo ==13,14
0x01, //inicio del string del descriptor de manufacturado. ==15
0x02, //inicio del string del descriptor del producto ==16
0x00, //inicio del dstring del descriptor del numero de serie ==17
USB_NUM_CONFIGURATIONS //numero de posibles configuraciones ==18
};

#if (sizeof(USB_DEVICE_DESC) != USB_DESC_DEVICE_LEN)
#error USB_DESC_DEVICE_LEN not defined correctly
#endif


////////////////////////////////////////////////////////////////
///
/// comienza el string de descriptores
/// String 0 es el lenguaje del teclado y debe ser definido. La gente de U.S.A. puede dejar esto
sin tocarlo.
///
////////////////////////////////////////////////////////////////
```

```
//el offset del comienzo de la localizacion del string. offset[0] es que comienza en 0, offset[1] es que comienza en 1, etc.
const char USB_STRING_DESC_OFFSET[]={0,4,12};

//numero de string que tienes, incluyendo string 0.
#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)

char const USB_STRING_DESC[]={
//string 0
4, //tamaño del string
USB_DESC_STRING_TYPE, //tipo de descriptor 0x03 (STRING)
0x09,0x04, //Definido para Microsoft US-English (usando este perdemos la ñ y cambian ciertos caracteres respecto al teclado español)
//string 1
8, //tamaño del string
USB_DESC_STRING_TYPE, //tipo de descriptor 0x03 (STRING)
'R',0,
'R',0,
'2',0,
//string 2
46, //tamaño del descriptor
USB_DESC_STRING_TYPE, //tipo de descriptor 0x03 (STRING)
'U',0,
'S',0,
'B',0,
' ',0,
'A',0,
'R',0,
'P',0,
'A',0,
' ',0,
'L',0,
'A',0,
'S',0,
'E',0,
'R',0,
' ',0,
' ',0,
'S',0,
'1',0,
'1',0,
'M',0,
'E',0,
'.',0
};
```

2.3-Código

```
////////////////////////////////////////////////////////////////////////
////
//// Arpa_Laser_S11ME.c                Version: Alpha V1
////
//// by scriptshow
////
//// 04/03/2015
////
////////////////////////////////////////////////////////////////////////

#include <18F2550.h>
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000)

#DEFINE USB_HID_DEVICE TRUE

#define USB_EP1_TX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE 8

#define USB_EP1_RX_ENABLE USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE 8

#include <pic18_usb.h>
#include <usb_kbd_HID.h> // Configuracion USB y descriptor del dispositivo
#include <usb.c> // Maneja los tokens del USB
#include <ctype.h>

////////////////////////////////////////////////////////////////////////////
//
// Defines
//
////////////////////////////////////////////////////////////////////////////

#define LED1 PIN_A5    //Multicolor 2
#define LED2 PIN_A1    //Verde
#define LED3 PIN_A3    //Multicolor 1
#define LED4 PIN_A2    //Rojo

#define NOTADO   PIN_B7
#define NOTADOb  PIN_B6
#define NOTARE   PIN_B5
#define NOTAREb  PIN_B4
#define NOTAMI   PIN_B3
#define NOTAFA   PIN_B2
#define NOTAFAb  PIN_B1
```

```
#define NOTASOL  PIN_B0
#define NOTASOLb PIN_C6
#define NOTALA   PIN_C0
#define NOTALAb  PIN_C1
#define NOTASI   PIN_C2

#define PEDAL    PIN_C7

#define LED_ON output_high
#define LED_OFF output_low

#define PIN_SOURCE PIN_A0     //Amarillo

////////////////////////////////////////////////////////////////////////
//
// RAM
//
////////////////////////////////////////////////////////////////////////

int8 enumerated;
int8 rx_msg[USB_EP1_RX_SIZE];
int8 tx_msg[8]={2,0,0,0,0,0,0,0};

char NextChar='0';
int1 hay_dato=0;
int8 ctrlpuls=0;



////////////////////////////////////////////////////////////////////////
//
// usb_debug_task()
//
// La llamada a esta funcion se realiza constantemente para comprobar el
// estado de la comunicacion Pic - Pc mediante USB. Los status se muestran mediante
// LED2 y LED3, que en nuestro caso, sera un unico LED bicolor.
//
////////////////////////////////////////////////////////////////////////

void usb_debug_task(void) {

  enumerated=usb_enumerated();//Devuelve 1 si el dispositivo fue enumerado correctamente

  if(enumerated){
    LED_ON(LED1);
    LED_OFF(LED3);
  }
  else{
    LED_ON(LED3);
```

```
   LED_OFF(LED1);
  }
}

////////////////////////////////////////////////////////////////////////////
//
// CHAR_2_USB_KBD_CODE
//
// Cambia el caracter introducido a un caracter con codigo de keyboard ASCII
//
////////////////////////////////////////////////////////////////////////////

int8 char_2_usb_kbd_code(char c){
  int8 ic;

  if(isAlpha(c)){
    ic=c-'a'+4;
  }
  else{
    if(c=='0'){
      ic=39;
    }
    else{
      ic=c-'1'+30;
    }
  }
  return(ic);
}

////////////////////////////////////////////////////////////////////////////
//
// explore_pins
//
// Comprueba si hay alguna nota musical activa.
//
////////////////////////////////////////////////////////////////////////////

void explore_pins(void){

        if((input(NOTADO) * input(NOTADOb) * input(NOTARE) * input(NOTAREb) *
input(NOTAMI) * input(NOTAFA) * input(NOTAFAb) * input(NOTASOL) * input(NOTASOLb) *
input(NOTALA) * input(NOTALAb) * input(NOTASI))==0){hay_dato=1; LED_ON(LED2);}
}

////////////////////////////////////////////////////////////////////////////
//
// pulsacion_simultanea
```

```
//
// Es la funcion encargada de que la multipulsacion de notas funcione correctamente
// alertando con un LED rojo cuando se estan pulsando mas notas de las que se pueden
// enviar simultaneamente.
//
////////////////////////////////////////////////////////////////////////////////

int pulsacion_simultanea(void){


        if(ctrlpuls>4){ctrlpuls=4;LED_ON(LED4);}
        else{LED_OFF(LED4);}

return (ctrlpuls+3);
}

////////////////////////////////////////////////////////////////////////////
//
// Nota Musical
//
// Realiza la conversion nota musical hacia tecla keyboard, enviando asi notas
// musicales o efectos de sonido.
//
////////////////////////////////////////////////////////////////////////////


void NotaMusical(void){

        ctrlpuls=0;

        if(input(PEDAL)==0){       // Notas musicales
                if(input(NOTADO)==0){
                        NextChar='a';
                        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                        ctrlpuls++;
                }
                if(input(NOTADOb)==0){
                        NextChar='w';
                        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                        ctrlpuls++;
                }
                if(input(NOTARE)==0){
                        NextChar='s';
                        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                        ctrlpuls++;
                }
                if(input(NOTAREb)==0){
```

```
                NextChar='e';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTAMI)==0){
                NextChar='d';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTAFA)==0){
                NextChar='f';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTAFAb)==0){
                NextChar='t';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTASOL)==0){
                NextChar='g';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTASOLb)==0){
                NextChar='y';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTALA)==0){
                NextChar='h';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTALAb)==0){
                NextChar='u';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
        if(input(NOTASI)==0){
                NextChar='j';
                tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                ctrlpuls++;
        }
}else{                                  // Efectos de sonido
```

```
if(input(NOTADO)==0){
        NextChar='z';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTADOb)==0){
        NextChar='x';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTARE)==0){
        NextChar='1';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTAREb)==0){
        NextChar='2';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTAMI)==0){
        NextChar='3';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTAFA)==0){
        NextChar='4';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTAFAb)==0){
        NextChar='5';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTASOL)==0){
        NextChar='6';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
if(input(NOTASOLb)==0){
        NextChar='7';
        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
        ctrlpuls++;
}
```

```
                if(input(NOTALA)==0){
                        NextChar='8';
                        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                        ctrlpuls++;
                }
                if(input(NOTALAb)==0){
                        NextChar='9';
                        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                        ctrlpuls++;
                }
                if(input(NOTASI)==0){
                        NextChar='0';
                        tx_msg[pulsacion_simultanea()]=char_2_usb_kbd_code(NextChar);
                        ctrlpuls++;
                }
        }
}

////////////////////////////////////////////////////////////////////////////
//
// usb_keyboard_task()
//
// Envia paquetes de datos mediante el USB con protocolo HID.
//
// tx_msg[0] = HID report id
// tx_msg[1] = Modificador (funciona para simular teclas especiales como la pulsacion de shift)
// tx_msg[2] = Constante 0
// tx_msg[3:7] = Array de las teclas que estan siendo presionadas. (Máximo 5 simultaneas)
// Si msg[2:7]={0} Cuando el valor es 0, es que no hay ninguna nota presionada.
//
// rx_msg[1] = HID report id
// rx_msg[0] = led status
//
////////////////////////////////////////////////////////////////////////////

void usb_keyboard_task(void) {

 if(hay_dato==1){

  hay_dato=0;

        do{
                NotaMusical();
                usb_put_packet(1,tx_msg,sizeof(tx_msg),USB_DTS_TOGGLE);
                delay_ms(5);
                tx_msg[3]=0;tx_msg[4]=0;tx_msg[5]=0;tx_msg[6]=0;tx_msg[7]=0;
        }while(input(NOTADO)*input(NOTADOb)*input(NOTARE)*input(NOTAREb)*input(NOA
```

```
MI)*input(NOTAFA)*input(NOTAFAb)*input(NOTASOL)*input(NOTASOLb)*input(NOTALA)*inp
ut(NOTALAb)*input(NOTASI) == 0);

   LED_OFF(LED2);

  }else{

   tx_msg[3]=0;tx_msg[4]=0;tx_msg[5]=0;tx_msg[6]=0;tx_msg[7]=0;
   usb_put_packet(1,tx_msg,sizeof(tx_msg),USB_DTS_TOGGLE);
   delay_ms(5);

  }
}

////////////////////////////////////////////////////////////////////////////
//
// usb_rx_task()
//
// Recibe paquetes de datos mediante la comunicacion USB
//
////////////////////////////////////////////////////////////////////////////

void usb_rx_task(void){


  if (usb_kbhit(1)){
   usb_get_packet(1, rx_msg, sizeof(rx_msg));
  }
}

////////////////////////////////////////////////////////////////////////////
//
//  FUNCION PRINCIPAL
//
////////////////////////////////////////////////////////////////////////////


void main() {

  hay_dato=0;

  delay_ms(500);

  LED_OFF(LED1);
  LED_OFF(LED2);
  LED_OFF(LED3);

  usb_init_cs();
```

```
  set_tris_a(0b00100000);
  set_tris_b(0b11111111);
  set_tris_c(0b11000111);
  output_high(PIN_SOURCE);

  while (TRUE) {

    usb_task();
    usb_debug_task();

    if (usb_enumerated()) {
            explore_pins();
      usb_keyboard_task();
      usb_rx_task();
      delay_ms(5);
    }
  }
}
```

4-Planos y esquemas