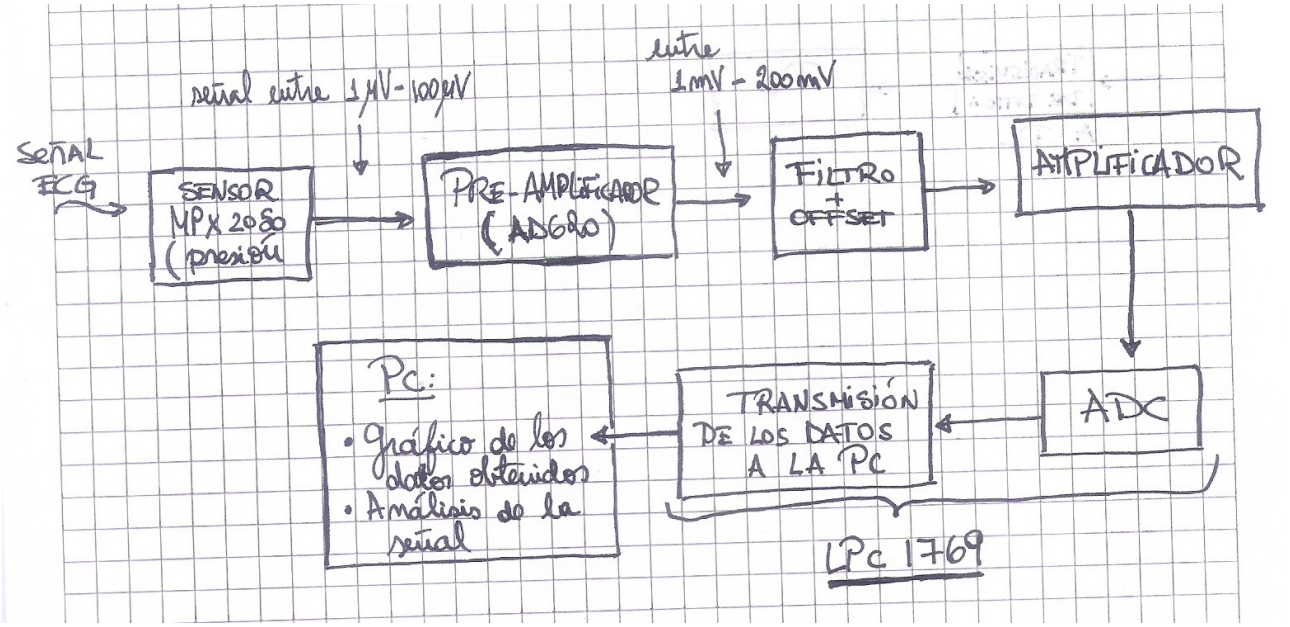
Trabajo Práctico Obligatorio INFO II :

Medidor de ECG

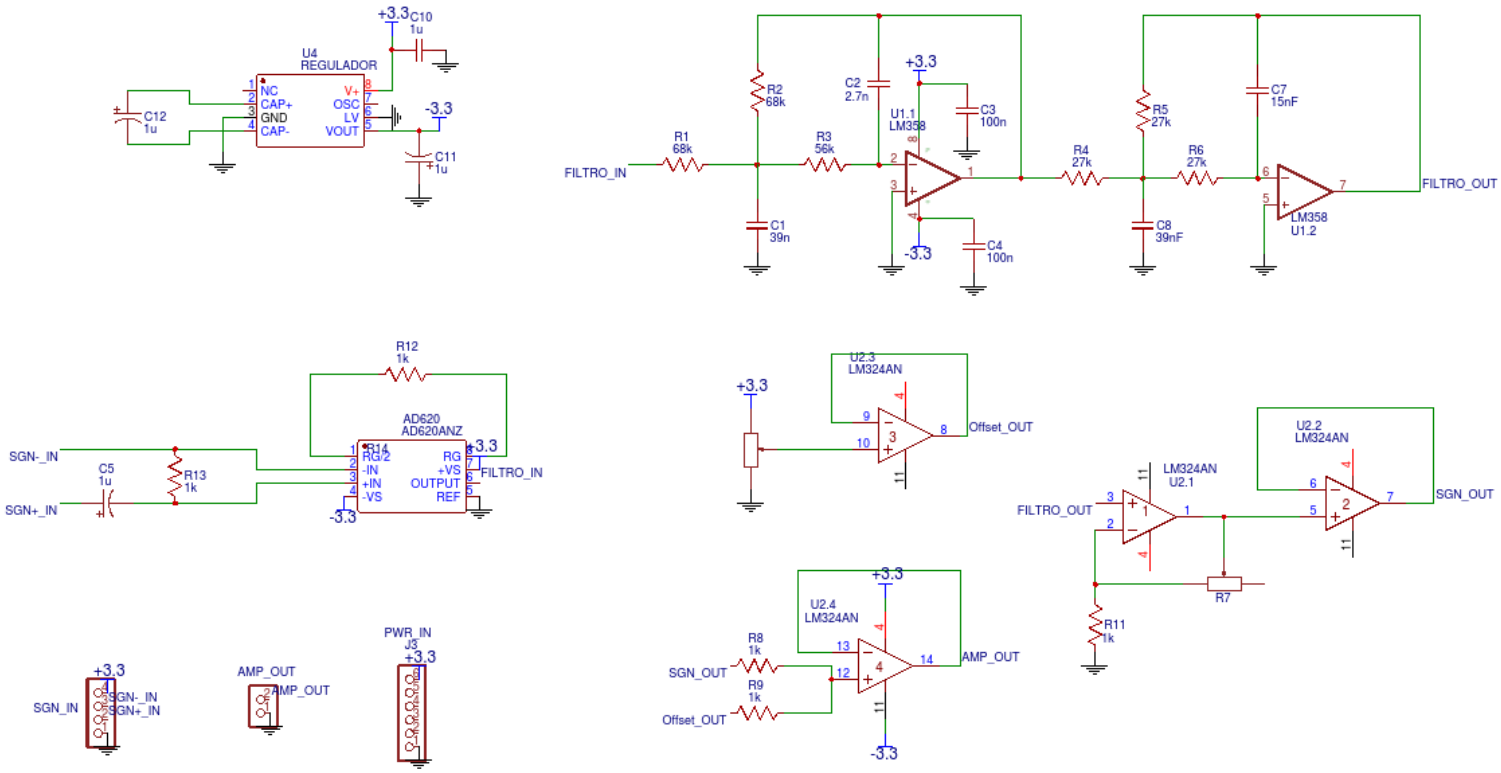
Introducción:

El proyecto consiste básicamente, en un sistema el cual le permite al usuario poder observar de manera gráfica la señales QRS originadas por el corazón mediante la implementación de un sensor de presión.

En qué consiste:



Como se puede observar en el esquema de arriba, el proyecto se divide en diferentes etapas.
Luego en la imagen de abajo, el esquemático completo del circuito.



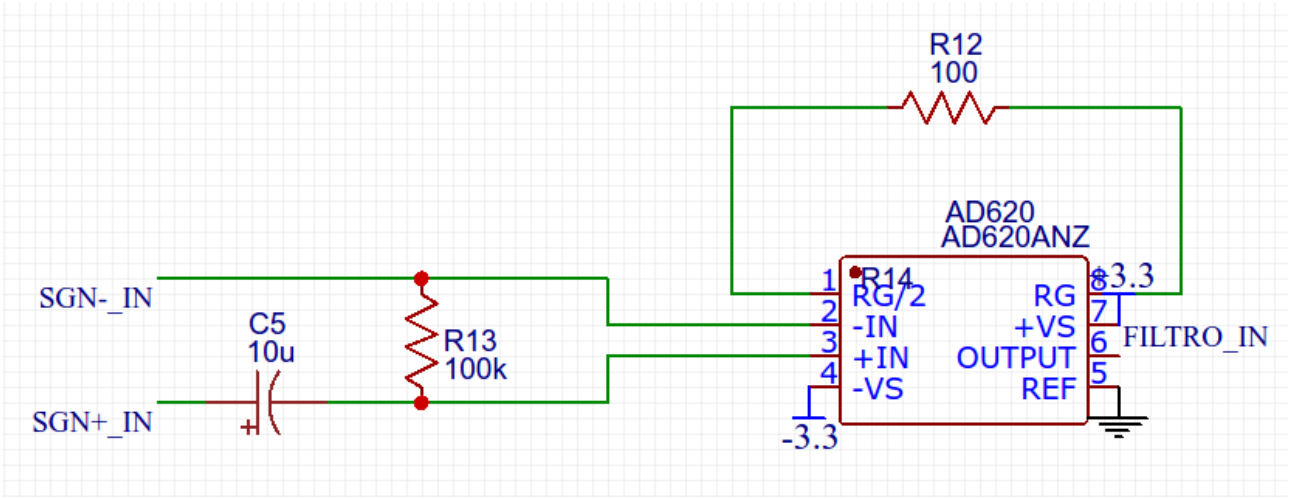
Adquisición de la señal QRS:

La primera etapa consiste en la adquisición de la señal mediante el sensor de presión MPX2050 (datasheet : http://www.nxp.com/docs/en/data-sheet/MPX2050.pdf). Este se encuentra dentro de un encapsulado especialmente diseñado para asegurarse de estar midiendo exactamente sobre la zona deseada.
Se ha descubierto, mediante sucesivas pruebas sobre distintas partes del cuerpo, que el mejor lugar sobre el cual medir es sobre la carótida de la persona.

Un factor importante a la hora de medir es la presión que se ejerza con el sensor sobre el lugar a medir. Lógicamente, mientras más presión se aplique, mayor será la amplitud de la señal que se obtenga. Sin embargo, todavía queda por probar si esto afecta o no al comportamiento normal de la arteria.
Para un futuro próximo se diseñará el sensor de tal manera de poder aplicar una presión estándar a la hora de medir, y así no depender del usuario que está midiendo.

Pre-Amplificador (AD620):

Como se explicó previamente, el sensor se encarga de transformar las variaciones de presión ocasionadas por la arteria en un valor de tensión proporcional a esta. Estos valores son muy chicos (del orden de los 120 uV como valor máximo) por lo que una etapa de amplificación es obligatoria.



Esta es llevada a cabo por el amplificador de instrumentación **AD620** (datasheet : http://www.analog.com/media/en/technical-documentation/data-sheets/AD620.pdf).
Si uno observa su hoja de datos, observará que el rendimiento de este disminuye a medida que la ganancia aumenta. Sin embargo, los mejores resultados se observaron para ganancias de 500 veces (Rg = 100Ω) hasta ganancias de 989 veces (Rg = 50Ω).

No obstante, uno de los defectos de esta configuración es el de la amplificación de la señales de modo común de la entrada. Esto es, si la salida del sensor de presión además de generar la señal deseada también está compuesta por tiene un nivel de continua u offset (por más chico que este sea) este también será amplificado, afectando el funcionamiento normal del sistema.
Por ejemplo, suponiendo que la señal de salida del sensor tiene valores máximos de 150uV en modo diferencial y 1mV en modo común – ó 1mV de offset – la salida, si se amplifican casi 1000 veces, será de 150 mV para la señal MÁS un nivel de tensión de offset de 1V completamente indeseado.

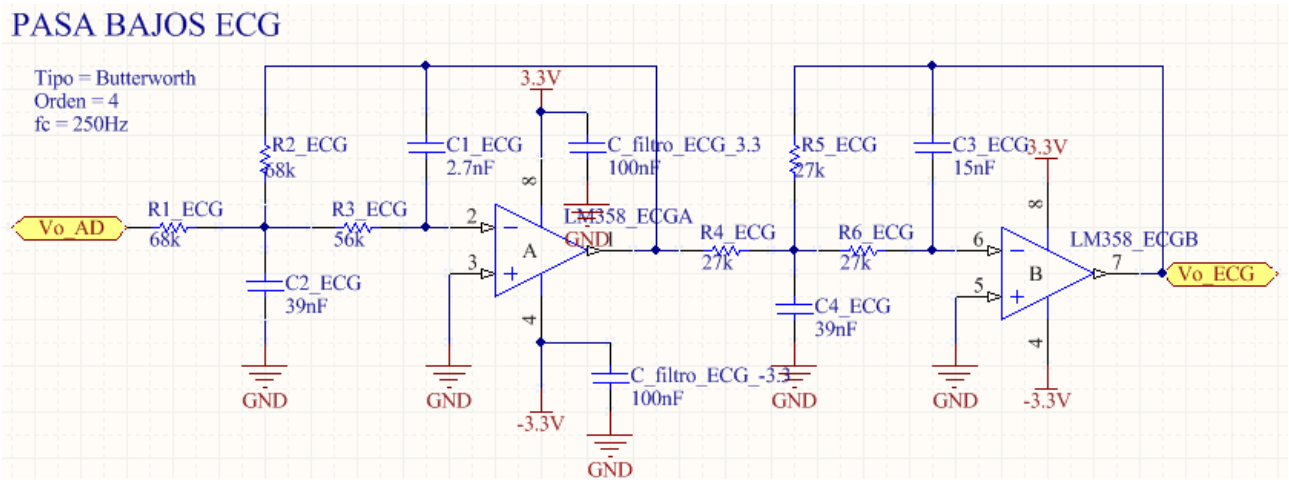
Para solucionar esto, se agregó un filtro pasa-altos en la entrada del amplificador de instrumentación con una frecuencia de corte de 1 Hz aproximadamente.
Este cambio permite hacer que uno olvide cualquier tipo de offset indeseado en la salida del amplificador debido a la señal de entrada.
La única desventaja que puede llegar a ocasionar, si es que así se la puede considerar, es que el tiempo de carga del capacitor debido al par capacitor-resistencia es de aproximadamente 5 segundos. Esto significa, que una vez conectado el circuito hay que esperar ese tiempo para que funcione correctamente.

Cabe aclarar que también se ha probado el amplificador de instrumentación INA114, pero los resultados obtenidos por éste no fueron tan satisfactorios como con el de AD.

Filtro :

Se han leido publicaciones y recomendaciones de distintas personas y fabricantes especializados en el tema, y todos llegan a la conclusión de que las frecuencias que superan las 250 Hz aproximadamente, sólo aportan ruido a la señal. Por ende, se aplicará un filtro pasa-bajos butterworth de 4to orden, con una frecuencia de corte de 250 Hz para asegurar que la señal obtenida sea lo más limpia y pura posible.

El circuito del filtro es el siguiente:



Dicho circuito consiste de dos filtros pasa-bajos activos e inversores Butterworth de 2do orden c/u. Al ponerlos en cascada, se invierte lo previamente invertido – por ende me quedo con la señal original – y los ordenes del filtro se multiplican.

Para llevar a cabo esta etapa, se utilizó el integrado LM358, que al disponer de dos amplificadores operaciones fue ideal para llevar a cabo y probar el circuito, logrando además muy buenos resultados finales con este.

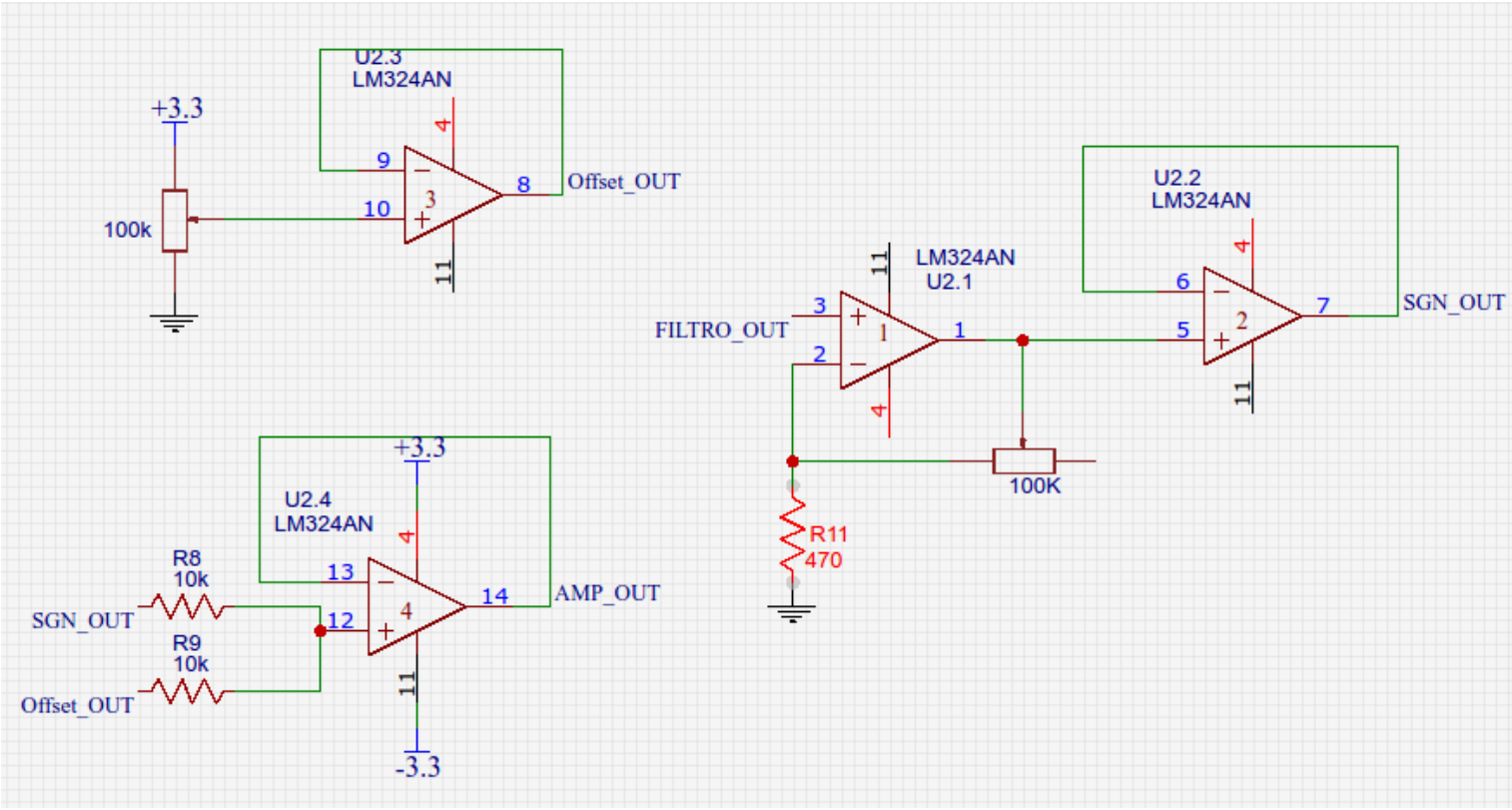
Es importante aclarar que la aplicación de un buen filtro analógico es fundamente para obtener una señal de salida lo más limpia posible. Lógicamente, esta también se podría haber retocado digitalmente (que de hecho, posteriormente también se hace) pero los mejores resultados se obtuvieron al probar circuitos de esta índole.

Amplificador :

Como se pudo ver en la etapa del pre-amplificador, el primer problema que se observa es que la señal sigue teniendo una amplitud bastante chica (llegando a valores máximos de 113 – 150 mVpp).
Otra dificultad que quizás pasa por desapercibida, es que la señal en ciertas ocasiones puede tomar valores negativos

Por ende, en esta etapa se tendrá que agregar un valor de offset a la señal original, de tal manera de asegurar que la señal tenga siempre valores positivos de tensión, y otra etapa amplificadora que me asegure que dichos valores se encuentren dentro de un rango de 0 a 3,3V.

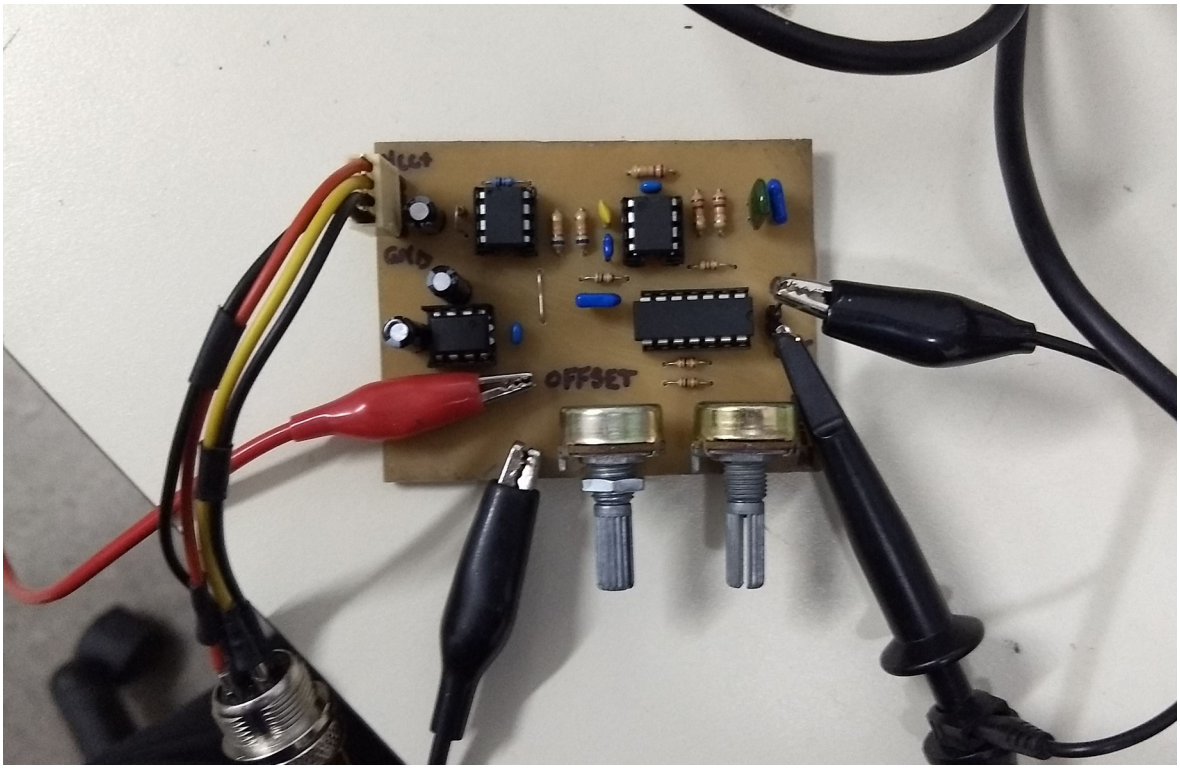
Se empleará entonces el siguiente circuito:



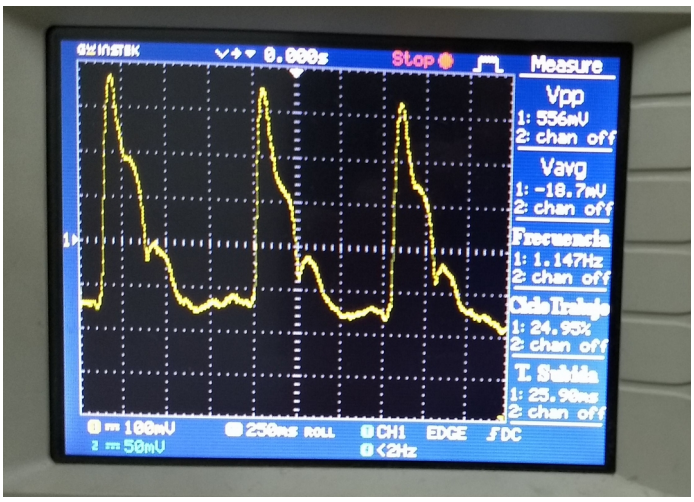
Este consiste de, por un lado, un amplificador no inversor (U2.1) cuya entrada es la salida del filtro y el cuál cumple la función de amplificar la señal ECG (cuya ganancia se puede modificar mediante un potenciómetro de 100k) , y de otro potenciómetro conectado entre +Vcc y 0V, cuya función es la de determinar el valor de continua (tensión de offset) de la señal.
Para evitar problemas de desacloples de impedancia, se conectaron estos dos últimos a dos operacionales en modo seguidor, y cuyas salidas a su vez se conectan a un circuito sumador, el cual se encarga de mostrar como salida final de la placa la señal ECG amplificada + su nivel de offset correspondiente.

Lo bueno de esta simple configuración radica en es su gran rendimiento final, cumpliendo todo como lo debería a la perfección, y en que se pudo englobar todo lo necesario con un sólo LM354 y un par de componentes básicos, lo cual facilita muchísimo el posterior diseño del pcb.

Finalmente, el circuito implementado en una PCB quedaría :



Donde su salida vista por un osciloscopio digital es:



LPC1769:

Antes de comenzar con la explicación, el programa del LPC se encuentra adjunto en la misma carpeta de este informe, con el nombre “Medidor_ECG_LPC”.

Las funciones que lleva a cabo el microcontrolador en este proyecto son varias, como por ejemplo:

- **La digitalización de la señal de salida del circuito ECG mediante muestreos sucesivos de esta, para además ser luego filtrada digitalmente.**

Para ello, se utilizó las funciones:

```
//Con esta funcion inicializo al conversor analogico digital ADC0.5, y configuro
su clock interno
//de tal manera que tome muestrar a una frecuencia de 98kHz

void ADC_Init(void)
{
	//1.- Activo la alimentación del dispositivo desde el registro PCONP
	PCONP |= 1<<12;
	//2.- Selecciono el clock del ADC como 25MHz, PCLK_ADC = 100 Mhz / 4 =
25MHz
	PCLKSEL0 &= ~(0x03<<24);
	//3.- Y el divisor como 255 -> clk_ADC = 25MHz/256 = 98kHz
	// NOTA: PCLK_ADC0 is divided by (this value plus one) to produce the
clock for the A/D converter)
	AD0CR = 0x3F00;
	//4.- Configuro los pines del ADC0
	//ADC0.5 (pote) : P1[31]->PINSEL3: 30:31
	SetPINSEL(P1, 31, PINSEL_FUNC3);
	//5.- Activo interrupción del canal 5
	AD0INTEN = 0x00000020;
	//6.- Selecciono que voy a tomar muestras del canal AD0.5
	AD0CR |= 0x00000020;
	//7.- Activo el ADC (PDN = 1)
	AD0CR |= 1<<21;
	//8.- Como vamos a trabajar en modo BURTS, pongo los bits de START = 000
	AD0CR &= ~(0x0F<<24);
	// NOTA: El BURTS no lo activo ahora, lo hago desde la aplicación.
	//AD0CR |= 1<<16;
	//9.- Habilito interrupción en el NVIC
	ISER0 |= (1<<22);
}



//Dicho muestreo se llevaron a cabo cuando el usuario invoque esta funcion
//para empezar a tomar muestras

void ADC_BurstEnable(void)
{
	//Activo el Burst, empieza a convertir.
	AD0CR |= 1<<16;
}
```

```
//Y esta funcion para que pare de muestrear

void ADC_BurstDisable(void)
{
	//Deshabilito el Burst, detiene la conversión.
	AD0CR &= ~(0x01 << 16);
}


//Siempre que el ADC haya terminado se llama a esta interrupcion, donde antes de ser transmitida el valor
//obtenido, este es pasado por un filtro digital.

void ADC_IRQHandler(void)
{
	uint32_t adc_data_reg = 0;
	static uint32_t   circularBuffer[WINDOW_SIZE]  = {0};   //circular buffer
	static uint32_t         *circularBufferAccessor       = circularBuffer;
//actual value
	static uint32_t         sum                                 = 0;
	//const  uint32_t       *circularBufferStart          =
&circularBuffer[0]; // constant pointer -> start buff
	static const  uint32_t  *circularBufferEnd                  =
&circularBuffer[WINDOW_SIZE]; // constant pointer -> end buff

	// leo el registro de resultado del canal 5
	adc_data_reg = AD0DR5;
	// chequeo bit DONE == 1 -> hay un resultado disponible
	if ((adc_data_reg >> 31) == 0x01)
	{

		//ARRANCA LA APLICACION DEL FILTRO
		sum -= *circularBufferAccessor;
		// obtengo los 12 bits del valor medido y los acumulo para promediar
		sum += (adc_data_reg >> 4) & 0xFFF;

		*circularBufferAccessor = (adc_data_reg >> 4) & 0xFFF;
		circularBufferAccessor++;

		if (circularBufferAccessor == circularBufferEnd )
			circularBufferAccessor = circularBuffer;

		if (ADC_delay == 0)
		{

		ADC_delay = 1;
		ADC_valor = ( sum / WINDOW_SIZE ) ;
		ADC_dato_disponible = 1;

		}
	}
}
```

El filtro digital consiste en el denominado “Filtro de Media Movil”, cuya ecuación de transferencia es:

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

Siendo M, en términos prácticos el coeficiente de suavizado de la señal

Además, al utilizar un buffer circular manejado únicamente por punteros para almacenar los datos, el filtro trabaja prácticamente a la frecuencia de clock, siendo esta, entre otras, la ventaja principal de porqué no se implementó un filtro digital más sencillo (como puede ser un típico promediador).

La señal filtrada se almacena en la variable global ADC_valor, la cual al activarse el flag “ `ADC_dato_disponible`”, es transmitida vía UART

- **Controlar un LCD para mostrar los datos transmitidos vía UART para una rápida verificación de que lo que se muestra en la PC se corresponde con lo deseado.**
- **Y además, una interfaz rápida para el usuario que le permite comenzar o poner en pausa el muestreo de la señal ECG.**

Para ello, se utilizó la función:

```
//Esta funcion es invocada en el main() y se encarga de ir controlando todos los procesos
//previamente explicados mediante los distintos estados de su maquina.

void Controlador(void)
{
	static uint32_t estado_medidor = MEDIDOR_RESET;
	static char msg_LCD[2*LARGO_RENGLON];

	switch (estado_medidor) {
		case MEDIDOR_RESET:
			LCD_DisplayMsg(" Medidor de ECG ", LCD_RENGLON_1, 0);
			LCD_DisplayMsg(" Prueba Sampling", LCD_RENGLON_2, 0);
			estado_medidor = MEDIDOR_INACTIVO;
			break;

			// Si recibo la palabra clave por UART, la maquina de estado
			// se queda aca hasta que el usuario arranca con la medicion
		case MEDIDOR_INACTIVO:
			if (flag_IniciarMedicion == 1)
			{
			flag_IniciarMedicion = 0;
			LCD_DisplayMsg("                 ", LCD_RENGLON_1, 0);
			LCD_DisplayMsg("                 ", LCD_RENGLON_2, 0);

			LCD_DisplayMsg(" KEY_2 -> PAUSE  ", LCD_RENGLON_2, 0);
			LCD_DisplayMsg(" KEY_1 -> START  ", LCD_RENGLON_1, 0);
			flag_IniciarMedicion = 0;

			estado_medidor = WAITING_KEY;

			}
			break;

			// chequeo si el usuario apreto la tecla (en este caso el pulsador 1 del kit infotronic)
			// el cual inicia las mediciones
		case WAITING_KEY:
				if (bufferTeclado5x1 == KEY_1)
				{
				ADC_BurstEnable();
				LCD_DisplayMsg(" > MIDIENDO      ", LCD_RENGLON_1, 0);
				IO_LED_Set(IO_LED_STICK, ON);
```

```
                        estado_medidor = MEDIDOR_ACTIVO;
                        }
                  break;

            case MEDIDOR_ACTIVO:
                  // si hay un dato disponible, lo envío por el puerto serie
                  if (ADC_dato_disponible) {
                        ADC_dato_disponible = 0;
                        // transmito la trama con un nuevo dato
// Para la transmisión utilizo las funciones PushTx deducidas en clase, y dado
que el ADC es de 12 bits, transmito primero los 4 bits mas significativos
( llenando los espacios con cero ) y luego los 8 bits menos significativos
restantes. Todo esto posterior del byte '$' que le indica a la PC el comienzo de
una trama nueva.
                        PushTx('$');
                        PushTx((char)((ADC_valor>>8) & 0xFF));
                        PushTx((char)(ADC_valor & 0xFF));
                  }
                  // chequeo si llegó un comando para detener la medición
                  else if (flag_DetenerMedicion == 1 || bufferTeclado5x1 ==
KEY_2) {
                        flag_DetenerMedicion = 0;
                        ADC_BurstDisable();
                        IO_LED_Set(IO_LED_STICK, OFF); // lo unico que hace aca
                                                      es prender y apagar el led
                                                            del stick para
                                                      // dejar un test de que se
                                                      llevo a cabo una medicion
                        flag_IniciarMedicion = 1;
                        estado_medidor = MEDIDOR_INACTIVO;
                  }
                  // actualizo el valor mostrado en el LCD
                  // esto me sirve para poder verificar luego que lo que se
muestra en la
                  // PC se corresponde con lo que se envió desde el kit.
                  else if (flag_updateValorLCD) {
                        flag_updateValorLCD  = 0;
                        sprintf(msg_LCD, "valorADC=%04d   ", ADC_valor);
                        LCD_DisplayMsg(msg_LCD, LCD_RENGLON_2, 0);
                  }
                  break;

            default:
                  estado_medidor = MEDIDOR_INACTIVO;
                  break;
      }
}
```

- **Activación de sistema siempre y cuando se reciba la trama especial “ $M1# “, la cual asegura que la comunicación con la computadora se realizó de manera correcta (dado que pudo recibir la trama), y configura a la maquina de estado de la funcion Controlador() para que las muestras sean activadas ahora por el usuario desde la placa.**

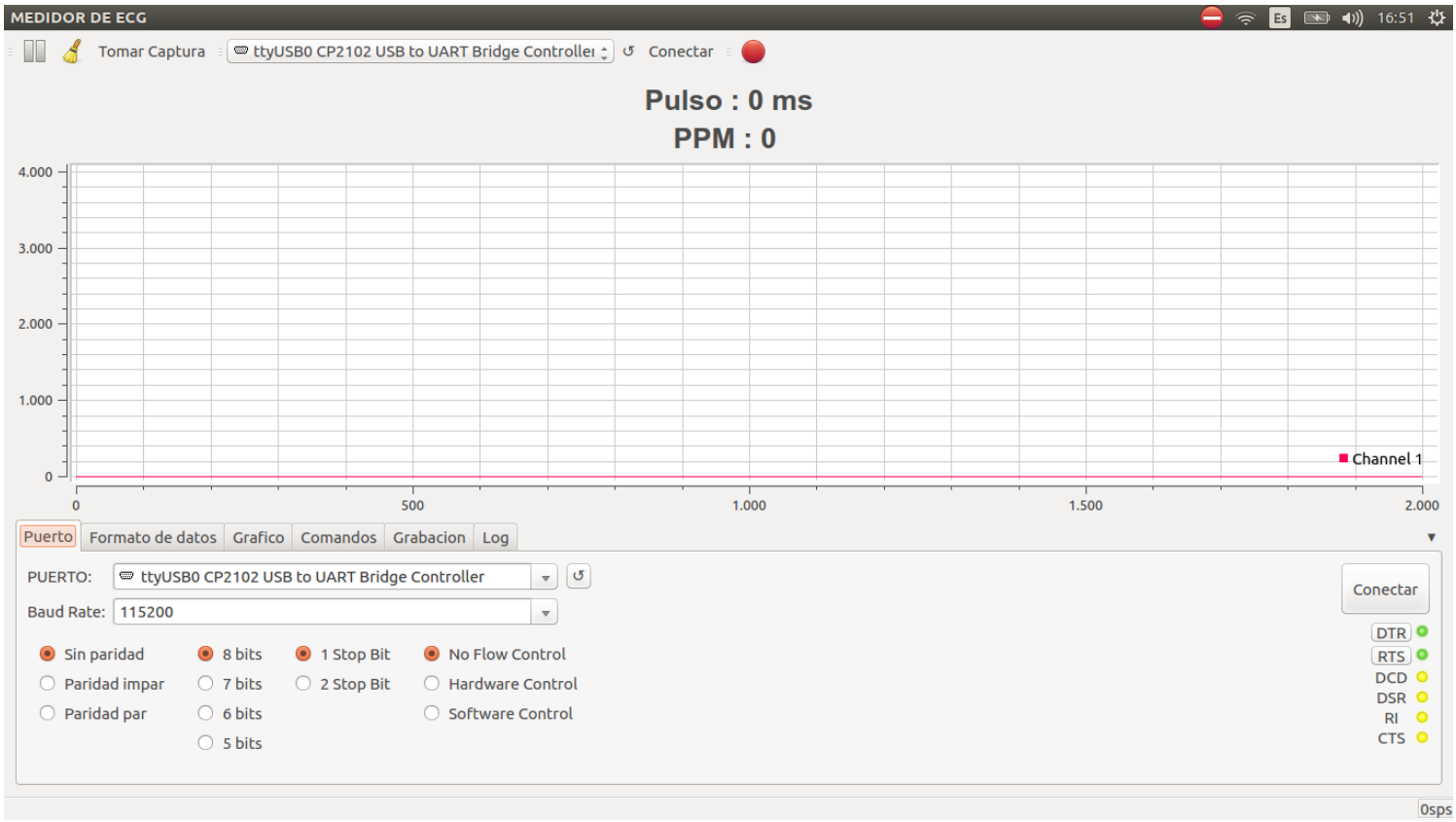
Para ello se implemento la siguiente máquina de estado:

```
//Funcion llamada por main() que espera la trama de activacion del sistema

void RX_Mensajes(void)
{
	uint8_t dato;
	static uint8_t index_msg_rx = 0;
	static char msg_rx[MAX_TRAMA_RX] = {0};
	static uint32_t estado_rx = ESPERANDO_TRAMA;

	// Chequeo si llegó un msje...
	if (!PopRx(&dato)) {
		// MdE: Análisis de la trama recibida
		switch (estado_rx) {
			case ESPERANDO_TRAMA:   // Espero el caracter de inicio de la
trama ('$')
				if ((char)dato == '$') {
					index_msg_rx = 0;
					estado_rx = RECIBIENDO_TRAMA;
				}
				break;
			case RECIBIENDO_TRAMA:  // Recibo y almaceno la trama completa
				// Chequeo si llegó el final de la trama ('#')
				if ((char)dato != '#') {
					msg_rx[index_msg_rx] = (char)dato;
					index_msg_rx++;
					if (index_msg_rx > MAX_TRAMA_RX)
						estado_rx = ESPERANDO_TRAMA;
				} else {
					// msg_rx[0]: 'M' (comando medición ADC)
					// msg_rx[1]: '0' detener, '1' iniciar
					if ((msg_rx[0]) == 'M' || (msg_rx[0]) == 'm') {
						if((msg_rx[1]) == '0') {
							flag_DetenerMedicion = 1;
						}
						else if ((msg_rx[1]) == '1')
								flag_IniciarMedicion = 1;

					}
					estado_rx = ESPERANDO_TRAMA;
				}
				break;


			default:
				estado_rx = ESPERANDO_TRAMA;
				break;
		}
	}
}
```

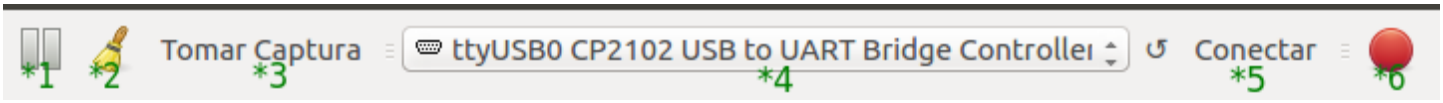
QTCREATOR:

El programa de QT se encuentra adjunto en la misma carpeta de este informe, con el nombre “Medidor_ECG”.

Imagen frontal del programa:



Donde :



***1 :** Permite poner en pausa el ploteo. Esto significa que, si por más que se sigan recibiendo datos via serie, la imagen del plot se queda congelada en el valor que tenía al activar el botón.

***2:** Permite limpiar la pantalla de Plot y los valores de Pulso y PPM

***3:** Toma una captura del ploteo y lo guarda en un archivo accesible desde la ventana “Capturas de Pantalla” propia del programa.

***4:** Muestra las conexiones serie disponible para conectarse.

***5:** Permite abrir el puerto previamente establecido en el punto *4, estableciendo la conexión entre la computadora y el dispositivo externo.

***6:** Permite grabar los datos que se reciben vía serie en un archivo de texto, para poder ser luego leído por el programa, o por cualquier otro (Ej: MatLab).

Datos de Importancia:

Para poder llevar a cabo una comunicación en serie exitosa, se debe configurar al puerto con los siguientes valores:

Baud Rate = 115200 ,
Sin paridad ,
8 bits ,
1 bit stop ;

Formato de trama:

La forma en la que se transmite la trama del LPC a la PC es de la siguiente manera:

Trama : '#' 'Byte Más Significativos ADC' 'Byte Menos Significativo'

Donde luego del byte de inicio '#', los siguientes dos bytes posteriores van a corresponderse a los 8 bits más significativos del ADC, y luego a lso 8 bits menos significativos restantes de este .

Este tipo de formato de envío es conocido como Big Endian, donde en orden en el cual los bytes que se reciben representan el valor "natural" del numero.

Funcionamiento del programa:

El programa utiliza una clase principal denominada MainWindow. Dicha clase está compuesta por otras clases, fundamentales para el funcionamiento del programa, como es por ejemplo:

ChannelManager, encargada del control de los distintos canales y de sus respectivos datos.

SnapShotManager, la cual contiene las funciones necesarias para llevar a cabo las capturas de la pantalla del plot.

QserialPort, utilizada para la comunicación serie entre el programa y el dispositivo externo.

DataRecorder, cuyas funciones permiten grabar los datos recibidos vía serie en un archivo .txt.

DataFormatPanel, el cual permite el analisis de los datos recibidos. En este caso se lo utiliza para poder detectar y determinar los datos de la trama recibida.

Las demás clases son utilizadas por las previamente mencionadas para llevar a cabo sus funciones, y se pueden observar a continuación:

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

    PlotViewSettings viewSettings() const;

    void messageHandler(QtMsgType type, const QMessageLogContext &context,
                        const QString &msg);

private:
    Ui::MainWindow *ui;

    QDialog aboutDialog;
    void setupAboutDialog();

    QSerialPort serialPort;
    PortControl portControl;

    unsigned int numOfSamples;

    QList<QwtPlotCurve*> curves;
    ChannelManager channelMan;
    PlotManager* plotMan;
    QWidget* secondaryPlot;
    SnapshotManager snapshotMan;
    DataRecorder recorder;       // operated by `recordPanel`

    QLabel spsLabel;
    CommandPanel commandPanel;
    DataFormatPanel dataFormatPanel;
    RecordPanel recordPanel;
    PlotControlPanel plotControlPanel;
    PlotMenu plotMenu;
    UpdateCheckDialog updateCheckDialog;
    PulsoECG pulsoecg;
    QElapsedTimer timer;
    QTime timer2;
```

Otro objeto que utiliza el programa es el determinado por la clase PulsoECG. Este lo que realiza es buscar la periodicidad de la señal ECG que se está midiendo, calculando el tiempo entre los picos de esta.
Para ellos se utiliza la clase de Qtime, la cual mediante la invocación de una de sus funciones, devuelve el tiempo en milisegundos desde que se arrancó el timer.

Todo este proceso se activa siempre y cuando el dato dentro del buffer del canal esté listo, y se lleva a cabo mediante el siguiente codigo:

```
//La función recibe el dato ( buffer ), el puntero a la ventana principal, y un tiempo determinado.

void PulsoECG::monitorizar_pulso(FrameBuffer * buffer,  Ui::MainWindow *ui , long timer)
{

	size_t size = buffer->size();

	double data = buffer->sample(size);
	lectura_ECG = (long) data;  // cargo el dato en una variable particular

	if(lectura_anterior_ECG < lectura_ECG) // Entra al if siempre y cuando la señal este en modo creciente
	{
	   medicion_de_pulso_activa=true;

	}
	else //y luego entra aca cuando empieza a decaer nuevamente
	{

	   if( medicion_de_pulso_activa == true && (lectura_ECG >= VALOR_MAXIMO_MINIMO))
	      //entro en este if si se trata de un maximo valido, determinado por el valor maximo aceptable de la señal
	   {
	      tiempo_actual = timer; //y como detecto un maximo guardo el valor del timer
	      tiempo_entre_pulso = (tiempo_actual - tiempo_pasado); // calculo el periodo de la señal mediante la diferencia
	                                                            //de tiempos guardados

	      if ( (tiempo_entre_pulso >= MINIMO_ENTRE_PULSO)  && (tiempo_entre_pulso <= MAXIMO_ENTRE_PULSO))
	      {

	         ui -> frecLabel->setFont( f);
	         ui -> frecLabel->setText("Pulso: " + QString::number((long)(tiempo_entre_pulso)) + " ms");

	         ppm = (float) (PPM / tiempo_entre_pulso);

	         ui -> ppmLabel ->setFont( f);
	         ui -> ppmLabel ->setText("PPM: " + QString::number(ppm));
	         // actualizo los datos del mainwindow

	       }

	      tiempo_pasado = tiempo_actual; // y cargo el tiempo que tome previamente como referencia al actual
	   }
```

```
        }

    lectura_anterior_ECG = lectura_ECG;
}
```