UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INDUSTRIAL
ELECTRÓNICA INDUSTRIAL

**PROYECTO FIN DE CARRERA**

*DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA*

# SISTEMA DE ADQUISICIÓN DE DATOS EMPLEANDO UN DISPOSITIVO WINDOWS MOBILE Y UN MICROPROCESADOR

Autor: Germán Hernández Rodríguez
Tutor: Dr. D. Luis Hernández Corporales

# ÍNDICE

# ÍNDICE DE FIGURAS

# 1. INTRODUCCIÓN

# 1. INTRODUCCIÓN

## 1.1 PROPÓSITO DEL PROYECTO

El propósito del proyecto es crear un sistema de representación de datos de telemetría autónomo y portátil. En este sistema es necesario acondicionar las señales, representarlas al usuario o almacenarlas.

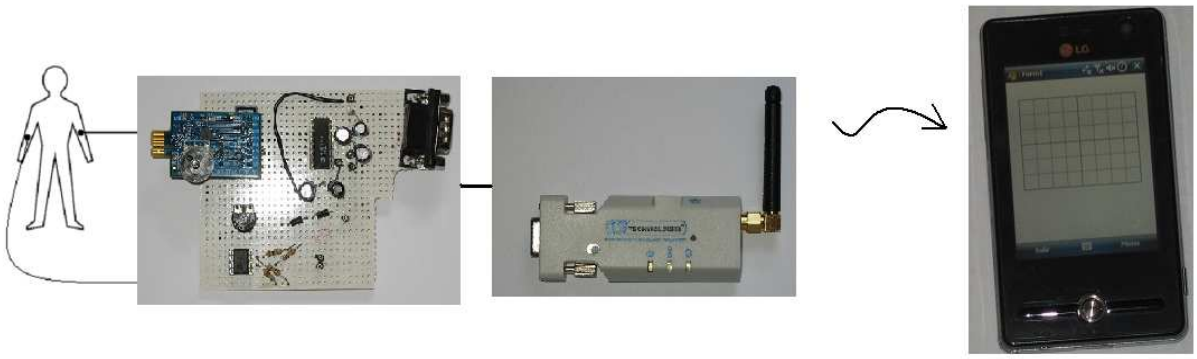


*Figura 1.1. Vista esquemática del sistema*

En nuestro proyecto se ha utilizado como display una PDA con Windows Mobile 6, el sistema de instrumentación esta formada por una etapa de acondicionamiento, el microprocesador y la electrónica adicional para mandar datos por el puerto serie. Para la realización de la comunicación entre el sistema de instrumentación y la PDA hemos utilizado el protocolo Bluetooth, por lo que tuvimos que adquirir un convertidor RS232-Bluetooth.

Para la realización del proyecto se han llevado a cabo las siguientes tareas:

- Diseño del hardware para el microprocesador.
- Programación del microprocesador usando lenguaje C.
- Programación de la PDA usando lenguaje C# y Visual Studio.

Nuestro sistema tiene un gran rango de aplicación ya que podemos monitorizar diferentes sensores analógicos de forma totalmente inalámbrica, lo que tiene muchas aplicaciones en sectores como la electromedicina, competiciones relacionadas con el mundo del motor…

Por ejemplo, este sistema podría ser aplicable para observar la actividad eléctrica del corazón de una persona. La utilización de este dispositivo permitiría el desplazamiento del paciente en el entorno hospitalario de una forma muy cómoda.

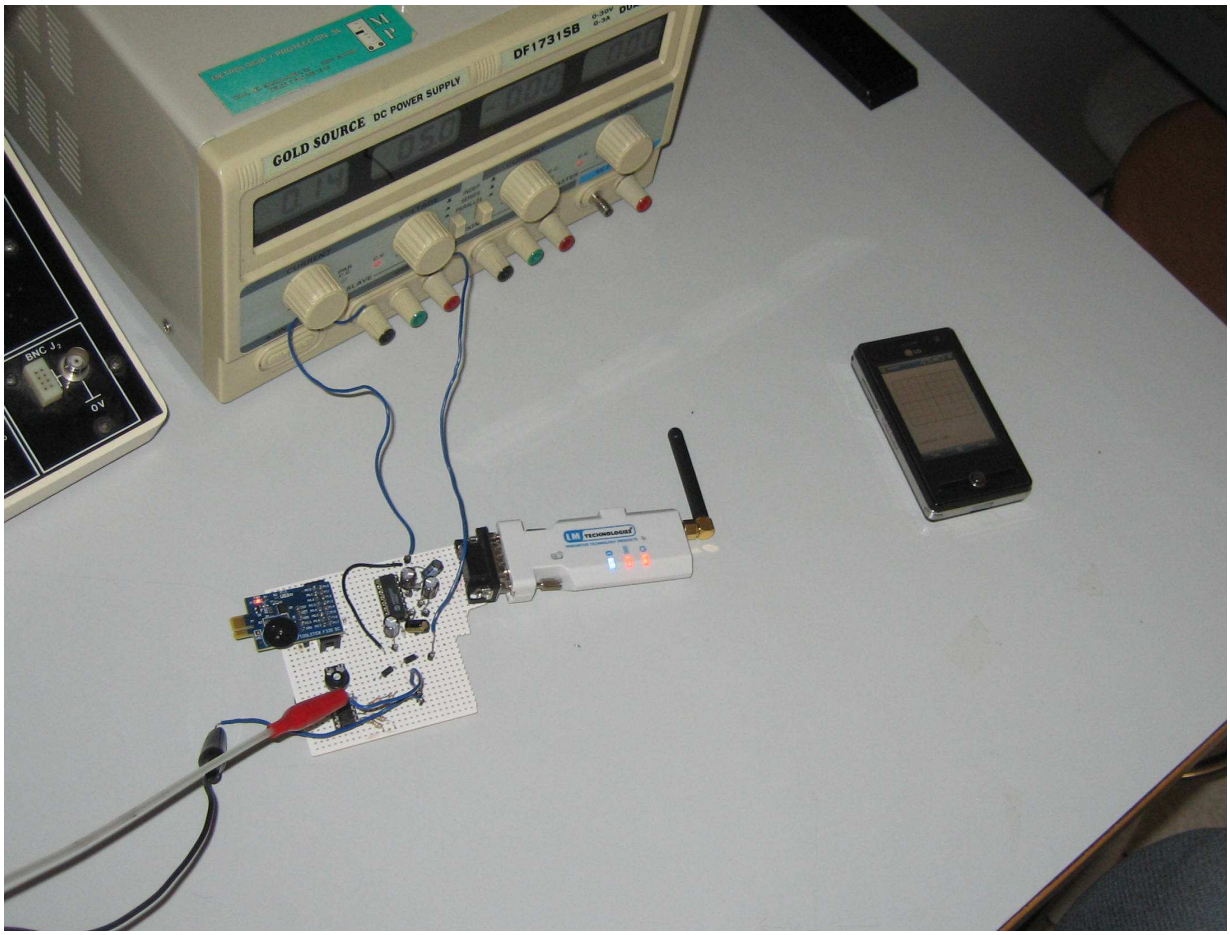



*Figura 1.2. Vista del proyecto completo*

## 1.2 ESTRUCTURA DEL PROYECTO

El proyecto se puede dividir en tres grandes tareas, la programación de la PDA, la programación del microprocesador y el diseño del hardware para el micro. A continuación vamos a dar una breve explicación de cada una de ellas:

Nuestra PDA posee como sistema operativo Windows Mobile 6, y su programación se ha realizado mediante el lenguaje de programación C#. El programa realizado tiene como objetivo final la representación de los datos provenientes del hardware. Para ello primeramente se debe programar la comunicación serie (esto implica la elección de puertos, conexión, desconexión…) y seguidamente hay que programar todo lo relacionado con la adaptación de los datos y su representación en la pantalla.

Para la programación del microprocesador hemos utilizado el lenguaje C, aplicando así los conocimientos adquiridos durante la carrera sobre desarrollo de software usando este lenguaje además del conocimiento sobre sistemas digitales. Su programación debe de cumplir dos funciones principales, capturar los datos que le entran por el convertidor A/D y la segunda, mandar los datos a través de su interfaz serie.

El hardware del proyecto debe de estar compuesto por la etapa de amplificación para adaptar a los niveles adecuados las tensiones procedentes del sensor, el microprocesador, un circuito que se encarga de convertir los niveles TTL procedentes del micro a RS-232 y el conector de puerto serie.

## 1.3 APLICACIONES DE TELEMETRÍA

Como hemos dicho anteriormente la telemetría tiene un gran rango de aplicación en sectores como la electromedicina, la ingeniería aeroespacial…
A continuación vamos a ver en más detalle su uso en cada alguna de sus aplicaciones:

### 1.3.1 Ingeniería aeroespacial

La telemetría espacial surgió de la necesidad de transmisión de medidas desde globos sonda y de la de controlar las pruebas de vuelo y verificación de aviones, cohetes, misiles, sondas, etc. Ha resultado de gran utilidad en los satélites artificiales para la transmisión a la Tierra de las mediciones efectuadas a bordo de los mismos, y en las cápsulas tripuladas, pues la seguridad del hombre en vuelo orbital depende estrechamente del sistema telemétrico.

La transmisión, mediante sondas interplanetarias de las mediciones realizadas en las proximidades de los cuerpos celestes, así como la transmisión de tomas televisivas desde, por ejemplo la Luna, ha sido posible gracias a los espectaculares avances de la telemetría.

*Figura 1.3. Imagen sobre ingeniería aeroespacial*

### 1.3.2 Electromedicina

La tecnología de comunicaciones actual aplicada a la atención sanitaria nos permite una flexibilidad y una movilidad de la monitorización de los pacientes utilizando redes de comunicaciones inalámbricas que suponen una mejora de la calidad y una reducción del coste de la atención del paciente.

Los sistemas de biotelemetría se utilizan para la monitorización remota de la situación del paciente. Algunas aplicaciones típicas son la monitorización cardíaca, la vigilancia de la presión sanguínea, la monitorización respiratoria o de episodios de apnea, etc. La telemetría biomédica registra parámetros fisiológicos y otra información relacionada con el paciente, y la envía por medio de señales electromagnéticas bidireccionales o unidireccionales permitiendo ofrecer una atención de calidad a pacientes con necesidad de atención crónica o aguda optimizando el coste, permitiendo reducir el tiempo de recuperación y la estancia en el hospital.

En una aplicación de telemetría en un entorno extrahospitalario, como puede ser el caso del hogar, se presentan en la siguiente tabla algunos de los parámetros y los tipos de señales que se pueden registrar:

| **PARAMETRO A MEDIR** | **RANGO** | **FRECUENCIA DE SEÑAL** | **TIPO DE SENSOR** |
|---|---|---|---|
| Electrocardiografía | 0,5 – 4 mV | 0,01-250 Hz | Electrodo de contacto |
| Electroencefalografía | 5 – 300 nV | DC- 150 Hz | Electrodo de contacto |
| Electromiografía | 0,1 – 5 mV | DC- 10000Hz | Electrodo de contacto |
| Flujo respiratorio | 0-600 l/m | DC- 40 Hz | Diferencia de temperatura |
| Saturación de oxígeno en sangre | 0-100 % | DC- 1 Hz | Colorimetría |

*Tabla 1.4. Diferentes tipos de mediciones en electromedicina*

## 1.4 TELEMETRÍA EN EL ENTORNO DE LA ELECTROMEDICINA

### 1.4.1 Electrocardiógrafos

Los electrocardiógrafos son aparatos electrónicos que captan y amplían la actividad eléctrica del corazón a través de electrodos. Los electrocardiógrafos suelen estar compuestos de los siguientes bloques: Circuito de protección, Regulación de offset, Circuito de aislamiento, Circuito de filtrado, Amplificador, Microcontrolador y un Registrador.

Hoy en día en el mercado existe una gran oferta variedad, entre los que podemos encontrar aquellos que utilizan el Standard Bluetooth para la transmisión de lo datos a semejanza de nuestro proyecto. A continuación mostramos las características más importantes de estos dispositivos:

- Medida continua de ECG
- Comunicación inalámbrica vía Bluetooth
- Almacenamiento interno de datos no enviados vía Bluetooth
- Calculo de la frecuencia cardiaca
- Frecuencia de muestreo de 500 Hz
- Ancho de Banda de 0-150 Hz


*Figura 1.5. Electrocardiógrafo con tecnología Bluetooth*

### 1.4.2 Terminales industriales Windows Mobile

En el ámbito industrial han empezado a proliferar terminales con Windows Mobile, estos dispositivos están orientados hacia el sector del correo y la logística, y en empresas de servicios que requieren el envío de datos e imágenes en tiempo real a la oficina central.

Los usuarios de estos terminales pueden aprovecharse del Gestor de Dispositivos de Microsoft (Mobile Device Manager) para mejorar tareas como la configuración de los equipos, las actualizaciones de software o las políticas de seguridad. Este gestor es especialmente importante para los directores de las compañías, ya que con él pueden contar, de forma remota, con una visibilidad total del estado y de las operaciones de la red de terminales portátiles. Una de las características más interesantes de este gestor reside en sus funcionalidades relacionadas con la seguridad, tanto del uso del terminal como del acceso de los usuarios a la red, los recursos corporativos y la información confidencial

Los terminales suelen estar provistos de un módulo GPS integrado que puede transmitir la ubicación actual, al mismo tiempo que la cámara digital integrada realiza rápidas fotografías y el programa de correo electrónico registra nueva información para el usuario. Todo esto ofrece una productividad y eficiencia máxima a los usuarios.


*Figura 1.6. Terminal Industrial con Windows Mobile*

### 1.4.3 Modulo Bluetooth

La tecnología inalámbrica *Bluetooth* es un sistema de comunicaciones de corto alcance, cuyo objetivo es eliminar los cables en las conexiones entre dispositivos electrónicos, tanto portátiles como fijos, para el intercambio de distintos tipos de datos entre ellos. Las características principales de esta tecnología son su fiabilidad, bajo consumo y mínimo coste. Varias de las funciones de la especificación principal son opcionales, lo que permite la diferenciación de los productos. El núcleo del sistema *Bluetooth* consiste en un transmisor de radio, una banda base y una pila de protocolos.

En el mercado hay una gran cantidad de módulos, estos se suelen diferenciar por su alcance llegando los de clase 1 a los 100 metros, y los de clase 2 a unos 20 metros. Los más modernos suelen disponer del Standard Bluetooth 2.1 +EDR (Enhanced Data Rate) con una velocidad de transmisión de 3Mbps. La característica principal de estos módulos es que facilitan enormemente la capacidad de añadir a nuestros diseños conectividad Bluetooth sin ningún tipo de complicación.

*Figura 1.7. Vista de un módulo Bluetooth*

# 2. DISEÑO DEL SISTEMA

# 2. DISEÑO DEL SISTEMA

A la hora de realizar el diseño de nuestro sistema, lo primero que teníamos que pensar era en que queríamos que hiciera nuestro sistema. Nuestro sistema se debe de encargar de acondicionar la señal proveniente de un sensor analógico, digitalizar los datos y enviarlos por el puerto serie para representarlos en un display. Para ello hemos tenido que utilizar una serie de componentes ya existentes, así el acondicionamiento lo realizamos a través del amplificador de instrumentación AD620, para la digitalización de datos y su posterior envío por el puerto serie usamos el microcontrolador 8051F330 y para la representación de lo datos utilizamos una PDA.

Una vez conocidos los componentes que íbamos a utilizar, la principal decisión que teníamos que tomar era la manera en que íbamos a comunicar nuestro sistema de instrumentación con la PDA. Para ello tuvimos que discernir entre las dos formas de comunicación que nos permitía la PDA, el Standard Bluetooth o través del puerto USB utilizando ActiveSync, pero este último era un procedimiento demasiado complejo y no era viable, lo que nos hizo decantarnos por el uso del Bluetooth.

Las ventajas que nos proporciona la tecnología Bluetooth es la de ser inalámbrica y a la vez, a la hora de la programación había que considerar la comunicación como un simple puerto serie sin tener que preocuparse de otras cuestiones. Esto se debe al protocolo RFCOMM, este protocolo se encarga de emular los parámetros de un cable de serie y el estado de un puerto RS-232 para transmitir datos en serie.

## 2.1 DIAGRAMA DE BLOQUES DEL SISTEMA

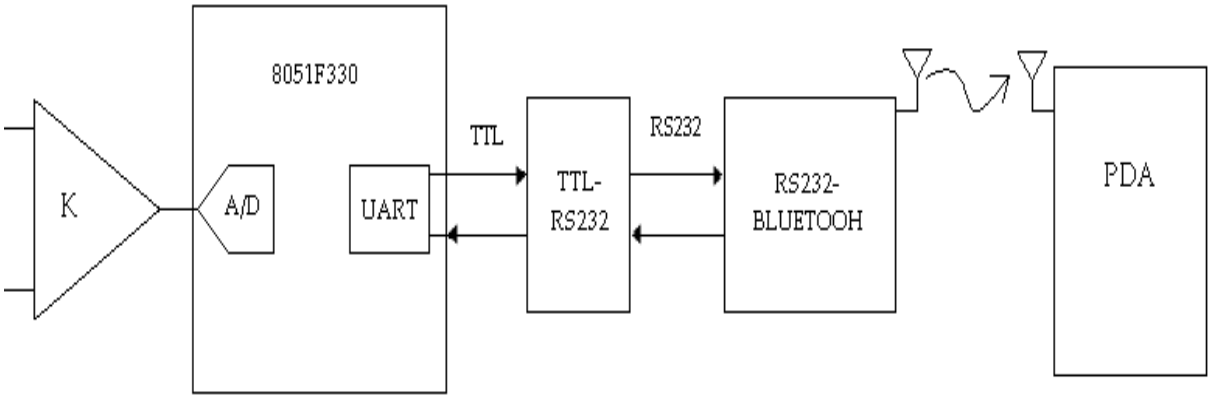



*Figura 2.1. Diagrama de bloques del sistema*

El primer paso en el sistema se basa en adaptar los valores procedentes del sensor analógico que se conecte al circuito, para ello hemos realizado una etapa de amplificación. A continuación, los valores se digitalizan mediante el convertidor A/D que posee el microprocesador, muestreando 10 veces por segundo. El microprocesador a su vez manda todos los valores a través de su interfaz serie en tiempo real, que tienen que ser adaptados de TTL a RS232, para que estos puedan ser transmitidos correctamente a la PDA gracias al adaptador RS232-Bluetooth. Una vez recibidos por la PDA el software se encarga de procesar los valores y representarlos en la pantalla mediante una gráfica.

## 2.2 ELEMENTOS NECESARIOS PARA LA IMPLEMENTACIÓN DEL SISTEMA.

Para la realización del proyecto será preciso contar con lo siguientes elementos:

- Microprocesador
- Amplificador de instrumentación
- Adaptador RS232/Bluetooth
- PDA

### 2.2.1 Amplificador de instrumentación

Para realizar la etapa de amplificación hemos utilizado el amplificador de instrumentación AD620, este es el que se encarga de adaptar a los niveles de tensión adecuados los valores procedentes de los sensores analógicos.

Sus características cumplen a la perfección las especificaciones de nuestro sistema ya que permite una ganancia con un rango 1-1000, y posee un pequeño consumo y bajo ruido.

Además para establecer su ganancia sólo tenemos que determinar una resistencia que viene definida por la siguiente expresión:

$$G=\frac{49.4K\Omega}{R_G}+1$$

### 2.2.2 Microprocesador

La elección del microprocesador es esencial ya que sus características deben satisfacer completamente, todos los requisitos de los que requiere un sistema. En nuestro caso el microprocesador debe estar provisto de un convertidor A/D que sea capaz de muestrear 50 veces por segundo y un interfaz serie, para poder realizar la comunicación. Por ello hemos elegido el PIC 8051F330, ya que cumple perfectamente los requerimientos del proyecto y el fabricante nos proporciona herramientas de desarrollo gratuitas, a la vez que una documentación técnica completa.

Para facilitar la programación y el posterior uso en el circuito hemos adquirido el componente ToolStick F330DC, que se trata de un circuito integrado en el que viene el microprocesador y su diseño nos permite programarlo de una forma muy cómoda y a la vez podemos usarlo distintos circuitos. En la siguiente figura podemos apreciar el diagrama de pines del microprocesador.

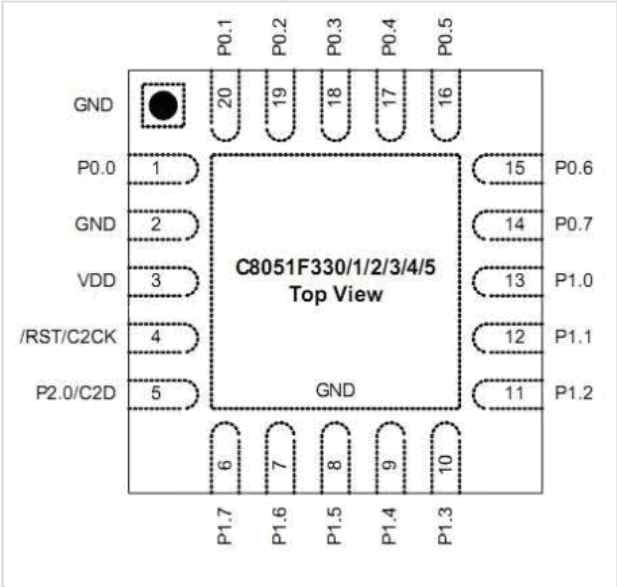



*Figura 2.2. Diagrama de pines del 8051F330*

### 2.2.3 PDA

A la hora de la elección de una PDA teníamos que tener en cuenta el Sistema Operativo con el que pensábamos trabajar ya que se nos presentaron dos alternativas, la elección de Windows Mobile o por el contrario utilizar Symbian. Finalmente la decisión fue la elección de Windows Mobile ya que había una gran cantidad de información en Internet tanto en la página oficial de Microsoft como en otra gran cantidad de páginas para poder desarrollar una aplicación.

La PDA que hemos elegido es LG-KS20. Esta dispone de Bluetooth 2.0 y su sistema operativo es Windows Mobile 6 edición profesional. Para su programación se podía usar dos tipos de lenguajes de programación el C++ o el C#, siendo este último el elegido. El lenguaje C# es un lenguaje orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET , este lenguaje recoge las mejores características de muchos otros lenguajes, fundamentalmente de Visual Basic, Java y C++.

Debido a que el lenguaje C# fue específicamente diseñado para la plataforma .NET, existen una gran cantidad de librerías de clases que pueden ser utilizados para escribir nuestro programa lo que facilita mucho el desarrollo de la aplicación. Además nos permite la utilización de la herramienta Visual Studio.NET, que como entorno visual que es, permite diseñar la interfaz de la aplicación de manera visual, sin más que arrastrar con el ratón los elementos que necesitemos (botones, lista de selección, etc.) sobre las posiciones adecuadas en la ventana de nuestra aplicación.

Todo este conjunto de características ha facilitado mucho la labor de comprensión, aprendizaje y posterior uso del lenguaje C# para el desarrollo de nuestra aplicación.

### 2.2.4 Adaptador RS232-Bluetooth

La elección de la tecnología Bluetooth conllevó la necesidad de la conversión de datos RS232-Bluetooth. En el mercado existe un gran variedad de ellos, pero no todos tienen las mismas características, ya que no todos tienen el mismo alcance, y además algunos tiene la dificultad de que hay que soldarlos directamente a las señales procedentes de la UART, por ello hemos utilizado el dispositivo LM058. Este dispositivo nos proporciona un gran alcance de hasta unos 100 metros, los requisitos de alimentación del aparato se adaptan perfectamente a nuestro sistema ya que puede ser alimentado entre +5V y +6V, y su consumo de corriente no es muy elevado ya que su máximo son 90mA.


*Figura 2.3. Vista del dispositivo LM058*

Además de poseer una serie de características que cumplen a la perfección nuestros requisitos, el fabricante pone a nuestra disposición toda la información necesaria para la programación del dispositivo. Así podemos configurarlo mediante el software LM049 en la que podemos establecer todos los parámetros de una forma rápida y sencilla, o podemos usar los comandos AT (*instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un terminal*) que nos indica el fabricante en el manual. En la figura se muestra una captura de pantalla del software de configuración.

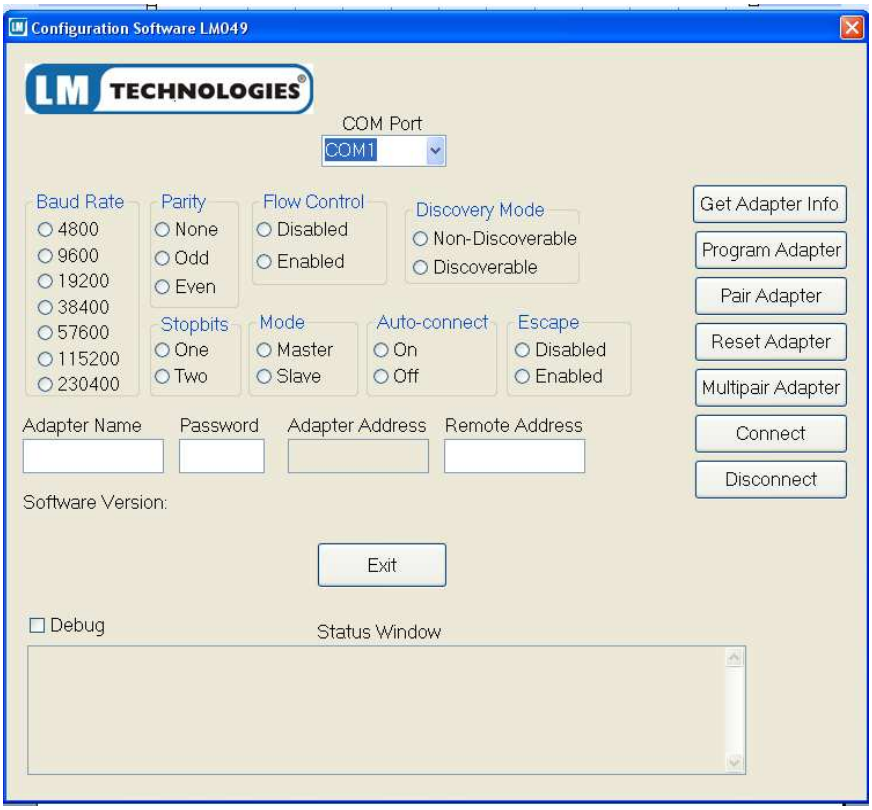



*Figura 2.4. Vista del programa LM049*

# 3. DISEÑO DEL HARDWARE

# 3. DISEÑO DEL HARDWARE

## 3.1 DESCRIPCIÓN DEL CIRCUITO

### 3.1.1 Circuito de amplificación

Como ya hemos visto en los capítulos anteriores el circuito de amplificación se basa en el amplificador de instrumentación AD620. El potenciómetro R1 es el que se encarga de determinar la ganancia lo que nos proporciona tener una ganancia variable y las resistencias R2 y R3 se encargan de polarizar el AD620 y para conseguir que el circuito funcione correctamente.

La salida de este circuito nos proporciona un determinado voltaje, lo que implica que el microprocesador deberá convertir estos voltajes mediante su conversor analógico digital para poder trabajar con ellos.

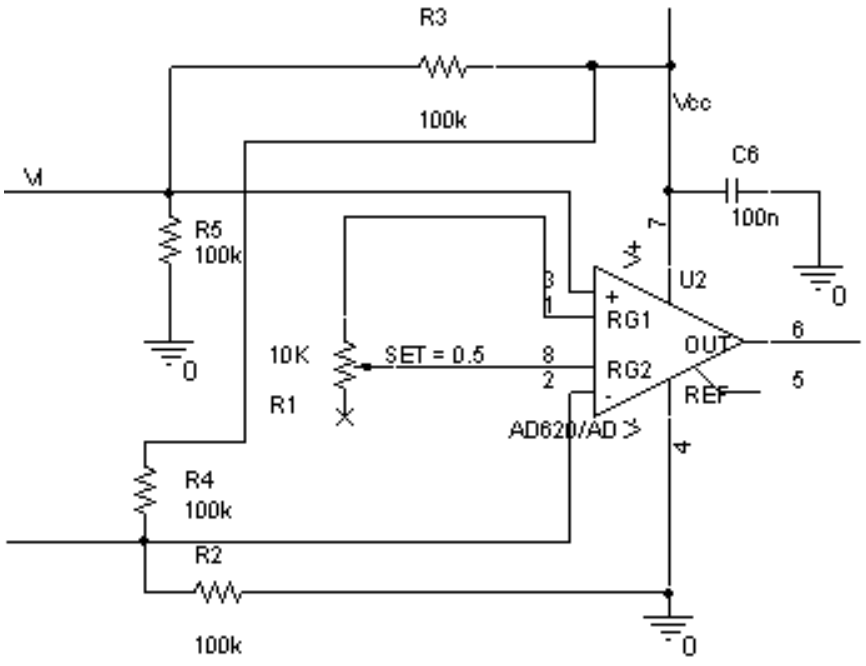



*Figura 3.1. Circuito de amplificación*

### 3.1.2 ToolStick F330 DC

Como ya hemos dicho usamos este modulo debido a la gran dificultad que seria usar el microprocesador 8051F330 ya que este tiene unas dimensiones muy reducidas lo que dificulta mucho la integración en el hardware. Por ello al usar este componente tenemos que colocar en nuestro hardware únicamente dos conectores.

Los componentes principales de este módulo son un par de LEDS, un potenciómetro, el microprocesador 8051F330, y un área en el que tenemos acceso a todos los pines del dispositivo. En la siguiente figura podemos apreciar el componente y sus distintas partes.

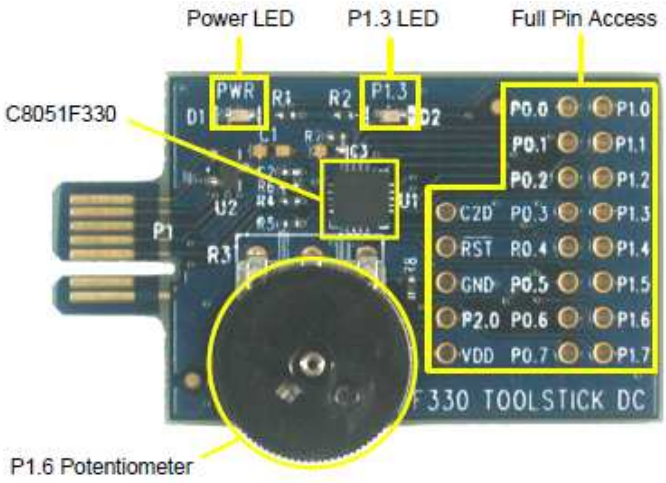



*Figura 3.2. Vista frontal del ToolStick F330DC*

### 3.1.3 Puerto serie

Durante la ejecución del programa el circuito irá enviando la información hacia la PDA para que este pueda representarla, y así el usuario pueda verla. Los datos se transmitirán por el puerto serie.

Para establecer el puerto serie necesitamos un conector DB9 macho colocado en la placa, al cual le será insertado el adaptador RS232-Bluetooth. La conexión que se ha realizado se usan sólo los pines de Transmisión y

Recepción de Datos, lo que implica que sea una conexión sin handshake y además hemos puesto el pin 9 a +5V ya que de esa manera alimentamos el adaptador RS232-Bluetooth. En la figura se muestra una vista del conector y las funciones de sus pines.

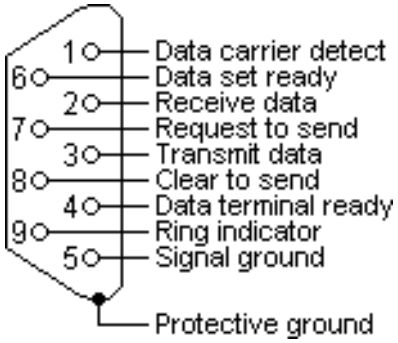



*Figura 3.3 Descripción de los pines del conector DB9*

El Standard serie funciona con +10 y -10 V. Los niveles de TTL con los que trabaja el PIC son 0 y 5V, por lo que es necesario adaptarlos. Lo conseguiremos con la ayuda del integrado MAX232, que tiene la ventaja de proporcionar los niveles RS232 a partir de una alimentación de 5V.

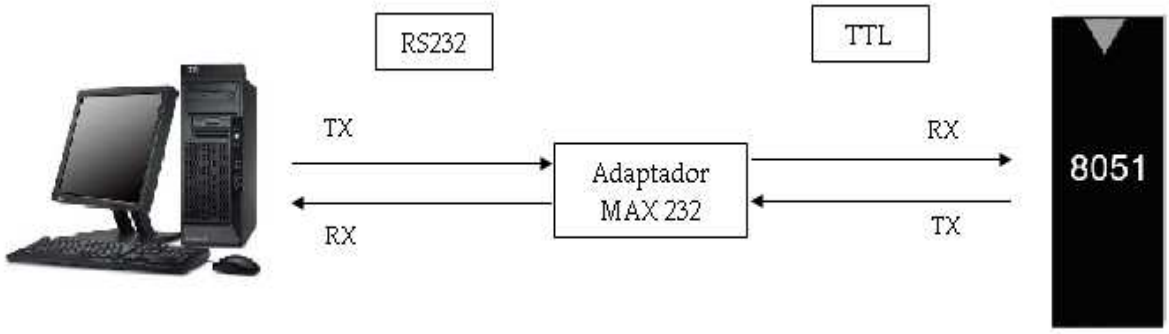



*Figura 3.4. Esquema de conexión del adaptador de niveles MAX232*

## 3.2 ESQUEMA ELÉCTRICO DETALLADO DEL CIRCUITO

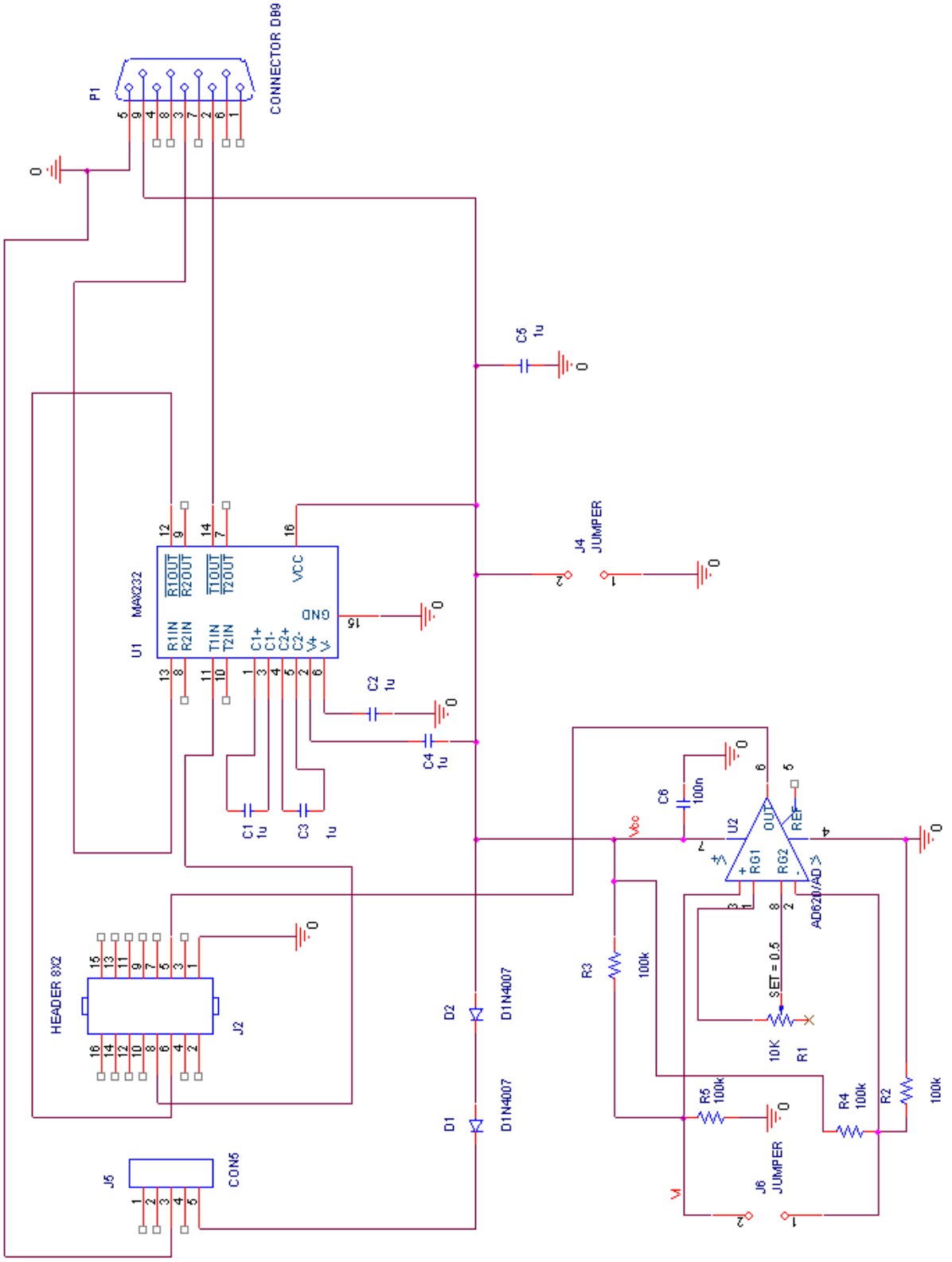



*Figura 3.5. Esquema eléctrico detallado del circuito*

# 4. DISEÑO DEL SOFTWARE EMBEBIDO

# 4. DISEÑO DEL SOFTWARE EMBEBIDO

## 4.1 ENTORNO DE DESARROLLO

El entono de desarrollo ha sido el Silicon Laboratories IDE versión 1.6, que al igual que el Toolstick F330 DC pertenece a la empresa Silabs. Es una herramienta gratuita que trabaja sobre Windows para el desarrollo de aplicaciones para microcontroladores. Su uso es bastante intuitivo y cuenta con las herramientas necesarias para la realización del proyecto. Permite editar, compilar y programar. Además se ha utilizado el Toolstick Terminal que nos permite verificar el buen funcionamiento del programa. En la figura se muestra una captura de pantalla de la aplicación.

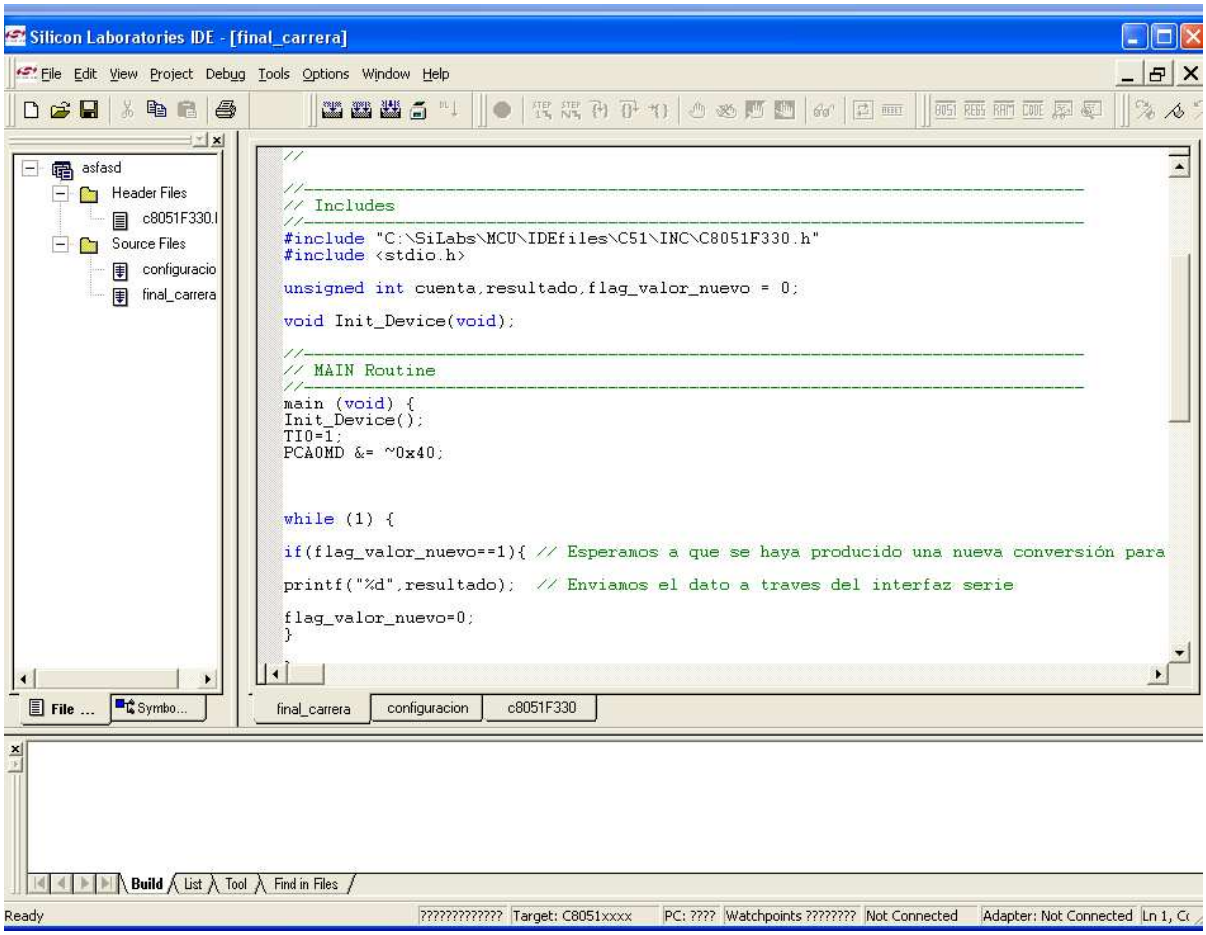



*Figura 4.1. Vista del programa Silicon Laboratories IDE*

## 4.2 ORGANIGRAMA DEL SOFTWARE

En este apartado se mostrarán los diagramas de flujo de las funciones que intervienen en el programa: main() y la subrutina de atención a interrupción del convertidor A/D.

- Función main():

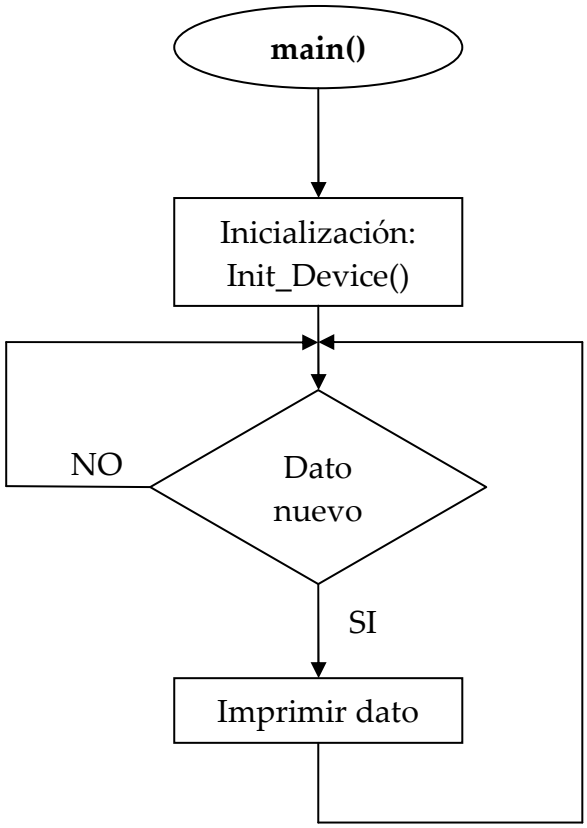



*Figura 4.2. Diagrama de flujo de la función main()*

- Interrupción:

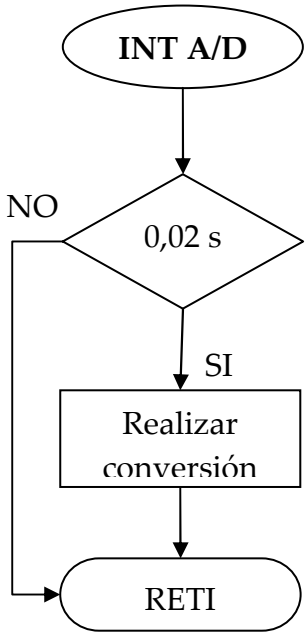



*Figura 4.3. Flujograma de la subrutina de la interrupción del A/D*

- Función Init_Device():


*Figura 4.4. Diagrama de flujo de Init_Device().*

En el siguiente apartado 4.3 se explicará detalladamente cada módulo, justificando los pasos dados para traducir los diagramas de bloques a lenguaje C.

## 4.3 DESCRIPCIÓN DE LOS MODULOS

### 4.3.1 Función main()

Es la función principal del programa. Lo primero que hace es llamar a la función Init_Device() para configurar todos los registros del microprocesador.

Una vez configurado el micro entramos en un bucle sin fin, en el que evaluamos continuamente si el A/D ha realizado una nueva conversión, una vez producida mandará por el puerto serie el valor adquirido en la conversión.

Una consideración muy importante a tener en cuenta es como mandamos cada nuevo valor de tal forma que la PDA pueda distinguir cada valor recibido. Para ello mandamos cada valor seguido de un "\n", para que así la PDA lea en su buffer cada vez hasta que encuentre el retorno de carro.

### 4.3.2 Interrupción

El programa hace uso de la interrupción del convertidor A/D. Esta interrupción está configurada para que salte con una frecuencia de 8 KHz, como son demasiadas valores por segundo, lo que hacemos es llevar una cuenta de las veces que entra y así cuando ha entrado 160 veces realizamos la adquisición de un nuevo valor, proporcionándonos así 50 conversiones cada segundo.

### 4.3.3 Función Init_Device()

En esta función inicializaremos los registros. Configuraremos los puertos como entradas o salidas, configuraremos las interrupciones y sus prioridades, el interfaz serie y los timers.

Así pues, configuraremos la comunicación serie para una transmisión asíncrona de 8 bits con una velocidad de 19200 baudios, y la interrupción del convertidor A/D para que esta salte 8000 veces por segundo.

## 4.4 LISTADO COMENTADO DEL PROGRAMA

### 4.4.1 Programa principal

```
//-----------------------------------------------------------------------------
----
// mi_programa.c
//-----------------------------------------------------------------------------
----
// Copyright I 2009 UC3M.
//
// AUTH: LH, MG
// DATE: 14 JUL 2009
//
// Este programa contiene las cabeceras del programa principal,
// la configuracion y las rutinas de interrupcion del 8051F330
//
// Target: C8051F330/1/2/3/4/5
//
// Tool chain: KEIL Eval ‘c’
//

//-----------------------------------------------------------------------------
----
// Includes
//-----------------------------------------------------------------------------
----
#include “C:\SiLabs\MCU\IDEfiles\C51\INC\C8051F330.h”
#include <stdio.h>

unsigned int cuenta,flag_valor_nuevo = 0;
unsigned short resultado= 0;

void Init_Device(void);

//-----------------------------------------------------------------------------
----
// MAIN Routine
//-----------------------------------------------------------------------------
----
main (void) {
Init_Device();
TI0=1;
PCA0MD &= ~0x40;

while (1) {

while(flag_valor_nuevo!=1){} // Esperamos a que se haya producido una nueva
conversión para enviar el dato

printf(“%d\n”,resultado);  // Enviamos el dato a traves del interfaz serie

flag_valor_nuevo=0;
}
```

```
}

}// FIN DEL PROGRAMA PRINCIPAL


// RUTINA DE INTERRUPCION DEL ADC. SE DISPARA AUTOMATICAMENTE
void ADC0_ISR (void) interrupt 10
{

if(cuenta<159){   // Llevamos un contandor para realizar menos conversiones
por segundo

cuenta++;

TMR2RLH       = 0xF9; // Cargo el timer 2 para tener 8.000 conversiones por
segundo

AD0INT=0;


}

else{

flag_valor_nuevo=1;

cuenta=0;

resultado=ADC0H;

TMR2RLH       = 0xF9;

AD0INT=0;

}
}
```

### 4.4.2 Configuración de los registros

```
//////////////////////////////////////////
//  Archivo de configuración de registros  //
//////////////////////////////////////////

#include "C:\SiLabs\MCU\IDEfiles\C51\INC\C8051F330.h"

// Funciones de inicialización espcificas,
// Llamadas desde la función Init_Device()
void Reset_Sources_Init()
{
    RSTSRC    = 0x04;
}

void Timer_Init()
{
    TCON      = 0x55;
    TMOD      = 0x21;
    TH1       = 0xCB; // Para una velocidad de transmisión de 19.200
baudios
    TMR2CN    = 0x0C;
    TMR2RLH   = 0xF9; // Cargo el timer 2 para tener 8.000 conversiones por
segundo

}


void UART_Init()
{
    SCON0     = 0x10;
}


void ADC_Init()
{
    AMX0P     = 0x0D;
    AMX0N     = 0x0F;
    ADC0CF    = 0xFC;
    ADC0CN    = 0x82;
}


void DAC_Init()
{
    IDA0CN    = 0xF2;
}


void Voltage_Reference_Init()
{
    REF0CN    = 0x0A;
}
```

```
void Port_IO_Init()
{
    // P0.0  -  Unassigned,  Open-Drain, Digital
    // P0.1  -  Skipped,     Open-Drain, Analog
    // P0.2  -  Unassigned,  Open-Drain, Digital
    // P0.3  -  Unassigned,  Open-Drain, Digital
    // P0.4  -  TX0 (UART0), Push-Pull,  Digital
    // P0.5  -  RX0 (UART0), Open-Drain, Digital
    // P0.6  -  Unassigned,  Open-Drain, Digital
    // P0.7  -  Unassigned,  Open-Drain, Digital

    // P1.0  -  Unassigned,  Open-Drain, Digital
    // P1.1  -  Unassigned,  Open-Drain, Digital
    // P1.2  -  Unassigned,  Open-Drain, Digital
    // P1.3  -  Unassigned,  Open-Drain, Digital
    // P1.4  -  Unassigned,  Open-Drain, Digital
    // P1.5  -  Skipped,     Open-Drain, Analog
    // P1.6  -  Skipped,     Open-Drain, Analog
    // P1.7  -  Skipped,     Open-Drain, Analog

    P0MDIN    = 0xFD;
    P1MDIN    = 0x1F;
    P0MDOUT   = 0x10;
    P0SKIP    = 0x02;
    P1SKIP    = 0xE0;
    XBR0      = 0x01;
    XBR1      = 0x40;
}

void Oscillator_Init()
{
    OSCICN    = 0x83;
}

void Interrupts_Init()
{
    IE        = 0x87;
    EIE1      = 0x08;
    EIP1      = 0x08;
    IT01CF    = 0x32;
}

// Función de inicialización del dispositivo
void Init_Device(void)
{
    Reset_Sources_Init();
    Timer_Init();
    UART_Init();
    ADC_Init();
    DAC_Init();
    Voltage_Reference_Init();
    Port_IO_Init();
    Oscillator_Init();
    Interrupts_Init();
}
```

# 5. DISEÑO DEL SOFTWARE DE LA PDA

# 5. DISEÑO DEL SOFTWARE DE LA PDA

## 5.1 ENTORNO DE VISUAL STUDIO

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C# y Visual C++ utilizan todos el mismo entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y hace más sencilla la creación de soluciones en varios lenguajes. A continuación hablaremos de las principales herramientas y recursos que tiene Visual Studio:

En la siguiente imagen podemos observar el **Toolbox**, en ella se pueden encontrar todos los controles que queremos agregar al proyecto como cajas de texto, label, tablas, combo box, listas, botones que simplemente con arrastrar ya tienes tu control en el Form. También nos encontramos con el **Solution Explorer,** aquí es donde podemos ver todos los archivos de tu proyecto como las imágenes, cantidad de formularios, base de datos y nos permite agregar elementos ya existentes o nuevos.

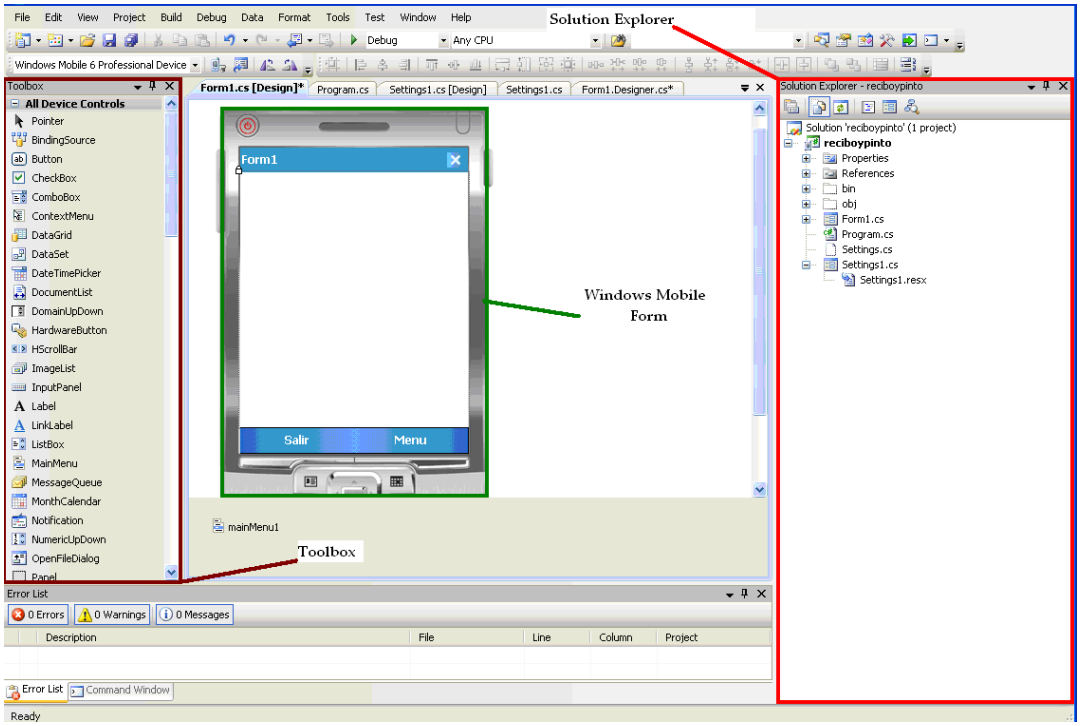



*Figura 5.1. Vista primera Entorno de Visual Studio*

Otra herramienta importante es la **Ventana de propiedades** cada objeto que este en nuestro formulario tiene sus propiedades distintas, así si seleccionamos uno de ellos, la ventada de propiedades mostrara cada una de ellas y podrás modificarlas.

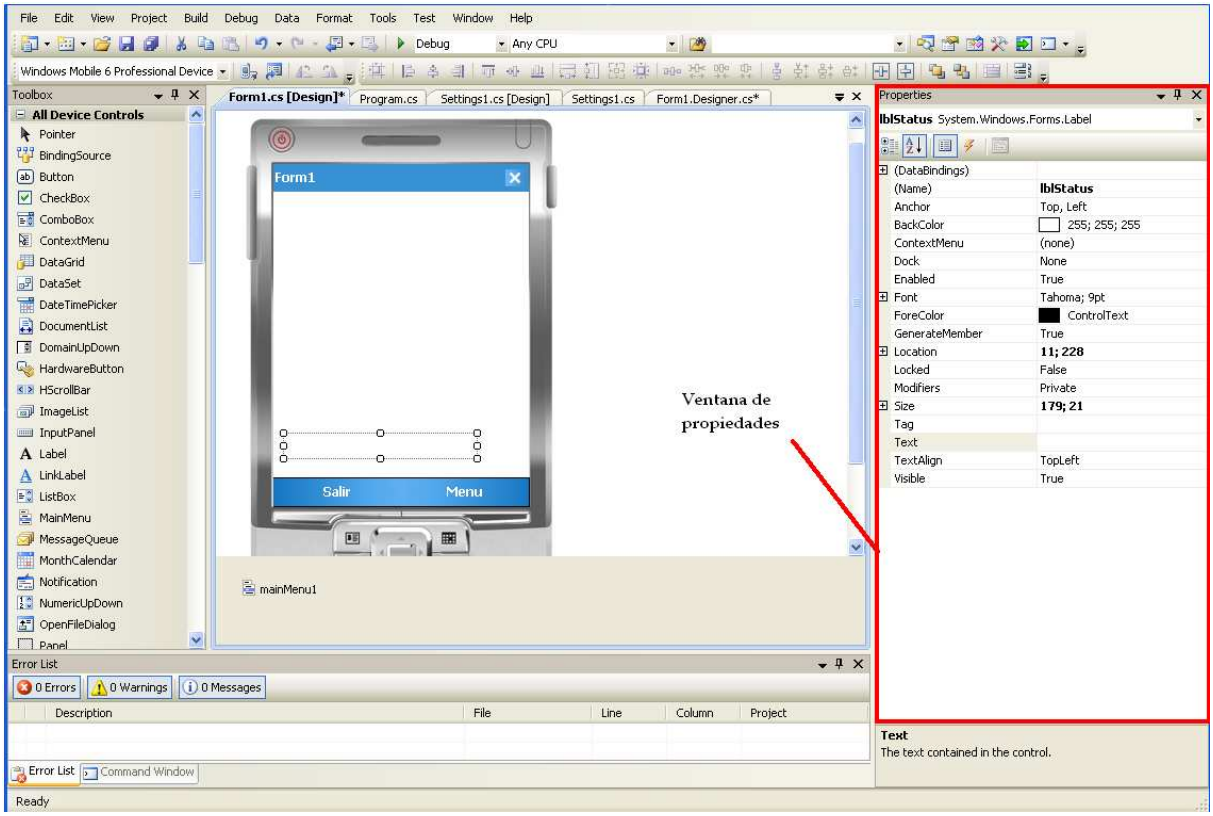



*Figura 5.2. Vista segunda Entorno de Visual Studio*

Una vez que agregamos los controles ahora debemos darle funcionalidad si damos dos veces click a uno de los controles vamos al Editor de código, en el que se agrega por defecto un método con un evento asociado a ese control, por ejemplo, en el caso de un botón el evento por defecto es Click. En el método es donde deberemos añadir el código para que nuestro programa haga lo que deseamos cuando tocamos el botón.

A todas estas funcionalidades hay que añadir las de compilación y depuración de errores. Los recursos anteriormente descritos son los más básicos y útiles para la realización de cualquiera proyectos utilizando Visual Studio.

## 5.2 PROCEDIMIENTO PARA GENERAR UNA APLICACIÓN

Lo primero que hay que hacer para generar una aplicación para un dispositivo Windows Mobile, es tener instalado en nuestro ordenador los siguientes componentes: Microsoft Visual Studio 2008, Microsoft .NET Compact Framework 3.5, ActiveSync 4.5 y Windows Mobile 6 Professional SDK.

Una vez que tengamos instalados todos los componentes y programas citados anteriormente estamos preparados para crear nuestra primera aplicación para Windows Mobile. A continuación describiremos los pasos básicos para ello:

1. Abriremos el programa Microsoft Visual Studio 2008, a continuación en el menú **File,** elegimos **New** y hacemos click en **Project.**

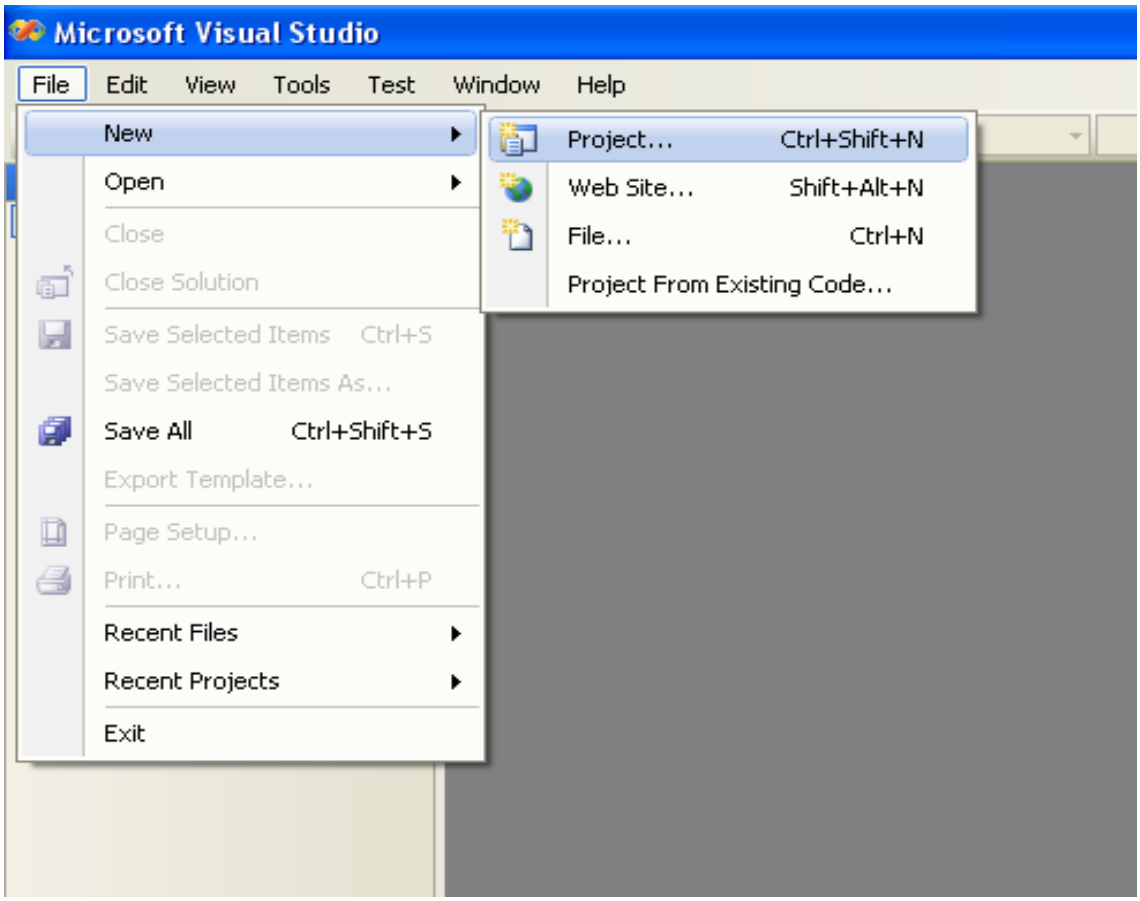



*Figura 5.3. Vista del primer paso para generar una aplicación*

2. En el panel de "Project types", expandir Visual C# y seleccionar **Smart Device.** También hay que ponerle un nombre a nuestro proyecto.

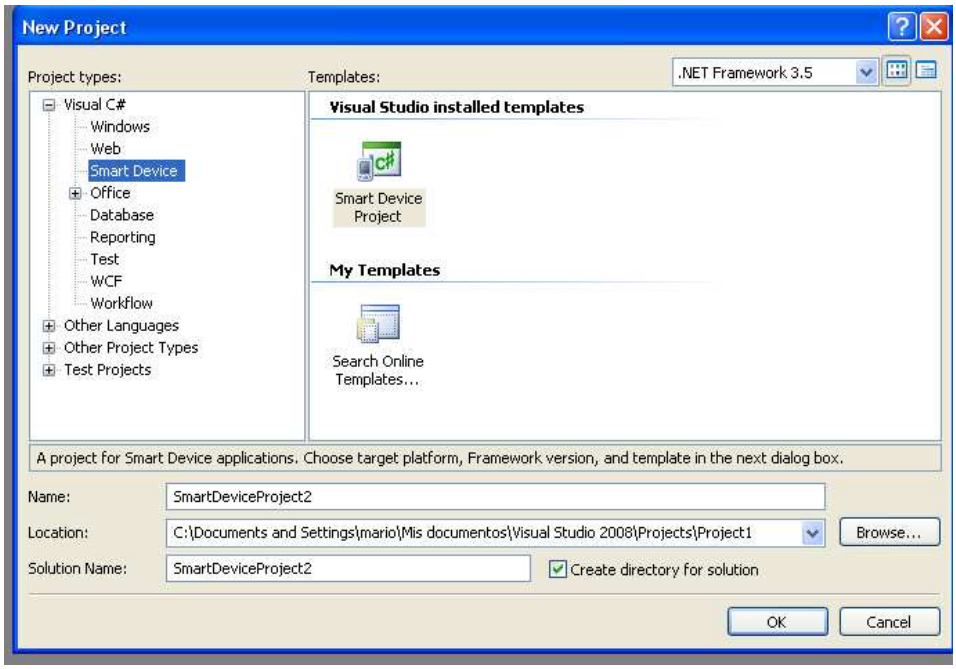



*Figura 5.4. Vista del segundo paso para generar una aplicación*

3. En el panel "Templates" seleccionar **Device Application** y en Target Platform elegir **Windows Mobile 6 Professional SDK.**

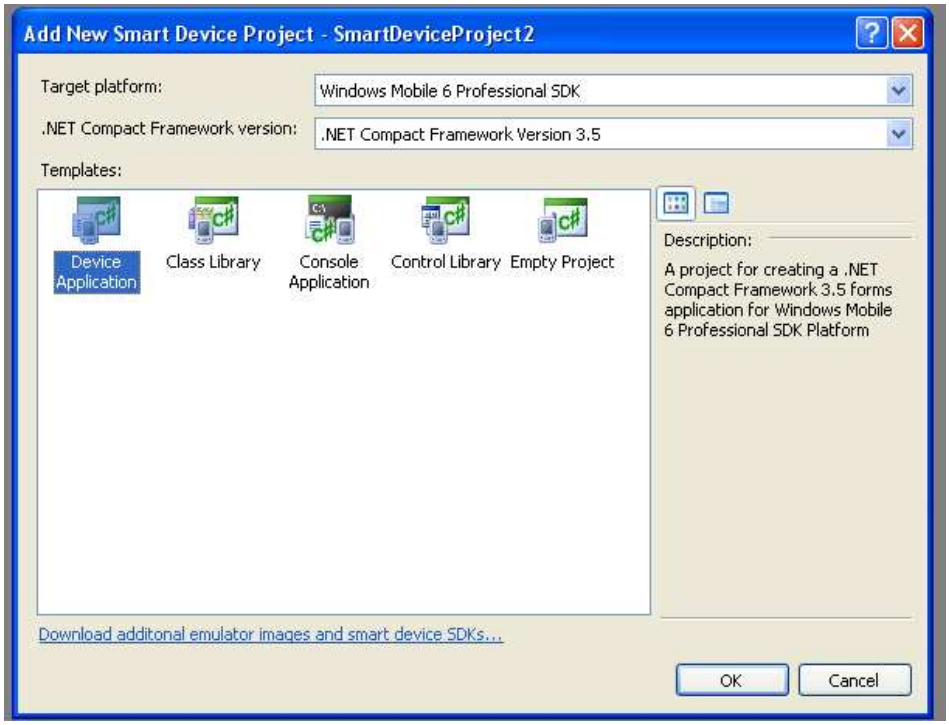



*Figura 5.5. Vista del tercer paso para generar una aplicación*

4. Una vez que el proyecto este creado, hay que desarrollar nuestra aplicación. Cuando nuestra aplicación este acabada y no contenga errores, podremos dar paso ha su depuración mientras esta funcionando en nuestra PDA. Para ello, tenemos que hacer click en **Start Debugging**, a continuación seleccionamos donde queremos depurar la aplicación en nuestro caso seleccionaremos **Windows Mobile 6 Professional Device**, con lo que se lanzará el programa en la PDA mientras podemos ir depurándolo en el entorno del Visual Studio.

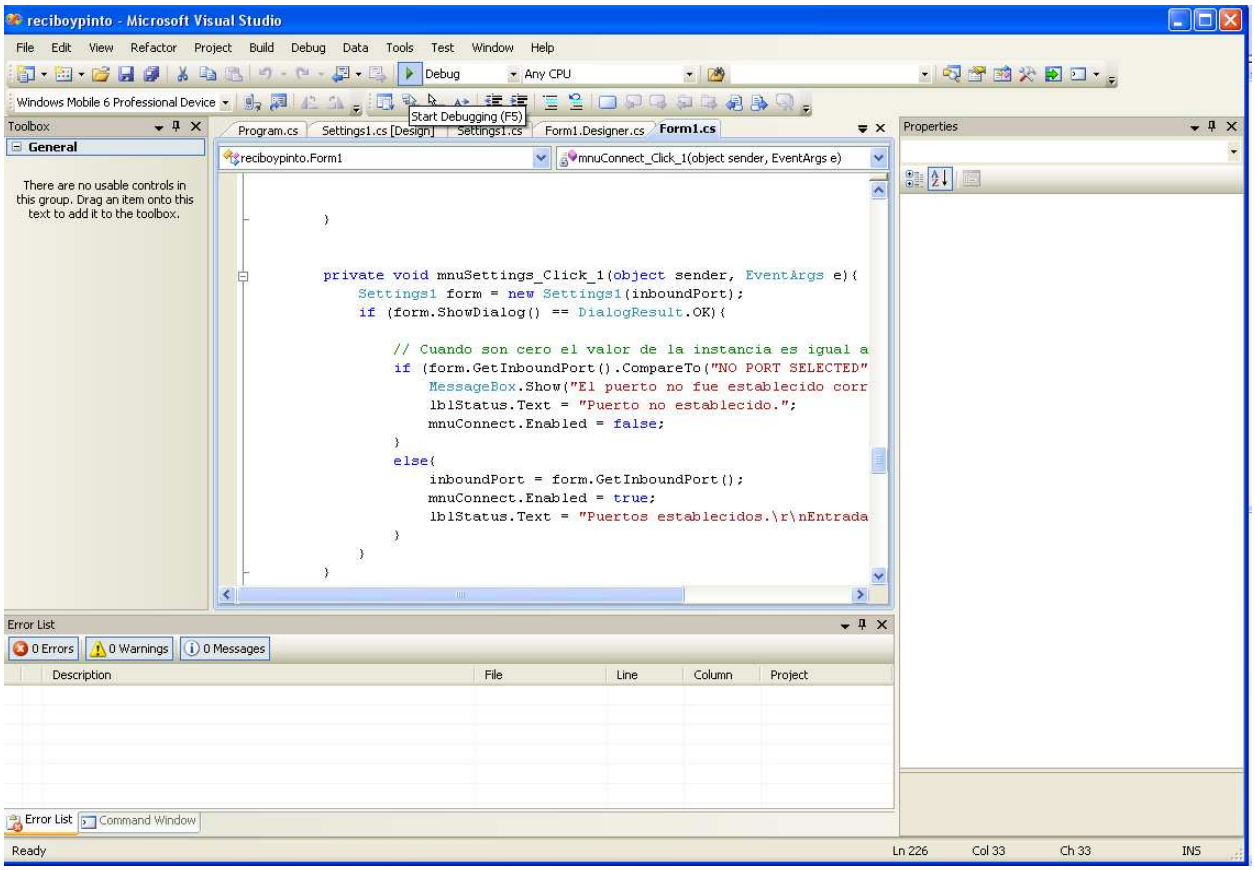



*Figura 5.6. Vista del cuarto paso para generar una aplicación*

5. Finalmente, ya comprobado que nuestra aplicación funciona correctamente no habrá que hacer ninguna operación más porque al hacer la depuración, se copia en la PDA un ejecutable de nuestro programa.

## 5.3 DIAGRAMAS DE FLUJO

El programa se divide en dos formularios, uno es el formulario principal *(Form1)* donde vemos la representación de los datos, y el otro formulario *(Settings1)* lo utilizamos para la elección del puerto para realizar la conexión. A continuación se mostrarán los diagramas de flujo de todo el programa:

### 5.3.1 Diagramas de flujo de *Form1*

- Evento tocar botón conectar:

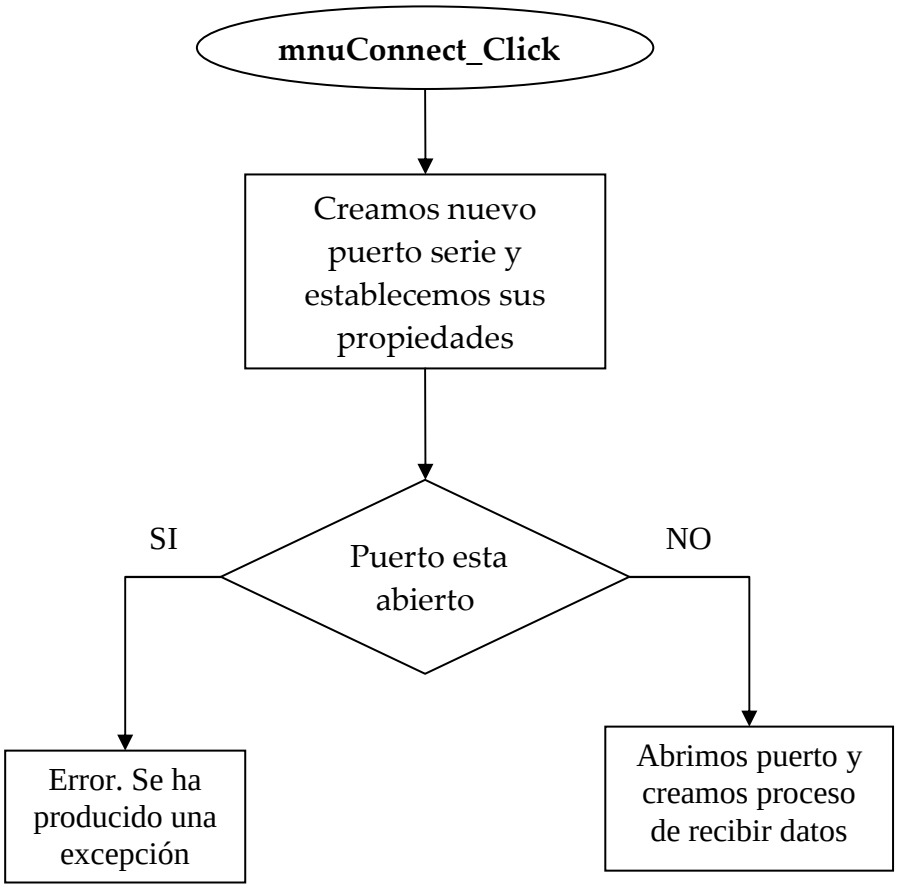



*Figura 5.7. Diagrama de flujo del evento tocar botón "Conectar"*

- Evento tocar botón desconectar:

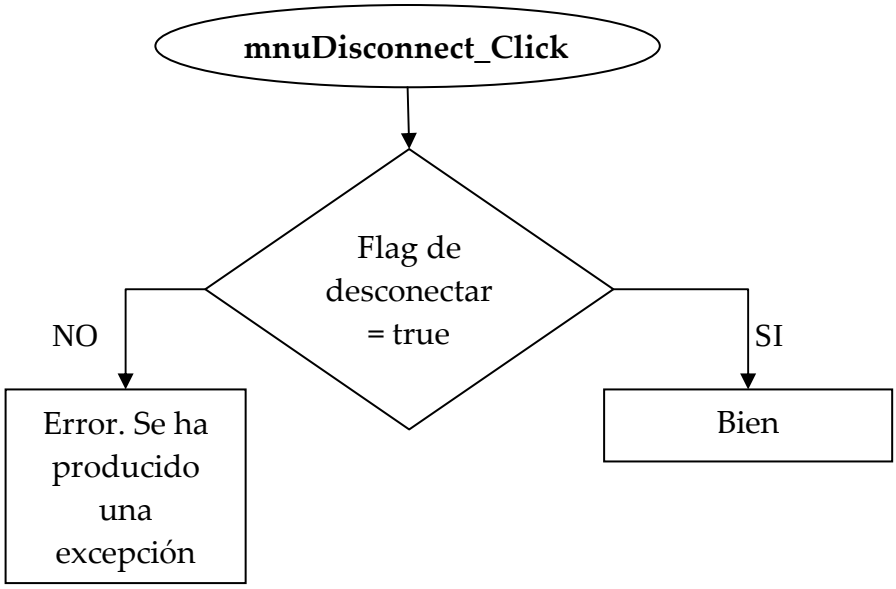



*Figura 5.8. Diagrama de flujo del evento tocar botón "Desconectar"*

- Evento tocar botón propiedades:

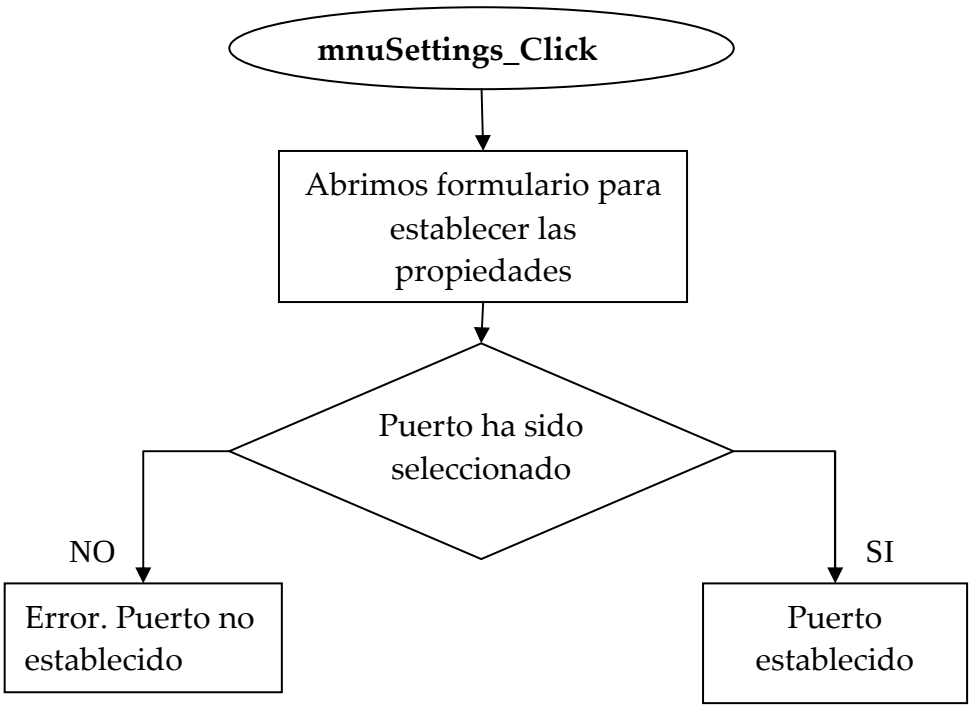



*Figura 5.9. Diagrama de flujo del evento tocar botón "Propiedades"*

- Hilo de recibir datos:

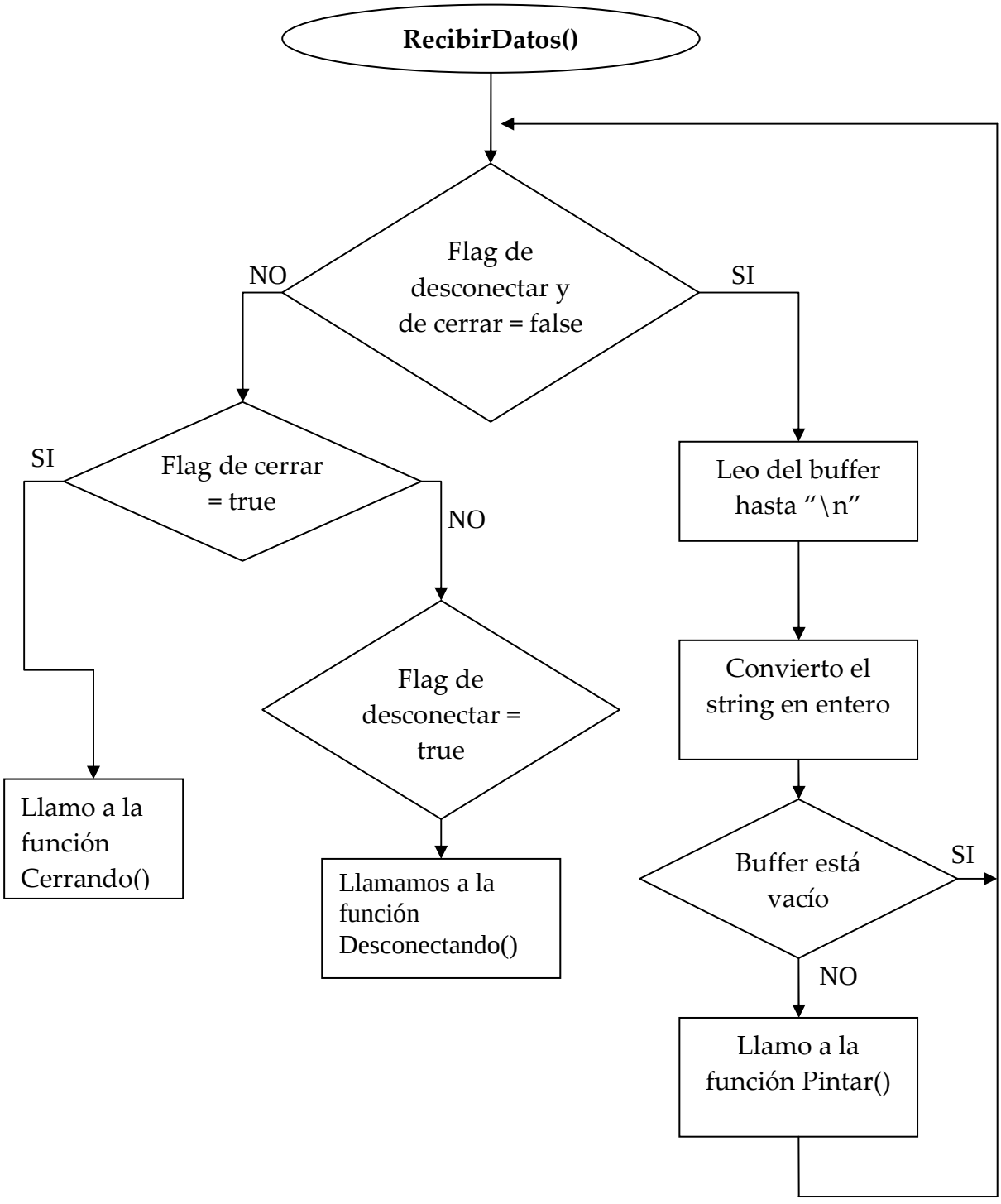



*Figura 5.10. Diagrama de flujo del hilo RecibirDatos()*

- Función Desconectando():

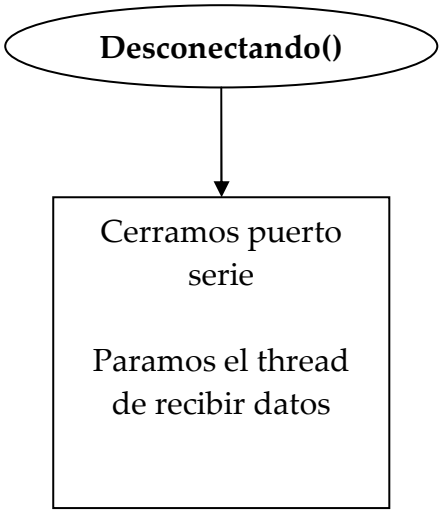



*Figura 5.11. Diagrama de flujo de la función Desconectando()*

- Función Cerrando():

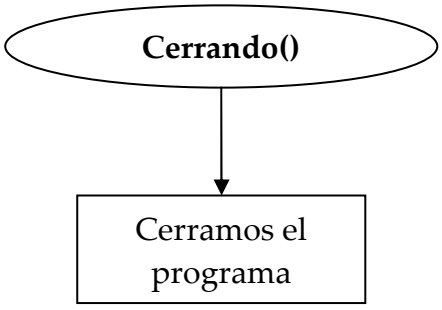



*Figura 5.12. Diagrama de flujo de la función Cerrando ()*

- Función Pintar():

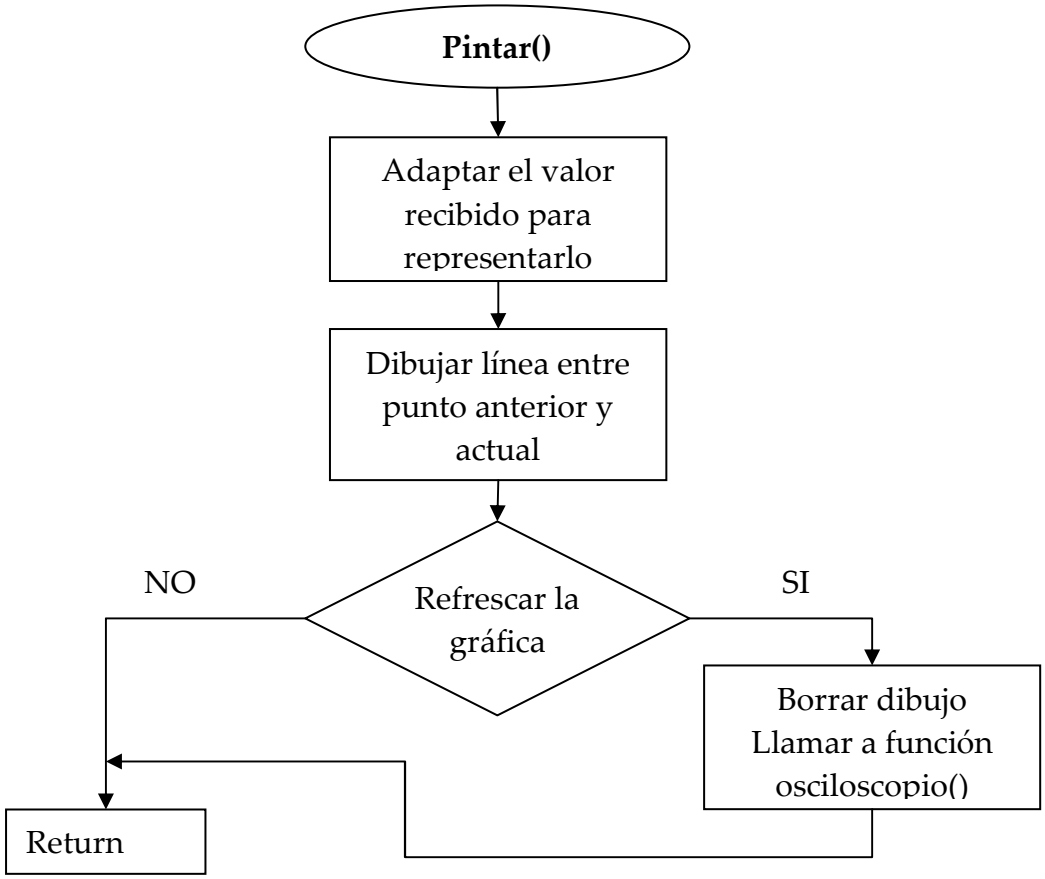



*Figura 5.13. Diagrama de flujo de la función Pintar()*

- Función osciloscopio:

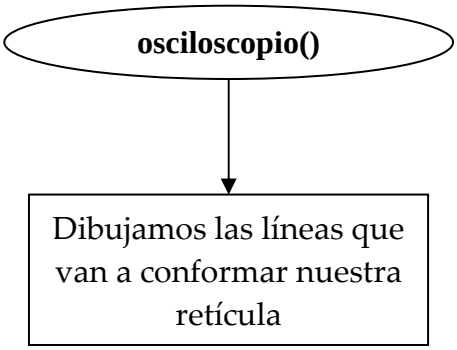



*Figura 5.14. Diagrama de flujo de la función osciloscopio()*

- Evento tocar botón salir:

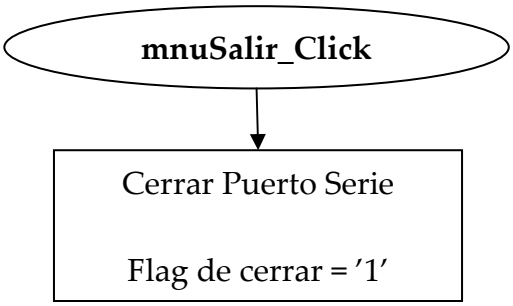



*Figura 5.15. Diagrama de flujo del evento tocar botón "Salir"*

5.3.2 Diagramas de flujo de *Settings1*

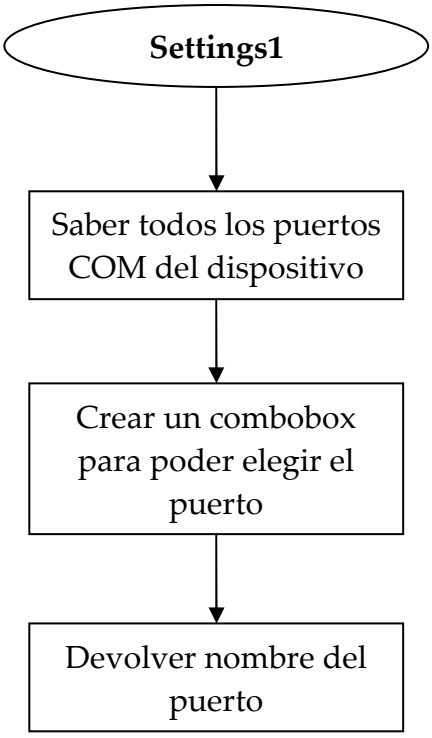



*Figura 5.16. Diagrama de flujo del formulario Settings1*

## 5.4 DESCRIPCIÓN DE LOS MÓDULOS

### 5.4.1 Función Pintar()

Esta función la usamos para representar los datos en forma de gráfica que nos llegan por el puerto serie.

Primeramente tenemos que adaptar los datos para representarlos correctamente en la pantalla, nótese que los datos que nos llegan son números positivos entre 0 y 255, pero el origen de coordenadas en la PDA esta arriba a la izquierda, por lo que hay que realizar una serie de operaciones, para que la representación sea correcta.

A continuación tenemos que crear un objeto "Pen" que nos permitirá dibujar las líneas y curvas. Seguidamente pintamos la línea entre el nuevo punto y el anterior.

Finalmente tenemos un if() para saber si debemos refrescar la gráfica, en el que borramos todo el dibujo y llamamos a la función osciloscopio para dibujar de nuevo la retícula.

### 5.4.2 Función osciloscopio()

Esta es una función muy simple que la utilizamos únicamente para dibujar "el osciloscopio" sobre el que se van a representar los datos.
Para ello tenemos que seguir los primeros pasos de la función Pintar() y después ya solamente tenemos que dibujar las líneas que conforman nuestra retícula.

### 5.4.3 Función Desconectando()

Esta función se encarga de cerrar el puerto serie y dejar al programa para poder realizar una nueva conexión.

### 5.4.4 Función Cerrando()

Esta función se encarga de cerrar el programa.

### 5.4.5 Evento mnuSettings_Click

El botón de propiedades nos sirve para establecer el puerto de la PDA con el que nos vamos a comunicar. Cuando el usuario toca el botón "Propiedades" salta este evento. Lo primero que hacemos es crear una instancia del formulario SettingsForm y nos lo muestra. A partir de ahí el usuario elegirá un puerto. A continuación, la función evalúa si el puerto se ha establecido correctamente mediante la función "string.compareTo(string)" de la librería .NET compact Framework.

Si el puerto no se ha establecido correctamente, saldrá una caja de texto que nos lo indicará y deberemos repetir el proceso, mientras que si se ha establecido correctamente tendremos la opción de conectarnos con algún dispositivo.

### 5.4.6 Evento mnuConnect_Click

El botón de conectar nos sirve para conectarnos al dispositivo con el que queremos realizar la conexión serie. Para ello lo primero que hacemos es instanciar el puerto serie y establecer sus propiedades. A continuación comprobamos que el puerto COM que hemos elegido no esta abierto, en ese caso abrimos el puerto y creamos un nuevo hilo para la recepción de datos. En caso de que el puerto COM este siendo usado por otro programa, saltará un mensaje que nos lo notificará.

### 5.4.7 Evento mnuDisconnect_Click

El botón de desconectar lo usamos cuando queremos interrumpir la conexión serie y dejar al programa en un estado de parada. Lo único que hacemos es darle un flag valor lógico '1', para que la función RecibirDatos() tome las medidas oportunas.

### 5.4.8 Evento mnuSalir_Click

El botón de salir, lo usamos cuando queremos cerrar el programa, para ello cerramos el puerto y ponemos un flag valor lógico '1', para que la función RecibirDatos() tome las medidas oportunas.

### 5.4.8 Función RecibirDatos()

Esta función es la que se encarga de recibir los datos del puerto serie. Esta formada por un bucle con un while(), que la única forma de pararlo es cuando el usuario cierra el programa o desconecta la conexión. Para leer los datos del puerto serie usamos la función "SerialPort.ReadTo(string)", esta función esta dentro de la librería .NET compact Framework, y su cometido es leer del buffer hasta que encuentra un carácter en nuestro caso "\n".

Debido a que lo que leemos es un string lo convertimos a entero mediante la función "Convert.ToInt16(string)" para que sea más sencilla la representación de los datos. A continuación, tenemos un if() para saber si el buffer estaba vacío. En el caso de que tuviéramos un dato llamamos a la función Pintar().

Finalmente comprobamos si el usuario quiere cerrar el programa o desconectar, para ello evaluamos dos flags.

### 5.4.9 Formulario de propiedades

En este formulario es donde se elige el puerto con el que se va a realizar la comunicación. Para ello se deben saber los puertos que tiene nuestro dispositivo, para ello se utiliza la función "SerialPort.GetPortNames()" que devuelve los nombres de los puertos. A continuación, añadimos los nombres de los puertos y un string ("PUERTO NO SELECCIONADO") creado por nosotros para poder saber si hemos elegido algún puerto. Por último tenemos que devolver el nombre del puerto.

## 5.5 LISTADO COMENTADO DEL PROGRAMA

### 5.5.1. Listado de *Form1*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;

namespace reciboypinto
{
    public partial class Form1 : Form{

        SerialPort serialP;

        Thread rcvThread;


        private delegate void closeDel();
        private delegate void disconnectDel();
        private string serial_Port = "";



        bool closeRequested = false;
        bool disconnectRequested = false;

        int x1 = 20;
        int y1 = 180;
        int x2 = 0;
        int valor = 0;

        double numero;







        public Form1(){

            InitializeComponent();

            }

        private void RecibirDatos(){
            while (!closeRequested && !disconnectRequested){
                try{
```

```
// Guardamos en un string los caracteres recibidos hasta encontrar "\n"
                    string line = serialP.ReadTo("\n");

                     //Convertimos el string a entero
                    valor = Convert.ToInt16(line);


                     if ((line !=null)){

                        Pintar(valor);



                     }
                     else { }


                }
                catch
                {

                }


            }
            if (closeRequested)
                Cerrando();
            if (disconnectRequested)
                this.Invoke(new disconnectDel(Desconectando));
        }


        private void Desconectando(){
            serialP.Close();

            mnuDisconnect.Enabled = false;
            mnuConnect.Enabled = false;
            menu.Enabled = true;
            lblStatus.Text = "Desconectado.\r\n.";
        }

        private void Cerrando(){

            this.Invoke(new closeDel(this.Close));
        }


        protected void Pintar(int valor){

            Graphics f = this.CreateGraphics();


            // Adaptamos el dato recibido para representarlo correctamente
            valor=valor-255;
            valor=-valor;
            numero = Convert.ToDouble(valor);
            numero = numero /(1.7);
            numero = numero + 30;
```

```
            valor = Convert.ToInt16(numero);



            Pen redPen = new Pen(Color.Red, 2);

            x2=x1+2;

            //Dibujamos una linea entre el punto anterior y el actual
            f.DrawLine(redPen, x1, y1, x2, valor);

            x1 = x2;
            y1 = valor;

            // Si la gráfica se ha llenado la refrescamos

            if (x1 == 220)
            {
                    x1 = 20;
                    f.Clear(Color.White);
                    osciloscopio();

            }




        }
public void osciloscopio()
        {
            Graphics g = this.CreateGraphics();
            Pen pn = new Pen(Color.Black);
            Pen pn1 = new Pen(Color.RosyBrown);
            Pen pn2 = new Pen(Color.Blue);

            // Pinto el rectancgulo de fuera
            g.DrawLine(pn, 20, 30, 20, 180);
            g.DrawLine(pn, 20, 180, 220, 180);
            g.DrawLine(pn, 220, 180, 220, 30);
            g.DrawLine(pn, 220, 30, 20, 30);

            //Pinto lineas de dentro

            g.DrawLine(pn1, 45, 30, 45, 180);
            g.DrawLine(pn1, 70, 30, 70, 180);
            g.DrawLine(pn1, 95, 30, 95, 180);
            g.DrawLine(pn, 120, 30, 120, 180);
            g.DrawLine(pn1, 145, 30, 145, 180);
            g.DrawLine(pn1, 170, 30, 170, 180);
            g.DrawLine(pn1, 195, 30, 195, 180);


            g.DrawLine(pn1, 20, 55, 220, 55);
            g.DrawLine(pn1, 20, 80, 220, 80);
            g.DrawLine(pn, 20, 105, 220, 105);
            g.DrawLine(pn1, 20, 130, 220, 130);
            g.DrawLine(pn1, 20, 155, 220, 155);
        }
```

```
            private void mnuSettings_Click_1(object sender, EventArgs e){
            Settings1 form = new Settings1(serial_Port);
            if (form.ShowDialog() == DialogResult.OK){

                // Con la siguiente función comparamos el string devuelto
con “PUERTO NO SELECCIONADO” , si el resultado es 0 es que son iguales
                if (form.Getserial_Port().CompareTo(“PUERTO NO
SELECCIONADO”) == 0){
                    MessageBox.Show(“El puerto no fue establecido
correctamente.”);
                    lblStatus.Text = “Puerto no establecido.”;
                    mnuConnect.Enabled = false;
                }
                else{
                    serial_Port = form.GetInboundPort();
                    mnuConnect.Enabled = true;
                    lblStatus.Text = “Puertos establecido. “ + serial_Port;
                }
            }
        }

        private void mnuConnect_Click_1(object sender, EventArgs e){

            //Creamos nuevo puerto serie y establecemos propiedades
            serialP = new SerialPort(serial_Port);
            serialP.ReadTimeout = 1000;
            serialP.BaudRate = 19200;
            serialP.Parity = Parity.None;
            serialP.StopBits = StopBits.One;
            serialP.DataBits = 8;
            serialP.Handshake = Handshake.None;

            disconnectRequested = false;

            try{
                // Comprobamos que el puerto no este abierto y en ese caso
lo abrimos
                if (¡serialP.IsOpen){
                    lblStatus.Text = “Abriendo puerto de entrada…”;
                    serialP.Open();
                }


                lblStatus.Text = “Puerto abierto”;
                // Creamos un nuevo hilo para la recepción de los datos
                rcvThread = new Thread(new ThreadStart(RecibirDatos));


                lblStatus.Text = “Conectado. “ + serial_Port ;
                MessageBox.Show(“Conectado.”);
                mnuConnect.Enabled = false;
                menu.Enabled = true;
                mnuDisconnect.Enabled = true;

                // Activamos el hilo de recepción
                rcvThread.Start();

            }
```

```
            catch (Exception ex){
                MessageBox.Show(ex.Message);
                lblStatus.Text = "Error.\r\n.";
                mnuConnect.Enabled = false;

}
        }

        private void mnuDisconnect_Click(object sender, EventArgs e){
            try{
                disconnectRequested = true;

            }
            catch (Exception ex){
                MessageBox.Show(ex.Message);
            }
        }

        private void mnuSalir_Click(object sender, EventArgs e)
        {
            SerialP.Close();
            closeRequested = true;
        }




    }
}

namespace reciboypinto
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Icontainer components = null;
        private System.Windows.Forms.MainMenu mainMenu1;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
```

```
        /// Required method for Designer support – do not modify
        /// the contents of this method with the code editor.
         /// </summary>
         private void InitializeComponent()
         {
              this.mainMenu1 = new System.Windows.Forms.MainMenu();

             this.menuSalir = new System.Windows.Forms.MenuItem();
             this.menu = new System.Windows.Forms.MenuItem();
             this.mnuConnect = new System.Windows.Forms.MenuItem();
             this.mnuDisconnect = new System.Windows.Forms.MenuItem();
             this.mnuSettings = new System.Windows.Forms.MenuItem();
             this.lblStatus = new System.Windows.Forms.Label();
             this.SuspendLayout();
             //
             // mainMenu1
             //
             this.mainMenu1.MenuItems.Add(this.menuSalir);
             this.mainMenu1.MenuItems.Add(this.menu);
             //
             // menuSalir
             //
             this.menuSalir.Text = "Salir";
             this.menuSalir.Click += new
System.EventHandler(this.menuSalir_Click_1);
             //
             // menu
             //
             this.menu.MenuItems.Add(this.mnuConnect);
             this.menu.MenuItems.Add(this.mnuDisconnect);
             this.menu.MenuItems.Add(this.mnuSettings);
             this.menu.Text = "Menu";
             //
             // mnuConnect
             //
             this.mnuConnect.Text = "Conectar";
             this.mnuConnect.Click += new
System.EventHandler(this.mnuConnect_Click_1);
             //
             // mnuDisconnect
             //
             this.mnuDisconnect.Text = "Desconectar";
             this.mnuDisconnect.Click += new
System.EventHandler(this.mnuDisconnect_Click);
             //
             // mnuSettings
             //
             this.mnuSettings.Text = "Propiedades";
             this.mnuSettings.Click += new
System.EventHandler(this.mnuSettings_Click_1);
             //
             // lblStatus
             //
             this.lblStatus.Location = new System.Drawing.Point(11, 228);
             this.lblStatus.Name = "lblStatus";
             this.lblStatus.Size = new System.Drawing.Size(179, 21);
             // Form1
```

```
            this.AutoScaleDimensions = new System.Drawing.SizeF(96F, 96F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi;
            this.AutoScroll = true;
            this.ClientSize = new System.Drawing.Size(240, 268);
            this.Controls.Add(this.lblStatus);
            this.Menu = this.mainMenu1;

            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.MenuItem menuSalir;
        private System.Windows.Forms.MenuItem menu;
        private System.Windows.Forms.MenuItem mnuConnect;
        private System.Windows.Forms.MenuItem mnuDisconnect;
        private System.Windows.Forms.MenuItem mnuSettings;
        private System.Windows.Forms.Label lblStatus;
    }
}
```

### 5.5.2 Listado de *Settings1*

```
using System.Linq;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace reciboypinto
{
   public partial class Settings1:Form
    {


        private string serial_Port;
        private Label label1;
        private ComboBox comboBox1;

        public Settings1(string serial)
        {
            InitializeComponent();
            serial_Port = serial;

            //Usamos la siguiente función para saber los nombres de los
puertos de nuestro dispositivo
```

```
            string[] ports = SerialPort.GetPortNames();

            comboBox1.Items.Add("PUERTO NO SELECCIONADO");

            //Añadimos a nuestro comboBox los nombres de los puertos
            for (int i = 0; i < ports.Length; i++)
                comboBox1.Items.Add(ports[i]);


            if (comboBox1.Items.Contains(serial_Port))
                comboBox1.SelectedItem = serial_Port;

            else
                // Si no elegimos ninguno entonces estará seleccionado
"PUERTO NO SELECCIONADO"
                comboBox1.SelectedIndex = 0;


        }
        public string Getserial_Port()
        {
            //Devolvemos el nombre del puerto
            return (string)comboBox1.SelectedItem;
        }


private void InitializeComponent()
       {
           this.comboBox1 = new System.Windows.Forms.ComboBox();
           this.label1 = new System.Windows.Forms.Label();
           this.SuspendLayout();
           //
           // comboBox1
           //
           this.comboBox1.Location = new System.Drawing.Point(31, 73);
           this.comboBox1.Name = "comboBox1";
           this.comboBox1.Size = new System.Drawing.Size(98, 22);
           this.comboBox1.TabIndex = 0;
           //
           // label1
           //
           this.label1.Location = new System.Drawing.Point(31, 28);
           this.label1.Name = "label1";
           this.label1.Size = new System.Drawing.Size(175, 19);
           this.label1.Text = "Elegir puerto de conexión :";
           //
           // Settings1
           //
           this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Inherit;
           this.ClientSize = new System.Drawing.Size(240, 294);
           this.Controls.Add(this.label1);
           this.Controls.Add(this.comboBox1);
           this.Name = "Settings1";
           this.ResumeLayout(false);

       }
```

# 6. CONSTRUCCIÓN Y MEDIDAS

# 6. CONSTRUCCIÓN Y MEDIDAS

## 6.1 DISEÑO DE LA PLACA

En la siguiente foto podemos observar nuestra placa y las partes del circuito que la componen.

1. Circuito de acondicionamiento.
2. Circuito de conversión TTL-RS232.
3. Microprocesador.

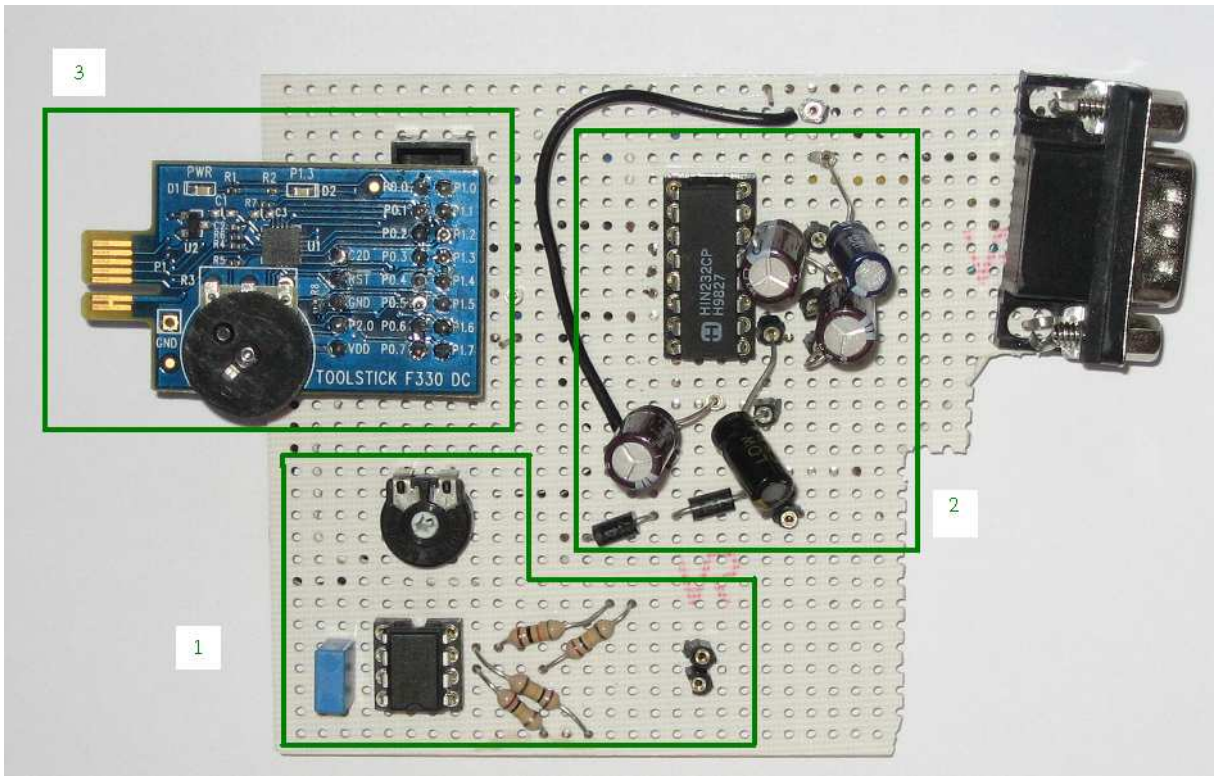



*Figura 6.1. Vista de la placa construida.*

También hemos realizado el diseño de la placa para crear un circuito impreso aunque finalmente no se ha llegado a construir:

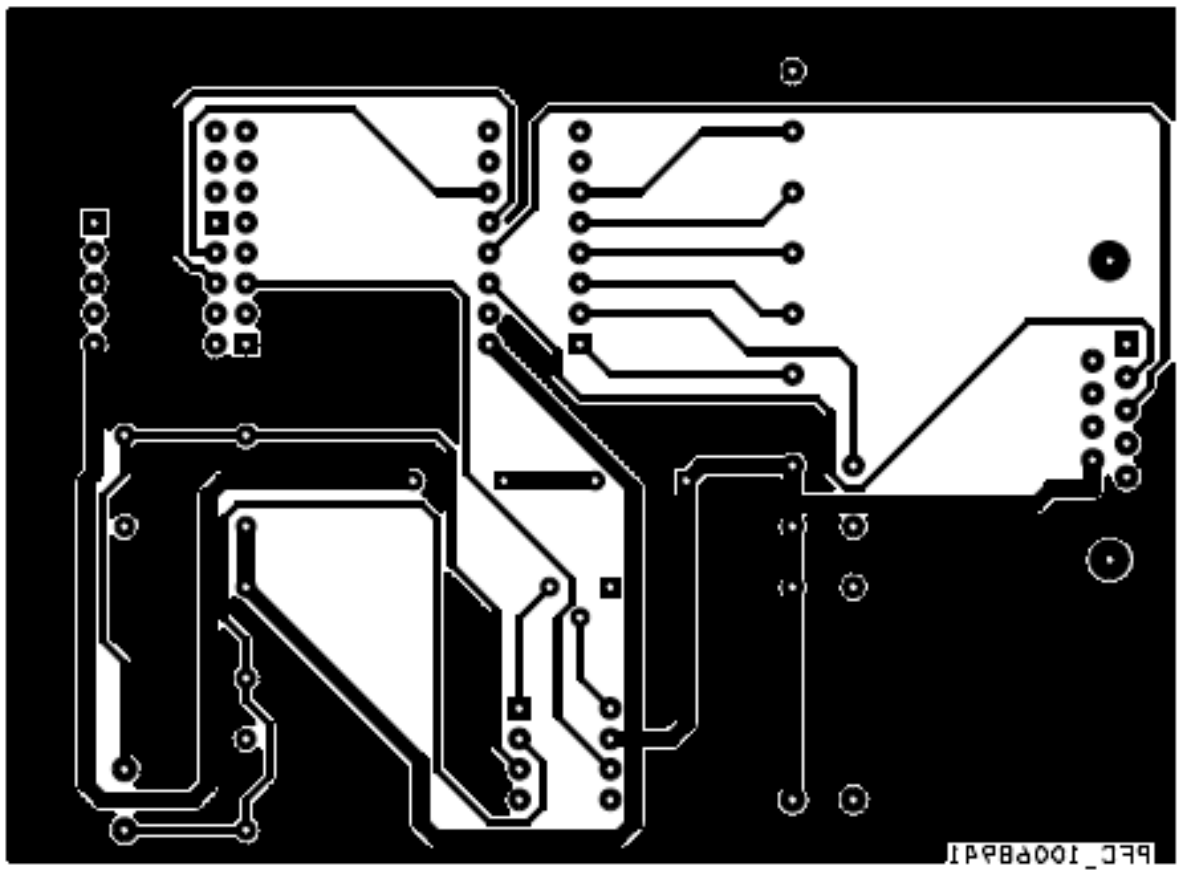


*Figura 6.2. Diseño de la placa utilizando Orcad Layout*

## 6.2 EL SISTEMA EN FUNCIONAMIENTO

Para comprobar el correcto funcionamiento de nuestro sistema necesitamos un generador de funciones que hará las veces de sensor analógico, un osciloscopio para ver las ondas del circuito y poder comprobar así que coinciden con lo esperado y por supuesto la placa de circuito impreso, el adaptador RS232-Bluetooth y nuestra PDA. A continuación se muestran la serie de pasos que hay que seguir para poner el sistema en funcionamiento:

1. Conectamos el adaptador RS232-Bluetooth a nuestra placa, a continuación la alimentamos y comprobamos que el adaptador este en perfecto estado según lo que nos dice el manual (se encienden los 3 LEDS intermitentemente).

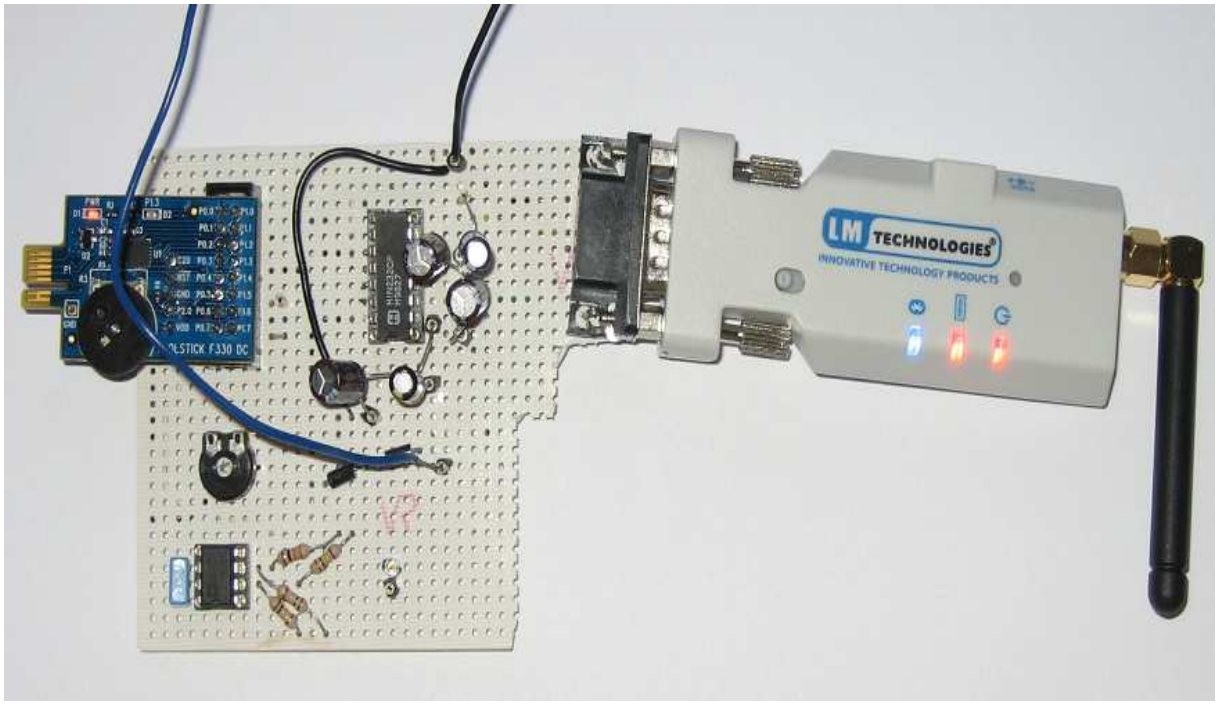



*Figura 6.3. Vista del primer paso para puesta en funcionamiento*

2. Encendemos nuestra PDA y abrimos el programa.

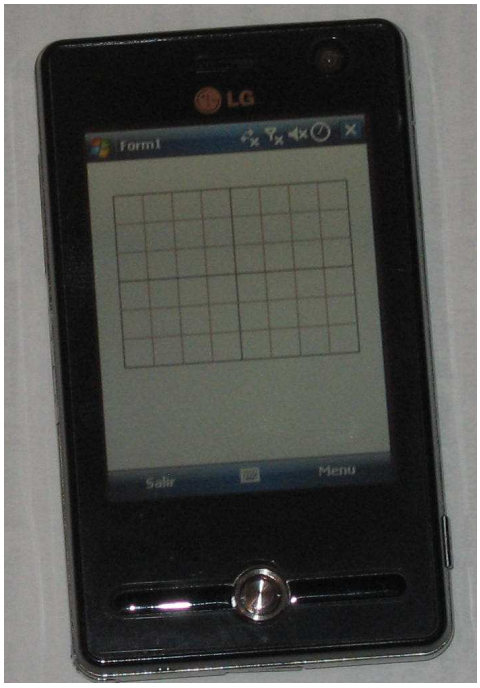



*Figura 6.4. Vista del segundo paso para puesta en funcionamiento*

3. Vamos al **Menú** y hacemos click en **Propiedades**.


*Figura 6.5. Vista del tercer paso para puesta en funcionamiento*

4. Nos saldrá una nueva pantalla en la que estableceremos el puerto de la PDA con el que nos vamos a comunicar (Nota: Es importante conocer cual es el numero del puerto de salida Bluetooth de la PDA ya que es el que debemos utilizar).

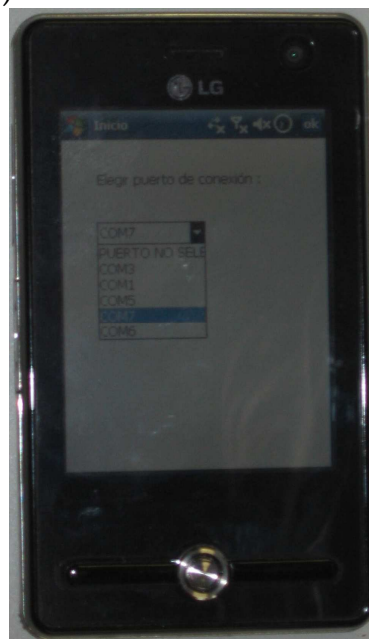



*Figura 6.6. Vista del cuarto paso para puesta en funcionamiento*

5. Una vez que hayamos establecido el puerto, daremos click a **Conectar** y nos saldrá una pantalla en el que aparecerán los dispositivos Bluetooth que estén a nuestro alrededor. En nuestro caso deberemos hacer click en Serial Adaptor y ya estaremos conectados.

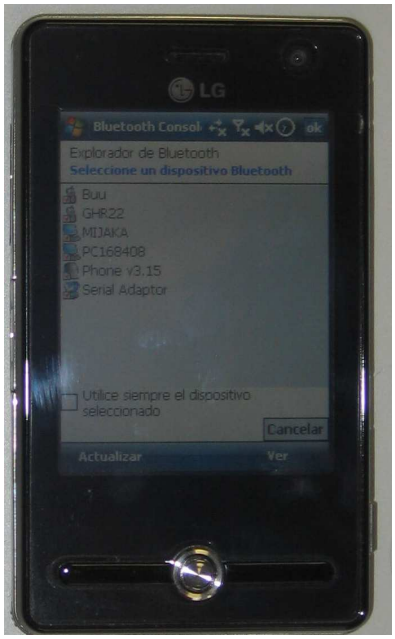



*Figura 6.7. Vista del quinto paso para puesta en funcionamiento*

6. Una vez que estemos conectados podremos salir del programa mediante el botón "**Salir**" o parar la conexión dando al botón **Desconectar**.

(Nota: puede ocurrir que el puerto que debemos usar este siendo utilizado por otro programa o que hemos seleccionado el puerto equivocado en ese caso se deberá ir al tercer paso).

A continuación mostramos las imágenes de la PDA, ante las distintas entradas del circuito:

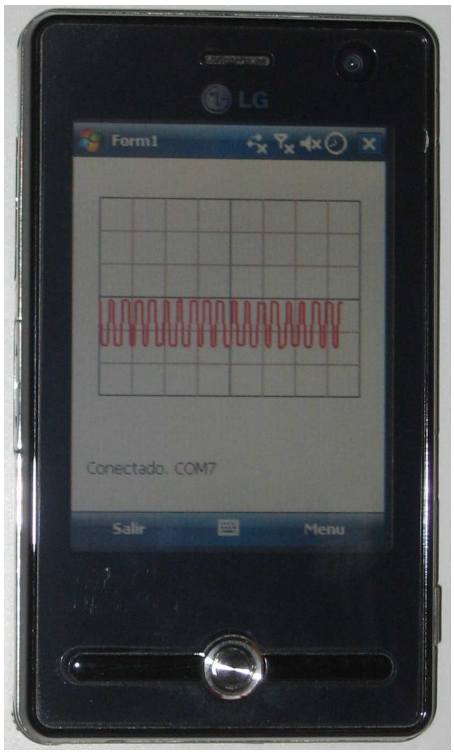



*Figura 6.8. Vista de la PDA ante entrada de onda cuadrada de 10 Hz.*

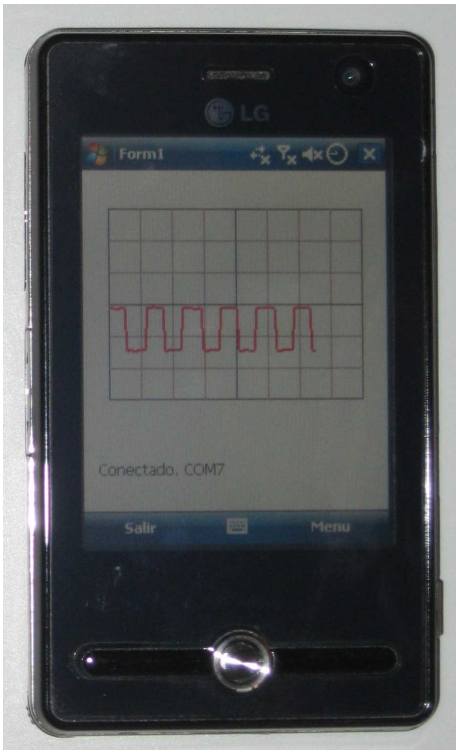



*Figura 6.9. Vista de la PDA ante entrada de onda cuadrada de 2 Hz.*

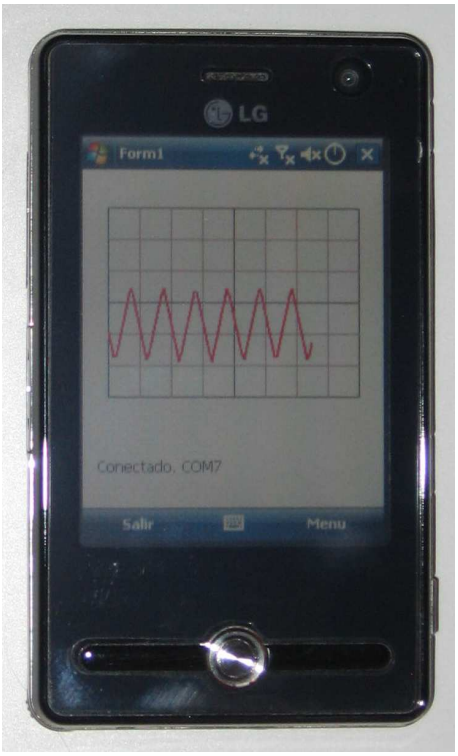



*Figura 6.10. Vista de la PDA ante entrada de onda triangular de 2 Hz.*

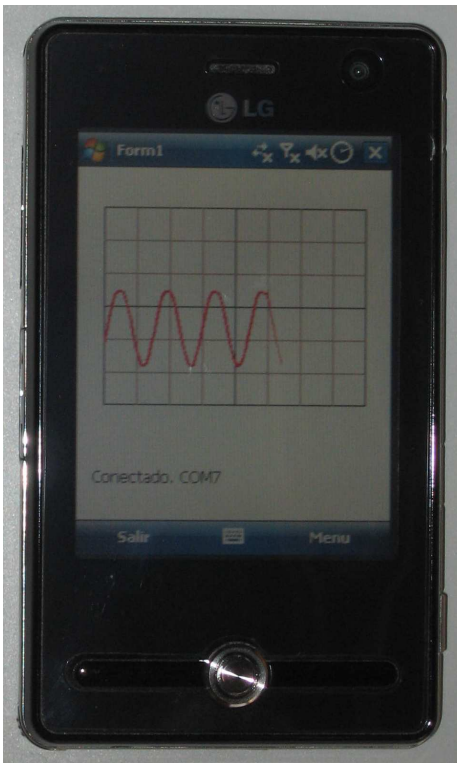



*Figura 6.11. Vista de la PDA ante entrada de onda senoidal de 2 Hz.*

# 7. CONCLUSIONES Y MEJORAS

# 7. CONCLUSIONES Y MEJORAS

A partir de los resultados obtenidos en las diversas pruebas podemos concluir que el funcionamiento del sistema es el deseado. No obstante es posible introducir algunas mejoras en el sistema. A continuación se presentan algunas ideas que se podrían llevar a cabo:

Sobre la aplicación de Windows Mobile en la PDA se podrían introducir una serie de mejoras para implementar su funcionalidad como la variación manual de la escala de tiempos y de la escala de voltaje. También se podrían implementar funciones para que se mostraran a petición del usuario distintas características de la señal representada como el valor máximo y mínimo de la señal, su frecuencia…

En cuanto a la aplicación sobre el microprocesador podríamos mejorarla subiendo la velocidad de transmisión lo que nos permitiría poder muestrear con una frecuencia superior y así tener la posibilidad de poder representar señales con mayor frecuencia.

# 8. PRESUPUESTO

# 8. PRESUPUESTO

En el presupuesto que se expone a continuación los costes totales del proyecto están desglosados en dos conceptos: coste de material y coste de personal.

Los costes de material son los costes asociados a la compra de los elementos necesarios para la construcción del diseño hardware. Los costes de personal son la parte más cuantiosa. Es el coste de la mano de obra.

COSTE DE MATERIAL

| Unidad | Descripción | Medición | Precio pu. | Total |
|---|---|---|---|---|
| Placa C.I. | Placa de puntos 100x100 | 1 | 4,21 | 4,21 |
| Toolstick F330 DC | Microcontrolador | 1 | 7,54 | 7,54 |
| MAX232 | Adaptador de niveles | 1 | 0,55 | 0,55 |
| Conectores | DB9, 8x2, 1x5 | 3 | 0,65 | 1,95 |
| Zócalos | 2x8,2x4 | 2 | 0,20 | 0,40 |
| AD620 | Amplificador de instrumentación | 1 | 0,40 | 0,40 |
| LM058 | Adaptador RS232-Bluetooth | 1 | 78,54 | 78,54 |
| LG KS-20 | PDA | 1 | 199,00 | 199,00 |
| R | Resistencias | 3 | 0,03 | 0,09 |
| D | Diodos | 2 | 0,10 | 0,20 |
| C | Condensadores | 6 | 0,05 | 0,30 |
| **TOTAL** | | | | **293,18** |

COSTE DE PERSONAL

| Concepto | Horas | Coste por hora (€) | Coste total (€) |
|---|---|---|---|
| Diseño y desarrollo del circuito | 60 | 15 | 900 |
| Implementación y pruebas | 100 | 15 | 1500 |
| Preparación del documento | 80 | 15 | 1200 |
| **TOTAL** | | | **3600** |

PRESUPUESTO TOTAL

| | |
|---|---|
| Coste de material | 293,18€ |
| Coste de personal | 3.600€ |
| **Coste total** | **3.893,18€** |

El presupuesto total es de:

*TRES MIL OCHOCIENTOS NOVENTA Y TRES COMA DIECIOCHO EUROS*

Madrid, a de de 2009
Germán Hernández Rodríguez
Ingeniero Técnico Industrial Electrónico

# 9. BIBLIOGRAFÍA

# 9. BIBLIOGRAFÍA

## BIBLIOGRAFÍA IMPRESA:


- Manual de prácticas de la asignatura *Laboratorio de Electrónica Industrial.* Tercero de ingeniería técnica industrial: electrónica industrial.

- Enciclopedia de Microsoft Visual C#**.** Ceballos Sierra, Francisco Javier


## BIBLIOGRAFÍA EN INTERNET:


- **http://msdn.microsoft.com/en-us/library/default.aspx**

# 10. ANEXOS

# 10. ANEXOS

En este capítulo se incluyen los planos de los circuitos empleados en el diseño de hardware, los listados de los programas del microprocesador y de la PDA, y las hojas de características de los componentes utilizados.

Debido a la gran cantidad de las hojas de catálogo de los componentes, se han incluido sólo las más primordiales de cada una de ellos.

Se puede encontrar información sobre los siguientes componentes:

- Microcontrolador PIC8051F330
- Adaptador de niveles MAX232
- Convertidor RS232-Bluetooth
- Diodo 1N4007
- LG KS-20
- ToolStick F330DC

# LISTADOS

## Listado de *Form1*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;

namespace reciboypinto
{
    public partial class Form1 : Form{

        SerialPort serialP;

        Thread rcvThread;


        private delegate void closeDel();
        private delegate void disconnectDel();
        private string serial_Port = "";



        bool closeRequested = false;
        bool disconnectRequested = false;

        int x1 = 20;
        int y1 = 180;
        int x2 = 0;
        int valor = 0;

        double numero;







        public Form1(){

            InitializeComponent();

            }

        private void RecibirDatos(){
            while (!closeRequested && !disconnectRequested){
                try{
```

```
                // Guardamos en un string los caracteres recibidos
                hasta encontrar "\n"
                string line = SerialP.ReadTo("\n");

                 //Convertimos el string a entero
                valor = Convert.ToInt16(line);


                 if ((line !=null)){

                    Pintar(valor);



                 }
                 else { }


            }
            catch
            {

            }


        }
        if (closeRequested)
            Cerrando();
        if (disconnectRequested)
            this.Invoke(new disconnectDel(Desconectando));
    }


    private void Desconectando(){
        SerialP.Close();

        mnuDisconnect.Enabled = false;
        mnuConnect.Enabled = false;
        menu.Enabled = true;
        lblStatus.Text = "Desconectado.\r\n.";
    }

    private void Cerrando(){

        this.Invoke(new closeDel(this.Close));
    }


    protected void Pintar(int valor){

        Graphics f = this.CreateGraphics();


        // Adaptamos el dato recibido para representarlo correctamente
        valor=valor-255;
        valor=-valor;
        numero = Convert.ToDouble(valor);
        numero = numero /(1.7);
```

```
            numero = numero + 30;
            valor = Convert.ToInt16(numero);
            Pen redPen = new Pen(Color.Red, 2);

            x2=x1+2;

            //Dibujamos una linea entre el punto anterior y el actual
            f.DrawLine(redPen, x1, y1, x2, valor);

            x1 = x2;
            y1 = valor;

            // Si la gráfica se ha llenado la refrescamos

            if (x1 == 220)
            {
                    x1 = 20;
                    f.Clear(Color.White);
                    osciloscopio();

            }



        }


        public void osciloscopio()
        {
            Graphics g = this.CreateGraphics();
            Pen pn = new Pen(Color.Black);
            Pen pn1 = new Pen(Color.RosyBrown);
            Pen pn2 = new Pen(Color.Blue);

            // Pinto el rectancgulo de fuera
            g.DrawLine(pn, 20, 30, 20, 180);
            g.DrawLine(pn, 20, 180, 220, 180);
            g.DrawLine(pn, 220, 180, 220, 30);
            g.DrawLine(pn, 220, 30, 20, 30);

            //Pinto lineas de dentro

            g.DrawLine(pn1, 45, 30, 45, 180);
            g.DrawLine(pn1, 70, 30, 70, 180);
            g.DrawLine(pn1, 95, 30, 95, 180);
            g.DrawLine(pn, 120, 30, 120, 180);
            g.DrawLine(pn1, 145, 30, 145, 180);
            g.DrawLine(pn1, 170, 30, 170, 180);
            g.DrawLine(pn1, 195, 30, 195, 180);


            g.DrawLine(pn1, 20, 55, 220, 55);
            g.DrawLine(pn1, 20, 80, 220, 80);
            g.DrawLine(pn, 20, 105, 220, 105);
            g.DrawLine(pn1, 20, 130, 220, 130);
            g.DrawLine(pn1, 20, 155, 220, 155);
        }
```

```
            private void mnuSettings_Click_1(object sender, EventArgs e){
            Settings1 form = new Settings1(serial_Port);
            if (form.ShowDialog() == DialogResult.OK){

                // Con la siguiente función comparamos el string devuelto
con “PUERTO NO SELECCIONADO” , si el resultado es 0 es que son iguales
                if (form.Getserial_Port().CompareTo(“PUERTO NO
SELECCIONADO”) == 0){
                    MessageBox.Show(“El puerto no fue establecido
correctamente.”);
                    lblStatus.Text = “Puerto no establecido.”;
                    mnuConnect.Enabled = false;
                }
                else{
                    serial_Port = form.GetInboundPort();
                    mnuConnect.Enabled = true;
                    lblStatus.Text = “Puertos establecido. “ + serial_Port;
                }
            }
        }

        private void mnuConnect_Click_1(object sender, EventArgs e){

            //Creamos nuevo puerto serie y establecemos propiedades
            serialP = new SerialPort(serial_Port);
            serialP.ReadTimeout = 1000;
            serialP.BaudRate = 19200;
            serialP.Parity = Parity.None;
            serialP.StopBits = StopBits.One;
            serialP.DataBits = 8;
            serialP.Handshake = Handshake.None;

            disconnectRequested = false;

            try{
                // Comprobamos que el puerto no este abierto y en ese caso
lo abrimos
                if (¡serialP.IsOpen){
                    lblStatus.Text = “Abriendo puerto de entrada…”;
                    serialP.Open();
                }


                lblStatus.Text = “Puerto abierto”;
                // Creamos un nuevo hilo para la recepción de los datos
                rcvThread = new Thread(new ThreadStart(RecibirDatos));


                lblStatus.Text = “Conectado. “ + serial_Port ;
                MessageBox.Show(“Conectado.”);
                mnuConnect.Enabled = false;
                menu.Enabled = true;
                mnuDisconnect.Enabled = true;

                // Activamos el hilo de recepción

            rcvThread.Start();
```

```
            }
            catch (Exception ex){

                MessageBox.Show(ex.Message);
                lblStatus.Text = "Error.\r\n.";
                mnuConnect.Enabled = false;

 }
        }

        private void mnuDisconnect_Click(object sender, EventArgs e){
            try{
                disconnectRequested = true;

            }
            catch (Exception ex){
                MessageBox.Show(ex.Message);
            }
        }

        private void mnuSalir_Click(object sender, EventArgs e)
        {
            SerialP.Close();
            closeRequested = true;
        }



    }
}

namespace reciboypinto
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Icontainer components = null;
        private System.Windows.Forms.MainMenu mainMenu1;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code
```

```
        /// <summary>
         /// Required method for Designer support – do not modify
         /// the contents of this method with the code editor.

       /// </summary>
         private void InitializeComponent()
         {
              this.mainMenu1 = new System.Windows.Forms.MainMenu();

               this.menuSalir = new System.Windows.Forms.MenuItem();
              this.menu = new System.Windows.Forms.MenuItem();
              this.mnuConnect = new System.Windows.Forms.MenuItem();
              this.mnuDisconnect = new System.Windows.Forms.MenuItem();
              this.mnuSettings = new System.Windows.Forms.MenuItem();
              this.lblStatus = new System.Windows.Forms.Label();
              this.SuspendLayout();
              //
              // mainMenu1
              //
              this.mainMenu1.MenuItems.Add(this.menuSalir);
              this.mainMenu1.MenuItems.Add(this.menu);
              //
              // menuSalir
              //
              this.menuSalir.Text = "Salir";
              this.menuSalir.Click += new
System.EventHandler(this.menuSalir_Click_1);
              //
              // menu
              //
              this.menu.MenuItems.Add(this.mnuConnect);
              this.menu.MenuItems.Add(this.mnuDisconnect);
              this.menu.MenuItems.Add(this.mnuSettings);
              this.menu.Text = "Menu";
              //
              // mnuConnect
              //
              this.mnuConnect.Text = "Conectar";
              this.mnuConnect.Click += new
System.EventHandler(this.mnuConnect_Click_1);
              //
              // mnuDisconnect
              //
              this.mnuDisconnect.Text = "Desconectar";
              this.mnuDisconnect.Click += new
System.EventHandler(this.mnuDisconnect_Click);
              //
              // mnuSettings
              //
              this.mnuSettings.Text = "Propiedades";
              this.mnuSettings.Click += new
System.EventHandler(this.mnuSettings_Click_1);
              //
              // lblStatus
              //
              this.lblStatus.Location = new System.Drawing.Point(11, 228);

             this.lblStatus.Name = "lblStatus";
```

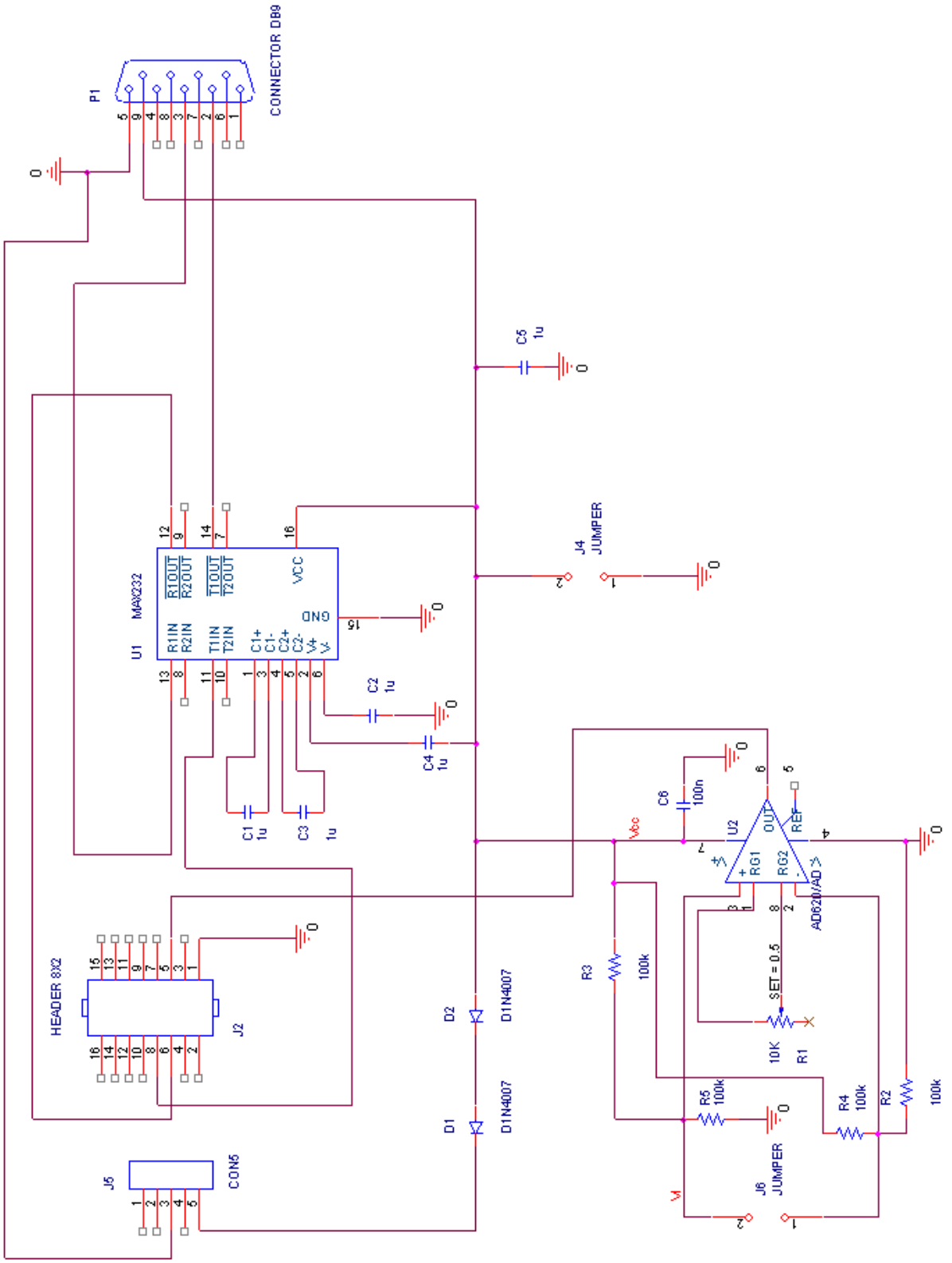
```
            this.lblStatus.Size = new System.Drawing.Size(179, 21);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(96F, 96F);

            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi;
            this.AutoScroll = true;
            this.ClientSize = new System.Drawing.Size(240, 268);
            this.Controls.Add(this.lblStatus);
            this.Menu = this.mainMenu1;

            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.MenuItem menuSalir;
        private System.Windows.Forms.MenuItem menu;
        private System.Windows.Forms.MenuItem mnuConnect;
        private System.Windows.Forms.MenuItem mnuDisconnect;
        private System.Windows.Forms.MenuItem mnuSettings;
        private System.Windows.Forms.Label lblStatus;
    }
}
```

Listado de *Settings1*

```
using System.Linq;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace reciboypinto
{
   public partial class Settings1:Form
    {


        private string serial_Port;
        private Label label1;
        private ComboBox comboBox1;

        public Settings1(string serial)

      {
            InitializeComponent();
```

```
            serial_Port = serial;
            //Usamos la siguiente función para saber los nombres de los
puertos de nuestro dispositivo
            string[] ports = SerialPort.GetPortNames();


            comboBox1.Items.Add("PUERTO NO SELECCIONADO");

            //Añadimos a nuestro comboBox los nombres de los puertos
            for (int i = 0; i < ports.Length; i++)
                comboBox1.Items.Add(ports[i]);


            if (comboBox1.Items.Contains(serial_Port))
                comboBox1.SelectedItem = serial_Port;

            else
                // Si no elegimos ninguno entonces estará seleccionado
"PUERTO NO SELECCIONADO"
                comboBox1.SelectedIndex = 0;


        }
        public string Getserial_Port()
        {
            //Devolvemos el nombre del puerto
            return (string)comboBox1.SelectedItem;
        }
private void InitializeComponent()
       {
           this.comboBox1 = new System.Windows.Forms.ComboBox();
           this.label1 = new System.Windows.Forms.Label();
           this.SuspendLayout();
           //
           // comboBox1
           //
           this.comboBox1.Location = new System.Drawing.Point(31, 73);
           this.comboBox1.Name = "comboBox1";
           this.comboBox1.Size = new System.Drawing.Size(98, 22);
           this.comboBox1.TabIndex = 0;
           //
           // label1
           //
           this.label1.Location = new System.Drawing.Point(31, 28);
           this.label1.Name = "label1";
           this.label1.Size = new System.Drawing.Size(175, 19);
           this.label1.Text = "Elegir puerto de conexión :";
           //
           // Settings1
           //
           this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Inherit;
           this.ClientSize = new System.Drawing.Size(240, 294);
           this.Controls.Add(this.label1);
           this.Controls.Add(this.comboBox1);
           this.Name = "Settings1";
           this.ResumeLayout(false);

       }
```

# ESQUEMAS


P1
CONNECTOR DB9
U1
MAX232
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
T1IN
T2IN
C1+
C1-
C2+
C2-
V+
V-
VCC
GND
C1 1u
C2 1u
C3 1u
C4 1u
C5 1u
C6 100n
J2
HEADER 8X2
J4
JUMPER
J5
CON5
J6
JUMPER
D1
D1N4007
D2
D1N4007
U2
AD620/AD
REF
OUT
RG1
RG2
R1
10K
SET = 0.5
R2
100k
R3
100k
R4
100k
R5
100k
0
Vcc
V-

ANALOG DEVICES

# Low Cost, Low Power Instrumentation Amplifier

## AD620

**FEATURES**
**EASY TO USE**
Gain Set with One External Resistor
(Gain Range 1 to 1000)
Wide Power Supply Range (±2.3 V to ±18 V)
Higher Performance than Three Op Amp IA Designs
Available in 8-Lead DIP and SOIC Packaging
Low Power, 1.3 mA max Supply Current

**EXCELLENT DC PERFORMANCE ("B GRADE")**
50 μV max, Input Offset Voltage
0.6 μV/°C max, Input Offset Drift
1.0 nA max, Input Bias Current
100 dB min Common-Mode Rejection Ratio (G = 10)

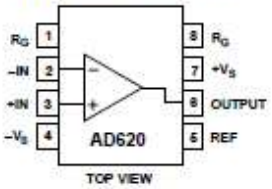
**LOW NOISE**
9 nV/√Hz, @ 1 kHz, Input Voltage Noise
0.28 μV p-p Noise (0.1 Hz to 10 Hz)

**EXCELLENT AC SPECIFICATIONS**
120 kHz Bandwidth (G = 100)
15 μs Settling Time to 0.01%

**APPLICATIONS**
Weigh Scales
ECG and Medical Instrumentation
Transducer Interface
Data Acquisition Systems
Industrial Process Controls
Battery Powered and Portable Equipment

**CONNECTION DIAGRAM**

**8-Lead Plastic Mini-DIP (N), Cerdip (Q) and SOIC (R) Packages**




**PRODUCT DESCRIPTION**

The AD620 is a low cost, high accuracy instrumentation amplifier that requires only one external resistor to set gains of 1 to 1000. Furthermore, the AD620 features 8-lead SOIC and DIP packaging that is smaller than discrete designs, and offers lower power (only 1.3 mA max supply current), making it a good fit for battery powered, portable (or remote) applications.

The AD620, with its high accuracy of 40 ppm maximum nonlinearity, low offset voltage of 50 μV max and offset drift of 0.6 μV/°C max, is ideal for use in precision data acquisition systems, such as weigh scales and transducer interfaces. Furthermore, the low noise, low input bias current, and low power of the AD620 make it well suited for medical applications such as ECG and noninvasive blood pressure monitors.

The low input bias current of 1.0 nA max is made possible with the use of Superβeta processing in the input stage. The AD620 works well as a preamplifier due to its low input voltage noise of 9 nV/√Hz at 1 kHz, 0.28 μV p-p in the 0.1 Hz to 10 Hz band, 0.1 pA/√Hz input current noise. Also, the AD620 is well suited for multiplexed applications with its settling time of 15 μs to 0.01% and its cost is low enough to enable designs with one in-amp per channel.

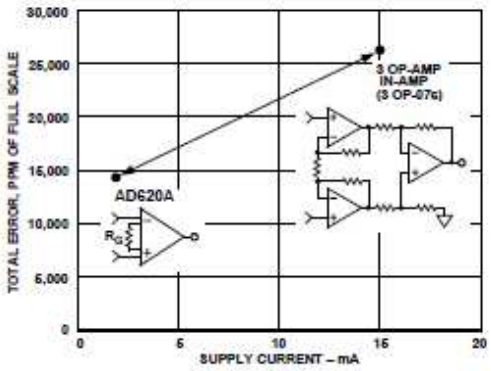



*Figure 1. Three Op Amp IA Designs vs. AD620*

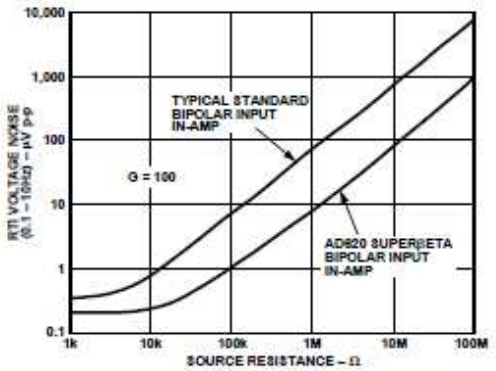



*Figure 2. Total Voltage Noise vs. Source Resistance*

REV. E



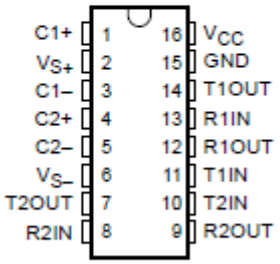
One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 World Wide Web Site: http://www.analog.com
Fax: 781/326-8703

# MAX232, MAX232I
# DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047I – FEBRUARY 1989 – REVISED OCTOBER 2002

- Meet or Exceed TIA/EIA-232-F and ITU Recommendation V.28
- Operate With Single 5-V Power Supply
- Operate Up to 120 kbit/s
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- Designed to be Interchangeable With Maxim MAX232
- ESD Protection Exceeds JESD 22 – 2000-V Human-Body Model (A114-A)
- Applications
  - TIA/EIA-232-F
  - Battery-Powered Systems
  - Terminals
  - Modems
  - Computers

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)




## description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

| $T_A$ | PACKAGE† | | ORDERABLE PART NUMBER | TOP-SIDE MARKING |
|---|---|---|---|---|
| 0°C to 70°C | PDIP (N) | Tube | MAX232N | MAX232N |
| | SOIC (D) | Tube | MAX232D | MAX232 |
| | | Tape and reel | MAX232DR | |
| | SOIC (DW) | Tube | MAX232DW | MAX232 |
| | | Tape and reel | MAX232DWR | |
| | SOP (NS) | Tape and reel | MAX232NSR | MAX232 |
| –40°C to 85°C | PDIP (N) | Tube | MAX232IN | MAX232IN |
| | SOIC (D) | Tube | MAX232ID | MAX232I |
| | | Tape and reel | MAX232IDR | |
| | SOIC (DW) | Tube | MAX232IDW | MAX232I |
| | | Tape and reel | MAX232IDWR | |

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

SILICON LABORATORIES

# C8051F330/1/2/3/4/5

## Mixed Signal ISP Flash MCU Family

### Analog Peripherals
- **10-Bit ADC ('F330/2/4 only)**
  - Up to 200 ksps
  - Up to 16 external single-ended or differential inputs
  - VREF from internal VREF, external pin or $V_{DD}$
  - Internal or external start of conversion source
  - Built-in temperature sensor
- **10-Bit Current Output DAC ('F330 only)**
- **Comparator**
  - Programmable hysteresis and response time
  - Configurable as interrupt or reset source
  - Low current (0.4 µA)

### On-Chip Debug
- On-chip debug circuitry facilitates full speed, non-intrusive in-system debug (no emulator required)
- Provides breakpoints, single stepping, inspect/modify memory and registers
- Superior performance to emulation systems using ICE-chips, target pods, and sockets
- Low cost, **complete** development kit

### Supply Voltage 2.7 to 3.6 V
- Typical operating current: 6.4 mA at 25 MHz; 9 µA at 32 kHz
- Typical stop mode current: 0.1 µA

### Temperature Range: –40 to +85 °C

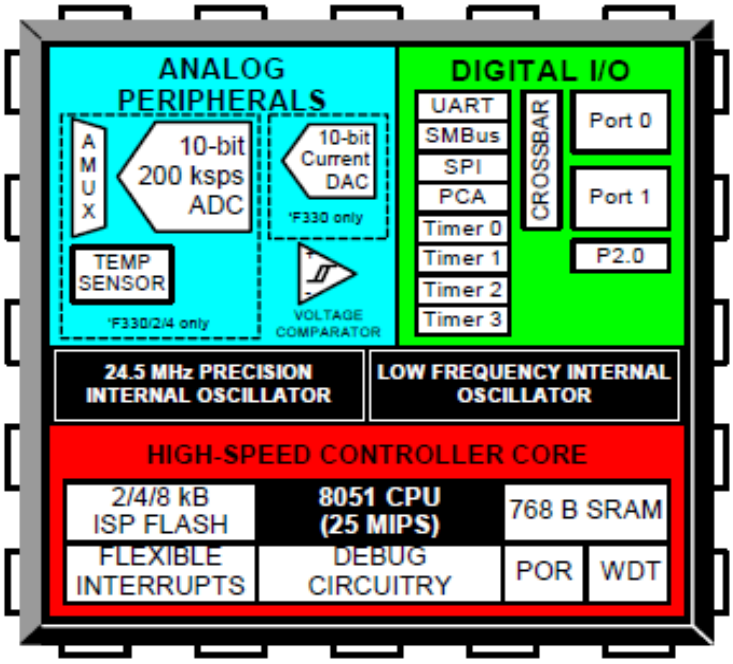
### High Speed 8051 µC Core
- Pipelined instruction architecture; executes 70% of instructions in 1 or 2 system clocks
- Up to 25 MIPS throughput with 25 MHz clock
- Expanded interrupt handler

### Memory
- 768 bytes internal data RAM (256 + 512)
- 8 kB ('F330/1), 4 kB ('F332/3), or 2 kB ('F334/5) Flash; In-system programmable in 512-byte Sectors—512 bytes are reserved in the 8 kB devices

### Digital Peripherals
- 17 Port I/O; All 5 V tolerant with high sink current
- Hardware enhanced UART, SMBus™, and enhanced SPI™ serial ports
- Four general purpose 16-bit counter/timers
- 16-Bit programmable counter array (PCA) with three capture/compare modules
- Real time clock mode using PCA or timer and external clock source

### Clock Sources
- Two internal oscillators:
  - 24.5 MHz with ±2% accuracy supports crystal-less UART operation
  - 80/40/20/10 kHz low frequency, low power
- External oscillator: Crystal, RC, C, or clock (1 or 2 pin modes)
- Can switch between clock sources on-the-fly; useful in power saving modes

### 20-Pin QFN or 20-pin PDIP



**MOTOROLA**
**SEMICONDUCTOR TECHNICAL DATA**

Order this document
by 1N4001/D

# Axial Lead Standard Recovery Rectifiers

This data sheet provides information on subminiature size, axial lead mounted rectifiers for general–purpose low–power applications.

**Mechanical Characteristics**

- Case: Epoxy, Molded
- Weight: 0.4 gram (approximately)
- Finish: All External Surfaces Corrosion Resistant and Terminal Leads are Readily Solderable
- Lead and Mounting Surface Temperature for Soldering Purposes: 220°C Max. for 10 Seconds, 1/16" from case
- Shipped in plastic bags, 1000 per bag.
- Available Tape and Reeled, 5000 per reel, by adding a "RL" suffix to the part number
- Polarity: Cathode Indicated by Polarity Band
- Marking: 1N4001, 1N4002, 1N4003, 1N4004, 1N4005, 1N4006, 1N4007

**1N4001 thru 1N4007**

1N4004 and 1N4007 are Motorola Preferred Devices

**LEAD MOUNTED RECTIFIERS 50–1000 VOLTS DIFFUSED JUNCTION**

CASE 59–03
DO–41

**MAXIMUM RATINGS**

| Rating | Symbol | 1N4001 | 1N4002 | 1N4003 | 1N4004 | 1N4005 | 1N4006 | 1N4007 | Unit |
|---|---|---|---|---|---|---|---|---|---|
| *Peak Repetitive Reverse Voltage<br>Working Peak Reverse Voltage<br>DC Blocking Voltage | $V_{RRM}$<br>$V_{RWM}$<br>$V_R$ | 50 | 100 | 200 | 400 | 600 | 800 | 1000 | Volts |
| *Non–Repetitive Peak Reverse Voltage (halfwave, single phase, 60 Hz) | $V_{RSM}$ | 60 | 120 | 240 | 480 | 720 | 1000 | 1200 | Volts |
| *RMS Reverse Voltage | $V_{R(RMS)}$ | 35 | 70 | 140 | 280 | 420 | 560 | 700 | Volts |
| *Average Rectified Forward Current (single phase, resistive load, 60 Hz, see Figure 8, $T_A$ = 75°C) | $I_O$ | 1.0 | | | | | | | Amp |
| *Non–Repetitive Peak Surge Current (surge applied at rated load conditions, see Figure 2) | $I_{FSM}$ | 30 (for 1 cycle) | | | | | | | Amp |
| Operating and Storage Junction Temperature Range | $T_J$<br>$T_{stg}$ | – 65 to +175 | | | | | | | °C |

**ELECTRICAL CHARACTERISTICS***

| Rating | Symbol | Typ | Max | Unit |
|---|---|---|---|---|
| Maximum Instantaneous Forward Voltage Drop ($i_F$ = 1.0 Amp, $T_J$ = 25°C) Figure 1 | $v_F$ | 0.93 | 1.1 | Volts |
| Maximum Full–Cycle Average Forward Voltage Drop ($I_O$ = 1.0 Amp, $T_L$ = 75°C, 1 inch leads) | $V_{F(AV)}$ | — | 0.8 | Volts |
| Maximum Reverse Current (rated dc voltage)<br>($T_J$ = 25°C)<br>($T_J$ = 100°C) | $I_R$ | <br>0.05<br>1.0 | <br>10<br>50 | µA |
| Maximum Full–Cycle Average Reverse Current ($I_O$ = 1.0 Amp, $T_L$ = 75°C, 1 inch leads) | $I_{R(AV)}$ | — | 30 | µA |

*Indicates JEDEC Registered Data

**Preferred** devices are Motorola recommended choices for future use and best overall value.

Rev 5



MOTOROLA

LM TECHNOLOGIES®
INNOVATIVE TECHNOLOGY PRODUCTS

# RS232- Bluetooth Adapter

Part Number: LM-058

## Features

- Constent updating of Firmware for global interpolority and compatibility
- Multiple power options, 9th Pin, AC mains PSU and USB power cable
- CSR BC04 EDR 2.0+Chipset
- Windows configuration software
- PSU and gender changers supplied
- Growing and improving AT command set
- Approved product by some of the worlds leading datalogger manufactures
- 600 Meter range when supplied with the LM057 600 Meter antenna
- Firmware upgradeable

## Technical Specification

| | |
|---|---|
| **Product Name** | RS232 Bluetooth Serial Adapter |
| **Chipset** | CSR BC4 EDR |
| **Bluetooth Rate** | 3Mbps |
| **Bluetooth Profiles** | SPP and GAP |
| **Baud Rate** | 4.8/9.6/19.2/38.4/57.6/115.2/230.4/460.8kbps |
| **Transmission Class** | Class 1 - 100 Meters |
| **Spread Spectrum** | 2.4000 Ghz - 2.4835 Ghz |
| **Modulation Method** | ModulationGFSK-1 Mbps, DQPSK-2 Mbps, and 8-DPSK-3 mBPS |
| **Transmitter Power** | 18 dBm |
| **Receive Sensitivity** | -86 dBm |
| **Operating Range** | 100 Meters - Open Space |
| **Operating Temperature** | 20°C to +75°C |
| **Signal** | TxD, RxD, GND, CTS, and RTSRS-232 |
| **Interface** | D_SUB 9-pin female |
| **Connections** | Mini USB and Female SMA |
| **Weight** | 57 Grams |
| **Dimensions** | 35 mm (W) x 65 mm (D) x 16mm |
| **Certificates** | CE, FCC, RoHS and BQB |

## Contact

LM Technologies Ltd
Radclyffe House
66-68 Hagley Road
Birmingham
B16 8PF, UK.

Tel : 0870 066 2740
Fax: 0871 431 0192

# ToolStick-F330DC

## 2. Contents

The ToolStick-F330DC kit contains the following items:

- ToolStick C8051F330 Daughter Card

The ToolStick Starter Kit includes the following items:

- ToolStick Base Adapter
- ToolStick C8051F330 Daughter Card
- 3-foot USB extension cable

A ToolStick daughter card requires a ToolStick Base Adapter to communicate with the PC. If the daughter card was not purchased as part of a Starter Kit, ToolStick Base Adapters can be purchased separately at www.silabs.com/toolstick.

## 3. ToolStick Overview

The purpose of the ToolStick is to provide a development and demonstration platform for Silicon Laboratories microcontrollers and to demonstrate the Silicon Laboratories software tools, including the Integrated Development Environment (IDE).

The ToolStick development platform consists of two components: the ToolStick Base Adapter and a daughter card. The ToolStick Base Adapter provides a USB debug interface and data communications path between a Windows PC and a target microcontroller.

The target microcontroller and application circuitry are located on the daughter card. Some daughter cards, such as the C8051F330 Daughter Card, are used as general-purpose development platforms for the target microcontrollers and some are used to demonstrate a specific feature or application.

The C8051F330 Daughter Card includes a pair of LEDs, a potentiometer, a resistor across the C8051F330's current DAC output pin, and a small prototyping area which provides access to all of the pins of the device. This prototyping area can be used to connect additional hardware to the microcontroller and use the daughter card as a development platform. See Section "10. Board Revision Information," on page 14 for information regarding the prototyping area.

Figure 3 shows the ToolStick C8051F330 Daughter Card and identifies the various components.

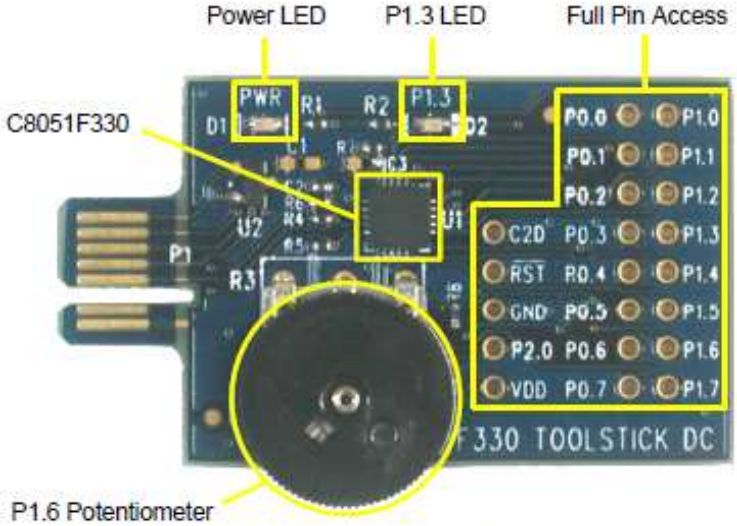



**Figure 3. ToolStick C8051F330 Daughter Card**


## Enrich Mobile Internet Experience

LG KS20 provides customers with an advanced mobile internet experience with its 'Broadband Mobile Internet', '2.8 inch Large Screen' and 'Intuitive Touch Input,' and is only 12.8 mm thick. Compatible with highly advanced Broadband Mobile Internet networks, the phone allows users to enjoy the full benefits of 3.6Mbps mobile internet, music downloads, video clips and Windows Live! ™ content, as well as push email.
Application of the 2.8 inch large screen allows you to have an easy-to-use mobile internet environment which is just the same as that of a PC screen. Also, the landscape function provides you with the optimum convenience to enjoy video clips.
A completely flat touchscreen, a perfectly stylish and slim design and a fluent and direct interaction further enhances your mobile internet convenience.
With LG KS20, you can enjoy your mobile internet experience anywhere at any time.

## Faster means more

- Broadband Mobile Internet
- Fast Music Download
- Seamless Video Streaming
- Push Email

## Enhanced Usability

- 2.8 inch Large Screen
- Intuitive Touch Input
- Natural Handwriting Recognition

## Stylish & Slim Design

- Pitch Black
- Full, Flat Touch Screen (Minimal Design)
- Slim Dimension - 12.8 mm thick

## Advanced feature

- 2Mega pixel Auto Focus Camera
- MP3 Player
- Downloadable free applications
- Bluetooth v2.0