uc3m | Universidad Carlos III de Madrid

Máster de Ingeniería en Sistemas Electrónicos y Aplicaciones
2018 – 2019

Proyectos experimentales II

Sintetizador MIDI

Manuel Carrasco Sánchez

Jorge Bustos Sánchez

3 de junio de 2019, Madrid

Índice

1 Introducción

En este documento se exponen el sistema mínimo y las extensiones del proyecto "Sintetizador MIDI de música":

- Descripción del sistema mínimo y extensiones.
- Los diagramas de bloques.
- Los esquemáticos.
- Los diagramas de flujo del software.
- Descripción de algoritmos y soluciones adoptadas.
- Código en C comentado.

2 Descripción del sistema mínimo y extensiones

El sistema mínimo consta de los siguientes elementos:

- Recepción e interpretación de datos MIDI.
- Generación de notas musicales en base a los códigos MIDI recibidos.
- Envío de la información MIDI recibida a un interfaz web.

Las extensiones que se han implementado han sido las siguientes:

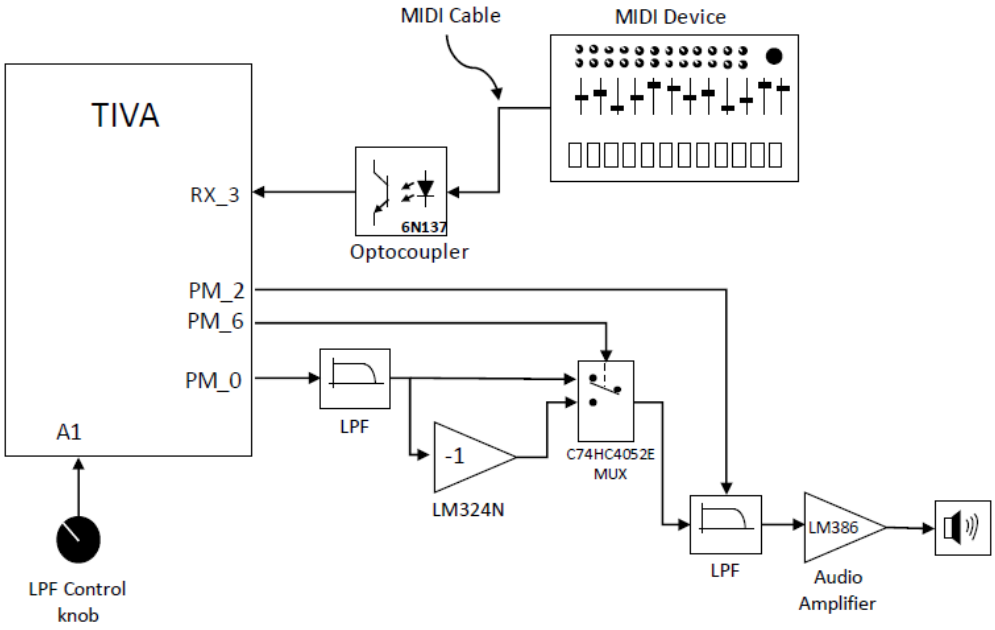
- Adición de un sistema de generación de envolvente basado en una señal PWM y el integrado CD74HC4052.
- Inclusión de un filtro con frecuencia basado en el integrado MF10 con control de la frecuencia de corte regulado por PWM, basada en la lectura de un potenciómetro.
- Implementación de diferentes escalas y funciones de cambio de octava y trasposición en el interfaz web.

3 Diagramas de bloques

El diagrama de bloques del sistema consta de dos subsistemas:

1. Comunicación y recepción MIDI: formado por el controlador MIDI que envía datos MIDI a través del cable MIDI al optoacoplador. Este se encarga de realizar la conversión MIDI de corriente a voltaje para poder realizar la lectura por puerto serie en la tiva a través del PIN RX_3.
2. Sintetizador de música: una vez recibidos los datos MIDI con la información de la nota, su comienzo y duración en la tiva se generan dos señales PWM que compondrán el sonido final. El PIN PM_6 que general señal de la nota y el PIN PM_ que genera la envolvente. La señal de la envolvente pasa por un filtro paso-bajo para eliminar los armónicos de alta frecuencia (debidos a la modulación PWM). Para generar una envolvente tal y como la conocemos, se introduce el PWM de la nota para controlar un switch que va conmutando y sacando a la. salida la envolvente positiva y la envolvente invertida, en un ciclo de conmutación. A parte se saca por el PIN PM_2 una señal PWM que permite controlar la

frecuencia de corte del integrado MF10. Para controlar esta frecuencia de corte se lee la tensión de potenciómetro por la entrada analógica A1.

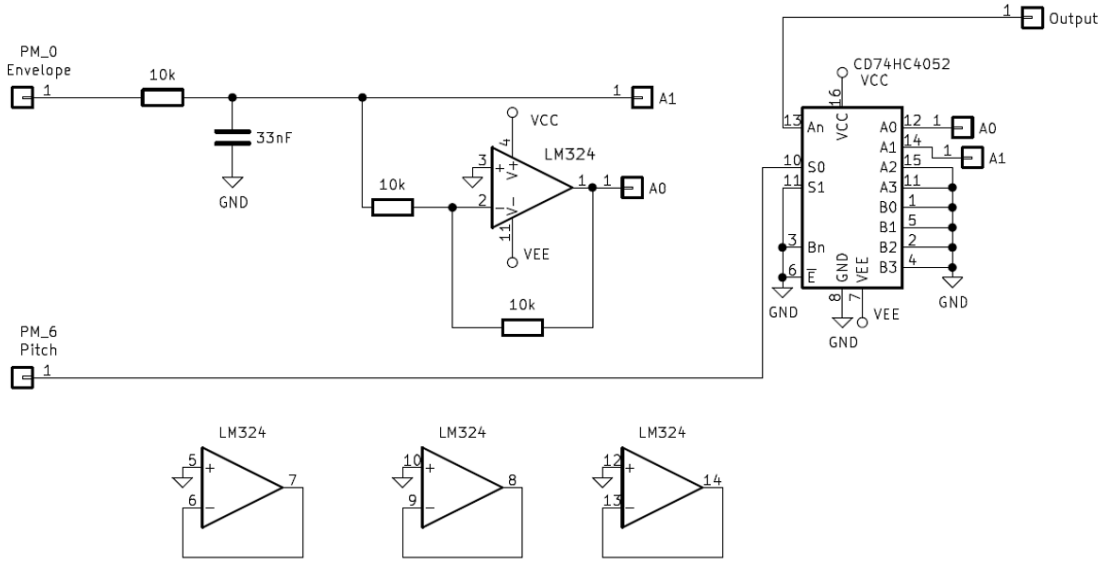


4 Esquemáticos

En el siguiente esquemático se muestra la circuitería encargada de invertir la señal de la envolvente y mezclarla con la señal de la nota.

Para invertirla se utiliza un amplificador operacional en configuración inversora (integrado LM324) y se opta por utilizar dos resistencias de 10kΩ para obtener una ganancia de 1 y que la envolvente negativa sea igual que la positiva. El resto de pines del IC que no se van a utilizar se conectan tal y como se muestra en la parte inferior de esquemático.

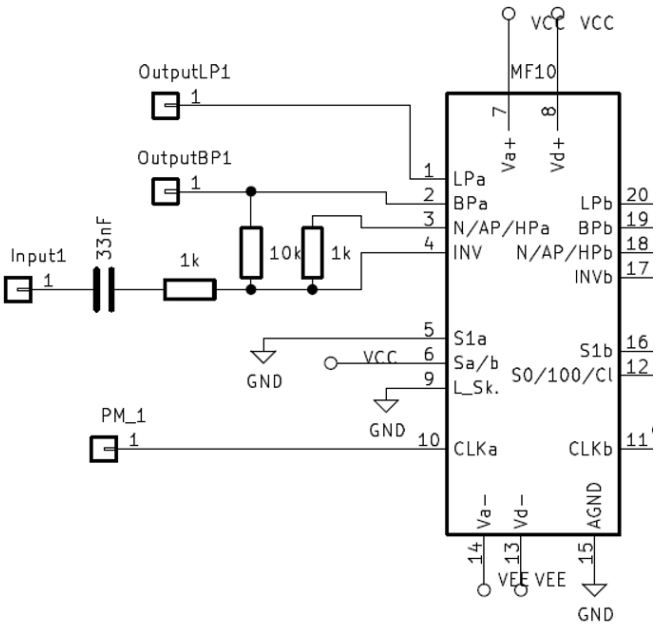
Para obtener una señal conjunta de envolvente y nota se utiliza el IC CD74HC4052. Este IC se trata de un multiplexor donde se empleará la señal de la nota para conmutar entre envolvente positiva y negativa, obteniendo en la salida la señal deseada.



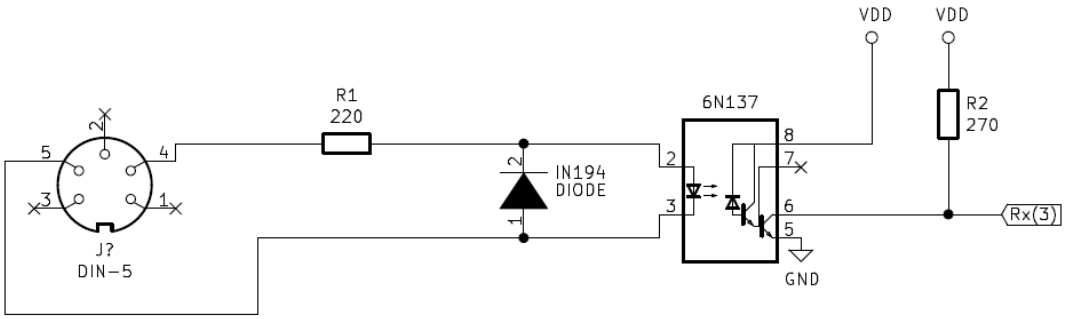
Este esquemático muestra el conexionado para la implementación del filtro paso bajo. Para ello se utiliza el integrado MF10.

Se conecta a la entrada un condensador de acoplo, para eliminar la componente continua y una resistencia de entrada. Con las resistencias de 1kΩ y 10 kΩ en paralelo se controla el factor Q o resonancia de los filtros, donde $Q = \frac{10\ k\Omega}{1\ k\Omega} = 10$. La señale PM_1 (señal PWM con DC del 50%) entra a la entrada CLKa, que controla la frecuencia de corte del filtro.

La parte derecha del integrado se deja sin conectar ya que no se requiere de su uso.



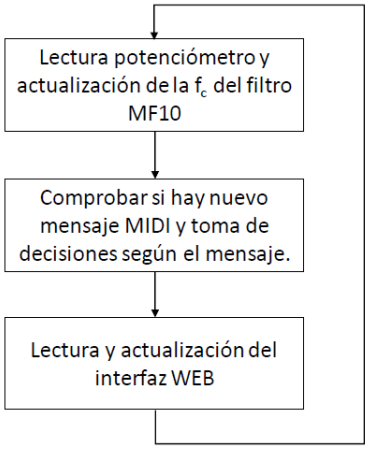
Y, por último, el esquemático de la comunicación MIDI. El circuito MIDI se trata de un lazo de corriente de 5mA con lógica invertida (ON para 0mA) Este consta del conector DIN-5 conectado al controlador MIDI. La señal MIDI sale a través del DIN-5 que circuila a través del optoacoplador que además de aislar la salida de la entrada realiza la conversión de corriente a tensión. Del optoacoplador se sacan los datos en voltaje por el PIN 6 que se llevan a la UART de la tiva al PIN RX3.



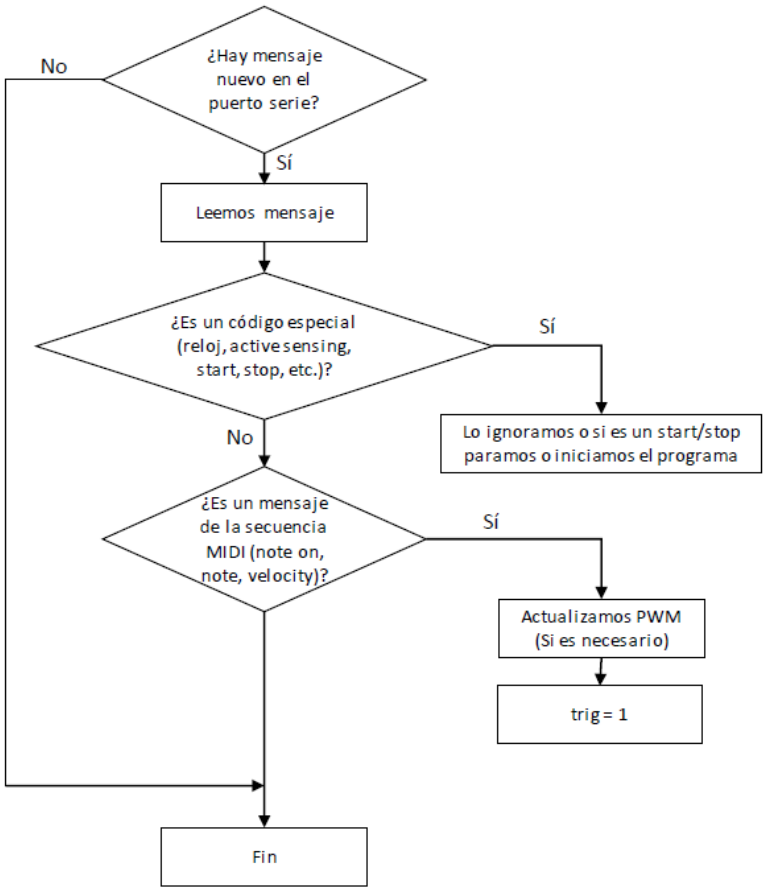
5 Diagramas de flujo

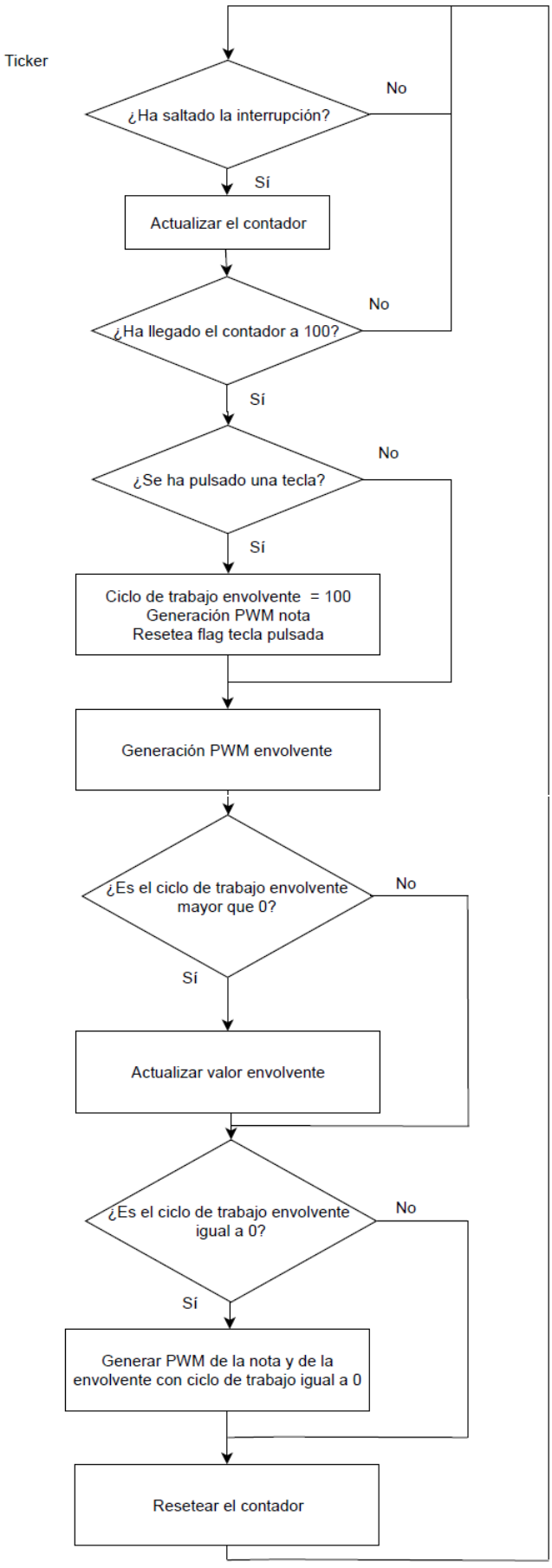
A continuación, se muestran los diagramas de flujo del bucle principal del programa, así como de las distintas funciones.

Loop principal



Comprobación de mensajes MIDI




Ticker
¿Ha saltado la interrupción?
No
Sí
Actualizar el contador
¿Ha llegado el contador a 100?
No
Sí
¿Se ha pulsado una tecla?
No
Sí
Ciclo de trabajo envolvente = 100
Generación PWM nota
Resetea flag tecla pulsada
Generación PWM envolvente
¿Es el ciclo de trabajo envolvente mayor que 0?
No
Sí
Actualizar valor envolvente
¿Es el ciclo de trabajo envolvente igual a 0?
No
Sí
Generar PWM de la nota y de la envolvente con ciclo de trabajo igual a 0
Resetear el contador

6 Descripción de algoritmos y soluciones adoptadas

Algoritmo de implementación de escalas y funciones de cambio de octava y trasposición.

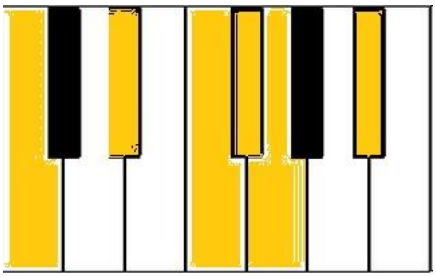
El conjunto de notas musicales se agrupa en octavas (con una relación en frecuencia de 2:1) y cada octava se subdivide a su vez en 12 notas geométricamente espaciadas (escala cromática).

De esta definición podemos obtener un coeficiente tal que, si multiplicamos cualquier nota por el mismo, obtenemos la siguiente nota de la escala cromática.

$$f \cdot \alpha^{12} = 2f$$

$$\alpha = \sqrt[12]{2} \approx 1{,}0594$$

Para implementar cada una de las escalas, bastaría con rellenar un vector con las relaciones entre cada nota. Un ejemplo sería la escala de blues, que se muestra en la siguiente imagen.



La relación de tonos y semitonos de la escala sería: $1\frac{1}{2}$ tonos, $1\frac{1}{2}$ tonos, 1 semitono, 1 semitono, 1 semitono y $1\frac{1}{2}$ tonos. Y su implementación en el vector sería: {0, 3, 5, 6, 7, 10} Agrupando cada uno de los vectores para cada una de las escalas más significativas que se han implementado formamos una matriz.

Basándonos en los fundamentos mencionados, se ha implementado la siguiente fórmula:

$$pitch = 32Hz \cdot (\sqrt[12]{2})^{traspose+escala} \cdot 2^{octava}$$

La variable *pitch* que hace referencia a la frecuencia de la nota que se reproduce cada vez que se pulsa una nota se obtiene a partir de 3 variables:

- Octava: esta variable se incrementa o decrementa en una unidad (en función del botón que se pulse en el interfaz).
- Transpose: Idem.
- Escala: toma el valor de una matriz cuyos coeficientes dependen de la escala que se ha seleccionado en la interfaz y de la tecla que se ha pulsado en la controladora MIDI.

7 Código

```
1. // MISEA-UC3M 2014-15
2. // Base program
3. // Proyectos Experimentales II
4. // Copy Right Universidad Carlos III de Madrid
5. // ------------------   Librerías  --------------------------//
6. #include <stdint.h>              // standard library for
   integers (used in the next libraries)
7. #include "driverlib\systick.h"   // standard library for the
   SysTick (header)
8. #include "driverlib\systick.c"   // standard library for the
   SysTick (functions)
9. #include "wiring_private.h"      // library to access the PWM
   with configurable frequency
10.      #include "Ethernet.h"
11.
12.      #define TickerPeriod 2721       // f = 44100Hz (T = 22,67
   useg)
13.
14.      // ---------------   Declaración pines  -----------------//
15.      #define PWMnote PM_6            //señal pwm cuadrada nota
   musical
16.      #define PWMenv  PM_0            //señal pwm envolvente
17.      #define PWMfc   PM_2            //señal pwm control
   frecuencia LP
18.      #define ReadV A1                //señal analogica lectura
   voltaje
19.                                      //potenciometro para
   control frecuencia
20.
21.      // -------------   Declaración constantes  ------------//
22.      #define FENV 10000              //Frecuencia PWM envolvente
23.
24.      // --------------   Declaración variables  --------------//
25.      char key;                       //caracter leido terminal
   serie
26.      int pitch = 440;                //frecuencia nota Hz
27.      int tickercount = 0;            //contador
28.      int trig = 0;                   //indica pulsación nueva
   nota
29.      int DC_env;                     //ciclo de trabajo PWM
   envolvente
30.
31.      byte MIDIcommand;               //lectura puerto MIDI
32.      byte MIDInote;                  //byte MIDI correspondiente
   a la nota
33.      byte MIDIvelocity;              //byte MIDI correspondiente
   a la velocity
34.
35.      double fc;                      //frecuencia de corte del
   LP
36.      int sensorValue = 0;            //lectura potenciómetro
   para LP
37.
38.      //Matriz de escalas
39.      int matrix[9][11]={{0, 2, 4, 5, 7, 9, 11, 12, 14, 16, 17},
             //Mayor
40.                         {0, 2, 3, 5, 7, 9, 10, 12, 14 ,15, 17},
             //Dorico
```

```
41.                         {0, 1, 3, 5, 7, 8, 10, 12, 13, 15, 17},
            //Frigio
42.                         {0, 2, 4, 6, 7, 9, 11, 12, 14, 16, 17},
            //Lidio
43.                         {0, 2, 4, 5, 7, 9, 10, 12, 14, 16, 17},
            //Mixolidio
44.                         {0, 2, 3, 5, 7, 8, 10, 12, 14, 15, 17},
            //Menor
45.                         {0, 1, 3, 5, 6, 8, 10, 12, 13, 15, 17},
            //Locrio
46.                         {0, 3, 5, 6, 7, 10, 12, 15, 17, 18, 19},
            //Blues
47.                         {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}};
          //Cromática
48.
49.      int scale = 0;                 //variable de asignación de
   la escala
50.      int traspose = 0;              //variable de trasposición
51.      int octave = 2;                //variable de asignación de
   osctava
52.
53.      int state = 0;                 //variable maquina de estados
54.
55.      bool flag_on = 0;              //flag de comienzo del loop
   secuenciador
56.
57.      EthernetServer server(80);     //crea un servidor que
   escucha conexiones entrantes en un puerto específico
58.      EthernetClient client;         //crea un cliente que se
   pueda conectar a un IP específico
59.      String currentLine;
60.
61.      //--------------   Declaración funciones  --------------//
62.      void printConfig();            //función de configuración de
   la interfaz web
63.      void printEthernetData();      //función que imprime datos
   IP
64.      void cliente();
65.      void checkMIDI();              //función de lectura de
   información MIDI
66.      void mf10filter();             //función de control de
   frecuencia de corte del MF10
67.      int pitch_calc (int MIDInote, int transpose, int scale, int
    octave);    //función de calculo de la nota a través de varias
   variables
68.      int note2number (byte note);       //convierte la nota a
   número correspondiente a una nota de una escala
69.
70.
71.
72.      ////////////////////////////////////////////////////////////
73.      //---------------------- SET UP ----------------------------//
74.      ////////////////////////////////////////////////////////////
75.
76.      void setup()
77.      {
78.
79.        Serial.begin(115200);                    // Inicializamos
   puerto serie
80.        Serial3.begin(31250);                    // Inicializamos
   puerto MIDI
```

```
81.
82.         //Inicialización ethernet
83.         Serial.println("Connecting to Ethernet....");
84.         Ethernet.begin(0);
85.         server.begin();
86.         printEthernetData();
87.
88.         //Inicialización Ticker y PWM
89.         SysTickDisable();                       // Disables
    SysTick during the configuration
90.         SysTickPeriodSet(TickerPeriod);         // Define the
    period of the counter. When it reaches 0, it activates the
    interrupt
91.         SysTickIntRegister(&Ticker);            // The interrupt
    is associated to the SysTick ISR
92.         SysTickIntEnable();                     // The SysTick
    interrupt is enabled
93.         SysTickEnable();                        // The SysTick is
    enabled, after having been configured
94.         IntMasterEnable();                      // All interrupts
    are enabled
95.
96.         //Inicialización entradas y salidas
97.         pinMode(PWMnote, OUTPUT);               // Pin PWM sonido
    como salida
98.         pinMode(PWMenv, OUTPUT);                // Pin PWM
    envolvente como salida
99.         pinMode(PWMfc, OUTPUT);                 // Pin PWM
    frecuencia control LP como salida
100.        pinMode(ReadV, INPUT);                  // Pin analógico
    lectura voltaje potenciometro
101.                                                // como entrada
102.        PWMWrite(PWMnote, 100, 0, 0);           // Incialización
    PWM nota a cero
103.        PWMWrite(PWMenv, 100, 0, 0);            // Incialización
    PWM envolvente a cero
104.
105.
106.
107.    }
108.
109.    ////////////////////////////////////////////////////////////
110.    //---------------------  LOOP  -------------------------//
111.    ////////////////////////////////////////////////////////////
112.
113.    void loop()
114.    {
115.      mf10filter();
116.
117.      checkMIDI();
118.
119.      cliente();
120.    }
121.
122.    ////////////////////////////////////////////////////////////
123.    //---------------------  TICKER -------------------------//
124.    ////////////////////////////////////////////////////////////
125.
```

```
126.     void Ticker()
127.     {
128.       //Actualiza valor contador
129.       tickercount++;
130.
131.       //Verifica contador llega a 100
132.       if (tickercount == 100) {
133.         //Verifica que se ha pulsado tecla
134.         if (trig == 1 ) {
135.           DC_env = 100;                                  //Ciclo
   trabajo envolvente a 100
136.           PWMWrite(PWMnote, 100, 50, pitch);         //Generació
   n nota pulsada
137.           PWMWrite(PWMfc,100,50,pitch*100);          //Genera
   señal control frecuencia filtro LP
138.           trig = 0;                                  //Resetea
   trig
139.         }
140.
141.         //Generación PWM envolvente
142.         PWMWrite(PWMenv, 100, DC_env, FENV);
143.
144.         //Actualiza ciclo de trabajo envolvente disminuyendolo
   en una unidad
145.         if (DC_env > 0) DC_env--;
146.
147.         //Pone a cero las señales PWM de la señal y la
   envolvente
148.         if (DC_env == 0) {
149.           PWMWrite(PWMnote, 100, 0, 0);
150.           PWMWrite(PWMenv, 100, 0, 0);
151.         }
152.
153.         //Resetea el contador
154.         tickercount = 0;
155.       }
156.     }
157.
158.
159.     ////////////////////////////////////////////////////////
160.     //----------------- DECLARACIÓN FUNCIONES  ---------------//
161.     ////////////////////////////////////////////////////////
162.
163.     void cliente(){
164.
165.     client = server.available();
166.
167.       if (client) {                             // if you get a
   client,
168.         Serial.print("new client on port ");           // print
   a message out the serial port
169.         Serial.println(client.port());
170.         currentLine = "";
171.         boolean newConnection = true;     // flag for new
   connections
172.         unsigned long connectionActiveTimer;  // will hold the
   connection start time
173.
174.         while (client.connected()) {       // loop while the
   client's connected
```

```
175.           if (newConnection){                  // it's a new
   connection, so
176.             connectionActiveTimer = millis(); // log when the
   connection started
177.             newConnection = false;           // not a new
   connection anymore
178.           }
179.           if (!newConnection && connectionActiveTimer + 1000 <
   millis()){
180.             // if this while loop is still active 1000ms after
   a web client connected, something is wrong
181.             break;  // leave the while loop, something bad
   happened
182.           }
183.
184.
185.           if (client.available()) {             // if there's
   bytes to read from the client,
186.             char c = client.read();             // read a byte,
   then
187.             // This lockup is because the recv function is
   blocking.
188.             Serial.print(c);
189.             if (c == '\n') {                    // if the byte
   is a newline character
190.               // if the current line is blank, you got two
   newline characters in a row.
191.               // that's the end of the client HTTP request, so
   send a response:
192.               if (currentLine.length() == 0) {
193.                 break;
194.               }
195.               else {      // if you got a newline, then clear
   currentLine:
196.                 currentLine = "";
197.               }
198.             }
199.             else if (c != '\r') {    // if you got anything
   else but a carriage return character,
200.               currentLine += c;      // add it to the end of
   the currentLine
201.             }
202.
203.             if (currentLine.endsWith("GET / ")) {
204.               //statusConfig = 0;
205.               printConfig();
206.              }
207.             if (currentLine.endsWith("GET /config.html ")) {
208.               printConfig();
209.             }
210.             // Check to see if the client request scale
211.             if (currentLine.endsWith("GET /MAYOR")) {
212.               scale = 0;
213.               printConfig();
214.             }
215.             if (currentLine.endsWith("GET /JONICO")) {
216.               scale = 1;
217.               printConfig();
218.             }
219.             if (currentLine.endsWith("GET /FRIGIO")) {
220.               scale = 2;
```

```
221.               printConfig();
222.             }
223.             if (currentLine.endsWith("GET /LIDIO")) {
224.               scale = 3;
225.               printConfig();
226.             }
227.             if (currentLine.endsWith("GET /MIXOLIDIO")) {
228.               scale = 4;
229.               printConfig();
230.             }
231.             if (currentLine.endsWith("GET /MENOR")) {
232.               scale = 5;
233.               printConfig();
234.             }
235.             if (currentLine.endsWith("GET /LOCRIO")) {
236.               scale = 6;
237.               printConfig();
238.             }
239.             if (currentLine.endsWith("GET /BLUES")) {
240.               scale = 7;
241.               printConfig();
242.             }
243.             if (currentLine.endsWith("GET /CROMATICA")) {
244.               scale = 8;
245.               printConfig();
246.             }
247.
248.             //Traspose upgrade
249.             if (currentLine.endsWith("GET /TRASPOSE_UP")) {
250.               traspose = traspose + 1;
251.               printConfig();
252.             }
253.             if (currentLine.endsWith("GET /TRASPOSE_DOWN")) {
254.               traspose = traspose - 1;
255.               printConfig();
256.             }
257.
258.             //Octave upgrade
259.             if (currentLine.endsWith("GET /OCTAVE_UP")) {
260.               octave = octave + 1;
261.               printConfig();
262.             }
263.             if (currentLine.endsWith("GET /OCTAVE_DOWN")) {
264.               octave = octave - 1;
265.               printConfig();
266.             }
267.
268.           }
269.         }
270.         // close the connection:
271.         client.stop();
272.         //Serial.println("client disonnected");
273.       }
274.
275.     }
276.
277.     //////////////////////////////////////////////////////
278.
```

```
279.     void printConfig()
280.     {
281.       // HTTP headers always start with a response code (e.g.
   HTTP/1.1 200 OK)
282.       // and a content-type so the client knows what's coming,
   then a blank line:
283.
284.       client.println("HTTP/1.1 200 OK");
285.       client.println("Content-type:text/html");
286.       client.println();
287.       client.print("\n");
288.       client.println("<html><head><title>Roland
   Server</title></head><body align=center>");
289.       client.println("<h1 align=center><font
   color=\"red\">Roland Interface</font></h1>");
290.       client.println("</body></html>");
291.
292.       client.println("Mayor    <button
   onclick=\"location.href='/MAYOR'\">ON</button><br>");
293.       client.println("Jonico    <button
   onclick=\"location.href='/JONICO'\">ON</button><br>");
294.       client.println("Frigio    <button
   onclick=\"location.href='/FRIGIO'\">ON</button><br>");
295.       client.println("Lidio    <button
   onclick=\"location.href='/LIDIO'\">ON</button><br>");
296.       client.println("Mixolidio    <button
   onclick=\"location.href='/MIXOLIDIO'\">ON</button><br>");
297.       client.println("Menor    <button
   onclick=\"location.href='/MENOR'\">ON</button><br>");
298.       client.println("Locrio    <button
   onclick=\"location.href='/LOCRIO'\">ON</button><br>");
299.       client.println("Blues    <button
   onclick=\"location.href='/BLUES'\">ON</button><br>");
300.       client.println("Cromatica    <button
   onclick=\"location.href='/CROMATICA'\">ON</button><br>");
301.
302.       client.print("<br><br>");
303.
304.       client.print("TRASPOSE    <button
   onclick=\"location.href='/TRASPOSE_UP'\">UP</button>");
305.       client.println(" <button
   onclick=\"location.href='/TRASPOSE_DOWN'\">DOWN</button><br>");
306.
307.       client.print("OCTAVE    <button
   onclick=\"location.href='/OCTAVE_UP'\">UP</button>");
308.       client.println(" <button
   onclick=\"location.href='/OCTAVE_DOWN'\">DOWN</button><br>");
309.
310.       // optional: client.println(" <input type=\"submit\"
   value=\"Update\"> ");
311.       client.println(" </form> ");
312.       // The HTTP response ends with another blank line:
313.       client.println();
314.       // break out of the while loop:
315.     }
316.




317.     ////////////////////////////////////////////////////////
318.
```

```
319.     void printEthernetData() {
320.       // print your IP address:
321.       Serial.println();
322.       Serial.println("IP Address Information:");
323.       IPAddress ip = Ethernet.localIP();
324.       Serial.print("IP Address:\t");
325.       Serial.println(ip);
326.
327.       // print your MAC address:
328.
329.       IPAddress subnet = Ethernet.subnetMask();
330.       Serial.print("NetMask:\t");
331.       Serial.println(subnet);
332.
333.       // print your gateway address:
334.       IPAddress gateway = Ethernet.gatewayIP();
335.       Serial.print("Gateway:\t");
336.       Serial.println(gateway);
337.
338.       // print your gateway address:
339.       IPAddress dns = Ethernet.dnsServerIP();
340.       Serial.print("DNS:\t\t");
341.       Serial.println(dns);
342.
343.     }
344.
345.     ////////////////////////////////////////////////////////////
346.
347.     void checkMIDI(){
348.
349.         if(Serial3.available()){                                //
    se chequea si hay un dato disponible por puerto serie
350.         MIDIcommand = Serial3.read();                          //
    lectura del dato entrante por el puerto serie
    3
351.
352.        //Verificamos si se ha pulsado el boton de PLAY 0XFA
353.         if(MIDIcommand == 0xFA){
354.
355.           flag_on = 1;
356.
357.         //Verificamos si se ha pulsado el boton de STOP 0XFA
358.         }else if(MIDIcommand == 0xFC){
359.           state = 0;
360.           flag_on = 0;
361.
362.         }else if(flag_on){
363.
364.           if(MIDIcommand != 0xF8){
365.             if(state == 0 && MIDIcommand == 0x99){             //
    Si hacemos 0x90 <= MIDIcommand <= 0x9F leemos todos los canales
    MIDI
366.              state = 1;
367.             }
368.             else if(state == 1){
369.               if(MIDIcommand == 0xFE){
370.                 state = 0;
371.               }
372.               else{
373.                  MIDInote = MIDIcommand;
374.                  state = 2;
```

```
375.               }
376.             }else if(state == 2){
377.                MIDIvelocity = MIDIcommand;
378.                state = 1;
379.                if(MIDIvelocity > 0x00){
380.                  pitch = pitch_calc(MIDInote,traspose,scale,oct
    ave);
381.                  trig = 1;
382.
383.
384.                }
385.              }
386.           }
387.         }
388.
389.       }
390.     }
391.
392.     ////////////////////////////////////////////////////////////
393.
394.     void mf10filter(){
395.
396.       int sensorValue = analogRead(ReadV);
397.       fc = 500*(sensorValue/1023.0)+250;
398.       PWMWrite(PWMfc,100,50,fc*100);
399.
400.     }
401.
402.     ////////////////////////////////////////////////////////////
403.
404.
405.     int pitch_calc (int MIDInote, int transpose, int scale, int
     octave){
406.       int pitch;
407.       int key;
408.
409.       key = note2number(MIDInote);
410.
411.       pitch = pow(1.0594, matrix[scale][key-
    1])*pow(1.0594, transpose)*pow(2, octave)*32.7;
412.
413.       return(pitch);
414.     }
415.
416.
417.     ////////////////////////////////////////////////////////////
418.
419.     int note2number (byte note) {
420.
421.       int number;
422.
423.       switch (note) {
424.
425.         case 35:
426.           number = 1;          //BD
427.
428.           break;
429.         case 36:
430.           number = 1;
431.
432.           break;
```

```
433.
434.          case 38:
435.            number = 2;          //SD
436.
437.            break;
438.          case 40:
439.            number = 2;
440.
441.            break;
442.
443.          case 41:
444.            number = 3;          //LT
445.
446.            break;
447.          case 43:
448.            number = 3;
449.
450.            break;
451.
452.          case 45:
453.            number = 4;          //MT
454.
455.            break;
456.          case 47:
457.            number = 4;
458.
459.            break;
460.
461.          case 48:
462.            number = 5;          //HT
463.
464.            break;
465.          case 50:
466.            number = 5;
467.
468.            break;
469.
470.          case 37:
471.            number = 6;          //RS
472.
473.            break;
474.
475.          case 39:
476.            number = 7;          //HC
477.
478.            break;
479.
480.          case 42:
481.            number = 8;          //CH
482.
483.            break;
484.          case 44:
485.            number = 8;
486.
487.            break;
488.
489.          case 46:
490.            number = 9;          //OH
491.
492.            break;
493.
```

```
494.          case 49:
495.            number = 10;          //CC
496.
497.            break;
498.
499.          case 51:
500.            number = 11;          //RC
501.
502.            break;
503.
504.          default:
505.            number = 11;
506.            break;
507.        }
508.
509.        return(number);
510.      }
```

8 Bibliografía

[1] «6n137 component datasheet». Vishay Semiconductors. Accedido 3 de junio de 2019. https://www.vishay.com/docs/84732/6n137.pdf.

[2] «cd74hc4052 component datasheet». Texas Intruments . Accedido 3 de junio de 2019. http://www.ti.com/lit/ds/symlink/cd74hc4052.pdf.

[3] «lm324 component datasheet». Texas Instruments. Accedido 3 de junio de 2019. http://www.ti.com/lit/ds/symlink/lm324-n.pdf.

[4] «mf10 component datasheet». Texas Instruments. Accedido 3 de junio de 2019. https://www.ti.com/lit/ds/symlink/mf10-n.pdf.

[5] «The Complete MIDI 1.0 Detailed Specification». Accedido 3 de junio de 2019. https://www.midi.org/specifications-old/item/the-midi-1-0-specification

[6] «Roland TR-8_MIDI specifications». Accedido 3 de junio de 2019. http://cdn.roland.com/assets/media/pdf/TR-8_MIDI_Imple_Chart_e02_W.pdf.