TRABAJO PRÁCTICO PII

Materia: Programación II

Profesores:Osimani, Cesar.
Salamero, Martin.

Integrantes: Alvarez Figueroa, Agustina.
Bobadilla Barcelo, Carlos Ignacio.

Fecha de entrega: 10/10/18

INTRODUCCION

El presente trabajo tiene como finalidad, muestrear e identificar una señal DTMF, , (Dual-Tone Multi-Frequency.). Este es un tipo de señalización de línea utilizado en sistemas de comunicaciones que consiste en pares de tonos audibles en la gama de frecuencias de la voz humana y se generan cuando se acciona el teclado alfanumérico de aparatos telefónicos.

Lo que se desea a lo largo de este trabajo, es poder ingresar una señal DTMF, identificar la frecuencia, y decodificar qué tecla se accionó. Para ello, aplicaremos el teorema del muestreo y analizaremos la señal y así poder identificar cuales son los tonos que componen la señal y asi asociarla al boton que se pulso.

A continuación se detalla, la base teórica y el hardware necesario para el trabajo práctico propuesto.

En principio se propone un sistema que permita la simulación de una transmisión de señalización DTMF , y partir de ello, ingresar dicha señal a un microprocesador dsPic para realizarle el tratamiento necesario. El lenguaje de programación utilizado en el dsPic será C, y su compilador será el software MikroC, simulado antes de pasar a la instancia de la placa por Proteus, el programa desarrollado se montara sobre el pic 30f4013.
Y luego se implementaran las interfaces de hardware, por medio de la placa de desarrollo.

DESARROLLO

DMTF

Una señal DTMF, es la suma de dos tonos, uno de un grupo bajo y el otro de un grupo alto de frecuencias, con cada grupo conteniendo cuatro tonos individuales.
Las frecuencias de los tonos fueron seleccionadas de tal forma que sus armónicos no se encuentran relacionados y que los productos de su intermodulación produzcan un deterioro mínimo en la señalización.
Este esquema permite dieciséis combinaciones únicas, diez de estos códigos representan los números del cero al nueve, los seis restantes (*, #, A, B, C, D) son reservados para señalización especial.
La mayoría de los teclados en los teléfonos contienen diez interruptores de presión numéricos más el asterisco (*) y el símbolo de numeral (#). Cuando hablamos de interruptores, nos referimos a las teclas que son pulsadas por el usuario. Los mismos se encuentran organizados en una matriz, cada uno selecciona el tono del grupo bajo de su fila respectiva y el tono del grupo alto de su columna correspondiente, formando un tono único.

La codificación DTMF, asegura que cada señal contienen uno y solo un componente de cada uno de los grupos de tonos alto y bajo.
Esto simplifica de manera significativa la decodificación, ya que la señal compuesta DTMF, puede ser separada con filtros pasa banda en sus dos componentes de frecuencia simples cada uno de los cuales puede ser manipulado de forma individual.
En el caso de los símbolos especiales, alfabéticos, como A, B, C y D respectivamente, todos ellos tienen en común 1633 Hz como su tono alto de frecuencia.

Como mencionamos anteriormente, al ser pulsada en el teléfono la tecla correspondiente al dígito que quiere marcar, se envían dos tonos, de distinta frecuencias: uno por columna y otro por fila en la que esté la tecla, formando así una matriz. Entonces, tendremos al pulsar una tecla, el sonido resultante.

Vale aclarar que, cada una de las frecuencias transmitidas puede variar ± 1,8 % de la frecuencia nominal, por lo que los productos de distorsión, producidos por intermodulación o por generación de armónicos, deben tener un nivel 20 db, por debajo de los que tienen las frecuencias fundamentales.

La tabla que a continuación se muestra indica las frecuencias de los tonos por cada fila y columna del teclado de los teléfonos de marcación por tono.

	1209 Hz	**1336 Hz**	**1477 Hz**	**1633 Hz**
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	5	8	9	C
941 Hz	*	0	#	D

IMPLEMENTACION PARCIAL, PIC 33FMC202

MIKCRO C

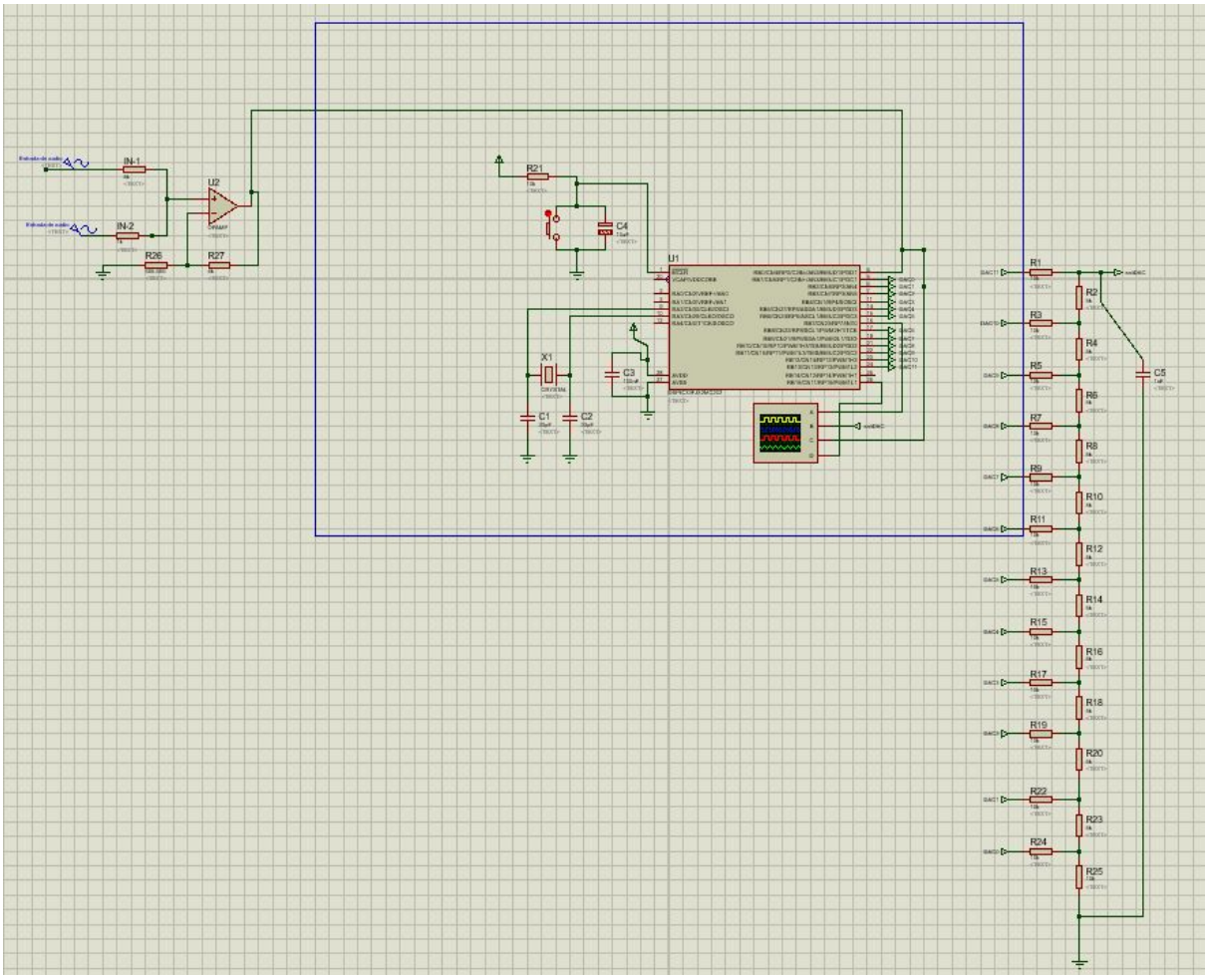
```
void config_adc()  {
   AD1CON2bits.VCFG = 0b000;        Referencia con Vcc y GND
   ADPCFG = 0xFFFB;                 Elije la entrada analogica a convertir en este caso AN2.
   AD1CHS0 = 0b0010;                El canal 0 es lo que se muestrea AN2
   AD1CON1bits.ADON = 0;            ADC apagado por ahora
   AD1CON1bits.AD12B = 1;           ADC  ( 0 para 10 bits - 1 para 12 bits
   AD1CON1bits.FORM = 0b00;         Formato de salida entero

  Para tomar muestras en forma manual. Porque lo vamos a controlar con Timer2
   AD1CON1bits.SSRC = 0b000;

Adquiere muestra cuando el SAMP se pone en 1. SAMP lo controlamos desde T2
   AD1CON1bits.ASAM = 0;
   AD1CON2bits.SMPI = 0b0000;  // Lanza interrupción luego de tomar n muestras.
}

void detectarInt_T2() org 0x0022  {
   IFS0bits.T2IF = 0;                Borramos la bandera de la interrupción
   AD1CON1bits.DONE = 0;             Antes de pedir una muestra ponemos en cero
   AD1CON1bits.SAMP = 1;             Pedimos una muestra
   asm nop;                          Tiempo que debemos esperar para que tome una muestra
   asm nop;
   asm nop;
   AD1CON1bits.SAMP = 0;            Pedimos que retenga la muestra
   LATBbits.LATB7 = !LATBbits.LATB7;
}
```

```
void detect_adc() org 0x002e  {

   IFS0bits.AD1IF = 0;                          Borramos el flag de interrupciones del ADC

   LATBbits.LATB15 = !LATBbits.LATB15;   Para debug de la interrupción ADC

   LATBbits.LATB13 =  (ADCBUF0 & 0b0000100000000000) >> 11;
   LATBbits.LATB12 =  (ADCBUF0 & 0b0000010000000000) >> 10;
   LATBbits.LATB11 =  (ADCBUF0 & 0b0000001000000000) >> 9;
   LATBbits.LATB10 =  (ADCBUF0 & 0b0000000100000000) >> 8;
   LATBbits.LATB9   =  (ADCBUF0 & 0b0000000010000000) >> 7;
   LATBbits.LATB8   =  (ADCBUF0 & 0b0000000001000000) >> 6;
   LATBbits.LATB6   =  (ADCBUF0 & 0b0000000000100000) >> 5;
   LATBbits.LATB5   =  (ADCBUF0 & 0b0000000000010000) >> 4;
   LATBbits.LATB4   =  (ADCBUF0 & 0b0000000000001000) >> 3;
   LATBbits.LATB3   =  (ADCBUF0 & 0b0000000000000100) >> 2;
   LATBbits.LATB2   =  (ADCBUF0 & 0b0000000000000010) >> 1;
   LATBbits.LATB1   =  (ADCBUF0 & 0b0000000000000001) >> 0;
}


void configuracionPuertos()  {
   TRISBbits.TRISB1 = 0;         Bit menos significativo de la señal generada
   TRISBbits.TRISB2 = 0;
   TRISBbits.TRISB3 = 0;
   TRISBbits.TRISB4 = 0;
   TRISBbits.TRISB5 = 0;
   TRISBbits.TRISB6 = 0;
   TRISBbits.TRISB8 = 0;
   TRISBbits.TRISB9 = 0;
   TRISBbits.TRISB10 = 0;
   TRISBbits.TRISB11 = 0;
   TRISBbits.TRISB12 = 0;
   TRISBbits.TRISB13 = 0;       Bit mas significativo de la señal generada

   TRISBbits.TRISB15 = 0;       Debug Timer ADC
   TRISBbits.TRISB0 = 1;
   TRISBbits.TRISB7 = 0;        Debug
}
```

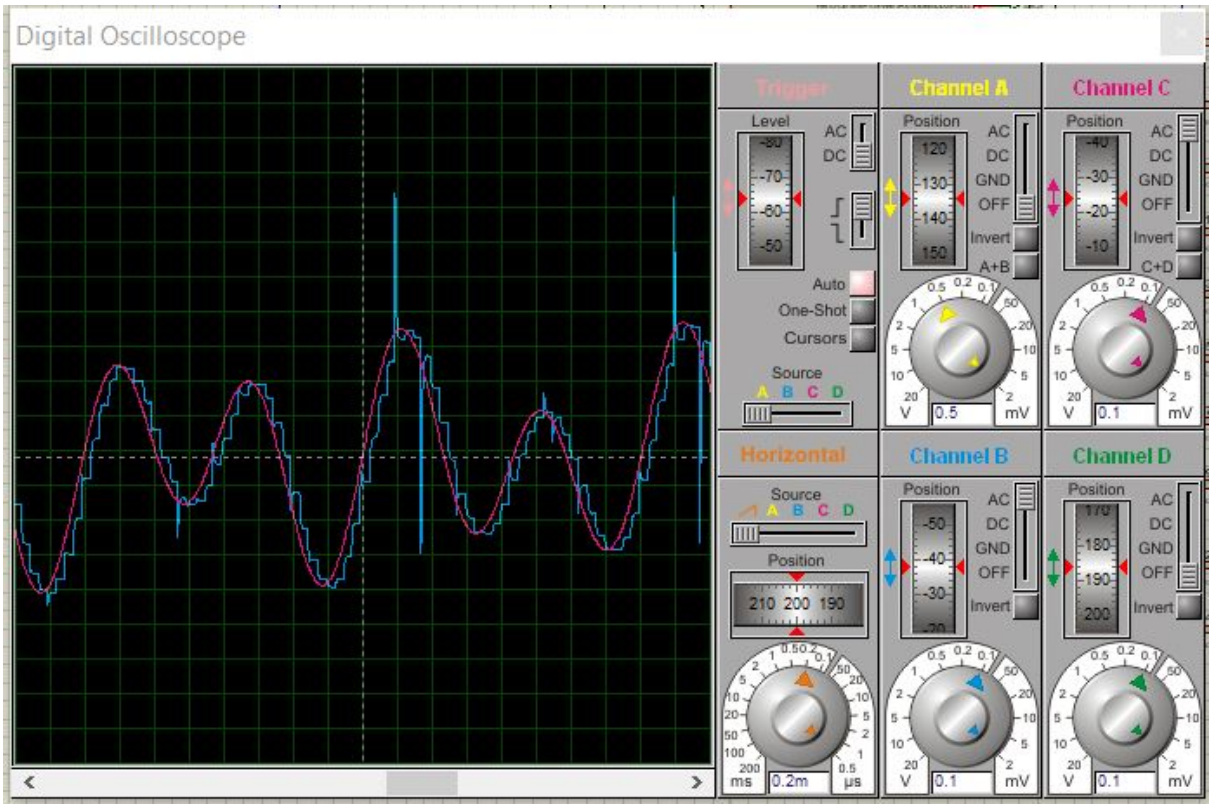
```
void configuracionT2()  {
   T2CONbits.TCKPS = 0b00;     Prescaler = 1:1
   PR2 = 250;                  Genera interrupción del Timer 2 a 20kHz
   IEC0bits.T2IE = 1;
}

int main()  {
   configuracionPuertos();
   configuracionT2();
   config_adc();
   IEC0bits.AD1IE = 1;           Habilitamos interrupcion del ADC
   T2CONbits.TON = 1;
   AD1CON1bits.ADON = 1; Encendemos el ADC

   while(1)  {
   }

   return 0;
}
```

PROTEUS

Diagrama general



SALIDA DEL OSCILOSCOPIO, PROTEUS



IMPLEMENTACIÓN FINAL, PIC 30F4013

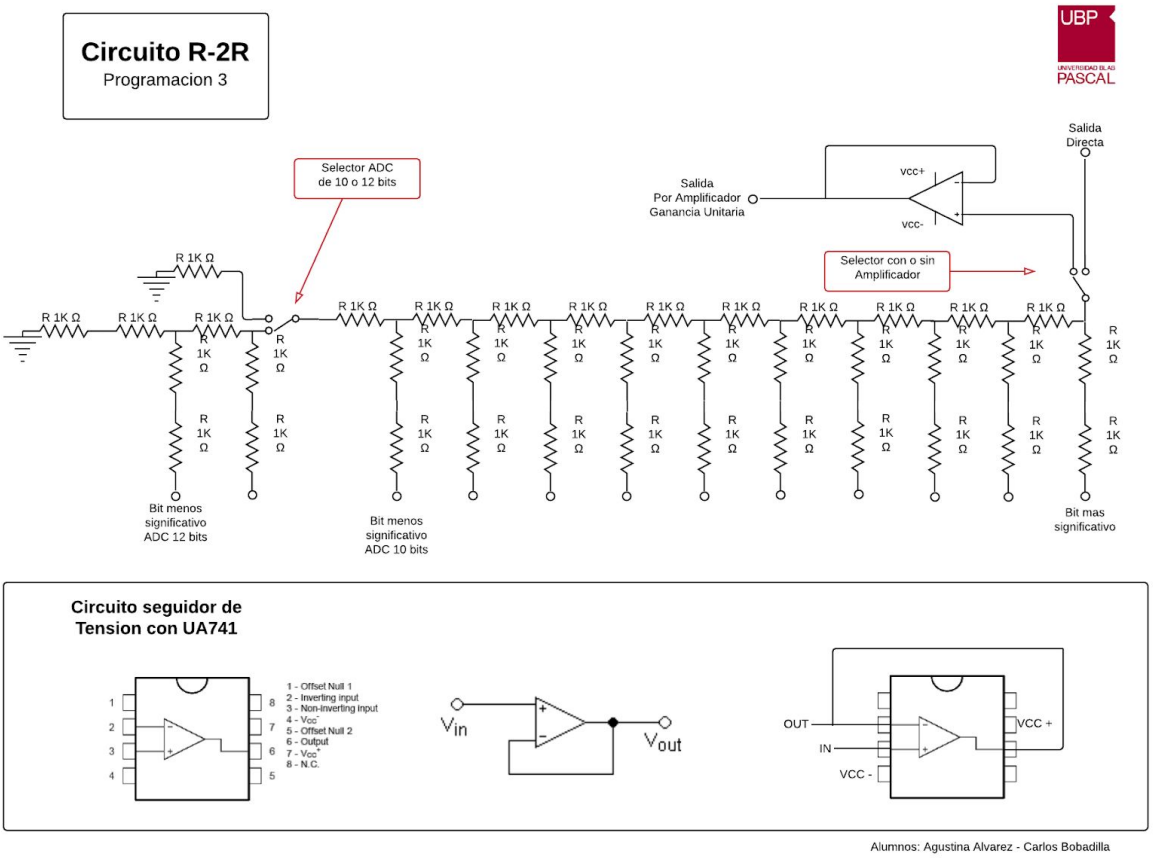
MIKCRO C

```
int valorActual = 0;

void configuracionPuertos()  {
   TRISBbits.TRISB0   = 0;
   TRISBbits.TRISB1   = 0;          Bit menos significativo de la señal generada
   TRISBbits.TRISB2   = 1;
   TRISBbits.TRISB3   = 0;
   TRISBbits.TRISB4   = 0;
   TRISBbits.TRISB5   = 0;
   TRISBbits.TRISB6   = 0;
   TRISBbits.TRISB8   = 0;
   TRISBbits.TRISB9   = 0;
   TRISBbits.TRISB10 = 0;
   TRISBbits.TRISB11 = 0;
   TRISBbits.TRISB12 = 0;
   TRISCbits.TRISC13 = 0;            Bit más significativo de la señal generada

   TRISDbits.TRISD1 = 0;             Debug Timer ADC

   TRISBbits.TRISB7 = 0;             Debug

   TRISDbits.TRISD2 = 0;             Debug
}

void config_adc()  {
   ADCON2bits.VCFG = 0b000;    Referencia con Vcc y GND
   ADPCFG = 0xFFFB;            Elije la entrada analogica a convertir en este caso AN2.
   ADCHS = 0b0010;             El canal 0 es lo que se muestrea AN2
   ADCON1bits.ADON = 0;        ADC apagado por ahora
   ADCON1bits.ADSIDL = 1;      No trabaja en modo idle
   ADCON1bits.FORM = 0b00;     Formato de salida entero

   Para tomar muestras en forma manual. Porque lo vamos a controlar con Timer2
   ADCON1bits.SSRC = 0b000;

   Adquiere muestra cuando el SAMP se pone en 1. SAMP lo controlamos desde T2
```

```
   ADCON1bits.ASAM = 0;

   ADCON2bits.SMPI = 0b0000;  Lanza interrupción luego de tomar n muestras.
  Con SMPI=0b0 -> 1 muestra ; Con SMPI=0b1 -> 2 muestras ; Con SMPI=0b10 -> 3 muestras
}

void detectarInt_T2() org 0x0020  {
   IFS0bits.T2IF = 0;              Borramos la bandera de la interrupcion
   ADCON1bits.DONE = 0;      Antes de pedir una muestra ponemos en cero
   ADCON1bits.SAMP = 1;      Pedimos una muestra
   asm nop;                        Tiempo que debemos esperar para que tome una muestra
  ADCON1bits.SAMP = 0;       Pedimos que retenga la muestra
   LATBbits.LATB7 = !LATBbits.LATB7;
}

void detect_adc() org 0x002a  {

   IFS0bits.ADIF = 0;              Borramos el flag de interrupciones

   LATDbits.LATD1 = !LATDbits.LATD1;

   valorActual = 2048;     valorActual = ADCBUF0;

   LATCbits.LATC13 =  (valorActual & 0b0000100000000000) >> 11;
   LATBbits.LATB12 =  (valorActual & 0b0000010000000000) >> 10;
   LATBbits.LATB11 =  (valorActual & 0b0000001000000000) >> 9;
   LATBbits.LATB10 =  (valorActual & 0b0000000100000000) >> 8;
   LATBbits.LATB9   =  (valorActual & 0b0000000010000000) >> 7;
   LATBbits.LATB8   =  (valorActual & 0b0000000001000000) >> 6;
   LATBbits.LATB6   =  (valorActual & 0b0000000000100000) >> 5;
   LATBbits.LATB5   =  (valorActual & 0b0000000000010000) >> 4;
   LATBbits.LATB4   =  (valorActual & 0b0000000000001000) >> 3;
   LATBbits.LATB3   =  (valorActual & 0b0000000000000100) >> 2;
   LATBbits.LATB1   =  (valorActual & 0b0000000000000010) >> 1;
   LATBbits.LATB0   =  (valorActual & 0b0000000000000001) >> 0;
}

void configuracionT2()  {
   T2CONbits.TCKPS = 0b00;    Prescaler = 1:1
   PR2 = 500;                      Genera interrupción del Timer 2 a 40kHz

   IEC0bits.T2IE = 1;
```

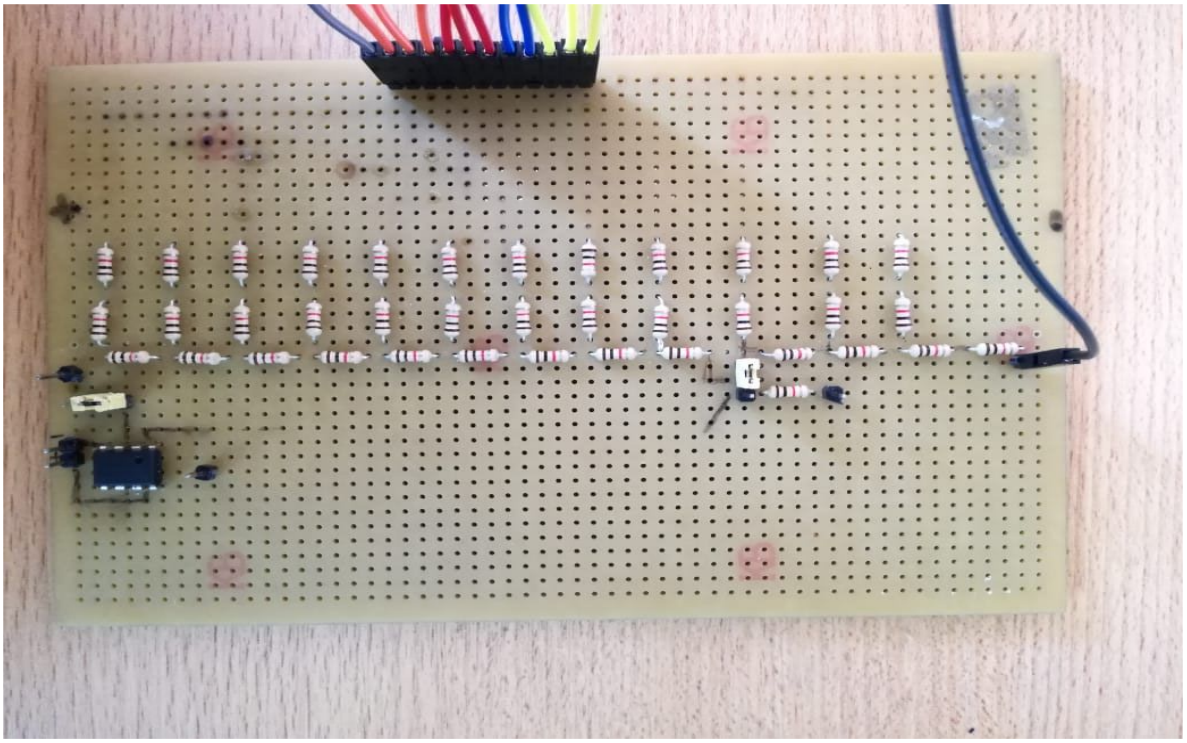
```
}

int main()  {
   configuracionPuertos();
   configuracionT2();
   config_adc();

   LATDbits.LATD2 = 1;

   IEC0bits.ADIE = 1;          Habilitamos interrupcion del ADC
   IEC0bits.T2IE=1;             Habilita interrupciones timer2
   T2CONbits.TON = 1;         Arranca el clock 2
   ADCON1bits.ADON = 1;   Encendemos el ADC

   while(1)  {
   }

   return 0;
}
```

DESARROLLO DEL R-2R



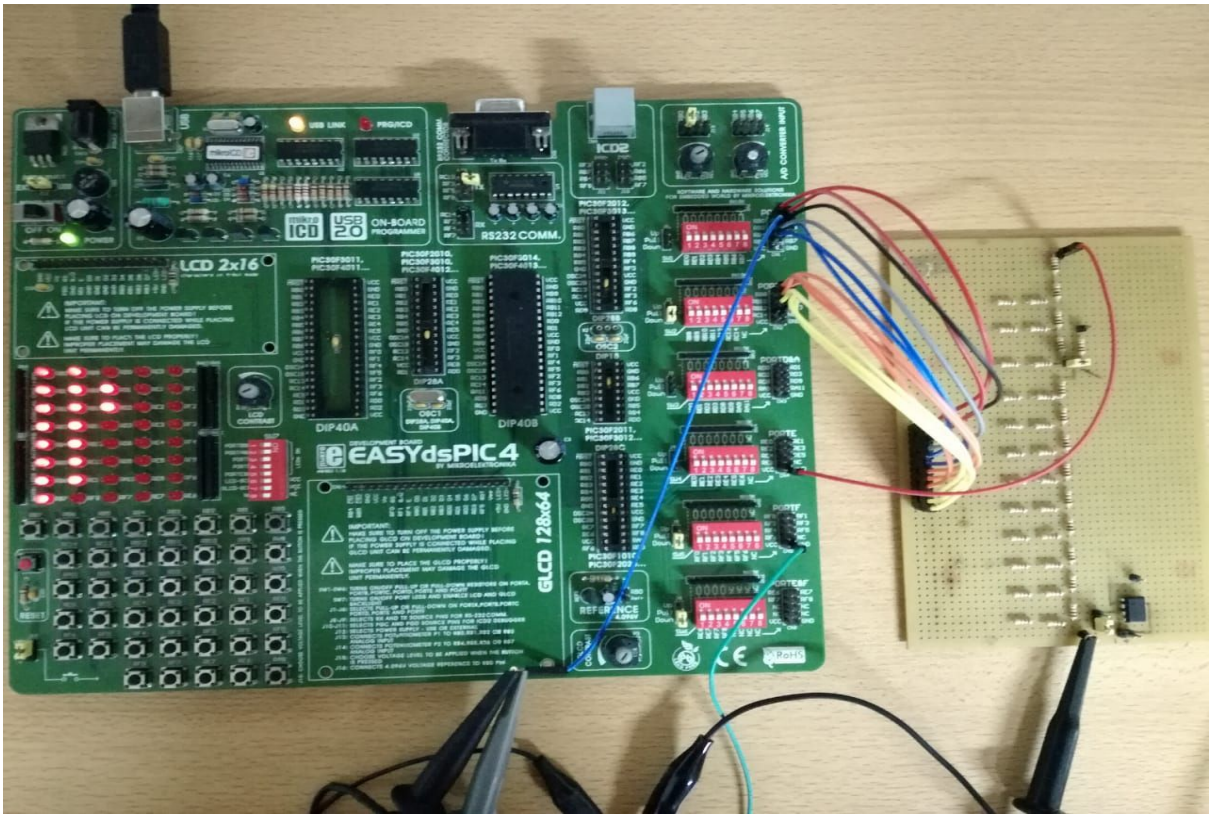
El circuito plasmado anteriormente, representa al circuito montado en la placa pcb.
Un circuito R-2R, está formado por resistencias alternando dos valores posibles, donde un valor debe ser el doble del otro. En el caso del trabajo propuesto, se utilizan resistencias de 1kohm.
Un circuito de este tipo, permite una forma simple y económica de implementar un DAC, conversor analogico digital.
Se le incorporó un amplificador operacional en el r-2r para aislar el circuito de la carga que se coloque a la salida, cumpliendo la funcion tambien de atenuador.
En la placa se colocaron selectores, para indicar como se ve en la imagen, con cuantos bits trabajaremos y si la salida será directa o si será por el amplificador de ganancia unitaria.

PLACA R-2R



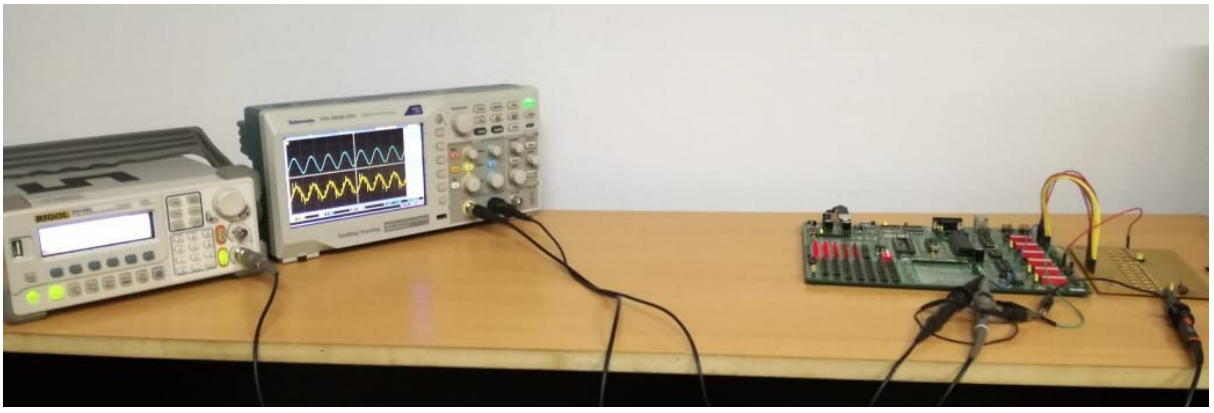
CONEXIONADO

PLACA R-2R Y EASYdsPIC4

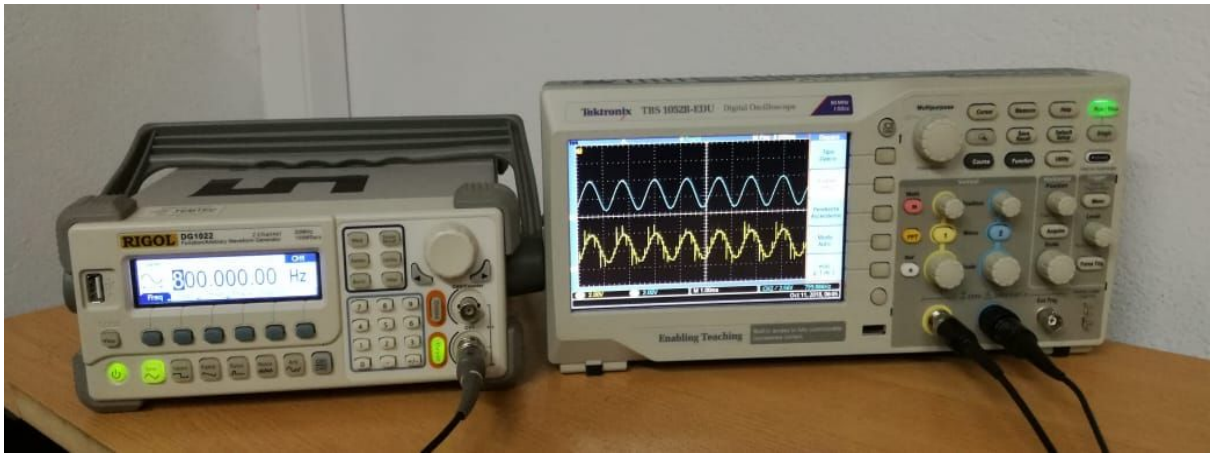


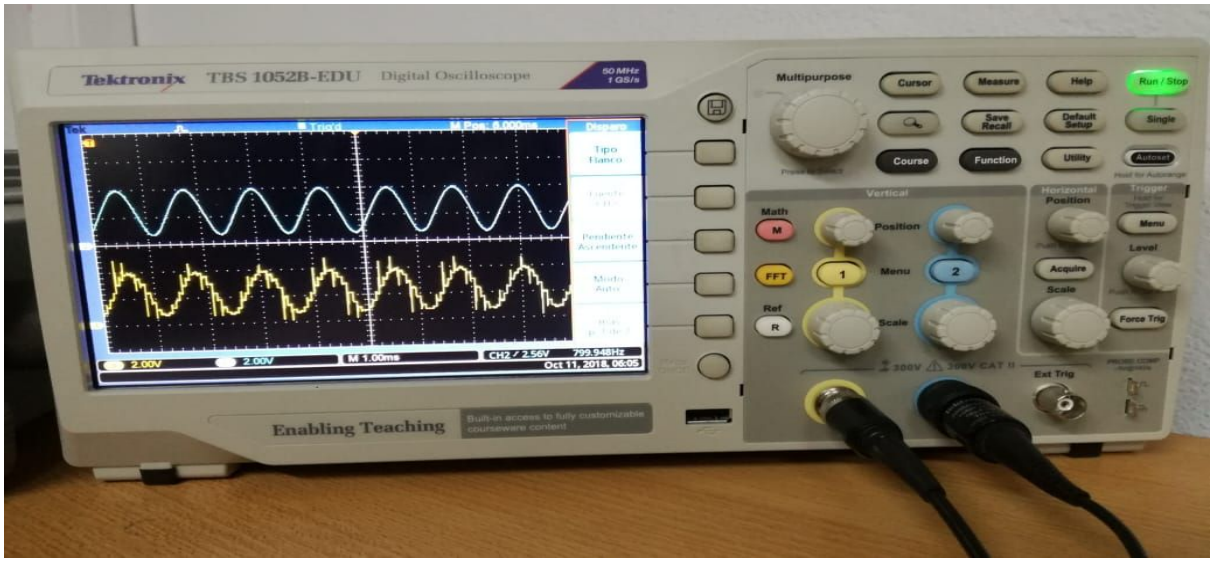
CONEXIONADO FINAL

Se trabaja con una frecuencia de muestreo de 20khz.

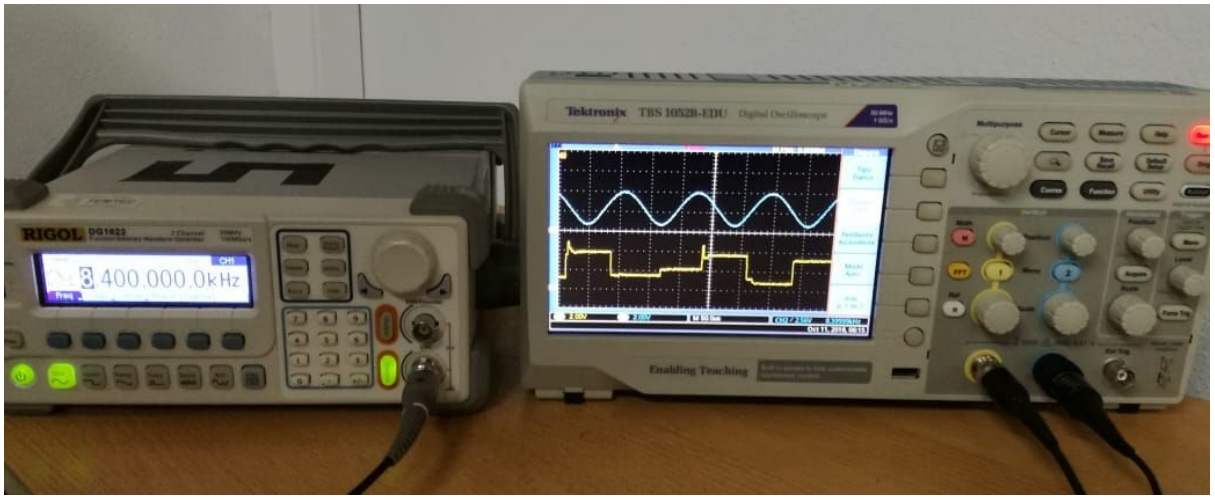


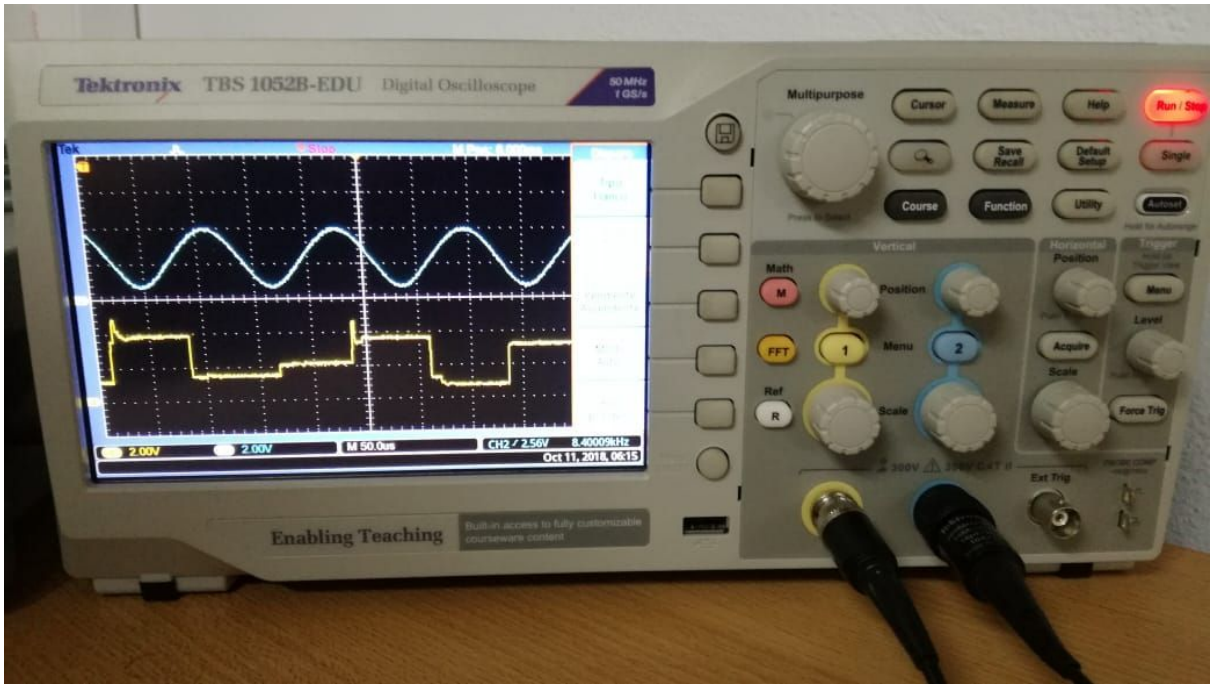
Trabajando con una frecuencia de 800 HZ, se obtiene:





Ahora bien si aumenta la frecuencia, por ejemplo a 8.400 khz, obtendremos:





A medida que aumentamos la frecuencia, menos muestras por ciclo se toman, por lo que se pueden identificar señales de hasta la mitad de frecuencia de muestreo.
La señal se torna de tipo cuadrada como se visualiza en la imagen anterior.