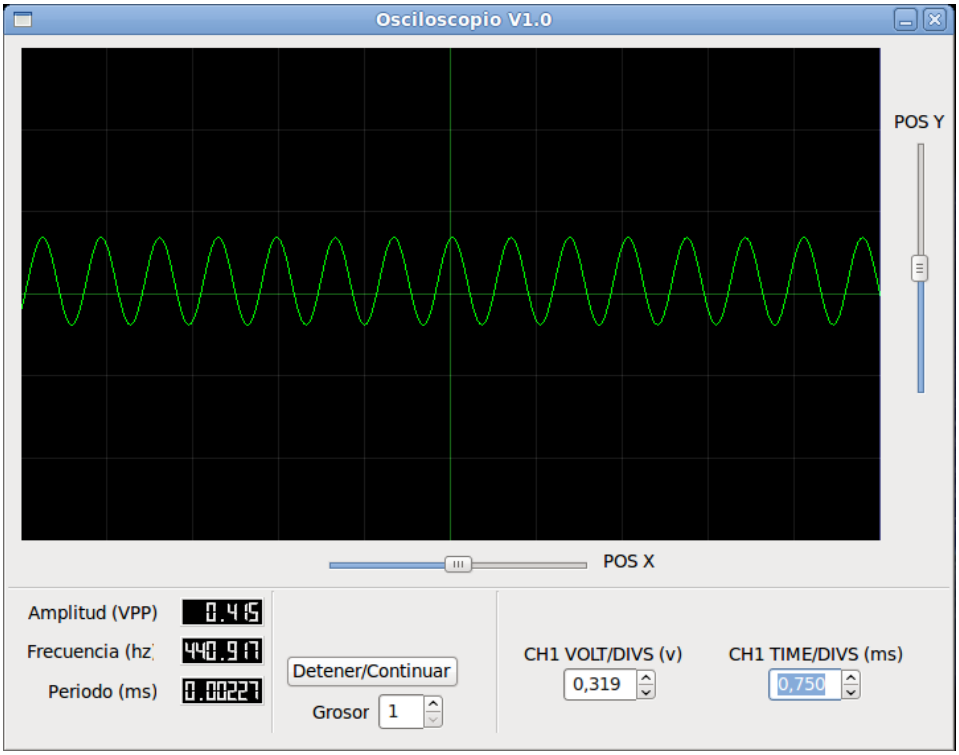
Universidad de Carabobo
Facultad Experimental de Ciencia y Tecnología
Departamento de Computación
Redes de Computadores
Prof. Antonio Castañeda



OsciloscopioMC

Yanina Aular, 19 588 966
Luis González, 19 919 029
Osval Reyes, 20 512 105
Richard Rodríguez, 19 553 202

Viernes, 11 de Febrero 2011

Introducción

Un osciloscopio es un tipo de instrumento electrónico que permite visualizar voltage en función del tiempo. Su uso es aplicable a diversos campos, tales como la química, física, electrónica, entre otros.

OsciloscopioMC es una aplicación que actúa como un osciloscopio electrónico; solo que, a diferencia de este último, no es un instrumento electrónico, si no un software que, haciendo uso de la tarjeta de sonido en donde se utilice, pretende cumplir con la misma función.

El presente documento explicará el funcionamiento de *OsciloscopioMC*, así como su proceso de desarrollo, y todo lo concerniente al software que hay en este CD. Adicionalmente, se abordarán conceptos necesarios para una mejor comprensión del tema, tales como onda, frecuencia, amplitud, entre otros.

Justificación

Un osciloscopio es una herramienta de mucha utilidad, necesario en dibersos campos. Debido a que este dispositivo normalmente tiene un costo elevado, hemos decidido hacer un software que, en la medida de lo posible, cumpla el mismo propósito; usando para ello la tarjeta de sonido del equipo en donde se va a utilizar; con esto, evitamos los costos del osciloscopio y facilitamos el uso y la configuración al momento de su utilización.

Limitaciones:

Debido a que se utiliza la tarjeta de sonido para el funcionamiento del *OsciloscopioMC*, está sujeto a ciertas restricciones, a saber:

Voltaje: El voltaje soportado por el Osciloscopio se encuentra entre 0V y 0.5V, lo que podría ser insuficiente dependiendo del uso que se desee darle.

Frecuencia de entrada: debido a que la tarjeta interpreta señales discretas (digitales) la frecuencia de entrada máxima que se puede procesar, estará proporcionalmente limitada por la frecuencia máxima de muestreo que soporte dicha tarjeta. Usualmente, este valor es de 48000 Hz; no obstante, para este proyecto utilizaremos 44100 Hz, ya que este valor permite procesar con precisión una frecuencia de 20000 Hz, que es mas que suficiente para el propósito de este proyecto.

Marco Teórico

Onda: Es la propagación de una perturbación (energía) a través de un medio.

Amplitud: Distancia entre el punto más alejado de una onda y el punto de equilibrio o medio.

Amplitud pico a pico: Magnitud que representa la diferencia entre el valor máximo y el valor mínimo de una onda.

Frecuencia: Magnitud cuantitativa de repeticiones por unidad de tiempo de cualquier fenómeno o suceso periódico.

Frecuencia de muestreo: Es el número de muestras por unidad de tiempo que se toman de de una señal continua (analógica) para producir una señal discreta (digital).

Hertz (Hz): Unidad de Frecuencia y según el Sistema Internacional (S.I.), expresa la cantidad de ciclos de reloj por segundo.

Voltio: Unidad que mide la tensión y según el Sistema Internacional (S.I.), expresa la diferencia de potencial eléctrico.

Voltaje pico a pico: Magnitud que representa la diferencia de voltaje entre el máximo y el mínimo.

Período: Cantidad de tiempo necesario para completar un ciclo.

Ancho de banda: Indican el rango de frecuencias que el osciloscopio puede medir con exactitud.

Tipos de ondas.

- Onda senoidal: También llamada Sinusoidal. Se trata de una señal analógica, puesto que existen infinitos valores entre dos puntos cualesquiera del dominio. De hecho, su representación es la gráfica de la función matemática Seno.

- Onda cuadrada: Onda que alterna su valor entre dos valores extremos, sin pasar por los valores intermedios, a diferencia de lo que ocurre con la onda sinusoidal.

- Otras: Existen muchos otros tipos de ondas, tales como triangular, cuadrática, sierra, aleatoria, entre otras; pero no serán abordadas en esta

GUI: Interfaz Gráfica de Usuario, ventanas con las que el usuario interactúa con las aplicaciones.

Qt: Biblioteca para interfaces gráficas, desarrollada por Trolltech y posteriormente comprada por Nokia.

Alsa: (Advanced Linux Sound Architecture) es un componente del núcleo GNU/Linux, que se encarga del manejo de dispositivos de sonido y su interacción con las aplicaciones.

PThreads: Biblioteca estándar para el manejo de hilos POSIX.

Marco metodológico

En esta fase del presente trabajo se muestran técnicas y el proceso de desarrollo que fue aplicado para el *OsciloscopioMC*hasta su versión 1.0. A continuación se describirá el proceso.

– Investigar sobre las bibliotecas disponibles para el manejo del sonido y para la interfaz gráfica. Luego de unos días de investigación sobre las distintas maneras de construir la interfaz gráfica, el equipo de desarrollo optó por usar sobre C++, Qt4 y OpenGL para las gráficas. Y para el manejo del sonido, el equipo decidió utilizar la librería compartida para aplicaciones ALSA, asoundlib2, por las facilidades de uso y excelente documentación en su página oficial del Proyecto ALSA.

- Con los elementos base para el inicio del desarrollo, parte del equipo comenzó la lectura de la documentación referencia de la API de ALSA, probando al mismo tiempo los ejemplos para el uso del dispositivo, como son: la apertura, modos de apertura y escritura en el mismo.

– Una vez decididas las bibliotecas que se iban a utilizar e investigado su funcionamiento, el equipo comenzó a experimentar leyendo datos del micrófono para el caso de ALSA, y a intentar graficarlos con OpenGL.

– Para poder graficar los datos obtenidos del micrófono de la tarjeta de sonido, se aplicó una transformada, de manera de que. en la gráfica se correspondieran con el voltaje de entrada. Adicionalmente, se necesitó truncar algunos decimales de los datos transformados, de manera de calcular con mayor precisión la frecuencia de entrada, y evitar que la máquina complete dichos decimales con números basura.

Desarrollo

Como producto de la investigación, estudio y análisis; se desarrolló el osciloscopio *OsciloscopioMC* en su version 1.0, y a continuación describiremos su funcionamiento.

1. Gráfica: Se visualizará la forma de onda, el tiempo se encontrará sobre el eje horizontal y la amplitud
sobre el eje vertical.

2. Pos Y: Nos permite modificar la posición de la forma de onda en el eje Y.

3. Pos X: Nos permite modificar la posición de la forma de onda en el eje X.

4. CH1 TIME/DIVS (ms): Nos permite modificar la cantidad de tiempo en milisegundos que existe por división sobre el eje X.

5. CH1 VOLT/DIVS (V): Nos permite modificar la cantidad de voltios que existe por división sobre el eje Y.

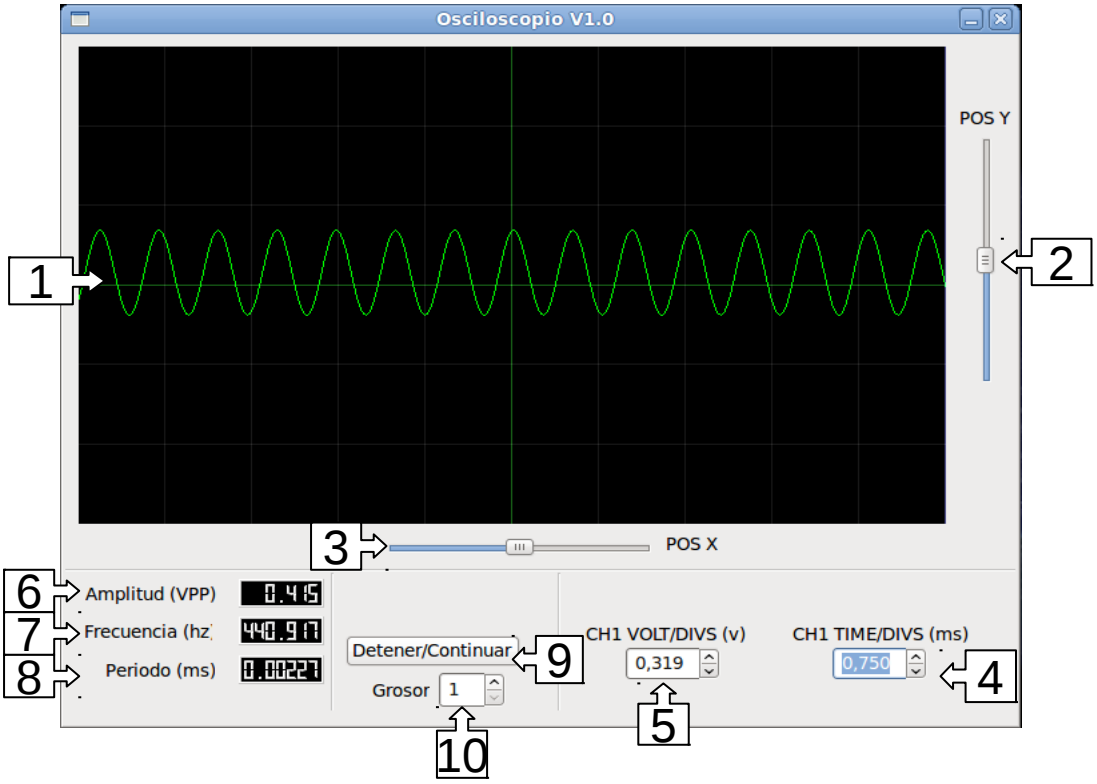
6. Amplitud: Se muestra la amplitud de la onda medida en voltios pico.

7. Frecuencia: Se muestra la frecuencia de la onda. Si una señal se repite, tiene una frecuencia. La frecuencia se mide en hert-zios (Hz) y es igual al número de veces que una señal se repite en un
segundo (ciclos por segundo).

8. Periodo: Se muestra el periodo de la onda, esto es la cantidad de tiempo necesario para completar un ciclo.

9. Detener/Continuar: Cuando la forma de onda se encuentra en movimiento, éste botón nos permite detenerla para poder visualizarla de manera estática. Cuando se encuentra detenida, la función del botón nos permite animar la onda nuevamente.

10. Grosor: Modifica la intensidad de la onda, el rango de valores se encuentra entre 1,0 y 5,0 de grosor.



Conclusiones y Recomendaciones

Luego de completar el desarrollo del software OsciloscopioMC, podemos concluir que:

– En muchos casos, no es necesario adquirir un osciloscopio, ya que puede ser reemplazado por nuestro software o uno equivalente, lo que se reflejaría en disminución de costos.

- Sí se paralelizan las distintas actividades de un programa, utilizando múltiples hilos, mejorará notablemente la eficiencia; especialmente en aquellos equipos que tengan varios núcleos de procesamiento.

Referencias Bibliográficas

http://www2.fices.unsl.edu.ar/~areaeyc/lme/services/Conceptos_Basicos_Osciloscopios.pdf

http://www.todoespia.com/productos/manuales/revistaTE/osciloscopio.pdf

http://es.wikipedia.org/wiki/Arquitectura_de_Sonido_Avanzada_para_Linux

Anexos

Anexo 1: main.cpp

```
#include <QApplication>

#include "ventana.h"

#include "utilidades.h"



int main(int argc, char *argv[])

{

	//Inicializa la aplicacion

	QApplication app(argc, argv);

	//Crea la ventana

	Ventana window;

	//Muestra la ventana por pantalla

	window.show();

	return app.exec();

}
```

Anexo 2: ventana.h

```
#ifndef VENTANA_H

#define VENTANA_H



#include <QtGui>

#include <QButtonGroup>

#include <QtCore/QVariant>

#include <QtGui/QAction>

#include <QtGui/QApplication>

#include <QtGui/QButtonGroup>

#include <QtGui/QHeaderView>

#include <QtGui/QLCDNumber>

#include <QtGui/QLabel>

#include <QtGui/QPushButton>

#include <QtGui/QSlider>

#include <QtGui/QWidget>

#include <qwt_knob.h>

#include <QSpinBox>

#include <QWidget>

#include "Escena.h"
```

```
class Ventana : public QWidget

{

   Q_OBJECT



        public:

                Escena *opengl_plot;

                QDoubleSpinBox *SpinBox_VoltDivs;

        QDoubleSpinBox *SpinBox_TimeDivs;

                QSpinBox       *SpinBox_Grosor;

                QLabel *label_amplitud;

                QLabel *label_frecuencia;

                QLabel *Label_Periodo;

                QLCDNumber *lcd_amplitud;

                QLCDNumber *lcd_frecuencia;

                QLCDNumber *Lcd_Periodo;

                QPushButton *button_detener_continuar;

                QSlider *slider_pos_x;

                QLabel *label_pos_x;

                QSlider *slider_pos_y;
```

```
        QLabel *labe_pos_y;
        QLabel *label_ch1_volts_div;
        QLabel *label_ch1_time_divs;
        QLabel *Label_Intensidad;
        QFrame *line;
        QFrame *line_2;
        QFrame *line_3;
        QFrame *line_4;
        Ventana();

    public slots:
        void actualizar_fre_y_ampli();
};
#endif
```

Anexo 3: ventana.cpp

```
#include "ventana.h"

Ventana::Ventana(): QWidget()
{
	setFixedSize(712, 550); //Tamanio de la Ventana
	setWindowTitle(tr("OsciloscopioMC")); //Nombre ventana
	opengl_plot = new Escena(this);//Se agrega el plano cartesiano a la ventana
	opengl_plot->setGeometry(QRect(10, 10, 650, 381));

	//Etiqueta que corresponde a la funcion que muestra la amplitud
	label_amplitud = new QLabel("Amplitud (VPP)",this);
	label_amplitud->setObjectName(QString::fromUtf8("label_amplitud"));
	label_amplitud->setGeometry(QRect(15, 440, 100, 16));

	//Etiqueta que corresponde a la funcion que muestra la frecuencia
	label_frecuencia = new QLabel("Frecuencia (hz)",this);
	label_frecuencia->setObjectName(QString::fromUtf8("label_frecuencia"));
	label_frecuencia->setGeometry(QRect(14, 470, 96, 16));

	//Pantallita que muestra la amplitud actual de la onda
	lcd_amplitud = new QLCDNumber(this);
	lcd_amplitud->setObjectName(QString::fromUtf8("lcd_amplitud"));
	lcd_amplitud->setGeometry(QRect(130, 435, 64, 23));
	lcd_amplitud->setNumDigits(7);
	lcd_amplitud->setAutoFillBackground(true);

	//Pantallita que muestra la frecuencia actual de la onda
	lcd_frecuencia = new QLCDNumber(this);
	lcd_frecuencia->setObjectName(QString::fromUtf8("lcd_frecuencia"));
	lcd_frecuencia->setGeometry(QRect(130, 465, 64, 23));
	lcd_frecuencia->setNumDigits(7);
	lcd_frecuencia->setAutoFillBackground(true);

	//Etiqueta que corresponde a la funcion que muestra el periodo
	Label_Periodo = new QLabel("Periodo (ms)",this);
	 Label_Periodo->setObjectName(QString::fromUtf8("label"));
	Label_Periodo->setGeometry(QRect(30, 500, 82, 17));

	//Pantallita que muestra el periodo actual de la onda
	Lcd_Periodo = new QLCDNumber(this);
	Lcd_Periodo->setObjectName(QString::fromUtf8("Lcd_Periodo"));
	Lcd_Periodo->setGeometry(QRect(130, 496, 64, 23));
	Lcd_Periodo->setNumDigits(7);
	Lcd_Periodo->setAutoFillBackground(true);

	//Se cambia los colores de las pantallitas con la informacion de la onda
	QPalette Pal = lcd_amplitud->palette();
	Pal.setColor(QPalette::Normal, QPalette::Window, Qt::black);
	lcd_amplitud->setPalette(Pal);
	lcd_frecuencia->setPalette(Pal);
	Lcd_Periodo->setPalette(Pal);

	//Se crea el boton que permite detener y animar la forma de onda
	button_detener_continuar = new QPushButton("Detener/Continuar",this);
```

```
button_detener_continuar->setObjectName(QString::fromUtf8("button_detener_continuar"));
button_detener_continuar->setGeometry(QRect(210, 480, 131, 25));

//Se crean y configuran los slider respectivos que se encargan del movimiento
//de lo onda a traves de los ejes X y Y
slider_pos_x = new QSlider(this);
slider_pos_x->setObjectName(QString::fromUtf8("slider_pos_x"));
slider_pos_x->setGeometry(QRect(240, 400, 201, 20));
slider_pos_x->setOrientation(Qt::Horizontal);
slider_pos_x->setRange(-100, 100);
slider_pos_x->setValue(0);
label_pos_x = new QLabel("POS X",this);
label_pos_x->setObjectName(QString::fromUtf8("label_pos_x"));
label_pos_x->setGeometry(QRect(450, 400, 41, 16));
slider_pos_y = new QSlider(this);
slider_pos_y->setObjectName(QString::fromUtf8("slider_pos_y"));
slider_pos_y->setGeometry(QRect(680, 80, 20, 201));
slider_pos_y->setOrientation(Qt::Vertical);
slider_pos_y->setRange(-100,100);
slider_pos_y->setValue(0);

//Creamos el SpinBox que permite cambiar la cantidad de voltios por division
SpinBox_VoltDivs = new QDoubleSpinBox(this);
SpinBox_VoltDivs->setObjectName(QString::fromUtf8("doubleSpinBox"));
 SpinBox_VoltDivs->setGeometry(QRect(420, 490, 71, 25));
SpinBox_VoltDivs->setDecimals(3);
SpinBox_VoltDivs->setRange(0.00625,0.50);
SpinBox_VoltDivs->setValue(0.25);
SpinBox_VoltDivs->setSingleStep(0.00625);

//Creamos el SpinBox que permite cambiar la cantidad de tiempo por division
 SpinBox_TimeDivs = new QDoubleSpinBox(this);
 SpinBox_TimeDivs->setObjectName(QString::fromUtf8("doubleSpinBox_2"));
SpinBox_TimeDivs->setGeometry(QRect(575, 490, 71, 25));
SpinBox_TimeDivs->setDecimals(3);
SpinBox_TimeDivs->setRange(0.125,2.500);
SpinBox_TimeDivs->setSingleStep(0.125);
SpinBox_TimeDivs->setValue(1.250);

//Creamos el SpinBox que permite cambiar la intensidad de la onda
SpinBox_Grosor = new QSpinBox(this);
SpinBox_Grosor->setRange(1,5);
SpinBox_Grosor->setValue(1);
SpinBox_Grosor->setGeometry(QRect(280, 510,  50, 27));

//Etiquetas con los nombres de cada funcion
labe_pos_y = new QLabel("POS Y",this);
labe_pos_y->setObjectName(QString::fromUtf8("labe_pos_y"));
labe_pos_y->setGeometry(QRect(670, 60, 41, 16));
label_ch1_volts_div = new QLabel("CH1 VOLT/DIVS (v)",this);
label_ch1_volts_div->setObjectName(QString::fromUtf8("label_ch1_volts_div"));
label_ch1_volts_div->setGeometry(QRect(390, 470, 132, 20));
label_ch1_time_divs = new QLabel("CH1 TIME/DIVS (ms)",this);
label_ch1_time_divs->setObjectName(QString::fromUtf8("label_ch1_time_divs"));
label_ch1_time_divs->setGeometry(QRect(545, 470, 140, 20));
line = new QFrame(this);
line->setObjectName(QString::fromUtf8("line"));
```

```
        line->setGeometry(QRect(0, 420, 751, 16));
        line->setFrameShape(QFrame::HLine);
        line->setFrameShadow(QFrame::Sunken);
        line_2 = new QFrame(this);
        line_2->setObjectName(QString::fromUtf8("line_2"));
        line_2->setGeometry(QRect(360, 430, 20, 110));
        line_2->setFrameShape(QFrame::VLine);
        line_2->setFrameShadow(QFrame::Sunken);

        line_4 = new QFrame(this);
        line_4->setObjectName(QString::fromUtf8("line_4"));
        line_4->setGeometry(QRect(190, 430, 20, 110));
        line_4->setFrameShape(QFrame::VLine);
        line_4->setFrameShadow(QFrame::Sunken);

        QMetaObject::connectSlotsByName(this);

        //Actualiza la forma de onda cada 5 microsegundos
        QTimer *timer = new QTimer(this);
        connect(timer, SIGNAL(timeout()), opengl_plot, SLOT(animate()));
        timer->start(50);

        //Actualiza la frecuencia y la amplitud cada 2 segundos
        QTimer *timer2 = new QTimer(this);
        connect(timer2, SIGNAL(timeout()), this, SLOT(actualizar_fre_y_ampli()));
        timer2->start(2000);

        //Etiqueta del nombre de la funcion Grosor
        Label_Intensidad = new QLabel("Grosor",this);
        Label_Intensidad->setGeometry(QRect(230,517,43,16));

        //Se configuran los evento de boton y otras funciones del osciloscopioMC
        connect(SpinBox_Grosor, SIGNAL(valueChanged(int)), opengl_plot, SLOT(setGrosor(int)) );
        connect(SpinBox_TimeDivs,SIGNAL(valueChanged(double)), opengl_plot, SLOT(setMult_dt(double)) );
        connect(SpinBox_VoltDivs,SIGNAL(valueChanged(double)), opengl_plot, SLOT(setMult_volt(double)) );
        connect(slider_pos_y, SIGNAL(valueChanged(int)), opengl_plot, SLOT(setposY(int)));
        connect(slider_pos_x, SIGNAL(valueChanged(int)), opengl_plot, SLOT(setposX(int)));
        connect(button_detener_continuar, SIGNAL(clicked()), opengl_plot, SLOT(detener()) );

}
//Nos permite actualizar los valores de la frecuencia y la amplitud en las pantallas
void Ventana::actualizar_fre_y_ampli()
{
        if(lcd_amplitud->value()!=opengl_plot->getAmplitud())
                lcd_amplitud->display(opengl_plot->getAmplitud());

        if(lcd_frecuencia->value()!=opengl_plot->getFrecuencia()){
                lcd_frecuencia->display(opengl_plot->getFrecuencia());
                Lcd_Periodo->display(1/opengl_plot->getFrecuencia() );
        }
}
```

Anexo 4: Escena.h

```
#ifndef ESCENA_H

#define ESCENA_H



#include <QGLWidget>

#include "EntradaAudio.h"



class Escena : public QGLWidget

{

	Q_OBJECT



	public:

		float lados[4]; // arriba, abajo, derecha, izquierda;

		float target_side[4];

		float trigger[2];

		float trigger_width;

		int t_dir;

		EntradaAudio* ea;

		float tiempos[3];  // comienzo, parada, trigger

		int indices[3];
```

```
        float grosor;
        char mult_dt;
        float mult_volt;
        float posY,posX;
        float valores[100]; //valores de amplitud predefinidos

private:
        int elapsed;

public:
        Escena(QWidget *parent);
        void calcular_frecuencia();
        void setProjection() ;
        void drawPlot();
        void dibujar_rejillas();
        void setgrosor(float g);
        float getAmplitud();
        float getFrecuencia();
        void scale(int ax, float s);
```

```
        public slots:
                void animate();
                void detener();
                void setGrosor(int g);
                void setMult_dt(double valor);
                void setMult_volt(double valor);
                void setposY(int valor);
                void setposX(int valor);

        protected:
                void paintGL();
};

#endif
```

Anexo 5: Escena.cpp

```
#include <GL/glut.h>

#include <GL/gl.h>

#include <alsa/asoundlib.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <pthread.h>

#include <sys/time.h>

#include <math.h>

#include "Escena.h"



Escena::Escena(QWidget *parent): QGLWidget(QGLFormat(QGL::SampleBuffers), parent)

{

            int i;

            float valor=0.0;



   elapsed = 0;

            t_dir = 1;

            lados[0] = 0.3;

            lados[1] = -0.3;

            lados[2] = 0;

            lados[3] = -0.1;



            for(i=0; i<4; i++ )
```

```
                target_side[i] = lados[i];


        trigger[0] = 0.0;
        trigger[1] = -0.05;
        trigger_width = 0.0002;

        ea = new EntradaAudio();
        grosor = 1;
        mult_dt=10;
        mult_volt=2.0;
        posY=posX=0;

        for(i=0;i<100;i++){
                valores[i] = valor;
                valor = valor + 0.005;
        }
}

void Escena::calcular_frecuencia() {

        double aux;
        double t_ini;
    tiempos[0] = ea->data_size * ea->dt;
    tiempos[1] = 0;
    ea->getTimeSpan(tiempos, indices, 3);
```

```
aux=0;
t_ini=0;

ea->amplitud=0;
int tiempo1=0;
	int tiempo2 =0;
	int contador=1;
	int k=0,band=1;

	//determinar la amplitud mayor
	for( int i=indices[0]; i<indices[1]; i++ ){
		if(ea->amplitud<ea->float_data[ea->index(i)]){
			ea->amplitud = ea->float_data[ea->index(i)];
		}
	}

	ea->amplitud = truncar(ea->amplitud);

	//calcular frecuencia
	for( int i=indices[0]; i<indices[1]; i++ ) {
	if( comparar( ea->amplitud, truncar(ea->float_data[ea->index(i)]) ) ){
		if(tiempo1==0){
		tiempo1= ((i-indices[0])*ea->dt)*(float)1000000;
		}
```

```
                else{
                        if(contador>10){
                        tiempo2 = ((i-indices[0])*ea->dt)*(float)1000000;
                        ea->periodo=(float)(tiempo2-tiempo1) / (float)1000000;
                        ea->frecuencia= (float)(1.0 /ea->periodo);

                        tiempo1=tiempo2;
                        contador=0;
                        }
                        else{
                        tiempo1=tiempo2;contador=0;
                        }
                }
        }
        contador+=1;
}
                        //buscar el valor exacto de la amplitud, con valores definidos
                        while(k<100 && band){
                                if(ea->amplitud < valores[k]){
                                        ea->amplitud = valores[k];
                                band=0;
                                }
                        k=k+1;
                        }
```

```
                        if(ea->amplitud >= 0.225 && ea->amplitud<0.3)
                                ea->amplitud = ea->amplitud-0.005;

                        if(ea->amplitud >=  0.3 &&  ea->amplitud<0.370)
                                ea->amplitud = ea->amplitud-0.01;

                        if(ea->amplitud >=  0.37 &&  ea->amplitud<0.44)
                                ea->amplitud = ea->amplitud-0.015;

                        if(ea->amplitud >=  0.44)
                        ea->amplitud = ea->amplitud-0.02;
}

void Escena::setMult_dt(double valor){
        valor=(float)2.5/(float)valor;
        mult_dt=valor;
}


void Escena::setMult_volt(double valor){
        mult_volt=(float)0.50/(float)valor;
}

void Escena::setposY(int valor){
        posY=(float)0.03*((float)valor/10.0);
```

```
}

void Escena::setposX(int valor){
        posX=(float)0.01*((float)valor/10.0);

}

void  Escena::setProjection()
{
        glLoadIdentity();
        glOrtho(lados[3], lados[2], lados[1], lados[0], -10, 10);
}

//Se dibuja la forma de onda en la grafica
void  Escena::drawPlot()
{
        float dt = ea->dt;

        tiempos[0] = -lados[3];
        tiempos[1] = -lados[2];
        tiempos[2] = -trigger[1];
        ea->getTimeSpan(tiempos, indices, 3);
```

```
	if( t_dir != 0 )
	{
		int dx = 0;
		int tw = (int)(trigger_width / dt);

		if( t_dir < 0 )
		{
			while( dx < 5000 && ! (ea->float_data[ea->index(indices[2]+dx)] > trigger[0] && ea->float_data[ea->index(indices[2]+dx+tw)] < trigger[0] )  )
			{
				dx++;
			}
		}
		else
		{
			while( dx < 5000 && ! (ea->float_data[ea->index(indices[2]+dx)] < trigger[0] && ea->float_data[ea->index(indices[2]+dx+tw)] > trigger[0] )  )
			{
				dx++;
			}
		}

		if( dx == 1000 )
			dx = 0;
```

```
                indices[0] = indices[0] + dx;

                indices[0] =indices[0] - 1;

                tiempos[0] += dt;

                indices[1] += dx;

                indices[1]= indices[1] + 1;


        }



        int start = indices[0];


        glLineWidth(grosor); //Grosor de la onda

        glBlendFunc (GL_SRC_ALPHA, GL_DST_ALPHA);

        glEnable (GL_BLEND);

        glEnable (GL_ALPHA_TEST);

        glAlphaFunc (GL_ALWAYS, 0.5);

        glDisable(GL_DEPTH_TEST);

        glEnable(GL_ALPHA);

        glDisable(GL_LIGHTING);

        glColor4f(0, 1.0, 0, 1); //Color de la onda

        glBegin(GL_LINE_STRIP);


        // perillas por defecto en volumen de captura  del dispositivo en 58%, las divisiones de voltage son de 0,1v = 1 mv

        //las perillas por defecto en volumen de captura del dispositivo en 58%, las disiones de tiempo son de 0,0025seg
```

```
float x =-0.1+posX;


for( int i=start; i<=start+(4411/mult_dt); i++ )
{
glVertex2f( x , posY+0.0065+(ea->float_data[ea->index(i)]/10.75)*mult_volt );
x += dt*mult_dt;
}


glEnd();


glLineWidth(1.0);//Grosor de los ejes X y Y


glColor4f(0, 1, 0, 0.3); //Color de los ejes
glBegin(GL_LINES);
        glVertex2f(lados[3], trigger[0]);
        glVertex2f(lados[2], trigger[0]);
        glVertex2f(trigger[1], lados[0]);
        glVertex2f(trigger[1], lados[1]);
        glVertex2f(trigger[1]+trigger_width, lados[0]);
glEnd();


glColor4f(0.5, 0.5, 1, 0.5);


glBegin(GL_LINES);
```

```
                glVertex2f(0, lados[0]);
                glVertex2f(0, lados[1]);
        glEnd();
}


void Escena::dibujar_rejillas(){
        glLineWidth(1.0);
        glColor4f(1, 1, 1, 0.1);
        glBegin(GL_LINES);
        //Dibuja las lines verticales de la grilla
        for( int i=-9; i<0; i++ ){
        glVertex2f(i*0.01, lados[0] < 1 ? lados[0] : 1);
        glVertex2f(i*0.01, lados[1] > -1 ? lados[1] : -1);
        }

        //Se dibujan las lineas horizontales de la grilla
        for( int i=-2; i<=2; i++ ){
        glVertex2f(lados[2] > 0 ? 0 : lados[2], i*0.10);
        glVertex2f(lados[3], i*0.10);
        }
        glEnd();
}
```

```
void  Escena::setGrosor(int g)
{
	grosor = g;
}

float Escena::getAmplitud()
{
	return ea->getAmplitud();
}

float Escena::getFrecuencia()
{
	return ea->getFrecuencia();
}

//Glwidget
void Escena::animate()
{
   calcular_frecuencia();
   repaint();
}

void Escena::detener()
{
	ea->pause = ! ea->pause;
```

```
}

void Escena::paintGL()
{
	//Las imagenes en pantalla estan organizadas en una especie de matriz
	//Existe una pila de matrices para cada uno de los modos de matriz.
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //Limpia el buffer de imagenes en pantalla
	glClearColor( 0, 0, 0, 0.0 ); //Color fondo del osciloscopio

	glMatrixMode(GL_PROJECTION); //Orden en la matriz de las imagenes en pantalla
	glLoadIdentity(); //inicializar la matriz
	setProjection(); //Es un procedimiento que debe manipular la proyeccion de la matriz
	glMatrixMode(GL_MODELVIEW);//Orden en la matriz de las imagenes en pantalla
	//glPushMatrix();//realiza una copia de la matriz superior y
	//la pone encima de la pila, de tal forma que las dos matrices superiores son iguales
	glLoadIdentity();//inicializar la nueva matriz
	dibujar_rejillas();
	drawPlot();
	glPopMatrix();//elimina la matriz superior, quedando en la parte
	//superior de la pila la matriz que estaba en el momento de
	//llamar a la función glPushMatrix().
	glFinish();

 }
```

Anexo 6: EntradaAudio.h

```
 #ifndef ENTRADAAUDIO_H
#define ENTRADAAUDIO_H

#include "utilidades.h"
#include <alsa/asoundlib.h>
#include <GL/glut.h>
#include <GL/gl.h>

union byte
{
	unsigned char uchar_val;
	char char_val;
};

class EntradaAudio
{
	public:
		char* data;// array de datos tipo caracter
		float* float_data; //array de datos tipo real
		int buffer_blocks;// cantidad de bloques del "buffer"
		int data_start;//punto de inicio de la data
		int data_size;//Tamaño del array de la datos
		int data_end;//Punto final del array de datos
		int data_write;//Escribe en la data, debe ser como un valor booleano
		int write_padding;
		int front_padding;

		//Amplitud y Frecuencia para mostrarlas por pantalla
		float amplitud;
		float periodo;
		float frecuencia;

		bool pause;
		pthread_t capture_thread;
		timeval last_write;
		float avg_write_interval;

		// Datos de audio del dispositivo
		int bytes_per_frame;//cantidad bytes por frame
		int read_frames, read_bytes;//cantidad de frames leidos, cantidad de bytes
leidos
		int rate; /* Frecuencia de muestreo */
		float dt;
```

```
        int exact_rate;  /* Frecuencia de muestreo devuelto por */
                                        /* snd_pcm_hw_params_set_rate_near */
        int dir;            /* exact_rate == rate --> dir =  0 */
                                        /* exact_rate < rate  --> dir = -1 */
                                        /* exact_rate > rate  --> dir =  1 */

        int periods;       // numero de periodos
        snd_pcm_uframes_t periodsize; // Tamanio del periodo (bytes)
        int size, exact_size;  // tamaño del buffer

        snd_pcm_t *pcm_handle;  // Controlador para el  dispositivo PCM
        snd_pcm_stream_t stream;// Reproduccion del flujo

        /* Esta estructura contiene informacion acerca del
        hardware y de como puede ser usado para especificar la
        configuracion que se utilizara para el flujo PCM . */
        snd_pcm_hw_params_t *hwparams;

        /* Nombre del dispositivo PCM, ejemplo plughw:0,0
        El primer número es el numero de la  tarjeta de sonido
        el segundo número es el numero del dispositivo  */
        char *pcm_name;

    public:
        EntradaAudio();
        int index( int i );
        static void* audioCapture(void* a);
        void getTimeSpan(float* time, int* index, int num);
        int initDevice();

        float getAmplitud();
        float getFrecuencia();
};
#endif
```

Anexo 7: EntradaAudio.cpp

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <alsa/asoundlib.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>
#include <sys/time.h>
#include "EntradaAudio.h"

//Constructor de la clase EntradaAudio(entrada de audio)
EntradaAudio::EntradaAudio()
{
	//En esta parte del codigo se inicializan los valores iniciales
	//de la clase EntraAudio
	pause = false;// no esta detenido
	buffer_blocks = 50;//cantidad de blockes de buffer 50
	bytes_per_frame = 2;//bytes por muestra  2, 16 bits
	read_frames = 1000;//Tamaño del vector de buffer
	write_padding = read_frames * 4;
	front_padding = read_frames;
	avg_write_interval = 0.1;

	read_bytes = read_frames * bytes_per_frame;
	data = new char[read_bytes];

	//Amplitud y Frecuencia de la onda
	amplitud=0;
	frecuencia=0;

	data_start = read_frames * 2;
	data_end = data_size-1;
	data_write = 0;
	data_size = buffer_blocks * read_frames;
	float_data = new float[data_size];

	rate = 44100; //Tasa de flujo
	dt = 1.0 / (float)rate; // delta t, tiempo existente entre cada muestra
	periods = 2;
	periodsize = 8192;
	size = exact_size = (periodsize * periods)>>2;
```

```
	if( initDevice() < 0 )
	{
		exit(1);
	}

	pthread_create(&capture_thread, NULL, audioCapture, (void*)this); //Comienza a capturar el hilo
}

int EntradaAudio::index( int i )
{
	while( i < 0 )
		i += data_size;

	return i % data_size;
}

//Aqui es donde se captura el sonido
void* EntradaAudio::audioCapture(void* a)
{
	EntradaAudio* ai = (EntradaAudio*) a;

	float inv256 = 1.0 / 256.0;
	float inv256_2 = inv256*inv256;

	while( true )
	{

		int n;
		if( ! ai->pause )
		{
			while((n = snd_pcm_readi(ai->pcm_handle, ai->data, ai->read_frames)) < 0 )
			{
				snd_pcm_prepare(ai->pcm_handle);

			}

			byte b;
			int write_ptr, read_ptr;

			//Se Transforma los valores capturados a una escala mas pequeña
			//para poder dibujar la forma de onda y visualizarla de manera
			//correcta
```

```
                        for( int i = 0; i < n; i++ )
                        {
                                read_ptr = i * 2;
                                write_ptr = ai->index(ai->data_write + i);
                                b.char_val = ai->data[read_ptr];
                                ai->float_data[write_ptr] = (float)ai-
>data[read_ptr+1]*inv256 + (float)b.uchar_val*inv256_2;
                        }

                        ai->data_end   = ai->data_write;
                        ai->data_write = ai->index(ai->data_write+n);
                        ai->data_start = ai->data_end - (ai->data_size - ai->write_padding);
                        timeval t;
                        gettimeofday(&t, NULL);
                        ai->avg_write_interval = ai->avg_write_interval * 0.7 + timeDiff(ai-
>last_write, t) * 0.3;
                        ai->last_write = t;
                }
                else
                {
                        usleep(10000);
                }
        }

        fprintf(stderr, "Captura de hilo de salida.\n");
}
//Fin de captura de sonido

void EntradaAudio::getTimeSpan(float* time, int* index, int num)
{
        float sh;

        if( pause )
        {
                sh = 0;
        }
        else
        {
                timeval t;
                gettimeofday(&t, NULL);
                sh = timeDiff(last_write, t);
        }

        float min_time = 0;
```

```
		float max_time = (data_end - data_start) * dt;

		for( int i=0; i<num; i++ )
		{
			if( time[i] < min_time )
				time[i] = min_time;

			if( time[i] > max_time )
				time[i] = max_time;

			index[i] = data_end - front_padding - (int)((time[i]-sh)*(float)rate);
			time[i] = sh + ((data_end - index[i] - front_padding) * dt);
		}
}
//Inicializacion del dispositivo PCM , verificando que no haya errores
int EntradaAudio::initDevice()
{
		stream = SND_PCM_STREAM_CAPTURE;
		pcm_name = strdup("plughw:0,0");// Iniciando pcm_name.
		snd_pcm_hw_params_alloca(&hwparams);// Asignar la estructura snd_pcm_hw_params_t en la pila

		//Abrir PCM (Codigo de pulso modulado),  El último parámetro de
		//esta función es el modo. Si esto se establece en 0, el modo estándar es utilizado.
		//Otros posibles valores son SND_PCM_NONBLOCK y SND_PCM_ASYNC. Si SND_PCM_NONBLOCK
		//acceden a leer y escribir en el dispositivo PCM.
		//Si SND_PCM_ASYNC se especifica, SIGIO se emitirá cada vez que un
		//período ha sido completamente procesados por la tarjeta de sonido.
		if (snd_pcm_open(&pcm_handle, pcm_name, stream, 0) < 0)
		{
			fprintf(stderr, "Error al abrir el dispositivo PCM %s\n", pcm_name);
			return(-1);
		}

		//Inicializamos hwparams con el espacio de configuración completa
		if (snd_pcm_hw_params_any(pcm_handle, hwparams) < 0) {
			fprintf(stderr, "No puede ser configurado este dispositivo PCM.\n");
			return(-1);
		}

		//Establecer el tipo de acceso. Esto puede ser
		//SND_PCM_ACCESS_RW_INTERLEAVED o SND_PCM_ACCESS_RW_NONINTERLEAVED.
```

```
	if (snd_pcm_hw_params_set_access(pcm_handle, hwparams,
SND_PCM_ACCESS_RW_INTERLEAVED) < 0)
	{
		fprintf(stderr, "Error estableciendo el acceso.\n");
		return(-1);
	}

	// Establecer formato de muestra
	if (snd_pcm_hw_params_set_format(pcm_handle, hwparams,
SND_PCM_FORMAT_S16_LE) < 0)
	{
		fprintf(stderr, "Error estableciendo el formato.\n");
		return(-1);
	}

	// Establecer la frecuencia de muestreo. Si la frecuencia exacta no es soportada
	// por el hardware, se usara la frecuencia mas cercana posible.
	exact_rate = rate;

	if (snd_pcm_hw_params_set_rate_near(pcm_handle, hwparams, (uint*)&exact_rate, 0)
< 0)
	{
		fprintf(stderr, "Error configurando la frecuencia.\n");
		return(-1);
	}

	if (rate != exact_rate)
	{
		fprintf(stderr, "La frecuencia %d Hz no es soportada por tu hadware.\n \
					==> En su lugar usa %d Hz.\n", rate,
exact_rate);
	}

	// Establecer numero de canales
	if (snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 1) < 0)
	{
		fprintf(stderr, "Error estableciendo los canales.\n");
		return(-1);
	}

	//Establecer el número de períodos. Periodos usados son llamados fragmentos.
	if (snd_pcm_hw_params_set_periods(pcm_handle, hwparams, periods, 0) < 0)
	{
		fprintf(stderr, "Error estableciendo los periodos.\n");
```

```
                return(-1);
        }

        // Establecer el tamaño del búfer (en marcos). La latencia resultante está dada por
        // latency = periodsize * periods / (rate * bytes_per_frame)
        if          (snd_pcm_hw_params_set_buffer_size_near(pcm_handle,          hwparams,
(snd_pcm_uframes_t*)&exact_size) < 0)
        {
                fprintf(stderr, "Error estableciendo el tamanio del buffer.\n");
                return(-1);
        }

        if( size != exact_size )
        {
                fprintf(stderr, "El tamanio del Buffer %d no es soportado, En su lugar usar
%d.\n", size, exact_size);
        }

        //Ajusta parametros de HW al dispositivo PCM y prepara el dispositivo
        if (snd_pcm_hw_params(pcm_handle, hwparams) < 0)
        {
                fprintf(stderr, "Error estableciendo los parametros de HW.\n");
                return(-1);
        }

        return 1;
}
//Consultamos la amplitud actual
float EntradaAudio::getAmplitud()
{
        return amplitud;
}
//Consultamos frecuencia actual
float EntradaAudio::getFrecuencia()
{
        return frecuencia;
}
```

Anexo 8: utilidades.h

```
#ifndef UTILIDADES_H_
#define UTILIDADES_H_
	typedef float Vector [4];
	int ValAbs(float val);
	int rot(int a, int mod);
	float snap(float val, float grid);
	float afloor_snap(float val, float grid);
	float min(float a, float b);
	float max(float a, float b);
	void doVertex(Vector v);
	float timeDiff(timeval a, timeval b);
	float truncar(float num);
	float comparar(float x1, float x2);
#endif
```

Anexo 9: utilidades.cpp

```
#ifndef UTILIDADES_H_
#define UTILIDADES_H_

#include <GL/glut.h>
#include <GL/gl.h>
#include <alsa/asoundlib.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>
#include <sys/time.h>


typedef float Vector [4];

//Valor Absoluto
int ValAbs(float val)
{
	return val<0 ? -1 : 1;
}

//Devuelve el residuo de a+1/mod.
int rot(int a, int mod)
{
	return (a+1) % mod;
}

//Tiene que ver con los graficos, con el plot.
float snap(float val, float grid)
{
	float chop_val = (int)(val * 1000.0) / 1000.0; // Para evitar pequeños errores de punto flotante
	float snapped = (int)((chop_val/grid)+ValAbs(chop_val) * 0.5) * grid;
	return(snapped);
}

float afloor_snap(float val, float grid)
{
```

```
	float chop_val = (int)(val * 1000.0) / 1000.0;  // Para evitar pequeños errores de punto flotante
	float snapped = (int)((chop_val/grid)) * grid;
	return(snapped);
}

//retorna el menor
float min(float a, float b)
{
	return a>b ? b : a;
}

//retorna el mayor
float max(float a, float b)
{
	return a>b ? a : b;
}

//Le asigna valores a Vertex
void doVertex(Vector v)
{
	glVertex3f(v[0], v[1], v[2]);
}

//tiene que ver con la diferencia de tiempo
float timeDiff(timeval a, timeval b)
{
	return (float)(b.tv_sec - a.tv_sec) + ((float)(b.tv_usec - a.tv_usec) * .000001);
}

//Truncar valores reales a 4 cifras
float truncar(float num){
	num = num*10000;
	int aux_trun = num;
	return ((float)aux_trun)/(float)10000;
}

//Si dos valores son iguales retornar verdad, de lo contrario, retorna falso
float comparar(float x1, float x2){
```

```
        int y1=x1*(float)1000;
        int y2=x2*(float)1000;

        if(y1 == y2)
        return true;
        else
        return false;
}
#endif
```