ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"Diseño e Implementación de un Analizador de protocolo RS-232 basado en el Procesador NIOS II"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN TELEMÁTICA

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Steven Kleber Caicedo Mejillones

Jeanneth Mirelly Paredes Cedeño

GUAYAQUIL – ECUADOR

AÑO 2014

AGRADECIMIENTO

A mi Padre celestial, a mi bella madre, a mis queridos hermanos y a aquellas personas que estuvieron conmigo en este largo camino.

Steven Caicedo Mejillones

En primer lugar a Dios por haberme guiado, en segundo lugar a cada uno de los que son parte de mi familia por siempre darme sus fuerzas y apoyo incondicional, también a mi director de tesis quien nos ayudo en todo momento.

Jeanneth Paredes Cedeño.

DEDICATORIA

Este proyecto lo dedico a Dios por canalizar mi vida por el camino correcto. A mi madre ser el apoyo incondicional durante toda mi vida. A mis hermanos porque siempre creyeron en mí. A todas aquellas personas que fueron pilares en la construcción de mi vida.

Steven Caicedo Mejillones

Quiero dedicar este trabajo a las personas mas importantes en mi vida, a mis padres, quienes además de darme todo, también fueron el motor que me ha permitido alcanzar esta meta tan importante en mi vida.

Jeanneth Paredes Cedeño.

TRIBUNAL DE SUSTENTACIÓN

Ing. Ronald Ponguillo Intriago

PROFESOR DEL SEMINARIO DE GRADUACIÓN

Ing. Victor Asanza Armijos

PROFESOR DELEGADO POR LA UNIDAD ACADÉMICA

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta tesina, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Steven Kleber Caicedo Mejillones

Jeanneth Mirelly Paredes Cedeño

RESUMEN

En el presente trabajo se desarrolló una herramienta que servirá para recolectar y analizar las tramas del protocolo de comunicación serial RS-232 mostrando dicha información en una PC o en un Dispositivo con Sistema Operativo Android, lo que nos permitirá entender e interpretar el funcionamiento del protocolo antes mencionado además de ayudarnos en la tarea de detectar y resolver problemas relacionados con este tipo de comunicación.

Para esto se desarrolló un Sistema Embebido basado en el Microprocesador NIOS II el cual fue implementado en la tarjeta de desarrollo y educación DE0 nano de Terasic Inc., cuyo componte principal es la FPGA EP4CE22F17C6N de la familia Cyclone IVE de Altera.

Este informe se lo ha estructurado en 4 capítulos que detallamos a continuación:

El capítulo 1, plantea el proyecto como un problema, sus objetivos generales y específicos, así como su alcance y sus limitaciones.

El capítulo 2, presenta los fundamentos teóricos en los que se basó el presente trabajo tales como el análisis del protocolo de comunicación serial RS-232, el Microprocesador NIOS II y la Tarjeta DE0 nano de Altera, interfaces de entrada y de salida como lo son el teclado numérico y el LCD 16x2, además de una breve descripción del lenguaje de programación Java y el Sistema operativo Android.

El capítulo 3, muestra la solución del problema, es decir, el diseño e implementación del proyecto, el cual está orientado a un ambiente de laboratorio.

En el capítulo 4, presenta el análisis y los resultados obtenidos en las pruebas realizadas en el laboratorio.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

ALU	Unidad Aritmética Lógica
CAD	Diseño Asistido por Computadora
CLK	Reloj
CMOS	Semiconductor Complementario de Óxido Metálico
DCE	Equipo de Comunicación de Datos
DTE	Equipo Terminal de Datos
DSP	Procesador de Señales Digitales
EEPROM	Memoria ROM Programable y Borrable Eléctricamente
EPCS	Dispositivo de Configuración Serial
FPGA	Matriz de Puertas Lógicas Programables
FIFO	Primero Entra Primero Sale
GPIO	Entradas y Salidas de Propósito General
GPL	Licencia Pública General

GUIDE	Entorno de Desarrollo Integrado con Interfaz Gráfica
HDL	Lenguaje de Descripción de Hardware
IDE	Entorno de Desarrollo Integrado
IEEE	Instituto de Ingenieros Electrónicos y Eléctricos
JTAG	Prueba de Acción en Conjunto
LCD	Pantalla de Cristal Líquido
LED	Diodo Emisor de Luz
LSB	Bit Menos Significativo
MMU	Unidad de Administración de Memoria
MSB	Bit Más Significativo
PCB	Circuito Impreso
PC	Computadora Personal
RAM	Memoria de Acceso Aleatorio
RISC	Computador con Conjunto de Instrucciones Reducidas
RS-232	Estándar Recomendado 232
RX	Línea de Recepción

SDRAM	Memoria Sincrónica Dinámica de Acceso Aleatorio
SOC	Sistemas en Chip
SOPC	Sistemas en Chip Programables
TTL	Lógica Transistor-Transistor
TX	Línea de Transmisión
UART	Transmisor-Receptor Asincrónico Universal
USB	Bus Serial Universal
VHDL	Lenguaje de Descripción de Hardware VHSIC
VHSIC	Circuitos Integrados de Alta Velocidad

INTRODUCCIÓN

Desde la antigüedad los hombres han tenido la necesidad de interactuar e intercambiar información entre sí, a este proceso se le llama comunicación en la cual es muy necesaria la correcta interpretación de la información por parte del receptor del mensaje emitido.

Con la aparición de dispositivos electrónicos como computadoras, celulares surgieron las llamadas comunicaciones electrónicas. Para efectuar de forma efectiva estas comunicaciones los expertos desarrollaron distintos protocolos, entre las cuales está el protocolo RS-232.

En una comunicación electrónica es muy importante que los dispositivos apliquen de forma correcta las reglas definidas por los protocolos. He aquí la importancia de los analizadores de protocolos.

Estas herramientas nos permiten analizar la comunicación, detectar fallos y verificar el uso inequívoco del protocolo.

Para la implementación desarrollaremos un sistema embebido usando la tarjeta de desarrollo y educación DE0 nano.

CAPÍTULO 1

1. GENERALIDADES

Este es un capítulo dedicado a describir el proyecto como una necesidad vigente a la cual planteamos una posible solución, también indicaremos los objetivos que queremos alcanzar con el proyecto así como el alcance y limitación del mismo.

1.1 Alcance y limitaciones del proyecto

El proyecto debe cumplir las siguientes características:

- Se realizará un analizador de protocolo RS-232 cuyos parámetros de comunicación sean configurables.
- La configuración de nuestro analizador puede ser automática como manual.

- El análisis se concentrará exclusivamente en los pines de trasmisión y recepción de datos.
- La trama RS-232 se la analizará después de haberla recibido completamente.
- La visualización del estado del analizador se mostrara en un LCD 16x2.
- La visualización de los datos se lo podrá realizar en una computadora y en un teléfono celular, tableta o cualquier dispositivo con sistema operativo Android.
- La aplicación para el usuario final en la computadora se la desarrollara en el lenguaje de programación java.
- Los datos llegarán al dispositivo Android a través de una comunicación inalámbrica usando el protocolo Bluetooth.
- Los datos llegarán a la computadora a través de una comunicación alámbrica usando una interfaz UART-USB.
- Las pruebas se las harán solo en ambiente de laboratorio.

1.2 Objetivos

1.2.1 Objetivos Generales

Los objetivos generales para llevar a cabo el trabajo son los siguientes:

Usar conceptos adquiridos en el seminario acerca de sistemas embebidos, comunicación serial y su implantación en una FPGA, instruirnos en el manejo de un LCD 16x2 y un teclado numérico hexadecimal usando la tarjeta de desarrollo y educación DE0 nano, además de experimentar con la comunicación entre dispositivos que usan diversas tecnologías.

1.2.2 Objetivo Específico

El objetivo principal con este proyecto es diseñar un analizador de comunicación serial RS-232 desarrollando un Sistema Embebido basado en el Microprocesador NIOSII.

Además, nos enfocaremos en los siguientes puntos:

- Conocer y entender el funcionamiento de la tarjeta DE0 nano, sus características principales y módulos incorporados.
- Conocer e interpretar el protocolo de comunicación serial RS-232.

- Hacer uso de los programas Quartus II para el diseño de hardware de nuestro sistema y NIOS II IDE para Eclipse para el desarrollo del software que controlará dicho hardware.

- Usar módulos UART-USB y UART-Bluetooth, ambos externos a la tarjeta DE0 nano para la comunicación con una computadora y un dispositivo Android.

- Hacer uso del programa NetBeans IDE para el desarrollo de las aplicaciones para el usuario final en la computadora y en el dispositivo Android.

1.3 Antecedentes

El tiempo avanza y con él la tecnología, en un mundo que cada vez se globaliza más, es necesario estar al tanto de todo, en un mundo de alianzas donde la comunicación es importante, el hombre tuvo la necesidad de crear nuevas formas de comunicación donde las distancias no importen.

Las comunicaciones electrónicas han facilitado el avance de la tecnología, para esto se crearon normas para estandarizar las distintas formas de comunicación entre dispositivos, sean estas seriales paralelas, sincrónicas, asincrónicas.

Es así como nacieron los protocolos de comunicaciones que se usan para estandarizar y regir a mencionadas comunicaciones.

RS-232 es un protocolo de comunicación serial que generalmente se usa para el intercambio de datos entre un DTE y un DCE, antiguamente usado para la comunicación punto a punto entre PCs ahora reemplazadas por el USB y ETHERNET.

Aunque RS-232 es un protocolo que está en desuso en el ámbito doméstico, aún es muy usado en aplicaciones industriales, sean estas las comunicaciones entre sensores y microcontroladores, o entre circuitos digitales.

1.4 Identificación del Problema

Un sistema electrónico siempre está sujeto a errores o fallos, el objetivo de los desarrolladores es minimizar estos problemas y en caso de tenerlos resolverlos con la brevedad posible.

Por eso se han desarrollado herramientas que nos ayuden en los análisis y mantenimiento de los mismos tales como multímetros, osciloscopios, puntas de prueba, etc.

Si surge algún problema de comunicación en un sistema, un multímetro u osciloscopio podría ser una herramienta poco eficaz para la búsqueda del error y poder solucionar el problema, ya que hay muchas variables involucradas en una comunicación.

1.5 Descripción breve de la posible solución

Para el análisis y la corrección de errores en un sistema de comunicación es necesario el uso de un analizador de protocolos de comunicación. Que se encargará de analizar y mostrar cada variable definida por el mismo. En nuestro caso realizaremos un analizador de Protocolo RS-232 concentrándonos en los pines de trasmisión y recepción de datos. El mismo será implementado en una tarjeta DE0 nano mostrando los parámetros de configuración en un LCD 16X2 y los datos procesados serán enviados a una computadora o un dispositivo Android para su visualización.

CAPÍTULO 2

2. MARCO TEÓRICO

En este capítulo se presentan los principales conceptos que utilizamos en este proyecto, para de esta manera familiarizarnos con cada uno de ellos.

2.1 Sistemas Embebidos

Se puede definir un sistema embebido como un sistema de computación que consta de una electrónica programable especialmente diseñada para dar solución a una o pocas tareas específicas, tales como controlar equipos e instrumentación industrial, operación de maquinarias, etc. A diferencia de un ordenador de propósito general como una PC el sistema embebido esta implementado solo con el hardware necesario para realizar las tareas para las cuales fue diseñado reduciendo el costo, el tamaño y el consumo de potencia.

Un sistema embebido está compuesto por circuitos integrados, un procesador digital de señal (DSP), memoria, y el software que generalmente se ejecuta sobre un procesador interno, o en un microcontrolador.

2.2 Sistemas embebidos basados en microprocesador NIOS II

El fabricante Altera proporciona una infraestructura completa para crear sistemas de microprocesador embebido, según las necesidades del diseñador, por medio de la combinación de una serie de componentes configurables sobre sus FPGAs. Para su desarrollo, este fabricante proporciona un entorno de desarrollo llamado Qsys que permite la configuración a medida de nuestro sistema; y que gracias a la herramienta de síntesis Quartus II puede ser implementado directamente sobre una FPGA. [1]

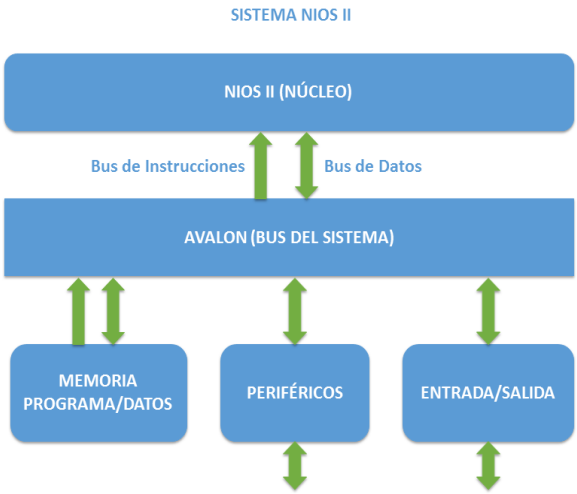


Figura 2-1Sistema Embebido basado en Microprocesador NIOS II [1]

El sistema está compuesto por: el núcleo procesador NIOS II, memoria interna de programa y de datos, periféricos integrados e interfaces para entrada, salida y memoria externa, ver Figura 2-1.

2.2.1 Versiones de NIOS II

NIOS II es un núcleo procesador configurable que se puede implementar en alguna de las tres versiones disponibles, según se busque minimizar el consumo de recursos de la FPGA o maximizar el rendimiento del procesador:

- El NIOS II/f (“fast”) es la versión diseñada para alto rendimiento, y que proporciona opciones para aumentar su desempeño, como memorias caché de instrucciones y datos, o una unidad de manejo de memoria (MMU, Memory Management Unit).
- El NIOS II/s (“standard”) es la versión que contiene la unidad aritmético lógica (ALU, Arithmetic Logic Unit) y busca combinar rendimiento y consumo de recursos.
- El NIOS II/e (“economy”) es la versión que requiere menos recursos de la FPGA, es muy limitada, dado que carece de las operaciones de multiplicación y división.[1]

2.2.2 Características y Arquitectura de NIOS II

NIOS II es un procesador de 32 bits de propósito general, basado en una arquitectura tipo Harvard, dado que usa buses separados para instrucciones y datos; y cuyas principales características son:

- Tamaño de palabra de 32 bits.
- Juego de instrucciones RISC de 32 bits.
- 32 registros de propósito general de 32 bits cada uno (r0 – r31).
- 6 registros de control de 32 bits (ctl0 - ctl5).
- 32 fuentes de interrupción externa.
- Capacidad de direccionamiento de 32 bits.
- Operaciones de multiplicación y división de 32 bits.
- Instrucciones dedicadas para multiplicaciones de 64 y 128 bits.
- Acceso a variedad de periféricos integrados e interfaces para manejo de memorias y periféricos.

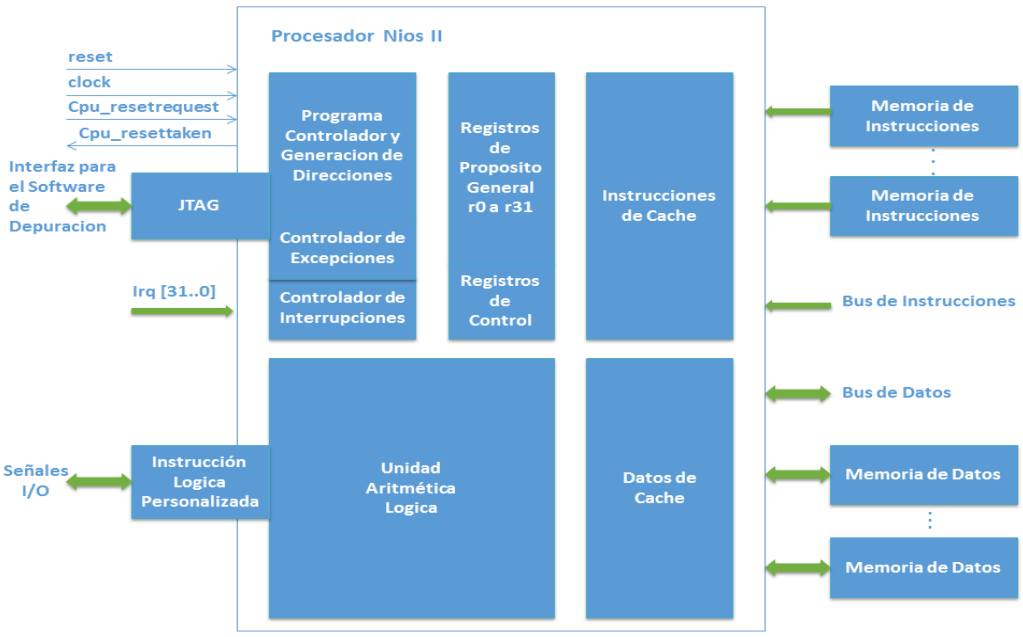


Figura 2-2 Diagrama de Bloques de la Arquitectura NIOS II [1]

2.2.3 Programación del NIOS II

Para la programación del NIOS II se dispone de una herramienta basada en Eclipse, con la que Altera da soporte para el desarrollo de aplicaciones en C/C++. NIOS II también permite que se realice programación en lenguaje ensamblador (“assembler”) como muestra la Figura 2-3b.

El carácter configurable del NIOS II exige que en todo programa se presente una primera etapa, la cual consiste en definir todos y cada uno de los periféricos del sistema, a los que se les asigna una dirección de memoria. [1]

```c
int main(void)
{
    volatile int * green_LED_ptr    = (int *) 0x10000010;   // green LED address
    volatile int * SW_switch_ptr    = (int *) 0x10000040;   // SW slider switch address
    volatile int * KEY_ptr          = (int *) 0x10000050;   // pushbutton KEY address
    int LEDG_bits = 0x0F0F0F0F;             // pattern for LEDG lights
    int SW_value, KEY_value;
    volatile int delay_count;                   // volatile so the C compiler doesn't remove the loop
    while(1)
    {
        SW_value = *(SW_switch_ptr);            // read the SW slider (DIP) switch values
        KEY_value = *(KEY_ptr);                 // read the pushbutton KEY values
        if (KEY_value != 0)                     // check if any KEY was pressed
        {
            /* set pattern using SW values */
            LEDG_bits = SW_value | (SW_value << 8) | (SW_value << 16) | (SW_value << 24);
            while (*KEY_ptr);                       // wait for pushbutton KEY release
        }
        *(green_LED_ptr) = LEDG_bits;           // light up the green LEDs
        /* rotate the pattern shown on the LEDs */
        if (LEDG_bits & 0x80000000)
            LEDG_bits = (LEDG_bits << 1) | 1;
        else
            LEDG_bits = LEDG_bits << 1;
        for (delay_count = 150000; delay_count != 0; --delay_count); // delay loop
    }
}
```

Figura 2-3a Ejemplo de código en C

```
    .text                                   /* executable code follows */
    .global _start
_start:
    movia       r16, 0x10000010     /* green LED base address */
    movia       r15, 0x10000040     /* SW slider switch (DIP switches) base address */
    movia       r17, 0x10000050     /* pushbutton KEY base address */
    movia       r19, LEDG_bits
    ldw     r6, 0(r19)              /* load pattern for LEDG lights */
DO_DISPLAY:
    ldwio       r4, 0(r15)              /* load slider (DIP) switches */
    ldwio       r5, 0(r17)              /* load pushbuttons */
    beq     r5, r0, NO_BUTTON
    mov     r6, r4                  /* use SW (DIP switch) values on LEDG */
    roli        r4, r4, 8
    or          r6, r6, r4
    roli        r4, r4, 8
    or          r6, r6, r4
    roli        r4, r4, 8
    or          r6, r6, r4
WAIT:
    ldwio       r5, 0(r17)              /* load pushbuttons */
    bne     r5, r0, WAIT            /* wait for button release */
NO_BUTTON:
    stwio       r6, 0(r16)              /* store to LEDG */
    roli        r6, r6, 1               /* rotate the displayed pattern */
    movia       r7, 150000              /* delay counter */
DELAY:
    subi        r7, r7, 1
    bne     r7, r0, DELAY
    br      DO_DISPLAY
    .data                                   /* data follows */
LEDG_bits:
    .word 0x0F0F0F0F
    .end
```

Figura 2-3b Ejemplo de código en Assembler

2.3 Bus Avalon

Avalon es una arquitectura de bus simple diseñada por Altera para interconectar procesadores integrados y periféricos dentro de un SOPC (Sistem-on-a-programmable chip), para utilizarlo junto a su procesador soft-core NIOS II. Avalon es un interfaz que especifica los pines de conexión entre los componentes maestros y esclavos, además de los tiempos requeridos para su comunicación.

La transacción básica de este bus es capaz de transferir de 8 a 32 bits entre un componente maestro y un periférico esclavo. Después de completarse una transferencia, el bus queda inmediatamente libre para que en el siguiente ciclo de reloj se pueda producir otra transferencia, bien entre los mismos componentes o bien entre otros distintos. El bus Avalon también soporta otros modos de transferencia más avanzados que logran varias transferencias, entre distintos componentes, simultáneos. Avalon soporta una técnica de arbitraje que permite conectar varios componentes maestros con un mismo componente esclavo, de manera que el árbitro decide en cada momento que componente maestro realizará una transferencia con el periférico esclavo.

Las características principales del bus Avalon son las siguientes:

- Arquitectura multi master.
- Espacio de direcciones de 32 bits donde mapear los distintos componentes de memoria y periféricos.

- Todas las señales del bus están sincronizadas con el reloj.
- Bus de direcciones y de datos separados.
- El bus Avalon genera automáticamente las señales de Chip Select para todos los periféricos.

Todo sistema que utilice el bus Avalon debe incorporar un módulo específico, el cual contiene todas las señales de control, de datos y de direcciones, además de la lógica de arbitraje necesaria para conectar todos los componentes del sistema entre sí. Este módulo, denominado módulo de bus Avalon, implementa una arquitectura de bus configurable que puede variar para adaptarse a los requerimientos de interconexión solicitados por el diseñador. El diseñador del sistema no tiene que conectar manualmente todos los componentes, ya que es la propia herramienta SOPC Builder o Qsys las que construyen el módulo dependiendo de los componentes a conectar. La visión del diseñador del bus se limita a los puertos específicos de cada componente. Se puede observar un ejemplo de sistema con bus Avalon en la Figura 2-1. Un periférico conectado al bus Avalon es un componente lógico que puede estar implementado dentro del mismo chip o fuera de él. Cada periférico puede tener una tarea diferente y pueden ser añadidos o eliminados del sistema (y por lo tanto su conexión al módulo del bus Avalon) en tiempo de diseño, dependiendo de los requerimientos.

Los periféricos conectados al bus Avalon pueden ser maestros o esclavos. Un periférico maestro puede iniciar una transferencia mediante el bus

Avalon y al menos tiene un puerto maestro, que se conecta al módulo del bus Avalon. Un periférico maestro también tiene un puerto esclavo que le permite recibir transferencias mediante el bus, iniciadas por otros componentes maestros. En cambio, un periférico esclavo únicamente acepta transferencias del bus Avalon y no puede iniciar él mismo las transferencias. Estos últimos, que suelen ser memorias u otros periféricos, normalmente tienen un puerto esclavo que se conecta al módulo del bus Avalon.

Las especificaciones del bus Avalon definen las señales y los requerimientos temporales requeridos para la correcta transferencia de datos entre un puerto maestro y un puerto esclavo, vía el módulo del bus Avalon. Las señales que componen la interfaz entre dicho módulo y el periférico pueden ser diferentes, dependiendo del tipo de transferencia.

Avalon es un bus síncrono, dirigido por un reloj. Todas las transferencias ocurren de forma síncrona con dicho reloj y se inician en el flanco de subida del reloj. [2]

2.4 Tarjeta DE0 nano de Altera

La tarjeta DE0-Nano es una plataforma de desarrollo basado en FPGA adecuado para una amplia variedad de proyectos de diseños portables, como robots y proyectos móviles.

La DE0-Nano es ideal para diseño con procesadores embebidos, además posee incorporada una herramienta para la programación de la FPGA llamada USB Blaster. La tarjeta puede ser alimentada eléctricamente

mediante este puerto USB o mediante una fuente de alimentación externa.[3]

2.4.1 Características

Las características principales de la tarjeta son las siguientes:

- Dispositivos Incluidos
    - FPGA Altera Cyclone ® IV EP4CE22F17C6N
    - 153 pines de E/S
- Elementos de Configuración
    - USB-Blaster para la programación
    - EPCS64
- Pines de Expansión
    - Dos puertos de expansión de 40 pines (GPIO) proporcionan 72 pines de E/S
    - 2 pines de alimentación de 5V
    - 2 pines de alimentación de 3,3 V
    - 4 pines de tierra.
- Dispositivos de Memoria
    - 32 MB SDRAM
    - 2Kb I2C EEPROM
- Interfaces de Entrada y Salida
    - 8 LEDs

- 2 pulsadores sin rebote
- 4 Interruptores

• G- Sensor

- ADI ADXL345, acelerómetro de 3 ejes con alta resolución (13 bit)

• Convertidor A/D

- NS ADC128S022, 8 canales, 12-bit convertidor A/D
- 50 Ksps a 200 Ksps

• Reloj del Sistema

- Oscilador de 50 MHz

• Fuente de alimentación

- Puerto USB Tipo mini-AB (5V)
- 2 pines de alimentación externa (3,6-5.7V)

2.4.2 Diseño y Componentes

En la Figura 2-4 y la Figura 2-5 se muestra la ubicación de los conectores y componentes principales. [3]

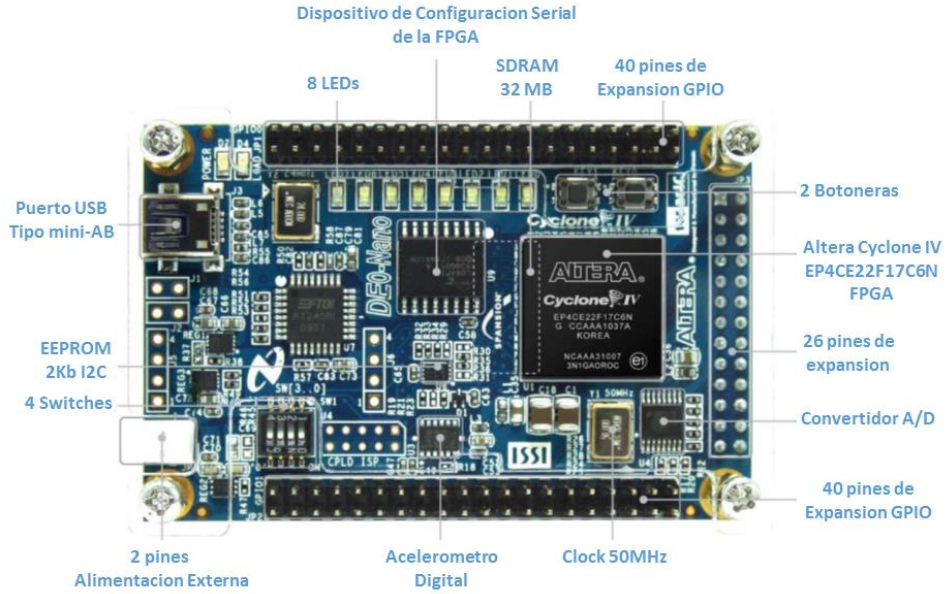


Figura 2-4 Diagrama de componentes de la DE0-Nano (vista superior)

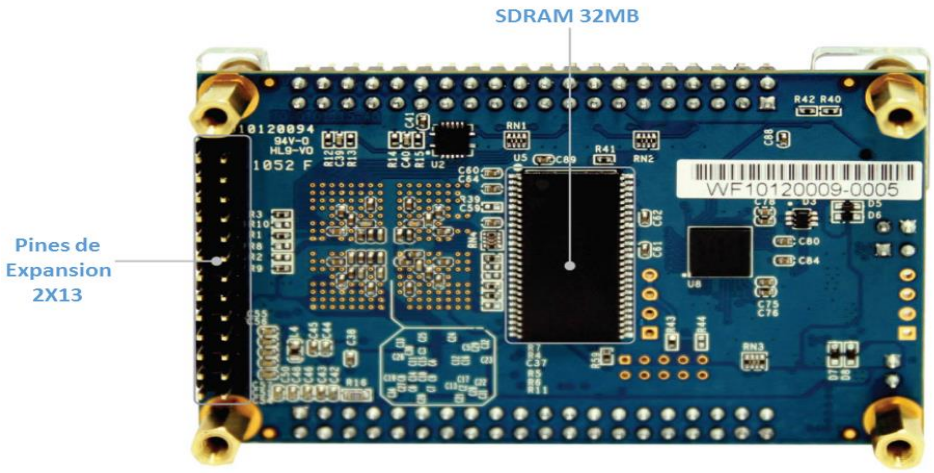


Figura 2-5 Diagrama de componentes de la DE0-Nano (vista inferior)

2.4.3 Diagrama de bloques

La Figura 2-6 muestra el diagrama de bloques de la tarjeta DE0-Nano. Todas las conexiones se realizan a través del dispositivo FPGA Ciclón IV, por lo tanto, el usuario puede configurar el FPGA para implementar cualquier diseño del sistema. [3]

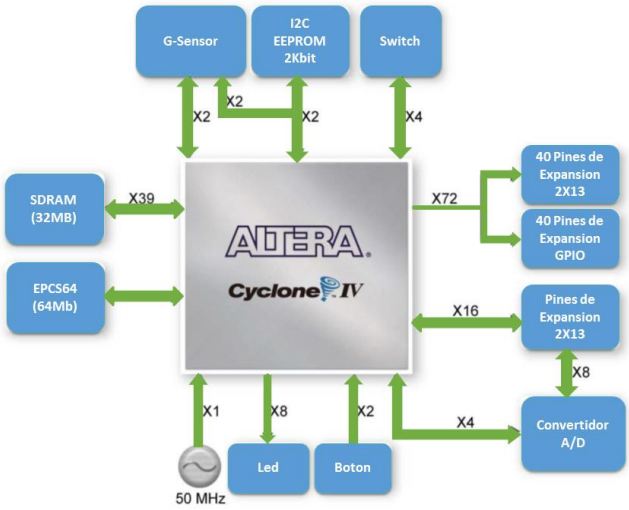


Figura 2-6 Diagrama de bloques de la Tarjeta DE0-Nano [3]

2.5 Comunicación Serial

La comunicación serial es el proceso de envío de bytes de información de manera secuencial, es decir, un bit a la vez. Existen dos formas de establecer una comunicación serial, en forma sincrónica y asincrónica. En la comunicación asincrónica, se puede enviar datos por una línea y recibir datos por otra ya que para la comunicación se utilizan 3 líneas de comunicación: tierra, Tx y Rx. En la comunicación sincrónica existe una señal de reloj común para los sistemas digitales que se interconectan, esta forma de comunicación es muy utilizada para conectar periféricos como convertidores A/D, memorias, acelerómetros, entre otros.

2.5.1 El protocolo RS-232

El protocolo RS-232 es una norma de comunicación serial que define especificaciones mecánicas, eléctricas. Regula la transmisión

de datos, el cableado, las señales eléctricas y los conectores en los que debe basarse. [4]

Especificaciones

- Mecánicas: El RS-232 consiste en un conector tipo DB-25 (de 25 pines), se lo puede encontrar también en la versión de 9 pines (DB-9). El estándar define que el conector hembra se situará en los DCE y el macho en el DTE. Cada pin puede ser de entrada o de salida, teniendo una función específica cada uno de ellos. [4]

- Eléctricas: Los estados lógicos son definidos por los siguientes niveles de voltaje: 1 lógico entre -3V y -15V, 0 lógico entre +3V y +15V. La interfaz RS-232 está diseñada para distancias cortas, de hasta 15 metros y para velocidades de comunicación bajas, de no más de 20 Kb/s.[4]

2.6 Módulo NIOS UART

El módulo NIOS UART es un componente de la librería SOPC Builder de Altera que implementa al protocolo RS-232 asíncrono. El módulo UART envía y recibe datos de forma serial a través de dos pines externos (RxD

y TxD). Para el control por Software del módulo UART se usan 5 registros de 16 bits (ver figura 2-7).

Para cumplir con los voltajes especificados en la norma RS-232 se necesita que los pines RxD y TxD estén conectados a un convertidor de voltaje y este a su vez al correspondiente conector DB9 o DB25 especificado en la Norma (ver figura 2-7). El rango de voltaje aceptable de los pines de entrada y salida del módulo depende sobre cómo se configuren los pines de E/S en el dispositivo de Altera. El módulo UART trabaja con una entrada de reloj síncrona, clk. [5]

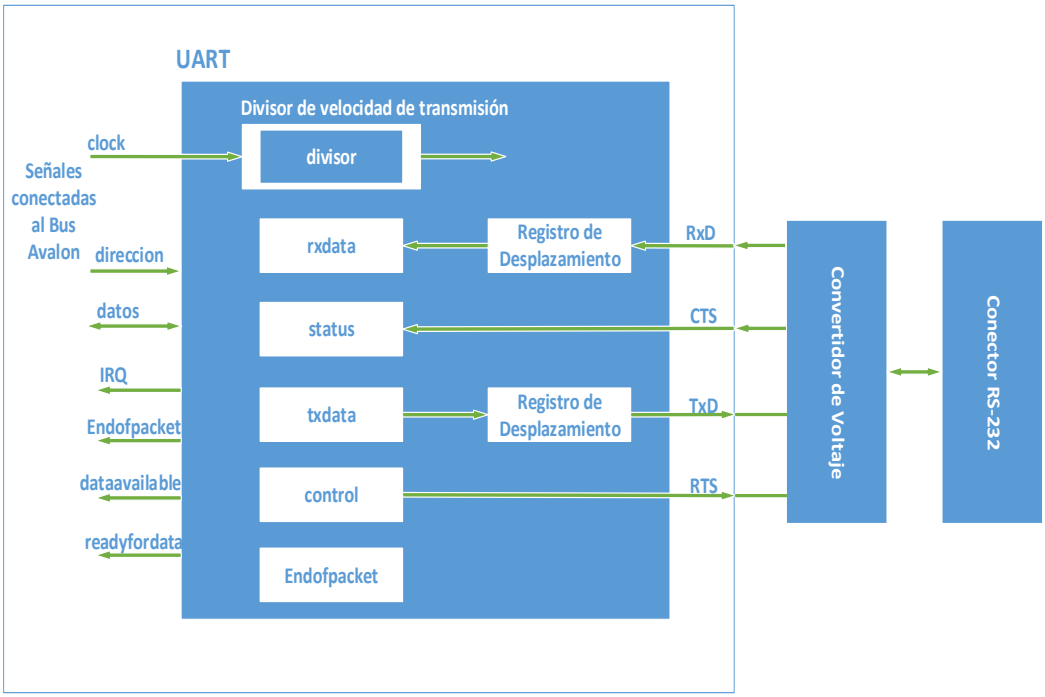


Figura 2-7 Diagrama de bloques del Módulo UART [5]

2.6.1 Trasmisor

El transmisor UART consiste en un registro de sostenimiento llamado TxData de 7, 8 o 9 bits y un registro de desplazamiento también de 7,8 o 9 bits. El registro TxData puede ser escrito directamente por software mientras que el registro de desplazamiento que es usado para la trasmisión se carga automáticamente desde el registro TxData siempre y cuando una operación de trasmisión no esté en proceso. El tamaño de estos registros depende de la configuración del módulo y no puede ser cambiado por software. El registro de desplazamiento está conectado directamente al pin TxD y desplaza los datos hasta el pin TxD empezando por el LSB.

Estos dos registros proporcionan un doble búfer, el software puede escribir un nuevo valor en el registro TxData mientras que el caracter escrito anteriormente es siendo desplazado hacia afuera. El software puede monitorear el estado del transmisor mediante la lectura de los bits TRDY (listo para trasmitir), TMT (registro de desplazamiento vacio) y TOE (sobre escritura en la trasmisión) del registro de estado.

El transmisor lógico inserta automáticamente el bit de inicio, bit de parada, y los bits de paridad de acuerdo a como se haya configurado. [5]

2.6.2 Receptor

El transmisor UART consiste en un registro de sostenimiento llamado RxData de 7, 8 o 9 bits y un registro de desplazamiento también de 7,8 o 9 bits. El registro RxData registro puede ser leído directamente por el software. El registro RxData es cargado desde el registro de desplazamiento cada vez que un nuevo caracter se recibe completamente.

Estos dos registros proporcionan doble buffer. El registro RxData puede mantener un carácter recibido previamente, mientras que el carácter subsiguiente es siendo desplazado en el registro de desplazamiento del receptor.

El software puede monitorear el estado del receptor mediante la lectura de los bits RRDY (listo para recibir), ROE (error de sobre escritura en la recepción), BRK (error de interrupción de trama), PE (error de paridad) y FE (error de encuadre) del registro de estado. La lógica del receptor detecta automáticamente el bit de inicio, parada y bits de paridad de acuerdo a como haya sido configurado el módulo UART. [5]

2.6.3 Registros del módulo UART

El módulo UART como ya lo habíamos mencionado tiene 5 registros que pueden ser leído directamente por software y se distribuyen como lo muestra la tabla 2-1.

UART REGISTER MAP																
NO	REGISTRO	R/W	DESCRIPCIÓN / REGISTRO BITS													
			15	...	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RXDATA	RO								RXDATA							
1	TXDATA	WO								TXDATA							
2	STATUS	RW		EOP	CTS	DCTS	-	E	RRDY	TRDY	TMT	TOE	ROE	BRK	FE	PE	
3	CONTROL	RW		IEOP	RTS	IDCTS	TRBK	IE	IRRDY	ITRDY	ITMT	ITOE	IROE	IBRK	IFE	IPE	
4	DIVISOR	RW	BAUD RATE DIVISOR (OPCIONAL)														
5	ENDOFPACKET	RW							END-PACKET VALUE								

Tabla 2-1 Registros del Módulo UART [5]

2.6.4 Registro de estado

El registro de estado se compone de bits individuales que indican condiciones especiales dentro del módulo UART (ver tabla 2-1). El registro de estado puede ser leído en cualquier momento por el software. La lectura del registro de estado no cambia el valor de cualquiera de los bits. Cada bit de estado está asociado con un correspondiente bit de habilitación en el registro de control. La mayoría de los bits de estado se ponen a 0 cuando el software realiza una operación de escritura para el registro de estado (el valor de datos escrito se ignora). [5]

Bit	Nombre	Descripción
0	PE	Error de Paridad
1	FE	Error de trama
2	BRK	Error de ruptura
3	ROE	Error de sobre escritura en la lectura
4	TOE	Error de sobre escritura en la trasmisión
5	TMT	Trasmisor vacío
6	TRDY	Listo para trasmitir
7	RRDY	Listo para leer
8	E	Excepción
10	DCTS	Cambio en la señal CTS
11	CTS	Señal CTS
12	EOP	Se encontró el carácter final de paquete

Tabla 2-2 Registro de Estado del Módulo UART [5]

Bit PE

Un error de paridad se produce cuando el bit de paridad recibido tiene un nivel lógico inesperado (incorrecto). El bit PE se establece en 1 lógico cuando el UART recibe un caracter con un incorrecto bit de paridad. Cuando el UART está configurado sin bit de paridad el bit PE siempre es 0.El bit PE permanece en 1 hasta que sea modificado manualmente del registro de estado. [5]

Bit FE

El bit FE se pone en 1 lógico cuando el UART recibe un carácter y no puede detectar correctamente el bit de parada. El bit FE

permanece en 1 hasta que sea modificado manualmente del registro de estado. [5]

Bit BRK

El bit BRK es establecido en 1 lógico cuando el pin RxD se mantiene bajo (0 lógico) continuamente durante más de un tiempo completo de caracteres (7, 8 o 9 ciclos de bit, además de bit de inicio, bit de parada y bit de paridad). El bit BRK permanece en 1 hasta que sea modificado manualmente del registro de estado. [5]

Bit ROE

El bit ROE se pone en 1 lógico cuando un carácter recién recibido es transferido al registro de sostenimiento RxData antes que el carácter anterior sea leído por el software. El bit ROE permanece en 1 hasta que sea modificado manualmente del registro de estado. [5]

Bit TOE

El bit toe se establece en 1 lógico cuando el software escribe un nuevo carácter en el registro TxData mientras que el bit de TRDY es 0 (es decir, antes de que el carácter anterior se transfiera al registro de desplazamiento transmisor). El bit TOE permanece en 1 hasta que sea modificado manualmente del registro de estado. [5]

Bit TMT

El bit TMT indica el estado actual del registro de desplazamiento. Cuándo el registro de desplazamiento transmisor está en el proceso de trasmisión hacia el pin TxD, TMT se establece en 0. Cuando el registro de desplazamiento transmisor está inactivo (es decir, un carácter no está siendo transmitido) el bit de TMT es 1. El software puede determinar si se ha completado la transmisión. El bit de TMT no se cambia por una operación de escritura en el registro de estado. [5]

Bit TRDY

El bit TRDY indica el estado actual del registro TxData. Cuándo el registro TxData está vacío (es decir, su contenido han sido transferido hacia el registro de desplazamiento transmisor), significa que está listo para un nuevo carácter, TRDY se establece en 1 lógico. Si el valor en el registro TxData no ha sido transferido hacia el registro de desplazamiento transmisor (debido a que este está ocupado trasmitiendo el carácter anterior), TRDY es 0. El software siempre debe esperar para TRDY para igualar 1 antes de escribir un nuevo dato en el registro TxData. El bit de TRDY no se cambia por una operación de escritura en el registro de estado. [5]

Bit RRDY

El bit RRDY indica el estado actual del registro RxData. Cuándo el registro RxData está vacío (es decir, la UART no ha recibido nuevos caracteres) RRDY es 0. Cuando un valor ha sido recibido se transfiere al registro RxData, RRDY se establece en 1. El bit RRDY se establece en 0 cuando el software realiza una operación de lectura en el registro RxData. El software siempre se debe esperar a que RRDY sea igual a 1 antes de leer un caracter del registro RxData. El bit de RRDY no se cambia por una operación de escritura en el registro de estado. [5]

Bit E

Es un OR lógico entre los bits TOE, BRK, FE, PE, ROE. El bit E se establece en 0 por una operación de escritura en el registro de estado, debido a que todos de sus bits constituyentes se establecen en 0 por esta acción. [5]

Bit DCTS

El registro de estado incluye el bit DCTS cuando se ha configurado el módulo para que tenga un control de flujo (use_cts_rts = 1). Este bit se establece en 1 siempre que se detecte una transición de nivel lógico (1 a 0 o 0 a 1) en el pin de sincronización cts_n. El bit de

DCTS mantiene el valor 1 hasta que el software realice una operación de escritura en el registro de estado. Cuando no este habilitado el control de flujo el bit DCTS es siempre 0. [5]

Bit CTS

El registro de estado incluye el bit CTS cuando se ha configurado el módulo para que tenga un control de flujo (use_cts_rts = 1). Este bit refleja la señal de entrada CTS instantánea. El hardware UART usa una lógica negativa. Por lo tanto, CTS = 1 cuando un nivel lógico 0 se aplica a la entrada cts_n externa, y CTS = 0 cuando una lógica de nivel 1 se aplica a la entrada cts_n externa. Cuando no este habilitado el control de flujo el bit CTS es siempre 0. [5]

Bit EOP

El registro de estado incluye el bit EOP cuando el parámetro use_eop_register = 1. Específicamente, el bit EOP se establece en 1 cuando el software ejecuta una operación de escritura en el registro TxData y el valor de los datos escritos en TxData es el mismo que el valor actual del registro endofpacket. Este bit también se establece a 1 cuando el software realiza una transacción de lectura del registro RxData y el valor de lectura de datos por el software es el mismo que el valor actual del registro endofpacket. El bit EOP se mantiene

en 1 hasta que el software realice una operación de escritura en el registro de estado. [5]

2.6.5 Registro de control

El registro de control se compone de bits individuales (ver tabla 2-3). El registro de control puede ser leído en cualquier momento por el software. Cada bit en el registro de control permite habilitar el bit de interrupción correspondiente en el registro de estado. Por ejemplo, el bit PE es el bit 0 del registro de estado, y el bit correspondiente IPE es el bit 0 del registro de control. Para cada bit del registro de estado, una solicitud de interrupción es se genera cuando tanto el bit de estado y su correspondiente bit habilitador son igual a 1. [5]

Bit	Nombre	Descripción
0	IPE	Habilitador de la interrupción de Pe
1	IFE	Habilitador de la interrupción de FE
2	IBRK	Habilitador de la interrupción de BRK
3	IROE	Habilitador de la interrupción de ROE
4	ITOE	Habilitador de la interrupción de TOE
5	ITMT	Habilitador de la interrupción de TMT
6	ITRDY	Habilitador de la interrupción de TRDY
7	IRRDY	Habilitador de la interrupción de RRDY
8	IE	Habilitador de la interrupción de E
9	TRBK	Transmitir un ciclo de datos de 0 lógicos
10	IDCTS	Habilitador de la interrupción de DCTS
11	RTS	Petición para enviar (RTS)
12	IEOP	Habilitador de la interrupción de EOP

Tabla 2-3 Registro de Control del Módulo UART [5]

TRBK Bit

La Bit de ruptura de transmisión, permite al software para transmitir un carácter de ruptura a través del pin TxD del módulo UART. El pin TxD se establece en 0 cuando el bit TRBK es igual a 1. El bit TRBK interfiere con cualquier transmisión en curso. El software debe establecer el bit TRBK a 0 después del período de ruptura deseado.

Bit RTS

El registro de control incluye el bit RTS cuando se ha configurado el modulo para que tenga un control de flujo (use_cts_rts = 1). Este bit puede ser leído y modificado. Alimenta directamente el pin de salida rts_n con lógica negativa. El software puede escribir RTS en cualquier momento. El valor de RTS no tiene ningún efecto sobre el módulo UART al transmitir o recibir. El único propósito de bits RTS es determinar el valor del pin de salida rts_n. Cuando RTS es 1, un nivel lógico bajo (0) es impulsado en el pin de salida rts_n. Cuando RTS es 0, un nivel lógico alto (1) es impulsado en el pin de salida rts_n. [5]

2.6.6 Registro Divisor

El registro divisor sólo se aplica cuando el parámetro fixed_baud= 0 en la configuración de modulo. Cuándo fixed_baud =1, el registro

divisor no existe y la escritura en el registro divisor no tienen ningún efecto, y el resultado de una lectura el funcionamiento del registro divisor no está definido. [5]

Cuando fixed_baud=0, se utiliza el contenido del registro divisor para generar la velocidad de transmisión del módulo UART. La velocidad de transmisión del módulo UART es calculado por la fórmula:

$$baud\ rate = \frac{clolck\_freq}{(divisor + 1)}$$

Figura 2-8 Formula para el cálculo de la velocidad de trasmisión

El software puede leer y escribir el valor en el registro divisor en cualquier momento. Para calcular el valor que debemos escribir en el registro divisor para una determinada velocidad de trasmisión usamos la siguiente fórmula:

$$divisor = int\left(\frac{clolck\ frequency}{baud\ rate} + 0.5\right)$$

Figura 2-9 Formula para el cálculo del valor del registro divisor

2.6.7 Registro endofpacket

El registro endofpacket es una característica de hardware opcional que almacena un carácter que determina el final de un paquete. Si

la opción incluir hardware registro al final de su paquete no está habilitado, el registro endofpacket no existe. En este caso, la escritura endofpacket no tiene ningún efecto, y la lectura devuelve un valor indefinido. El valor por defecto es cero que es el ASCII carácter nulo (\ 0). [5]

2.7 Universal Serial Bus

El USB es una interfaz para la transmisión serie de datos que provee una mayor velocidad de transferencia comparado con el protocolo de comunicación también serial RS-232 (12 Mbps en su versión 1.1). USB permite conectar dispositivos periféricos al ordenador rápidamente, sin necesidad de reiniciarlo ni de volver a configurar el sistema. USB tiene un diseño Maestro-Esclavo es decir un solo host USB y múltiples periféricos conectados en serie, en una estructura de árbol utilizando concentradores. Se pueden conectar hasta 127 dispositivos a un sólo host USB (incluyendo los concentradores).

2.8 Bluetooth

Bluetooth es un protocolo de comunicación para redes inalámbricas de área personal (WPAN) que hace posible la trasmisión de voz y datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda de los 2.4 GHZ.

2.9 LCD 16x2

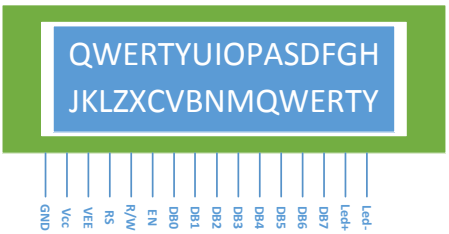


Figura 2-10 LCD 16x2

El LCD (Liquid Crystal Display) es básicamente un dispositivo que sirve para la visualización de datos en un sistema electrónico (ver figura 2-10). Está compuesto básicamente por una pantalla de cristal líquido y un circuito microcontrolador que regula los parámetros necesarios para la presentación de caracteres ASCII, Kanji y Griego. El LCD 16x2 dispone de 2 filas de 16 caracteres cada una y cada carácter dispone de una matriz de 5x7 pixeles. Puede almacenar 40 caracteres por línea de pantalla, además puede ser controlado por un procesador usando un interfaz de 4 u 8 bits.

2.9.1 Módulo Controlador de LCD Optrex 16207

Optrex 16207 LCD es un módulo que proporciona la interfaz de hardware y controladores de software necesarios para que el procesador Nios II pueda controlar un LCD de 16x2.

El núcleo del controlador LCD consta de dos componentes visibles para el usuario:

- Once señales que se conectan al LCD16X2
    - E: Habilitador (Salida)
    - RS: Selector de registro (Salida)
    - R/W: Lectura/ Escritura (Salida)
    - Db0-Db7: Bus de datos (Bidireccional)
- Una interfaz Avalon que da acceso a los 4 registros que posee este módulo. [6]

2.10 Teclado Hexadecimal

Es un dispositivo de entrada de datos que consta de 16 teclas o pulsadores, dispuestos e interconectados en filas y columnas. En la siguiente figura vemos el esquema de conexionado interno del teclado matricial y sus correspondientes pines de salida. Cuando se presiona un pulsador se conecta una fila con una columna, teniendo en cuenta este hecho es muy fácil averiguar que tecla fue pulsada.

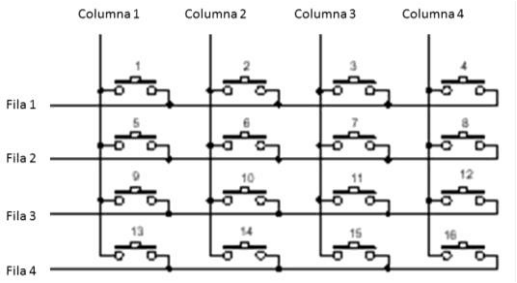


Figura 2-11 Teclado Hexadecimal de 4x4

2.11 Java (Lenguaje de Programación)

Java es un lenguaje compilado e interpretado. Todo programa en Java ha de compilarse y el código que se genera bytecodes es interpretado por la máquina virtual de Java (JVM). De este modo se consigue la independencia de la plataforma, el código compilado se ejecuta en la JMV que si es dependiente de la plataforma. Una de las principales características de Java es que es un lenguaje orientado a objetos y para el desarrollo de aplicaciones en este lenguaje es necesario usar el llamado JDK o kit de desarrollo de Java.

2.11.1 RXTX

RXTX es una Librería de Java, utilizando una aplicación nativa (a través de JNI), que proporciona comunicación serie y paralelo para el JDK. Toda la librería está bajo la licencia GNU LGPL. Se basa en la especificación del API Java Communications, sin embargo, en muchas de las descripciones se utiliza el paquete gnu.io. Esta librería da soporte para los sistemas operativos Windows, Linux y MacOS X. [7]

2.12 Android

Android es un sistema Operativo desarrollado por Google basado en Linux para teléfonos móviles, tabletas, portátiles e incluso PCs.

Android permite programar en Java para el desarrollo de sus aplicaciones. En la figura 2-12 se muestran las capas que componen este Sistema Operativo:

- Kernel de Linux.
- Bibliotecas creadas para el desarrollo de aplicaciones.
- Administradores de recursos.
- Aplicaciones. [8]

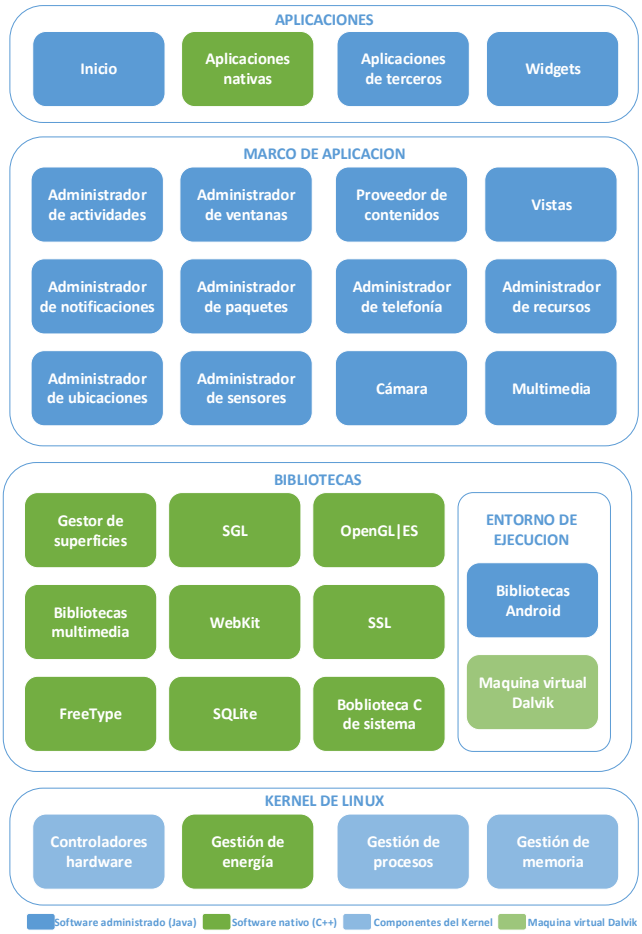


Figura 2-12 Capas de Android [8]

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN

Este capítulo presenta el diseño de nuestro analizador de protocolos así como la implementación del mismo. Este fue realizado desarrollando un sistema embebido, el cual consta de componentes de hardware y software que administra dicho hardware.

Adicionalmente se desarrolló aplicaciones intuitivas para el uso del usuario final de nuestro proyecto, una aplicación para un usuario en una PC y una para un usuario en un dispositivo móvil con Sistema Operativo Android.

La figura 3-1 nos da una idea general del uso del analizador, este diagrama facilita el entendimiento del diseño del analizador de protocolo RS-232 ya que muestra una idea clara del producto final. Como vemos nuestro analizador está conectado a las tres líneas de comunicación de los dispositivos a analizar.

Tiene una conexión con una PC a través de una interfaz UART-USB y con un equipo Android a través de una comunicación Bluetooth, ambos equipos usados para la visualización de los datos que intercambian los dispositivos analizados.

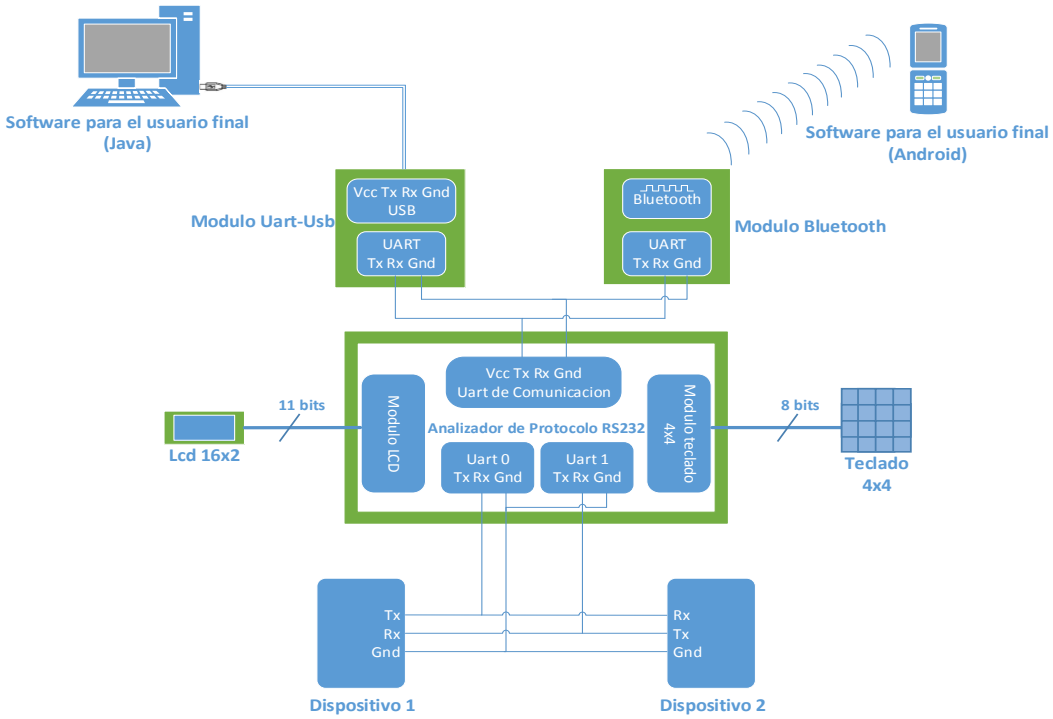


Figura 3-1 Diagrama de funcionamiento

3.1 Diseño del Hardware

Esta sección está dedicada a mostrar el diseño del hardware del analizador. Entre los componentes físicos que se usaron en este proyecto están: la tarjeta de desarrollo y educación DE0 nano de Altera, un módulo de comunicación UART-Bluetooth, un módulo de comunicación UART-USB, un LCD 16x2, un Teclado Hexadecimal de

4x4, un bus de datos de 40 pines y resistencias. Adicionalmente en la FPGA que se encuentra en la tarjeta DE0 nano se embebió módulos de hardware para implementar el sistema computacional que es el cerebro de nuestro analizador.

3.1.1 Diseño del Hardware embebido en la FPGA

La figura 3-2 muestra claramente el hardware que se embebió en la FPGA, diferenciándolo así del hardware físico que lo complementa. El diseño se lo realizó basándonos en un sistema computacional funcional: Procesador, memoria e interfaces de entrada y salida, adicionando módulos de comunicación UART, controlador de LCD, etc., adecuados a la necesidad de nuestro proyecto.

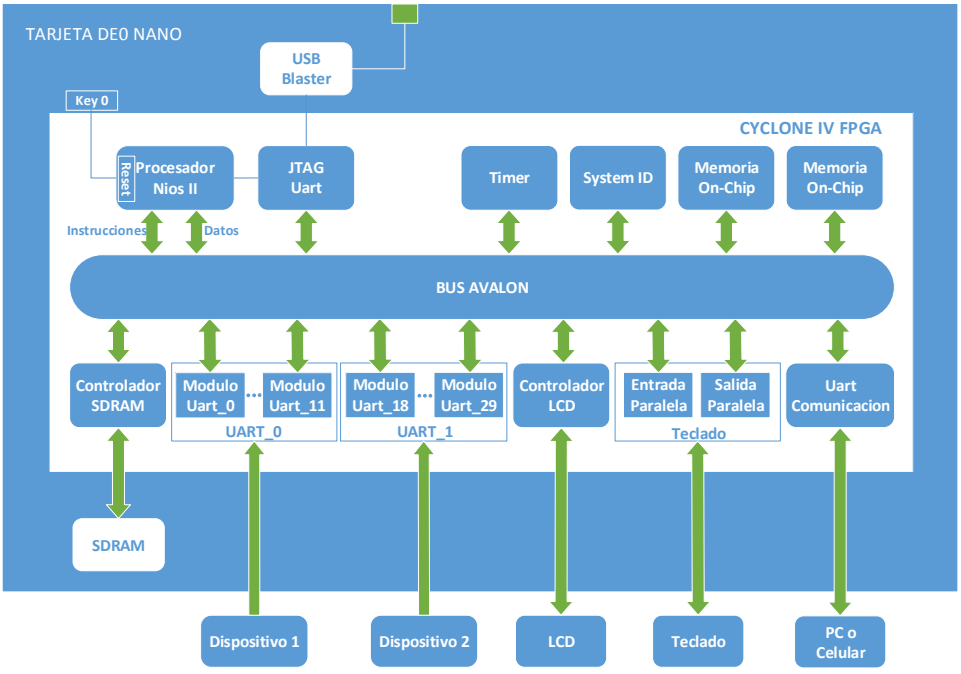


Figura 3-2 Diagrama de Bloque General

Para la implementación de este sistema se usaran el Software Quartus II y una de sus herramientas de diseño de hardware Qsys.

3.1.2 Creación del Proyecto en Quartus II

El primer paso en la implementación de nuestro sistema embebido es la creación de un proyecto en Quartus II. Para esto vamos a la pestaña File opción New Proyect Wizard y a continuación saldrá la ventana de la figura 3-3.

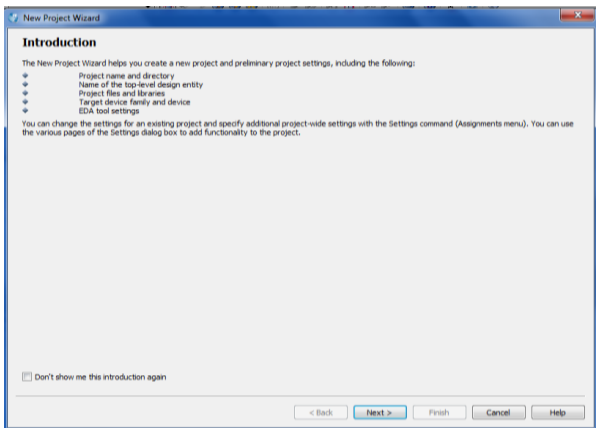


Figura 3-3 Nuevo Proyecto en Quartus II

Seleccionamos el botón Next y a continuación nos pedirá ingresar el nombre del proyecto. Añadimos el nombre “Analizador” y pulsamos Next, luego nos preguntara si deseamos añadir archivos ya desarrollados previamente para usar en el proyecto, a lo que simplemente hicimos pulsamos en Next. En el siguiente paso nos pedirá que seleccionemos el chip con el que vamos a trabajar, el

cual es FPGA EP4CE22F17C6N de la familia Cyclone IVE de Altera. (Ver figura 3-4)

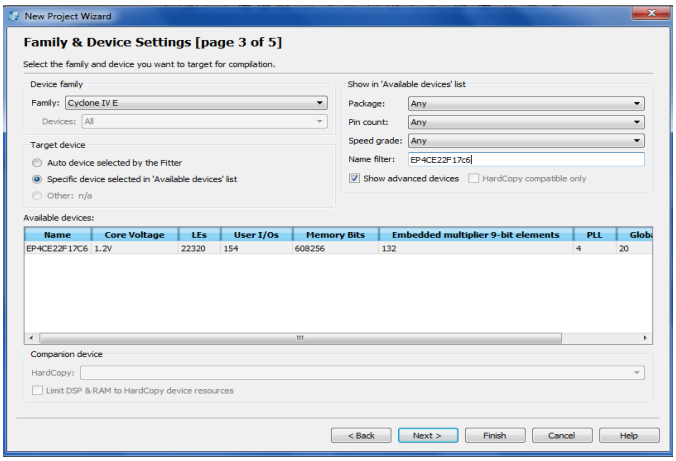


Figura 3-4 Asignación del Dispositivo a Usarse en Quartus II

En la siguiente pestaña nos pedirá que especifiquemos las Herramientas EDA (Electronic Desing Automation) a usar en nuestro proyecto, a lo que simplemente pulsamos Next. La última ventana en la creación de nuestro proyecto en Quartus la podemos ver en la figura 3-5 y simplemente pulsamos Finish.

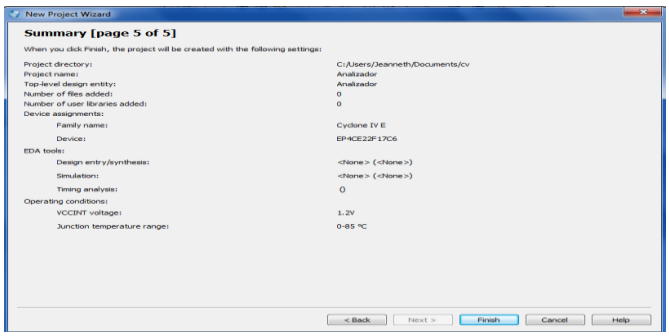


Figura 3-5 Finalización de la Creación de un Proyecto en Quartus II

3.1.3 Creación del Sistema en Qsys

Luego de haber creado el proyecto en Quartus II nos dirigimos a la herramienta de diseño de hardware asistido por computadora Qsys para el desarrollo de nuestro Sistema Embebido NIOS II. Para esto vamos a la pestaña Tools y seleccionamos la opción Qsys.

Qsys incluye módulos configurables como: el Procesador NIOS II, el Módulo UART, el Controlador del LCD, Interfaces Paralelas, etc., los cuales fueron usados para el diseño de nuestro proyecto.

Ahora, un sistema embebido funcional consta de Procesador, memoria, además de interfaces de entradas y salidas e interfaces de comunicación. Por lo cual nuestro sistema consta de las siguientes partes (ver figura 3-2):

- El Procesador NIOS II (CPU).
- Interfaz de comunicación con el computador JTAG UART
- Memoria On-Chip.
- Timer
- System ID

- Controlador de Memoria SDRAM
- Controlador de LCD
- Puertos paralelos para el manejo del Teclado hexadecimal
- 24 Módulos UART, para la recepción y análisis de datos provenientes de los dispositivos a analizar.
- 1 Módulo UART, para la comunicación entre la PC y/o Dispositivo Android.

A continuación mostraremos la instanciación de cada uno de estos módulos con sus respectivas configuraciones.

Instanciación del Procesador NIOS II

En la biblioteca Embedded Processors encontramos el NIOS II Procesor.

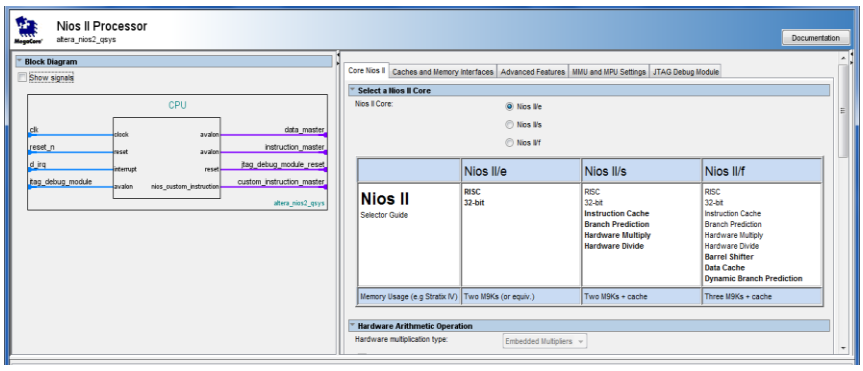


Figura 3-6 Asistente de Configuración el Procesador NIOS II

La tabla 3-1 nos muestra la configuración del procesador usada en nuestro proyecto, la mayoría de los valores se han quedado por defecto. Hay que tomar en cuenta que al momento de instanciar este módulo es necesario indicar cuales son los módulos de memoria donde se ubicaran los vectores de reset y excepciones, estos valores los dejamos sin configurar ya que aún no se ha instanciado ningún módulo de memoria

Core Nios II	Nios II Core	NiosII/e
Caches and Memory Interfaces	Instruction cache	4 Kbytes
	Burst transfers (burst size=32bytes)	Disable
	Number of tightly coupled instruction	None
	Data cache	2 Kbytes
	Data cache line size	4 Bytes
	Bursts transfers	Disable
	Number of tightly coupled data master	None
Advanced Features	Interrupt controller	Internal
	Number of shadow register sets (0-63)	0
MMU and MPU Settings	Process ID (PID) bits	10 Bits
	TLB entries	128 Entries
	TLB Set-Associativity	16 Ways
	Micro DTLB entries	6 Entries
	Micro ITLB	4 Entries
	Number of data regions	8
	Inimun data region size	4 Kbytes
	Number of instruction regions	8
	Minimun instruction region size	4 Kbytes
JTAG Debug Module	Debug level	Level 1

Tabla 3-1 Configuración del Procesador NIOS II

Instanciación del módulo System ID

En la biblioteca Peripherals encontramos la sub-biblioteca Debug and Performance ahí encontramos el módulo System ID peripheral.

Este módulo da una identificación al hardware e impide la descarga accidental de un software que no sea compilado para este sistema NIOS II específico.

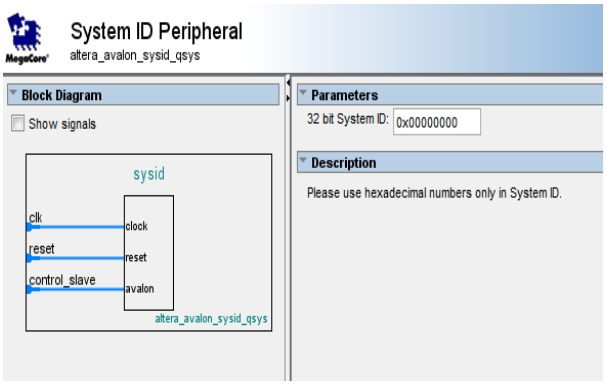


Figura 3-7 Asistente de Configuración de System ID

Un sistema computacional funcional requiere al menos una memoria para datos e instrucciones, a continuación se instancian los módulos de memoria de nuestro sistema.

Instanciación del controlador de la SDRAM

En la biblioteca Memories and Memory Controllers encontramos la sub-biblioteca External Memory Interfaces luego seleccionamos SDRAM Interfaces donde se encuentra el módulo SDRAM Controller. Este módulo se usa como controlador de la memoria SDRAM para establecer los valores del vector reset y de excepción del procesador NIOS II. Entonces una vez instanciado procedemos a efectuar esta configuración en el procesador.

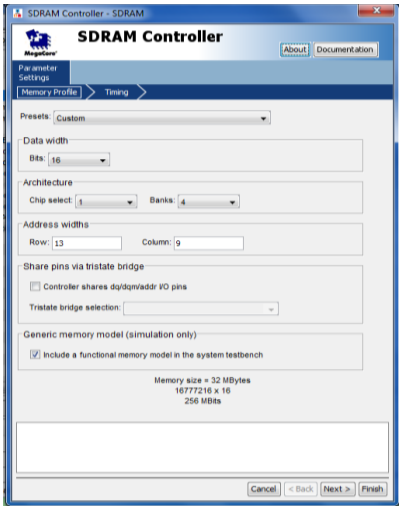


Figura 3-8 Asistente de Configuración del Controlador SDRAM

La tabla 3-2 nos muestra la configuración usada en el módulo SDRAM Controller de acuerdo a la memoria SDRAM que posee la tarjeta DE0 nano.

Memory Profile	Presets	Custom
	Bits	16
	Chip select	1
	Banks	4
	Row	13
	Column	9
Timing	CAS latency cycles	3
	Initialization refresh cycles	2
	Issue one refresh command every	15.625
	Delay after powerup, before initialization	100
	Duration of refresh command (t_rfc)	70
	Duration of precharge command (t_rp)	20
	ACTIVE to READ or WRITE delay (t_rcd)	20
	Access time (t_ac)	5.5
	Write recovery time	14

Tabla 3-2 Configuración del Módulo SDRAM Controller

Instanciación de Memoria On-Chip

En la biblioteca Memories and Memory Controllers encontramos la sub-biblioteca On Chip donde a su vez se encuentra el modulo On Chip Memory (RAM o ROM).

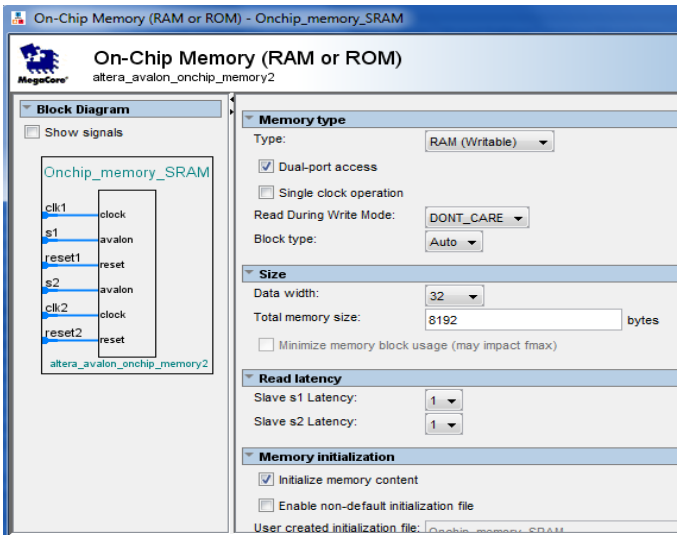


Figura 3-9 Asistente de Configuración de la Memoria On-Chip

La tabla 3-3 nos muestra la configuración del módulo Memoria On Chip usado en nuestro proyecto.

Type	RAM (Writable)
Dual-port Access	Si
Read During Write Mode	DONT_CARE
Block type	Auto
Data width	32
Total memory size	8192
Slave s1 Latency	1
Slave s2 Latency	1
Initialize memory content	Si

Tabla 3-3 Configuración del Módulo Memoria On Chip

Instanciación del JTAG UART

En la biblioteca Interface Protocols encontramos la sub-biblioteca Serial donde a su vez se encuentra el módulo JTAG UART.

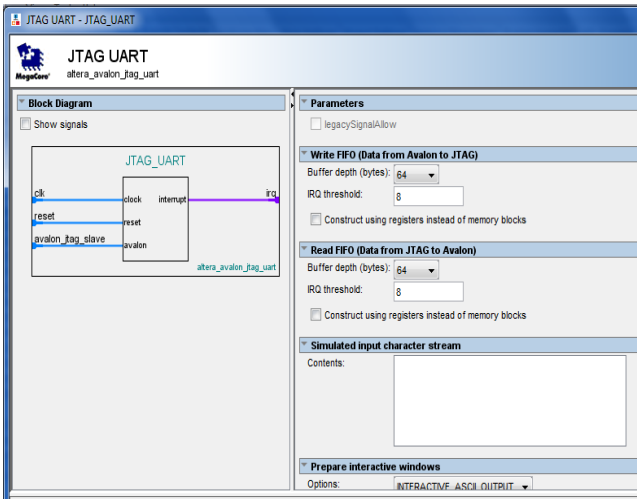


Figura 3-10 Asistente de Configuración del JTAG UART

La tabla 3-4 nos muestra la configuración usada en el módulo JTAG UART que sirve para la configuración de la FPGA y para la depuración del software que se ejecutara en el Procesador NIOS II.

Write FIFO	
Buffer depth (bytes)	64
IRQ threshold	8
Read FIFO	
Buffer depth (bytes)	64
IRQ threshold	8
Options	INTERACTIVE_ASCII_OUTPUT

Tabla 3-4 Configuración del Módulo JTAG UART

Instanciación del módulo Interval Timer

En la biblioteca Peripherals encontramos la sub-biblioteca Microcontroller Peripherals donde está el módulo Interval Timer.

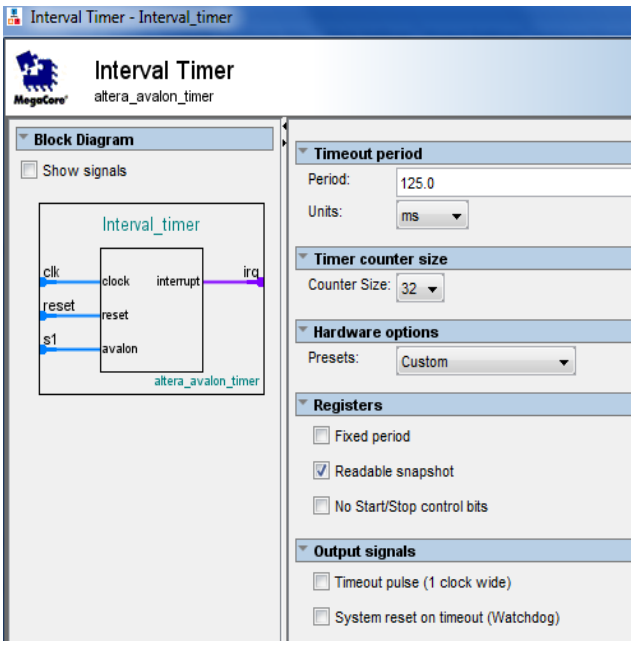


Figura 3-11 Asistente de Configuración del módulo Interval Timer

La tabla 3-5 facilita la configuración usada en el módulo Interval Timer.

Period	125.0
Units	Ms
Counter Size	32
Presets	Custom
Readable snapshot	Si

Tabla 3-5 Configuración del Módulo Interval TImer

Instanciación de los módulos UART (RS-232 Serial Port)

En la biblioteca Interface Protocols encontramos la sub-biblioteca Serial donde está el módulo UART (RS-232 Serial Port).

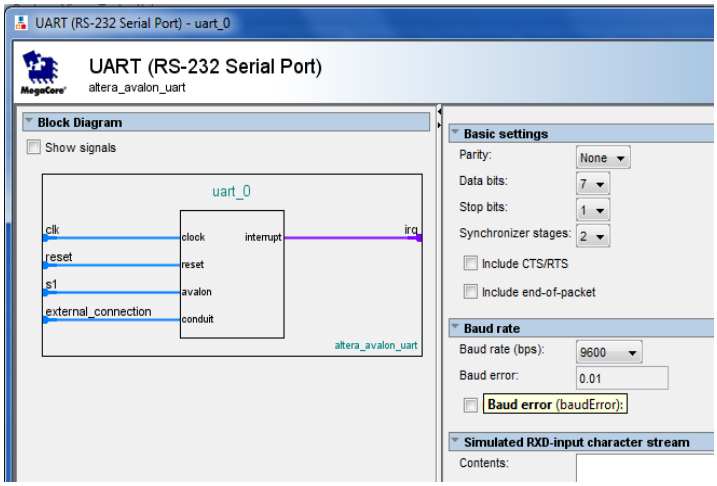


Figura 3-12 Asistente de Configuración del módulo UART

La ventaja de este módulo es que su velocidad de trasmisión puede ser modificada por software después de haber sido instanciada, por el contrario parámetros como paridad, tamaño de los datos, número de bits de parada; no pueden ser modificados después de haber sido instanciado, lo que limitaría a nuestro proyecto a analizar dispositivos comunicándose bajo un solo tipo de configuración. Para darle solución a este problema analizamos todas las combinaciones posibles de configuración. Para tamaño de datos: 7, 8, 9 bits; para la paridad: none, even y odd; y para el números de bits de parada: 1 y 2. Luego el número de combinaciones posible es 18.

Entonces para suplir la necesidad de analizar la comunicación entre dos dispositivos bajo cualquier tipo de configuración necesitaremos un bloque al que llamaremos UART 0 que está conformado por 18 módulos seriales para el análisis del dispositivo 1 y un bloque al que llamaremos UART 1 conformado otros 18 módulos seriales para el para el análisis del dispositivo 2. (Ver figura 3-2).

La tabla 3-6 muestra la configuración de cada módulo serial instanciado para suplir el análisis de dispositivos bajo cualquier tipo de configuración. Además muestra la abreviatura que representa a cada tipo de configuración.

UART 0	UART 1	Bits de Datos	Paridad	Bits de Parada	Abreviatura
uart_0	uart_18	7	None	1	7N1
uart_1	uart_19	7	None	2	7N2
uart_2	uart_20	8	None	1	8N1
uart_3	uart_21	8	None	2	8N2
uart_4	uart_22	9	None	1	9N1
uart_5	uart_23	9	None	2	9N2
uart_6	uart_24	7	Even	1	7E1
uart_7	uart_25	7	Even	2	7E2
uart_8	uart_26	8	Even	1	8E1
uart_9	uart_27	8	Even	2	8E2
uart_10	uart_28	9	Even	1	9E1
uart_11	uart_29	9	Even	2	9E2
uart_12	uart_30	7	Odd	1	7O1
uart_13	uart_31	7	Odd	2	7O2
uart_14	uart_32	8	Odd	1	8O1
uart_15	uart_33	8	Odd	2	8O2
uart_16	uart_34	9	Odd	1	9O1
uart_17	uart_35	9	Odd	2	9O2

Tabla 3-6 Configuraciones de los módulos UART

La instanciación de todos estos módulos provoca un error que no se había tomado en cuenta en el diseño, el Procesador NIOS II solo permite 32 fuentes de interrupción externa y para instanciar todos estos módulos se necesitan 36 ya que es necesaria una interrupción por cada módulo.

La situación es compleja ya que hay que disminuir la cantidad de interrupciones externas, es decir disminuir la cantidad de módulos seriales pero mantener la posibilidad de analizar todos los tipos de configuraciones.

La solución fue eliminar del uart_12 al uart_17 del bloque UART 0 y del uart_30 al uart_35 del bloque UART 1 correspondientes a la configuración de paridad odd. Las razones son simples: un dato puede ser recibido correctamente con un receptor configurado con la paridad even si el emisor la envió con la paridad odd, siempre y cuando los demás parámetros de configuración son los mismos en el receptor y emisor. Esto se debe a que el bit de paridad solo se usa para verificar la validez de la trama recibida. Por ejemplo, un emisor con configuración 7O1 quiere enviar el dato 11100100, entonces el trasmisor primero envía el bit de inicio que es un 0 lógico, luego el dato desde el bit menos significativo y luego envía el bit de paridad que es un 0 lógico en este caso ya que hay un

número par de unos, por ultimo añade el bit de parada que es un 1 lógico, entonces la trama completa es 00010011101. El receptor configurado con 7E1 detecta el bit de inicio, los siguientes 7 bits son de datos leídos desde el menos significativo por lo que el dato recibido es 11100100, es decir el mismo enviado, luego busca un 1 lógico como bit de paridad pero encuentra un 0 por lo que habilita la interrupción por error de bit de paridad y finalmente encuentra el 1 lógico como bit de parada. Fuera del error provocado por error en el bit de paridad el receptor configurado con 7E1 recibe exactamente el mismo dato enviado por un receptor configurado con 7O1. Entonces para configuraciones con paridad odd usamos la configuración even aunque nos dé una desventaja al provocar una interrupción por error de paridad los beneficios son mayores ya que reducimos a 24 la cantidad de interrupciones necesarias para el grupo de módulos seriales de nuestro analizador.

Cabe recalcar que cada bloque de módulos seriales va conectado a un solo pin de entrada de la FPGA, entonces a cada módulo de un mismo bloque de módulos seriales le llega la misma señal eléctrica, pero cada uno interpretara los datos de acuerdo a su configuración. El software se encarga de elegir un solo módulo de cada bloque para el análisis de los datos de acuerdo a la configuración de los dispositivos a analizar.

Adicionalmente añadimos el módulo uart_35 con configuración 8N1 y con una velocidad de trasmisión fija de 115200 baudios cuyo pin de trasmisión está conectado en forma paralela al módulo Bluetooth y UART-USB para enviar los datos analizados a una PC y/o dispositivo móvil Android para su respectiva visualización.

Instanciación del Controlador del LCD

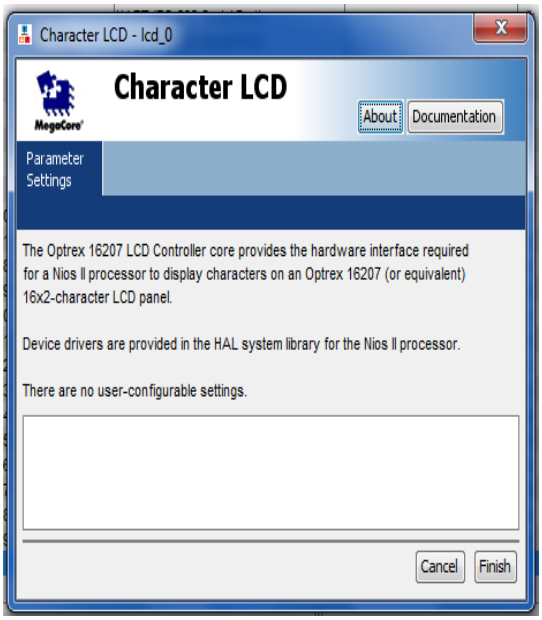


Figura 3-13 Asistente de Configuración del controlador LCD

La instanciación de este módulo no tiene mayor configuración y es usado como driver para el LCD de 16x2, el cual se usa para la interacción con el usuario, proporcionando un menú para elección de configuración manual o automática de nuestro analizador, además de mostrar la configuración a la que está trabajando nuestro analizador en determinado momento.

Instanciación de los Puertos Paralelos

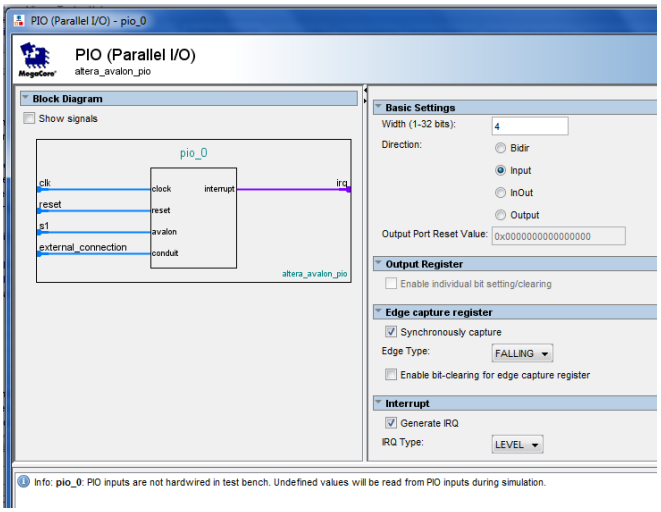


Figura 3-14 Asistente de Configuración del puerto paralelo

Se instanciaron 2 grupos de puertos paralelos de 4 bits cada uno, los cuales son usados para la manipulación del teclado Hexadecimal de 4x4.

La tabla 3-7 muestra la configuración de cada uno de los dos puertos paralelos.

	Puerto de entrada	Puerto de salida
Width(1-32 bits)	4	4
Direction	Input	Output
Synchronously capture	Si	Si
Edge Type	RISING	RISING
Generate IRQ	Si	-
IRQ Type	EDGE	-

Tabla 3-7 Configuración de los puertos paralelos

3.1.4 Generación del Sistema en Qsys

Hasta este punto se ha finalizado el diseño de hardware que se embebió en la FPGA (ver figura 3-15).

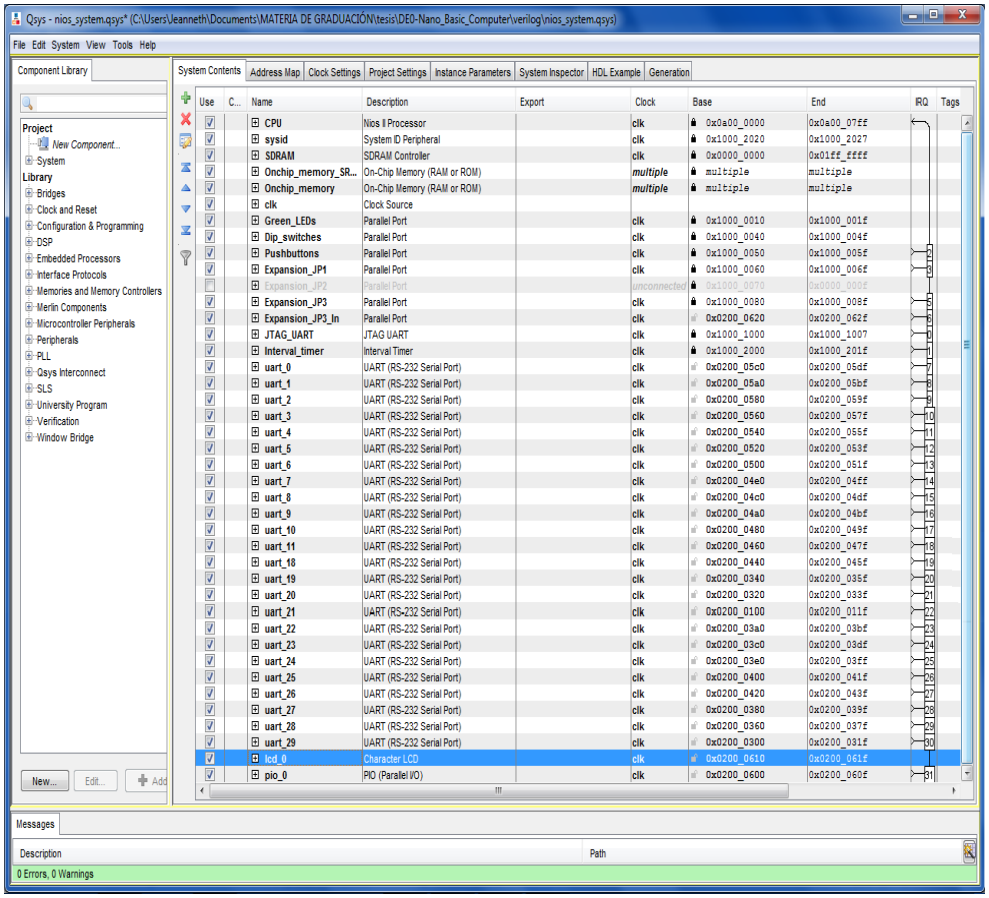


Figura 3-15 Diseño del sistema en QSYS

El siguiente paso es la generación del sistema, para esto hacemos click en la pestaña Generation de Qsys y luego en el botón Generate (ver figura 3-16).

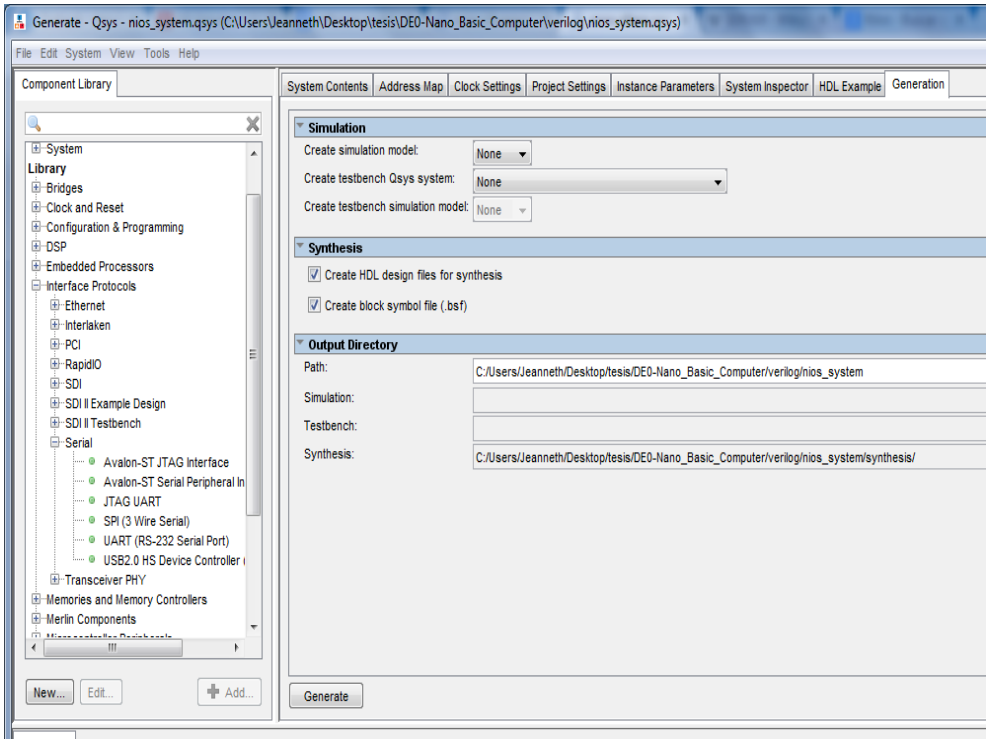


Figura 3-16 Ventana de generación del sistema en QSYS

La figura 3-14 nos muestra que la generación del sistema se realizó sin ningún error.

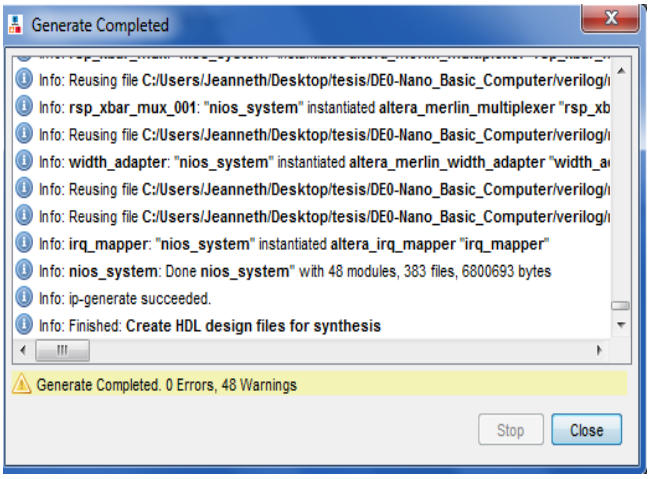


Figura 3-17 Generación correcta del sistema en QSYS

3.1.5 Instanciación del Sistema en Quartus II

Ahora es necesario integrar nuestro sistema creado en la herramienta Qsys con el proyecto creado en Quartus II anteriormente. Existen básicamente dos formas para realizar este paso: instanciar el sistema NIOS II usando el editor gráfico o usando algún tipo de lenguaje de descripción de Hardware como VHDL o Verilog. Este paso es muy importante ya que es aquí donde se declaran las señales de entrada y de salida de la FPGA que sirven para interactuar con el hardware adicional a la DE0 nano y con el usuario.

Nuestra elección fue usar el lenguaje Verilog. Para esto hacemos click en la pestaña File y hacemos click en New, nos aparecerá la ventana de la figura 3-18, seleccionamos la opción Verilog HDL File y pulsamos OK.

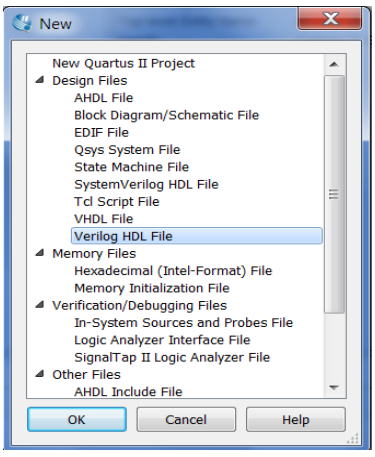


Figura 3-18 Creación de archivo Verilog en Quartus II

Luego le asignamos el nombre “Analizador” ponemos aceptar y nos aparece un editor de texto vacio donde escribimos el código necesario para la instanciación de nuestro sistema (ver anexo).

3.1.6 Asignación de Pines y Compilación del proyecto en Quartus

Luego de haber escrito y guardado el código necesario para la instanciación del sistema en Quartus II es necesaria la asignación de los pines en la FPGA así como los pines correspondientes en el puerto de expansión JP2 (ver figura 2-4) de la tarjeta DE0 nano, el cual se usa para la interacción del sistema embebido en la FPGA antes diseñado con el hardware adicional necesario para el funcionamiento correcto de nuestro proyecto (LCD, teclado numérico, UART-USB, Bluetooth).

FPGA	GPIO	DE0-NANO	SEÑAL	I/O	MODULO
PIN_T14	GPIO 12	5	UART_0_tx	Salida	UART 0
PIN_R13	GPIO 14	7	UART_0_rx	Entrada	
PIN_T12	GPIO 15	8	UART_1_rx	Entrada	UART 1
PIN_T13	GPIO 13	6	UART_1_tx	Salida	
PIN_M10	GPIO 128	35	DP_DATA[7]	Bidireccional	LCD
PIN_L14	GPIO 126	33	DP_DATA[6]	Bidireccional	
PIN_N15	GPIO 124	31	DP_DATA[5]	Bidireccional	
PIN_R14	GPIO 122	27	DP_DATA[4]	Bidireccional	
PIN_P15	GPIO 120	25	DP_DATA[3]	Bidireccional	
PIN_R16	GPIO 118	23	DP_DATA[2]	Bidireccional	
PIN_L16	GPIO 116	21	DP_DATA[1]	Bidireccional	
PIN_N9	GPIO 114	19	DP_DATA[0]	Bidireccional	
PIN_N12	GPIO 112	17	DP_E	Salida	
PIN_T10	GPIO 110	15	DP_RS	Salida	
PIN_P11	GPIO 18	13	DP_RW	Salida	

PIN_R11	GPIO 19	14	entrada[3]	Entrada	TECLADO
PIN_R10	GPIO 111	16	entrada[2]	Entrada	
PIN_P9	GPIO 113	18	entrada[1]	Entrada	
PIN_N11	GPIO 115	20	entrada[0]	Entrada	
PIN_K16	GPIO 117	22	salida[3]	Salida	
PIN_L15	GPIO 119	24	salida[2]	Salida	
PIN_P16	GPIO 121	26	salida[1]	Salida	
PIN_N16	GPIO 123	28	salida[0]	Salida	
PIN_J14	GPIO 133	40	Com_rx	Entrada	UART-USB Bluetooth
PIN_J13	GPIO 132	39	Com_tx	Salida	

Tabla 3-8 Asignación de pines

Para realizar esta operación hacemos click en la pestaña Assignments y luego en Pin Planner lo que nos muestra la ventana de la figura 3-19 y procedemos a efectuar la asignación mostrada en la tabla 3-8.

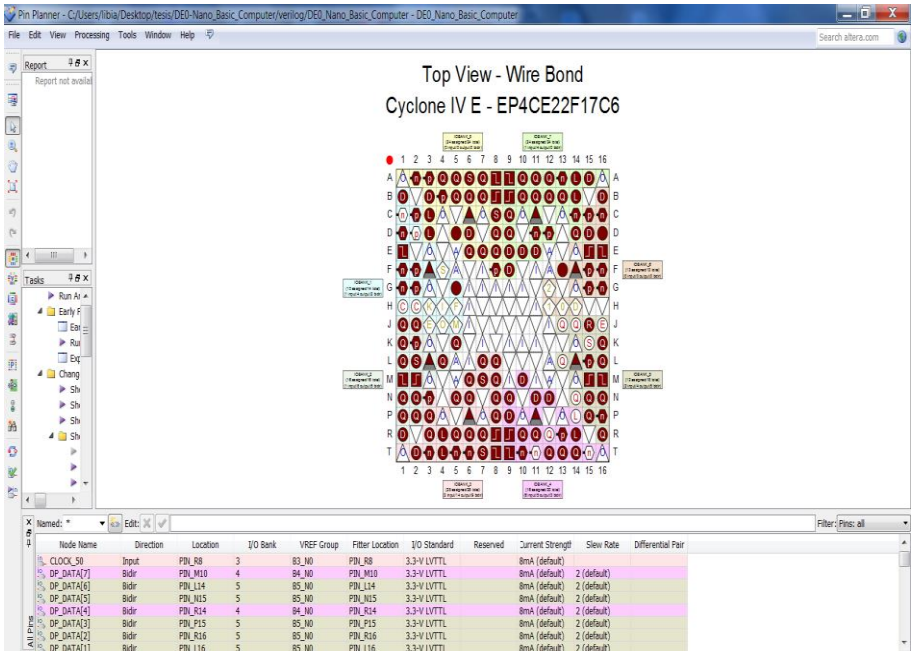


Figura 3-19 Pin Planner

3.1.7 Compilación y Programación del Hardware en la FPGA

El penúltimo paso en esta parte del diseño es la compilación de nuestro sistema en Quartus II. Para realizar esta operación vamos a la pestaña Processing y luego hacemos clik en Start Compilation.

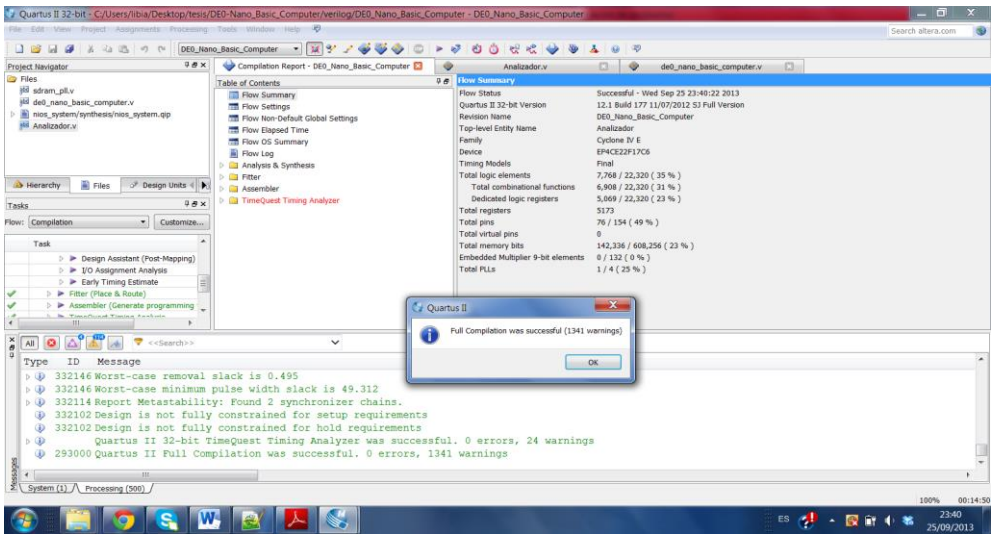


Figura 3-20 Compilación en Quartus II

La figura 3-20 nos muestra la compilación realizada de forma correcta, así como un resumen de los recursos de la FPGA usado por nuestro sistema. (Ver tabla 3-9).

Flow Status	Successful - Wed Sep 25 23:40:22 2013
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Full Version
Revision Name	DE0_Nano_Basic_Computer
Top-level	Entity Name Analizador
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	7,768 / 22,320 (35 %)

Total combinational functions	6,908 / 22,320 (31 %)
Dedicated logic registers	5,069 / 22,320 (23 %)
Total registers	5173
Total pins	76 / 154 (49 %)
Total virtual pins	0
Total memory bits	142,336 / 608,256 (23 %)
Embedded Multiplier	9-bit elements 0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)

Tabla 3-9 Reporte de Aprovechamiento de la FPGA

Luego de haber realizado la compilación de forma correcta se debe cargar el Hardware en la FPGA, entonces hacemos click en la pestaña Tools y hacemos click en la opción Programer y nos aparecerá la ventana de la figura 3-21. En esta ventana añadimos el archivo .sof que se va a cargar en la FPGA, hacemos click en Start asegurándonos que la PC haya reconocido el USB-Blaster. En esta ventana la opción Progress debe establecerse en Successful para confirmar que la programación ha sido efectuada de manera correcta.

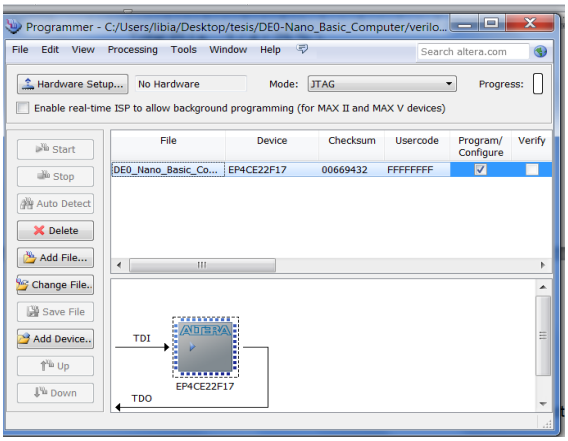


Figura 3-21 Programación de la FPGA

3.1.8 Diseño del Hardware que Complementa a la Tarjeta DE0-Nano

Como ya habíamos mencionado antes se va a usar el puerto de expansión JP2 de la tarjeta DE0 nano (ver figura 2-4) para la interacción con el teclado 4x4, LCD 16x2, módulo Bluetooth, módulo UART-USB y los dispositivos a analizar. Estos módulos se conectan a un PCB (circuito impreso) que se ha diseñado, el cual a su vez se conecta con la tarjeta DE0 nano a través de un bus de datos de 40 pines como se muestra en la figura 3-22.

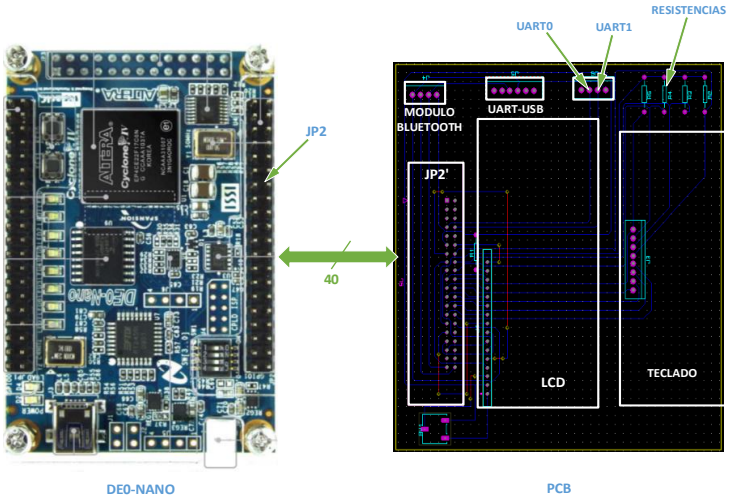


Figura 3-22 Diagrama de conexión del PCB y la DE0 nano

Con la ayuda de la tabla 3-8 que muestra la asignación de pines en la FPGA y en la Tarjeta DE0 nano hemos hecho un diagrama que muestra el detalle de las conexiones eléctricas de los módulos antes mencionados en el PCB. Como vemos el módulo Bluetooth y el LCD

necesitan de 5 voltios para su funcionamiento, además el teclado hexadecimal necesita de 3.3 voltios, estos voltajes salen de JP2' que es un conector hembra para el bus de datos que viene desde la DE0 nano la cual alimenta eléctricamente al PCB. (Ver figura 3-23)

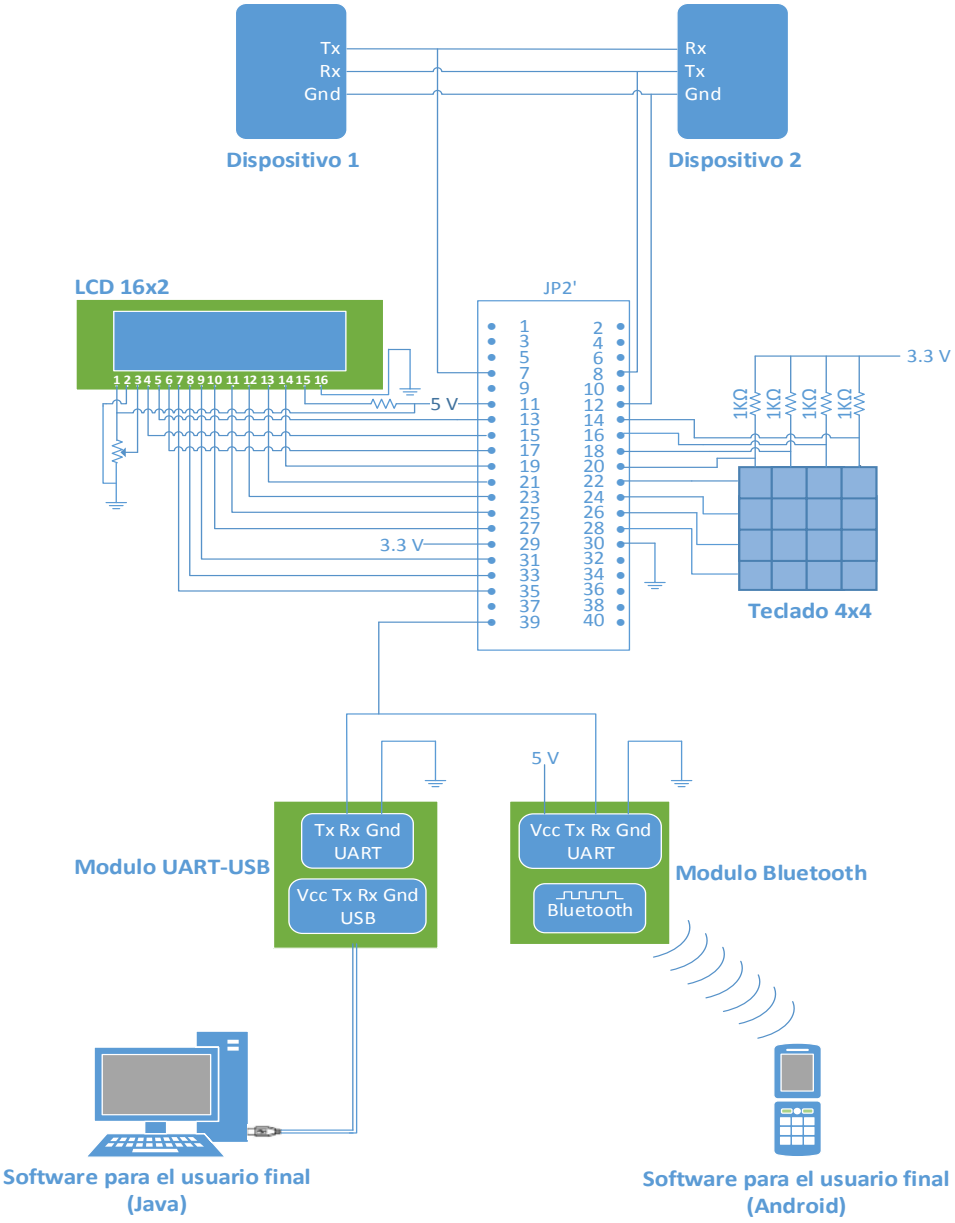


Figura 3-23 Diagrama de conexiones eléctricas

Teniendo el diagrama de conexiones ya establecido procederemos a realizar nuestro último paso en el diseño de hardware de nuestro proyecto. Usamos la herramienta ARES del programa de diseño y simulación Proteus para el diseño de nuestra PCB. El circuito impreso finalizado puede encontrarse en los anexos.

Finalmente la figura 3-24 muestra una fotografía de la implementación final del hardware de nuestro proyecto.

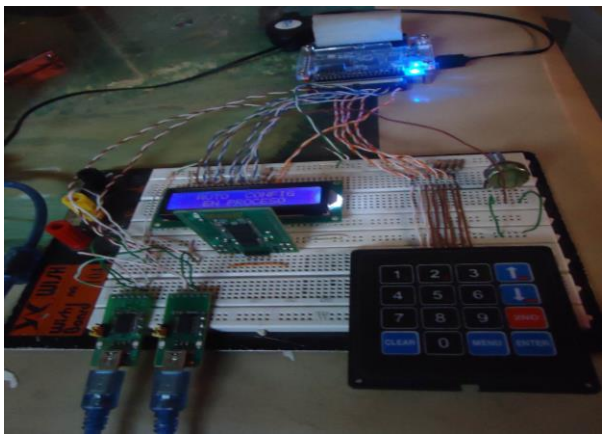


Figura 3-24 Implementación del hardware

3.2 Diseño de Software

Una vez terminado con el diseño de la parte física de nuestro proyecto lo siguiente es el diseño del software que administra dicho hardware. Esta sección está comprendida por: la programación del procesador que se la realizara usando la herramienta NIOS II IDE para Eclipse, el desarrollo

del software para el usuario final en Java y en Android usando NetBeans IDE.

3.2.1 Programación del procesador NIOS II

Ahora lo principal para establecer una comunicación RS-232 es que los dispositivos a comunicarse deben tener los mismos parámetros de configuración. Entonces para poder interceptar los datos que se están intercambiando dos dispositivos que están usando el protocolo de comunicación RS-232 es necesario que nuestro analizador este usando la misma configuración. Los parámetros de configuración son velocidad de trasmisión, tamaño en bits de datos, paridad y número de bits de parada.

Para esto nuestro analizador de protocolos tiene dos opciones: un modo de configuración manual y un modo de autoconfiguración.

En el modo de configuración manual al usuario se le solicitara que ingrese todos los parámetros de configuración de la comunicación.

En el modo de configuración automático el analizador auto configura todos los parámetros de configuración de la comunicación.

La figura 3-25 muestra el diagrama de flujo del programa principal del analizador. Como vemos primero pide que ingrese el modo de configuración si es automático o manual.

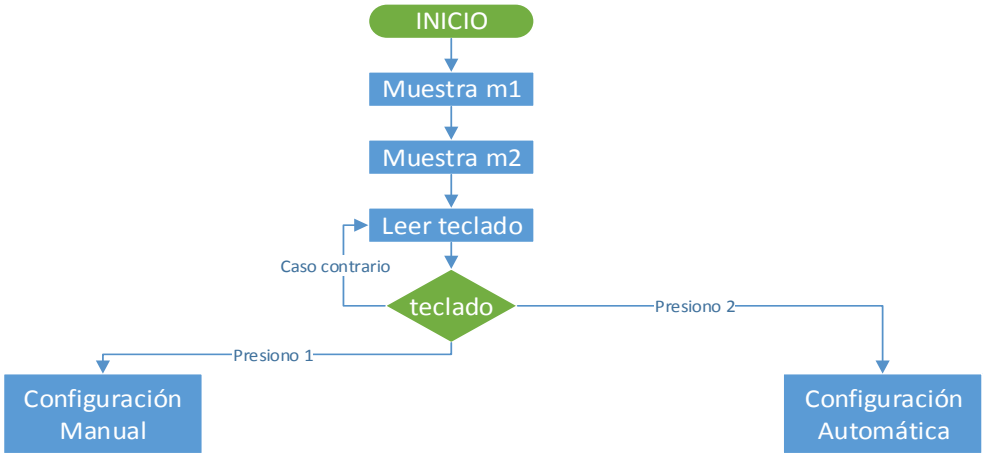


Figura 3-25 Flujo del Programa Principal NIOS II

El código que implementa este diagrama de flujos lo puede encontrar en el Anexo B.

3.2.1.1 Configuración Manual

En la opción configuración manual al usuario se le solicita primeramente que ingrese la velocidad de trasmisión, luego el tamaño en bits de los datos junto con la paridad y el número de bits de parada. Cada vez que un parámetro se ingresa se verifica si está correcto. Una vez ingresada la configuración. El analizador procede a interceptar los datos, cada vez que llega un dato, este es enviado vía Bluetooth y vía UART-USB para su visualización en un dispositivo Android o en una PC respectivamente. El flujo de esta sección se puede observar en la figura 3-26.

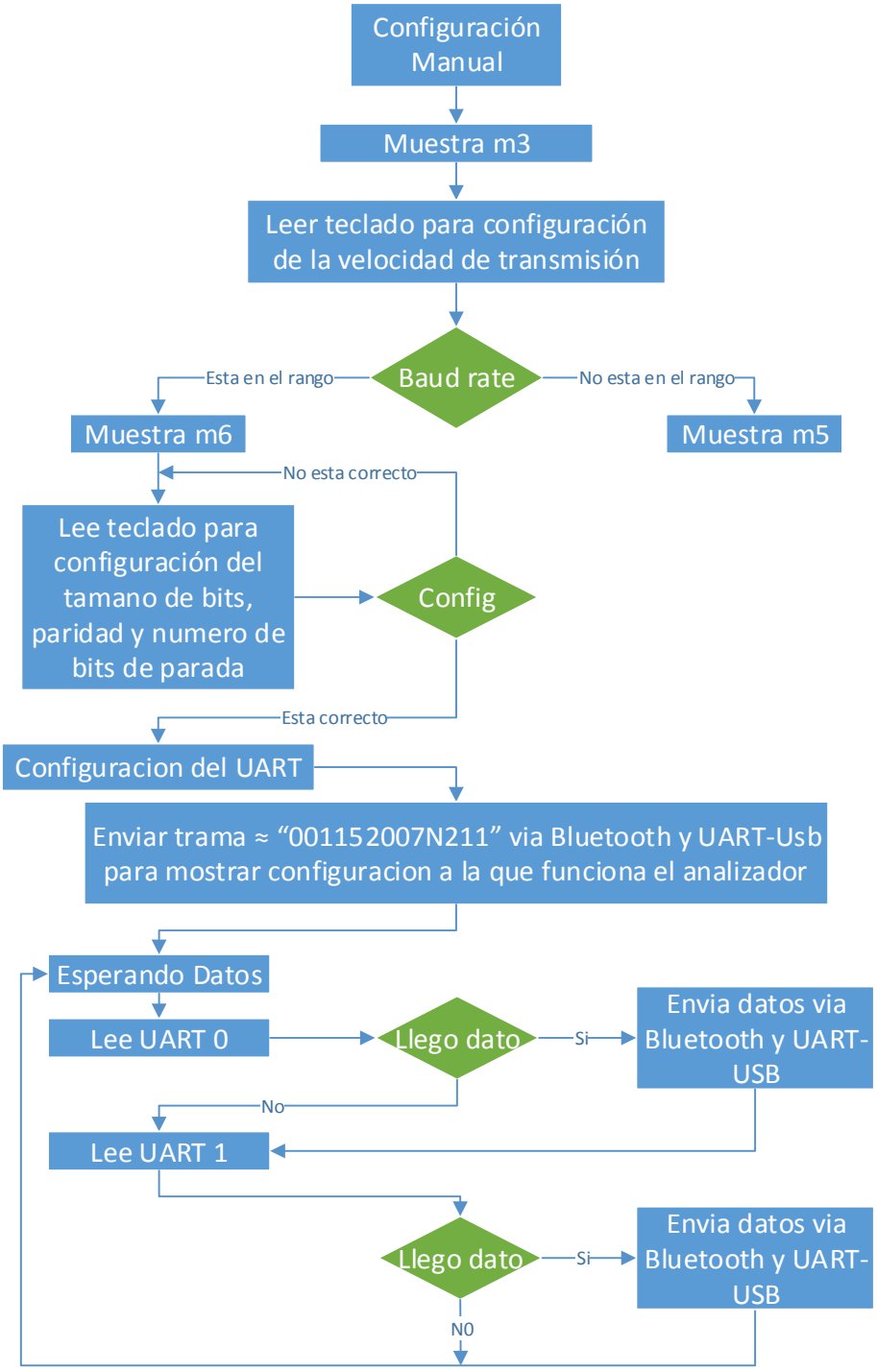


Figura 3-26 Flujo de la configuración Manual

El código que implementa este diagrama de flujos se encuentra en el Anexo B.

3.2.1.2 Configuración Automática

En la opción configuración automática el analizador tiene la capacidad de auto configurarse, para esto debe esperar que el o los dispositivos a analizar estén trasmitiendo datos para poder analizarlos y de esta forma determinar la velocidad con que se están trasmitiendo los bits y la configuración de tamaño de bits, paridad y parada. Para esta operación nos vamos a valer de algunas banderas o bits de error que posee nuestro módulo NIOS UART. Una de ellas es el bit BRK del registro STATUS (ver capítulo 2), o error de ruptura, este bit toma un papel fundamental en la determinación de la velocidad de trasmisión. Este bit se establece en 1 lógico cuando el del bit de inicio dura más de un ciclo completo de datos incluyendo bits de parada y bit de paridad.

Por ejemplo asumamos que nuestro dispositivo analizador está configurado con una velocidad de 19200 baudios y con configuración 8N1, esto quiere decir que un ciclo completo de datos es de tamaño 10 bits, además que los dispositivos a analizar tengan la misma configuración pero diferente velocidad de trasmisión como lo muestra la figura 3-27. En este caso el bit BRK se establecerá en 1 lógico en el ciclo 10 cuando

analicemos el dispositivo que este trasmitiendo a 1200 baudios ya que es el caso donde el bit de inicio dura más que un ciclo completo de bits. De esta manera este bit nos podría indicar que la velocidad a la que nuestro dispositivo analizador está leyendo los datos es mayor a la que se están trasmitiendo, por lo tanto debería leer los datos a una velocidad menor.

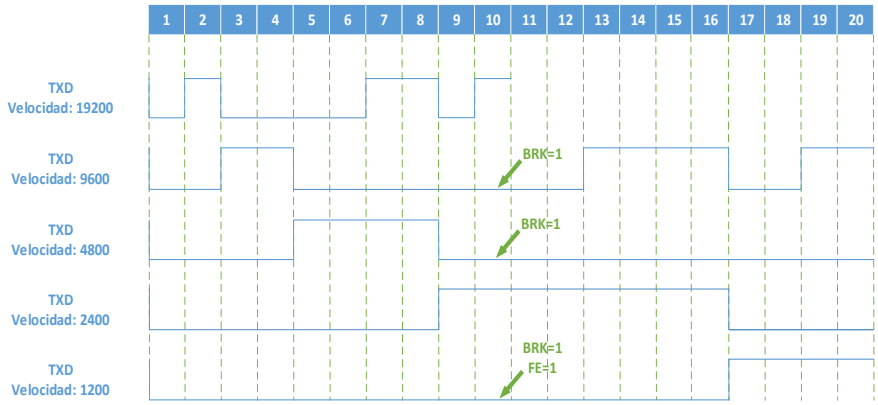


Figura 3-27 Análisis en el tiempo del bit FE y BRK

El otro bit fundamental en la autoconfiguración es el bit del registro status FE o error de trama, este bit se establece en 1 lógico cuando no se encuentra el bit de parada, para el ejemplo de la figura 3-27 el bit FE se establecerá en 1 lógico en el ciclo 10 cuando analicemos al dispositivo que está trasmitiendo a 9600, 4800 y 1200 ya que se esperaba un 1 lógico como bit de parada pero no se encontró. Este bit junto con el bit BRK nos servirá para indicar si la velocidad a la que está leyendo los datos nuestro analizador es la misma que la que se está

trasmitiendo, además nos indica incluso si la configuración a la que el analizador está leyendo los datos es la correcta. Además el analizador auto configura velocidades de trasmisión puntuales y estándares como lo son de mayor a menor 115200, 57600, 28800, 19200, 9600, 4800, 2400,1200 baudios.

Cuando se menciona UART0 [ ] en la figura 3-28 se refiere al grupo de UARTs que lo conforman (ver figura 3-2) además cuando se menciona UART0 [i] se refiere a un elemento en particular de ese grupo de UARTs.

Inicialmente en la configuración automática se analizan todos los elementos del UART0 y se establece la velocidad mayor, cuando llega un dato y un elemento de este grupo cumple con la cantidad máxima de errores (FE y BRK), se deshabilita, es decir, se descarta esa configuración. Si se han descartado todas las configuraciones se disminuye la velocidad a la siguiente velocidad estándar. Si se mantienen 4 UARTs habilitados por más de 10 datos recibidos, se escoge el primero de la lista de los 4 UARTs habilitados y la velocidad de trasmisión actual para la lectura de los datos y se muestra en el LCD las 4 posibles configuraciones a la que se comunican los dispositivos y la posible velocidad de trasmisión. Básicamente

este es el algoritmo de la autoconfiguración y se puede ver como un flujo en la figura 3-28

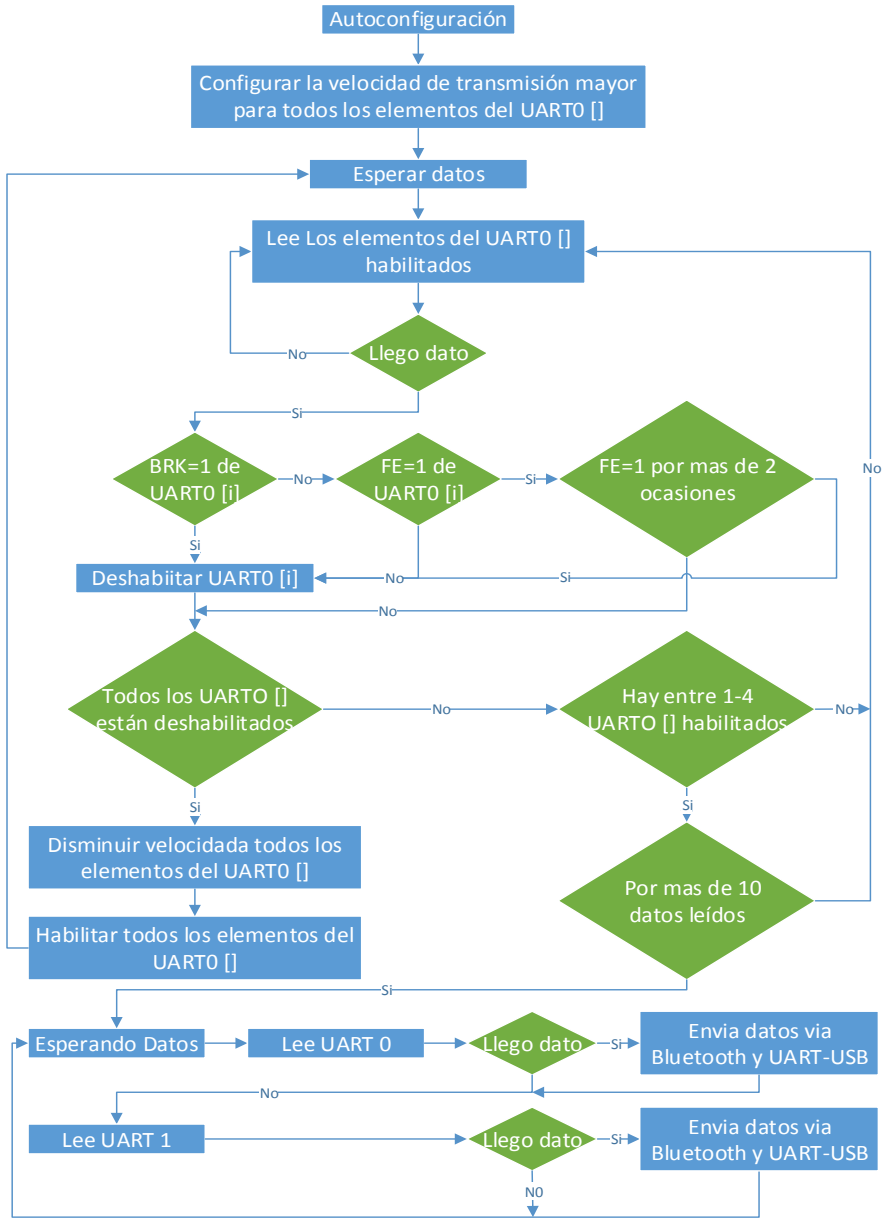


Figura 3-28 Flujo de la configuración automática

El código que implementa este diagrama de flujos lo puede encontrar en el Anexo B.

3.2.2 Desarrollo de Aplicación de usuario final en Java

Una de las opciones para la presentación de los datos analizados al usuario final es en una PC. Para esto se desarrolló una aplicación en el lenguaje de programación Java que muestra en una tabla los datos.

La recepción de los datos a mostrar se realiza a través de una conexión UART-USB (ver figura 3-2), la conexión con la PC es USB pero los datos llegan usando el protocolo RS232 con velocidad de trasmisión 115200 y configuración 8N1.

Java es un lenguaje de alto nivel, además no se manejan punteros, debido a estas razones se necesitan librerías adicionales para las comunicaciones con bases de datos, puertos seriales y paralelos.

Nuestra necesidad es el uso del puerto serial por lo que se añadió la librería RxTx (ver capítulo 2) para el acceso al puerto serial desde la aplicación desarrollada.

Para la interfaz gráfica se usó el editor gráfico del software Netbeans IDE y el estilo de programación que se usó fue el orientado a objetos. La figura 3-29 muestra el diagrama de flujos del programa

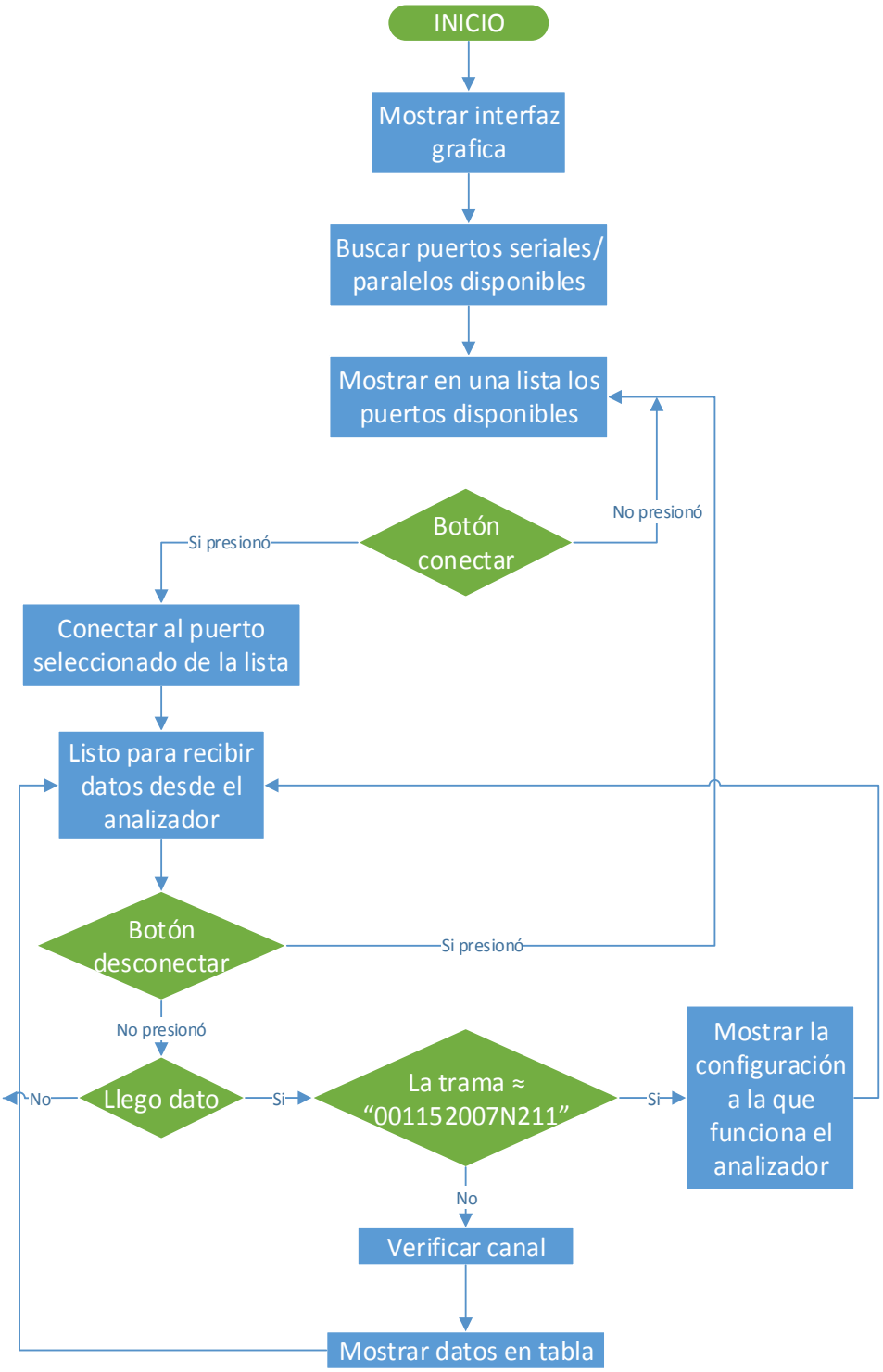


Figura 3-29 Flujo del Programa en Java

El código que implementa este diagrama de flujos lo puede encontrar en el Anexo C.

Finalmente la figura 3-30 muestra la interfaz gráfica de nuestro software finalizado.

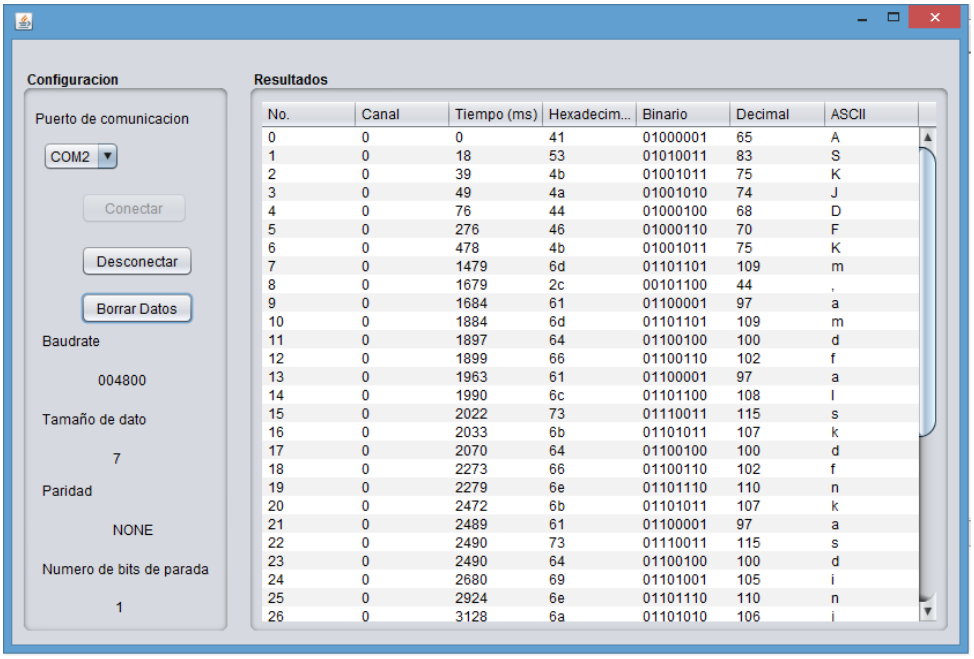


Figura 3-30 Interfaz gráfica del software en Java

3.2.3 Desarrollo de Aplicación de usuario final en Android

Para darle un toque de portabilidad a nuestro proyecto, una aplicación para un dispositivo móvil Android es la alternativa para mostrar los datos analizados los cuales se reciben vía Bluetooth desde el analizador.

Ahora, una aplicación en Android está formada por un conjunto de elementos básicos de visualización, conocidos como actividades. Estas actividades son las que controlan el ciclo de vida de las aplicaciones.

Nuestra aplicación tendrá una sola interfaz gráfica es decir una sola actividad, por tanto hablar del ciclo de vida de esta actividad es como hablar del ciclo de vida de nuestra aplicación. Para crear una aplicación estable en Android es necesario conocer perfectamente el ciclo de vida de una actividad.

Una actividad pasa por varios estados en su ciclo de vida, cuando existe un cambio de estado se dispara un evento el cual puede ser capturado por métodos o funciones definidos para dicha operación.

La figura 3-31 muestra claramente el método que se usa para capturar cada evento que se dispara dado un cambio de estado en la actividad dentro de su ciclo de vida.

Estos métodos pueden ser sobrescritos de acuerdo a la necesidad de la aplicación. Los métodos que sobrescribiremos en nuestra aplicación son: onCreate(), onResume() y onPause(), que son básicamente los más importantes en el desarrollo de una actividad.

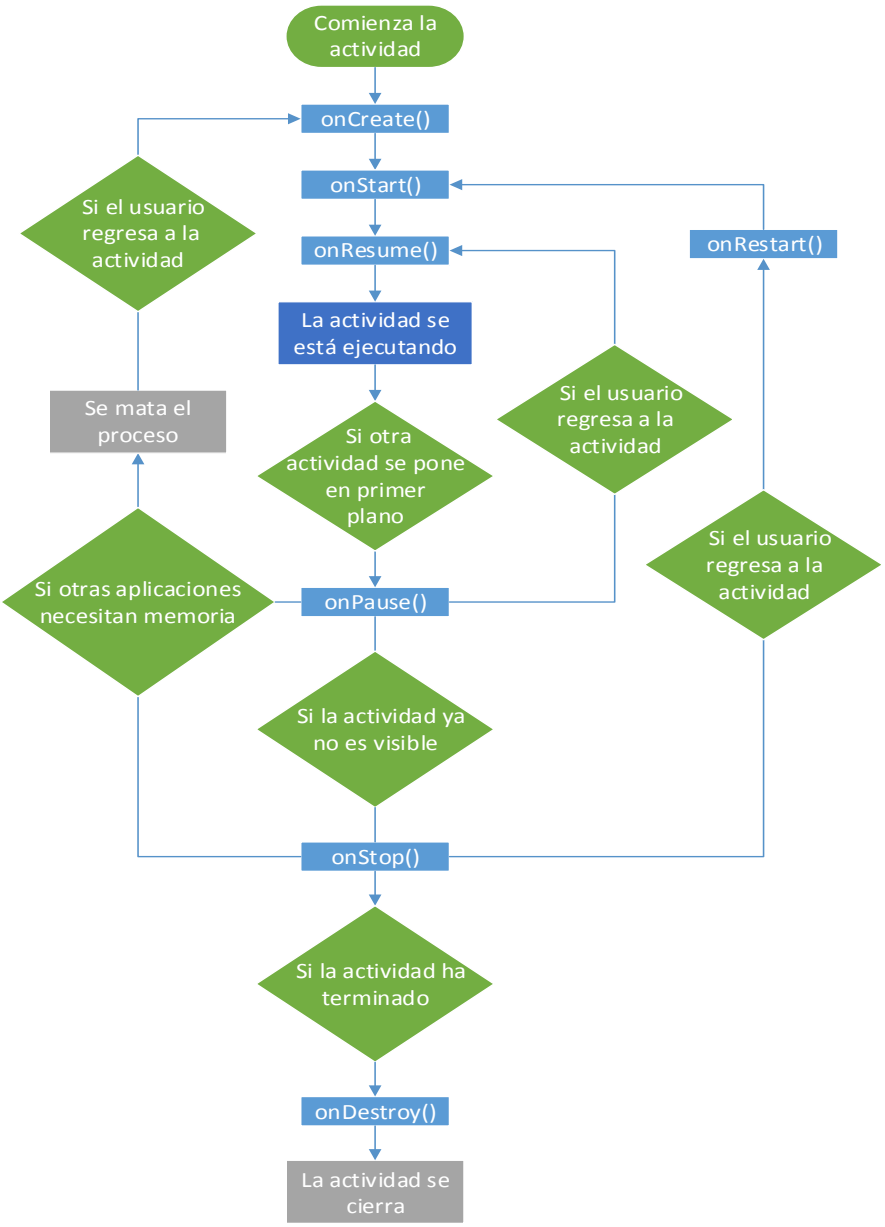


Figura 3-31 Ciclo de vida de una aplicación Android

El método onCreate() es llamado en la creación de la actividad. Lo usaremos para establecer la interfaz de usuario e inicializar las variables que manejaran dicha interfaz además de la verificación y la activación del Bluetooth del dispositivo, también se crea el

manejador para la estructura de datos que se usa para la recepción de la información vía Bluetooth.

La figura 3-32 muestra el flujo que tomara del método onCreate() en nuestro proyecto.

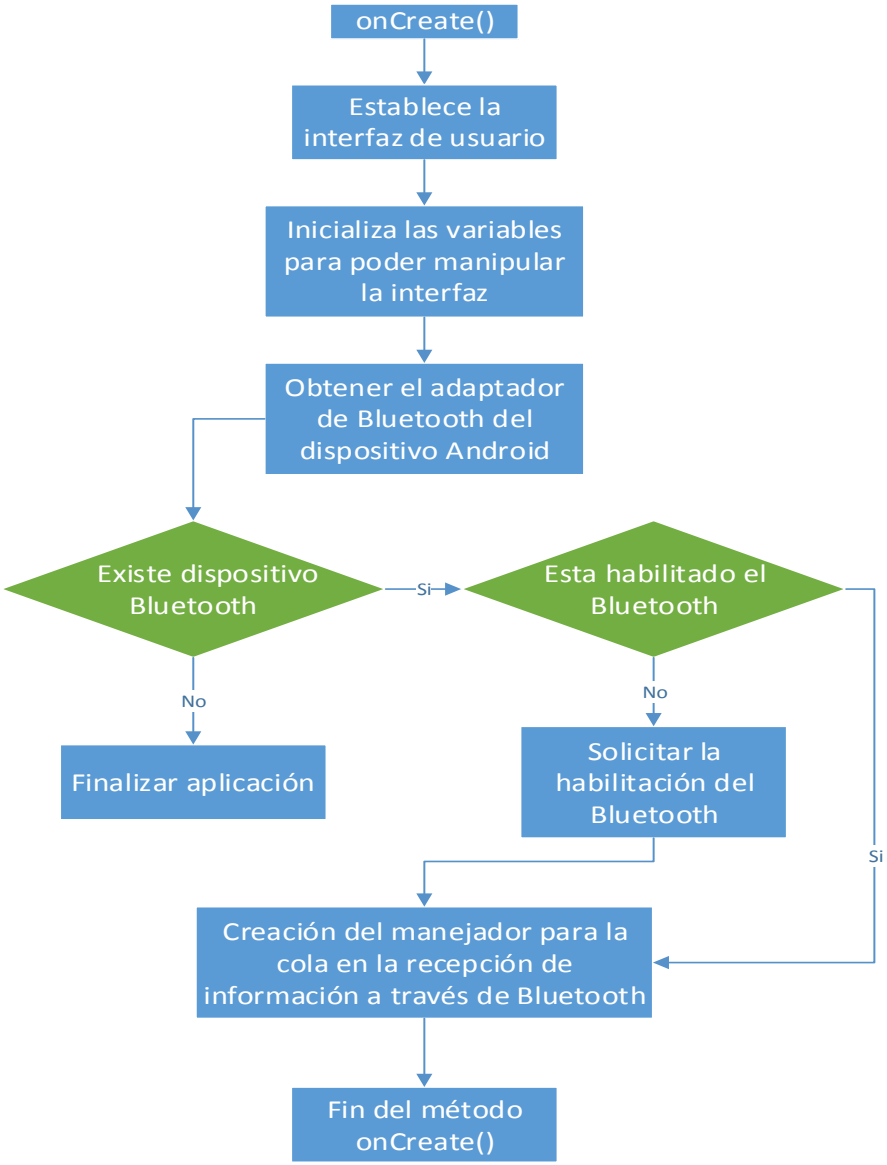


Figura 3-32 Flujo del método onCreate()

El método onResume() es llamado cuando la actividad ha terminado de cargarse en el dispositivo y comienza a interactuar con el usuario. En este método nuestra aplicación intenta establecer una conexión con el dispositivo Bluetooth de nuestro analizador y se crea el hilo de conexión para poder recibir o enviar información vía Bluetooth. La figura 3-33 muestra el flujo que toma del método onResume() en nuestro proyecto.

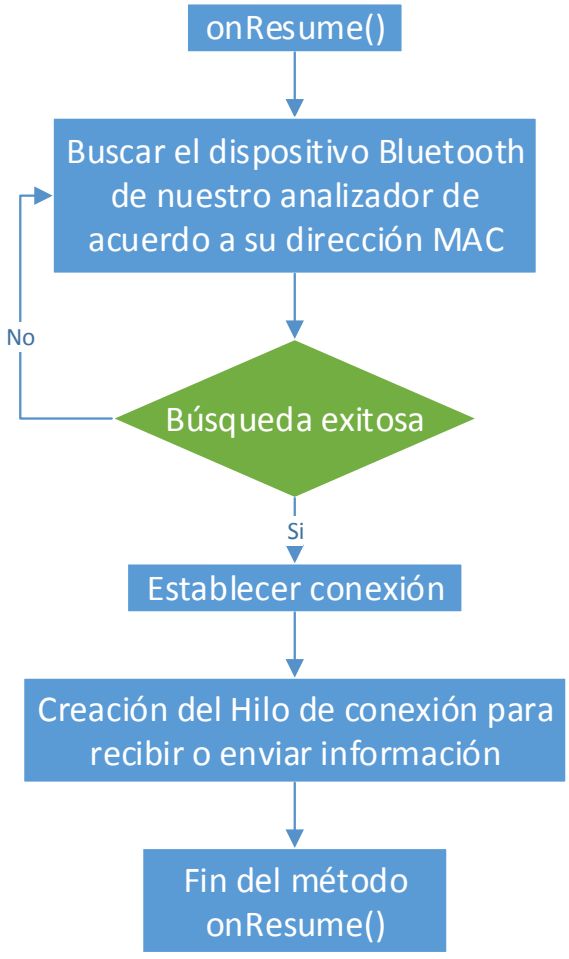


Figura 3-33 Flujo del método onResume()

La creación de un hilo de conexión se usa para que el proceso de lectura del flujo de entrada en la recepción vía Bluetooth sea constante e independiente del flujo del programa. El hilo de conexión es un bucle del que solo se sale si existe una excepción, la cual puede ser causada por cualquier fallo en la conexión Bluetooth. La figura 3-34 muestra el flujo que toma nuestro hilo de conexión.

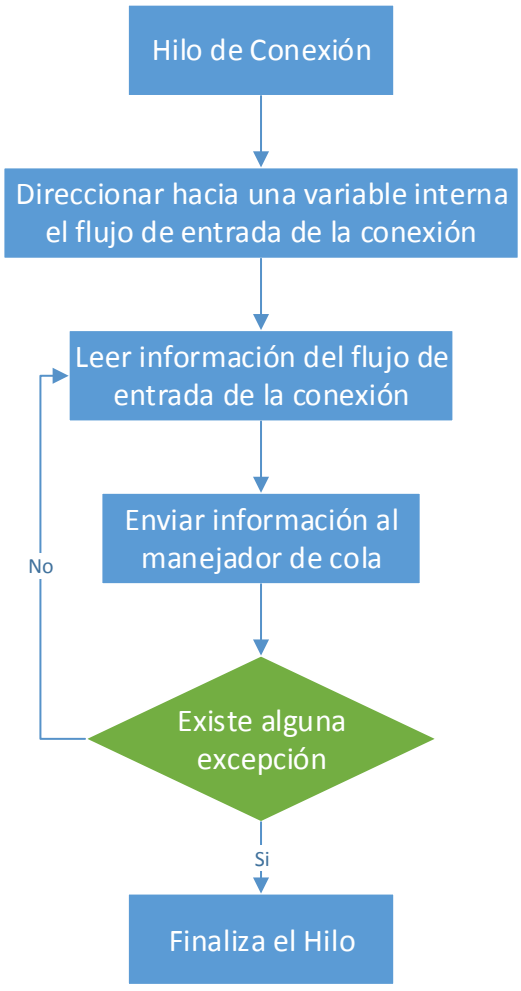


Figura 3-34 Flujo del hilo de conexión

Dentro del flujo del hilo encontramos un proceso en el que se envía la información al manejador de la estructura de datos, el cual fue creado al inicio de la aplicación. La estructura de dato que se usa para la recepción de la información vía Bluetooth es una cola que usa el método FIFO para la manipulación de los datos. La figura 3-35 muestra el flujo del manejador de la cola que se crea al recibir datos.

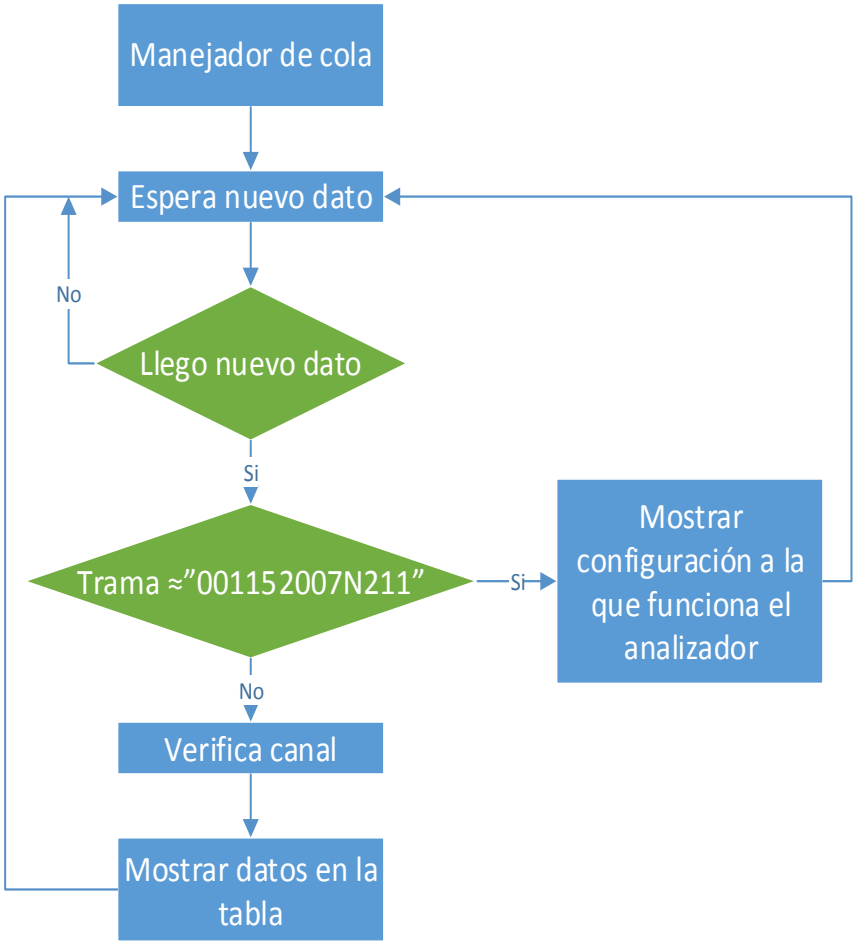


Figura 3-35 Flujo del manejador de cola

EL método onPause() es llamado cuando la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra aplicación es lanzada. En este método simplemente se cierra la conexión Bluetooth.

Finalmente la figura 3-36 muestra la interfaz gráfica finalizada de la aplicación Android y el código de dicha aplicación se encuentra en el Anexo D.

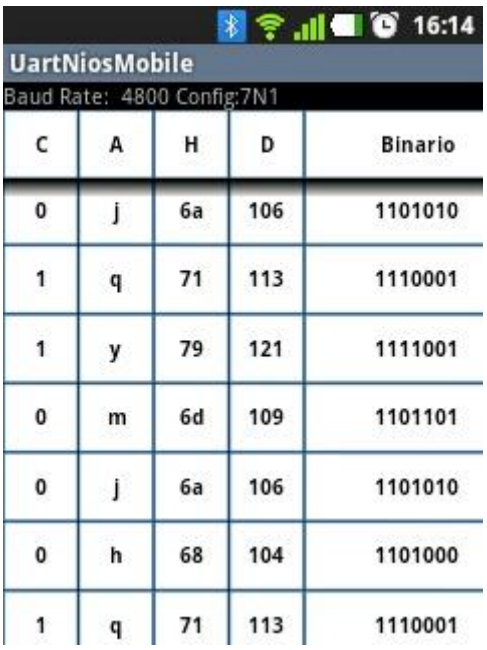


Figura 3-36 Interfaz gráfica de la aplicación en Android

CAPÍTULO 4

4. PRUEBAS Y RESULTADOS

Finalmente, en este capítulo se muestran algunas pruebas de funcionamiento del analizador de protocolo RS232, además de un análisis de la eficiencia del mismo.

El diagrama esquemático de todas las pruebas es el mismo, y como podemos observar en la figura 4-1, lo que cambia son los equipos que participan en la comunicación a analizar (dispositivo 1 y 2).

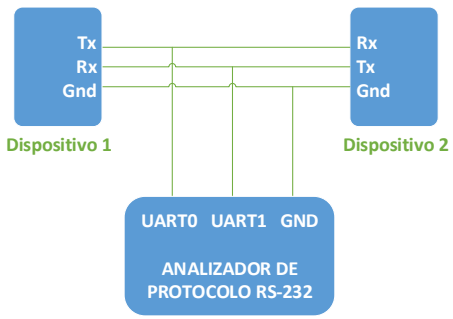


Figura 4-1 Diagrama de análisis

Los dispositivos que se usaron para realizar las pruebas son:

- 1 Tarjeta de desarrollo DE0 nano.
- 1 Tarjeta de desarrollo DE2.
- 2 Computadoras personales con el software AccesPort como terminales seriales.
- 1 Modulo de comunicación UART-Bluetooth.

Con estos dispositivos se efectuarán todas las combinaciones de pares posibles para efectuar el análisis de dichas comunicaciones.

4.1 Pruebas funcionales

El teclado Hexadecimal y el LCD 16x2 ayuda al analizador interactuar con el usuario. El LCD básicamente muestra mensajes para que el usuario pueda conocer el estado del analizador, el teclado nos ayudara a escoger las opciones que tiene el analizador, así como realizar la configuración manual del mismo. La secuencia de estos mensajes nos ayuda a conocer el flujo del programa principal de nuestro analizador.

4.1.1 Configuración manual

Al encender el analizador, se ejecuta el programa principal del procesador, el cual envía un mensaje al LCD, este mensaje es el

nombre del proyecto. La figura 4-2 nos muestra el mensaje mostrado en la prueba realizada.

Figura 4-2 Mensaje inicial del proyecto

El mensaje aparece por 3 segundos e inmediatamente solicita al usuario que escoja el tipo de configuración que se va a efectuar con el analizador, si es manual o automático. La figura 4-3 nos muestra el mensaje mostrado en la prueba realizada.

Figura 4-3 Menú del programa principal

La elección se la realiza presionando en el teclado hexadecimal el número 1 para la configuración manual o el número 2 para la automática. En esta prueba la elección fue la configuración manual. Luego se procede a solicitar primeramente que ingrese la velocidad de trasmisión a la que se va a analizar como lo muestra la figura 4-4.

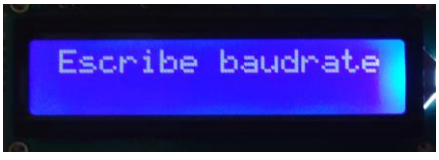


Figura 4-4 Ingreso manual de velocidad de trasmisión

En este ingreso se validó que el teclado solo reaccione a números para evitar errores de configuración, sin embargo el usuario aun puede ingresar velocidades errónea, sea está o muy baja o muy alta, como lo muestra la figura 4-5 de la prueba realizada. Luego de haber ingresado el valor el usuario debe presionar el botón entrar del teclado.

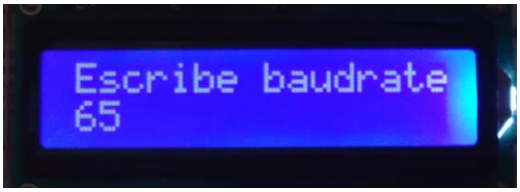


Figura 4-5 Ingreso de velocidad errónea

Al presionar el botón entrar del teclado se validó la velocidad de trasmisión y se mostró el mensaje de error de la figura 4-6, ya que la velocidad ingresada no está en el rango establecido que es mayor igual que 1200 y menor igual que 115200.

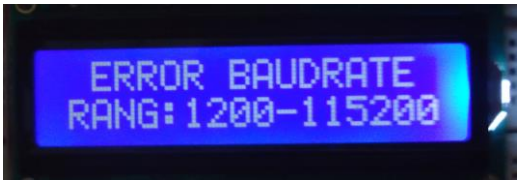


Figura 4-6 Mensaje de velocidad errónea

Luego de mostrar el mensaje de error se vuelve a solicitar el ingreso de la velocidad de trasmisión. Esta vez se ingresó el valor que muestra la figura 4-7.

Figura 4-7 Velocidad correcta

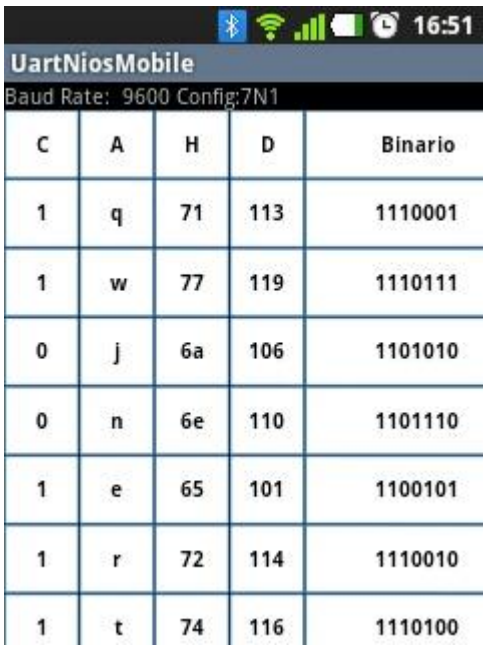
Ya que esta velocidad si está en el rango establecido se procederá a solicitar la configuración de tamaño de dato, paridad y número de bits de parada en el formato 8E1. Cuando se ingresa el primer carácter de la configuración se valida que solo reaccione y muestre en pantalla cuando se presione las teclas 7, 8 o 9 que son los tamaños de datos soportados por nuestro analizador, para el segundo carácter se validad que solo reaccione al presionar las teclas N, E u O ya que son las iniciales de None, Even y Odd respectivamente, que es la configuración de la paridad, finalmente para el último caracter está validado para que solo reaccione al presionar las teclas 1 o 2 que son la cantidad de bits de parada que se puede configurar. En nuestra prueba se usó la configuración 7N1 como lo muestra la figura 4-8.

Figura 4-8 Configuración del tamaño, paridad y bit de parada

Este fue el último paso en la configuración manual, luego de esto se muestra la configuración elegida en el LCD como se lo muestra en la figura 4-9.

Figura 4-9 Configuración manual terminada LCD

Además también se envía la misma información a la PC mediante conexión UART-USB y/o al dispositivo Android mediante conexión Bluetooth para la visualización en dichos dispositivos de la configuración usada por el analizador.



C	A	H	D	Binario
1	q	71	113	1110001
1	w	77	119	1110111
0	j	6a	106	1101010
0	n	6e	110	1101110
1	e	65	101	1100101
1	r	72	114	1110010
1	t	74	116	1110100

Figura 4-10 Visualización datos Android- Manual

Luego de haber configurado el Analizador está listo para interceptar los datos de la comunicación. La figura 4-10 y 4-11 muestra las interfaces de usuario en Android y Java respectivamente mostrando la configuración usada por el analizador, además de los datos interceptados.

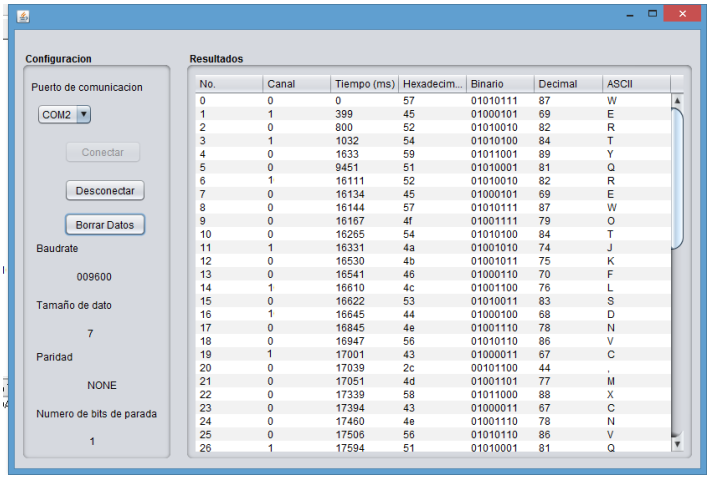


Figura 4-11 Visualización datos PC- Manual

4.1.2 Configuración Automática

Si cuando el analizador pide que escojamos el tipo de configuración, pulsamos el botón 2 la elección será la configuración automática, luego aparecerá la el mensaje de la figura 4-12.

Figura 4-12 Auto Configuración en proceso

Este mensaje indica que el analizador está receptando los datos desde el UART0 enviados desde el dispositivo 1 con el propósito de detectar los parámetros de configuración de la comunicación para poder auto configurarse. Este mensaje aparecerá hasta que el analizador pueda terminar con este procedimiento.

Figura 4-13 Auto Configuración Finalizada.

Una vez finalizado con el proceso de autoconfiguración, el analizador procederá a mostrar en el LCD los parámetros de configuración de la comunicación analizada, además también se envía la misma información a la PC mediante conexión UART-USB y/o al dispositivo Android mediante conexión Bluetooth para la visualización en dichos dispositivos de la configuración usada por el analizador. Las figuras 4-14 y 4-15 muestran las interfaces de usuario en Android y Java respectivamente mostrando la configuración usada por el analizador, además de los datos interceptados.

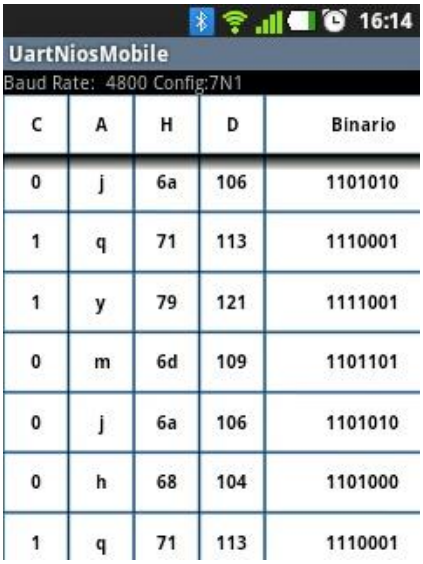


Figura 4-14 Visualización datos Android- Automático

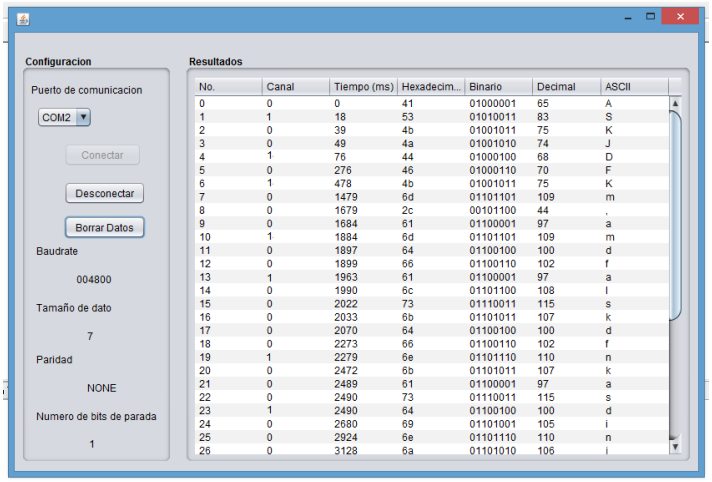


Figura 4-15 Visualización datos PC- Automático

4.1.3 Prueba del software para el PC

En esta sección se realizan pruebas de funcionamiento del software desarrollado en Java para el PC. Al iniciar el programa, se visualiza la interfaz gráfica y en la parte superior izquierda en una lista

desplegable los puertos seriales disponibles como lo muestra la figura 4-16.

Se debe seleccionar el puerto serial asignado para el módulo de comunicación UART-USB. Esta asignación la realiza automáticamente al momento de instalar el driver del módulo en la PC. Si usa el sistema operativo Windows con el cual se realizo esta prueba, el puerto se puede ver en el Administrador de dispositivos de Windows.

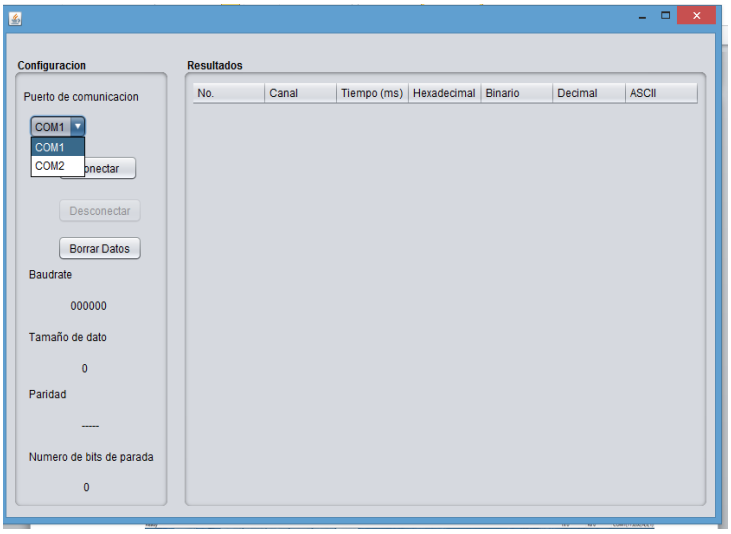


Figura 4-16 Inicio del software desarrollado para el PC

Luego presionar el botón CONECTAR, si el puerto está ocupado simplemente no se efectúa la conexión, si esta libre se efectúa la conexión, se deshabilita el botón CONECTAR y se habilita el botón DESCONECTAR como lo muestra la figura 4-17. Este paso siempre

se debería efectuar antes de que el analizador sea configurado, de forma manual o automática. Ahora el software está listo para recibir información a través del módulo de comunicación UART-USB.

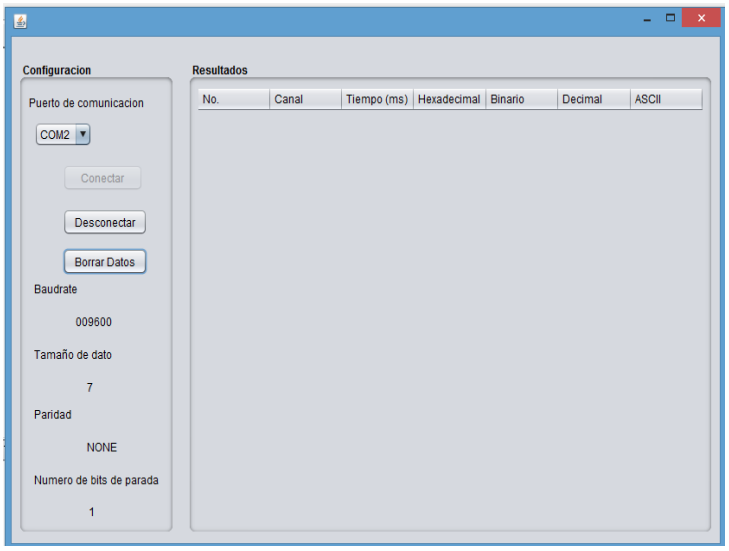


Figura 4-17 Conexión efectuada en el software para el PC

Una vez listo para recibir información, el software espera a que lleguen datos para mostrar. Lo primero que envía el analizador después de ser configurado de forma manual o automática es una trama con la información de la configuración que está usando. En la prueba llegó la trama 000048007N111, entonces el software muestra en su interfaz la velocidad de trasmisión igual a 4800 baudios, tamaño de bits igual a 7, paridad None y cantidad de bits de parada igual a 1. Luego espera que el analizador intercepte los datos de la comunicación y envié estos datos para su respectiva

visualización. El formato de presentación de los datos es una tabla con las siguientes columnas:

- No: Indica el número de dato que ha llegado al analizador.
- Canal: Indica que canal del analizador ha recibido el dato, canal 0 para el UART0 y canal 1 para el UART1.
- Tiempo: Indica el tiempo de recepción de un determinado dato contado desde el primer dato recibido.
- Hexadecimal: El dato analizado o interceptado en el sistema Hexadecimal.
- Binario: El dato analizado o interceptado en el sistema Binario.
- Decimal: El dato analizado o interceptado en el sistema decimal
- ASCII: El dato analizado o interceptado en el formato de un carácter.

La figura 4-18 muestra la interfaz gráfica del programa, mostrando la configuración del analizador y los datos analizados.

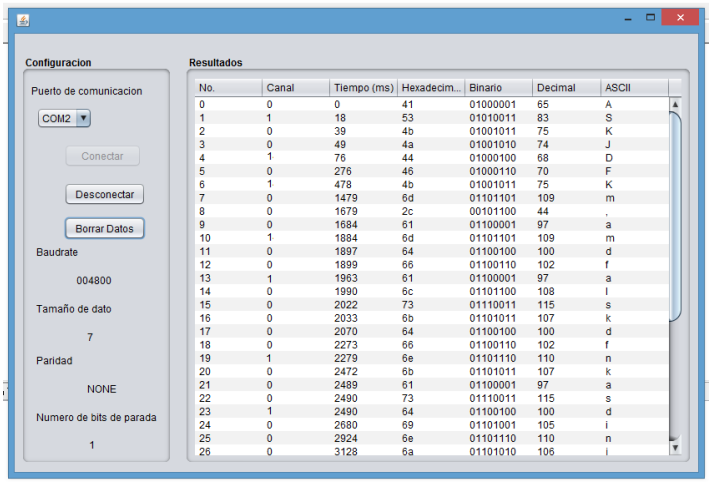


Figura 4-18 Visualización de datos y configuración en el software para el PC

4.1.4 Prueba de la aplicación Android

En esta sección se realizan pruebas de funcionamiento de la aplicación desarrollada para el dispositivo Android. La aplicación le dará un toque de portabilidad al analizador ya que la visualización de datos será a través de un dispositivo móvil. Esta prueba se la realizó en el teléfono inteligente LG-L3. Al inicial la aplicación, primero verifica que el teléfono soporte la tecnología Bluetooth, si no la soporta simplemente se cierra la aplicación. Si esta verificación fue exitosa, luego verifica si está habilitado el Bluetooth, sino pide habilitarlo como lo muestra la figura 4-19.

Figura 4-19 Solicitud de permiso de Bluetooth - Android

La elección a la pregunta de habilitación o activación, debe ser afirmativa, de no serlo no habría comunicación con el analizador. Luego de haber pulsado en el botón SI, saldrá el mensaje de estado de la figura 4-20.

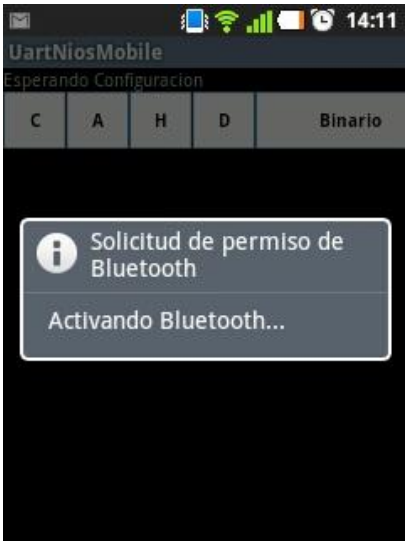


Figura 4-20 Activando Bluetooth – Android

Una vez activado el Bluetooth la aplicación procederá a intentar conectarse al dispositivo Bluetooth del analizador usando su dirección MAC: 00:13:01:04:07:89, como lo muestra la figura 4-21. Para la conexión correcta el analizador no debe estar a una distancia mayor a los 10 metros. Al igual que la aplicación desarrollada para el PC este paso siempre se debería efectuar antes de que el analizador sea configurado, ya sea de forma manual o automática. Una vez realizado este paso la aplicación esta lista para recibir la información del analizador.

Figura 4-21 Conexión a 00:13:01:04:07:89 - Android

La prueba de esta aplicación se la realizó a la par del software desarrollado para el PC por lo que también recibió la trama 000048007N111, la figura 4-22 muestra que se actualizó la interfaz con la configuración, además de datos recibidos que han sido

mostrados con el formato de una tabla muy parecida a la mostrada por el software desarrollado para el PC. Las columnas de dicha tabla son:

- C: Indica que canal del analizador ha recibido el dato, canal 0 para el UART0 y canal 1 para el UART1.
- A: El dato analizado o interceptado en el formato de un carácter.
- H: El dato analizado o interceptado en el sistema Hexadecimal.
- D: El dato analizado o interceptado en el sistema decimal.
- Binario: El dato analizado o interceptado en el sistema Binario.

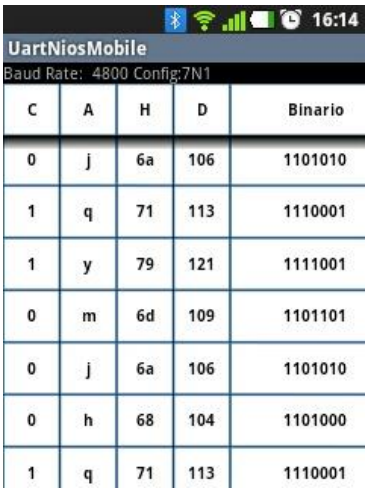


Figura 4-22 Visualización de datos y configuración- Android

4.2 Análisis estadístico de las pruebas realizadas

En esta sección se muestra la tabla de resultados de las pruebas realizadas para determinar la efectividad del analizador para detectar los parámetros de configuración de una comunicación y auto configurarse.

Al inicio de este capítulo habíamos mencionado los dispositivos disponibles para realizar estas pruebas, se han hecho todas las combinaciones de pares posibles para el análisis de dichas comunicaciones. La tabla 4-1 nos muestra todas las pruebas realizadas.

Para presentar los resultados de efectividad del analizador se usan tablas, para mayor entendimiento de las mismas describiremos lo que representa cada columna:

- Comunicación: Los dispositivos que participan en la comunicación.
- Vel. Real: La velocidad a la que se configuró los dispositivos a analizar.
- Conf. Real: La paridad, tamaño de datos y número de bits de parada a la que se configuró los dispositivos a analizar.
- Vel. Ok: El analizador auto configuró la velocidad correctamente.
- Conf. Ok: El analizador auto configuró correctamente el tamaño de datos, paridad y número de bits de parada.

- No. datos: Numero datos que se enviaron en la comunicación hasta auto configurarse.

- Posibles configuraciones: El analizador muestra un grupo de configuraciones posibles, en el caso de que la usada para la auto configuración no sea la correcta.

- Datos ok: los datos que se intercambian en la comunicación analizada son recibidos e interpretados correctamente por el analizador después de auto configurarse.

Comunicación	Vel. Real	Conf. Real	Vel. Ok	Conf. Ok	No. Datos	Posibles Configuraciones	Datos Ok
DE2-PC	115200	7N1	SI	SI	6x20	7N1, 7N2, 8N1, 8N2	SI
	115200	7N2	SI	NO	8x20	7N1, 7E2, 8N1, 8N2	SI
BLUETOOTH-PC	115200	8N1	SI	SI	10x20	8N1, 8N2, 7N1, 7N2	SI
PC-PC	115200	8N2	SI	SI	15x20	8N1, 8N2, 9N1, 7E1	SI
	115200	7E1	SI	SI	4x20	8N1, 8N2, 7E1, 7E2	SI
	115200	7E2	SI	NO	12x20	8N1, 8N2, 8E1, 8E2	SI
	115200	7O1	SI	NO	10x20	8N2, 7E2	SI
	115200	7O2	SI	NO	15x20	8N1, 8N2, 8E1, 8E2	NO
	115200	7N1	SI	SI	4x20	7N1, 7N2, 8N1, 9E1	SI
BLUETOOTH-PC	115200	8N1	SI	SI	4x20	8N1, 8N2, 7E1, 7E2	SI
DE2-PC	57600	7N1	SI	SI	6x20	7N1, 7N2, 9E2, 8E2	SI
	57600	7N2	SI	NO	10x20	7N1, 7E2, 8N1, 8N2	SI
BLUETOOTH-DE2	57600	8N1	SI	SI	7x20	8N2, 7E1, 7E2, 8N1	SI
PC-PC	57600	8N2	SI	SI	15x20	8N1, 8N2. 8E1, 8E2	SI
	57600	7E1	SI	SI	9x20	8N1, 8N2, 7E1, 7E2	SI
	57600	7E2	SI	NO	20x20	8E2, 8N2, 9N1, 9N2	NO
	57600	7O1	SI	NO	10x20	8N1, 8N2, 7E1, 7E2	NO
	57600	7O2	SI	NO	15x20	8N1, 8N2. 8E1, 8E2	NO
	57600	7N2	SI	SI	12x20	7N1, 7N2, 8N1, 8N2	SI
	57600	8E1	SI	SI	10x20	9N1, 9N2, 8E1, 8E2	SI

DE0 nano - DE2	57600	9N1	SI	SI	10x20	9N1, 9N2, 8E1, 8E2	SI
	57600	9E 1	SI	SI	20x20	9E1, 9E2	SI
	57600	9N2	SI	NO	10x20	9E1, 9E2	NO
	57600	9E 2	SI	NO	15x20	9N1, 9N2, 8E1, 8E2	NO
	57600	9O1	SI	SI	15x20	9E1, 9E2, 9O1, 9O2	SI
DE2-PC	28800	7N1	SI	SI	4x20	7N1, 7N2, 9E1, 9E2	SI
	28800	7N2	SI	SI	14x20	7N1, 7E2, 8N1, 8N2	SI
BLUETOOTH-PC	28800	8N1	SI	SI	11x20	8N1, 8N2, 7E1, 7E2	SI
PC-PC	28800	8N2	SI	NO	20x20	8N1, 7E2. 8E1, 8E2	SI
	28800	7E1	SI	SI	11x20	8N1, 8N2, 7E1, 7E2	SI
	28800	7E2	SI	NO	25x20	8N1, 8N2, 8E1, 8E2	NO
	28800	7O1	SI	NO	24x20	7E1, 7E2	SI
	28800	8O1	SI	NO	13x20	9N1, 9N2, 8E1, 8E2	SI
	28800	8N2	SI	NO	25x20	8N1, 7E2, 8E1, 8E2	SI
	28800	8E1	SI	SI	12x20	9N1, 9N2, 8E1, 8E2	SI
DE2-PC	19200	7N1	SI	SI	6x20	7N1, 7N2, 9E1, 9E2	SI
	19200	7N2	NO	NO	7x20	9E1, 9E2	NO
BLUETOOTH-DE0 nano	19200	8N1	SI	NO	21x20	8N2, 7E1, 7E2	SI
PC-PC	19200	8N2	SI	NO	30x20	8N1, 7E2, 8E1, 8E2	SI
	19200	7E1	SI	SI	20x20	8N2, 7E1, 7E2	SI
	19200	7E2	SI	SI	32x20	8N1, 7E2, 8E1, 8E2	SI
	19200	7O1	SI	NO	20x20	8N1, 8N2, 7E1, 7E2	NO
	19200	8O1	SI	NO	19x20	8E1, 8E2	SI
	19200	7N2	NO	NO	4x20	9E1, 9E2, 9N1	NO
	19200	8N1	SI	NO	14x20	7E2, 8N2, 7E1	SI
DE0 nano-PC	9600	7N1	SI	SI	5x20	7N1, 7N2, 9E1, 9E2	SI
	9600	7N2	SI	NO	45x20	7N1, 7E2, 8N1, 8N2	SI
BLUETOOTH-DE2	9600	8N1	SI	NO	18x20	7E1, 7E2	SI
PC-PC	9600	8N2	SI	SI	17x20	8N1, 8N2, 8E1, 8E2	SI
	9600	7E1	SI	SI	21x20	8N1, 8N2, 7E1, 7E2	SI
	9600	7E2	SI	NO	18x20	8N1, 8N2, 8E1, 8E2	NO
	9600	7O1	SI	NO	25x20	7E1, 7E2	SI
	9600	8O1	SI	NO	14x20	9N1, 9N2, 8E1, 8E2	SI
	9600	7E2	SI	NO	30x20	8N1, 8N2, 8E1, 8E2	SI
BLUETOOTH-DE2	9600	8N1	SI	NO	29x20	7E2, 8N2, 7E1	SI
DE0 nano-PC	4800	7N1	SI	SI	3x20	7N1, 7N2, 8N1, 8N2	SI
	4800	7N2	SI	SI	15x20	7N1, 7N2, 8N1, 8N2	SI
BLUETOOTH-DE0 nano	4800	8N1	SI	SI	10x20	8N1, 8N2, 7E1, 7E2	SI

PC-PC	4800	8N2	SI	NO	35x20	7E2, 8E1, 8E2, 7E1	SI
	4800	7E1	SI	SI	8x20	7E2, 8N2, 7E1	SI
	4800	7E2	SI	SI	25x20	7E2, 8E1, 8E2, 7E1	SI
	4800	7O1	SI	NO	8x20	8N1, 8N2, 7E2	NO
	4800	8O1	SI	NO	9x20	9N1, 9N2, 8E1, 8E2	SI
	4800	7N2	SI	SI	6x20	7N1, 7N2, 9N1, 8E1	SI
	4800	8N2	SI	SI	19x20	8N1, 8N2, 9N1, 8E2	SI
DE0 nano - DE2	4800	9N1	SI	SI	6x20	9N1, 9N2, 8E1, 8E2	SI
	4800	9E 1	SI	NO	35x20	9N1, 9N2, 8E1, 8E2	SI
DE2 - DE0 nano	4800	9N2	SI	NO	19x20	9E1, 9E2	NO
	4800	9E 2	SI	SI	9x20	9E1, 9E2, 9O1, 9O2	SI
	4800	9O1	SI	NO	25x20	9N1, 9N2, 8E1, 8E2	NO
DE0 nano-PC	2400	7N1	SI	SI	4x20	7N1, 7N2, 9N1, 9N2	SI
	2400	7N2	SI	NO	13x20	7N1, 8N1, 8N2, 7E1	SI
BLUETOOTH-DE0 nano	2400	8N1	SI	SI	5x20	8N1, 7E1, 7E2	SI
PC-PC	2400	8N2	SI	SI	4x20	7E2, 8N2, 7E1	SI
	2400	7E1	SI	SI	6x20	8N2, 7E1, 7E2	SI
	2400	7E2	SI	NO	7x20	8N2, 9N1, 9N2	NO
	2400	7O1	SI	NO	5x20	8N1, 7E1, 7E2	NO
	2400	8O1	SI	NO	9x20	9N1, 9N2, 8E1, 8E2	SI
	2400	7N2	SI	SI	11x20	7N1, 7N2, 8N1, 8N2	SI
BLUETOOTH-DE0 nano	2400	8N1	SI	SI	5x20	8N1, 7E1, 7E2,	SI
DE0 nano-PC	1200	7N1	SI	SI	13x20	7N1, 7N2, 8E1, 8E2	SI
	1200	7N2	SI	SI	7x20	7N2, 7E2, 8N1, 8N2	SI
	1200	8N1	SI	SI	4x20	8N1, 8N2, 7E1, 7E2	SI
PC-PC	1200	8N2	SI	NO	3x20	9N2, 8E1, 8E2	SI
	1200	7E1	SI	SI	3x20	7E1, 7E2	SI
	1200	7E2	SI	NO	5x20	8N1, 9N1, 9N2, 8E1	NO
	1200	7O1	SI	NO	5x20	8N1, 8N2, 7E1, 7E2	NO

Tabla 4-1 Pruebas realizadas

Para un entendimiento más profundo de la efectividad del analizador, analizamos los resultados agrupados por tres características de las

pruebas: la velocidad de trasmisión, la comunicación y la configuración conformada por el tamaño de datos, paridad y número de bits de parada.

Los resultados de efectividad agrupados por la velocidad de trasmisión y dispositivos analizados se muestran en la tabla 4-2.

Vel. Real	Vel. Ok	Conf. Ok	Datos Ok	No. Datos
115200	100%	60%	90%	9x20
57600	100%	60%	70%	12x20
28800	100%	50%	90%	16x20
19200	80%	30%	70%	17x20
9600	100%	30%	90%	22x20
4800	100%	60%	90%	16x20
2400	100%	60%	80%	7x20
1200	100%	60%	80%	7x20

Tabla 4-2 Efectividad por Velocidades

Como vemos las pruebas realizadas a 19200 baudios son las que tienen la efectividad más baja, esto se debe al algoritmo usado para determinar la velocidad (descarte), ya que todas las velocidades excepto esta son la mitad de su antecesor, ejemplo 57600=115200/2, pero 19200!= 28800/2, esto dificulta descartar la velocidad 28800. Las pruebas realizadas a las demás velocidades mantienen un valor de efectividad muy parecido.

Por otro lado la tabla 4-3 muestra los resultados de efectividad agrupados por la configuración usada.

Conf. Real	Vel. Ok	Conf. Ok	Datos Ok	No. Datos
7N1	100%	100%	100%	6x20
7N2	83%	50%	83%	12x20
7E1	100%	100%	100%	12x20
7E2	100%	22%	44%	19x20
7O1	100%	0%	37%	13x20
7O2	100%	0%	0%	15x20
8N1	100%	66%	100%	11x20
8N2	100%	45%	90%	16x20
8E1	100%	100%	100%	11x20
8O1	100%	16%	100%	11x20
9N1	100%	66%	66%	7x20
9N2	100%	33%	33%	10x20
9E1	100%	66%	100%	21x20
9E2	100%	66%	66%	9x20
9O1	100%	33%	33%	17x20

Tabla 4-3 Efectividad por Configuración

Al agrupar los resultados por configuración usada verificamos que las pruebas realizadas con configuración de paridad Odd son las que tienen menos efectividad. La causa de esto es que se eliminaron los módulos UARTs con configuración Odd para dar solución al problema de limitación de interrupciones externas del procesador Nios II descrito en la instanciación de los módulos UART en el punto 3.1.1.

La tabla 4-4 muestra los resultados agrupados por el dispositivo 1 usado en la comunicación analizada, ya que el analizador se auto configura de acuerdo a la configuración de este dispositivo que está conectado al UART0.

Comunicación		Vel.	Conf.	Datos	No.
Disp1	Disp2	Ok	Ok	Ok	datos
PC	*	98%	43%	73%	14x20
DE0 nano	*	100%	68%	87%	14x20
DE2	*	94%	56%	68%	9x20
Bluetooth	*	100%	70%	100%	12x20

Tabla 4-4 Efectividad por Comunicación

Agrupando las pruebas por dispositivos a analizar verificamos que los resultados mantienen valores de efectividad muy cercanos.

Y finalmente analizando las pruebas como un todo, obtendremos la efectividad general del analizador, la cual mostraremos en la tabla 4-4.

Velocidad	Configuración	Lectura de datos	No de datos
97%	53%	78%	13x20

Tabla 4-5 Efectividad del Analizador

CONCLUSIONES

1. La tarjeta de desarrollo DE0 nano nos proporciona un entorno de desarrollo flexible en hardware. Esto debido a dos razones: en la FPGA integrada se pueden implementar sistemas computacionales funcionales ajustados a la medida de la aplicación específica que se le quiera dar, sus puertos de expansión permiten la integración de hardware adicional para la complementación de dichos sistemas.
2. El protocolo RS-232 aparentemente en desuso, aún es uno de los principales métodos de comunicación de dispositivos periféricos como módulos GPS, módulos Bluetooth, módulos GSM, esto se debe a la sencillez de la configuración y al fácil entendimiento del método usado para la comunicación.
3. Los analizadores de protocolos son herramientas útiles para verificar si una determinada comunicación entre dos dispositivos se está efectuando de forma correcta.

4. El analizador de protocolo RS-232 que se diseñó en el presente trabajo es una herramienta que tiene una efectividad del 97% para la determinación de la velocidad de trasmisión en una comunicación, y del 53% en parámetros de configuración como tamaño en bits de datos, paridad y número de bits de parada.

5. El módulo NIOS II UART de Altera es un componente incluido en el kit de desarrollo de NIOS II que implementa el protocolo RS-232. La velocidad de trasmisión puede ser modificada desde el software después de la generación del sistema, parámetros como tamaño en bits de datos, paridad y número de bits de parada se fijan en la generación del sistema NIOS II y no pueden ser alterados después.

6. El procesador NIOS II soporta hasta 32 fuentes de interrupción externa, esto limita la cantidad de módulos que se necesiten ser instanciados en el sistema y que además necesiten interrumpir al procesador, la mayoría de los dispositivos que poseen interfaces de entrada de datos necesitan interrumpir al procesador.

RECOMENDACIONES

1. Para el diseño de sistemas embebidos se debe empezar conociendo los componentes principales de dichos sistemas. De forma explícita para desarrollar sistemas basados en el procesador NIOS II, antes debemos analizar en el procesador parámetros como número de interrupciones externas que puede recibir, arquitectura y lenguajes usados para su programación, además de los módulos disponibles para que junto con el procesador formen un sistema funcional.
2. Una vez seleccionado los módulos a usar en el sistema NIOS II realizar un análisis exhaustivo del funcionamiento, registros y las instrucciones necesarias para que el procesador pueda interactuar con dichos módulos.
3. Efectuar cuidadosamente la asignación de pines en la FPGA, relacionándolo correctamente con el correspondiente pin en la tarjeta de

desarrollo DE0 nano y en el PCB diseñado, esto debido a que un pequeño error en esta acción puede provocar daños en el hardware del proyecto así como un funcionamiento inesperado del mismo.

4. Debido a que el análisis de la comunicación se está haciendo a niveles de voltaje TTL, la longitud del cable usado para la conexión entre el analizador y los dispositivos debe ser hasta máximo 1m.
5. Si usted desea verificar que una comunicación RS-232 se esté efectuando de forma correcta conociendo los parámetros de configuración, lo recomendable es analizar usando la configuración manual del analizador.

.

ANEXOS

ANEXO A

CODIGO EN VERILOG DE INSTANCIACION DEL HARDWARE NIOS II

```
module Analizador (
    // Inputs
    CLOCK_50,
    KEY,
    SW,
/*****************************************************************/
    //uart
    UART_0_rx,
    UART_0_tx,
    UART_1_rx,
    UART_1_tx,
    //comunicacion
    Com_rx,
    Com_tx,
    //  Memory (SDRAM)
    DRAM_DQ,
/*****************************************************************/
    // Outputs
    //  Simple
    LEDG,
    //LCD
    DP_DATA,
    DP_E,
    DP_RS,
    DP_RW,
    //telcado
    entrada,
    salida,
    //  Memory (SDRAM)
    DRAM_ADDR,
    DRAM_BA,
    DRAM_CLK,
    DRAM_CKE,
    DRAM_CS_N,
    DRAM_CAS_N,
    DRAM_RAS_N,
    DRAM_WE_N,
    DRAM_DQM
);

/*****************************************************************
                            Port Declarations
*****************************************************************/
// Inputs
input               CLOCK_50;
input       [ 1: 0] KEY;
input       [ 3: 0] SW;
input          UART_0_rx;
```

```
input           UART_1_rx;
input           Com_rx;
//  Memory (SDRAM)
inout       [15: 0] DRAM_DQ;
// Outputs
//  Simple
output      [ 7: 0] LEDG;
//LCD
inout       [ 7: 0] DP_DATA;
output          DP_E;
output          DP_RS;
output          DP_RW;
//teclado
input    [ 3: 0] entrada;
output    [ 3: 0] salida;
//  Memory (SDRAM)
output      [11: 0] DRAM_ADDR;
output      [ 1: 0] DRAM_BA;
output              DRAM_CLK;
output              DRAM_CKE;
output              DRAM_CS_N;
output              DRAM_CAS_N;
output              DRAM_RAS_N;
output              DRAM_WE_N;
output      [ 1: 0] DRAM_DQM;
output              UART_0_tx;
output              UART_1_tx;
output            Com_tx;
/*****************************************************************************
                 Internal Wires and Registers Declarations
*****************************************************************************/
// Internal Wires
// Used to connect the Nios 2 system clock to the non-shifted output of
the PLL
wire                system_clock;

// Internal Registers
// State Machine Registers
// Output Assignments
/*****************************************************************************
                                Internal Modules
*****************************************************************************/
nios_system NiosII (
    // 1) global signals:
    .clk                                    (system_clock),
    .reset_n                                (KEY[0]),
    // the_Dip_switches
    .DIP_to_the_Dip_switches                (SW),
    // the_Pushbuttons
    .KEY_to_the_Pushbuttons                 ({KEY[1], 1'b1}),
    .LEDG_from_the_Green_LEDs               (LEDG),
    // the SDRAM
    .zs_addr_from_the_SDRAM                 (DRAM_ADDR),
    .zs_ba_from_the_SDRAM                   (DRAM_BA),
```

```
    .zs_cas_n_from_the_SDRAM                (DRAM_CAS_N),
    .zs_cke_from_the_SDRAM                  (DRAM_CKE),
    .zs_cs_n_from_the_SDRAM                 (DRAM_CS_N),
    .zs_dq_to_and_from_the_SDRAM            (DRAM_DQ),
    .zs_dqm_from_the_SDRAM                  (DRAM_DQM),
    .zs_ras_n_from_the_SDRAM                (DRAM_RAS_N),
    .zs_we_n_from_the_SDRAM                 (DRAM_WE_N),
// UARTs PARA ANALIZADOR
    .uart_0_external_connection_rxd       (UART_0_rx),
    .uart_0_external_connection_txd       (UART_0_tx),
    .uart_1_external_connection_rxd       (UART_0_rx),
    .uart_2_external_connection_rxd       (UART_0_rx),
    .uart_3_external_connection_rxd       (UART_0_rx),
    .uart_4_external_connection_rxd       (UART_0_rx),
    .uart_5_external_connection_rxd       (UART_0_rx),
    .uart_6_external_connection_rxd       (UART_0_rx),
    .uart_7_external_connection_rxd       (UART_0_rx),
    .uart_8_external_connection_rxd       (UART_0_rx),
    .uart_9_external_connection_rxd       (UART_0_rx),
    .uart_10_external_connection_rxd      (UART_0_rx),
    .uart_11_external_connection_rxd      (UART_0_rx),
    .uart_18_external_connection_rxd      (UART_1_rx),
    .uart_18_external_connection_txd      (UART_1_tx),
    .uart_19_external_connection_rxd      (UART_1_rx),
    .uart_20_external_connection_rxd      (UART_1_rx),
    .uart_21_external_connection_rxd      (UART_1_rx),
    .uart_22_external_connection_rxd      (UART_1_rx),
    .uart_23_external_connection_rxd      (UART_1_rx),
    .uart_24_external_connection_rxd      (UART_1_rx),
    .uart_25_external_connection_rxd      (UART_1_rx),
    .uart_26_external_connection_rxd      (UART_1_rx),
    .uart_27_external_connection_rxd      (UART_1_rx),
    .uart_28_external_connection_rxd      (UART_1_rx),
    .uart_29_external_connection_rxd      (UART_1_rx),
// LCD
    .lcd_data                             (DP_DATA),
    .lcd_E                                (DP_E),
    .lcd_RS                               (DP_RS),
    .lcd_RW                               (DP_RW),
//TECLADO
    .tecladoin_export                   (entrada),
    .tecladoout_export                  (salida),
//UART PARA BLUETOOTH Y USB
    .comunicacion_rxd                   (Com_rx),
    .comunicacion_txd                   (Com_tx)
);
sdram_pll neg_3ns (CLOCK_50, DRAM_CLK, system_clock);

endmodule
```

ANEXO B

CODIGO EN C PARA LA PROGRAMACION DEL PROCESADOR NIOS II

```
//LIBRERIAS
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "altera_avalon_uart.h"
#include "altera_avalon_uart_fd.h"
#include "altera_avalon_uart_regs.h"
#include "io.h"
#define __ALTERA_AVALON_LCD_16207_FD_H__
#define __ALTERA_AVALON_LCD_16207_REGS_H__
#define __ALTERA_AVALON_LCD_16207_H__
#define UART_NUEVO_NAME "/dev/uart_0"

//DECLARACION DE FUNCIONES
    void selec_uart (volatile int** b0,volatile int** b1, char** n0,
    char** n1, char ua[]);
    void selec_baudrate(volatile int* ,volatile int* ,int , int );
    int prueba_uart ( int , int* , int* , char *);
    void change_baudrate(int , int  );
    void reset_control();
    int leer_tecla();
    int char_int(char);
    int Gchar_int( char c[], int no);
    void encerar_status();
    void open_uart (FILE* , FILE* , char* , char* );
    void close_uart (FILE* , FILE* );
//DECLARACION DE VARIABLES
    int serial_speed = 9600; //uart speed
    char dato_config[10]="0000000000";
    char config[4]="000";
    char *config_auto[4]={"000","000","000","000"};
    int FPGA_CLK = 50000000; //set at 50Mhz in Quartus
    int  Baudrate[8]={115200,57600,28800,19200,9600,4800,2400,1200};
    int posbaud=0;
    int tiempo=0;
    int escoge=0;
    volatile int * UART0_ptr = (int *) UART_1_BASE;
    volatile int * UART1_ptr = (int *) UART_18_BASE;
    volatile int *leetec= (int *) PIO_0_BASE;
    volatile int *esctec= (int *) PIO_1_BASE;
    char *name0;
    char *name1;
    char *selec;
    char uauno[14]="00000000000000";
    char uados[14]="00000000000000";
```

```
    int data0=0;
    int data1=0;
    int f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,sumaf;
    int b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,sumab;
    FILE* LED;
    FILE* COM;

int main ()
{
//PRESENTACION

    LED = fopen ("/dev/lcd_0", "r+");
    fprintf(LED, " ANALIZADOR  DE \nPROTOCOLO  RS232\n");
    fclose(LED);
    usleep(2000000);
    escogemodo:
    usleep(20000);
//SELECCIÓN DE MODO
    LED = fopen ("/dev/lcd_0", "r+");
    fprintf(LED, " ESCOJA CONFIG \n1)MANUAL  2)AUTO\n");
    fclose(LED);
    escoge=char_int(leer_tecla());
    printf ("%c", escoge);
        if(!((escoge==1) ||(escoge==2))){
                usleep(100000);
                LED = fopen ("/dev/lcd_0", "r+");
                fprintf(LED, "     OPCION     \n   INCORRECTA   \n");
                fclose(LED);
            goto escogemodo;
        }
    while (1) {
      switch ( escoge ) {
//CONFIGURACION MANUAL
       case 1:
              escribebaud:
              printf("Configuracion Manual\n");
              usleep(100000);
              LED = fopen ("/dev/lcd_0", "r+");
              fprintf(LED, "Escribe baudrate\n      \r");
              fclose(LED);
              int i=0;
              ingresodigito:
              dato_config[i]=leer_tecla();
              if((i==8)||(dato_config[i]==35)){
              printf("%d",i);
              serial_speed=Gchar_int(dato_config,i);
//VALIDACION DE INGRESO DE LA VELOCIDAD DE TRASMISION
               if((serial_speed<1200)||(serial_speed>115200)){
                usleep(100000);
                LED = fopen ("/dev/lcd_0", "r+");
                fprintf(LED, "\n ERROR BAUDRATE \nRANG:1200-115200\n");
                fclose(LED);
                usleep(1000000);
                goto escribebaud;
```

```
                }
                     goto escoja_config;
                }
                if(char_int(dato_config[i])==-1){
                         goto ingresodigito;
                }
                LED = fopen ("/dev/lcd_0", "r+");
                fprintf(LED, "%c",dato_config[i]);
                fclose(LED);
                i=i+1;
                goto ingresodigito;
                escoja_config:
                printf("velocidad:%d",serial_speed);
                usleep(100000);
                LED = fopen ("/dev/lcd_0", "r+");
                fprintf(LED, "\n Escribe config:\n                \r");
                fclose(LED);
                i=0;
                ingresoconfig:
                printf("\n%d\n",i);
                config[i]=leer_tecla();
//VALIDACION DE INGRESO DEL TAMAÑO DE BIT
                if((i==0) &&! ( (char_int(config[i])==7) ||
                   (char_int(config[i])==8) ||
                   (char_int(config[i])==9))){
                      printf("NO ES 7 NI 9 NI 8");
                      goto ingresoconfig;
                 }
//VALIDACION DE INGRESO DE PARIDAD (N, O, E)
                if((i==1) &&!( (config[i]==78) ||(config[i]==79)||
                    (config[i]==69))){
                       printf("NO ES NOE");
                       goto ingresoconfig;
                    }
//VALIDACION DE INGRESO DE NUMERO DE BITS DE PARADA
                if((i==2) &&!( (char_int(config[i])==1)||
                    (char_int(config[i])==2))){
                       goto ingresoconfig;
                    }
                      LED = fopen ("/dev/lcd_0", "r+");
                      fprintf(LED, "%c",config[i]);
                      fclose(LED);
                      if(i==2){
//MUESTRA EN LCD LA CONFIGURACION SELECCIONADA
                        printf("config:%s",config);
                        selec=config;
                        printf("selec:%s",selec);
                        usleep(10000);
                        LED = fopen ("/dev/lcd_0", "r+");
                        fprintf(LED, "\nCONFIG:%3.3s\nBAUDRATE:%6d\n",
                                config,serial_speed);
                        fclose(LED);
```

```
//VALIDACION DE INGRESO DE NUMERO DE BITS DE PARADA

                        COM = fopen ("/dev/uart_15", "r+");
                        fprintf(COM, "00%6d%3.3s11",serial_speed,config);
                        fclose(COM);
                        goto leedatos;
                              }
                     i=i+1;
                     goto ingresoconfig;
                     leedatos:
//SELECCIONA EL UART A USAR DE ACUERDO A LA CONFIGURACION
                     selec_uart (&UART0_ptr, &UART1_ptr, &name0,
                                 &name1,config);
//SETEO DE VELOCIDAD DE TRASMISION
                     selec_baudrate(UART0_ptr, UART1_ptr, serial_speed,
                                    FPGA_CLK);
                     IOWR_ALTERA_AVALON_UART_CONTROL (UART0_ptr, 0);
                     IOWR_ALTERA_AVALON_UART_CONTROL (UART1_ptr, 0);
                      *(UART0_ptr+2)=0;
                      *esctec=0xe;
                        while (1)
                          {
                            if(*leetec==0xe){
                              goto escogemodo;
                            }
//SI UN NUEVO DATO ESTA EN EL REGISTRO RX (ANALISIS DEL BIT RRDY DEL
REGISTRO STATUS)-UART0 y ENVIO DE DATOS AL UART DE COMUNICACION
                            if ((*(UART0_ptr+2) & 0x80))
                                 {
                                    data0=*(UART0_ptr);
                                    printf ("%c",(char) data0);
                                    IOWR(UART_15_BASE, 1, data0);

                                  }
//SI UN NUEVO DATO ESTA EN EL REGISTRO RX (ANALISIS DEL BIT RRDY DEL
REGISTRO STATUS)-UART1 y ENVIO DE DATOS AL UART DE COMUNICACION
                            if ((*(UART1_ptr+2) & 0x80))
                                 {
                                   data1=*(UART1_ptr);
                                   printf ("%c",(char) data1);
                                   data1=data1+128;
                                   IOWR(UART_15_BASE, 1, data1);

                  break;
//CONFIGURACION AUTOMATICA
    case 2:
           LED = fopen ("/dev/lcd_0", "r+");
           fprintf(LED, "  AUTO  CONFIG  \n   EN PROCESO   \n");
           fclose(LED);
           printf("Auto Configuracion!\n");
           printf ("%d", Baudrate[posbaud]);
           serial_speed=Baudrate[posbaud];
           int suma,posi,actual,j=0;
           esctec=0xe; change_baudrate(Baudrate[posbaud], FPGA_CLK);
```

```
            reset_control();
            encerar_status();
            sumaf=0;sumab=0;
            f0=0;f1=0;f2=0;f3=0;f4=0;f5=0;f6=0;f7=0;f8=0;
            f9=0;f10=0;f11=0;
            b0=0;b1=0;b2=0;b3=0;b4=0;b5=0;b6=0;b7=0;b8=0;
            b9=0;b10=0;b11=0;
            while ((sumaf<12) && (sumab<23))
                {
                  if(*leetec==0xe){
                    goto escogemodo;
                  }
             suma=0;posi=0;
//ANALISIS DE LOS DATOS PARA AUTOCONFIGURARSE
            if((actual=prueba_uart ( UART_0_BASE, &f0 , &b0,"7N1"))==1){
                config_auto[posi]="7N1";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_1_BASE, &f1 , &b1,"7N2"))==1){
                config_auto[posi]="7N2";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_2_BASE, &f2 , &b2,"8N1"))==1){
                config_auto[posi]="8N1";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_3_BASE, &f3 , &b3,"8N2"))==1){
                config_auto[posi]="8N2";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_4_BASE, &f4 , &b4,"9N1"))==1){
                config_auto[posi]="9N1";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_5_BASE, &f5 , &b5,"9N2"))==1){
                config_auto[posi]="9N2";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_6_BASE, &f6 , &b6,"7E1"))==1){
                config_auto[posi]="7E1";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_7_BASE, &f7 , &b7,"7E2"))==1){
                config_auto[posi]="7E2";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_8_BASE, &f8 , &b8,"8E1"))==1){
                config_auto[posi]="8E1";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart ( UART_9_BASE, &f9 , &b9,"8E2"))==1){
                config_auto[posi]="8E2";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart (UART_10_BASE,&f10, &b10,"9E1"))==1){
                config_auto[posi]="9E1";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
            if((actual=prueba_uart (UART_11_BASE, &f11,&b11,"9E2"))==1){
                config_auto[posi]="9E2";posi=posi+1;if(posi==4){posi=0;}
              }suma=suma+actual;
```

```
                    if((suma>0)&&(suma<4)){
                       printf("aqui");
                       j=j+1;
                       if(j>10){
                       printf("ya termino");
                       strcpy (config, config_auto[0]);
                       posbaud=0;
                       usleep(10000);
//MUESTRA VELOCIDAD Y POSIBLES CONFIGURACIONES EN EL LCD
                       LED = fopen ("/dev/lcd_0", "r+");
                       printf(LED, "\n%3.3s-%3.3s-%3.3s-%3.3s
                              \nBAUDRATE:%6d\n",config_auto[0],
                               config_auto[1],config_auto[2],
                               config_auto[3],serial_speed);
                       fclose(LED);
//ENVIO DE VELOCIDAD Y CONFIGURACION USADA AL UART DE COMUNICACION
                       COM = fopen ("/dev/uart_15", "r+");
                       fprintf(COM, "00%6d%3.3s11",serial_speed,config);
                       fclose(COM);
                       goto leedatos;
                     }
                }
               sumaf=f0+f1+f2+f3+f4+f5+f6+f7+f8+f9+f10+f11;
               sumab=b0+b1+b2+b3+b4+b5+b6+b7+b8+b9+b10+b11;
               }posbaud=posbaud+1;
               if(posbaud==9){posbaud=0;}
               break;
               default:
               printf("No selecciono ningún modo!\n");
               break;
           }
         }
       return 0;
}
//FUNCION PARA INTERACTUAR CON EL TECLADO
int leer_tecla(){
volatile int *p_gpio_in= (int *) PIO_0_BASE;
volatile int *p_gpio_out=(int *) PIO_1_BASE;
int val[4]={0x7,0xb,0xd,0xe};
int col=0, fil=0,valor=0,salida=0;
int teclado[4][4]={{'1','2','3','N'},
                   {'4','5','6','E'},
                   {'7','8','9','O'},
                   {'*','0','+','#'}};
     while(1){
            for(col=0;col<4;col++){
                  *p_gpio_out=val[col];
                  for(fil=0;fil<4;fil++){
                        if(*p_gpio_in==val[fil]){
                             valor=teclado[col][fil];
                             printf("Tecla %c\n", valor);
                                 printf("Saliendo\n");
                                 usleep(200000);
```

```
                                goto etiqueta_1;
                            }
                        }
                    }
                }
      etiqueta_1:printf("ya salio es:%c\n", salida);//usleep(200000);
//return salida;
   return valor;
}
//FUNCION USADA PARA LA AUTOCONFIGURACION
int prueba_uart ( int base, int* no_fram_error , int* no_break_error,
char * conf)
{
    int salida=0;
     if ((IORD_ALTERA_AVALON_UART_STATUS(base) & 0x80))
          if((*no_fram_error<1)&& (*no_break_error<2)){
                    printf ("%s",conf);
                    printf ("%c",(char)
                IORD_ALTERA_AVALON_UART_RXDATA(base));
//VERIFICA SI HAY FRAMING ERROR
             if ((IORD_ALTERA_AVALON_UART_STATUS(base) & 0x2) ){
               (*no_fram_error)=(*no_fram_error)+1;
             }
//VERIFICA SI HAY BREAK ERROR
             if ((IORD_ALTERA_AVALON_UART_STATUS(base) & 0x4)){
               (*no_break_error)=(*no_break_error)+1;
              }
               IOWR_ALTERA_AVALON_UART_STATUS (base, 0);
                   salida=1;
           }
     }
     return salida;
}
//FUNCION USADA PARA LA CONFIGURACION MANUAL
void selec_uart (volatile int** b0,volatile int** b1, char** n0, char**
n1, char ua[])//char* ua
{
    if (strcmp (ua, "7N1")==0){
              printf("7n1\n");
              *b0 = (int *) UART_0_BASE;
              *b1 = (int *) UART_18_BASE;
              *n0 =  UART_0_NAME;
              *n1 =  UART_18_NAME;
    }
    else if (strcmp (ua, "7N2")==0){
              printf("7n2\n");
              *b0 = (int *) UART_1_BASE;
              *b1 = (int *) UART_19_BASE;
              *n0 = UART_1_NAME;
              *n1 = UART_19_NAME;
    }
    else if (strcmp (ua, "8N1")==0){
              printf("8n1\n");
              *b0 = (int *) UART_2_BASE;
```

```
            *b1 = (int *) UART_20_BASE;
            *n0 =  UART_2_NAME;
            *n1 =  UART_20_NAME;
}
else if (strcmp (ua, "8N2")==0){
            printf("8n2\n");
            *b0 = (int *) UART_3_BASE;
            *b1 = (int *) UART_21_BASE;
            *n0 = UART_3_NAME;
            *n1 = UART_21_NAME;
}
else if (strcmp (ua, "9N1")==0){
            printf("9n1\n");
            *b0 = (int *) UART_4_BASE;
            *b1 = (int *) UART_22_BASE;
            *n0 = UART_4_NAME;
            *n1 = UART_22_NAME;
}
else if (strcmp (ua, "9N2")==0){
            printf("9n2\n");
            *b0 = (int *) UART_5_BASE;
            *b1 = (int *) UART_23_BASE;
            *n0 = UART_5_NAME;
            *n1 = UART_23_NAME;
}
else if (strcmp (ua, "7E1")==0){
            printf("7e1\n");
            *b0 = (int *) UART_6_BASE;
            *b1 = (int *) UART_24_BASE;
            *n0 =  UART_6_NAME;
            *n1 =  UART_24_NAME;
}
else if (strcmp (ua, "7E2")==0){
            printf("7e2\n");
            *b0 = (int *) UART_7_BASE;
            *b1 = (int *) UART_25_BASE;
            *n0 = UART_7_NAME;
            *n1 = UART_25_NAME;
}
else if (strcmp (ua, "8E1")==0){
            printf("8e1\n");
            *b0 = (int *) UART_8_BASE;
            *b1 = (int *) UART_26_BASE;
            *n0 = UART_8_NAME;
            *n1 = UART_26_NAME;
}
else if (strcmp (ua, "8E2")==0){
     printf("8e2\n");
            *b0 = (int *) UART_9_BASE;
            *b1 = (int *) UART_27_BASE;
            *n0 =  UART_9_NAME;
            *n1 =  UART_27_NAME;
}
else if (strcmp (ua, "9E1")==0){
```

```
            printf("9e1\n");
            *b0 = (int *) UART_10_BASE;
            *b1 = (int *) UART_28_BASE;
            *n0 =  UART_10_NAME;
            *n1 =  UART_28_NAME;
}
else if (strcmp (ua, "9E2")==0){
            printf("9e2\n");
            *b0 = (int *) UART_11_BASE;
            *b1 = (int *) UART_29_BASE;
            *n0 =  UART_11_NAME;
            *n1 =  UART_29_NAME;
}
else if (strcmp (ua, "7O1")==0){
            printf("7o1\n");
            *b0 = (int *) UART_6_BASE;
            *b1 = (int *) UART_24_BASE;
            *n0 =  UART_6_NAME;
            *n1 =  UART_24_NAME;
}
else if (strcmp (ua, "7O2")==0){
            printf("7o2\n");
            *b0 = (int *) UART_7_BASE;
            *b1 = (int *) UART_25_BASE;
            *n0 =  UART_7_NAME;
            *n1 =  UART_25_NAME;
}
else if (strcmp (ua, "8O1")==0){
            printf("8o1\n");
            *b0 = (int *) UART_8_BASE;
            *b1 = (int *) UART_26_BASE;
            *n0 =  UART_8_NAME;
            *n1 =  UART_26_NAME;
}
else if (strcmp (ua, "8O2")==0){
            printf("8o2\n");
            *b0 = (int *) UART_9_BASE;
            *b1 = (int *) UART_27_BASE;
            *n0 =  UART_9_NAME;
            *n1 =  UART_27_NAME;
}
else if (strcmp (ua, "9O1")==0){
            printf("9o1\n");
            *b0 = (int *) UART_10_BASE;
            *b1 = (int *) UART_28_BASE;
            *n0 = UART_10_NAME;
            *n1 = UART_28_NAME;
}
else if (strcmp (ua, "9O2")==0){
            printf("9o2\n");
            *b0 = (int *) UART_11_BASE;
            *b1 = (int *) UART_29_BASE;
            *n0 = UART_11_NAME;
            *n1 = UART_29_NAME;
```

```
    }
    else {
          printf("estamos jodidos!\n");
    }
}
//SETEO DE BAUD RATE, USADO EN LA CONFIGURACION MANUAL
void selec_baudrate(volatile int *baseuno,volatile int *basedos,int
baud, int clkfpga )
{
   IOWR_ALTERA_AVALON_UART_DIVISOR(baseuno, (int)(clkfpga/baud+0.5));
   IOWR_ALTERA_AVALON_UART_DIVISOR(basedos, (int)(clkfpga/baud+0.5));
}
//RESETEO DEL REGISTRO DE CONTROL DE LOS MODULOS UART
void reset_control()
{
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_0_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_1_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_2_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_3_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_4_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_5_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_6_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_7_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_8_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_9_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_10_BASE, 0);
   IOWR_ALTERA_AVALON_UART_CONTROL (UART_11_BASE, 0);
}
//RESETEO DEL REGISTRO DE ESTADO DE LOS MODULOS UART
void encerar_status()
{
    IOWR_ALTERA_AVALON_UART_STATUS (UART_0_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_1_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_2_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_3_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_4_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_5_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_6_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_7_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_8_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_9_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_10_BASE, 0);
    IOWR_ALTERA_AVALON_UART_STATUS (UART_11_BASE, 0);
}
//CAMBIO DE LA VELOCIDAD DE TRASMISION DE TODOS LOS MODULSO UART, USADO
EN LA AUTOCONFIGURACION
void change_baudrate(int baud, int clkfpga )
{
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_0_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_1_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_2_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_3_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_4_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_5_BASE, (int)(clkfpga/baud+0.5));
```

```
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_6_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_7_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_8_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_9_BASE, (int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_10_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_11_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_18_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_19_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_20_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_21_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_22_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_23_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_24_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_25_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_26_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_27_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_28_BASE,(int)(clkfpga/baud+0.5));
 IOWR_ALTERA_AVALON_UART_DIVISOR(UART_29_BASE,(int)(clkfpga/baud+0.5));
}
//CONVIERTE UN CARÁCTER EN ENTERO
int char_int(char c){
if(c=='1'){return 1;}
else if(c=='2'){return 2;}
else if(c=='3'){return 3;}
else if(c=='4'){return 4;}
else if(c=='5'){return 5;}
else if(c=='6'){return 6;}
else if(c=='7'){return 7;}
else if(c=='8'){return 8;}
else if(c=='9'){return 9;}
else if(c=='0'){return 0;}
else {return -1;}
}
// CONVIERTE UN GRUPO DE CARÁCTERES EN ENTERO
int Gchar_int( char c[], int no){
int i=0,total=0,x=0;
for(i=0;i<no;i++){
x=char_int(c[i]);
if(x>=0)
total=(total*10)+x;
}
return total;
}
```

ANEXO C

CODIGO EN JAVA PARA LA PRESENTACIÓN DE LOS DATOS EN UN PC

Clase Dato

```
package Dato;

public class Dato {
    int pos;
    int canal;
    long time;
    byte buffer;// = new byte[1024];
    String hex;
    String binario;
    int decimal;
    char ascii;

    public Dato(int pos, int canal, long time, byte buffer) {
        this.pos = pos;
        this.canal = canal;
        this.time = time;
        this.buffer = buffer;
        this.binario= Binarify(buffer);
        this.decimal= (int)buffer;
        this.hex=Integer.toHexString(this.decimal);
        this.ascii=(char)this.decimal;//Byte.toString(buffer);

    }
     public static String Binarify( byte ByteToCheck ) {
        String binaryCode = "";
        byte[] reference = new byte[]{ (byte) 0x80, 0x40, 0x20, 0x10,
                                       0x08, 0x04, 0x02, 0x01 };
        for ( byte z = 0; z < 8; z++ ) {
            if ( ( reference[z] & ByteToCheck ) != 0 ) {
                binaryCode += "1";
            }
            else {binaryCode += "0";}
        }
      return binaryCode;
    }

    public void setPos(int pos) {this.pos = pos;}
    public void setCanal(int canal) {this.canal = canal;}
    public void setTime(long time) {this.time = time;}
    public void setBuffer(byte buffer) {this.buffer = buffer;}
    public void setHex(String hex) {this.hex = hex;}
    public void setBinario(String binario) {this.binario = binario;}
    public void setDecimal(int decimal) {this.decimal = decimal;}
    public void setAscii(char ascii) {this.ascii = ascii;}
```

```
    public int getPos() {return pos;}
    public int getCanal() {return canal;}
    public long getTime() {return time;}
    public byte getBuffer() {return buffer;}
    public String getHex() {return hex;}
    public String getBinario() {return binario;}
    public int getDecimal() {return decimal;}
    public char getAscii() {return ascii;}
}
```

Clase SerialReader

```
package uart;
import Dato.Dato;
import java.io.*;
import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;
import java.util.ArrayList;
import javax.swing.JLabel;
import javax.swing.table.DefaultTableModel;

public  class SerialReader implements SerialPortEventListener
    {
        private InputStream in;
        private byte[] buffer = new byte[1024];
        int pos=0;
        Dato dato;

        long t0;
        long tf;
        long time;
        String config;
        DefaultTableModel dtm;
        ArrayList<JLabel> label;
        long[] tiempos={0,0} ;
        public SerialReader ( InputStream in,DefaultTableModel
        dtm,ArrayList<JLabel> label )
        {
            this.in = in;
            this.dtm=dtm;
            this.label=label;
        }
        public void serialEvent(SerialPortEvent arg0) {
            int data;

            try
            {
                int len = 0;
                while ( ( data = in.read()) > -1 )
                {
                    if ( data == '\n' ) {
```

```
                    break;
                }           time = 1;

                if (pos==0){t0=System.currentTimeMillis();}

                tf=System.currentTimeMillis();
                time=tf-t0;
                buffer[len++] = (byte) data;
                dato=new Dato(pos++,1,time,(byte) data);
                Object[] newRow={dato.getPos(), dato.getCanal(),
                                 dato.getTime(), dato.getHex(),
                            dato.getBinario(),dato.getDecimal(),
                                 dato.getAscii()};
                dtm.addRow(newRow);
                System.out.println(Binarify((byte) data));
            }
            config=new String(buffer,0,len);
            if((config.length()==13) &&
               (config.substring(0, 2).compareTo("00")==0)&&
               (config.substring(11, 13).compareTo("11")==0))
            {

                label.get(0).setText(config.substring(2, 8));
                label.get(1).setText(config.substring(8, 9));
                label.get(3).setText(config.substring(10, 11));
                if (config.substring(9, 10).compareTo("N")==0)
                   {label.get(2).setText("NONE");}
                else if (config.substring(9, 10).compareTo("O")==0)
                   {label.get(2).setText("ODD");}
                else if (config.substring(9, 10).compareTo("E")==0)
                   {label.get(2).setText("EDD");}
                else {label.get(2).setText("---");}


            }

        }
        catch ( IOException e )
        {
            e.printStackTrace();
            System.exit(-1);
        }
    }


}
```

CLASE UART

```
package uart;
import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
```

```
import gnu.io.SerialPortEventListener;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import javax.swing.JLabel;
import javax.swing.table.DefaultTableModel;

public  class Uart
{
    private SerialPort serialPort;
    private OutputStream out;
    private InputStream in;
    public Uart()
    {
        super();
    }

   public void connect ( String portName,DefaultTableModel
                         dtm,ArrayList<JLabel> label ) throws Exception
    {
        CommPortIdentifier portIdentifier =
        CommPortIdentifier.getPortIdentifier(portName);
        if ( portIdentifier.isCurrentlyOwned() )
        {
            System.out.println("Error: Port is currently in use");
        }
        else
        {
            CommPort commPort =
            portIdentifier.open(this.getClass().getName(),2000);

            if ( commPort instanceof SerialPort )
            {
                 serialPort = (SerialPort) commPort;
                 serialPort.setSerialPortParams(115200,
                    SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                    SerialPort.PARITY_NONE);

                 in = serialPort.getInputStream();
                 out = serialPort.getOutputStream();

                serialPort.addEventListener(
                   new SerialReader(in,dtm,label)
                );
                serialPort.notifyOnDataAvailable(true);

           }
           else
            {
                System.out.println("Error: Only serial ports are
                 handled by this example.");
            }
```

```
        }
    }
    public void disconnect() {
        if (serialPort != null) {
            try {
                // Cerrar los i/o streams.
                out.close();
                in.close();
            } catch (IOException ex) {
                // don't care
            }
            // Cerrar el puerto
            serialPort.close();
            serialPort = null;
        }
    }


    public ArrayList listPorts() //public void listPorts()
    {
        ArrayList puertos = new ArrayList();
        java.util.Enumeration<CommPortIdentifier> portEnum =
        CommPortIdentifier.getPortIdentifiers();
        while ( portEnum.hasMoreElements() )
        {
            CommPortIdentifier portIdentifier = portEnum.nextElement();
            puertos.add(portIdentifier.getName());
            System.out.println(portIdentifier.getName()  +  " - " +
            getPortTypeName(portIdentifier.getPortType()) );
        }
       return puertos;
    }

     String getPortTypeName ( int portType )
    {
        switch ( portType )
        {
            case CommPortIdentifier.PORT_I2C:
                return "I2C";
            case CommPortIdentifier.PORT_PARALLEL:
                return "Parallel";
            case CommPortIdentifier.PORT_RAW:
                return "Raw";
            case CommPortIdentifier.PORT_RS485:
                return "RS485";
            case CommPortIdentifier.PORT_SERIAL:
                return "Serial";
            default:
                return "unknown type";
        }
    }
```

CLASE SHOWDATA

Esta clase se la implemento usando el editor gráfico, la figura 3-30 muestra los componentes instanciados. Las modificaciones realizadas las mostramos a continuación:

```
//VARIABLES
    ArrayList<JLabel> label;
    DefaultTableModel dtm;
    int a;
    Uart ua;
//CONSTRUCTOR
    public ShowData() {
        initComponents();
        jButton2.setEnabled(false);
        ArrayList puertos ;
        this.label= new ArrayList<JLabel>();
        String[] columnNames = {"No.", "Canal", "Tiempo (ms)",
        "Hexadecimal","Binario", "Decimal", "ASCII"};
        this.dtm= new DefaultTableModel( columnNames,0);
        jTable1.setModel(dtm);
 try
        {
            ua=new Uart();
            puertos=ua.listPorts();
            for ( int i = 0; i < puertos.size(); i ++ ) {
                  jComboBox1.addItem(puertos.get(i));
             }
            label.add(jLabel5);
            label.add(jLabel6);
            label.add(jLabel7);
            label.add(jLabel8);

        }
        catch ( Exception e )
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

//EVENTOS DE LOS BOTONES
  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
      {
       try
        {
         ua.connect(jComboBox1.getSelectedItem().toString(),dtm,label);
         jButton1.setEnabled(false);
         jButton2.setEnabled(true);
```

```
        }
        catch ( Exception e )
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}

 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
   {
        // TODO add your handling code here:
        ua.disconnect();
        jButton2.setEnabled(false);
        jButton1.setEnabled(true);
   }
```

ANEXO D

CODIGO EN ANDROID PARA LA PRESENTACIÓN DE LOS DATOS EN UN DISPOSITIVO MOBIL

```
package uart.analizador;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.UUID;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.res.Resources;
import android.os.Bundle;
import android.view.Gravity;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
private static final String TAG = "bluetooth2";
    private TextView conf;
    TableLayout tabla;
    TableLayout cabecera;
    TableRow.LayoutParams layoutFila;
    TableRow.LayoutParams layoutId;
    TableRow.LayoutParams layoutTexto;
    final int RECIEVE_MESSAGE = 1;        // Estatus para el Handler
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private StringBuilder sb = new StringBuilder();
    Handler h;
    private ConnectedThread mConnectedThread;
    // SPP UUID service
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-
    1000-8000-00805F9B34FB");
```

```
// Direccion MAC del modulo Bluetooth a conectarse
  private static String address = "00:13:01:04:07:89";
  private int MAX_FILAS = 10;
  Resources rs;
  @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      rs = this.getResources();
      conf = (TextView) findViewById(R.id.conf);
      tabla = (TableLayout)findViewById(R.id.tabla);
      cabecera = (TableLayout)findViewById(R.id.cabecera);
      layoutFila =
      new TableRow.LayoutParams(TableRow.LayoutParams.WRAP_CONTENT,
                                TableRow.LayoutParams.WRAP_CONTENT);
      layoutId =
    new TableRow.LayoutParams(50,TableRow.LayoutParams.WRAP_CONTENT);
      layoutTexto =
   new TableRow.LayoutParams(150,TableRow.LayoutParams.WRAP_CONTENT);
      agregarCabecera();

      h = new Handler() {
           @Override
           public void handleMessage(android.os.Message msg) {
              switch (msg.what) {
               case RECIEVE_MESSAGE:
// Si un mensaje es recibido
               byte[] readBuf = (byte[]) msg.obj;
               String strIncom = new String(readBuf, 0, msg.arg1);
// Crear string desde el arreglo de bytes
               int value=new Byte(readBuf[0]).intValue();
               sb.append(strIncom);
// Añadir al final del string
               int endOfLineIndex = sb.indexOf("\r\n");
// Determinar el final del dato
                  if (endOfLineIndex > 0) {
// if end-of-line,
                String sbprint = sb.substring(0, endOfLineIndex);
// Extraer el string
                  sb.delete(0, sb.length());
// Valida si el dato que ingresa son los parametros de configuracion
                  if((strIncom.length()>=13) &&
                     (strIncom.substring(0, 2).compareTo("00")==0) &&
                     (strIncom.substring(11, 13).compareTo("11")==0))
                   {
                   conf.setText("Baud Rate:"+strIncom.substring(2, 8)
                              +" Config:"+strIncom.substring(8, 11));
                   }
                   else{agregarFilas(value);}


              break;
              }
```

```
        };
    };

    // obtener el adaptador de Bluetooth
    btAdapter = BluetoothAdapter.getDefaultAdapter();
     checkBTState();
}
    // función que agrega la cabecera de la tabla
public void agregarCabecera(){
 TableRow fila;
 TextView txtId;
 TextView txtNombre;
 TextView txtH;
 TextView txtD;
 TextView txtB;
 fila = new TableRow(this);
 fila.setLayoutParams(layoutFila);
 txtId = new TextView(this);
 txtNombre = new TextView(this);
 txtH = new TextView(this);
 txtD = new TextView(this);
 txtB = new TextView(this);
 txtId.setText(rs.getString(R.string.ca));
 txtId.setGravity(Gravity.CENTER_HORIZONTAL);
 txtId.setTextAppearance(this,R.style.etiqueta);
 txtId.setBackgroundResource(R.drawable.tabla_celda_cabecera);
 txtId.setLayoutParams(layoutId);
 txtNombre.setText(rs.getString(R.string.dat));
 txtNombre.setGravity(Gravity.CENTER_HORIZONTAL);
 txtNombre.setTextAppearance(this,R.style.etiqueta);
 txtNombre.setBackgroundResource(R.drawable.tabla_celda_cabecera);
 txtNombre.setLayoutParams(layoutId);
 txtH.setText(rs.getString(R.string.h));
 txtH.setGravity(Gravity.CENTER_HORIZONTAL);
 txtH.setTextAppearance(this,R.style.etiqueta);
 txtH.setBackgroundResource(R.drawable.tabla_celda_cabecera);
 txtH.setLayoutParams(layoutId);
 txtD.setText(rs.getString(R.string.d));
 txtD.setGravity(Gravity.CENTER_HORIZONTAL);
 txtD.setTextAppearance(this,R.style.etiqueta);
 txtD.setBackgroundResource(R.drawable.tabla_celda_cabecera);
 txtD.setLayoutParams(layoutId);
 txtB.setText(rs.getString(R.string.b));
 txtB.setGravity(Gravity.CENTER_HORIZONTAL);
 txtB.setTextAppearance(this,R.style.etiqueta);
 txtB.setBackgroundResource(R.drawable.tabla_celda_cabecera);
 txtB.setLayoutParams(layoutTexto);
 fila.addView(txtId);
 fila.addView(txtNombre);
 fila.addView(txtH);
 fila.addView(txtD);
 fila.addView(txtB);
 cabecera.addView(fila);
}
```

```
public void agregarFilas(int dato){

 TableRow fila;
 TextView txtId;
 TextView txtNombre;
 TextView txtH;
 TextView txtD;
 TextView txtB;
 int canal=0;
        if (dato<0){
                    dato=dato+128;
                 canal=1;
                }
         if (dato>127){
                    dato=dato-128;
                 canal=1;
                }
      fila = new TableRow(this);
     fila.setLayoutParams(layoutFila);

     txtId = new TextView(this);
     txtNombre = new TextView(this);
     txtH = new TextView(this);
     txtD = new TextView(this);
     txtB = new TextView(this);
     txtId.setText(String.valueOf(canal));
     txtId.setGravity(Gravity.CENTER_HORIZONTAL);
     txtId.setTextAppearance(this,R.style.etiqueta);
     txtId.setBackgroundResource(R.drawable.tabla_celda);
     txtId.setLayoutParams(layoutId);
     txtNombre.setText((String.valueOf((char)dato)));
     txtNombre.setGravity(Gravity.CENTER_HORIZONTAL);
     txtNombre.setTextAppearance(this,R.style.etiqueta);
     txtNombre.setBackgroundResource(R.drawable.tabla_celda);
     txtNombre.setLayoutParams(layoutId);
     txtH.setText(String.valueOf(Integer.toHexString(dato)));
     txtH.setGravity(Gravity.CENTER_HORIZONTAL);
     txtH.setTextAppearance(this,R.style.etiqueta);
     txtH.setBackgroundResource(R.drawable.tabla_celda_cabecera);
     txtH.setLayoutParams(layoutId);
     txtD.setText(String.valueOf(dato));
     txtD.setGravity(Gravity.CENTER_HORIZONTAL);
     txtD.setTextAppearance(this,R.style.etiqueta);
     txtD.setBackgroundResource(R.drawable.tabla_celda_cabecera);
     txtD.setLayoutParams(layoutId);
     txtB.setText(String.valueOf(Integer.toBinaryString(dato)));
     txtB.setGravity(Gravity.CENTER_HORIZONTAL);
     txtB.setTextAppearance(this,R.style.etiqueta);
     txtB.setBackgroundResource(R.drawable.tabla_celda_cabecera);
     txtB.setLayoutParams(layoutTexto);
     fila.addView(txtId);
     fila.addView(txtNombre);
     fila.addView(txtH);
```

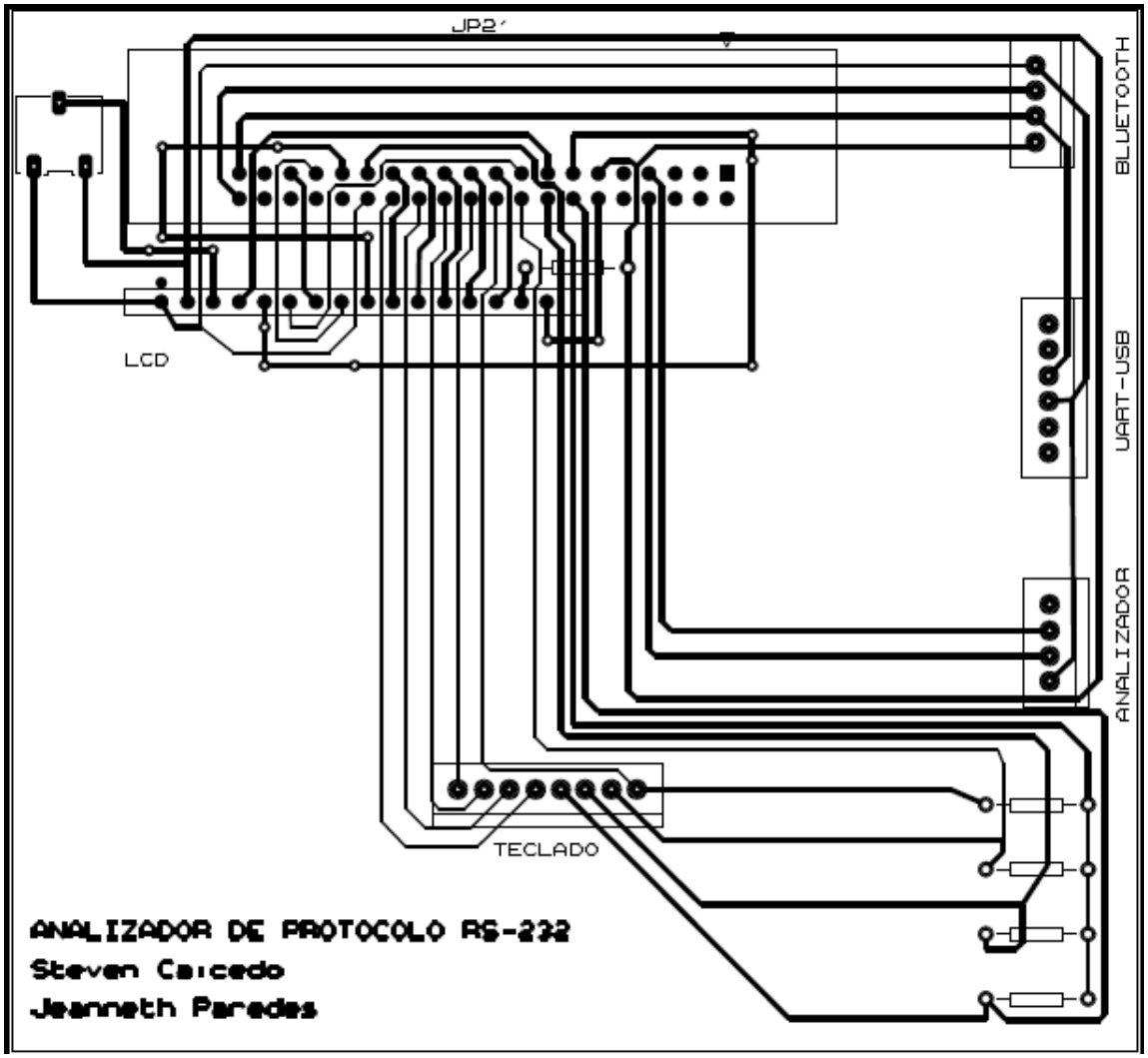
```
        fila.addView(txtD);
        fila.addView(txtB);

        tabla.addView(fila);

    }
    public void agregarstring(String dato){

     TableRow fila;
     TextView txtId;
     TextView txtNombre;
     int canal=0;

        //int posicion = i + 1;
        fila = new TableRow(this);
        fila.setLayoutParams(layoutFila);

        txtId = new TextView(this);
        txtNombre = new TextView(this);
        txtId.setText(String.valueOf(canal));
        txtId.setGravity(Gravity.CENTER_HORIZONTAL);
        txtId.setTextAppearance(this,R.style.etiqueta);
        txtId.setBackgroundResource(R.drawable.tabla_celda);
        txtId.setLayoutParams(layoutId);
        txtNombre.setText((String.valueOf(dato)));
        txtNombre.setGravity(Gravity.CENTER_HORIZONTAL);
        txtNombre.setTextAppearance(this,R.style.etiqueta);
        txtNombre.setBackgroundResource(R.drawable.tabla_celda);
        txtNombre.setLayoutParams(layoutTexto);
        fila.addView(txtId);
        fila.addView(txtNombre);
        tabla.addView(fila);

    }

     private BluetoothSocket createBluetoothSocket(BluetoothDevice
            device) throws IOException {
      if(Build.VERSION.SDK_INT >= 10){
          try {  final Method  m  = device.getClass().getMethod(
                      "createInsecureRfcommSocketToServiceRecord",
                       new Class[] { UUID.class });
              return (BluetoothSocket) m.invoke(device, MY_UUID);
          } catch (Exception e) {
              Log.e(TAG, "Could not create Insecure RFComm
                    Connection",e);
          }
      }
      return  device.createRfcommSocketToServiceRecord(MY_UUID);
      }
       @Override
public void onResume() {
  super.onResume();

  Log.d(TAG, "...onResume - try connect...");
```

```
    // intentar Conectarse usando la direccion mac.
    BluetoothDevice device = btAdapter.getRemoteDevice(address);

    try {
        btSocket = createBluetoothSocket(device);
    } catch (IOException e) {
        errorExit("Fatal Error", "In onResume() and socket create
                  failed: " + e.getMessage() + ".");
    }

    // Establecer conexion

    try {
      btSocket.connect();

    } catch (IOException e) {
      try {
        btSocket.close();
      } catch (IOException e2) {
        errorExit("Fatal Error", "In onResume() and unable to close
         socket during connection failure" + e2.getMessage() + ".");
      }
    }

    // Creacion de data stream para hablar con el servidor
    Log.d(TAG, "...Create Socket...");

    mConnectedThread = new ConnectedThread(btSocket);
    mConnectedThread.start();
    mConnectedThread.write("hola");
  }
      @Override
  public void onPause() {
    super.onPause();
      Log.d(TAG, "...In onPause()...");
    try     {
      btSocket.close();
    } catch (IOException e2) {
      errorExit("Fatal Error", "In onPause() and failed to close
       socket." + e2.getMessage() + ".");
    }
  }

  private void checkBTState() {
    // Verificar si el dispositivo soporta bluetooth y si esta
habilitado
    if(btAdapter==null) {
      errorExit("Fatal Error", "Bluetooth not support");
    } else {
      if (btAdapter.isEnabled()) {
        Log.d(TAG, "...Bluetooth ON...");
      } else {
        //Solicitar al usario que encienda el Bluetooth
```

```
        Intent enableBtIntent =
          new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, 1);
      }
    }
  }
  private void errorExit(String title, String message){
    Toast.makeText(getBaseContext(), title + " - " + message,
    Toast.LENGTH_LONG).show();
    finish();
  }
// Clase interna conexion
    private class ConnectedThread extends Thread {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;
        public ConnectedThread(BluetoothSocket socket) {
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        @Override
        public void run() {
// buffer reservado para el stream
            byte[] buffer = new byte[256];
            int bytes;
// Mantenerce escuchando el InputStream mientras no ocurra una
excepcion
            while (true) {
                try {
                    // Leer el InputStream
                    bytes = mmInStream.read(buffer);
h.obtainMessage(RECIEVE_MESSAGE, bytes, -1, buffer).sendToTarget();
// Enviar el mensaje al manejador de cola
                } catch (IOException e) {
                    break;
                }
            }
        }


        }
    }
}
```

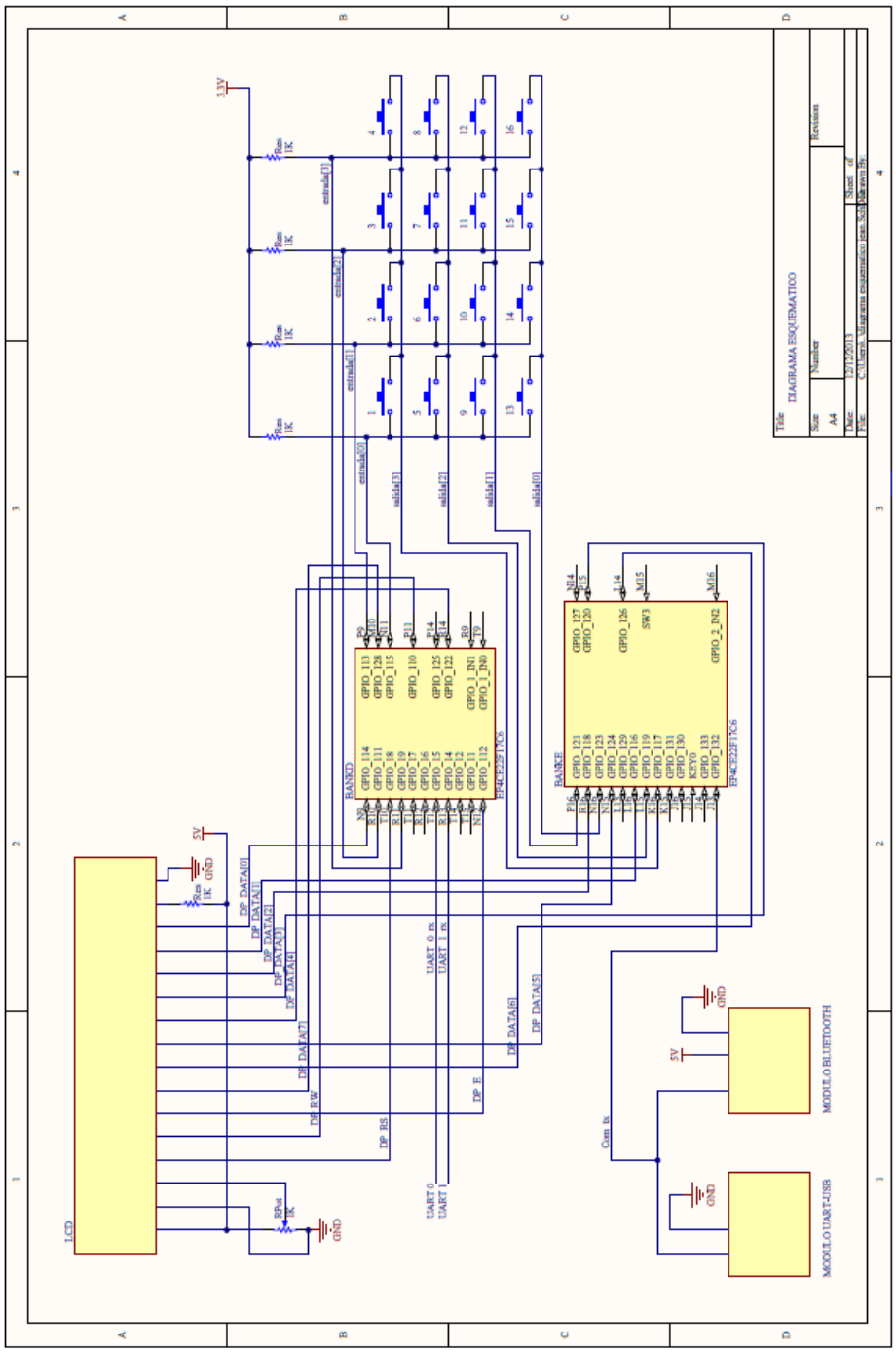
ANEXO E

PCB


JP2
BLUETOOTH
LCD
UART-USB
ANALIZADOR
TECLADO
ANALIZADOR DE PROTOCOLO RS-232
Steven Caicedo
Jeanneth Paredes

ANEXO F

DIAGRAMA ESQUEMATICO



BIBLIOGRAFÍA

[1] Jorge Rodríguez Araújo, Microprocesador NIOS II.

http://es.scribd.com/doc/28358833/Estudio-del-microprocesador-Nios-II

Fecha de consulta: Septiembre 2013

[2] Javier Gimbert Moreno, Análisis y diseño de un procesador RISC simple para adquisición y proceso de datos. http://www.recercat.cat/bitstream/handle/2072/5416/PFCGimbertMoreno.pdf?sequence=1

Fecha de consulta: Septiembre 2013

[3] Altera, DE0 nano User Manual

http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=593&FID=75023fa36c9bf8639384f942e65a46f3

Fecha de consulta: Septiembre 2013

[4] Ronald Mijail Dueñas, EL ESTÁNDAR RS-232 y V24

http://interface-serial-rs232.blogspot.com/

Fecha de consulta: Septiembre 2013

[5] Altera, Nios Uart Data Sheet

http://extras.springer.com/2001/978-0-306-47635-8/ds/ds_nios_uart.pdf

Fecha de consulta: Septiembre 2013

[6] Altera, Optrex 16207 LCD Controller Core

http://myweb.wit.edu/johnsont/Classes/667/LCD_DE2_Nios.pdf

Fecha de consulta: Septiembre 2013

[7] RXTX wiki

http://rxtx.qbang.org/wiki/index.php/FAQ

Fecha de consulta: Septiembre 2013

[8] Universidad Complutense de Madrid

http://pendientedemigracion.ucm.es/info/tecnomovil/documentos/android.pdf

Fecha de consulta: Septiembre 2013