ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"SISTEMA PARA ELABORACIÓN DE MAPAS DE NAVEGACIÓN PARA ROBOTS MÓVILES UTILIZANDO NIOS II"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Ronny Andrés Carvajal Falcón

José Luis Kam Zou

GUAYAQUIL – ECUADOR

AÑO 2014

AGRADECIMIENTO

Primeramente a Dios todo poderoso por darme la fuerza y la sabiduría para terminar esta tan importante etapa de mi vida.

A mi familia por brindarme todo su apoyo moral y económico.

A nuestro profesor del seminario de graduación el Ing. Ronald Alberto Ponguillo Intriago por todas sus valiosas enseñanzas.

José Luis Kam Zou

Gracias a Dios por haberme permitido llegar a esta meta dándome salud para lograr uno a uno mis objetivos con su amor y bondad.

Agradezco a mi familia y amigos por su apoyo incondicional en mis aciertos y errores siendo el pilar fundamental para mi formación humana y profesional.

A los profesores que marcaron cada etapa de nuestro camino universitario aportando cada uno con mi educación.

Todo este trabajo es posible gracias a ellos.

Ronny Andrés Carvajal Falcón

DEDICATORIA

Dedico este trabajo con mucho orgullo, esfuerzo y empeño a aquellas personas que estuvieron velando por mi progreso. A Dios por no permitirme desmayar sin haber primero cumplido con una de mis metas y plasmar en estas hojas el anhelo de tantos años de estudio. A mis padres Rafael y Shaoling por tantos buenos valores inculcados en mí, por todo ese esfuerzo económico que realizaron y por brindarme su amor incondicional. A mis hermanos Vicente y Cecilia por creer en mí, y he aquí el fruto de aquello. A mis amigos cercanos Fernanda, Mayra, Karen, Germán, Marcelo, Milton, Ronny. Millón gracias.

José Luis Kam Zou

A mi madre Melva por nunca dejar de creer en mí siendo ejemplo de perseverancia, sacrificio y trabajo, por el valor mostrado para salir adelante y por todo su amor. A mi padre Adolfo (QEPD) por sentirse siempre orgulloso de mí, por sembrar las ganas de buscar y conseguir grandes metas.

A mi hermano Danny por ser ejemplo como hermano mayor, a mi tía Fanny y mi primo Oscar por hacer más grande el significado de la palabra Familia.

A mis amigos que nos apoyamos en nuestra carrera profesional Fernanda Hidalgo, Mayra Macías, Germán Ramos, Marcelo Torres, Milton Romero y en especial a José Luis Kam por ayudarme a realizar este trabajo.

Ronny Andrés Carvajal Falcón

TRIBUNAL DE SUSTENTACIÓN

Ing. Ronald A. Ponguillo Intriago

PROFESOR DEL SEMINARIO DE GRADUACIÓN

Ing. Luis F. Vásquez Vera

PROFESOR DELEGADO POR EL DECANO DE LA FACULTAD

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este trabajo, 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)

_____

Ronny Andrés Carvajal Falcón.

_____

José Luis Kam Zou.

RESUMEN

El proyecto consiste en la implementación de un sistema con el programa NIOS II Trabajando sobre la Tarjeta FPGA, para lo siguiente realizamos la programación en Lenguaje C para elaborar un mapa de navegación móvil mostrada en una interfaz gráfica.

El sistema recepta y procesa la información mediante comunicación inalámbrica bluetooth, esta información es enviada por un robot armado con el Kit Mindstorms NXT 2.0 que recorre un entorno cerrado seleccionado por el usuario.

El robot mediante el uso de sensores del kit realizara un recorrido sin colisión a través de los diferentes obstáculos ubicados en el ambiente siendo capaz de dimensionar y ubicar la posición de los objetos, esto será mostrado por medio

de una interfaz simulando un mapa con coordenadas exactas de cada obstáculo.

El sistema almacena la información enviada durante el recorrido del robot que tendrá como limitante un punto de inicio y fin descrito por el usuario, luego de esto el robot mediante una serie de algoritmos tendrá una función autónoma seleccionando apropiadamente el camino para llegar al objetivo dado por el usuario mostrando el mapa del entorno recorrido

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

ASIC	Application Specific Integrated Circuit.
CPLD	Complex Programmable Logic Device.
EEPROM	Electrically Erasable Programmable Read Only Memory.
EPCS	Erasable Programmable Configurable Serial.
FHSS	Frecuency Hopping Spread Spectrum.
FPGA	Campo Matriz de Puertas Programables.
GFSK	Gaussian Frecuency Shift Keying
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
I2C	Inter Integrated Circuit
JTAG	Joint Test Action Group
MIT	Massachusetts Institute of technology
PCB	Printed Circuit Board
PDA	Personal Digital Assistant
PLD	Programmable Logic Device
PLL	Phase Lock Loop

RS-232	Recommended Standard 232
SDRAM	Synchronous Dynamic Random-Access Memory
SMT	Surface-mount technology
SOPC	System on a Programable Chip
UART	Universal Asynchronous Receiver-Transmitter
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language

INTRODUCCIÓN

La robótica se ha posicionado como una de las expresiones de ingeniería más dominantes en el campo de las aplicaciones y prototipos, ya que integra muchas disciplinas que aportan al desarrollo de robots que actualmente perciben e interactúan con el ambiente, si se utilizan robots para interactuar con el entorno, se están integrando los campos de la inteligencia ambiental con la robótica autónoma lo cual da lugar a ingresar en un campo más complejo de investigar que converge en la inteligencia artificial.

Sin lugar a dudas, uno de los pasos imprescindibles para lograr que los robots puedan estar presentes de forma natural en entornos inteligentes, es que adquieran habilidades similares a las humanas. Además de nuestra inteligencia una de las capacidades más interesantes que el ser humano posee es la comprensión del espacio y la posibilidad de realizar cualquier tarea espacial con gran precisión.

Esta capacidad se debe a la constante necesidad de interactuar con el entorno que nos rodea, con este gran precedente esta capacidad ha evolucionado y se ha perfeccionado a través del tiempo. Por lo tanto, uno de los requerimientos necesarios para cualquier robot que se desee ingresar en entornos humanos es que sea capaz de comprender, interpretar y representar el entorno de manera eficiente y consistente.

Durante las últimas décadas se han realizado importantes avances en el desarrollo de estrategias de localización y la construcción de mapas, el manejo de la información simbólica es el recurso más fiable en la actualidad pero este no resuelve la dificultad de que un mismo objeto puede dar lugar a la proyección de un número infinito de imágenes distintas en el plano de la imagen del observador; dependiendo de la posición, orientación y tamaño del objeto dando lugar a la inclusión del robot en un sistema autónomo.

En el primer capítulo, describimos los objetivos de realizar este proyecto, con una visión más profunda sobre el tema analizando la metodología a utilizar con

sus limitaciones y aplicaciones que podrían servir, así como la identificación de los problemas que podamos tener.

En el segundo capítulo, se da la fundamentación teórica de los recursos de Hardware y Software utilizados tales como FPGA CYCLONE IV, MÓDULO SERIAL BLUETOOTH, MODULO SERIAL USB y el KIT NXT. Así como, QUARTUS II, ECLIPSE y NETBEANS IDE.

En el tercer capítulo, describe el proceso de diseño y posterior implementación del proyecto. Además de detallar los códigos de programación, diagramas de flujos indicando el comportamiento del sistema.

En el cuarto capítulo se realizan las pruebas, simulaciones y se muestran los resultados obtenidos luego de haber finalizado el proyecto, analizar la efectividad del sistema.

CAPÍTULO 1

1. DESCRIPCIÓN GENERAL DEL PROYECTO

1.1. OBJETIVOS

1.1.1 OBJETIVO GENERAL

Implementar un algoritmo estratégico en NIOS II SBT que pueda percibir la presencia de objetos de un entorno específico y que a su vez los represente mediante una interfaz con la finalidad de crear un mapa de navegación de robots autónomos.

1.1.2 OBJETIVOS ESPECÍFICOS

- Aprender el uso y manejo de la Tarjeta DE0-Nano de Altera.
- Implementar un sistema embebido que contenga los módulos necesarios para interactuar con los componentes externos de manera adecuada.
- Establecer comunicación inalámbrica bluetooth de nuestro sistema embebido con el bloque de programación LEGO MINDSTORM NXT.
- Establecer comunicación USB-UART entre nuestro sistema embebido y una aplicación desarrollada en JAVA que represente el entorno.
- Aprende el uso de los DIRECT-COMMANDS para interactuar con los sensores y controlar los servomotores que conforman el robot autónomo.
- Elaborar un Software mediante la herramienta NIOS II para Eclipse que pueda procesar la información

obtenida de los sensores y controlar los movimientos del robot.

- Desarrollar una aplicación en JAVA usando NETBEANS IDE que reciba las coordenadas de los objetos percibidos y los represente de manera gráfica simulando un mapa.

1.1.3 IDENTIFICACIÓN DEL PROBLEMA

La necesidad que surge en el campo de la Robótica hace no más de 20 años atrás es la de mantener perfectamente localizado y orientado a un robot autónomo, principalmente en el interior de un entorno donde desarrolle su servicio demostrando que este tipo de investigación en la actualidad se torna imprescindible, sin importar la edificación donde pueda ser aplicado.

Así pues, a pesar de la existencia de varios modelos matemáticos predictivos estocásticos, al momento de implementar los modelos antes mencionados surgen interrogantes de la veracidad y exactitud de la obtención de los datos del entorno que rodea dicha actividad. Esto implica la acumulación excesiva de errores a largo plazo que dejarían sin validez el mapa y ubicación obtenidos.

La carrera por conseguir este gran objetivo empieza en comprender las mediciones sensoriales individuales de los elementos del robot para que en conjunto den una aproximación del recorrido y forma del entorno.

1.2. METODOLOGÍA

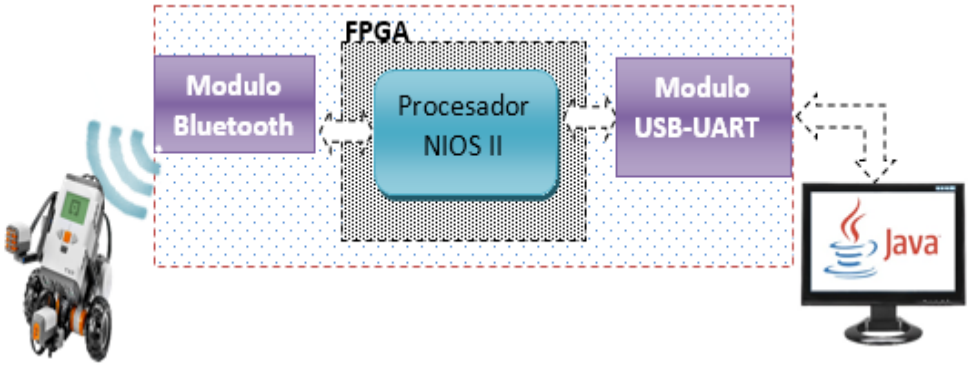


Figura 1.1 Diagrama de bloques del sistema

El método para conseguir el objetivo principal, es diseñar y ensamblar un robot usando el KIT LEGO MINDSTORM NXT que cuente con dos sensores ultrasónicos para percibir el entorno en que se encuentre, de los cuales se tendrá que realizar un estudio para determinar su comportamiento. Con la información adquirida de los sensores se podrá tomar decisiones para establecer el siguiente movimiento del robot.

Establecer un enlace de comunicación bluetooth usando el dispositivo RN42 para extraer los datos de la mediciones de distancia de los sensores y enviar comandos de control del movimiento del robot.

Desarrollar una aplicación JAVA que simule el entorno en el que se encuentre el robot, esta aplicación deberá ubicar cada uno de los obstáculos percibidos por el robot durante su recorrido con la finalidad de obtener una representación gráfica aproximada del entorno.

Establecer la comunicación entre el sistema embebido y la aplicación desarrollada en JAVA usando el modulo USB-UART FT232 para que la aplicación obtenga los datos procesados por el sistema embebido mediante coordenadas.

1.3. ALCANCES

- Los movimientos del robot se realizan de manera autónoma de acuerdo a los resultados obtenidos mediante los sensores ultrasónicos.

- La representación gráfica del entorno se realiza de manera continua mientras el robot sigue avanzando.

- Las lecturas de los sensores se realizan mientras el robot se encuentra detenido para obtener una medición aproximada de la distancia real hacia el objeto.

- El uso de la interfaz de comunicación inalámbrica entre el sistema embebido y el robot facilita la movilidad y ubicación del dispositivo en el que se encuentre ejecutando la aplicación JAVA.

1.4. LIMITACIONES

- La distancia a la que puede navegar el robot móvil debe estar dentro del alcance del dispositivo bluetooth. El robot NXT de LEGO cuenta con un módulo bluetooth clase B con alcance de 20 metros.

- Al trabajar de manera inalámbrica no podemos controlar de manera constante la medición de los dos sensores ultrasónicos debido a problemas de protocolo que maneja el bloque NXT.

- El tamaño del robot debe ser considerado para referenciar las coordenadas del entorno.

- La altura de los objetos debe ser considerada en base a la ubicación de los sensores.

1.5. APLICACIONES

Son muchos los edificios que hoy en día utilizan microprocesadores, sensores y actuadores para medir y controlar la iluminación, la climatización, o los sistemas de seguridad de forma automatizada. Sin embargo, la tecnología actual es todavía bastante limitada en cuanto a inteligencia y autonomía, y los avances más innovadores en estos aspectos se están realizando gracias al progreso de la inteligencia ambiental que se refiere a un mundo en el que los microcomputadores estén tan perfectamente integrados e interconectados entre sí en cualquier parte del entorno, que se puedan utilizar sin ser conscientes de estar interactuando con ellos.

Para poder aplicar la inteligencia ambiental en el mundo real es imprescindible disponer de la capacidad de percibir y actuar en el entorno. Esto bien se puede lograr por medio de robots.

La navegación de un robot móvil en ambientes desconocidos depende estrictamente de un mapa especialmente en entornos cerrados. Se puede combinar este sistema de navegación e identificación para dar autonomía a robots que interactúen en ambientes con personas y las variantes del medio como por ejemplo: robots para hospitales, robots para el rescate de personas encerradas en lugares pequeños por desastres naturales, sistema móvil de seguridad y así un sin número de ejemplos de móviles que necesiten utilizar navegación sin ningún tipo de monitoreo y modificación externa.

CAPÍTULO 2

2. MARCO TEÓRICO.

2.1. HARDWARE

2.1.1. TARJETA DE DESARROLLO DE0-NANO

La Tarjeta de Desarrollo DE0-Nano diseñada por la empresa Terasic presenta una plataforma FPGA de tamaño compacto adecuado para el desarrollo de diseños de circuitos prototipos tales como robots y proyectos "portátiles".

La placa está estrictamente diseñada para ser utilizada en la aplicación más sencilla posible, dirigida al dispositivo de ciclón IV, cuenta con una colección de interfaces, incluyendo dos GPIO headers externos para extender más allá de los diseños del tablero de0-Nano, contiene dispositivos de memoria SDRAM y EEPROM para mayor almacenamiento de datos y almacenamiento en búfer, así como el uso general de los perifericos como LEDs y pulsadores.

Las ventajas del DE0-Nano son su tamaño y peso, así como su capacidad para ser reconfigurado sin llevar hardware superfluo, para distinguirse de otras tarjetas de desarrollo. Además, para los diseños móviles donde el poder portátil es crucial, el DE0-Nano proporciona a los diseñadores tres opciones de combinación de energía que incluyen un mini-USB AB puerto, 2-pin headers de alimentación externa y dos DC 5V pin

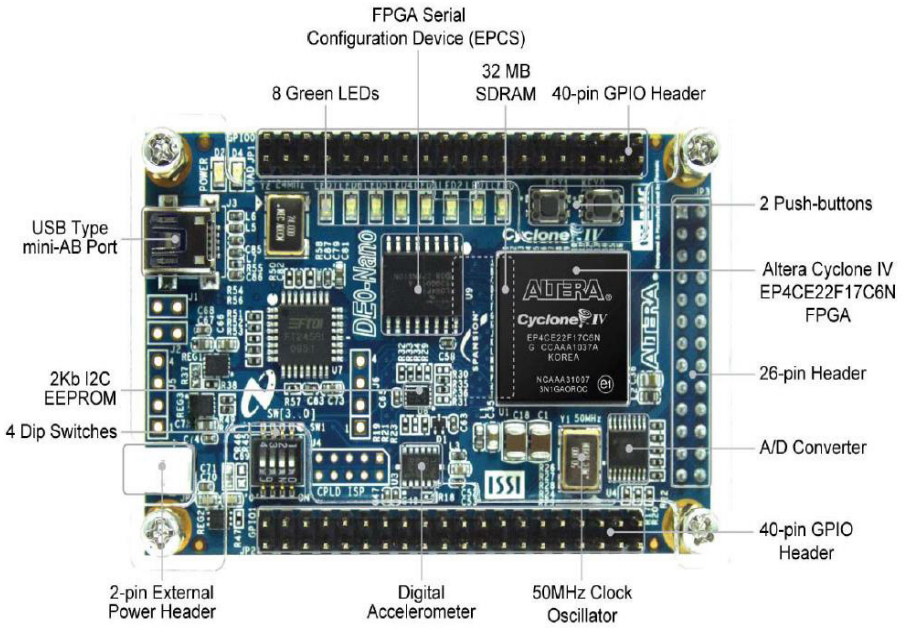


Figura 2.1 Tarjeta DE0-NANO

La tarjeta DE0-NANO esta basada en la Familia Lógica Programable FPGA Cyclone IV, chip desarrollado por Altera que tiene 153 pines de I/O en donde están conectados todos los dispositivos embebidos en la tarjeta, permite al usuario controlar todos los componentes usando un sistema NIOS II almacenado en la FPGA Cyclone IV.

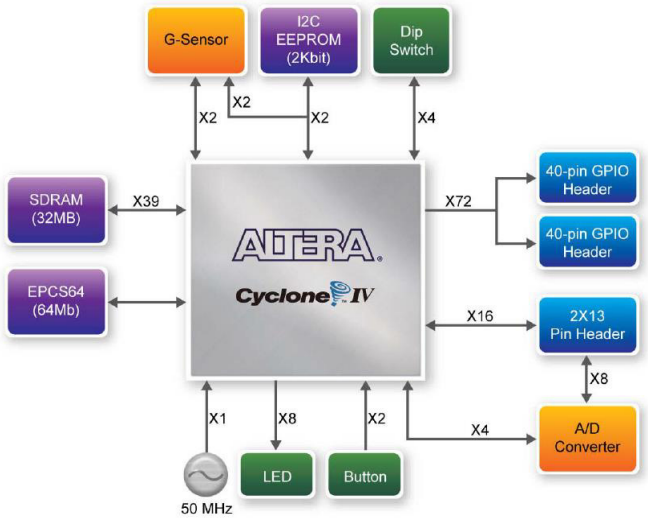


Figura 2.2 Diagrama de bloques de la tarjetaDE0-NANO

La tarjeta DE0-NANO tiene las siguientes características:

- FPGA Altera Cyclone IV EP4CE22 con 153 pines I/O
- Circuito USB-Blaster para configurar la FPGA
- Dispositivo de configuración serial EPCS16 de 16Mbits
- 2 cabeceras de expansión de 40 pines
- 1 cabecera de expansión de 26 pines
- Memoria SDRAM de 32MB
- EEPROM de 2Kbits con comunicación I2C
- 8 LEDs verdes
- 2 Pulsadores

- 4 interruptores DIP
- Acelerómetro de 3 ejes con una resolución de 13 bits
- Convertidor A/D de 8 canales y 12 bits de resolución
- Reloj de 50MHz
- Puerto USB tipo mini-AB (5V)
- Conector para fuente de poder externa 3.6v - 5.7V

2.1.2. FPGA CYCLONE IV

Las FPGA (del inglés Field Programmable Gate Array) son circuitos lógicos programables directamente por el usuario, lo cual requiere de herramientas de costo relativamente bajo. La grabación o programación de uno de estos dispositivos se puede llevar a cabo en milisegundos.

Básicamente, es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir

desde funciones tan sencillas como las llevadas a cabo por una compuerta lógica o un sistema combinatorial hasta complejos sistemas en un chip.

Una jerarquía de interconexiones programables permite a los bloques lógicos de una FPGA ser interconectados según la necesidad del diseñador del sistema. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario/diseñador, así que la FPGA puede desempeñar cualquier función lógica necesaria. Se combinan a los bloques lógicos e interconexiones de las FPGA con microprocesadores y periféricos, para formar un sistema programable en un chip.

Además, muchas FPGA modernas soportan la reconfiguración parcial del sistema, permitiendo que una

parte del diseño sea reprogramada, mientras las demás partes siguen funcionando.

Figura 2.3 FPGA CYCLONE IV

El diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas digitales en FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL (Hardware Description Language).

Los HDLs más utilizados son:

- VHDL
- Verilog
- ABEL

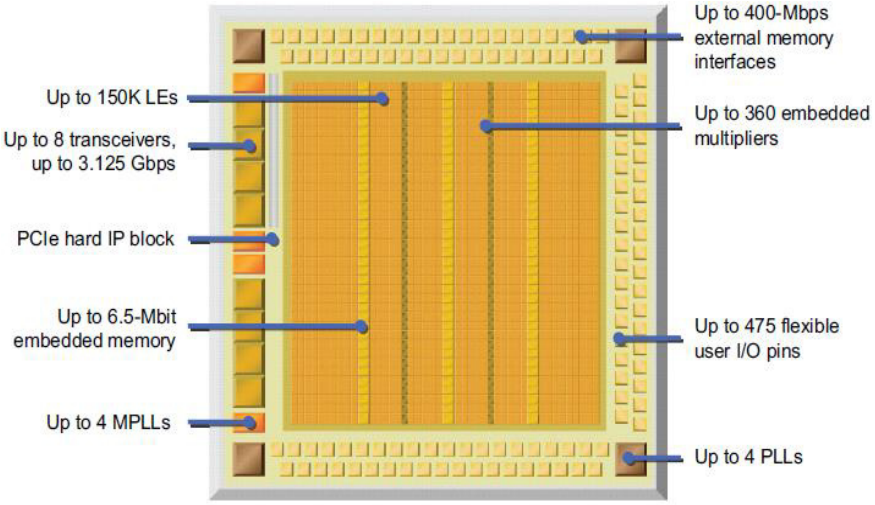


Figura 2.4 Arquitectura de la FPGA Cyclone IV

El núcleo de la FPGA Cyclone IV consiste de elementos lógicos compuestos de 4 entradas de tabla de consultas LUTs, bloques de memoria M9K y multiplicadores. Cada bloque de memoria M9K ofrece 9Kbits de memoria SRAM embebida y es posible configurar cada bloque como un “single port”, “simple dual port” o “true dual port RAM”, así

como también buffer FIFO o ROM. Los multiplicadores pueden implementarse como un único bloque de 18x18 o como 2 bloques de 9x9 multiplicadores.

Además incluye una red de hasta 20 clock globales (GCLK), soporta hasta 4 PLLs con 4 salidas por PLL y ofrecer soporte para memorias SDR, DDR, DDR2 SDRAM y QDRII SRAM. Los datos de configuración de la FPGA Cyclone IV se almacenan en celdas de memoria SRAM cada vez que el dispositivo es energizado.

La FPGA Cyclone IV EP4CE22 ofrece las siguientes características:

- Voltaje del núcleo 1.0v y 1.2v
- 22,320 elementos lógicos
- 594Kbits de memoria embebida
- 66 multiplicadores embebidos de 18x18
- 4 PLLs de propósito general
- Red global de 20 clock

- 8 bancos de I/O
- 153 pines I/O
- Soporte para DDR2 SDRAM de hasta 200MHz

2.1.3. COMUNICACIÓN UART RS-232

El corazón del sistema de comunicaciones serie es la UART, “Universal Asynchronous Receiver/Transmitter”. Es un controlador cuya misión principal es convertir los datos recibidos del bus del PC o en este caso los datos recibidos del bus Avalon de la DE0-NANO en formato paralelo, a un formato serie que se utiliza en la transmisión hacia el exterior.

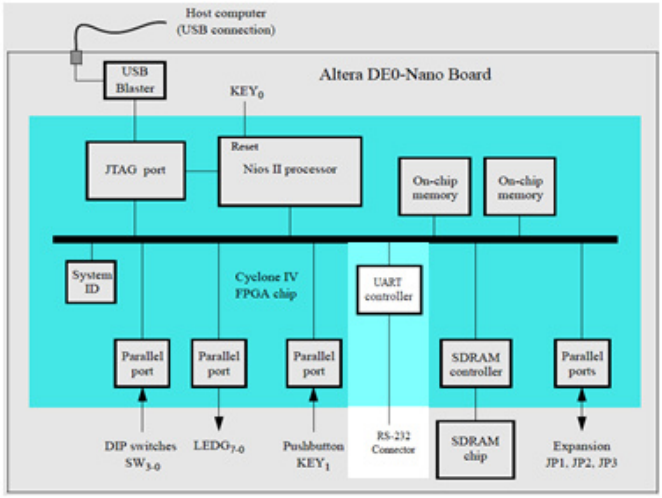


Figura 2.5 Controlador UART en un sistema NIOS II

El controlador UART implementa el protocolo RS-232 y es un dispositivo configurable en el que pueden establecerse las condiciones que se utilizarán para la transmisión (velocidad, paridad, longitud y bits de parada). Además de las líneas de transmisión TX y recepción RX, las comunicaciones seriales poseen otras líneas de control de flujo RTS/CTS (Hand-shake), donde su uso es opcional dependiendo del dispositivo a conectar.

El driver provee un simple mapa de registro Avalon Memory-Mapped ESCLAVO que permite al procesador NIOS II comunicarse con él a través del bus Avalon Memory-Mapped MASTER para poder leer y escribir registros de control y datos.

A nivel de software, la configuración principal que se debe dar a una conexión a través de puertos seriales RS-232 es básicamente la selección de la velocidad en baudios (1200,

2400, 4800, 9200, etc.), la verificación de datos o paridad (paridad par o paridad impar o sin paridad), los bits de parada luego de cada dato (1 ó 2), y la cantidad de bits por dato (7 u 8), que se utiliza para cada símbolo o caracter enviado. Toda esta configuración se la realiza desde el software SOPC BUILDER durante la implementación de los componentes que conforman la FPGA.

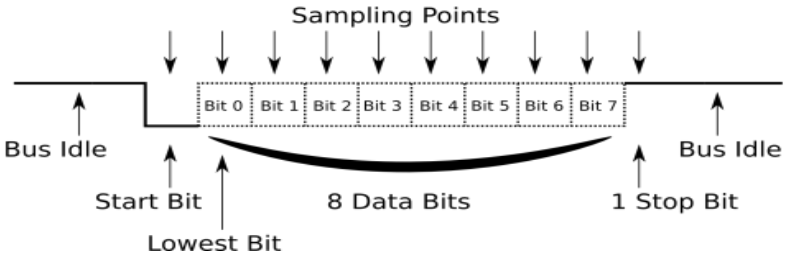


Figura 2.6 Trama de la comunicación UART

El UART toma bytes de datos y transmite los bits individuales de forma secuencial. En el destino, un segundo UART re-ensambla los bits en bytes completos. La transmisión serie de la información digital (bits) a través de un cable único u otros medios es mucho más efectiva en cuanto a costo que la transmisión en paralelo a través de múltiples cables. Se utiliza un UART para convertir la información transmitida entre su

forma secuencial y paralela en cada terminal de enlace. Cada UART contiene un registro de desplazamiento que es el método fundamental de conversión entre las formas serie y paralelo.

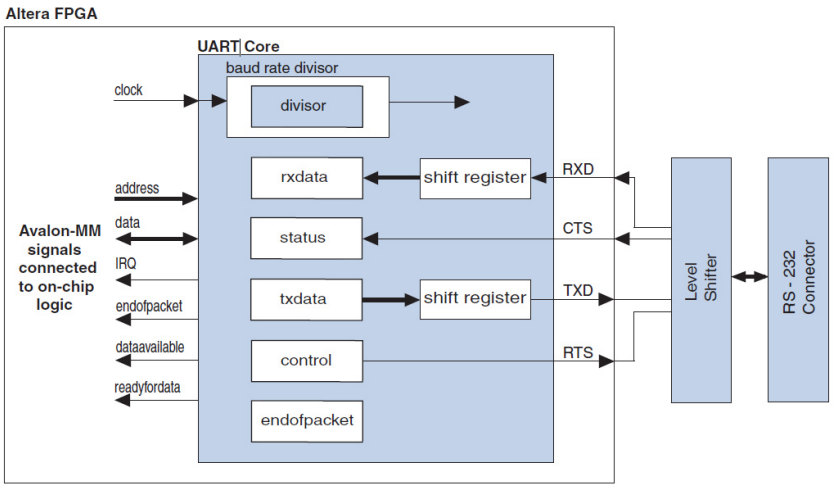


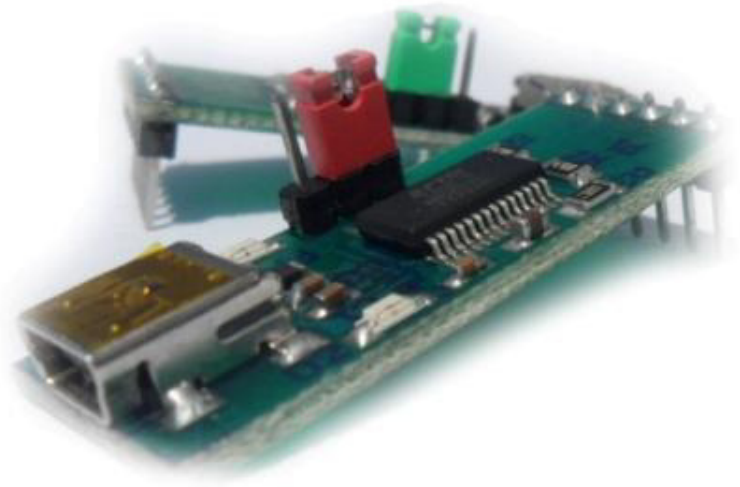
Figura 2.7 Diagrama de bloques del CORE UART de SOPC BUILDER

Para nuestro proyecto usamos el controlador UART RS-232 que se encuentra disponible en las librerías de SOPC BUILDER listo para ser integrado en la FPGA de la tarjeta DE0-NANO.

La tarjeta de trabajo DE0-NANO no cuenta con un conector RS-232 tipo DB-9 pero utilizaremos 3 pines del puerto de expansión GPIO-1 para comunicar la tarjeta DE0-NANO y el modulo Bluetooth. Consulte la Tabla 1-1.

2.1.4. USB - UART FT232R

Es un modulo de comunicación serial por puerto USB para hacer interface con un computador o PC, permite alimentar circuitos de 5V.



.

Figura 2.8 Modulo FT232R

ESPECIFICACIONES:

- Cable mini –USB.
- Leds indicadores de TX y RX.
- Transferencia de datos 300 Baud hasta 3MBaud.
- Soporta 7 a 8 bits datos, 1 o 2 bits stop, y odd/even/mark/space/no parity.
- Datos serial con amplitud seleccionable de 5V/3.3V.

CARACTERÍSTICAS

- Fuente de Alimentación
- Alimentación por interfaz USB desde la PC.
- Señales de Control:
    - VCC: Salida de voltaje 5V
    - TX: Transmisión de Datos
    - RX: Recepción de Datos
    - GND: Tierra

APLICACIONES:

- Interface de comunicación serial con el computador
- Adquisición y envío de datos desde circuitos con microcontroladores hacia el computador.

2.1.5. MODULO BLUETOOTH RN-42

El Bluetooth es una tecnología orientada a la conectividad inalámbrica entre distintos dispositivos como PCs, PDAs, teléfonos móviles, electrodomésticos, equipos de sonido, etc. El Bluetooth, aparte de ser una nueva tecnología, es también una especificación abierta para comunicaciones inalámbricas de voz y datos.

Está basado en un enlace de radio de bajo coste y corto alcance, el cual proporciona conexiones instantáneas (ad-hoc) tanto para entornos de comunicaciones móviles como estáticos. La principal ventaja que ofrece esta tecnología es la conectividad sin cables de todos los dispositivos, pero más que reemplazar los incómodos cables, esta tecnología ofrece un puente entre las redes de datos hoy existentes y el exterior.

El Bluetooth, al ser un estándar abierto, pretende conectar una amplia gama de dispositivos sin importar su marca. Sus principales características son:

- Robustez
- Bajo coste
- Necesidad de poca potencia
- Baja complejidad
- Es un estándar global

El modulo Bluetooth RN-42 de la compañía Roving Networks es un dispositivo de clase 2 que provee un rango de alcance entre 10 y 20 metros con un bajo consumo de energía. Es perfecto para aplicaciones de corto alcance alimentado mediante batería. Usa solamente 26μA en modo "sleep" mientras permanece aún conectado y reconocible.

Soporta múltiples perfiles Bluetooth tales como SPP y HCI y provee una interfaz de comunicación UART lo cual facilita su integración simple con sistemas embebidos o para su conexión con dispositivos existentes.

Figura 2.9 Módulo Bluetooth RN-42

Las Características más importantes son:

- Módulo Bluetooth Clase II v2.1 + módulo EDR
- Soporta Módulo Bluetooth 2.1/2.0/1.2/1.1
- Interfaces de conexión de datos UART (SPP o HCI) y USB (sólo HCI)
- Soporta velocidades de datos en modo SPP - 240Kbps slave, 300Kbps master
- Soporta velocidades de datos en modo HCI - 1.5Mbps slave, 3.0Mbps master
- Antena tipo chip
- Alcance: hasta 20m con línea de vista.
- Modulación: FHSS/GFSK (79 canales a intervalos de 1MHz)
- Comunicación segura, encriptación de 128 bits
- Potencia de salida: 4dBm
- Sensitividad: -80dBm
- Consumo de corriente en transmisión/recepción: 25mA/25mA
- Voltaje de alimentación: 3V ~ 3.6V
- Tipo de montaje: SMT

Antes de poner en funcionamiento el módulo bluetooth es necesario realizar una configuración de varios parámetros que garantizarán el correcto funcionamiento en el sistema que se está implementando.

Este módulo tiene 2 modos de funcionamiento:

- **Data Mode:** Es el modo de transmisión en el que todos los comandos son ignorados.
- **Command Mode:** Es el modo de configuración en el que determinados comandos pueden configurar ciertos parámetros de funcionamiento del módulo. Dentro de este modo solo tenemos 60 segundos para realizar la configuración, pasado este tiempo el módulo regresa a Data Mode e ignorara los comandos.

El módulo se lo puede conectar por RS-232 con los acoples respectivos como MAX232 y el divisor de voltaje a un puerto serie por medio de un DB9, se debe abrir el hyperterminal o cualquier programa que permita leer y enviar comandos AT.

La segunda opción es prender el bluetooth de un computador o laptop y por medio de algún programa que controle bluetooth crear un COM virtual que le permita al hyperterminal enviar y recibir datos de forma inalámbrica (de esta manera se conecta TX y RX).

El método de configuración que usamos será a través del puerto RS-232 del PC y usamos el software AccessPort (similar a Hyperterminal); para este método necesitamos crear un circuito para poder comunicar la PC con el módulo. Cuando se lo conecta viene calibrado de fábrica a una velocidad de 115200bps con nombre de fábrica según el serial y en modo esclavo

Para nuestros propósitos nos toca cambiarle el modo de operación de esclavo a master. En modo master el dispositivo no será descubierto por otros dispositivos bluetooth y solo permitirá iniciar comunicaciones, también cambiaremos el nombre del dispositivo y usaremos el baud rate que viene por defecto.

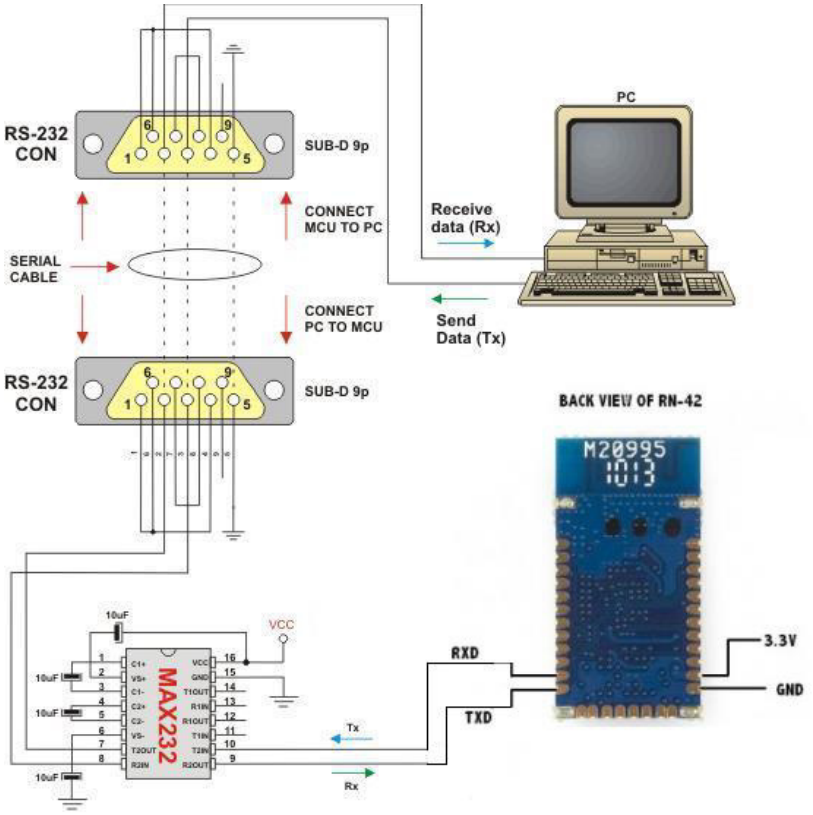


Figura 2.10 Conexión con el PC para la configuración del Módulo Bluetooth RN-42

1. Una vez realizado el circuito abrimos el AccessPort y seleccionamos el número de puerto COM y elegimos baud rate 115200bps.
2. Para entrar en el modo de configuración del módulo se debe enviar 3 signos de dólar $$$.
3. Tenemos 60 segundos para enviar los comandos al dispositivo, ya que si no entra en Data mode y los comandos son ignorados. Si los datos fueron leídos, el modulo responderá con CMD y después del envío de instrucciones nos regresa un AOK. La forma de ver si el modulo está bien, es mirar el led de status, siempre debe estar parpadeando, después de entrar en Command mode la oscilación del led es más rápida y cuando esta enlazado con algún dispositivo el led deja de parpadear y también se enciende el led de estado conectado.
4. Ingrese los comandos 1-4 que se encuentran en la Tabla 2-1 para configurar el módulo bluetooth de acuerdo los requerimientos de nuestro proyecto.

	COMANDO	DESCRIPCION
1	$$$	Modo comando
2	SM,1	Modo master
3	SN,DE0-NANO	Cambiar nombre
4	R,1	Forzar reinicio
5	C	Iniciar conexión
6	K	Terminar conexión
7	H	Muestra lista de comandos
8	D	Muestra configuración básica
9	E	Muestra configuración avanzada

Tabla 2.1 Comandos del módulo bluetooth

La Tabla 2-2 muestra la distribución de pines del módulo bluetooth. Para nuestro proyecto solo usamos 7 pines y construimos un PCB en donde soldamos el módulo SMT y lo acoplamos al puerto GPIO1 de la tarjeta DE0-NANO. Solo detallamos los pines que se utilizan en el proyecto, la información de los otros pines podrá ser consultada en el datasheet.

PIN	NOMBRE	DESCRIPCION
1	GND	Señal de tierra
5	RESET	Reset. Activo en bajo
11	VDD	Señal de voltaje 3.3v
19	PIO2	Alto cuando está conectado
13	UART_RX	Entrada de recepción UART
14	UART_TX	Salida de transmisión UART

Tabla 2.2 Pines del módulo bluetooth usados en el proyecto

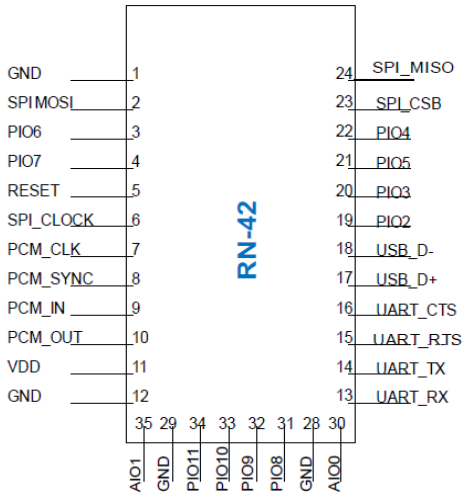


Figura 2.11 Distribucion de pines del modulo bluetooth

2.1.6. ROBOT LEGO MINDSTORM NXT

Lego Mindstorms es una herramienta educacional de robótica fabricado en conjunto por la empresa Lego y el MIT, el cual posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones en forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998. Puede ser usado para construir un modelo de sistema integrado con partes electromecánicas controladas por computador. Prácticamente todo puede ser representado con las piezas tal como en la vida real, como un elevador o robots industriales.

Figura 2.12 ROBOT LEGO MINDSTORM NXT

Para nuestro proyecto usamos el robot LEGO MINDSTORM NXT ya que cuenta con varias características que se requieren para implementar el proyecto como bluetooth incorporado lo cual facilita la comunicación con la tarjeta DE0-NANO, servos motores que permiten controlar el movimiento del robot y una fácil programación gráfica basada en LabView.

El bloque de NXT puede comunicarse con el computador mediante la interfaz de USB que posee, además para comunicarse con otros robots en las cercanías usa una interfaz Bluetooth que es compatible con la Clase II v2.0. Esta conectividad con Bluetooth no sólo permite conectarse con otros bloques, sino también con computadores, palms, teléfonos móviles y otros dispositivos con esta interfaz de comunicación.

La programación del Lego Mindstorms NXT se realiza mediante el entorno de programación gráfico llamado NXT-G el cual usa lenguaje gráfico basado en LabView de National Instrument, también es posible programarlo en RoboLab y directamente en LabView.

El bloque NXT cuenta con 4 botones que nos permite movernos a través del menú de forma fácil y rápida. Posee una amigable interfaz gráfica donde el usuario puede elegir las distintas opciones que presenta el menú.

En la parte superior de la pantalla del NXT, podemos ver el tipo de conexión que estamos usando (Bluetooth y/o USB), el nombre de nuestro robot, el símbolo que indica que está en operación y finalmente el estado de la batería.

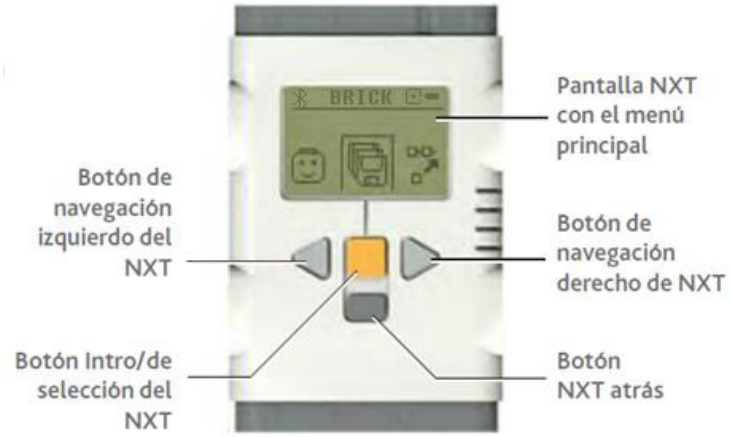


Figura 2.13 Bloque NXT

SIMBOLO	DESCRIPCION
ᛒ	Bluetooth encendido
ᛒ<	Bluetooth visible para otros dispositivos
ᛒ◇	Bluetooth conectado a otro dispositivo
USB	USB conectado y trabajando bien
⊱	USB conectado, con problemas
⊡	NXT operando correctamente
▭	Batería baja
▬	Batería bien
BRICK	Nombre del NXT

Tabla 2.3 Simbología de la interfaz de usuario del bloque NXT

Figura 2.14 Opciones del menú del ROBOT LEGO NXT

Características técnicas:

- Microprocesador Atmel ARM7 de 32 bits 48Mhz
- Memoria FLASH de 256 Kbytes, 64 Kbytes de Memoria RAM
- Microprocesador Atmel AVR de 8 bits 8Mhz
- Memoria FLASH de 4Kbytes, 512 Bytes de RAM
- Comunicación Inalámbrica Bluetooth Clase II v2.0
- Puerta de alta velocidad USB (12 Mbit/s)
- Cuatro puertos de entrada de seis contactos
- Tres puertos de salida de seis contactos
- Pantalla LCD de 100x64 pixeles
- Parlante, calidad de sonido 8KHz
- Fuente de poder: Batería de Litium recargable o 6 baterías AA.

2.2. SOFTWARE

2.2.1. QUARTUS II

Quartus II es una herramienta de software producida por Altera para el análisis y la síntesis de diseños realizados en HDL que permite al diseñador compilar sus diseños de circuitos lógicos, realizar análisis de tiempo, examinar diagramas RTL, simular la reacción de un diseño a diferentes estímulos y configurar el dispositivo de destino con el programador JTAG. Para la realización de este proyecto usamos la versión 12.1 lanzada al mercado en Julio del 2012.

Se trata de un entorno completo para el diseño de SOPC “System-On-a- Programable-Chip” que incluye soluciones para todas las fases de diseño de FPGA y CPLD.

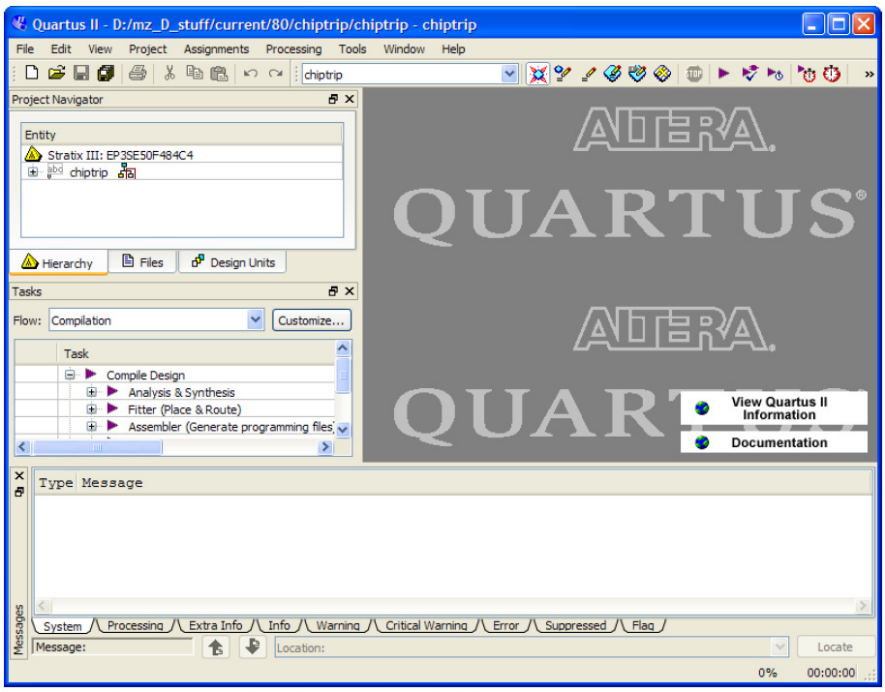


Figura 2.15 Interfaz de usuario del QUARTUS II

Quartus II utiliza una implementación de lenguaje VHDL o Verilog para realizar la descripción de hardware. Ambos lenguajes definidos por la IEEE son usados por ingenieros para describir circuitos digitales. Aunque pueden ser usados de forma general para describir cualquier circuito, se usan principalmente para programar PLD (Programable Logic Device - Dispositivo Lógico Programable), FPGA (Field Programmable Gate Array), ASIC y similares.

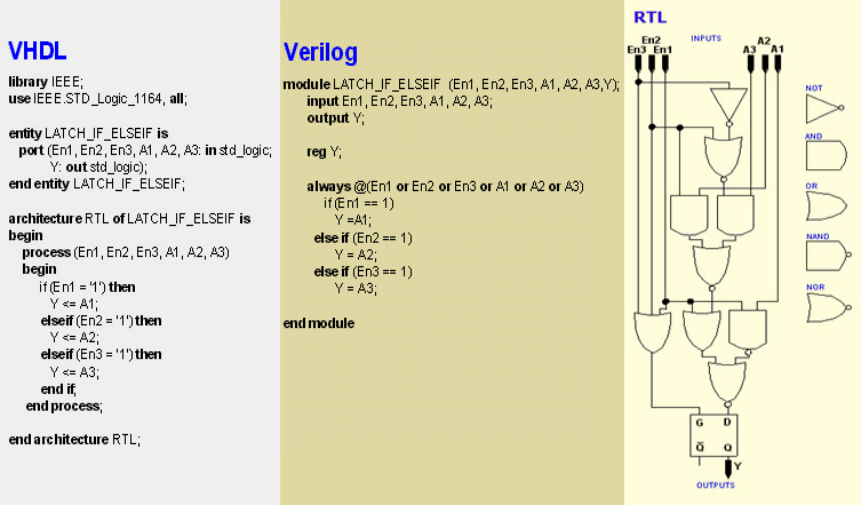


Figura 2.16 Descripcion de un mismo circuito en lenguaje VHDL, VERILOG y RTL

Estos lenguajes están diseñados para cubrir una serie de necesidades en el proceso de diseño ya que nos permite la descripción de la estructura de un sistema, es decir, la forma en que se descompone en subsistemas y como estos subsistemas están interconectados.

Además, permite la especificación de la función de un sistema mediante las conocidas formas de lenguaje de

programación, permitiendo el diseño de un sistema para ser simulado antes de ser fabricado, por lo que los diseñadores pueden rápidamente comparar alternativas poniendo a prueba la corrección sin demora y comparando gastos en la creación de un prototipo de hardware.

2.2.2. SOPC BUILDER

Altera proporciona una infraestructura completa para crear sistemas con microprocesadores embebidos, completamente a medida según las necesidades del diseñador por medio de la combinación de una serie de componentes configurables sobre sus FPGAs.

Para ello, proporciona un entorno específico, al que denomina SOPC BUILDER que se utiliza en conjunto con el software Quartus II permitiendo al usuario diseñar sistemas basados en el procesador NIOS II, simplemente seleccionando las

unidades funcionales deseadas y especificando sus parámetros y puede ser implementado directamente sobre una FPGA de la marca.

Para implementar un sistema útil es necesario añadir otras unidades funcionales. SOPC Builder incorpora una biblioteca de componentes pre-configurados incluyendo el procesador NIOS II, controladores de memoria, interfaces I/O, temporizadores, interfaces de comunicación y periféricos que son interconectados por medio de una red de interconexión llama la matriz de conmutación Avalon (Avalon Switch Fabric), resultando un sistema virtual que puede ser conectado con el mundo exterior a través de los pines programables de la FPGA o conectado internamente a otros componentes.

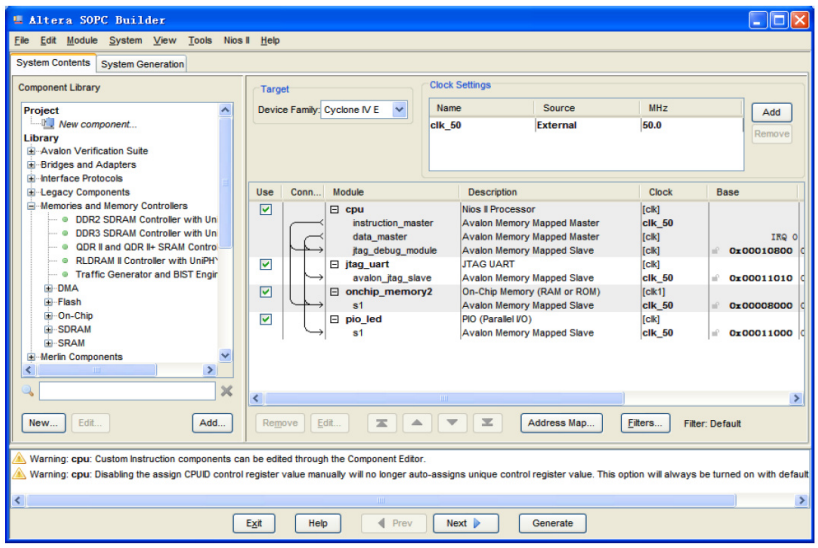


Figura 2.17 interfaz de usuario de SOPC BUILDER

2.2.3. NIOS II SOFTWARE BUILD TOOLS FOR ECLIPSE

NIOS II Software Build Tools for Eclipse es una pequeña capa de interfaz gráfica que presenta un entorno de desarrollo unificado que funciona para todos los procesadores NIOS II.

Es una herramienta que se puede integrar con ECLIPSE un entorno de desarrollo integrado de código abierto multiplataforma desarrollado originalmente por IBM. Se puede llevar a cabo todas las tareas de desarrollo de software

dentro de Eclipse incluyendo creación, edición, construcción, funcionamiento, depuración y perfilado de programas. Por otro lado, Eclipse proporciona amplias capacidades de interacción, depuración y gestión de archivos.

Figura 2.18 Presentación del entorno de desarrollo ECLIPSE

NIOS II Software Build Tools nos permite crear las librerías indispensables para programar sistemas basados en NIOS II. Al utilizar este entorno se emplea una capa de software HAL (Hardware Abstraction Layer) que oculta los detalles de la configuración del hardware, haciendo transparente al programador el desarrollo de aplicaciones.

Estas librerías son creadas a partir del archivo SOPC Information File (.sopcinfo) generado por SOPC BUILDER que contiene el diseño interno del hardware para la FPGA. Una vez generadas las librerías, NIOS II nos permite utilizar lenguaje C/C++ para desarrollar aplicaciones que se encarga de controlar el procesador NIOS II. Estas aplicaciones generan un archivo ejecutable .ELF que se almacena en la FPGA para ser ejecutado por el procesador NIOS II.

Muchos sistemas basados en procesadores NIOS II utilizan memoria flash externa para almacenar:

- Código de programa
- Datos del programa
- Datos de configuración de la FPGA
- Sistema de archivos

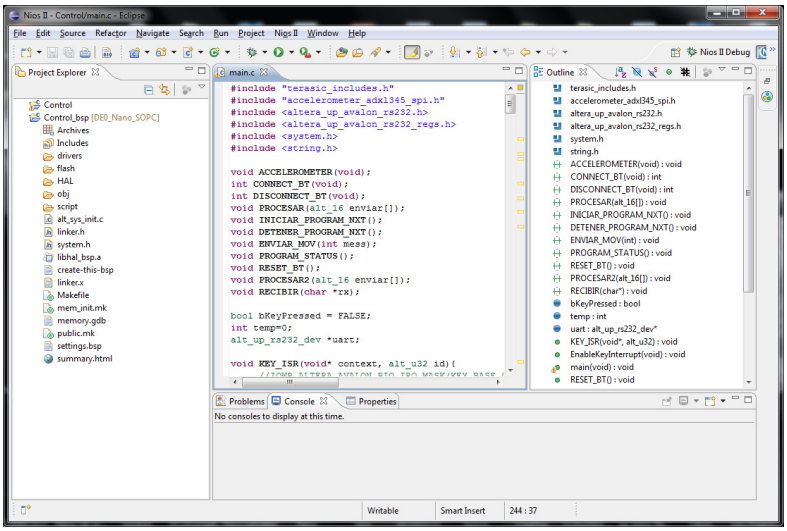


Figura 2.19 Interfaz de usurario de NIOS II SOFTWARE BUILD TOOLS

Nios II Software Build Tools for Eclipse proporciona una utilidad que permite programar los chips de memoria flash. El programador permite grabar y administrar programas en cualquier tarjeta, incluyendo las placas de desarrollo Altera.

2.2.4. NETBEANS IDE

NetBeans IDE es un entorno de desarrollo integrado (IDE), modular, de base estandar (normalizado), escrito en el lenguaje de programación Java.

El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general (framework) para compilar cualquier tipo de aplicación.

Figura 2.20 Presentación del entorno de desarrollo ECLIPSE

Características principales:

- Suele dar soporte a casi todas las novedades en el lenguaje Java. Cualquier preview del lenguaje es rápidamente soportada por Netbeans.
- Asistentes para la creación y configuración de distintos proyectos, incluida la elección de algunos frameworks.
- Buen editor de código, multilenguaje, con el habitual coloreado y sugerencias de código, acceso a clases pinchando en el código, control de versiones, localización de ubicación de la clase actual, comprobaciones sintácticas y semánticas, plantillas de código, coding tips, herramientas de refactorización,… y un largo etcétera. También hay tecnologías donde podemos usar el pulsar y arrastrar para incluir componentes en nuestro código.
- Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, y estructurando la visualización de manera ordenada, lo que ayuda en el trabajo diario. Una vez que nos metemos en una clase java, por poner un ejemplo, se nos mostrarán distintas

ventanas con el código, su localización en el proyecto, una lista de los métodos y propiedades (ordenadas alfabéticamente), también hay una vista que nos presenta las jerarquías que tiene nuestra clase y otras muchas opciones. Por supuesto personalizable según el gusto de cada usuario.

CAPÍTULO 3

3. DISEÑO E. IMPLEMENTACION

3.1. BLUETOOTH RN-42

Para lograr comunicarnos con el robot lego nxt utilizamos el módulo bluetooth RN-42 en modo maestro, el cual permite enviar órdenes desde la tarjeta FPGA y recibir datos del robot para procesarlas en nuestro ordenador utilizando el código escrito en el software NIOS II.

3.2. SENSOR ULTRASÓNICO

En este proyecto utilizamos dos sensores ultrasónicos, uno en la parte frontal y otro en la parte lateral derecha del robot.

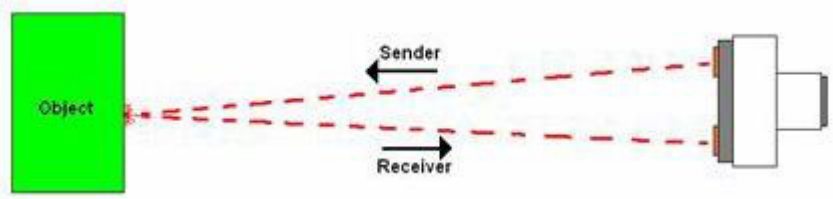


Figura 3.1 Funcionamiento del sensor ultrasónico.

Luego de establecer las conexiones requeridas para la comunicación JAVA- NXT.

El programa se encarga de enviar un pulso al sensor 1 (frontal), esperando su respuesta para calcular la distancia, haciendo lo mismo con el sensor 2 (lateral), luego de lo cual con estos datos le permite saber si se encuentra o no un obstáculo cercano al robot y decidir automáticamente cual siguiente paso dar.

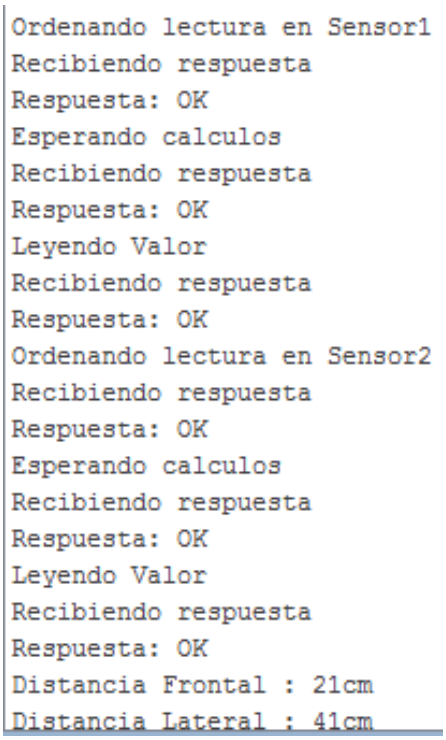


Figura 3.2 Flujo de información entre sensores y consola NIOS II

3.3. CONVERTIDOR USB-SERIAL FT232

Tuvimos la necesidad de utilizar este convertidor para poder comunicar la tarjeta FPGA con la interfaz creada en JAVA y así procesar los datos recibidos del robot lego mindstorm nxt.

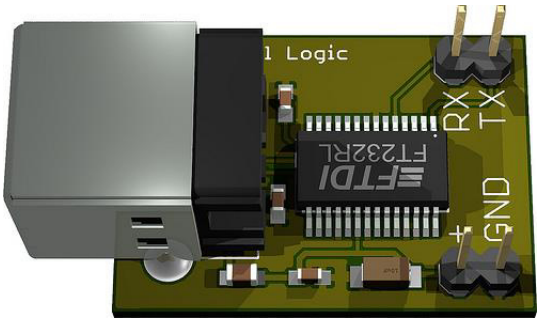


Figura 3.3 Modulo USB-UART (FT232R)

3.4. COMPUTADOR BÁSICO USANDO QSYS

Para controlar el ROBOT LEGO MINDSTORM NXT con la FPGA debemos crear dentro de ella una mini-computadora con la herramienta Qsys la cual nos permite añadir una serie de librerías de componentes requeridos para nuestro proyecto.

Los componentes básicos requeridos para un correcto funcionamiento de nuestra mini-computadora son:

System ID Peripheral: Es uno de los periféricos más importantes que debe contener el sistema, ya que permite a la herramienta de software de NIOS II saber si la aplicación desarrollada pertenece al hardware previamente configurado. Es esencial que solo exista un solo periférico de este tipo en cada sistema.

NIOS II Processor: Encargado de ser el cerebro para todo el sistema, ya que todo el software será interpretado por él y este a su vez enviara las señales necesarias y correspondientes a cada periférico para su correcto funcionamiento.

On-Chip Memory: Esta es una memoria que se implementará en la FPGA con la cual almacenaremos el software del sistema y también todos los datos necesarios en el programa.

JTAG UART: Este periférico nos permite establecer la comunicación, es esencial incluir este módulo ya que de no ser así no se podría realizar ningún cambio a la configuración del chip Cyclone IV.

SDRAM Controller: Este módulo nos ayuda a acceder y controlar la memoria SDRAM externa incluida en la tarjeta DE0-Nano, la cual tiene la capacidad de almacenar 32Mbytes de datos.

Interval Timer: Este módulo nos ayudara a medir intervalos de tiempo necesarios para nuestro programa.

PIO (Parallel I/O): Es de gran importancia ya que es mediante este módulo que se obtienen los datos del exterior para poder ser procesados. De igual manera se entregan datos al exterior para el control de otros dispositivos acoplados al sistema o proporcionar información útil al usuario.

RS232 UART: Es de gran importancia ya que es mediante este módulo que se realizara la comunicación entre nuestro sistema embebido u el bloque NXT y además la comunicación con nuestra aplicación JAVA.

Luego de haber colocado cada módulo y realizar las conexiones necesarias entre los distintos módulos, se muestra en la Figura:

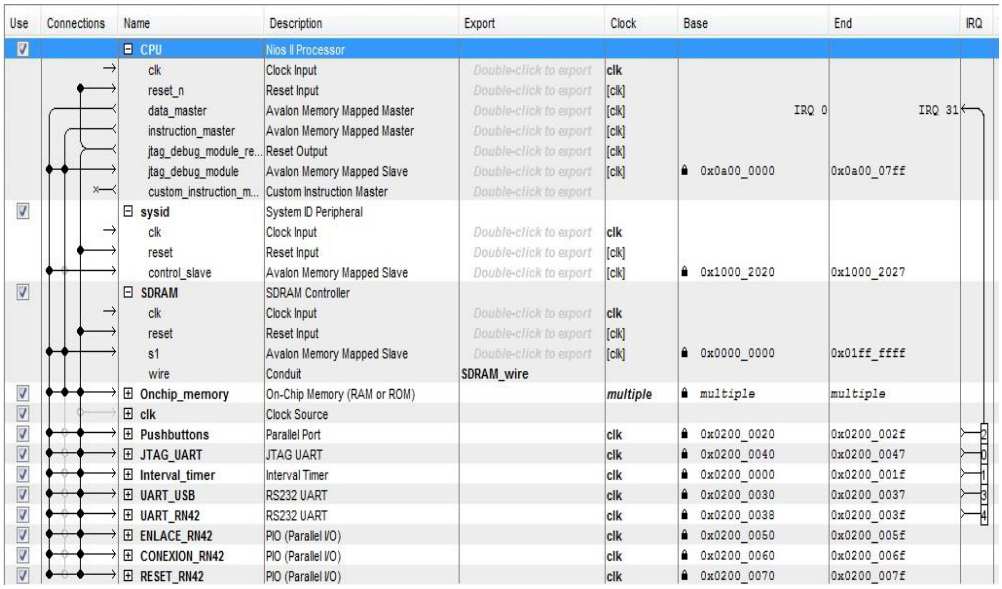


Figura 3.4 Diseño e interconexión de componentes para la computadora básica

3.5. ASIGNACIÓN DE PINES A LA COMPUTADORA BÁSICA

Luego de haber creado la mini-computadora en Qsys, procedemos a observar el bloque creado de nuestro sistema utilizando Quartus II.

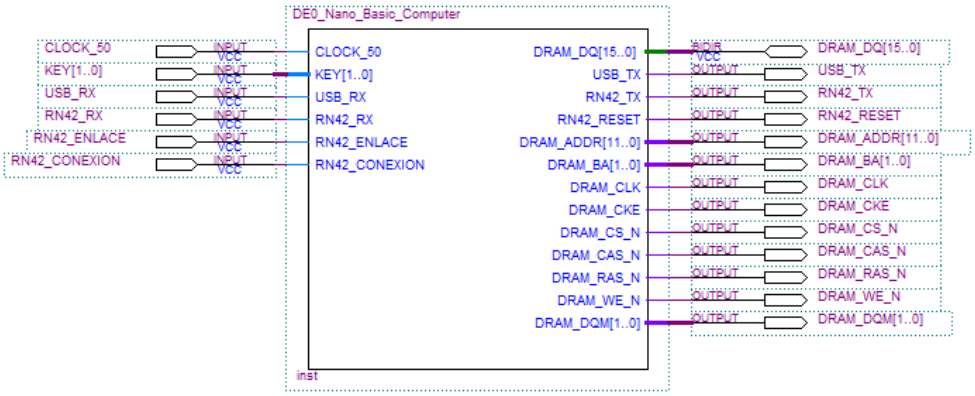


Figura 3.5 Diagrama esquematico de la computadora básica

Luego en el mismo programa Quartus II le damos a la opción de PIN PLANNER el cual nos permitirá asignar los pines para el correcto funcionamiento de cada uno de los módulos y periféricos que conforman la mini-computadora.

Primero asignamos los pines para usar el reloj a 50 Mhz y también el KEY0 de la De0-Nano que servirá como un reset. (Tabla 3.1)

Node Name	Location
CLOCK_50	PIN_R8
KEY 0	PIN_J15

Tabla 3.1 Configuración de pines para las señales globales

Continuamos con los pines del módulo controlador de la memoria SDRAM externa (Tabla 3.2)

SDRAM			
Node Name	Location	Node Name	Location
DRAM_ADDR[11]	PIN_N1	DRAM_DQ[15]	PIN_K1
DRAM_ADDR[10]	PIN_N2	DRAM_DQ[14]	PIN_N3
DRAM_ADDR[9]	PIN_P1	DRAM_DQ[13]	PIN_P3
DRAM_ADDR[8]	PIN_R1	DRAM_DQ[12]	PIN_R5
DRAM_ADDR[7]	PIN_T6	DRAM_DQ[11]	PIN_R3
DRAM_ADDR[6]	PIN_N8	DRAM_DQ[10]	PIN_T3
DRAM_ADDR[5]	PIN_T7	DRAM_DQ[9]	PIN_T2
DRAM_ADDR[4]	PIN_P8	DRAM_DQ[8]	PIN_T4
DRAM_ADDR[3]	PIN_M8	DRAM_DQ[7]	PIN_R7
DRAM_ADDR[2]	PIN_N6	DRAM_DQ[6]	PIN_J1
DRAM_ADDR[1]	PIN_N5	DRAM_DQ[5]	PIN_J2
DRAM_ADDR[0]	PIN_P2	DRAM_DQ[4]	PIN_K2
DRAM_BA[1]	PIN_M6	DRAM_DQ[3]	PIN_K5
DRAM_BA[0]	PIN_M7	DRAM_DQ[2]	PIN_L8
DRAM_CAS_N	PIN_L1	DRAM_DQ[1]	PIN_G1
DRAM_CKE	PIN_L7	DRAM_DQ[0]	PIN_G2
DRAM_CLK	PIN_R4	DRAM_DQM[1]	PIN_T5
DRAM_CS_N	PIN_P6	DRAM_DQM[0]	PIN_R6
DRAM_RAS_N	PIN_L2	DRAM_WE_N	PIN_C2

Tabla 3.2 Configuracion de pines para controlar el modulo SDRAM

Además también añadimos los pines utilizados al establecer la comunicación entre el robot LEGO NXT MINDSTORM y el módulo bluetooth RN-42. (Tabla 3.3).

RN-42	
Node Name	Location
RN42_RX	PIN_L15
RN42_TX	PIN_K16
RN42_CONEXION	PIN_P15
RN42_ENLACE	PIN_R14
RN42_RESET	PIN_R16

Tabla 3.3 Configuración de pines para el Módulo de Comunicación Bluetooth

Y por último agregamos los pines utilizados para la comunicación SERIAL-USB.

SERIAL-USB	
Node Name	Location
USB_RX	PIN_A4
USB_TX	PIN_B5

Tabla 3.4 Configuración de pines para el Módulo de Comunicación Serial

3.6. CÓDIGO DEL PROGRAMA PRINCIPAL

A continuación mostraremos los códigos de programación utilizados para poder lograr hacer funcionar nuestro proyecto.

Empezamos con las librerías necesarias para utilizar cada componente requerido.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "system.h"
#include "altera_up_avalon_rs232.h"
#include "altera_up_avalon_rs232_regs.h"
```

Figura 3.6 Librerias necesaria para cada componente

Ahora hablaremos brevemente de cada función creada para el funcionamiento del programa

```
void RN42_Enviar(char *data, int n_data);
void RN42_Recibir();
void RN42_Decodificar();
void USB_Enviar(char *data, int n_data);
void USB_Recibir();
void Leer_Sensor1();
void Leer_Sensor2();
void Iniciar_Tablero();
void Comando(int Fila, int Columna, int Estado);
```

Figura 3.7 Prototipos de funciones a usar en el programa

RN42_Enviar: Esta función nos permite enviar datos desde el módulo bluetooth al robot LEGO MINDSTORM NXT

```
void RN42_Enviar(char *data, int n_data){
    int index;
    for(index=0 ; index<n_data ; index++)
        alt_up_rs232_write_data(RN42, data[index]);
}
```

Figura 3.8 Contenido de la función para enviar datos mediante el modulo RN42

RN42_Recibir: Esta función se encarga de recibir los mensajes enviados por el robot LEGO MINDSTORM NXT

```
void RN42_Recibir(){
    while(IORD_ALT_UP_RS232_RAVAIL(UART_RN42_BASE)==0);
    // Capturamos respuesta
    printf("Recibiendo respuesta\n");
    index_rn42=0;
    while(IORD_ALT_UP_RS232_RAVAIL(UART_RN42_BASE)!=0){
        string_rn42[index_rn42] = (int)IORD_ALT_UP_RS232_DATA(UART_RN42_BASE);
        usleep(10000);
        index_rn42++;
    }
    string_rn42[index_rn42] = '\0';
}
```

Figura 3.9 Contenido de la función para recibir datos mediante el módulo RN42

RN42_Decodificar: Esta función se encarga de interpretar el mensaje enviado por el robot LEGO MINDSTORM NXT para que así nuestro módulo bluetooth sepa que hacer.

```
void RN42_Decodificar(){
    if(string_rn42[4]==-64)
        printf("Respuesta: ERROR\n");
    else if(string_rn42[4]==0)
        printf("Respuesta: OK\n");
    else
        printf("Respuesta: %s",string_rn42);
}
```

Figura 3.10 Contenido de la función para interpretar datos recibidos del modulo RN42

USB_Enviar: Esta función se encarga de enviar datos mediante el convertidor USB-Serial desde la FPGA a la interfaz en JAVA.

```
void USB_Enviar(char *data, int n_data){
    int index;
    for(index=0 ; index<n_data ; index++){
        alt_up_rs232_write_data(USB, data[index]);
        usleep(20000);
    }
}
```

Figura 3.11 Contenido de la función para enviar datos mediante el modulo FT232R

USB_Recibir: Esta función se encarga de recibir los mensajes enviados por la interfaz JAVA y transmitirlo a la FPGA

```
void USB_Recibir(){
    while(IORD_ALT_UP_RS232_RAVAIL(UART_USB_BASE)==0);
    // Capturamos respuesta
    printf("Recibiendo respuesta\n");
    index_usb=0;
    while(IORD_ALT_UP_RS232_RAVAIL(UART_USB_BASE)!=0){
        string_usb[index_usb] = (char)IORD_ALT_UP_RS232_DATA(UART_USB_BASE);
        usleep(20000);
        index_usb++;
    }
    string_usb[index_usb] = '\0';
    printf("Respuesta: %s",string_usb);
}
```

Figura 3.12 Contenido de la función para recibir datos mediante el modulo FT232R

Leer_Sensor1: Esta función es la encargada de recibir la información del sensor ultrasónico frontal del robot, utilizando algunas funciones anteriormente descritas, genera un pulso por el cual espera y procede a realizar un cálculo de la distancia a la cual se encuentra un obstáculo.

```
void Leer_Sensor1(){
    printf("Ordenando lectura en Sensor1\n");
    RN42_Enviar(ORDER_SENSOR1,9);
    RN42_Recibir();
    RN42_Decodificar();
    printf("Esperando calculos\n");
    RN42_Enviar(STATUS_SENSOR1,5);
    RN42_Recibir();
    RN42_Decodificar();
    printf("Leyendo Valor\n");
    RN42_Enviar(VALUE_SENSOR1,5);
    RN42_Recibir();
    RN42_Decodificar();
    Lectura_Sensor1 = (int)string_rn42[6];
    if(Lectura_Sensor1 < 0)
        Lectura_Sensor1 = 255;
}
```

Figura 3.13 Descripcion del proceso de lectura del sensor 1

Leer_Sensor2: Esta función realiza lo mismo que la anterior descrita pero lo hace con el segundo sensor (lateral) que posee nuestro robot.

```
void Leer_Sensor2(){
    printf("Ordenando lectura en Sensor2\n");
    RN42_Enviar(ORDER_SENSOR2,9);
    RN42_Recibir();
    RN42_Decodificar();
    printf("Esperando calculos\n");
    RN42_Enviar(STATUS_SENSOR2,5);
    RN42_Recibir();
    RN42_Decodificar();
    printf("Leyendo Valor\n");
    RN42_Enviar(VALUE_SENSOR2,5);
    RN42_Recibir();
    RN42_Decodificar();
    Lectura_Sensor2 = (int)string_rn42[6];
    if(Lectura_Sensor2 <0)
        Lectura_Sensor2 = 255;
}
```

Figura 3.14 Descripcion del proceso de lectura del sensor 2

Iniciar_Tablero: esta función se encarga de dibujar las dimensiones del tablero virtual en JAVA y de ir pintando cada cuadro de acuerdo como vaya indicando el robot.

```
void Iniciar_Tablero(){
    int i,j;
    for(i=0;i<DIM;i++){
        for(j=0;j<DIM;j++){
            Tablero[i][j]=-1;
        }
    }
    Tablero[DIM-1][DIM-1]=VACIO;
    Fila=DIM-1;
    Columna=DIM-1;
    Direccion=NORTE;
}
```

Figura 3.15 Descripcion del proceso de inicialización del tablero

Comando: Esta función al comunicar el robot con JAVA permite conocer tres valores necesarios para marcar en el tablero virtual los cuales son la fila, columna y el estado en que se encuentra ubicado nuestro robot.

```
void Comando(int Fila, int Columna, int Estado){
    CMD_JAVA[1] = Fila;
    CMD_JAVA[2] = Columna;
    CMD_JAVA[3] = Estado;
    USB_Enviar(CMD_JAVA,4);
    Tablero[Fila][Columna]=Estado;
}
```

Figura 3.16 Descripcion de la trama de envio a JAVA

Para poder comunicar la FPGA con el robot LEGO MINDSTORM NXT tuvimos que recurrir al lenguaje utilizado por el robot, empezando con una frecuencia de 115200 y luego la siguiente secuencia de funciones obtenidas del primer manual de la primera versión creada del robot, el cual lo puede encontrar en la sección de anexos.

```
char CMD_MODE [] = {'$','$','$'};
char CONECTAR_NXT [] = {'C',13,10};
char CONFIG_SENSOR1 [] = {5,0,0,5,0,11,0};
char ORDER_SENSOR1 [] = {7,0,0,15,0,2,1,2,66};
char STATUS_SENSOR1 [] = {3,0,0,14,0};
char VALUE_SENSOR1 [] = {3,0,0,16,0};
char CONFIG_SENSOR2 [] = {5,0,0,5,3,11,0};
char ORDER_SENSOR2 [] = {7,0,0,15,3,2,1,2,66};
char STATUS_SENSOR2 [] = {3,0,0,14,3};
char VALUE_SENSOR2 [] = {3,0,0,16,3};
char MOVE_NXT [] = {13,0,128,4,255,15,7,1,0,32,0,0,0,0,0};
char STOP_NXT [] = {13,0,128,4,255,0,0,0,0,0,0,0,0,0,0};
char RIGHT1_NXT [] = {13,0,128,4,2,15,7,1,0,32,0,0,0,0,0};
char RIGHT2_NXT [] = {13,0,128,4,1,-15,7,1,0,32,0,0,0,0,0};
char LEFT1_NXT [] = {13,0,128,4,1,15,7,1,0,32,0,0,0,0,0};
char LEFT2_NXT [] = {13,0,128,4,2,-15,7,1,0,32,0,0,0,0,0};
char CMD_JAVA [] = {'*',0,0,0};
int Tablero[DIM][DIM];
int Finalizar=0;
```

Figura 3.17 Codigos usados para poder enviar ordenes al bloque NXT mediante bluetooth

Explicadas todas las funciones del programa procedemos con el “main” en el cual lo primero que realiza es dar un reset al módulo bluetooth. Luego de esto se ingresa al modo comando enviando ($,$,$).

```
int main(void){

    // Archivos para comunicacion serial
    RN42 = alt_up_rs232_open_dev(UART_RN42_NAME);
    USB = alt_up_rs232_open_dev(UART_USB_NAME);

    do{
        // Damos Reset al modulo RN42
        printf("Reset RN42\n");
        IOWR(RESET_RN42_BASE,0,0);
        usleep(250000);
        IOWR(RESET_RN42_BASE,0,1);
        usleep(1000000);
        // Enviamos codigo para entrar en Modo-Comando y esperamos por alguna respuesta.
        printf("Ingresando al Modo Comando\n");
        usleep(100000);
        RN42_Enviar(CMD_MODE,3);
        RN42_Recibir();
        RN42_Decodificar();
        usleep(100000);
        if(strcmp(string_rn42,"CMD\r\n")!=0)
            printf("Estado: Error al ingresar al Modo Comando\n");
        else
            printf("Estado: Ingreso al Modo Comando Exitoso\n");
    // Se espera la secuencia CMD
    }while(strcmp(string_rn42,"CMD\r\n")!=0);
    usleep(1000000);
```

Figura 3.18 Proceso inicial de ingreso al modo comando en RN42

Luego de un ingreso exitoso al modo comando, procedemos a realiza la conexión entre el De0-NANO con el NXT.

```
// Ordenamos la Conexion entre DE0-NANO y NXT
printf("Iniciando Conexion DE0-NANO <<-->> NXT\n");
usleep(100000);
RN42_Enviar(CONECTAR_NXT,3);
RN42_Recibir();
RN42_Decodificar();
usleep(100000);
// Esperamos a que se establesca la conexion
while(IORD(CONEXION_RN42_BASE,0)==0);
printf("Conexion DE0-NANO <<-->> NXT establecida...\n");
usleep(1000000);
```

Figura 3.19 Proceso de conexión DE0-NANO con bloque NXT

Una vez establecida aquella conexión, procedemos a realizar la conexión entre DE0-NANO con la interfaz gráfica JAVA.

```
do{ // Ordenamos la Conexion entre DE0-NANO y JAVA
    printf("Iniciando Conexion DE0-NANO <<-->> JAVA\n");
    usleep(100000);
    //do{   // Ordenamos la Conexion entre DE0-NANO y JAVA
        USB_Enviar("*JAVA?",6);
    //}while(IORD_ALT_UP_RS232_RAVAIL(UART_USB_BASE)!=1);
    // Capturamos respuesta desde JAVA
    USB_Recibir();
    usleep(100000);
    if(strcmp(string_usb,"JAVA_OK\n")!=0)
        printf("Estado: Error al conectar con JAVA\n");
    else
        printf("Estado: Conexion con JAVA Exitosa\n");
}while(strcmp(string_usb,"JAVA_OK\n")!=0);
printf("Conexion DE0-NANO <<-->> JAVA establecida...\n");
usleep(1000000);
```

Figura 3.20 Proceso de conexión entre DE0-NANO y la aplicación JAVA

Iniciamos el NXT con una primera lectura de sus sensores.

```
// Enviamos comando para iniciar el programa NXT
printf("Iniciando Programa NXT\n");
USB_Enviar("*NXT_RUN",8);
// Inicializamos el Tablero para empezar recorrido
printf("Estableciendo Condiciones Iniciales\n");
Iniciar_Tablero();
// Configuracion de Sensor en Puerto1
printf("Configurando Sensor en Puerto1\n");
RN42_Enviar(CONFIG_SENSOR1,7);
RN42_Recibir();
RN42_Decodificar();
// Configuracion de Sensor en Puerto4
printf("Configurando Sensor en Puerto4\n");
RN42_Enviar(CONFIG_SENSOR2,7);
RN42_Recibir();
RN42_Decodificar();
usleep(1000000);

Leer_Sensor1();
Leer_Sensor2();
```

Figura 3.21 Envio de señales para proceso inicial

Luego de que todas las conexiones se hayan establecidos correctamente, preguntamos por los sensores, y sus cálculos de distancia, respecto con aquellos datos se decide si se avanza o se gira.

```
// Lazo infinito para lectura de datos NXT
Direccion = NORTE;
while(Finalizar!=1){
    usleep(20000);
    Comando(Fila,Columna,VACIO);
    Leer_Sensor1();
    Leer_Sensor2();
    printf("Distancia Frontal : %dcm\n", Lectura_Sensor1);
    printf("Distancia Lateral : %dcm\n", Lectura_Sensor2);
    // Preguntamos si se puede Girar a la Derecha
    if(Lectura_Sensor2>30){
        RN42_Enviar(RIGHT1_NXT,15);
        RN42_Enviar(RIGHT2_NXT,15);
        usleep(501200);
        RN42_Enviar(MOVE_NXT,15);
        usleep(1680000);
        RN42_Enviar(STOP_NXT,15);
        switch(Direccion){
            case NORTE: Direccion = ESTE; Columna++; break;
            case SUR: Direccion = OESTE; Columna--; break;
            case ESTE: Direccion = SUR; Fila++; break;
            case OESTE: Direccion = NORTE; Fila--; break;
        }
        Tablero[Fila][Columna] = 1;
    }
```

Figura 3.22 Proceso de adquisición y envio de datos hasta realizarel recorrido completo del entorno

```
// Preguntamos si se puede Avanzar
else if(Lectura_Sensor1>30){
        if(Direccion==SUR && Columna==DIM-1 && Fila+1==DIM-1){
            Finalizar=1;
            break;
        }
        if(Direccion==ESTE && Columna+1==DIM-1 && Fila==DIM-1){
            Finalizar=1;
            break;
        }
        RN42_Enviar(MOVE_NXT,15);
        usleep(1680000);
        RN42_Enviar(STOP_NXT,15);
        switch(Direccion){
            case NORTE: Fila--;
                        if(Columna > 0 && Columna < DIM-1)
                            Comando(Fila,Columna+1,OCUPADO);
                        break;
            case SUR:   Fila++;
                        if(Columna > 0 && Columna < DIM-1)
                            Comando(Fila,Columna-1,OCUPADO);
                        break;
            case ESTE:  Columna++;
                        if(Fila > 0 && Fila < DIM-1)
                            Comando(Fila+1,Columna,OCUPADO);
                        break;
            case OESTE: Columna--;
                        if(Fila > 0 && Fila < DIM-1)
                            Comando(Fila-1,Columna,OCUPADO);
                        break;
        }
}
```

Figura 3.23 Proceso en caso de poder avanzar

Si no se puede avanzar, se buscará girar hacia la izquierda primeramente, ya que el robot siempre bordeará el tablero.

```
    // Ultima Opcion Girar a la Izquierda
    else{
        RN42_Enviar(LEFT1_NXT,15);
        RN42_Enviar(LEFT2_NXT,15);
        usleep(480000);
        RN42_Enviar(STOP_NXT,15);
        switch(Direccion){
            case NORTE: Direccion = OESTE;
                        if(Fila > 0)
                            Comando(Fila-1,Columna,OCUPADO);
                        break;
            case SUR:   Direccion = ESTE;
                        if(Fila < DIM-1)
                            Comando(Fila+1,Columna,OCUPADO);
                        break;
            case ESTE:  Direccion = NORTE;
                        if(Columna < DIM-1)
                            Comando(Fila,Columna+1,OCUPADO);
                        break;
            case OESTE: Direccion = SUR;
                        if(Columna > 0)
                            Comando(Fila,Columna-1,OCUPADO);
                        break;
        }
    }
}
```

Figura 3.24 Proceso a seguir en caso de no poder avanzar y poder girar a la izquierda

Este proceso se repite durante todo el recorrido, pero si el siguiente cuadro es aquel de donde empezó a moverse el robot, este automáticamente se detendrá, indicando que ha llegado al final de su recorrido.

```
printf("Final de Recorrido \n");
usleep(1000000);
//USB_Enviar("*Z",2);

return 0;
```

Figura 3.25 Muestra del proceso final de mapeo del entorno

3.6.1. DIAGRAMA DE FLUJO DEL PROGRAMA

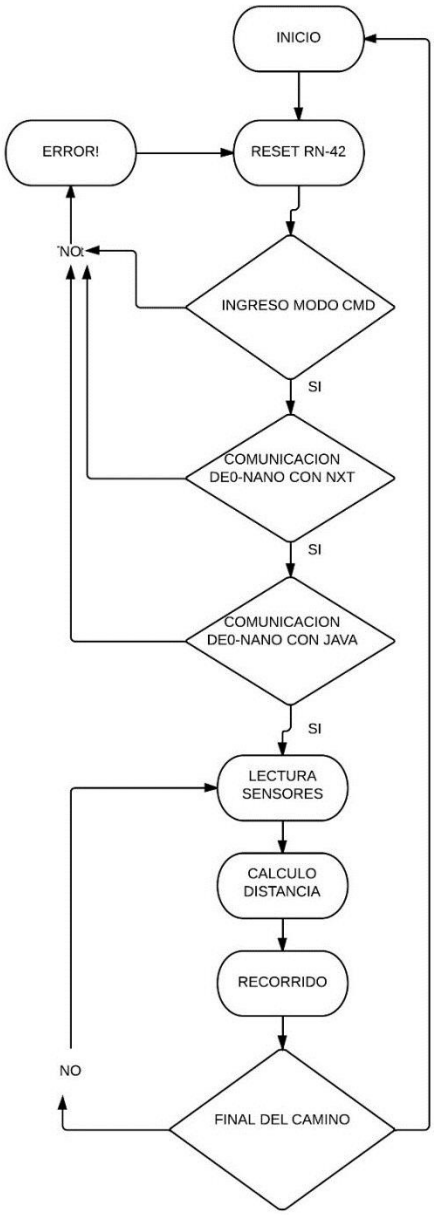


Figura 3.26 Diagrama de flujo del programa

3.7. MATERIALES Y COSTOS

Componente	Cantidad	Valor Unidad [$]	Valor total [$]
Tarjeta DE0 Nano	1	120	120
Módulo Bluetooth	1	25	25
Robot LEGO NXT MINDSTORM	1	540	540
Módulo FT232	1	23	23
TOTAL			894

Tabla 3.5 Componente y precios

CAPÍTULO 4

4. PRUEBAS Y RESULTADOS

Para este capítulo mostraremos las pruebas, mediciones y calibraciones realizadas para comparar los datos reales con los obtenidos en nuestra implementación.

4.1. ANÁLISIS DE DISTANCIAS

Los sensores ultrasónicos juegan un papel fundamental en nuestro proyecto debido a eso le realizamos unas pruebas de calibración para observar su comportamiento, pudiendo realizar las siguientes adquisiciones.

Figura 4.1 Toma de medidas para el sensor ultrasonico

PRUEBA SENSOR ULTRASONICO				
DISTANCIA	SENSOR1	ERROR	SENSOR2	ERROR
7	7	0,00%	7	0,00%
10	10	0,00%	10	0,00%
14	14	0,00%	14	0,00%
21	20	4,76%	20	4,76%
30	28	6,67%	27	10,00%
34	31	8,82%	32	5,88%

Tabla 4.1 Resumen de mediciones teóricas y experimentales

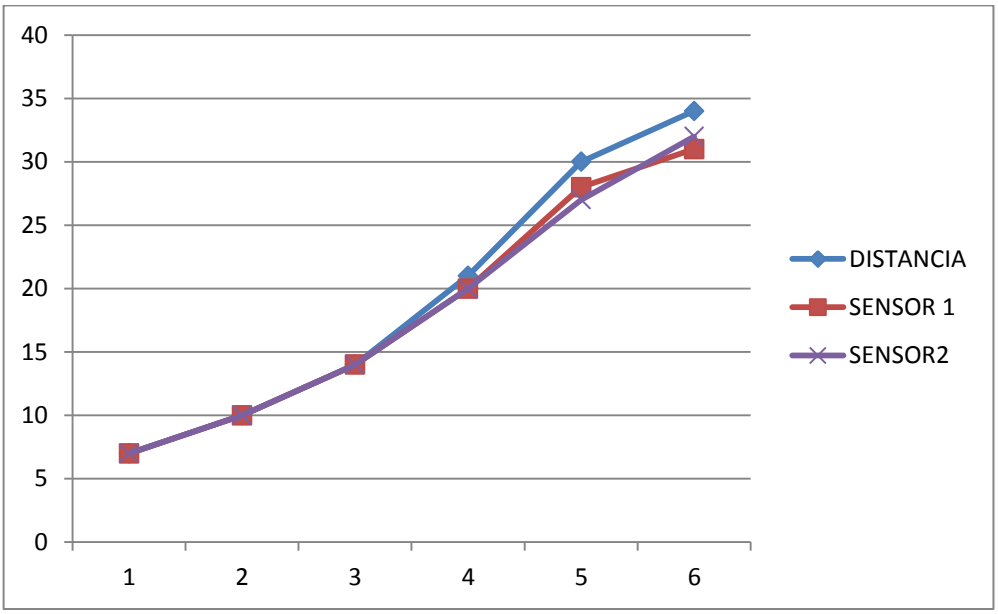


Figura 4.2 Estadistica de mediciones con el sensor

4.2. PRUEBAS

En entornos interiores, uno de los problemas habituales que afrontan los robots es mantenerse localizados. Este proyecto maneja dos frentes de percepción (frontal y lateral derecho) para lo cual necesitamos giros precisos de 90 grados que varían los cuatro estados de avance (norte, sur, este, oeste) las ruedas y la superficie por la que se desplaza puede modificar el correcto funcionamiento del recorrido.

Se Realizan 3 pruebas del funcionamiento completo del sistema. Donde pondremos a prueba la orientación, capacidad de percepción y evasión de obstáculos el color blanco indica una casilla vacía por la que puede circular un robot con características similares, el cuadro en color negro localiza un obstáculo en la posición exacta donde fue ubicado que impide el transito del robot. Creamos un espacio físico cuadrado de 150x150 cm con división cuadricular de 30x30cm y obstáculos de similares características. Estas medidas se muestran a escala con el programa en JAVA.

PRUEBA 1.

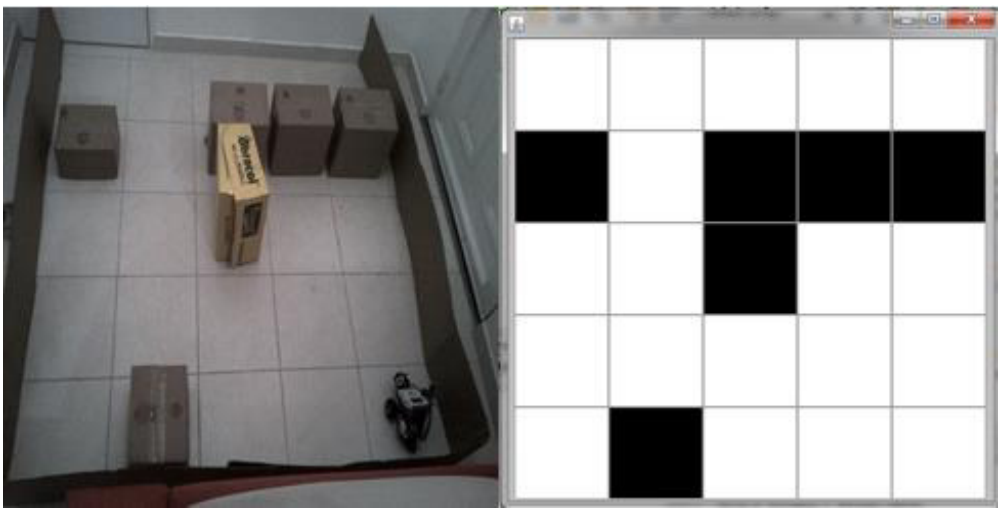


Figura 4.3 Resultado de recorrido de entorno 1

El primer entorno muestra un recorrido que implica un retorno por casillas marcadas poniendo a prueba la orientación del robot, la parte superior derecha muestra un desafío a la detección y cambio constante de orientación.

PRUEBA 2.

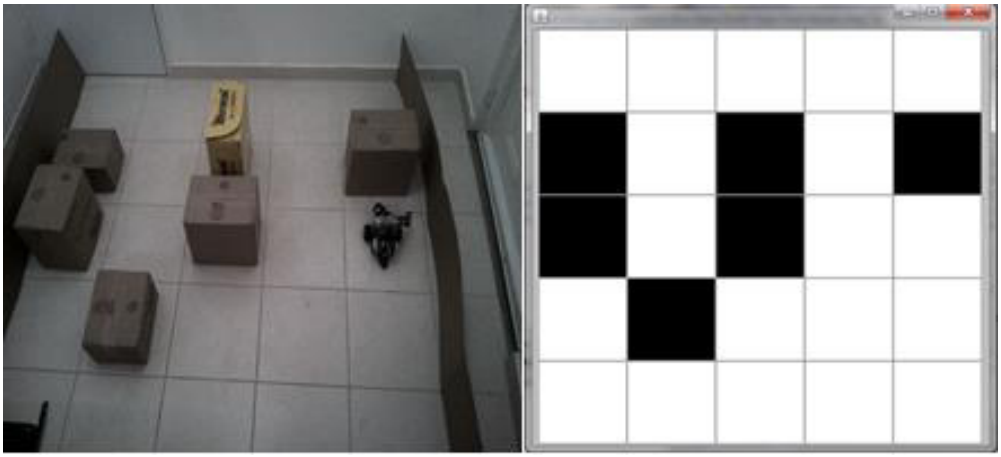


Figura 4.4 Resultado de recorrido de entorno 2

Variando la ubicación de los obstáculos con un ambiente más desafiante tratamos de que el robot pierda orientación. Hay que recordar que mientras más giros realice el robot se pueden cometer errores que dejarían sin validez el mapa.

PRUEBA 3.

Figura 4.5 Resultado de recorrido de entorno 3

Ahora ponemos a prueba las capacidades de la aplicación Java montando un ambiente que solo represente aproximadamente el 50% del área total. Como se muestra en la figura las casillas grises representan el área no explorada

4.3. TIEMPO DE RESPUESTA DEL ROBOT

Para esta prueba tomaremos el tiempo que ocupa nuestro programa en NIOS II con los distintos periféricos tales como: entrar al modo comando del módulo bluetooth RN42, el tiempo de conexión entre nuestra DE0-NANO y con el robot LEGO MINDSTORM NXT, el tiempo que le toma al programa configurar automáticamente ambos sensores ultrasónicos y por último el tiempo de lectura que ocupa los sensores desde el momento en que emite el pulso hasta que procesa el dato recibido.

Para las pruebas realizadas tomamos 6 medidas y tomamos los datos de cada una de ellas a 1, 2, 5, 10, 15, 20 metros de distancia en ambientes abiertos.

Para obtener los tiempos en nuestro programa, tuvimos que crear dos nuevas función, la cual no está descrita en el capítulo anterior, ya que solo lo utilizamos para tomar los tiempos en este capítulo de prueba. Estas nuevas funciones son **void Iniciar_Timer**(); y **void Detener_Timer**();

```
void Iniciar_Timer(){

    IOWR(INTERVAL_TIMER_BASE,2,0xFFFF);
    IOWR(INTERVAL_TIMER_BASE,3,0xFFFF);
    IOWR(INTERVAL_TIMER_BASE,1,4);
}
```

Figura 4.6 Funcion Iniciar_Timer

```
void Detener_Timer(){
    long Parte_baja, Parte_alta, Numero_Flancos;

    IOWR(INTERVAL_TIMER_BASE,1,8);      //STOP COUNTER
    IOWR(INTERVAL_TIMER_BASE,4,0);      //CAPTURA PARTE BAJA
    IOWR(INTERVAL_TIMER_BASE,5,0);      //CAPTURA PARTE ALTA
    Parte_baja = 0xFFFF & IORD(INTERVAL_TIMER_BASE,4);
    Parte_alta = 0xFFFF & IORD(INTERVAL_TIMER_BASE,5);
    Numero_Flancos = 0xFFFFFFFF & ((Parte_alta<<16) + Parte_baja);
    Contador_useg = ((0xFFFFFFFF - Numero_Flancos)/50);
}
```

Figura 4.7 Funcion Detener_Timer

Lo único que se debe realizar con estas funciones es colocarla entre las otras funciones de las cuales deseamos obtener el tiempo de ejecución. Y como resultado nos muestra esto dentro del programa NIOS II.

Los mensajes mostrados al ejecutar nuestro código en NIOS II sobre el ingreso al modo Comando del módulo bluetooth y el tiempo de 0.1475 segundos que toma para alcanzar con éxito este objetivo.

```
Ingresando al Modo Comando
Recibiendo respuesta
Respuesta: CMD
Estado: Ingreso al Modo Comando Exitoso
Tiempo de Ingreso a modo Comando: 147529 [uSeg]
```

Figura 4.8 Tiempo de Ingreso a Modo Comando

Ahora ejecutamos la sección de la conexión entre la DE0-NANO y nuestro LEGO MINDSTORM NXT y podemos observar los mensajes del NIOS y el tiempo de 2.2524 segundos que toma lograr la conexión.

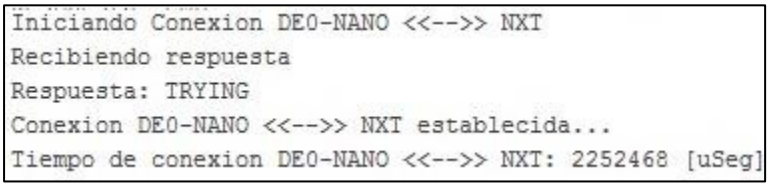


Figura 4.9 Tiempo de conexión DE0NANO con NXT

Una vez realizadas todas las comunicaciones procedemos a configurar los sensores ultrasónicos de nuestro robot, para esto también lo realiza de manera autónoma nuestro programa en NIOS II. Ahora nos muestra el tiempo que le toma configurar ambos sensores, 0.2244 segundos para el sensor 1 (frontal), y 0.2015 para el sensor 2 (lateral derecho).

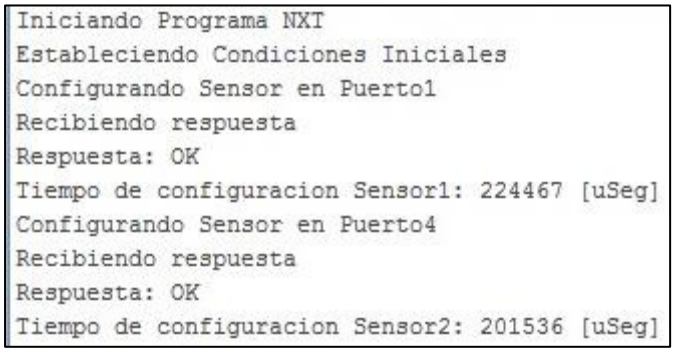


Figura 4.10 Tiempo configuración sensor 1 y sensor 2

Y por último tomamos el tiempo que ocupa los sensores desde que emitieron un pulso hasta que lo recibieron. En los mensajes del NIOS II nos muestra que el sensor 1 demora 1.1086 segundos y con el sensor 2 demora 1.0259 segundos.

```
Recibiendo respuesta
Recibiendo respuesta
Tiempo de lectura Sensor1: 1108654 [uSeg]
Recibiendo respuesta
Recibiendo respuesta
Recibiendo respuesta
Tiempo de lectura Sensor2: 1025900 [uSeg]
Recibiendo respuesta
```

Figura 4.11 Tiempo de lectura sensor 1 y sensor 2

Una vez obtenido los tiempos de prueba con las 6 distancias antes descritas tomadas como referencias, mostraremos las siguientes tablas.

Como observamos el tiempo no varía de acuerdo a la distancia.

TIEMPO CONEXION MODO COMANDO	
DISTANCIA(m)	**TIEMPO (us)**
1	147560
2	147594
5	147660
10	147529
15	147588
20	147532

Tabla 4.2 Tiempo de Conexión a Modo Comando

El tiempo si varía de acuerdo a la distancia debido a que el modulo bluetooth debe emitir más, para lograr localizar el receptor.

TIEMPO CONEXION DE0NANO CON NXT	
DISTANCIA (m)	**TIEMPO (us)**
1	2492000
2	2545149
5	2642474
10	4243966
15	4759155
20	117782373

Tabla 4.3 Tiempo de conexión De0Nano con NXT

Una vez lograda la comunicación bluetooth, el tiempo que le toma al programa NIOS II configurar los sensores, no varía significativamente.

TIEMPO CONFIGURAR SENSORES		
DISTANCIA (m)	**SENSOR1 (us)**	**SENSOR2 (us)**
1	201391	182079
2	220774	182370
5	236827	187662
10	211435	273332
15	216828	246504
20	193911	308625

Tabla 4.4 Tiempo para configurar sensores

El tiempo de lectura de los sensores tampoco varia significativamente, es decir que NIOS II lo realiza de manera autónoma todo el proceso.

TIEMPO LECTURA DE SENSORES		
DISTANCIA (m)	**SENSOR1 (us)**	**SENSOR2 (us)**
1	1093559	1070320
2	1112937	1105584
5	1089846	1073708
10	1238382	1221289
15	1495530	1268473
20	1188856	1863055

Tabla 4.5 Tiempo de lectura de sensores

CONCLUSIONES

- Se ha conseguido realizar un robot completamente autónomo con las características necesarias para realizar un mapeo en el cual se identifique claramente los obstáculos y los caminos libres para recorrer.

- La estimación de la posición está basada en la extracción de la información percibida por los sensores del robot con respecto al entorno en tiempo real, fusionar los datos y expresarlos en un sistema de referencia común para ambos frentes.

- Se ha desarrollado un método para extraer características en el espacio sensorial y definir regiones de interés para la toma de decisión del siguiente movimiento.

- El tiempo utilizado para realizar todo el proceso, desde el ingreso al modo comando del módulo bluetooth, hasta la lectura de ambos sensores para predecir el siguiente paso, no varía de manera significativa, teniendo retardos de apenas micro segundos, imperceptibles al momento de la práctica.

RECOMENDACIONES

1. Se recomienda trabajar sobre la computadora básica que Altera proporciona como ejemplo, pero adaptándola para nuestros requerimientos, con el afán de agilitar la creación de la misma.
2. Se debe tomar en cuenta la frecuencia de transmisión al momento de la comunicación de los dispositivos, ya que estos trabajan a distintos baudios, por ejemplo para conectar el dispositivo bluetooth RN42 con el LEGO

MINDSTORM NXT se trabaja con 115200 baudios, pero al conectar el USB UART este trabaja a 9600 baudios.

3. Para futuros proyectos se recomienda el uso de otro sensor distinto al ultrasónico para mayor precisión en las mediciones, ya que este sabe fallar a grandes distancias.

BIBLIOGRAFIA

[1] Terasic, DE0-Nano Development and Education Board, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=593.

[2] Altera, Cyclone4 Handbook, http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf.

[3] Altera, Nios II Processor Reference Handbook, http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.

[4] Altera, Nios II Software Developer's Handbook, http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf.

[5] Altera, Documentation: Nios II Processor, http://www.altera.com/literature/lit-nio2.jsp.

[6] Altera, Nios II Hardware Development, http://www.altera.com/support/examples/nios2/exm-hardware-tutorial.html.

[7] Cursomicros, UART y la Interface RS-232, http://www.cursomicros.com/avr/usart/estandar-rs232.html, fecha de consulta julio 2012

[8] Altera, UART Core - Quartus II 9.1 Handbook, Volume 5, http://www.altera.com.cn/literature/hb/nios2/n2cpu_nii51010.pdf.

[9] Roving Networks, Bluetooth Module RN-42, http://www.rovingnetworks.com/products/RN42.

[10] Microelectronicos, Trabajando con módulos Bluetooth RN-41 y RN-42, http://www.microelectronicos.net/?p=1075.

[11] Lego, Lego Mindstorms NXT Hardware Developer Kit, http://mindstorms.lego.com/en-us/support/files/default.aspx.

[12] Lego, Lego Mindstorms NXT Bluetooth Developer Kit, http://mindstorms.lego.com/en-us/support/files/default.aspx.

[13] Lego, Lego Mindstorms NXT User Guide, http://mindstorms.lego.com/en-us/support/files/default.aspx.

[14] Altera, Introduction to the Quartus II Software, http://www.altera.com/literature/manual/archives/intro_to_quartus2.pdf.

[15] Altera, Quartus II Handbook v12.1.0 Complete Three-Volume Set, http://www.altera.com/literature/lit-qts.jsp

[16] Altera, Introduction to SOPC Builder, http://www.altera.com/literature/hb/qts/qts_qii54001.pdf.

[17] Altera, SOPC Builder User Guide, http://www.altera.com/literature/ug/ug_sopc_builder.pdf.T

[18] Altera, SOPC Builder Support, http://www.altera.com/support/software/system/sopc/sof-sopc_builder.html.

[19] Altera, Nios II Software Build Tools for Eclipse Support, http://www.altera.com/support/ip/processors/nios2/ide/ips-nios2-ide.html.

[20] Altera, Nios II Software Developer's Handbook, http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf.

[21] Altera, Software Development Tools for the Nios II Processor, http://www.altera.com/devices/processor/nios2/tools/ide/ni2-ide.html.

[22] Rosenberg Neil, NXT Programming For Beginners, http://www.rocwnc.org/Beginning_NXT_Programming_Workshop.pdf.

[23] Floyd James, LEGO MINDSTORMS NXT-G Programming Guide Second Edition

[24] Griffin Terry, The Art of LEGO MINDSTORMS NXT-G Programming, No Starch Press 1st Ed, 2010

[25] Lego, Appendix 1-LEGO MINDSTORMS NXT Communication Protocol, http://mindstorms.lego.com/en-us/support/files/default.aspx.

[26] Lego, Appendix 2–LEGO MINDSTORMS NXT Direct commands, http://mindstorms.lego.com/en-us/support/files/default.aspx.

[27] Poliakoff Pierre, Communicating with LEGO NXT via Bluetooth in C#, http://www.codeproject.com/Articles/18857/Communicating-with-LEGO-NXT-via-Bluetooth-in-C.

[28] Roving Networks, Bluetooth RN-42 Command Reference, http://www.rovingnetworks.com/resources/download/47/Advanced_User_Manual.

[29] Altera, Nios II Flash Programmer User Guide, http://www.altera.com/literature/ug/ug_nios2_flash_programmer.pdf.

[30] Altera, Serial Configuration (EPCS) Devices, http://www.altera.com/literature/hb/cfg/cyc_c51014.pdf

ANEXOS

Version: 1.00

LEGO® MINDSTORMS® NXT

Direct Commands

TABLE OF CONTENTS

INTRODUCTION

This document describes a sub-protocol of the main LEGO® MINDSTORMS® NXT Communication Protocol specifically designed for direct commands which make it possible to control the NXT brick from outside devices. These outside devices may be other NXT bricks, a PC, or any other Bluetooth® capable device which used the serial port profile.

The main intent behind including this protocol is to provide a simple interface for these outside devices to utilize basic brick functionality (i.e. motor control, sensor readings, and power management) without the need to write or run specialized remote control programs on the brick. The protocol also includes an interface to send arbitrary messages to a target brick, which may be picked up and used in a user-defined NXT program.

OVERALL ARCHITECTURE

It will be possible to control the brick either through the USB communication channel or through the Bluetooth® communication channel, which both uses the LEGO® MINDSTORMS® NXT Communication protocol.

For further details and explanation on the USB and Bluetooth® communication channels, please refer to LEGO® MINDSTORMS® NXT Communication protocol document.

The figure below shows the general telegram architecture:

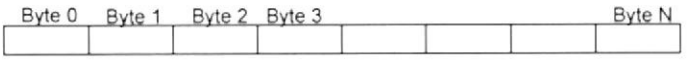


Figure 1: General protocol architecture

Byte 0: Telegram type, as dictated by main LEGO® MINDSTORMS® NXT Communication protocol specification. Note that for the purposes of this document, only "direct command telegrams" and "reply telegrams" are important – "system commands" are specified and handled at the main LEGO® MINDSTORMS® NXT Communication protocol document.

- 0x00: Direct command telegram, response required
- 0x01: System command telegram, response required
- 0x02: Reply telegram
- 0x80: Direct command telegram, no response
- 0x81: System command telegram, no response

Byte 1 – N: the command itself or a reply, depending on telegram type

MAXIMUM COMMAND LENGTH

Currently, total direct command telegram size is limited to 64 bytes, including the telegram type byte as listed above. As specified in the LEGO® MINDSTORMS® NXT Communication protocol document, Bluetooth® packets have an additional two bytes for size tacked onto the front, but these are not included in this limit.

BLUETOOTH® MESSAGES

As explained above all Bluetooth® messages needs to have two bytes in front of the messages itself which indicates how many bytes the message includes. The length of the packages is counted without the two length bytes.

The figure below shows a Bluetooth® message:

Length, LSB	Length, MSB	Command Type	Command	Byte 5	Byte 6	Etc.

Figure 2: Bluetooth® protocol packages

OPTIONAL RESPONSES

The LEGO® MINDSTORMS® NXT Communication protocol specification states that any incoming protocol telegram may be marked with the 0x80 mask on its telegram type byte to indicate that no response is expected. Direct commands is a primary use case for this functionality, as requiring a response on all telegrams could lead up to approximately 60 ms latency. Of course, this concept doesn't hold for all commands – for example, attempting to "GetInputValues" without requiring a response would just be wasting time.

LIST OF COMMANDS

Each of the valid commands is described in detail in the following list.

Format Notes:

- All response packages include a status byte, where 0x00 means "success" and any non-zero value indicates a specific error condition.
- All single byte values are unsigned, unless specifically stated. Internal data type is listed for all multi-byte values and all are assumed to be little-endian as in the LEGO® MINDSTORMS® NXT Communication Protocol.
- If a legal range is not specified here explicitly, it is generally documented in the relevant module documents and/or code.
- Variable length packet fields are specified as in this example: "Byte 4 – N: Message data", where 'N' is the variable size of a given field plus command overhead. 'N' may not exceed the Maximum Command Length mentioned above, minus 1.

STARTPROGRAM

Byte 0: 0x00 or 0x80
Byte 1: 0x00
Byte 2 - 21: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x00
Byte 2: Status Byte

STOPPROGRAM

Byte 0: 0x00 or 0x80
Byte 1: 0x01

Return package:
Byte 0: 0x02
Byte 1: 0x01
Byte 2: Status Byte

PLAYSOUNDFILE

Byte 0: 0x00 or 0x80
Byte 1: 0x02
Byte 2: Loop? (Boolean; TRUE: Loop sound file indefinitely, FALSE: Play file once only)
Byte 3 - 22: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x02
Byte 2: Status Byte

PLAYTONE

Byte 0: 0x00 or 0x80
Byte 1: 0x03
Byte 2 - 3: Frequency for the tone, Hz (UWORD; Range: 200 – 14000 Hz)
Byte 4 - 5: Duration of the tone, ms (UWORD; Range: ???)

Return package:
Byte 0: 0x02
Byte 1: 0x03
Byte 2: Status Byte

SETOUTPUTSTATE

Byte 0: 0x00 or 0x80
Byte 1: 0x04
Byte 2: Output port (Range: 0 – 2; 0xFF is special value meaning 'all' for simple control purposes)
Byte 3: Power set point (Range: -100 – 100)
Byte 4: Mode byte (Bit-field)
Byte 5: Regulation mode (UBYTE; enumerated)
Byte 6: Turn Ratio (SBYTE; -100 – 100)
Byte 7: RunState (UBYTE; enumerated)
Byte 8 – 12: TachoLimit (ULONG; 0: run forever)

Return package:
Byte 0: 0x02
Byte 1: 0x04
Byte 2: Status Byte

Valid enumeration for "Mode":

MOTORON	0x01	Turn on the specified motor
BRAKE	0x02	Use run/brake instead of run/float in PWM
REGULATED	0x04	Turns on the regulation

Valid enumeration for "Regulation Mode":

REGULATION_MODE_IDLE	0x00	No regulation will be enabled
REGULATION_MODE_MOTOR_SPEED	0x01	Power control will be enabled on specified output
REGULATION_MODE_MOTOR_SYNC	0x02	Synchronization will be enabled (Needs enabled on two output)

Valid enumeration for "RunState":

MOTOR_RUN_STATE_IDLE	0x00	Output will be idle
MOTOR_RUN_STATE_RAMPUP	0x10	Output will ramp-up
MOTOR_RUN_STATE_RUNNING	0x20	Output will be running
MOTOR_RUN_STATE_RAMPDOWN	0x40	Output will ramp-down

SETINPUTMODE

Byte 0: 0x00 or 0x80
Byte 1: 0x05
Byte 2: Input port (Range: 0 – 3)
Byte 3: Sensor type (enumerated)
Byte 4: Sensor mode (enumerated)

Return package:
Byte 0: 0x02
Byte 1: 0x05
Byte 2: Status Byte

Valid enumeration for "Sensor Type":

NO_SENSOR	0x00
SWITCH	0x01
TEMPERATURE	0x02
REFLECTION	0x03
ANGLE	0x04
LIGHT_ACTIVE	0x05
LIGHT_INACTIVE	0x06
SOUND_DB	0x07
SOUND_DBA	0x08
CUSTOM	0x09
LOWSPEED	0x0A
LOWSPEED_9V	0x0B
NO_OF_SENSOR_TYPES	0x0C

Valid enumeration for "Sensor Mode":

RAWMODE	0x00
BOOLEANMODE	0x20
TRANSITIONCNTMODE	0x40
PERIODCOUNTERMODE	0x60
PCTFULLSCALEMODE	0x80
CELSIUSMODE	0xA0
FAHRENHEITMODE	0xC0
ANGLESTEPSMODE	0xE0
SLOPEMASK	0x1F
MODEMASK	0xE0

GETOUTPUTSTATE

Byte 0: 0x00 or 0x80
Byte 1: 0x06
Byte 2: Output port (Range: 0 – 2)

Return package:
Byte 0: 0x02
Byte 1: 0x06
Byte 2: Status Byte
Byte 3: Output port (Range: 0 – 2)
Byte 4: Power set point (-100 - 100)
Byte 5: Mode (bit-field)
Byte 6: Regulation mode (UBYTE; enumerated)
Byte 7: Turn Ratio (SBYTE; -100 – 100)
Byte 8: RunState (UBYTE; enumerated)
Byte 9 – 12: TachoLimit (ULONG; Current limit on a movement in progress, if any)
Byte 13 – 16: TachoCount (SLONG; Internal count. Number of counts since last reset of the motor counter)
Byte 17 – 20: BlockTachoCount (SLONG; Current position relative to last programmed movement)
Byte 21 – 24: RotationCount (SLONG; Current position relative to last reset of the rotation sensor for this motor)

Look at the entry for SetOutputState for details about the enumerations.

GETINPUTVALUES

Byte 0: 0x00 or 0x80
Byte 1: 0x07
Byte 2: Input port (Range: 0 – 3)

Return package:
Byte 0: 0x02
Byte 1: 0x07
Byte 2: Status Byte
Byte 3: Input port (Range: 0 – 3)
Byte 4: Valid? (Boolean; TRUE if new data value should be seen as valid data)
Byte 5: Calibrated? (Boolean; TRUE if calibration file found and used for "Calibrated Value" field below)
Byte 6: Sensor type (enumerated)
Byte 7: Sensor mode (enumerated)
Byte 8 – 9: Raw A/D value (UWORD; device dependent)
Byte 10 – 11: Normalized A/D value (UWORD; type dependent; Range: 0 - 1023)
Byte 12 – 13: Scaled value (SWORD; mode dependent)
Byte 14 – 15: Calibrated value (SWORD; Value scaled according to calibration. CURRENTLY UNUSED.)

Look at the entry SetInputMode for details about the enumerations.

RESETINPUTSCALEDVALUE

Byte 0: 0x00 or 0x80
Byte 1: 0x08
Byte 2: Input port (Range: 0 – 3)

Return package:
Byte 0: 0x02
Byte 1: 0x08
Byte 2: Status Byte

MESSAGEWRITE

Byte 0: 0x00 or 0x80
Byte 1: 0x09
Byte 2: Inbox number (0 – 9)
Byte 3: Message size
Byte 4 - N: Message data, where N = Message size + 3

Message data is treated as a string; it must include null termination to be accepted. Accordingly, message size must include the null termination byte. Message size must be capped at 59 for all message packets to be legal on USB!

Return package:
Byte 0: 0x02
Byte 1: 0x09
Byte 2: Status Byte

RESETMOTORPOSITION

Byte 0: 0x00 or 0x80
Byte 1: 0x0A
Byte 2: Output port (Range: 0 – 2)
Byte 3: Relative? (Boolean; TRUE: position relative to last movement, FALSE: absolute position)

Return package:
Byte 0: 0x02
Byte 1: 0x0A
Byte 2: Status Byte

GETBATTERYLEVEL

Byte 0: 0x00 or 0x80
Byte 1: 0x0B

Return package:
Byte 0: 0x02
Byte 1: 0x0B
Byte 2: Status Byte
Byte 3-4: Voltage in millivolts (UWORD)

STOPSOUNDPLAYBACK

Byte 0: 0x00 or 0x80
Byte 1: 0x0C

Return package:
Byte 0: 0x02
Byte 1: 0x0C
Byte 2: Status Byte

KEEPALIVE

Byte 0: 0x00 or 0x80
Byte 1: 0x0D

Return package:
Byte 0: 0x02
Byte 1: 0x0D
Byte 2: Status Byte
Byte 3 – 6: Current sleep time limit, milliseconds (ULONG)

LSGETSTATUS

Byte 0: 0x00 or 0x80
Byte 1: 0x0E
Byte 2: Port (0 – 3)

Return package:
Byte 0: 0x02
Byte 1: 0x0E
Byte 2: Status Byte
Byte 3: Bytes Ready (count of available bytes to read)

LSWRITE

Byte 0: 0x00 or 0x80
Byte 1: 0x0F
Byte 2: Port (0 – 3)
Byte 3: Tx Data Length (bytes)
Byte 4: Rx Data Length (bytes)
Byte 5 – N: Tx Data, where N = Tx Data Length + 4

For LS communication on the NXT, data lengths are limited to 16 bytes per command. Rx Data Length MUST be specified in the write command since reading from the device is done on a master-slave basis.

Return package:
Byte 0: 0x02
Byte 1: 0x0F
Byte 2: Status Byte

LSREAD

Byte 0: 0x00 or 0x80
Byte 1: 0x10
Byte 2: Port (0 – 3)

Return package:
Byte 0: 0x02
Byte 1: 0x10
Byte 2: Status Byte
Byte 3: Bytes Read
Byte 3 - 19: Rx Data (padded)

For LS communication on the NXT, data lengths are limited to 16 bytes per command. Furthermore, this protocol does not support variable-length return packages, so the response will always contain 16 data bytes, with invalid data bytes padded with zeroes.

GETCURRENTPROGRAMNAME

Byte 0: 0x00 or 0x80
Byte 1: 0x11

Return package:
Byte 0: 0x02
Byte 1: 0x11
Byte 2: Status Byte
Byte 3 - 22: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

If no program is currently running, an error will be returned and File name field will be all zeroes.

MESSAGEREAD

Byte 0: 0x00 or 0x80
Byte 1: 0x13
Byte 2: Remote Inbox number (0 – 9)
Byte 3: Local Inbox number (0 – 9)
Byte 4: Remove? (Boolean; TRUE (non-zero) value clears message from Remote Inbox)

Return package:
Byte 0: 0x02
Byte 1: 0x13
Byte 2: Status Byte
Byte 3: Local Inbox number (0 – 9)
Byte 4: Message size
Byte 5 - 63: Message data (padded)

Message data is treated as a string; it must include null termination. Accordingly, message size includes the null termination byte. Furthermore, return packages have a fixed size, so the message data field will be padded with null bytes.

Note that the remote Inbox number may specify a value of 0-19, while all other mailbox numbers should remain below 9. This is due to the master-slave relationship between connected NXT bricks. Slave devices may not initiate communication transactions with their masters, so they store outgoing messages in the upper 10 mailboxes (indices 10-19). Use the MessageRead command from the master device to retrieve these messages.

When reading remote messages from slave devices, send the following commands:
0x05, 0x00, 0x00, 0x13, 0x0A, 0x00, 0x01 => Read Mailbox 0 from slave and clear message on slave
0x05, 0x00, 0x00, 0x13, 0x0B, 0x01, 0x01 => Read Mailbox 1 from slave and clear message on slave

ERROR MESSAGE BACK TO THE HOST:

The information below describes how we categorize error messages; it does not describe a separate or special error message. This information will be included within the return packages.

Return package:
Byte 0: 0x02
Byte 1: Command that is receiving a reply
Byte 2, Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.

The following error messages may be received from the NXT unit:

• Pending communication transaction in progress	0x20
• Specified mailbox queue is empty	0x40
• Request failed (i.e. specified file not found)	0xBD
• Unknown command opcode	0xBE
• Insane packet	0xBF
• Data contains out-of-range values	0xC0
• Communication bus error	0xDD
• No free memory in communication buffer	0xDE
• Specified channel/connection is not valid	0xDF
• Specified channel/connection not configured or busy	0xE0
• No active program	0xEC
• Illegal size specified	0xED
• Illegal mailbox queue ID specified	0xEE
• Attempted to access invalid field of a structure	0xEF
• Bad input or output specified	0xF0
• Insufficient memory available	0xFB
• Bad arguments	0xFF

```
/*=========================================================================
  Program:   Insight Segmentation & Registration Toolkit
  Module:    $RCSfile: Style.tex,v $
  Language:  C++
  Date:      $Date: 2002/07/29 15:51:48 $
  Version:   $Revision: 1.8 $
Copyright (c) 2003 Insight Software Consortium
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
 * Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.
 * The name of the Insight Consortium, nor the names of any consortium members,
   nor of any contributors, may be used to endorse or promote products derived
   from this software without specific prior written permission.
 * Modified source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS ``AS IS''
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
=========================================================================*/

/** \class NXT_USB
 * \brief Controls a LEGO Mindstorms NXT robot via a USB connection
 *
 * NXT_USB controls a LEGO Mindstorms NXT robot over a USB connection.  Namely
 * one can read from the sensors, move the motors and read rotational
information
 * from the motors.
 */

#include "NXT_USB.h"
#include <cstring>
#include <iostream>

// in and out ports
const int IN_1 = 0;
const int IN_2 = 1;
const int IN_3 = 2;
const int IN_4 = 3;
const int OUT_A = 0;
const int OUT_B = 1;
const int OUT_C = 2;

// response codes for direct commands
static const char RESPONSE = 0x00;
static const char NO_RESPONSE = 0x80;

// some command enumerations
static const char PLAYTONE = 0x03;
static const char SETOUTPUTSTATE = 0x04;
static const char SETINPUTMODE = 0x05;
static const char GETOUTPUTSTATE = 0x06;
static const char GETINPUTVALUES = 0x07;
static const char RESETMOTORPOSITION = 0x0A;
static const char LSGETSTATUS = 0x0E;
static const char LSWRITE = 0x0F;
static const char LSREAD = 0x10;
```

```
// some enumerations for "Sensor Type"
static const char SWITCH = 0x01; // touch sensor
static const char LIGHT_ACTIVE = 0x05;
static const char LIGHT_INACTIVE = 0x06;
static const char SOUND_DB = 0x07;
static const char SOUND_DBA = 0x08;
static const char LOWSPEED_9V = 0x0B;

// some enumerations for "Sensor Mode"
static const char RAWMODE = 0x00;
static const char BOOLEANMODE = 0x20;
static const char PCTFULLSCALEMODE = 0x80;

// some enumerations for "Mode"
static const char FLOAT = 0x00; // not an official enumation
static const char MOTORON = 0x01;
static const char BRAKE = 0x02;
static const char REGULATED = 0x04;

// some enumerations for "Regulation Mode"
static const char REGULATION_MODE_IDLE = 0x00;
static const char REGULATION_MODE_MOTOR_SPEED = 0x01;

// some enumerations for "RunState"
static const char MOTOR_RUN_STATE_IDLE = 0x00;
static const char MOTOR_RUN_STATE_RUNNING = 0x20;

// some enumerations for LS commands
static const char US_ADDRESS = 0x02;

// some enumerations for the US
static const char SET_US_MODE = 0x41;
static const char READ_US_BYTE0 = 0x42;

static const char SET_US_CONTINUOUSINTERVAL = 0x40;
static const char US_MODE_OFF = 0x00;
static const char US_MODE_SINGLESHOT = 0x01;
static const char US_MODE_CONTINUOUS = 0x02;
static const char US_MODE_EVENTCAPTURE = 0x03;

/** Constructor */
NXT_USB::NXT_USB()
{
  usbConn = new NXT_USB_linux();

}
/** Destructor */
NXT_USB::~NXT_USB()
{
  delete usbConn;
}

/** Open the USB connection for the LEGO NXT device */

int NXT_USB::OpenLegoUSB()
{
  return this->usbConn->OpenLegoUSB();
}

/** Closes the USB connection for the LEGO NXT device */
int NXT_USB::CloseLegoUSB()
{
  return this->usbConn->CloseLegoUSB();
}

/** Set the sensor type of the input port defined by the "port" parameter
 * to a light sensor.  Use active = true if the light sensor should be active
```

```
 * (reflected light) and active = false if it should be passive (ambient light)
*/
void NXT_USB::SetSensorLight(int port, bool active)
{

  char outbuf[] = {NO_RESPONSE, SETINPUTMODE, port, 0, PCTFULLSCALEMODE};
  char inbuf[3];
  if (active)
  {
    outbuf[3] = LIGHT_ACTIVE;
  }
  else
  {
    outbuf[3] = LIGHT_INACTIVE;
  }
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the sensor type of the input port defined by the "port" parameter to a
 * touch sensor. */
void NXT_USB::SetSensorTouch(int port)
{
  char outbuf[] = {NO_RESPONSE, SETINPUTMODE, port, SWITCH, BOOLEANMODE};
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the sensor type of the input port defined by the "port" parameter to a
 * sound sensor.  Use dba = true for the DBA reading, and dba = false for the
 * DB reading */
void NXT_USB::SetSensorSound(int port, bool dba)
{
  char outbuf[] = {NO_RESPONSE, SETINPUTMODE, port, 0, PCTFULLSCALEMODE};

  if (dba)
  {
  outbuf[3] = SOUND_DBA;
  }
  else
  {
  outbuf[3] = SOUND_DB;
  }
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the sensor type of the input port defined by the "port" parameter to a
 * ultrasonic sensor. */
void NXT_USB::SetSensorUS(int port)
{
  char outbuf[] = {RESPONSE, SETINPUTMODE, port, LOWSPEED_9V, RAWMODE};
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Turn the ultrasonic sensor off. */
void NXT_USB::SetUSOff(int port)
{
  char outbuf[] = {RESPONSE, LSWRITE, port, 3, 0, US_ADDRESS, SET_US_MODE,
US_MODE_OFF};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the ultrasonic sensor to single shot mode. */
void NXT_USB::SetUSSingleShot(int port)
{
  char outbuf[] = {RESPONSE, LSWRITE, port, 3, 0, US_ADDRESS, SET_US_MODE,
```

```
US_MODE_SINGLESHOT};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the ultrasonic sensor to continuous mode. */
void NXT_USB::SetUSContinuous(int port)
{
  char outbuf[] = {RESPONSE, LSWRITE, port, 3, 0, US_ADDRESS, SET_US_MODE,
US_MODE_CONTINUOUS};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the ultrasonic sensor to event capture mode. */
void NXT_USB::SetUSEventCapture(int port)
{
  char outbuf[] = {RESPONSE, LSWRITE, port, 3, 0, US_ADDRESS, SET_US_MODE,
US_MODE_EVENTCAPTURE};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Set the ultrasonic sensor to continuous mode, with a defined interval. */
void NXT_USB::SetUSContinuousInterval(int port, int interval)
{
  char outbuf[] = {RESPONSE, LSWRITE, port, 3, 0, US_ADDRESS,
SET_US_CONTINUOUSINTERVAL, interval};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Read from the light sensor (returns -1 on error). */
int NXT_USB::GetLightSensor(int port)
{
  char outbuf[] = {RESPONSE, GETINPUTVALUES, port};
  char inbuf[16];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
  return (((int) inbuf[13])*256) + ((int) inbuf[12]);
}

/** Read from the touch sensor (returns -1 on error). */
bool NXT_USB::GetTouchSensor(int port)
{
  char outbuf[] = {RESPONSE, GETINPUTVALUES, port};
  char inbuf[16];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
  return (bool) (((int) inbuf[13])*256) + ((int) inbuf[12]);
}

/** Read from the sound sensor (returns -1 on error). */
int NXT_USB::GetSoundSensor(int port)
{
  char outbuf[] = {RESPONSE, GETINPUTVALUES, port};
  char inbuf[16];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
  return (((int) inbuf[13])*256) + ((int) inbuf[12]);
}

/** Read from the ultrasonic sensor (returns -1 on error). */
int NXT_USB::GetUSSensor(int port)
{
  //command the sensor to read one byte
  char outbuf[] = {RESPONSE, LSWRITE, port, 2, 1, US_ADDRESS, READ_US_BYTE0};
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));

  // wait until there are bytes to be read
```

```
  int lsStatus = 0;
  int i = 0; // timeout if we wait too long
  do
  {
    lsStatus = LSGetStatus(port);
    i++;
  } while (lsStatus == 0 && i < 10);

  // timed out
  if (lsStatus == 0)
  {
    return -1;
  }

  // perform read
  char outbuf2[] = {RESPONSE, LSREAD, port};
  char inbuf2[20];
  this->usbConn->SendCommand(outbuf2, sizeof(outbuf2), inbuf2, sizeof(inbuf2));

  return (int) inbuf2[4];
}

/** Turn on the motor connected to the output port defined by the "port"
parameter
 * at the power level defined by "power" (Power must be between -100 (100%
 * power reverse) and 100 (100% power forward).  The motor will turn
indefinitely
 * until it is turned off */
void NXT_USB::SetMotorOn(int port, int power)
{
  if (power < -100 || power > 100)
  {
    return;
  }

  char outbuf[] = {NO_RESPONSE, SETOUTPUTSTATE, port, power, MOTORON | BRAKE |
REGULATED, REGULATION_MODE_MOTOR_SPEED, 0, MOTOR_RUN_STATE_RUNNING, 0, 0, 0, 0,
0};
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Turn on the motor connected to the output port defined by the "port"
parameter
 * at the power level defined by "power" for a specified number of degrees
 *   Power must be between -100 (100% power reverse) and 100 (100% power
forward).
 *   TachoCount is the absolute number of degrees to turn and must be greater
than
 *   or equal to zero
 *   Use this function instead of MoveMotor if you want to be able to run other
 *   pieces of code while the motor is turning */
void NXT_USB::SetMotorOn(int port, int power, int tachoCount)
{
  if (power < -100 || power > 100 || tachoCount < 0)
  {
    return;
  }

  char byte8 = 0;
  char byte9 = 0;

  if (tachoCount > 256)
  {
    byte9 = tachoCount / 256;
    tachoCount = tachoCount - (byte9 * 256);
  }
  if (tachoCount > 0)
```

```
  {
    byte8 = tachoCount;
  }

  char outbuf[] = {NO_RESPONSE, SETOUTPUTSTATE, port, power, MOTORON | BRAKE |
REGULATED, REGULATION_MODE_MOTOR_SPEED, 0, MOTOR_RUN_STATE_RUNNING, byte8,
byte9, 0, 0, 0};
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Turn on the motor connected to the output port defined by the "port"
parameter
 * at the power level defined by "power" for a specified number of degrees
 *   Power must be between -100 (100% power reverse) and 100 (100% power
forward).
 *   TachoCount is the absolute number of degrees to turn and must be greater
than
 *   or equal to zero
 *   Use this function instead of SetMotorOn if you want your program to stall
until
 *   the motor has finished turning */
void NXT_USB::MoveMotor(int port, int power, int tachoCount)
{
  if (power < -100 || power > 100 || tachoCount <= 0)
  {
    return;
  }

    // figure out where we want to be
  int goalDegrees;
  if (power > 0)
  {
    goalDegrees = this->GetMotorRotation(port, false) + tachoCount;
  }
  else
  {
    goalDegrees = this->GetMotorRotation(port, false) - tachoCount;
  }

    //peform the movement
  this->SetMotorOn(port, power);
    int i = 0; // new
  if (power > 0)
  {
    while (this->GetMotorRotation(port, false) < goalDegrees) {}

  }
  else
  {
    while (this->GetMotorRotation(port, false) > goalDegrees) {}

  }
  this->StopMotor(port, true);
}

/** Stop the motor connected to the output port defined by the "port" variable.
 * Use brake = true for an active brake, and brake = false if you just want to
 * stop the motor from continuing to turn */
void NXT_USB::StopMotor(int port, bool brake)
{
  char outbuf[] = {NO_RESPONSE, SETOUTPUTSTATE, port, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0};

  if (brake)
  {
    outbuf[4] = MOTORON | BRAKE | REGULATED;
    outbuf[5] = REGULATION_MODE_MOTOR_SPEED;
```

```
    outbuf[7] = MOTOR_RUN_STATE_RUNNING;
  }
  else
  {
    outbuf[4] = FLOAT;
    outbuf[5] = REGULATION_MODE_IDLE;
    outbuf[7] = MOTOR_RUN_STATE_IDLE;
  }
  char inbuf[3];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Get the current rotation of the motor.  Use relative = true to get the
current
 *  number of degrees that the motor has rotated through since the last
programmed
 *  movement, and relative = false to get the current number of degrees that the
 *  motor has rotated through since to the last reset */
int NXT_USB::GetMotorRotation(int port, bool relative)
{
  char outbuf[] = {RESPONSE, GETOUTPUTSTATE, port};
  char inbuf[25];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));

  if (relative)
  {
    return (((int) inbuf[20])*16777216) + (((int) inbuf[19])*65536) + (((int)
inbuf[18])*256) + ((int) inbuf[17]);
  }
  else
  {
    return (((int) inbuf[24])*16777216) + (((int) inbuf[23])*65536) + (((int)
inbuf[22])*256) + ((int) inbuf[21]);
  }
}

/** Reset the rotation sensor of the motor.  Use relative = true to reset the
relative
 * number of degrees that the motor has rotated through since the last
programmed
 * movement, and relative = false to get the current number of degrees that the
 * motor has rotated through since the last reset */
void NXT_USB::ResetMotorPosition(int port, bool relative)
{
  char outbuf[] = {NO_RESPONSE, RESETMOTORPOSITION, port, relative};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Play a tone on the LEGO Mindstorms NXT.  Frequency must be in the range
 * 200 - 14000 and duration (in ms) must be in the range 0-65535. */
void NXT_USB::PlayTone(int frequency, int duration)
{
  if (frequency < 200 || frequency > 14000 || duration < 0 || duration > 65535)
  {
    return;
  }

  char byte2 = 0;
  char byte3 = 0;
  char byte4 = 0;
  char byte5 = 0;

  if (frequency > 256)
  {
    byte3 = frequency / 256;
    frequency = frequency - (byte3 * 256);
  }
```

```
  if (frequency > 0)
  {
    byte2 = frequency;
  }

  if (duration> 256)
  {
    byte5 = duration / 256;
    duration = duration - (byte5 * 256);
  }
  if (duration > 0)
  {
    byte4 = duration;
  }
  char outbuf[] = {NO_RESPONSE, PLAYTONE, byte2, byte3, byte4, byte5};
  char inbuf[3];
  this->usbConn->SendCommand(outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
}

/** Get the device filename for the LEGO Mindstorms NXT device */
char * NXT_USB::GetDeviceFilename()
{
  return this->usbConn->GetDeviceFilename();;
}

/** Get the vendor ID number for the LEGO Mindstorms NXT device */
int NXT_USB::GetIDVendor()
{
  return this->usbConn->GetIDVendor();
}

/** Get the product ID number for the LEGO Mindstorms NXT device */
int NXT_USB::GetIDProduct()
{
  return this->usbConn->GetIDProduct();
}

/** Get a textual version of the status of the USB connection */
char * NXT_USB::GetStatus()
{
  return this->usbConn->GetStatus();
}

/** Sends a direct command to the Lego NXT over USB.  This function should be
used by only advanced users.
    -outbuf - the direct command (see the official LEGO Mindstorms documentation
for direct commands)
    -outbufSize - the size of the outbuf array
    -inbuf - the response message, NULL if no response is needed
    -inbufSize - the size of the response message, 0 if no response is needed
*/
void NXT_USB::SendCommand (char * outbuf, int outbufSize, char * inbuf, int
inbufSize)
{
  return this->usbConn->SendCommand(outbuf, outbufSize, inbuf, inbufSize);
}

/** Get the LS status of the Lego Mindstorms NXT - should only be used by
advanced
* users */
int NXT_USB::LSGetStatus(int port)
{
  char outbuf[] = {RESPONSE, LSGETSTATUS, port};
  char inbuf[4];
  this->usbConn->SendCommand (outbuf, sizeof(outbuf), inbuf, sizeof(inbuf));
  return (int) inbuf[3];
}
```

Future Technology Devices International Ltd.

FT232R USB UART I.C.

Incorporating Clock Generator Output and FTDIChip-ID™ Security Dongle

The **FT232R** *is the latest device to be added to FTDI's range of USB UART interface Integrated Circuit Devices. The FT232R is a USB to serial UART interface with optional clock generator output, and the new FTDIChip-ID™ security dongle feature. In addition, asynchronous and synchronous bit bang interface modes are available. USB to serial designs using the FT232R have been further simplified by fully integrating the external EEPROM, clock circuit and USB resistors onto the device.*

The FT232R adds two new functions compared with its predecessors, effectively making it a "3-in-1" chip for some application areas. The internally generated clock (6MHz, 12MHz, 24MHz, and 48MHz) can be brought out of the device and used to drive a microcontroller or external logic. A unique number (the FTDIChip-ID™) is burnt into the device during manufacture and is readable over USB, thus forming the basis of a security dongle which can be used to protect customer application software from being copied.

The FT232R is available in Pb-free (RoHS compliant) compact 28-Lead SSOP and QFN-32 packages.

1. Features

1.1 Hardware Features

- Single chip USB to asynchronous serial data transfer interface.
- Entire USB protocol handled on the chip - No USB-specific firmware programming required.
- UART interface support for 7 or 8 data bits, 1 or 2 stop bits and odd / even / mark / space / no parity.
- Fully assisted hardware or X-On / X-Off software handshaking.
- Data transfer rates from 300 baud to 3 Megabaud (RS422 / RS485 and at TTL levels) and 300 baud to 1 Megabaud (RS232).
- 256 byte receive buffer and 128 byte transmit buffer utilising buffer smoothing technology to allow for high data throughput.
- FTDI's royalty-free VCP and D2XX drivers eliminate the requirement for USB driver development in most cases.
- In-built support for event characters and line break condition.
- New USB FTDIChip-ID™ feature.
- New configurable CBUS I/O pins.
- Auto transmit buffer control for RS485 applications.
- Transmit and receive LED drive signals.
- New 48MHz, 24MHz,12MHz, and 6MHz clock output signal Options for driving external MCU or FPGA.
- FIFO receive and transmit buffers for high data throughput.
- Adjustable receive buffer timeout.
- Synchronous and asynchronous bit bang mode interface options with RD# and WR# strobes.
- New CBUS bit bang mode option.
- Integrated 1024 Bit internal EEPROM for storing USB VID, PID, serial number and product description strings, and CBUS I/O configuration.
- Device supplied preprogrammed with unique USB serial number.
- Support for USB suspend and resume.
- Support for bus powered, self powered, and high-power bus powered USB configurations.
- Integrated 3.3V level converter for USB I/O .
- Integrated level converter on UART and CBUS for interfacing to 5V - 1.8V Logic.
- True 5V / 3.3V / 2.8V / 1.8V CMOS drive output and TTL input.
- High I/O pin output drive option.
- Integrated USB resistors.
- Integrated power-on-reset circuit.
- Fully integrated clock - no external crystal, oscillator, or resonator required.
- Fully integrated AVCC supply filtering - No separate AVCC pin and no external R-C filter required.
- UART signal inversion option.
- USB bulk transfer mode.
- 3.3V to 5.25V Single Supply Operation.
- Low operating and USB suspend current.
- Low USB bandwidth consumption.
- UHCI / OHCI / EHCI host controller compatible
- USB 2.0 Full Speed compatible.
- -40°C to 85°C extended operating temperature range.
- Available in compact Pb-free 28 Pin SSOP and QFN-32 packages (both RoHS compliant).

1.2 Driver Support

Royalty-Free VIRTUAL COM PORT (VCP) DRIVERS for...

- Windows 98, 98SE, ME, 2000, Server 2003, XP.
- Windows Vista / Longhorn*
- Windows XP 64-bit.*
- Windows XP Embedded.
- Windows CE.NET 4.2 & 5.0
- MAC OS 8 / 9, OS-X
- Linux 2.4 and greater

Royalty-Free D2XX *Direct* Drivers (USB Drivers + DLL S/W Interface)

- Windows 98, 98SE, ME, 2000, Server 2003, XP.
- Windows Vista / Longhorn*
- Windows XP 64-bit.*
- Windows XP Embedded.
- Windows CE.NET 4.2 & 5.0
- Linux 2.4 and greater

The drivers listed above are all available to download for free from the FTDI website. Various 3rd Party Drivers are also available for various other operating systems - see the FTDI website for details.

* Currently Under Development. Contact FTDI for availability.

1.3 Typical Applications

- USB to RS232 / RS422 / RS485 Converters
- Upgrading Legacy Peripherals to USB
- Cellular and Cordless Phone USB data transfer cables and interfaces
- Interfacing MCU / PLD / FPGA based designs to USB
- USB Audio and Low Bandwidth Video data transfer
- PDA to USB data transfer
- USB Smart Card Readers
- USB Instrumentation
- USB Industrial Control
- USB MP3 Player Interface
- USB FLASH Card Reader / Writers
- Set Top Box PC - USB interface
- USB Digital Camera Interface
- USB Hardware Modems
- USB Wireless Modems
- USB Bar Code Readers
- USB Software / Hardware Encryption Dongles

2. Enhancements

2.1 Device Enhancements and Key Features

This section summarises the enhancements and the key features of the FT232R device. For further details, consult the device pin-out description and functional description sections.

Integrated Clock Circuit - Previous generations of FTDI's USB UART devices required an external crystal or ceramic resonator. The clock circuit has now been integrated onto the device meaning that no crystal or ceramic resonator is required. However, if required, an external 12MHz crystal can be used as the clock source.

Integrated EEPROM - Previous generations of FTDI's USB UART devices required an external EEPROM if the device were to use USB Vendor ID (VID), Product ID (PID), serial number and product description strings other than the default values in the device itself. This external EEPROM has now been integrated onto the FT232R chip meaning that all designs have the option to change the product description strings. A user area of the internal EEPROM is available for storing additional data. The internal EEPROM is programmable in circuit, over USB without any additional voltage requirement.

Preprogrammed EEPROM - The FT232R is supplied with its internal EEPROM preprogrammed with a serial number which is unique to each individual device. This, in most cases, will remove the need to program the device EEPROM.

Integrated USB Resistors - Previous generations of FTDI's USB UART devices required two external series resistors on the USBDP and USBDM lines, and a 1.5 kΩ pull up resistor on USBDP. These three resistors have now been integrated onto the device.

Integrated AVCC Filtering - Previous generations of FTDI's USB UART devices had a separate AVCC pin - the supply to the internal PLL. This pin required an external R-C filter. The separate AVCC pin is now connected internally to VCC, and the filter has now been integrated onto the chip.

Less External Components - Integration of the crystal, EEPROM, USB resistors, and AVCC filter will substantially reduce the bill of materials cost for USB interface designs using the FT232R compared to its FT232BM predecessor.

Transmit and Receive Buffer Smoothing - The FT232R's 256 byte receive buffer and 128 byte transmit buffer utilise new buffer smoothing technology to allow for high data throughput.

Configurable CBUS I/O Pin Options - There are now 5 configurable Control Bus (CBUS) lines. Options are **TXDEN** - transmit enable for RS485 designs, **PWREN#** - Power control for high power, bus powered designs, **TXLED#** - for pulsing an LED upon transmission of data, **RXLED#** - for pulsing an LED upon receiving data, **TX&RXLED#** - which will pulse an LED upon transmission OR reception of data, **SLEEP#** - indicates that the device going into USB suspend mode, **CLK48 / CLK24 / CLK12 / CLK6** - 48MHz, 24MHz,12MHz, and 6MHz clock output signal options. There is also the option to bring out bit bang mode read and write strobes (see below). The CBUS lines can be configured with any one of these output options by setting bits in the internal EEPROM. The device is supplied with the most commonly used pin definitions preprogrammed - see Section 10 for details.

Enhanced Asynchronous Bit Bang Mode with RD# and WR# Strobes - The FT232R supports FTDI's BM chip bit bang mode. In bit bang mode, the eight UART lines can be switched from the regular interface mode to an 8-bit general purpose I/O port. Data packets can be sent to the device and they will be sequentially sent to the interface at a rate controlled by an internal timer (equivalent to the baud rate prescaler). With the FT232R device this mode has been enhanced so that the internal RD# and WR# strobes are now brought out of the device which can be used to allow external logic to be clocked by accesses to the bit bang I/O bus. This option will be described more fully in a separate application note.

Synchronous Bit Bang Mode - Synchronous bit bang mode differs from asynchronous bit bang mode in that the interface pins are only read when the device is written to. Thus making it easier for the controlling program to measure the response to an output stimulus as the data returned is synchronous to the output data. The feature was previously seen in FTDI's FT2232C device. This option will be described more fully in a separate application note.

CBUS Bit Bang Mode - This mode allows four of the CBUS pins to be individually configured as GPIO pins, similar to Asynchronous bit bang mode. It is possible to use this mode while the UART interface is being used, thus providing up to four general purpose I/O pins which are available during normal operation. An application note describing this feature is available separately from the FTDI website.

Lower Supply Voltage - Previous generations of the chip required 5V supply on the VCC pin. The FT232R will work with a Vcc supply in the range 3.3V - 5.25V. Bus powered designs would still take their supply from the 5V on the USB bus, but for self powered designs where only 3.3V is available and there is no 5V supply there is no longer any need for an additional external regulator.

Integrated Level Converter on UART Interface and Control Signals - VCCIO pin supply can be from 1.8V to 5V. Connecting the VCCIO pin to 1.8V, 2.8V, or 3.3V allows the device to directly interface to 1.8V, 2.8V or 3.3V and other logic families without the need for external level converter I.C. devices.

5V / 3.3V / 2.8V / 1.8V Logic Interface - The FT232R provides *true* CMOS Drive Outputs and TTL level Inputs.

Integrated Power-On-Reset (POR) Circuit- The device incorporates an internal POR function. A RESET# pin is available in order to allow external logic to reset the FT232R where required. However, for many applications the RESET# pin can be left unconnected, or pulled up to VCCIO.

Lower Operating and Suspend Current - The device operating supply current has been further reduced to 15mA, and the suspend current has been reduced to around 70µA. This allows greater margin for peripheral designs to meet the USB suspend current limit of 500µA.

Low USB Bandwidth Consumption - The operation of the USB interface to the FT232R has been designed to use as little as possible of the total USB bandwidth available from the USB host controller.

High Output Drive Option - The UART interface and CBUS I/O pins can be made to drive out at three times the standard signal drive level thus allowing multiple devices to be driven, or devices that require a greater signal drive strength to be interfaced to the FT232R. This option is enabled in the internal EEPROM.

Power Management Control for USB Bus Powered, High Current Designs- The PWREN# signal can be used to directly drive a transistor or P-Channel MOSFET in applications where power switching of external circuitry is required. An option in the internal EEPROM makes the device gently pull down on its UART interface lines when the power is shut off (PWREN# is high). In this mode any residual voltage on external circuitry is bled to GND when power is removed, thus ensuring that external circuitry controlled by PWREN# resets reliably when power is restored.

UART Pin Signal Inversion - The sense of each of the eight UART signals can be individually inverted by setting options in the internal EEPROM. Thus, CTS# (active low) can be changed to CTS (active high), or TXD can be changed to TXD#.

FTDIChip-ID™ - Each FT232R is assigned a unique number which is burnt into the device at manufacture. This ID number cannot be reprogrammed by product manufacturers or end-users. This allows the possibility of using FT232R based dongles for software licensing. Further to this, a renewable license scheme can be implemented based on the FTDIChip-ID™ number when encrypted with other information. This encrypted number can be stored in the user area of the FT232R internal EEPROM, and can be decrypted, then compared with the protected FTDIChip-ID™ to verify that a license is valid. Web based applications can be used to maintain product licensing this way. An application note describing this feature is available separately from the FTDI website.

Improved EMI Performance - The reduced operating current and improved on-chip VCC decoupling significantly improves the ease of PCB design requirements in order to meet FCC, CE and other EMI related specifications.

Programmable Receive Buffer Timeout - The receive buffer timeout is used to flush remaining data from the receive buffer. This time defaults to 16ms, but is programmable over USB in 1ms increments from 1ms to 255ms, thus allowing the device to be optimised for protocols that require fast response times from short data packets.

Extended Operating Temperature Range - The FT232R operates over an extended temperature range of -40° to +85° C thus allowing the device to be used in automotive and industrial applications.

New Package Options - The FT232R is available in two packages - a compact 28 pin SSOP (**FT232RL**) and an ultra-compact 5mm x 5mm pinless QFN-32 package (**FT232RQ**). Both packages are lead (Pb) free, and use a 'green' compound. Both packages are fully compliant with European Union directive 2002/95/EC.

3. Block Diagram

3.1 Block Diagram (Simplified)

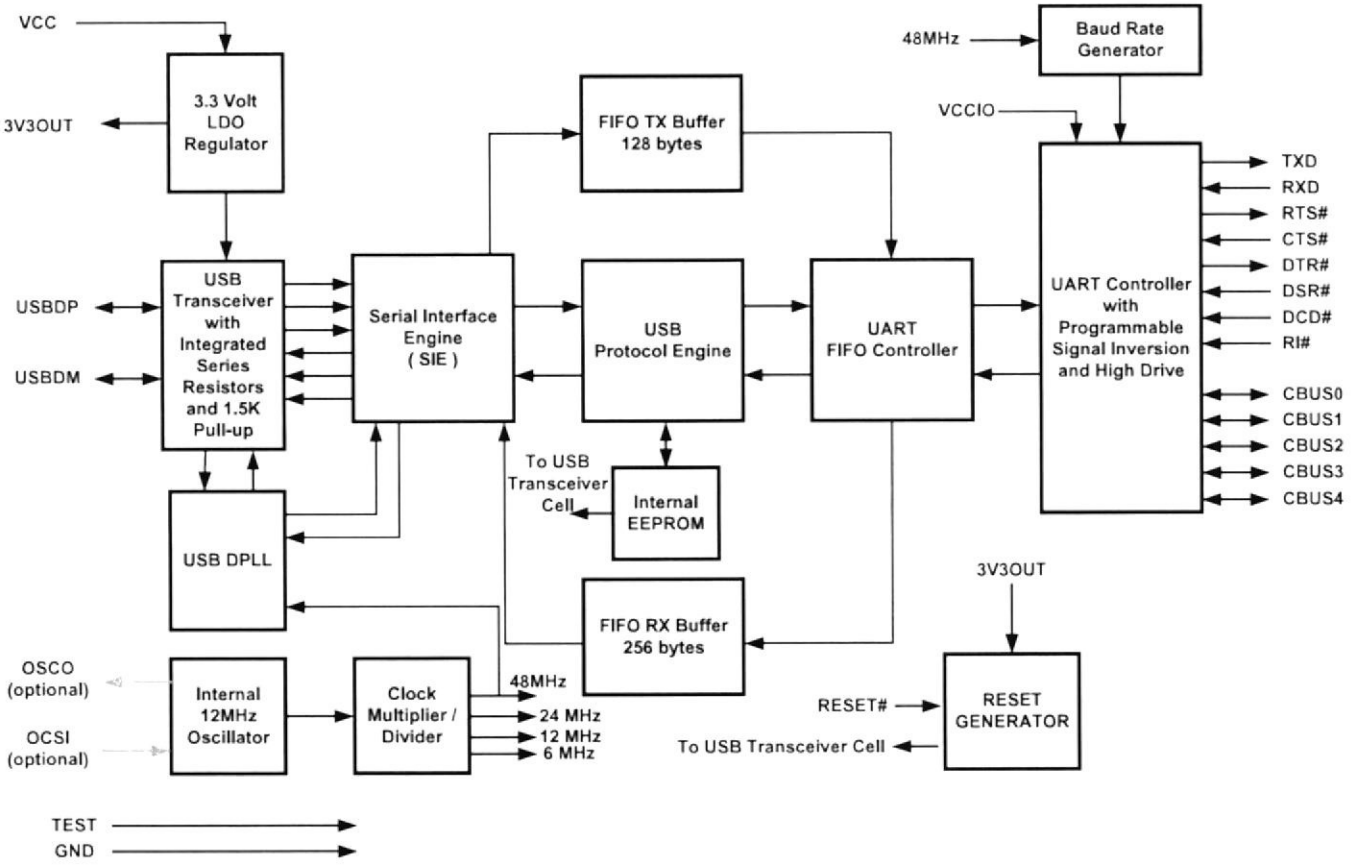


Figure 1 - FT232R Block Diagram

3.2 Functional Block Descriptions

3.3V LDO Regulator - The 3.3V LDO Regulator generates the 3.3V reference voltage for driving the USB transceiver cell output buffers. It requires an external decoupling capacitor to be attached to the 3V3OUT regulator output pin. It also provides 3.3V power to the 1.5kΩ internal pull up resistor on USBDP. The main function of this block is to power the USB Transceiver and the Reset Generator Cells rather than to power external logic. However, external circuitry requiring a 3.3V nominal supply at a current of around than 50mA could also draw its power from the 3V3OUT pin, if required.

USB Transceiver - The USB Transceiver Cell provides the USB 1.1 / USB 2.0 full-speed physical interface to the USB cable. The output drivers provide 3.3V level slew rate control signalling, whilst a differential receiver and two single ended receivers provide USB data in, SEO and USB Reset condition detection. This Cell also incorporates internal USB series resistors on the USB data lines, and a 1.5kΩ pull up resistor on USBDP.

USB DPLL - The USB DPLL cell locks on to the incoming NRZI USB data and provides separate recovered clock and data signals to the SIE block.

Internal 12MHz Oscillator - The Internal 12MHz Oscillator cell generates a 12MHz reference clock input to the x4 Clock multiplier. The 12MHz Oscillator is also used as the reference clock for the SIE, USB Protocol Engine and UART FIFO controller blocks

Clock Multiplier / Divider - The Clock Multiplier / Divider takes the 12MHz input from the Oscillator Cell and generates the 48MHz, 24MHz, 12MHz, and 6MHz reference clock signals. The 48Mz clock reference is used for the USB DPLL and the Baud Rate Generator blocks.

Serial Interface Engine (SIE) - The Serial Interface Engine (SIE) block performs the Parallel to Serial and Serial to Parallel conversion of the USB data. In accordance to the USB 2.0 specification, it performs bit stuffing / un-stuffing and CRC5 / CRC16 generation / checking on the USB data stream.

USB Protocol Engine - The USB Protocol Engine manages the data stream from the device USB control endpoint. It handles the low level USB protocol (Chapter 9) requests generated by the USB host controller and the commands for controlling the functional parameters of the UART.

FIFO TX Buffer (128 bytes) - Data from the USB data out endpoint is stored in the FIFO TX buffer and removed from the buffer to the UART transmit register under control of the UART FIFO controller.

FIFO RX Buffer (256 bytes) - Data from the UART receive register is stored in the FIFO RX buffer prior to being removed by the SIE on a USB request for data from the device data in endpoint.

UART FIFO Controller - The UART FIFO controller handles the transfer of data between the FIFO RX and TX buffers and the UART transmit and receive registers.

UART Controller with Programmable Signal Inversion and High Drive - Together with the UART FIFO Controller the UART Controller handles the transfer of data between the FIFO RX and FIFO TX buffers and the UART transmit and receive registers. It performs asynchronous 7 / 8 bit Parallel to Serial and Serial to Parallel conversion of the data on the RS232 (RS422 and RS485) interface. Control signals supported by UART mode include RTS, CTS, DSR , DTR, DCD and RI. The UART Controller also provides a transmitter enable control signal pin option (TXDEN) to assist with interfacing to RS485 transceivers. RTS / CTS, DSR / DTR and X-On / X-Off handshaking options are also supported. Handshaking, where required, is handled in hardware to ensure fast response times. The UART also supports the RS232 BREAK setting and detection conditions. A new feature, programmable in the internal EEPROM allows the UART signals to each be individually inverted. Another new EEPROM programmable feature allows a high signal drive strength to be enabled on the UART interface and CBUS pins.

Baud Rate Generator - The Baud Rate Generator provides a x16 clock input to the UART Controller from the 48MHz reference clock and consists of a 14 bit prescaler and 3 register bits which provide fine tuning of the baud rate (used to divide by a number plus a fraction or "sub-integer"). This determines the Baud Rate of the UART, which is programmable from 183 baud to 3 million baud.

The FT232R supports all standard baud rates and non-standard baud rates from 300 Baud up to 3 Megabaud. Achievable non-standard baud rates are calculated as follows -

Baud Rate = 3000000 / (n + *x*)

where n can be any integer between 2 and 16,384 (= 2^{14}) and *x* can be a sub-integer of the value 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, or 0.875. When n = 1, *x* = 0, i.e. baud rate divisors with values between 1 and 2 are not possible.

This gives achievable baud rates in the range 183.1 baud to 3,000,000 baud. When a non-standard baud rate is required simply pass the required baud rate value to the driver as normal, and the FTDI driver will calculate the required divisor, and set the baud rate. See FTDI application note AN232B-05 for more details.

RESET Generator - The integrated Reset Generator Cell provides a reliable power-on reset to the device internal circuitry on power up. A RESET# input pin is provided to allow other devices to reset the FT232R. RESET# can be tied to VCCIO or left unconnected, unless it is a requirement to reset the device from external logic or an external reset generator I.C.

Internal EEPROM - The internal EEPROM in the FT232R can be used to store USB Vendor ID (VID), Product ID (PID), device serial number, product description string, and various other USB configuration descriptors. The internal EEPROM is also used to configure the CBUS pin functions. The device is supplied with the internal EEPROM settings preprogrammed as described in Section 10.

4. Device Pin Out and Signal Descriptions

4.1 28-LD SSOP Package

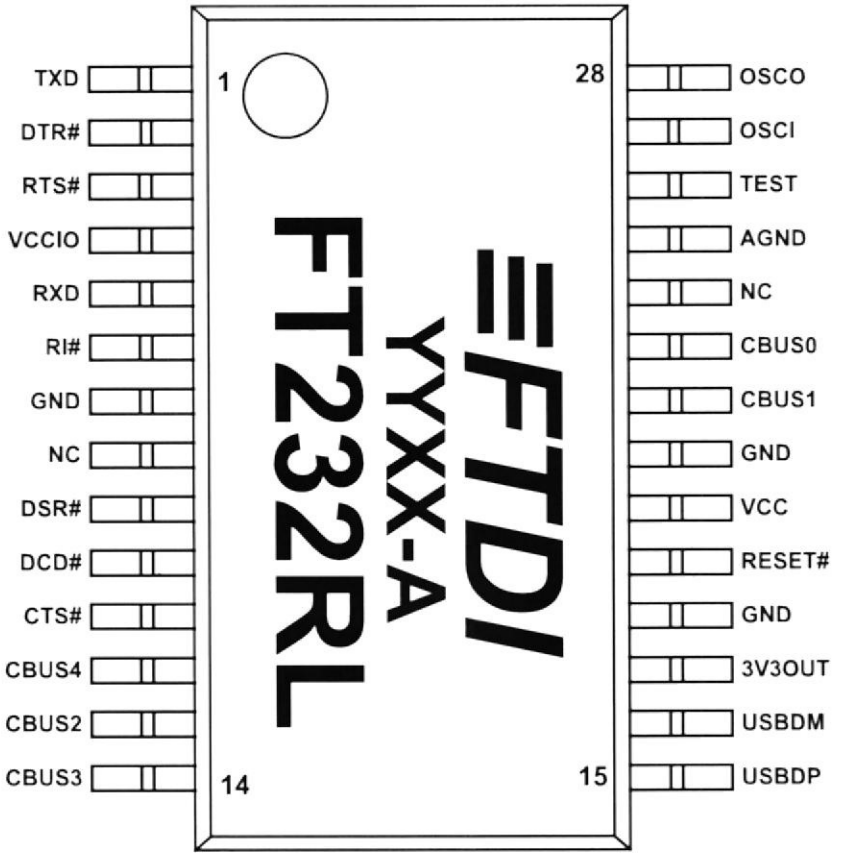


Figure 2 - 28 Pin SSOP Package Pin Out

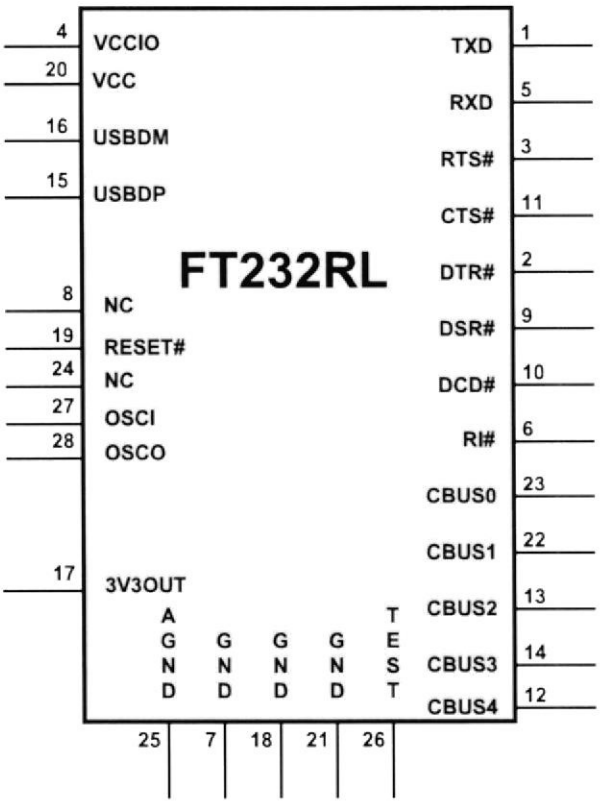


Figure 3 - 28 Pin SSOP Package Pin Out (Schematic Symbol)

4.2 SSOP-28 Package Signal Descriptions

Table 1 - SSOP Package Pin Out Description

Pin No.	Name	Type	Description
USB Interface Group			
15	USBDP	I/O	USB Data Signal Plus, incorporating internal series resistor and 1.5kΩ pull up resistor to 3.3V
16	USBDM	I/O	USB Data Signal Minus, incorporating internal series resistor.
Power and Ground Group			
4	VCCIO	PWR	+1.8V to +5.25V supply to the UART Interface and CBUS group pins (1...3, 5, 6, 9...14, 22, 23). In USB bus powered designs connect to 3V3OUT to drive out at 3.3V levels, or connect to VCC to drive out at 5V CMOS level. This pin can also be supplied with an external 1.8V - 2.8V supply in order to drive out at lower levels. It should be noted that in this case this supply should originate from the same source as the supply to Vcc. This means that in bus powered designs a regulator which is supplied by the 5V on the USB bus should be used.
7, 18, 21	GND	PWR	Device ground supply pins
17	3V3OUT	Output	3.3V output from integrated L.D.O. regulator. This pin should be decoupled to ground using a 100nF capacitor. The prime purpose of this pin is to provide the internal 3.3V supply to the USB transceiver cell and the internal 1.5kΩ pull up resistor on USBDP. Up to 50mA can be drawn from this pin to power external logic if required. This pin can also be used to supply the FT232R's VCCIO pin.
20	VCC	PWR	3.3V to 5.25V supply to the device core.
25	AGND	PWR	Device analog ground supply for internal clock multiplier
Miscellaneous Signal Group			
8, 24	NC	NC	No internal connection.
19	RESET#	Input	Can be used by an external device to reset the FT232R. If not required can be left unconnected, or pulled up to VCCIO.
26	TEST	Input	Puts the device into I.C. test mode. Must be tied to GND for normal operation.
27	OSCI	Input	Input to 12MHz Oscillator Cell. Optional - Can be left unconnected for normal operation. *
28	OSCO	Output	Output from 12MHz Oscillator Cell. Optional - Can be left unconnected for normal operation if internal oscillator is used. *
UART Interface and CBUS Group **			
1	TXD	Output	Transmit Asynchronous Data Output.
2	DTR#	Output	Data Terminal Ready Control Output / Handshake signal.
3	RTS#	Output	Request To Send Control Output / Handshake signal.
5	RXD	Input	Receive Asynchronous Data Input.
6	RI#	Input	Ring Indicator Control Input. When remote wake up is enabled in the internal EEPROM taking RI# low can be used to resume the PC USB host controller from suspend.
9	DSR#	Input	Data Set Ready Control Input / Handshake signal.
10	DCD#	Input	Data Carrier Detect Control input.
11	CTS#	Input	Clear to Send Control input / Handshake signal.
12	CBUS4	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is SLEEP#. See CBUS Signal Options, Table 3.
13	CBUS2	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is TXDEN. See CBUS Signal Options, Table 3.
14	CBUS3	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is PWREN#. See CBUS Signal Options, Table 3.
22	CBUS1	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is RXLED#. See CBUS Signal Options, Table 3.
23	CBUS0	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is TXLED#. See CBUS Signal Options, Table 3.

* Contact FTDI technical support for details on how to use an external crystal, ceramic resonator, or oscillator with the FT232R.

** When used in Input Mode, these pins are pulled to VCCIO via internal 200kΩ resistors. These pins can be programmed to gently pull low during USB suspend (PWREN# = "1") by setting an option in the internal EEPROM.

4.3 QFN-32 Package

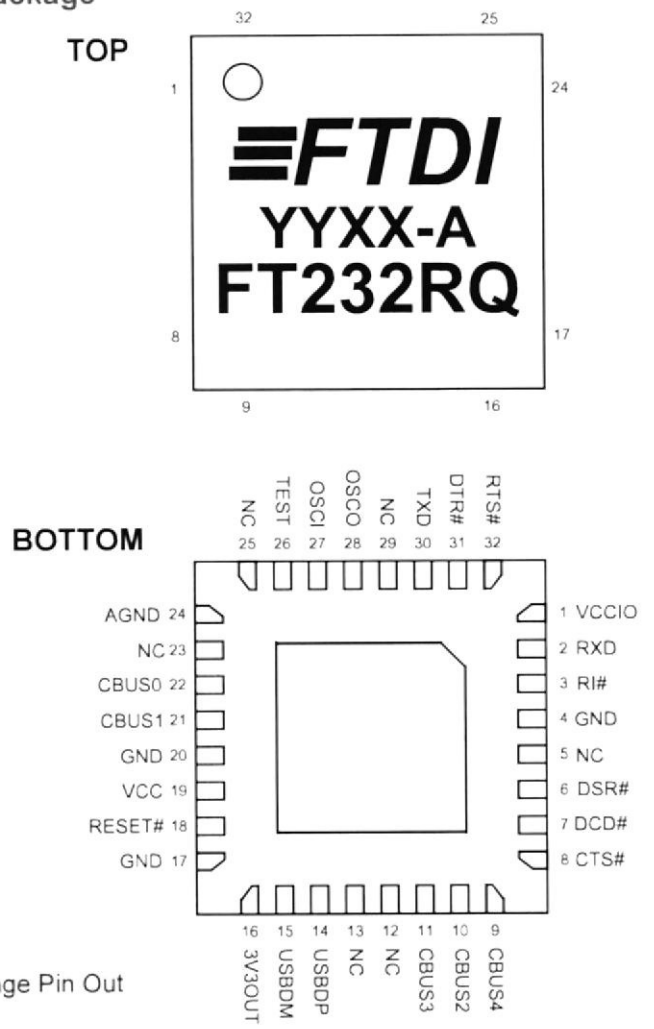


Figure 4 - QFN-32 Package Pin Out

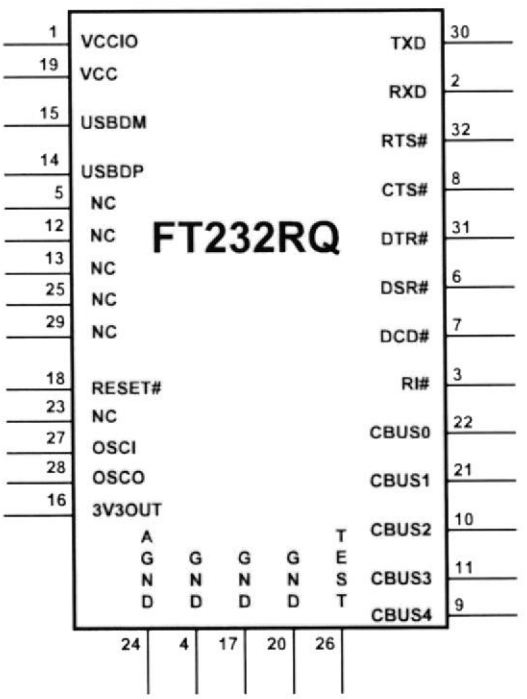


Figure 5 - QFN-32 Package Pin Out (Schematic Symbol)

4.4 QFN-32 Package Signal Descriptions

Table 2 - QFN Package Pin Out Description

Pin No.	Name	Type	Description
USB Interface Group			
14	USBDP	I/O	USB Data Signal Plus, incorporating internal series resistor and 1.5kΩ pull up resistor to 3.3V
15	USBDM	I/O	USB Data Signal Minus, incorporating internal series resistor.
Power and Ground Group			
1	VCCIO	PWR	+1.8V to +5.25V supply to UART Interface and CBUS group pins (2,3, 6, ...,11, 21, 22, 30,..32). In USB bus powered designs connect to 3V3OUT to drive out at 3.3V levels, or connect to VCC to drive out at 5V CMOS level. This pin can also be supplied with an external 1.8V - 2.8V supply in order to drive out at lower levels. It should be noted that in this case this supply should originate from the same source as the supply to Vcc. This means that in bus powered designs a regulator which is supplied by the 5V on the USB bus should be used.
4, 17, 20	GND	PWR	Device ground supply pins
16	3V3OUT	Output	3.3V output from integrated L.D.O. regulator. This pin should be decoupled to ground using a 100nF capacitor. The prime purpose of this pin is to provide the internal 3.3V supply to the USB transceiver cell and the internal 1.5kΩ pull up resistor on USBDP. Up to 50mA can be drawn from this pin to power external logic if required. This pin can also be used to supply the FT232R's VCCIO pin.
19	VCC	PWR	3.3V to 5.25V supply to the device core.
24	AGND	PWR	Device analog ground supply for internal clock multiplier
Miscellaneous Signal Group			
5, 12, 13, 23, 25, 29	NC	NC	No internal connection.
18	RESET#	Input	Can be used by an external device to reset the FT232R. If not required can be left unconnected or pulled up to VCCIO.
26	TEST	Input	Puts the device into I.C. test mode. Must be tied to GND for normal operation.
27	OSCI	Input	Input to 12MHz Oscillator Cell. Optional - Can be left unconnected for normal operation. *
28	OSCO	Output	Output from 12MHz Oscillator Cell. Optional - Can be left unconnected for normal operation if internal oscillator is used. *
UART Interface and CBUS Group **			
30	TXD	Output	Transmit Asynchronous Data Output.
31	DTR#	Output	Data Terminal Ready Control Output / Handshake signal.
32	RTS#	Output	Request To Send Control Output / Handshake signal.
2	RXD	Input	Receive Asynchronous Data Input.
3	RI#	Input	Ring Indicator Control Input. When remote wake up is enabled in the internal EEPROM taking RI# low can be used to resume the PC USB host controller from suspend.
6	DSR#	Input	Data Set Ready Control Input / Handshake signal.
7	DCD#	Input	Data Carrier Detect Control input.
8	CTS#	Input	Clear to Send Control input / Handshake signal.
9	CBUS4	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is SLEEP#. See CBUS Signal Options, Table 3.
10	CBUS2	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is TXDEN. See CBUS Signal Options, Table 3.
11	CBUS3	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is PWREN#. See CBUS Signal Options, Table 3.
21	CBUS1	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is RXLED#. See CBUS Signal Options, Table 3.
22	CBUS0	I/O	Configurable CBUS I/O Pin. Function of this pin is configured in the device internal EEPROM. Factory Default function is TXLED#. See CBUS Signal Options, Table 3.

* Contact FTDI technical support for details on how to use an external crystal, ceramic resonator, or oscillator with the FT232R.

** When used in Input Mode, these pins are pulled to VCCIO via internal 200kΩ resistors. These pins can be programmed to gently pull low during USB suspend (PWREN# = "1") by setting an option in the internal EEPROM.

4.5 CBUS Signal Options

The following options can be configured on the CBUS I/O pins. CBUS signal options are common to both package versions of the FT232R. These options are all configured in the internal EEPROM using the utility software MPROG, which can be downloaded from the FTDI website. The default configuration is described in Section 10.

Table 3 - CBUS Signal Options

CBUS Signal Option	Available On CBUS Pin...	Description
TXDEN	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Enable transmit data for RS485
PWREN#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Goes low after the device is configured by USB, then high during USB suspend. Can be used to control power to external logic P-Channel logic level MOSFET switch. Enable the interface pull-down option when using the PWREN# pin in this way.
TXLED#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Transmit data LED drive - pulses low when transmitting data via USB. See Section 9 for more details.
RXLED#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Receive data LED drive - pulses low when receiving data via USB. See Section 9 for more details.
TX&RXLED#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	LED drive - pulses low when transmitting or receiving data via USB. See Section 9 for more details.
SLEEP#	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	Goes low during USB suspend mode. Typically used to power down an external TTL to RS232 level converter I.C. in USB to RS232 converter designs.
CLK48	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	48MHz Clock output.
CLK24	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	24MHz Clock output.
CLK12	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	12MHz Clock output.
CLK6	CBUS0, CBUS1, CBUS2, CBUS3, CBUS4	6MHz Clock output.
CBitBangI/O	CBUS0, CBUS1, CBUS2, CBUS3	CBUS bit bang mode option. Allows up to 4 of the CBUS pins to be used as general purpose I/O. Configured individually for CBUS0, CBUS1, CBUS2 and CBUS3 in the internal EEPROM. A separate application note will describe in more detail how to use CBUS bit bang mode.
BitBangWRn	CBUS0, CBUS1, CBUS2, CBUS3	Synchronous and asynchronous bit bang mode WR# strobe Output
BitBangRDn	CBUS0, CBUS1, CBUS2, CBUS3	Synchronous and asynchronous bit bang mode RD# strobe Output

5. Package Parameters

The FT232R is supplied in two different packages. The FT232RL is the SSOP-28 option and the FT232RQ is the QFN-32 package option. The solder reflow profile for both packages is described in Section 5.3.

5.1 SSOP-28 Package Dimensions

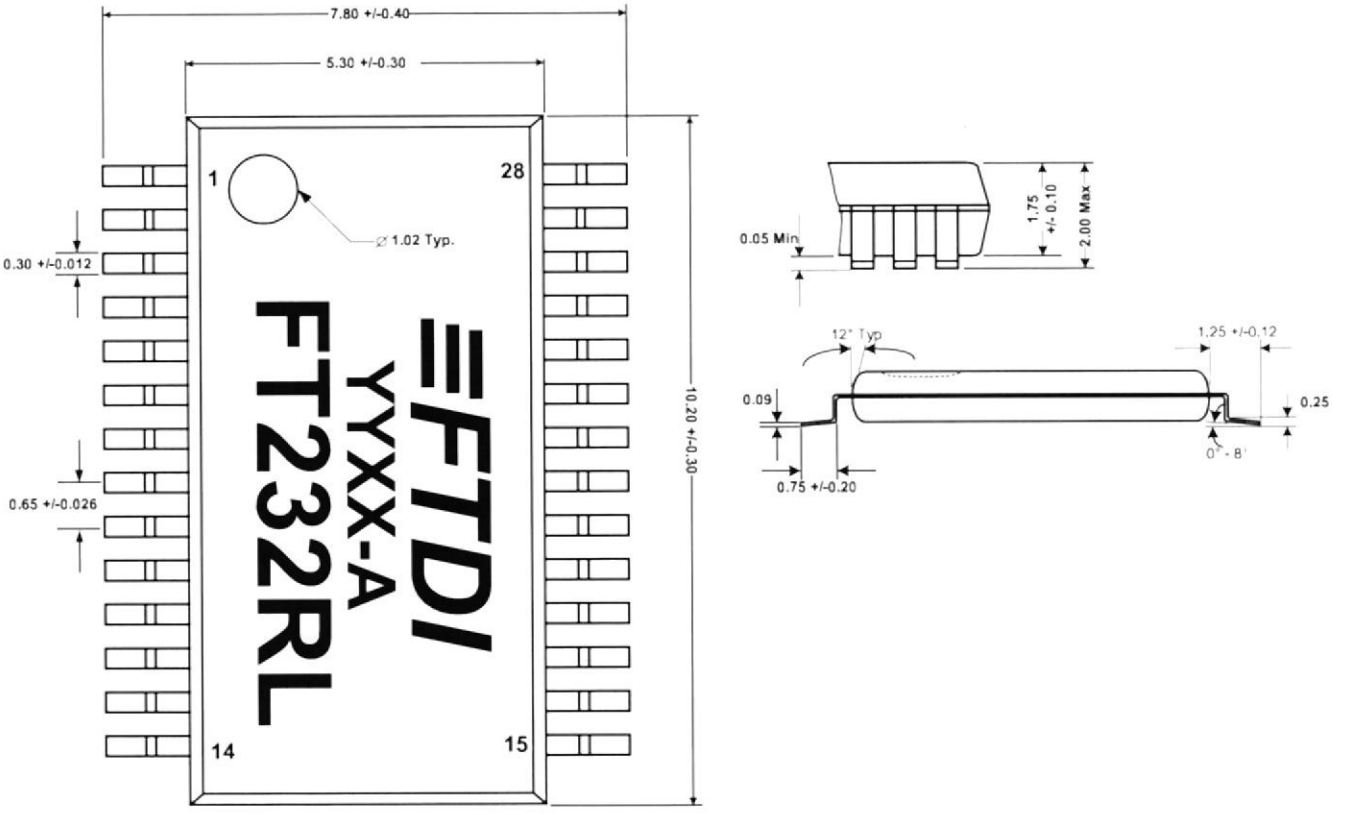


Figure 6 - SSOP-28 Package Dimensions

The FT232RL is supplied in a RoHS compliant 28 pin SSOP package. The package is lead (Pb) free and uses a 'green' compound. The package is fully compliant with European Union directive 2002/95/EC.

This package has a 5.30mm x 10.20mm body (7.80mm x 10.20mm including pins). The pins are on a 0.65 mm pitch. The above mechanical drawing shows the SSOP-28 package – all dimensions are in millimetres.

The date code format is **YYXX** where XX = 2 digit week number, YY = 2 digit year number.

5.2 QFN-32 Package Dimensions

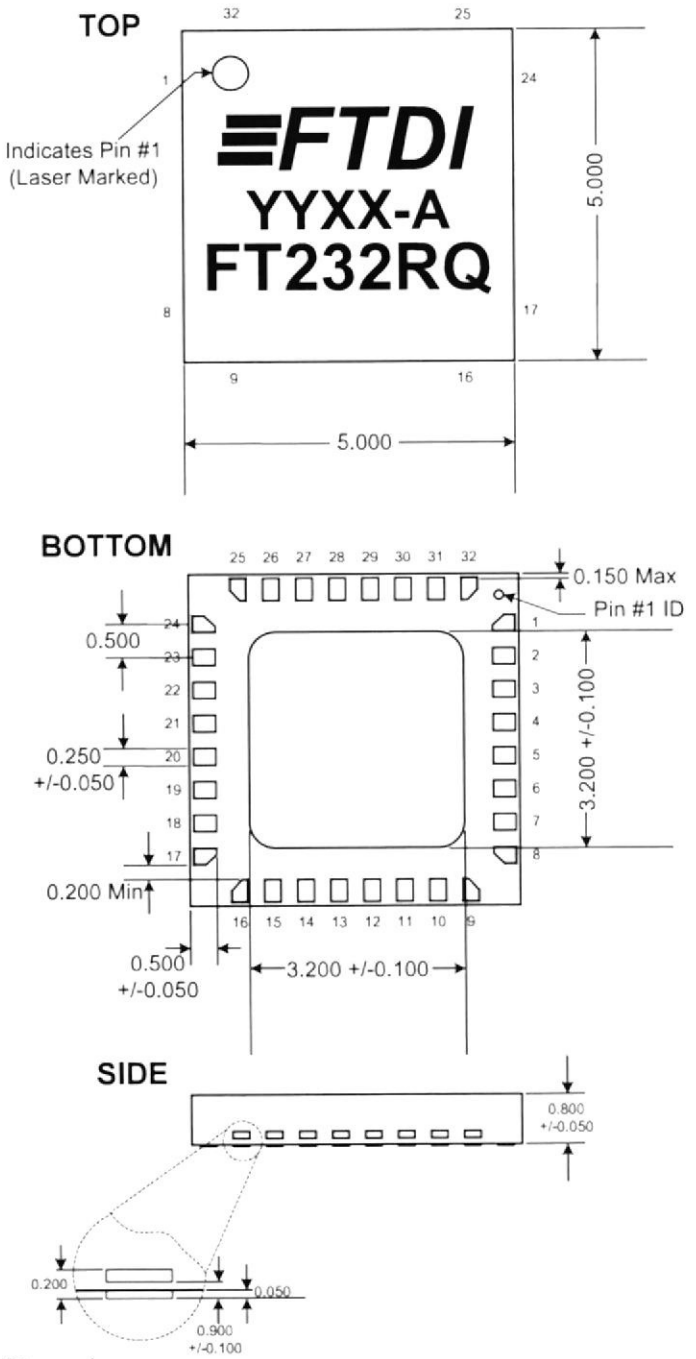


Figure 7 - QFN-32 Package Dimensions

The FT232RQ is supplied in a RoHS compliant leadless QFN-32 package. The package is lead (Pb) free, and uses a 'green' compound. The package is fully compliant with European Union directive 2002/95/EC.

This package has a compact 5.00mm x 5.00mm body. The solder pads are on a 0.50mm pitch. The above mechanical drawing shows the QFN-32 package – all dimensions are in millimetres.

The centre pad on the base of the FT232RQ is not internally connected, and can be left unconnected, or connected to ground (recommended).

The date code format is **YYXX** where XX = 2 digit week number, YY = 2 digit year number.

5.3 QFN-32 Package Typical Pad Layout

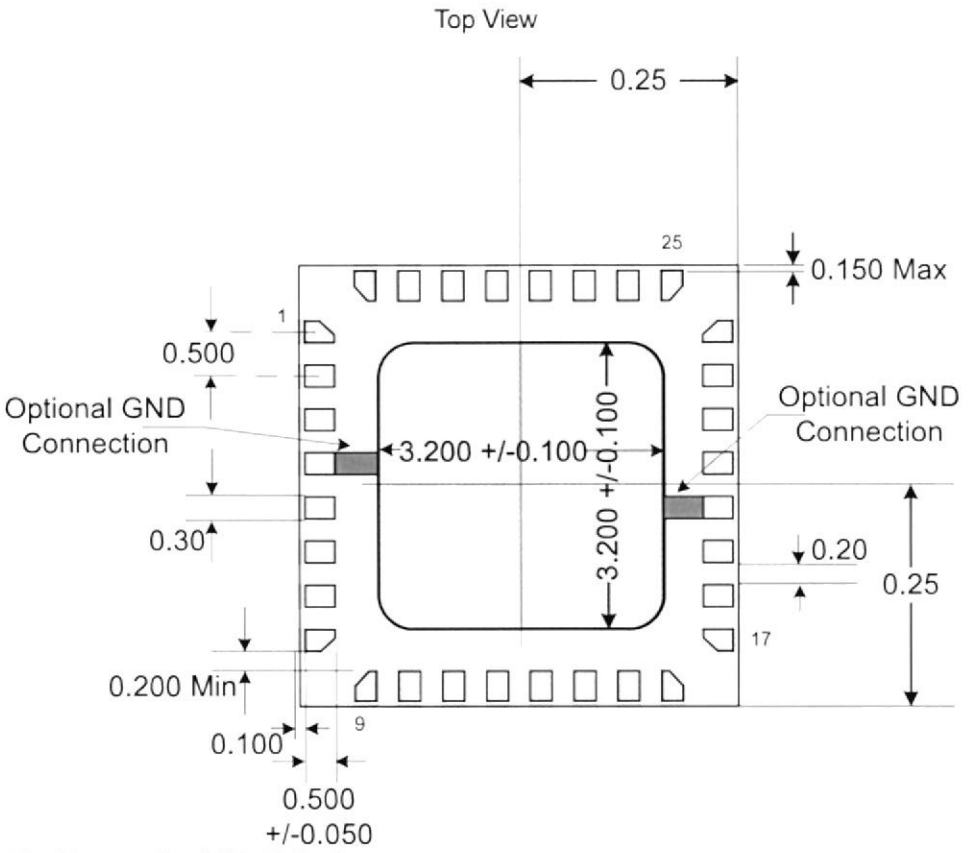


Figure 8 - Typical Pad Layout for QFN-32 Package

5.4 QFN-32 Package Typical Solder Paste Diagram

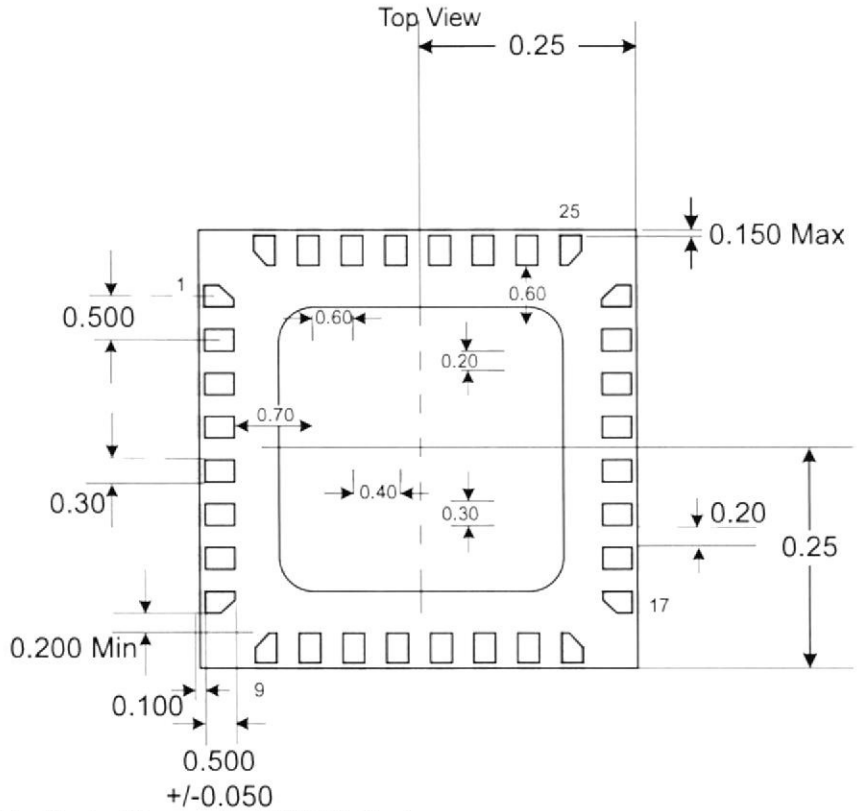


Figure 9 - Typical Solder Paste Diagram for QFN-32 Package

5.5 Solder Reflow Profile

The FT232R is supplied in Pb free 28 LD SSOP and QFN-32 packages. The recommended solder reflow profile for both package options is shown in Figure 10.

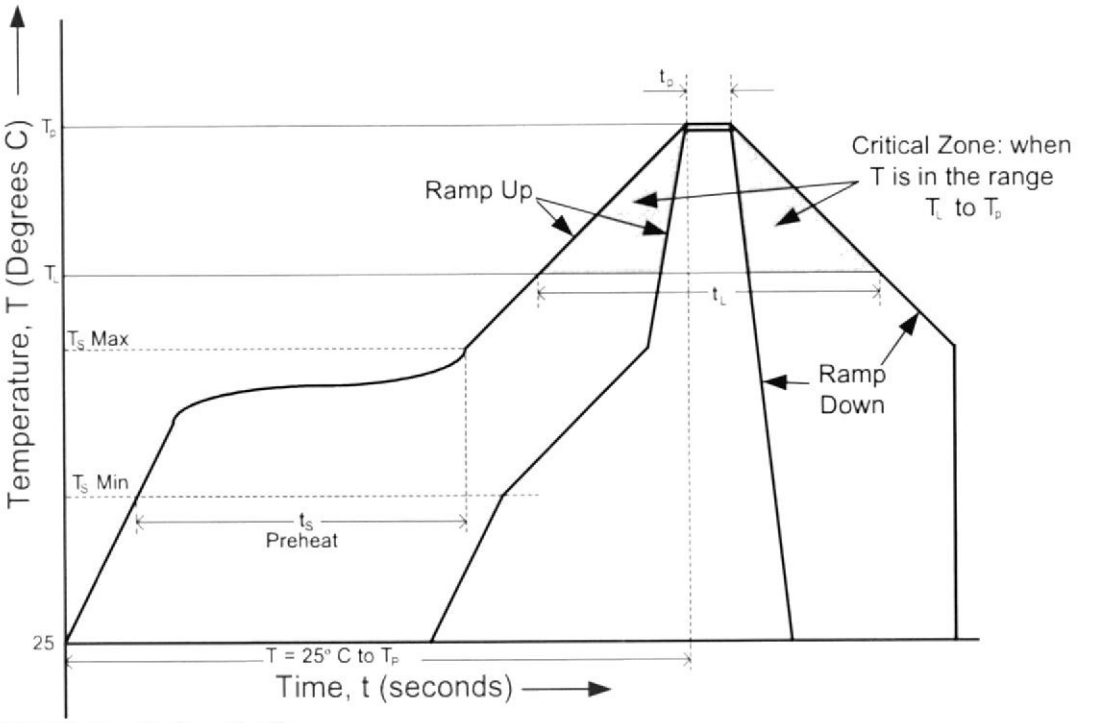


Figure 10 - FT232R Solder Reflow Profile

The recommended values for the solder reflow profile are detailed in Table 4. Values are shown for both a completely Pb free solder process (i.e. the FT232R is used with Pb free solder), and for a non-Pb free solder process (i.e. the FT232R is used with non-Pb free solder).

Table 4 - Reflow Profile Parameter Values

Profile Feature	*Pb Free Solder Process*	*Non-Pb Free Solder Process*
Average Ramp Up Rate (T_s to T_p)	3°C / second Max.	3°C / Second Max.
Preheat - Temperature Min (T_S Min.) - Temperature Max (T_S Max.) - Time (t_S Min to t_S Max)	 150°C 200°C 60 to 120 seconds	 100°C 150°C 60 to 120 seconds
Time Maintained Above Critical Temperature T_L: - Temperature (T_L) - Time (t_L)	 217°C 60 to 150 seconds	 183°C 60 to 150 seconds
Peak Temperature (T_p)	260°C	240°C
Time within 5°C of actual Peak Temperature (t_p)	20 to 40 seconds	20 to 40 seconds
Ramp Down Rate	6°C / second Max.	6°C / second Max.
Time for T= 25°C to Peak Temperature, T_p	8 minutes Max.	6 minutes Max.

6. Device Characteristics and Ratings

6.1 Absolute Maximum Ratings

The absolute maximum ratings for the FT232R devices are as follows. These are in accordance with the Absolute Maximum Rating System (IEC 60134). Exceeding these may cause permanent damage to the device.

Table 5 - Absolute Maximum Ratings

Parameter	Value	Unit
Storage Temperature	-65°C to 150°C	Degrees C
Floor Life (Out of Bag) At Factory Ambient (30°C / 60% Relative Humidity)	168 Hours (IPC/JEDEC J-STD-033A MSL Level 3 Compliant)*	Hours
Ambient Temperature (Power Applied)	-40°C to 85°C	Degrees C.
Vcc Supply Voltage	-0.5 to +6.00	V
D.C. Input Voltage - USBDP and USBDM	-0.5 to +3.8	V
D.C. Input Voltage - High Impedance Bidirectionals	-0.5 to +(Vcc +0.5)	V
D.C. Input Voltage - All other Inputs	-0.5 to +(Vcc +0.5)	V
D.C. Output Current - Outputs	24	mA
DC Output Current - Low Impedance Bidirectionals	24	mA
Power Dissipation (Vcc = 5.25V)	500	mW

* If devices are stored out of the packaging beyond this time limit the devices should be baked before use. The devices should be ramped up to a temperature of 125°C and baked for up to 17 hours.

6.2 DC Characteristics

DC Characteristics (Ambient Temperature = -40°C to +85°C)

Table 6 - Operating Voltage and Current

Parameter	Description	Min	Typ	Max	Units	Conditions
Vcc1	VCC Operating Supply Voltage	3.3	-	5.25	V	
Vcc2	VCCIO Operating Supply Voltage	1.8	-	5.25	V	
Icc1	Operating Supply Current	-	15	-	mA	Normal Operation
Icc2	Operating Supply Current	50	70	100	µA	USB Suspend

Table 7 - UART and CBUS I/O Pin Characteristics (VCCIO = 5.0V, Standard Drive Level)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	3.2	4.1	4.9	V	I source = 2mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 2mA
Vin	Input Switching Threshold	1.3	1.6	1.9	V	**
VHys	Input Switching Hysteresis	50	55	60	mV	**

Table 8 - UART and CBUS I/O Pin Characteristics (VCCIO = 3.3V, Standard Drive Level)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	2.2	2.7	3.2	V	I source = 1mA
Vol	Output Voltage Low	0.3	0.4	0.5	V	I sink = 2mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 9 - UART and CBUS I/O Pin Characteristics (VCCIO = 2.8V, Standard Drive Level)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	2.1	2.6	3.1	V	I source = 1mA
Vol	Output Voltage Low	0.3	0.4	0.5	V	I sink = 2mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 10 - UART and CBUS I/O Pin Characteristics (VCCIO = 5.0V, High Drive Level)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	3.2	4.1	4.9	V	I source = 6mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 6mA
Vin	Input Switching Threshold	1.3	1.6	1.9	V	**
VHys	Input Switching Hysteresis	50	55	60	mV	**

Table 11 - UART and CBUS I/O Pin Characteristics (VCCIO = 3.3V, High Drive Level)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	2.2	2.8	3.2	V	I source = 3mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 8mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

Table 12 - UART and CBUS I/O Pin Characteristics (VCCIO = 2.8V, High Drive Level)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	2.1	2.8	3.2	V	I source = 3mA
Vol	Output Voltage Low	0.3	0.4	0.6	V	I sink = 8mA
Vin	Input Switching Threshold	1.0	1.2	1.5	V	**
VHys	Input Switching Hysteresis	20	25	30	mV	**

**Inputs have an internal 200kΩ pull-up resistor to VCCIO.

Table 13 - RESET# and TEST Pin Characteristics

Parameter	Description	Min	Typ	Max	Units	Conditions
Vin	Input Switching Threshold	1.3	1.6	1.9	V	
VHys	Input Switching Hysteresis	50	55	60	mV	

Table 14 - USB I/O Pin (USBDP, USBDM) Characteristics

Parameter	Description	Min	Typ	Max	Units	Conditions
UVoh	I/O Pins Static Output (High)	2.8		3.6	V	RI = 1.5kΩ to 3V3Out (D+) RI = 15kΩ to GND (D-)
UVol	I/O Pins Static Output (Low)	0		0.3	V	RI = 1.5kΩ to 3V3Out (D+) RI = 15kΩ to GND (D-)
UVse	Single Ended Rx Threshold	0.8		2.0	V	
UCom	Differential Common Mode	0.8		2.5	V	
UVDif	Differential Input Sensitivity	0.2			V	
UDrvZ	Driver Output Impedance	26	29	44	Ohms	***

***Driver Output Impedance includes the internal USB series resistors on USBDP and USBDM pins.

6.3 EEPROM Reliability Characteristics

The internal 1024 Bit EEPROM has the following reliability characteristics-

Table 15 - EEPROM Characteristics

Parameter Description	Value	Unit
Data Retention	15	Years
Read / Write Cycles	100,000	Cycles

6.4 Internal Clock Characteristics

The internal Clock Oscillator has the following characteristics.

Table 16 - Internal Clock Characteristics

Parameter	Value			Unit
	Min	Typical	Max	
Frequency of Operation	11.98	12.00	12.02	MHz****
Clock Period	83.19	83.33	83.47	ns
Duty Cycle	45	50	55	%

****Equivalent to +/-1667ppm.

Table 17 - OSCI, OSCO Pin Characteristics (Optional - Only applies if external Oscillator is used*****)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	2.8	-	3.6	V	Fosc = 12MHz
Vol	Output Voltage Low	0.1	-	1.0	V	Fosc = 12MHz
Vin	Input Switching Threshold	1.8	2.5	3.2	V	

*****When supplied the device is configured to use its internal clock oscillator. Users who wish to use an external oscillator or crystal should contact FTDI technical support.

7. Device Configurations

Please note that pin numbers on the FT232R chip in this section have deliberately been left out as they vary between the FT232RL and FT232RQ versions of the device. All of these configurations apply to both package options for the FT232R device. Please refer to Section 4 for the package option pin-out and signal descriptions.

7.1 Bus Powered Configuration

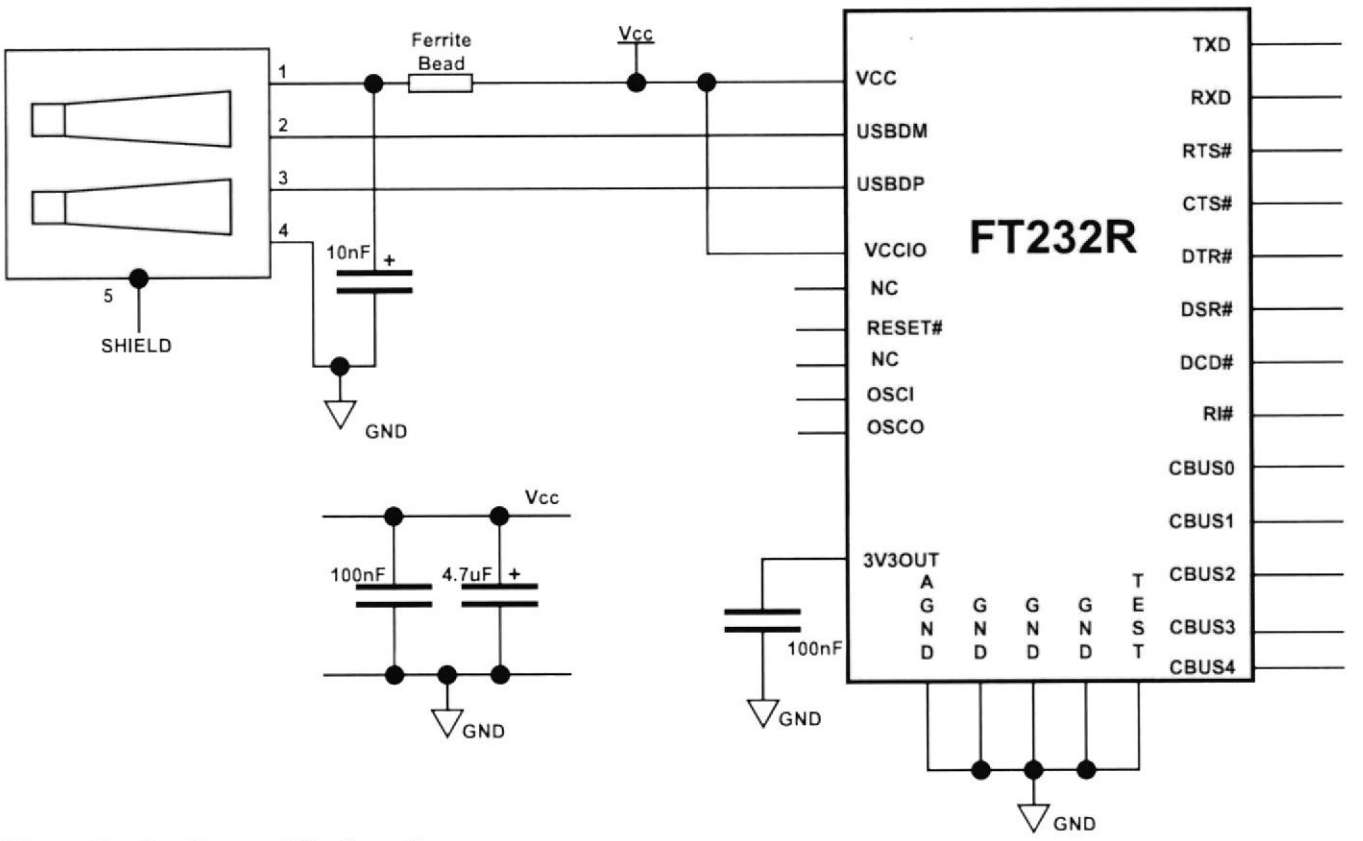


Figure 11 - Bus Powered Configuration

Figure 11 illustrates the FT232R in a typical USB bus powered design configuration. A USB Bus Powered device gets its power from the USB bus. Basic rules for USB Bus power devices are as follows –

i) On plug-in to USB, the device must draw no more than 100mA.

ii) On USB Suspend the device must draw no more than 500µA.

iii) A Bus Powered High Power USB Device (one that draws more than 100mA) should use one of the CBUS pins configured as PWREN# and use it to keep the current below 100mA on plug-in and 500µA on USB suspend.

iv) A device that consumes more than 100mA can not be plugged into a USB Bus Powered Hub.

v) No device can draw more that 500mA from the USB Bus.

The power descriptor in the internal EEPROM should be programmed to match the current draw of the device.
A Ferrite Bead is connected in series with USB power to prevent noise from the device and associated circuitry (EMI) being radiated down the USB cable to the Host. The value of the Ferrite Bead depends on the total current required by the circuit – a suitable range of Ferrite Beads is available from Steward (www.steward.com) for example Steward Part # MI0805K400R-00.

7.2 Self Powered Configuration

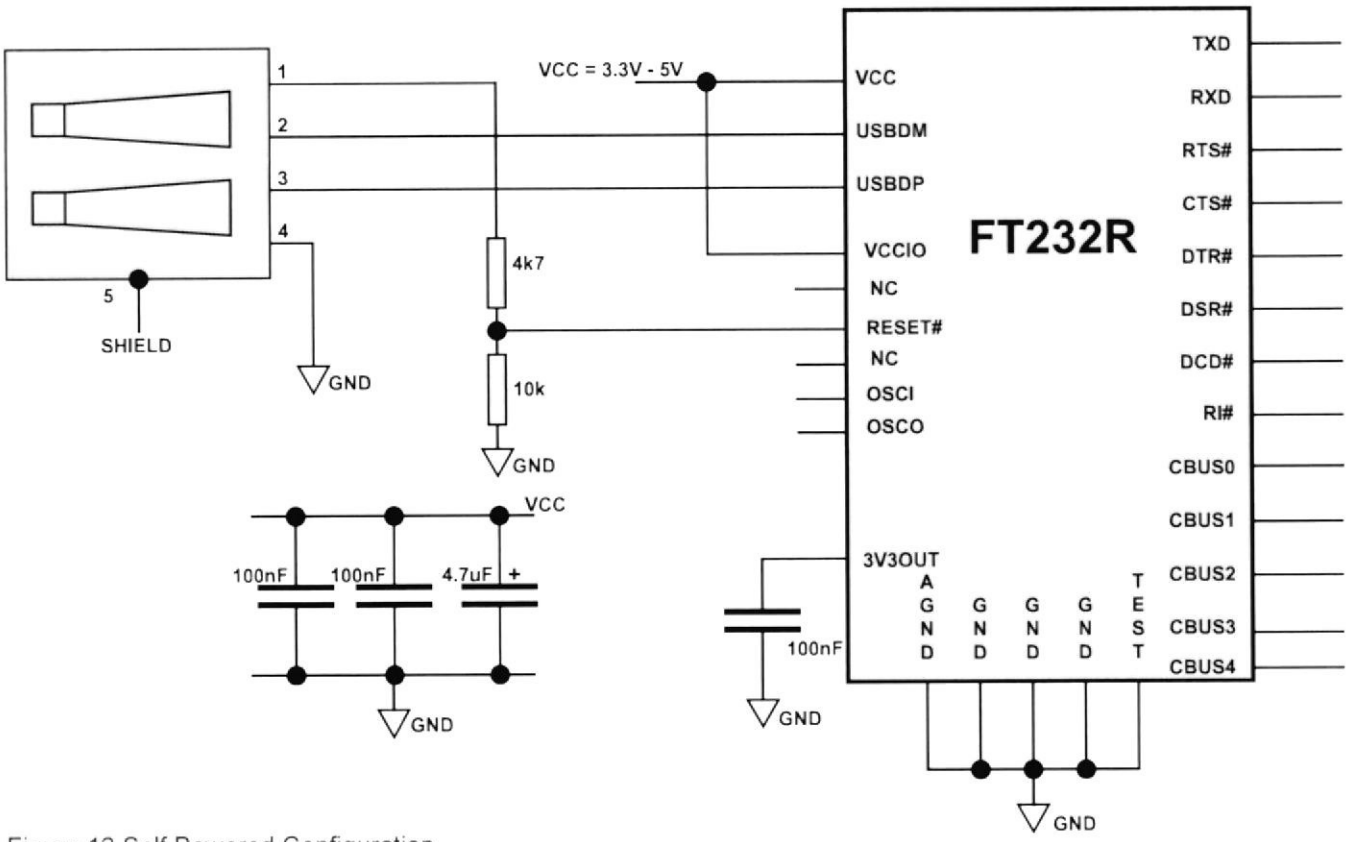


Figure 12 Self Powered Configuration

Figure 12 illustrates the FT232R in a typical USB self powered configuration. A USB Self Powered device gets its power from its own power supply and does not draw current from the USB bus. The basic rules for USB Self powered devices are as follows –

i) A Self Powered device should not force current down the USB bus when the USB Host or Hub Controller is powered down.

ii) A Self Powered Device can use as much current as it likes during normal operation and USB suspend as it has its own power supply.

iii) A Self Powered Device can be used with any USB Host and both Bus and Self Powered USB Hubs

The power descriptor in the internal EEPROM should be programmed to a value of zero (self powered).

In order to meet requirement (i) the USB Bus Power is used to control the RESET# Pin of the FT232R device. When the USB Host or Hub is powered up the internal 1.5kΩ resistor on USBDP is pulled up to 3.3V, thus identifying the device as a full speed device to USB. When the USB Host or Hub power is off, RESET# will go low and the device will be held in reset. As RESET# is low, the internal 1.5kΩ resistor will not be pulled up to 3.3V, so no current will be forced down USBDP via the 1.5kΩ pull-up resistor when the host or hub is powered down. Failure to do this may cause some USB host or hub controllers to power up erratically.

Figure 10 illustrates a self powered design which has a 3.3V - 5V supply. A design which is interfacing to 2.8V - 1.8V logic would have a 2.8V - 1.8V supply to VCCIO, and a 3.3V - 5V supply to VCC

Note : When the FT232R is in reset, the UART interface pins all go tri-state. These pins have internal 200kΩ pull-up resistors to VCCIO, so they will gently pull high unless driven by some external logic.

7.3 USB Bus Powered with Power Switching Configuration

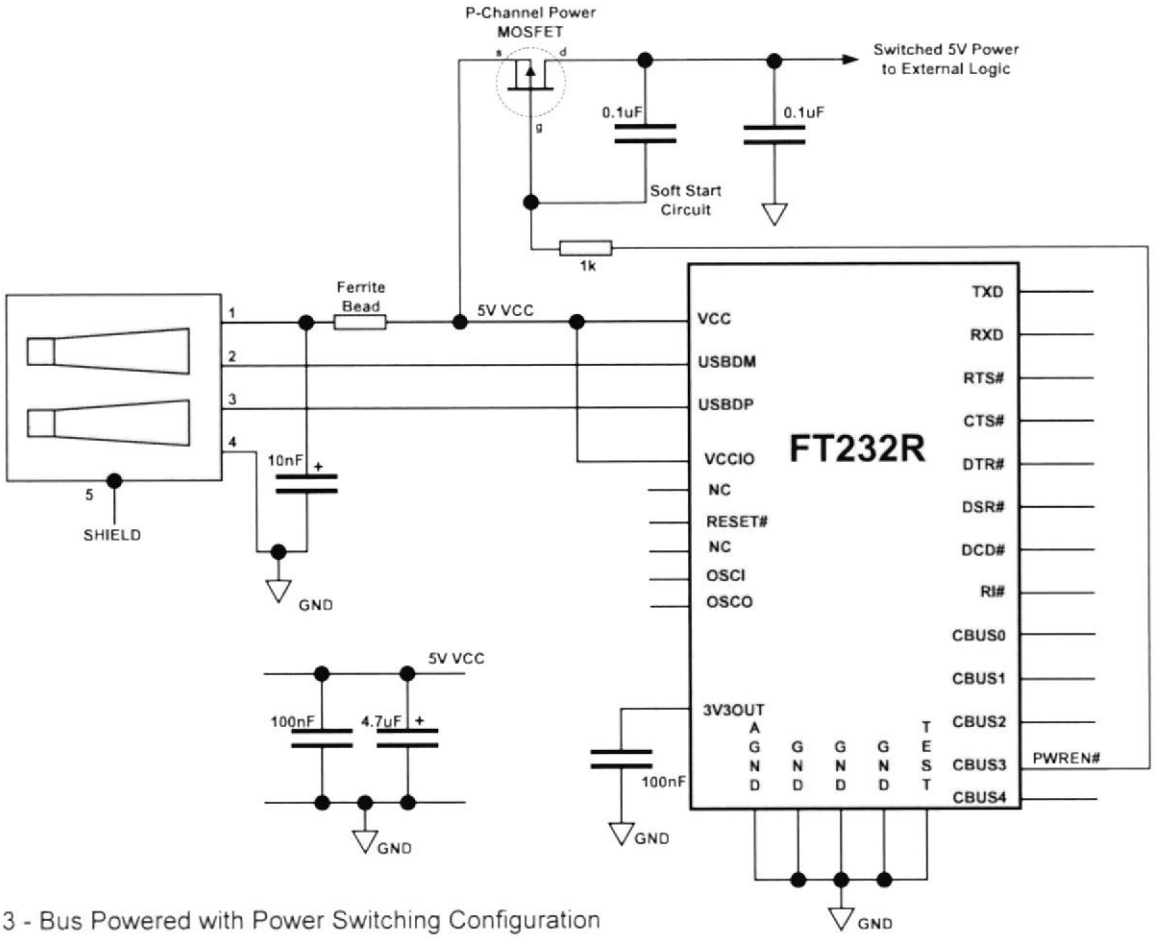


Figure 13 - Bus Powered with Power Switching Configuration

USB Bus powered circuits need to be able to power down in USB suspend mode in order to meet the <= 500µA total USB suspend current requirement (including external logic). Some external logic can power itself down into a low current state by monitoring the PWREN# signal. For external logic that cannot power itself down in this way, the FT232R provides a simple but effective way of turning off power to external circuitry during USB suspend.

Figure 13 shows how to use a discrete P-Channel Logic Level MOSFET to control the power to external logic circuits. A suitable device would be an International Rectifier (www.irf.com) IRLML6402, or equivalent. It is recommended that a "soft start" circuit consisting of a 1kΩ series resistor and a 0.1µF capacitor are used to limit the current surge when the MOSFET turns on. Without the soft start circuit there is a danger that the transient power surge of the MOSFET turning on will reset the FT232R, or the USB host / hub controller. The values used here allow attached circuitry to power up with a slew rate of ~12.5V per millisecond, in other words the output voltage will transition from GND to 5V in approximately 400 microseconds.

Alternatively, a dedicated power switch I.C. with inbuilt "soft-start" can be used instead of a MOSFET. A suitable power switch I.C. for such an application would be a Micrel (www.micrel.com) MIC2025-2BM or equivalent.

Please note the following points in connection with power controlled designs –

i) The logic to be controlled must have its own reset circuitry so that it will automatically reset itself when power is re-applied on coming out of suspend.
ii) Set the Pull-down on Suspend option in the internal EEPROM.
iii) One of the CBUS Pins should be configured as PWREN# in the internal EEPROM, and should be used to switch the power supply to the external circuitry..
iv) For USB high-power bus powered device (one that consumes greater than 100mA, and up to 500mA of current from the USB bus), the power consumption of the device should be set in the max power field in the internal EEPROM. A high-power bus powered device must use this descriptor in the internal EEPROM to inform the system of its power requirements.
v) For 3.3V power controlled circuits the VCCIO pin must not be powered down with the external circuitry (the PWREN# signal gets its VCC supply from VCCIO). Either connect the power switch between the output of the 3.3V regulator and the external 3.3V logic or power VCCIO from the 3V3OUT pin of the FT232R.

7.4 USB Bus Powered with 3.3V / 5V Supply and Logic Drive / IO Supply Voltage

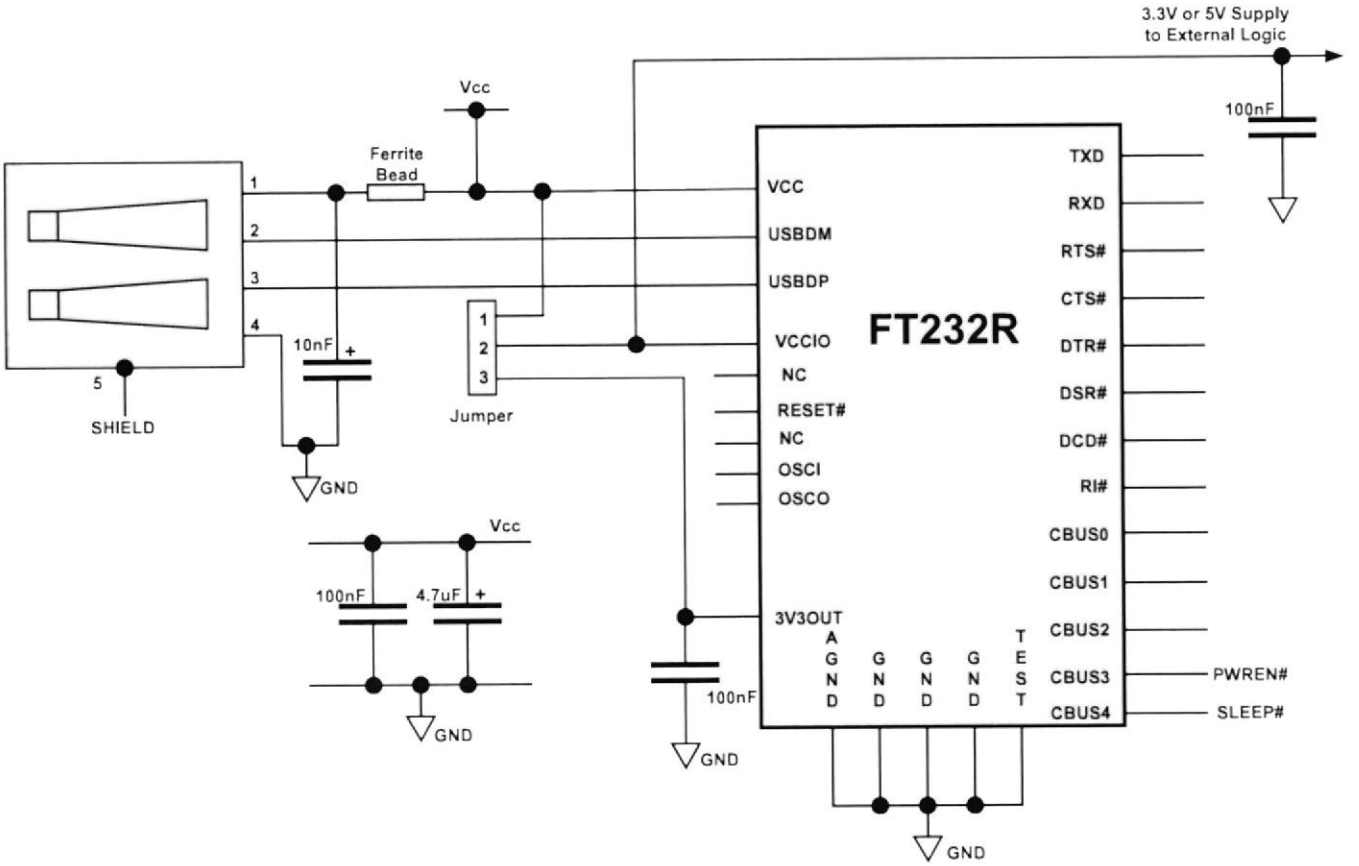


Figure 14 - Bus Powered with 3.3V / 5V Supply and Logic Drive

Figure 14 shows a configuration where a jumper switch is used to allow the FT232R to be interfaced with a 3.3V or 5V logic devices. The VCCIO pin is either supplied with 5V from the USB bus, or with 3.3V from the 3V3OUT pin. The supply to VCCIO is also used to supply external logic.

Please note the following in relation to bus powered designs of this type -

i) PWREN# or SLEEP# signals should be used to power down external logic during USB suspend mode, in order to comply with the limit of 500µA. If this is not possible, use the configuration shown in Section 7.3.

ii) The maximum current source from USB Bus during normal operation should not exceed 100mA, otherwise a bus powered design with power switching (Section 7.3) should be used.

Another possible configuration would be to use a discrete low dropout regulator which is supplied by the 5V on the USB bus to supply 2.8V - 1.8V to the VCCIO pin and to the external logic. VCC would be supplied with the 5V from the USB bus. With VCCIO connected to the output of the low dropout regulator, would in turn will cause the FT232R I/O pins to drive out at 2.8V - 1.8V logic levels.

For USB bus powered circuits some considerations have to be taken into account when selecting the regulator –

iii) The regulator must be capable of sustaining its output voltage with an input voltage of 4.35V. A Low Drop Out (L.D.O.) regulator must be selected.

iv) The quiescent current of the regulator must be low in order to meet the USB suspend total current requirement of <= 500µA during USB suspend.

An example of a regulator family that meets these requirements is the MicroChip / Telcom TC55 Series of devices (www.microchip.com). These devices can supply up to 250mA current and have a quiescent current of under 1µA.

8. Example Interface Configurations

As in the Device Configurations section, please note that pin numbers on the FT232R chip in this section have deliberately been left out as they vary between the FT232RL and FT232RQ versions of the device. All of these configurations apply to both package options for the FT232R device. Please refer to Section 4 for the package option pin-out and signal descriptions.

8.1 USB to RS232 Converter Configuration

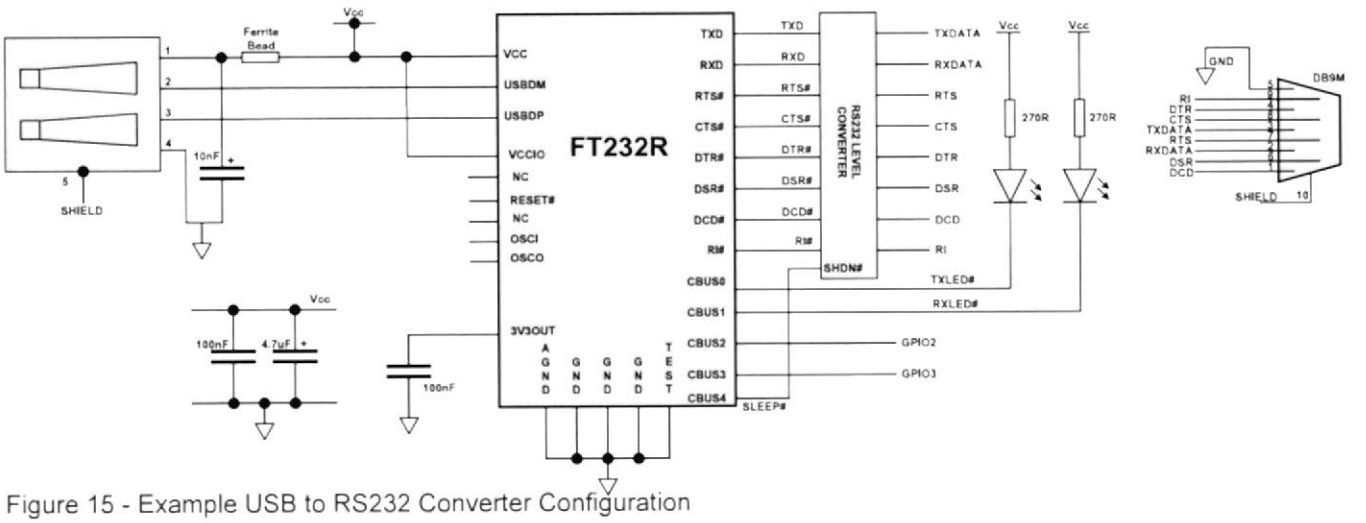


Figure 15 - Example USB to RS232 Converter Configuration

Figure 15 illustrates how to connect an FT232R as a USB to RS232 converter. A TTL – RS232 Level Converter I.C. is used on the serial UART of the FT232R to make the RS232 level conversion. This, for example can be done using the popular "213" series of TTL to RS232 level converters. These devices have 4 transmitters and 5 receivers in a 28-LD SSOP package and feature an in-built voltage converter to convert the 5V (nominal) VCC to the +/- 9 volts required by RS232. An important feature of these devices is the SHDN# pin which can power down the device to a low quiescent current during USB suspend mode.

An example of a device which can be used for this is a Sipex SP213EHCA which is capable of RS232 communication at up to 500kΩ baud. If a lower baud rate is acceptable, then several pin compatible alternatives are available such as the Sipex SP213ECA , the Maxim MAX213CAI and the Analog Devices ADM213E, which are all good for communication at up to 115,200 baud. If a higher baud rate is desired, use a Maxim MAX3245CAI part which is capable of RS232 communication at rates of up to 1M baud. The MAX3245 is not pin compatible with the 213 series devices, also its SHDN pin is active high, so connect it to PWREN# instead of SLEEP#.

In the above example CBUS0 and CBUS1 have been configured as TXLED# and RXLED#, and are being used to drive two LEDs.

8.2 USB to RS485 Converter Configuration

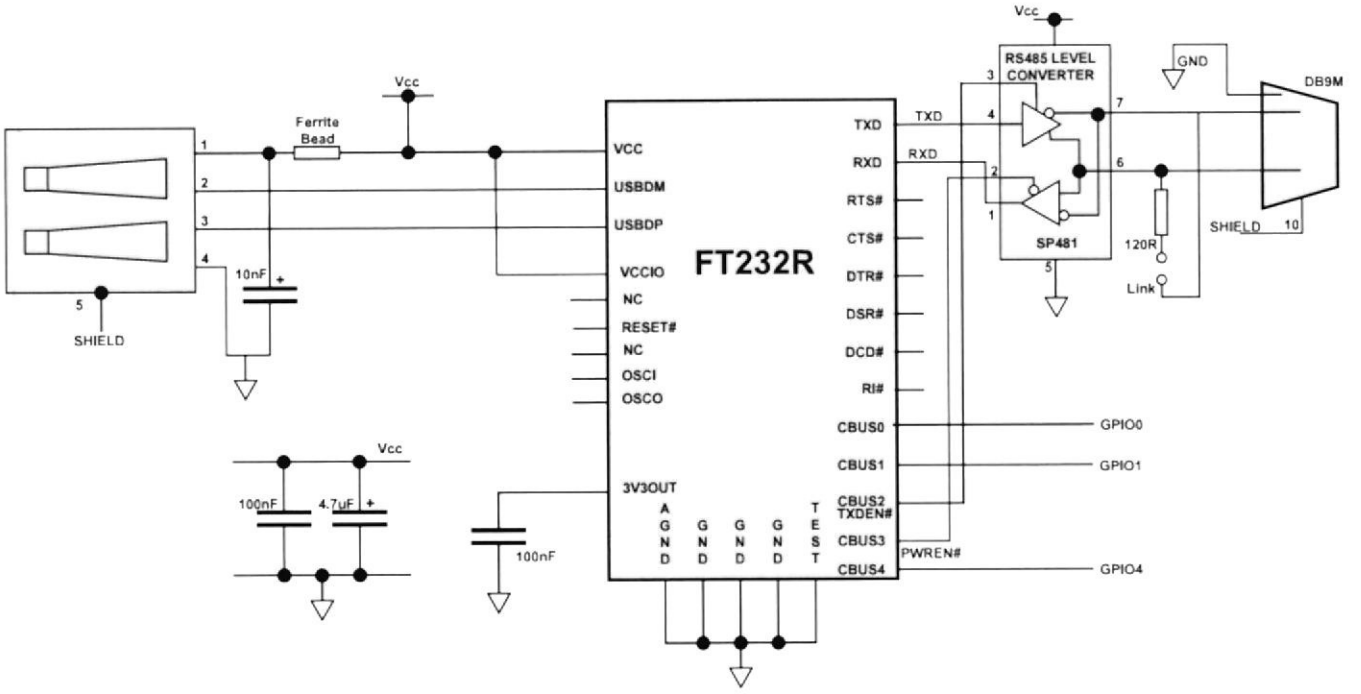


Figure 16 - Example USB to RS485 Converter Configuration

Figure 16 illustrates how to connect the FT232R's UART interface to a TTL – RS485 Level Converter I.C. to make a USB to RS485 converter. This example uses the Sipex SP481 device but there are similar parts available from Maxim and Analog Devices amongst others. The SP481 is a RS485 device in a compact 8 pin SOP package. It has separate enables on both the transmitter and receiver. With RS485, the transmitter is only enabled when a character is being transmitted from the UART. The TXDEN signal CBUS pin option on the FT232R is provided for exactly this purpose and so the transmitter enable is wired to CBUS2 which has been configured as TXDEN. Similarly, CBUS3 has been configured as PWREN#. This signal is used to control the SP481's receiver enable. The receiver enable is active low, so it is wired to the PWREN# pin to disable the receiver when in USB suspend mode. CBUS2 = TXDEN and CBUS3 = PWREN# are the default device configurations of these pins. See Section 10.

RS485 is a multi-drop network – i.e. many devices can communicate with each other over a single two wire cable connection. The RS485 cable requires to be terminated at each end of the cable. A link is provided to allow the cable to be terminated if the device is physically positioned at either end of the cable.

In this example the data transmitted by the FT232R is also received by the device that is transmitting. This is a common feature of RS485 and requires the application software to remove the transmitted data from the received data stream. With the FT232R it is possible to do this entirely in hardware – simply modify the schematic so that RXD of the FT232R is the logical OR of the SP481 receiver output with TXDEN using an HC32 or similar logic gate.

8.3 USB to RS422 Converter Configuration

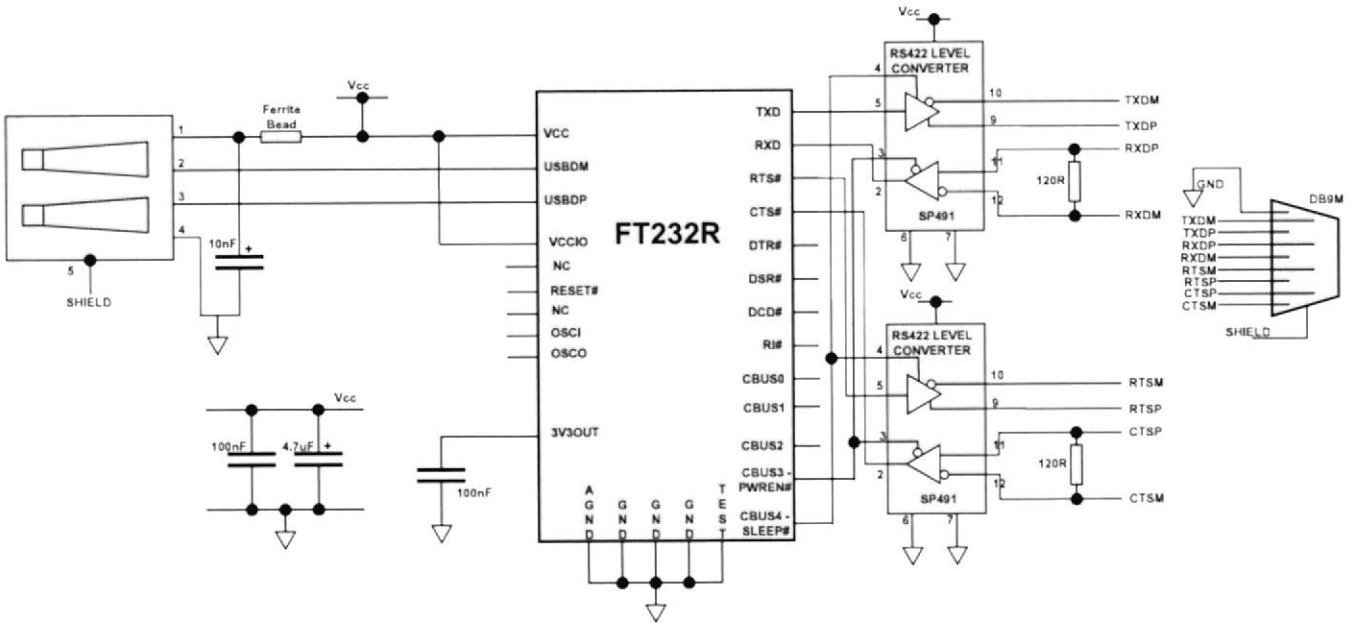


Figure 17 -Example USB to RS422 Converter Configuration

Figure 17 illustrates how to connect the UART interface of the FT232R to a TTL – RS422 Level Converter I.C. to make a USB to RS422 converter. There are many such level converter devices available – this example uses Sipex SP491 devices which have enables on both the transmitter and receiver. Because the transmitter enable is active high, it is connected to a CBUS pin in SLEEP# configuration. The receiver enable is active low and so is connected to a CBUS pin PWREN# configuration. This ensures that both the transmitters and receivers are enabled when the device is active, and disabled when the device is in USB suspend mode. If the design is USB BUS powered, it may be necessary to use a P-Channel logic level MOSFET (controlled by PWREN#) in the VCC line of the SP491 devices to ensure that the USB stand-by current of 500µA is met.

The SP491 is good for sending and receiving data at a rate of up to 5 Megbaud – in this case the maximum rate is limited to 3 Megabaud by the FT232R.

8.4 USB to MCU UART Interface

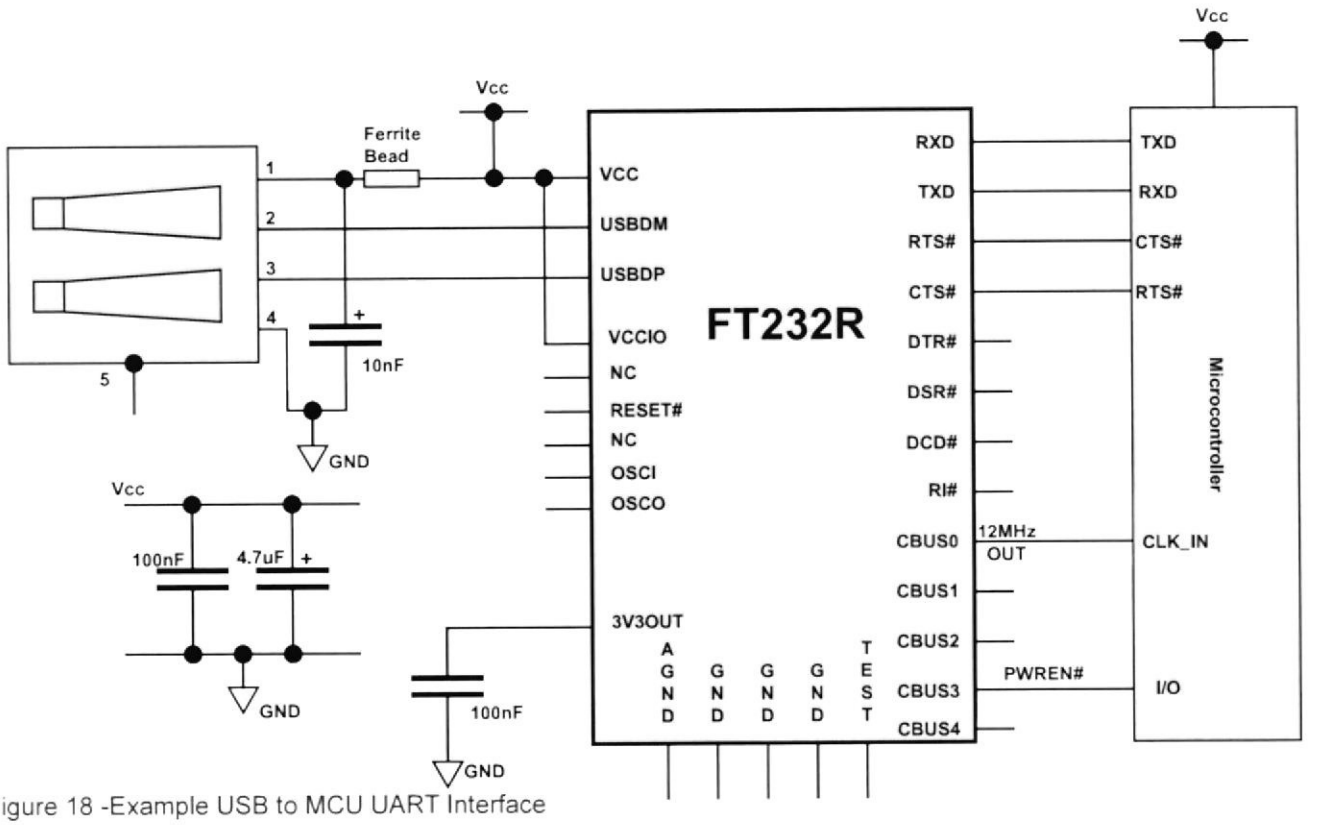


Figure 18 -Example USB to MCU UART Interface

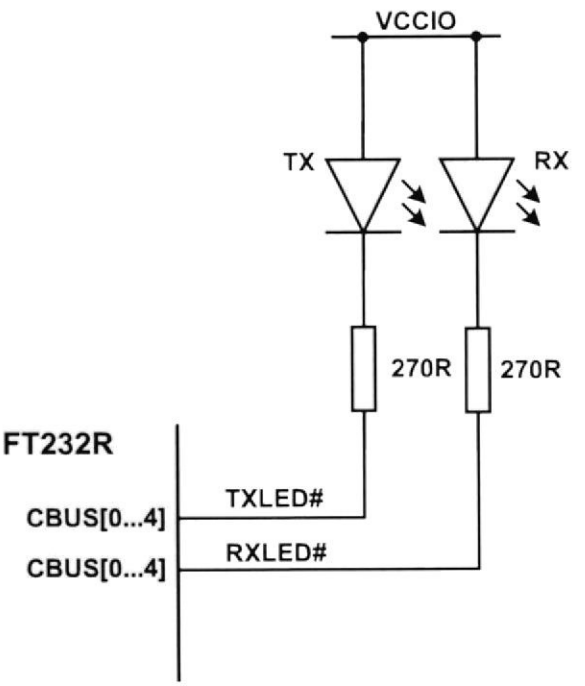
Figure 18 is an example of interfacing the FT232R to a Microcontroller (MCU) UART interface. This example uses TXD and RXD for transmission and reception of data, and RTS# / CTS# hardware handshaking. Also in this example CBUS0 has been configured as a 12MHz output which is being used to clock the MCU.

Optionally, RI# can be connected to another I/O pin on the MCU and could be used to wake up the USB host controller from suspend mode. If the MCU is handling power management functions, then a CBUS pin can be configured as PWREN# and should also be connected to an I/O pin of the MCU.

9. LED Interface

Any of the 5 CBUS I/O pins can be configured to drive an LED. The FT232R has 3 options for driving an LED - these are TXLED#, RXLED#, and TX&RXLED#.

Figure 19 -Dual LED Configuration



ure 19 illustrates the configuration where one pin is used to indicate transmission of data (TXLED#) and another used to indicate receiving data (RXLED#). When data is being transmitted or received the respective pins will drive om tri-state to low in order to provide indication on the LEDs of data transfer. A digital one-shot time is used so that ven a small percentage of data transfer is visible to the end user.

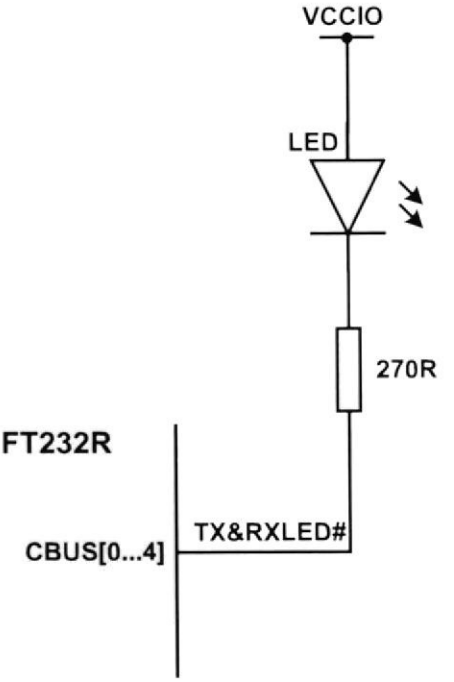


Figure 20 -Single LED Configuration

In figure 20 the TX&RXLED CBUS option is used. This option will cause the pin to drive a single LED when data is being transmitted or received by the device.

10. Internal EEPROM Configuration

Following a power-on reset or a USB reset the FT232R will scan its internal EEPROM and read the USB configuration descriptors stored there. The default values programmed into the internal EEPROM in a brand new device are defined in Table 18.

Table 18 - Default Internal EEPROM Configuration

Parameter	Value	Notes
USB Vendor ID (VID)	0403h	FTDI default VID (hex)
USB Product ID (PID)	6001h	FTDI default PID (hex)
Serial Number Enabled?	Yes	
Serial Number	*See Note*	A unique serial number is generated and programmed into the EEPROM during device final test.
Pull Down I/O Pins in USB Suspend	Disabled	Enabling this option will make the device pull down on the UART interface lines when the power is shut off (PWREN# is high)
Manufacturer Name	FTDI	
Manufacturer ID	FT	Serial number prefix
Product Description	FT232R USB UART	
Max Bus Power Current	90mA	
Power Source	Bus Powered	
Device Type	FT232R	
USB Version	0200	Returns USB 2.0 device descriptor to the host. Note: The device is be a USB 2.0 Full Speed device (12Mb/s) as opposed to a USB 2.0 High Speed device (480Mb/s).
Remote Wake up	Enabled	Taking RI# low will wake up the USB host controller from suspend.
High Current I/Os	Disabled	Enables the high drive level on the UART and CBUS I/O pins
Load VCP Driver	Enabled	Makes the device load the VCP driver interface for the device.
CBUS0	TXLED#	Default configuration of CBUS0 - Transmit LED drive
CBUS1	RXLED#	Default configuration of CBUS1 - Receive LED drive
CBUS2	TXDEN	Default configuration of CBUS2 - Transmit data enable for RS485
CBUS3	PWREN#	Default configuration of CBUS3 - Power enable. Low after USB enumeration, high during USB suspend.
CBUS4	SLEEP#	Default configuration of CBUS4 - Low during USB suspend.
Invert TXD	Disabled	Signal on this pin becomes TXD# if enabled.
Invert RXD	Disabled	Signal on this pin becomes RXD# if enabled.
Invert RTS#	Disabled	Signal on this pin becomes RTS if enabled.
Invert CTS#	Disabled	Signal on this pin becomes CTS if enabled.
Invert DTR#	Disabled	Signal on this pin becomes DTR if enabled.
Invert DSR#	Disabled	Signal on this pin becomes DSR if enabled.
Invert DCD#	Disabled	Signal on this pin becomes DCD if enabled.
Invert RI#	Disabled	Signal on this pin becomes RI if enabled.

The internal EEPROM in the FT232R can be programmed over USB using the utility program MPROG. MPROG can be downloaded from the FTDI website. Version 2.8a or later is required for the FT232R chip. Users who do not have their own USB Vendor ID but who would like to use a unique Product ID in their design can apply to FTDI for a free block of unique PIDs. Contact FTDI support for this service.

Disclaimer

Copyright © Future Technology Devices International Limited , 2005.

Version 0.90 - Initial Datasheet Created August 2005
Version 0.96 - Revised Pre-release datasheet October 2005
Version 1.00 - Full datasheet released December 2005
Version 1.02 - Minor revisions to datasheet 7th December 2005
Version 1.03 - 9th January 2006 - Manufacturer ID added to default EEPROM configuration; Buffer sizes added.
Version 1.04 - 30th January 2006 - QFN-32 Pad layout and solder paste diagrams added.

This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied.

Future Technology Devices International Ltd. will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected.

This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury.

This document provides preliminary information that may be subject to change without notice.

Contact FTDI

Head Office -
Future Technology Devices International Ltd.
373 Scotland Street,
Glasgow G5 8QB,
United Kingdom

Tel. : +(44) 141 429 2777
Fax. : +(44) 141 429 2758

E-Mail (Sales) : **sales1@ftdichip.com**
E-Mail (Support) : **support1@ftdichip.com**
E-Mail (General Enquiries) : **admin1@ftdichip.com**

Regional Sales Offices -

Future Technology Devices International Ltd. (Taiwan)
4F, No 16-1,
Sec. 6 Mincyuan East Road,
Neihu District,
Taipei 114,
Taiwan, R.o.C.

Tel.: +886 2 8791 3570
Fax: +886 2 8791 3576

E-Mail (Sales): **tw.sales@ftdichip.com**
E-Mail (Support): **tw.support@ftdichip.com**
E-Mail (General Enquiries): **tw.admin@ftdichip.com**

Future Technology Devices International Ltd. (USA)
5285 NE Elam Young
Parkway, Suite B800
Hillsboro,
OR 97124-6499
USA

Tel.: +1 (503) 547-0988
Fax: +1 (503) 547-0987

E-Mail (Sales): **us.sales@ftdichip.com**
E-Mail (Support): **us.support@ftdichip.com**
E-Mail (General Enquiries): **us.admin@ftdichip.com**

Website URL : **http://www.ftdichip.com**