UNIVERSIDAD TECNICA FEDERICO SANTA MARIA
DEPARTAMENTO DE ELECTRONICA

"Plataforma de desarrollo para Microcontroladores"

Memoria es presentada por
Michael Martin Kusch von Bischhoffshausen
como requisito parcial para optar al título de
Ingeniero Civil Electrónico
Mención Computadores

Profesor Guía: Leopoldo Silva Bijit

11 de Enero de 2006

A mis padres, quienes siempre me dieron la libertad y el apoyo para convertir mi pasión en mi profesión.

Resumen

En este trabajo de título se desarrollarán pequeños circuitos impresos que permitan utilizar chips modernos que debido su pequeño tamaño no pueden ser conectados directamente a un protoboard. Aprovechando la necesidad de tener que construir circuitos impresos sobre los cuales soldar estos chips, se incluirán también en el diseño todas las componentes mínimas necesarias para el funcionamiento de éstos. En conjunto, estos módulos proveerán una plataforma flexible para desarrollar equipos entorno a microcontroladores y para su utilización didáctica en laboratorios.

Esta memoria nace por la limitación actual que existe para incorporar los nuevos circuitos integrados que son cada vez más pequeños en el desarrollo de aplicaciones modernas. La experiencia obtenida del seminario de computadores del primer semestre de 2004 titulado "Diseño con microcontroladores", demostró que los Kits de desarrollo que existen actualmente en el mercado sólo tienen utilidad como herramienta didáctica. Éstos incluyen muchos tipos de interfaces y circuitería ya soldados y listos para utilizarse lo que es una ventaja en el proceso didáctico, pero se transforman en un inconveniente y una limitación a la hora de querer desarrollar un equipo.

La metodología de trabajo comienza con el análisis de las interfaces que poseen los Kits de desarrollo disponibles en el mercado y evaluar cuáles pueden ser armados en protoboard y cuáles requieren de estar soldados a un circuito impreso. Luego se evaluaron qué interfaces serían interesantes de incorporar. Finalmente se diseñó cada módulo independientemente, comenzando por el estudio del respectivo datasheet y circuitos postulados por el fabricante, para continuar con el análisis comparativo de circuitos en internet. Luego se elaboró un diseño esquemático y el posterior diseño y construcción del circuito impreso. Finalmente, cada módulo fue probado, para lo cuál se diseño un programa simple que permitiera una operación básica.

La contribución del autor de este trabajo de título consistió en haber dotado al departamento de electrónica de una plataforma de desarrollo construida en el mismo departamento. Quedó así demostrado que no es necesario comprar tarjetas de desarrollo en el extranjero y que el departamento posee las herramientas necesarias para crear sus propias tarjetas a un costo menor o comparable al de importación. La plataforma de desarrollo creada permite la incorporación de microcontroladores en otros laboratorios cuyas unidades de procesamiento han estado clásicamente reservadas a computadores o circuitería análoga. Finalmente esta plataforma no solo permite una utilización didáctica, sino que abre la puerta para desarrollar equipos con fines comerciales, que puedan incorporar tecnologías como internet y dispositivos USB.

Palabras clave: Kit de desarrollo, microcontrolador, uC, USB Host, LAN, MP3, MSP430, STA013, CS8900.

INDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

Introducción

Para nadie es desconocido el gran auge que ha tenido la computación en estos últimos años. Cada vez existen más equipos y electrodomésticos que incorporan microprocesadores y microcontroladores, aumentando sus prestaciones y funcionalidades. Podemos encontrar computadores en un sinnúmero de equipos. Desde refrigeradores que detectan cuando falta leche o mantequilla y la piden automáticamente por internet, hasta tostadoras que utilizan internet para obtener el pronóstico del tiempo y tatuar un sol o una nube en el pan, la cantidad de equipos que se comunican con internet sigue en aumento.

A estos equipos que incorporan procesadores, pero que no son computadores de escritorio, se les ha llamado sistemas embebidos. Un sistema embebido es un computador que está dentro de otro equipo. Computadores embebidos son los que se encuentran en los automóviles, celulares, agendas electrónicas, lavadoras, etc. A diferencia de computadores de escritorio, un computador embebido sólo debe ser capaz de correr una aplicación o un puñado de aplicaciones, las cuales han sido diseñadas e integradas en conjunto con el hardware. Por esto el enfoque no está en tener una gran capacidad de procesamiento, sino en disminuir los costos de producción del equipo. Normalmente, el usuario de un sistema embebido ni siquiera se percata que dentro del equipo existe un pequeño computador. El crecimiento de computadores embebidos es de alrededor del 40% por año, lo que es muy superior al cerca de 7% anual de los computadores personales y servidores.

Una herramienta clásica para hacer prototipos electrónicos es el llamado *protoboard*. Por años esta herramienta ha permitido el interconexionado de componentes electrónicas como circuitos integrados o *chips*, resistencias, condensadores, etc. sin la necesidad de diseñar y construir un circuito impreso, y sin tener que soldar las componentes a un circuito impreso, permitiendo una rápida construcción y modificación de prototipos, además de la reutilización de las componentes. Sin embargo, el alto grado de miniaturización, junto con el aumento en la frecuencia de operación de los nuevos protocolos de transferencia de datos, están haciendo obsoleto el uso del clásico protoboard.

Para poder incluir componentes modernas en el diseño de nuevos equipos, es necesario diseñar y construir circuitos impresos o PCBs (del inglés Printed Circuit Board) que contengan la interconexión específica que requiere el equipo en construcción. Para poder hacer un PCB, primero deben hacerse las fotoherramientas. Debe fabricarse una fotoherramienta por cada “capa” que tenga el PCB, esto incluye las pistas de cobre, la máscara y la guía de componentes de cada cara, además de la guía de agujeros. Cada fotoherramienta puede tener un costo de un 80% del costo total de fabricación del circuito impreso. De esta forma, para hacer un primer circuito impreso típico de 2 caras, se requieren 6 o 7 fotoherramientas, sumando un costo que puede llegar a ser 5 o 6 veces superior al de fabricación del circuito impreso en sí. Esto redunda en un alto costo para la fabricación del primer prototipo. Además, cualquier cambio que deba hacerse al conexionado entre componentes implica tener que rehacer las fotoherramientas y comprar todas las componentes nuevamente, ya que las componentes ya soldadas rara vez pueden ser reutilizadas.

Por otro lado, debido a la alta densidad de pines de las nuevas componentes, el soldado de éstas requiere de cautines y otras herramientas profesionales que por su alto costo, no siempre son asequibles a universidades y al emprendedor que está comenzando su propia empresa.

La logística en relación a la obtención de componentes modernas es un problema adicional. Los distribuidores nacionales de venta directa tienen una muy baja selección de componentes, lo cuál junto con el bajo stock que poseen, los convierte en una alternativa poco confiable. Por otro lado se encuentran los representantes de distribuidores extranjeros como RS y Farnell. Estos tienen una gran variedad de componentes, pero cada componente es importada al momento que el cliente la ordena, debiendo esperar de 7 a 10 días hasta su arribo. Además, sus precios suelen ser bastante altos, y para el caso de componentes simples, excesivos. Lo anterior se resume en una dificultad adicional que debe ser tomada en cuenta durante el proceso de diseño del hardware.

Objetivos

El objetivo de esta memoria es diseñar y construir una plataforma de desarrollo para microcontroladores que sea flexible, permitiendo no sólo la introducción de microcontroladores modernos a los ramos dictados en las universidades, sino que también permita y facilite la creación de equipos y productos, ya sea en el marco de un proyecto para un ramo universitario, o un prototipo comercial en pequeñas y medianas empresas.

La motivación personal tras esta memoria es dejar un legado tras mi estadía en esta universidad. Un aporte a las nuevas generaciones de electrónicos que comienzan su etapa universitaria, otorgándoles una herramienta útil para el desarrollo de sus propios proyectos e ideas.

1 Análisis de plataformas actuales

Actualmente existen en el mercado varias tarjetas de desarrollo para microcontroladores, las cuales incorporan varios circuitos externos al microcontrolador. El departamento de electrónica de la UTFSM adquirió 8 Kits de desarrollo *EasyWeb II* como el de la Figura 1 y que fueron utilizados durante el seminario de computadores "Diseño con Microcontroladores", dictado el 1er semestre de 2004. De esta experiencia se sacaron conclusiones positivas y negativas sobre estos Kits.

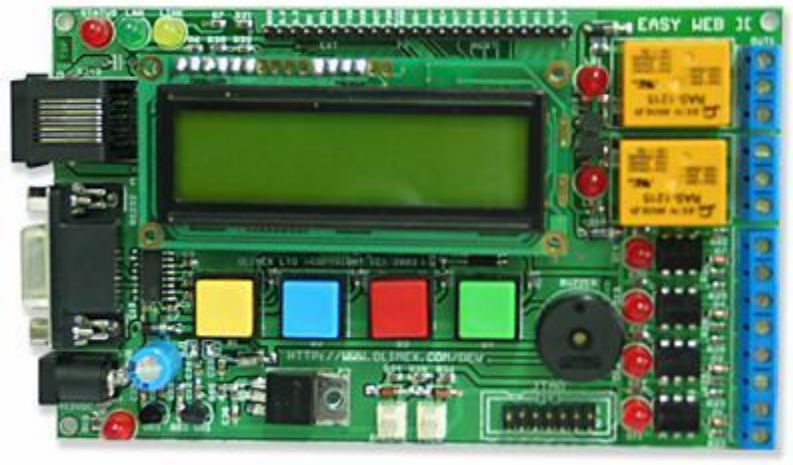


Figura 1. Tarjeta EasyWeb II

Características relevantes de la tarjeta EasyWeb II:

- Interfaz JTAG para programación/emulación.
- Alimentación desde una fuente simple de 9 a 12 Volts.
- Display LCD inteligente de 2 filas por 16 caracteres alfanuméricos.
- 4 pulsadores.
- 2 Relés de salida
- 4 optoacopladores de entrada
- Disponibles todos los pines análogos en una regleta de conexión.
- Interfaz RS-232.
- Interfaz Ethernet 10BaseT.
- Salidas de 5V, 3.6V y 3.3V.

La mayor ventaja de este tipo de Kits consiste en tener botones, relés, optoacopladores, un display LCD de 2x16, una interfaz serial RS-232 y una interfaz ethernet, todo ya cableado al microcontrolador, pudiendo rápidamente experimentar con los recursos periféricos del microcontrolador sin tener que lidiar con hardware externo ni problemas de cableado.

Lo que en un principio resultó ser una ventaja, al llegar al proyecto final se transformó en una restricción. El hecho de tener todos los puertos, a excepción del puerto 6, conectados por hardware a las componentes electrónicas externas, inutilizó los recursos periféricos

conectados a estos pines. A continuación se muestran algunos ejemplos de las dificultades encontradas:

- Sólo el puerto 6 está disponible directamente a través de una regleta de conexión.
- El Kit sólo posee un cristal de 8MHz y no posee acceso a los pines para el 2º cristal. Así se imposibilita la conexión de un cristal de 32kHz para aplicaciones que requieren de un reloj de baja frecuencia, como son aplicaciones de bajo consumo.
- Los pines de entrada y salida del Timer A que se encuentran en el puerto 1 de la MSP se encuentran cableados a optoacopladores de entrada y relés de salida, imposibilitando la generación de señales PWM y la captura de (o interrupción por) señales de frecuencia alta.
- Los pines del puerto 2 están conectados al display LCD y a un LED. Por esta razón no puede utilizarse el comparador análogo del la MSP.
- Sólo puede utilizarse una de las interfaces USART y sólo en modo asincrónico y con niveles RS-232. No puede utilizarse ninguna de las 2 interfases SPI, ni el segundo puerto asincrónico, ya que todas estas señales están conectadas al chip de red.
- Sólo 2 salidas del Timer B están disponibles, lo que permite generar una sola PWM. Las otras 5 están conectadas a botones y al buzzer, lo que sólo permite la generación de una PWM por hardware, en vez de poder generar las 6 disponibles en el chip. Esto también implica que sólo puede utilizarse una entrada del módulo de captura, en vez de las 7 disponibles.

Con otros kits de desarrollo ocurren los mismos problemas, todo pin que ha sido cableado a una interfaz específica, no puede utilizarse para una función alternativa no cubierta por la electrónica a la que está conectado.

Debe entonces encontrarse una forma de conectar el hardware externo que posibilite un cambio rápido de conexión de un puerto a otro, además de dejar todos los pines disponibles para conexión externa.

La primera opción estudiada fue utilizar una FPGA ó una CPLD para hacer la conexión. Esto resultó ser inviable, puesto requeriría una inmensa cantidad de pines para poder habilitar todas las opciones de interconexión posibles. El costo de una FPGA de tal tamaño sería mayor que el resto del hardware a interconectar.

Una alternativa es hacer una gran tarjeta con todo el hardware separado en módulos, con todos sus pines disponibles mediante una regleta. El problema es que un mismo puerto del microcontrolador podría ser utilizado por más de un módulo, como por ejemplo un bus de datos. Para un proyecto específico, es difícil que se vayan a utilizar todos los módulos, lo que implica un gran desperdicio de recursos.

La solución propuesta es crear módulos independientes que puedan ser interconectados a través de un protoboard. Así puede cambiarse el puerto de conexión a determinado hardware en un abrir y cerrar de ojos. Además, se puede sacar un mayor provecho a los módulos, ya que difícilmente van a utilizarse todos los módulos para todos los proyectos. Entonces sólo se conectan al protoboard los módulos necesarios, y luego se hace la

interconexión entre ellos. Aquí el problema es la gran cantidad de cables necesarios para hacer la interconexión. Fue entonces necesario crear un sistema que facilite la conexión de buses de datos entre un módulo y otro.

Buses de datos

La primera opción para interconexión fue utilizar un cable plano y terminales tipo Header de 1 fila. Uno de los problemas de esta solución es que no existen conectores macho, teniendo que utilizar conectores hembra y regletas para circuito impreso. El otro problema es el soldado de los terminales, ya que los conductores de un cable plano son muy delicados y delgados. La cantidad de horas hombre que sería necesaria para confeccionar este tipo de cables es alta.

Finalmente se encontró un tipo especial de conectores macho para cable plano, llamados "*FCI Clincher Assembly*", distribuidos por Mouser.com. Éstos tienen una separación de pines de 0.1 pulgadas, que es la misma que utilizan los chips y protoboards. Además, la conexión al cable plano se hace por medio de terminales a presión. El único detalle es que como los cables planos tienen una separación de 0.05 pulgadas, el conector sólo utiliza la mitad de los conductores. Por ejemplo, un conector de 4 bits requiere un cable plano de 7 conductores: 4 para datos y 1 entre cada 2 de datos. Estos detalles pueden verse en la Figura 2.

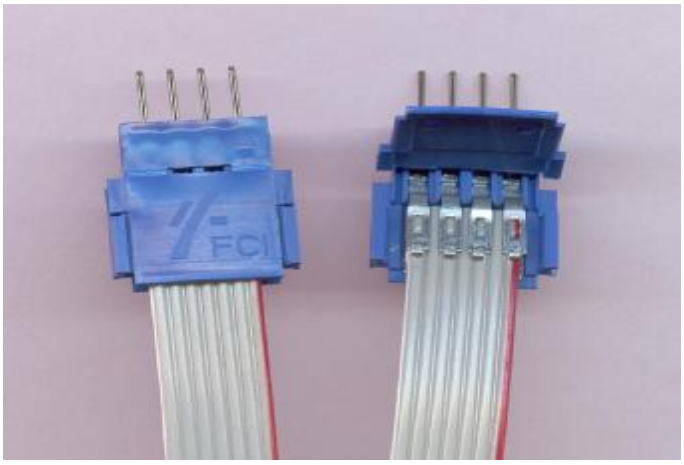


Figura 2. Bus de datos con conectores FCI y cable plano IDC.

Una vez solucionada la forma de interconexión para los módulos, se decidió qué módulos implementar. Lo primero fue crear un *Header* para la familia de microprocesadores MSP430 de 64 pines, que permitiera su conexión a un protoboard. La selección de esta familia específica de microcontroladores se debe a que son los mismos presentes en los Kits de desarrollo EasyWeb II, recientemente adquirido por el departamento. Además, debido a la estructura modular implementada, basta con cambiar el Header por otro microcontrolador más moderno o de otra familia, y pueden utilizarse todos los demás módulos diseñados. Luego de crear el Header MSP430 se diseñó un programador JTAG

para la conexión de la MSP con un entorno de desarrollo en el PC, el cuál además permite la emulación.

Los elementos que se consideraron para ser incluidos en esta plataforma de desarrollo se resumen a continuación:

- Header con microcontrolador
- Programador/emulador.
- Memoria de datos no volátil.
- Display LCD para la visualización de mensajes.
- Teclado numérico para la introducción de datos.
- Interfaz RS-232.
- Interfaz ethernet.
- Interfaz USB.
- Descompresor MP3.
- Memoria RAM externa.
- Relés.
- Optoacopladores.

2 Módulos Básicos: Header y Programador

2.1 Header MSP430

El problema con la familia de microcontroladores MSP430 de 64 pines radica en que sólo están disponibles en empaque QFP-64. Éste es un empaque cuadrado con pines por los 4 costados. Además, el ancho de los pines es de 10[*mil*](milésmias de pulgada) y el espaciamiento entre pines es también de 10[*mil*], quivalentes a 0.254[mm]. Esto hace imposible su conexión directa a un protoboard, lo que requiere de soldarlos sobre un circuito impreso adaptador, muchas veces llamados *Header*.

Junto con los Kits EasyWeb II, el departamento compró algunos Headers que traían el mismo microcontrolador MSP430F149 que los Kits, desarrollados por el mismo fabricante Olimex. Pero esos Headers no solucionan el problema, ya que son circuitos impresos cuadrados que poseen pines por los 4 costados. Si bien el espaciamiento entre pines es el mismo que utilizan los protoboards, no existen protoboards para placas con pines por los 4 costados. La función de esos Headers no es la conexión de una MSP a un protoboard, sino que solucionar el problema del soldado del chip. Utilizando el Header de Olimex, como el chip ya se encuentra en la placa, no es necesario disponer de elementos profesionales de soldado. Así, los requisitos de diseño y soldado de un circuito impreso propio son mucho menores, dados por los pines del Header que tienen un espaciamiento 9 veces mayor que el chip. Uno de estos Headers se muestra en la Figura 3.

Figura 3. Header MSP430 de Olimex.

Se comenzó entonces a finales de ese semestre (semestre 1 de 2004) a diseñar un circuito impreso que permitiera la conexión de una MSP de 64 pines a un protoboard. El interés y apoyo del profesor que dictó el ramo posibilitó la rápida construcción del primer prototipo de este Header. Aprovechando el hecho de que se estaba diseñando un Header propio, se agregaron características para hacerlo lo más útil y flexible posible. Los puntos considerados durante este diseño se detallan a continuación y dieron origen al circuito esquemático presentado en la figura 4.

- Tener absolutamente todos los pines del microcontrolador disponibles en el protoboard, aún los que normalmente no requieran conexión externa al Header como es el puerto JTAG y los pines para los cristales. Esto significa 32 pines a cada lado del header.
- Hacer el circuito impreso tan angosto como sea posible de manera de dejar la mayor cantidad de espacio en el protoboard para conexionado del resto del circuito.
- Incluir el circuito de interfaz para el programador/emulador JTAG, disponiendo de un conector especial. Utilizar un conector con llave para impedir una conexión inversa de forma accidental.
- El conector JTAG debe ser compatible con los de Olimex actualmente utilizados para sus Header y para la EasyWeb II.
- Incluir jumpers para seleccionar alimentación externa o desde el JTAG, para poder experimentar con el microprocesador sin la necesidad de conectar una fuente externa.
- No puentear la alimentación análoga a la alimentación digital en el Header, para dejar abierta la posibilidad de una fuente de poder de bajo ruido separada para la alimentación análoga.
- Utilizar jumpers para la interconexión de los pines de alimentación análogo y digital, para no tener que utilizar un protoboard para hacer esta interconexión al utilizar la fuente del JTAG para la alimentación.
- Disponer de un switch de reset para la utilización del Header en forma offline, es decir, al utilizar la MSP ya programada sin conexión al emulador del PC.
- Incluir condensadores de filtro para la alimentación.
- Incluir los cristales para los 2 circuitos de reloj de la MSP. Un cristal primario de máxima frecuencia (8MHz) para la ejecución de instrucciones a la mayor velocidad posible, y un cristal típico de 32.768Hz para aplicaciones de bajo consumo y que requieren de un reloj de tiempo real.
- Utilizar componentes de fácil adquisición, en lo posible *Thru-Hole*, para poder soldar las MSP en taller y que los alumnos puedan soldar el resto de las componentes.

La Figura 5 muestra el Header construido, montado en un protoboard. Puede apreciarse que sobra una corrida de pines a cada lado del Header, para la conexión de las componentes externas. Sin embargo, el espaciamiento también permite una conexión sobre el bus de poder entre 2 protoboards como muestra la figura 6. Esta es una forma mucho más conveniente de conexión, puesto que deja 4 corridas puntos de conexión disponibles. Para probar este Header, se utilizó uno de los programadores JTAG que habían sido adquiridos junto con las EasyWeb II.

La tabla 1 contiene la lista de componentes del Header MSP430.

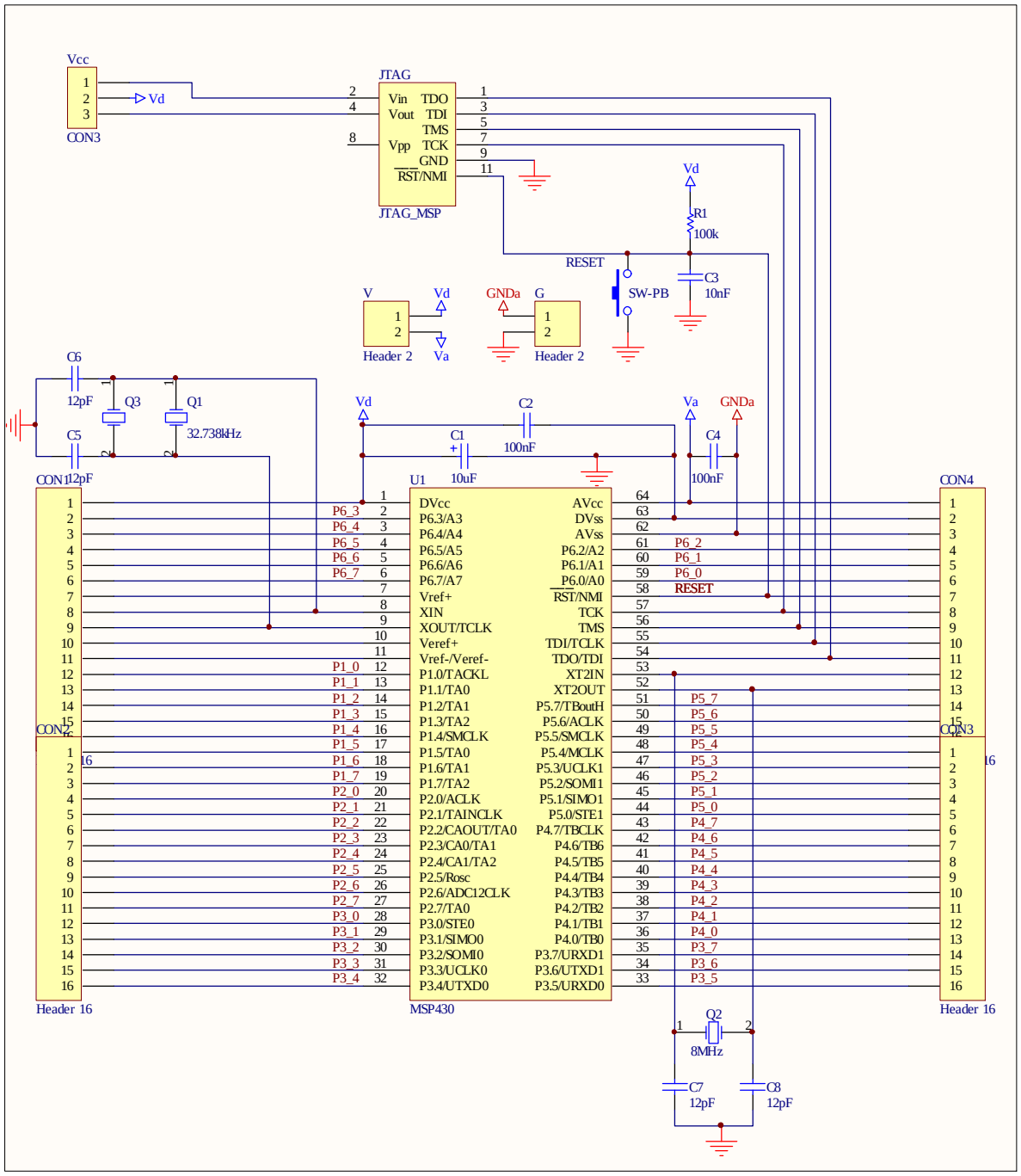


Figura 4. Circuito esquemático Header MSP430.

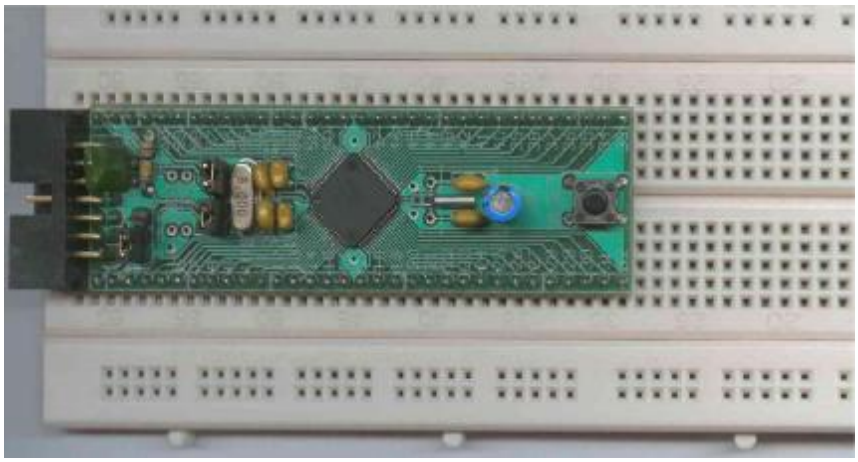


Figura 5. Header MSP430 construido.

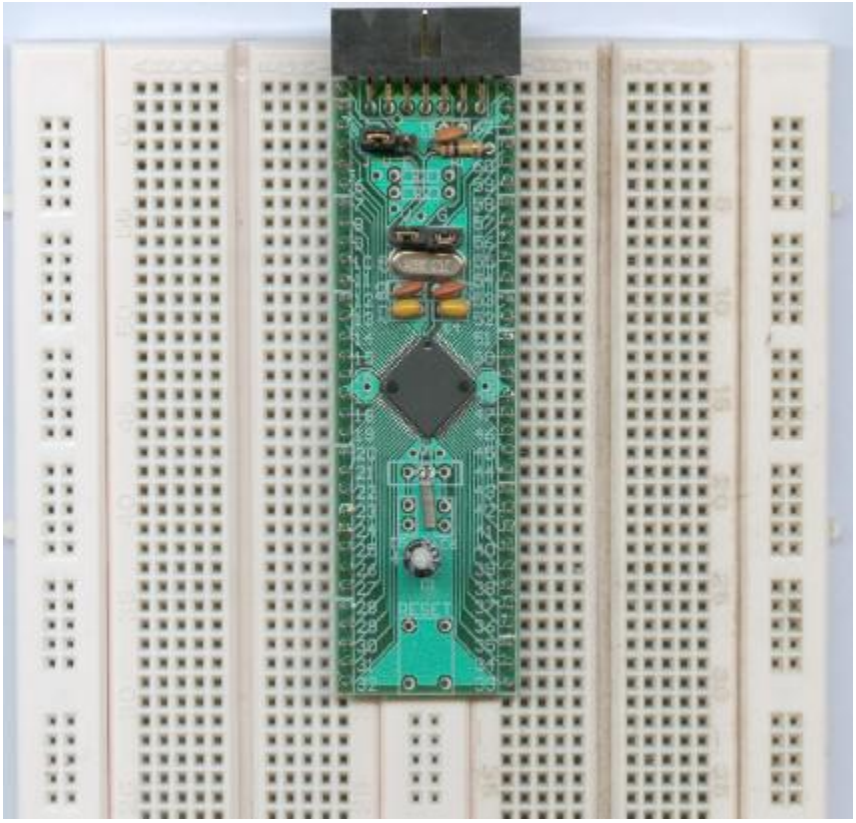


Figura 6. Header MSP430 sobre 2 protoboards.

Tabla 1. Lista de Componentes Header MSP430

Cant.	Designador	Valor	Empaque	Descripción
1	Q2	8MHz	XTAL-0.2	Oscilador
1	C3	10nF	Thru-hole	Cond. Cerámico
1	C1	10uF/6,3V	Radial 0,1"	Condensador Electrolítico
2	C7, C8	12pF,10pF	Radial 0,1"	Cond. Cerámico
1	Q1	32.768 Hz	XTAL-0.05 ó 0.2	Oscilador
1	R1	100K	AXIAL 1/4W	Resistencia Carbón
2	C2, C4	100nF	Thru-hole 0,1"	Condensador Cerámico
71		pines	separación 0,1"	Header 1 fila, 71 pines, para PCB
1	JTAG	IDC 7X2	HDR2X7	IDC 2x7 para PCB, ángulo recto
1	SW-PB	N/A	Pulsador	Código Global 165-630111
3		Jumper		Jumper
1	U1	MSP430F1611	TQFP-64	Microcontrolador

2.2 Programador/emulador JTAG

Debido a que el programador aún debía ser importado, el siguiente paso fue diseñar un programador propio. Los JTAG de Olimex tienen las siguientes características:

- Conexión al puerto paralelo.
- Compatible con el entorno de desarrollo *IAR Electronic Workbench for MSP430.*
- Compacto, toda la circuitería de interfaz se encuentra dentro de la carcaza de un conector DB25.
- Salida es un cable plano de 14 pines con un conector tipo IDC de 2 x 7 pines.
- No requiere fuente externa, extrae la alimentación desde el puerto paralelo.
- Tiene una salida regulada de 3.0 Volts para alimentar la MSP430.

Sin embargo, tienen un gran inconveniente: El cable de conexión a la MSP es de apenas 15[cm]. Esto implica la necesidad de conectar un cable alargador para puerto paralelo, lo cual aumenta su costo final, pero tiene una desventaja mucho mayor: el peso y la rigidez del cable alargador arrastran el Header con protoboard y circuito y lo tiran debajo de la mesa de trabajo, cosa que sucedió a menudo en los laboratorios con los Kits adquiridos. Alargar el cable plano es difícil, pues no existe un conector IDC macho para el cable. Desoldar internamente el cable y reemplazarlo por uno más largo es complicado, puesto que no posee una regleta de conexión, sino que cada conductor del cable plano se separa y se conecta en un punto distinto del circuito impreso.

En este punto del desarrollo, no existía un distribuidor en chile para los productos de Olimex. Las dificultades anteriormente mencionadas, junto con el costo que implicaba la importación del programador JTAG, hizo tomar la decisión de diseñar y construir un JTAG propio. El objetivo fue lograr un JTAG que tuviera las mismas características funcionales que el de Olimex, pero que tuviera un cable de conexión al microcontrolador de 1[m], y que pudiese fácilmente alargarse o reemplazarse por uno más largo.
Para lograr que el circuito entre en una carcaza DB25 hubo que utilizar componentes de montaje superficial. Un primer intento con las resistencias ***thru-hole*** de 1/8 Watt, que son

las más pequeñas, dejó claro que se necesitaban componentes de montaje superficial. Se utilizaron las resistencias y condensadores SMD más comunes, es decir, las más grandes, cuyo código de tamaño es 1206. Esto básicamente para hacer la construcción (soldado) más fácil, ya que el posicionamiento de las componentes debe hacerse a mano. Sin embargo, debido a la gran cantidad de componentes, aún no había espacio suficiente. La solución fue utilizar packs de resistencias, ya que muchas de las resistencias eran de un mismo valor. Así, 12 de las resistencias fueron reemplazadas por 3 packs de tamaño 1206 con 4 resistencias cada uno. Esto sin embargo agregó una complejidad mucho mayor para el ruteo de las pistas. Antes de mandar a hacer el circuito impreso, se consultó con el personal del Taller ELO la factibilidad de construir el circuito como estaba diseñado. Por consideraciones mecánicas (la tendencia a tirar del cable para desenchufar el programador del PC) se optó por colocar un conector IDC macho en el circuito impreso en vez de soldar directamente el cable plano e intentar de afirmarlo con silicona o algo parecido.

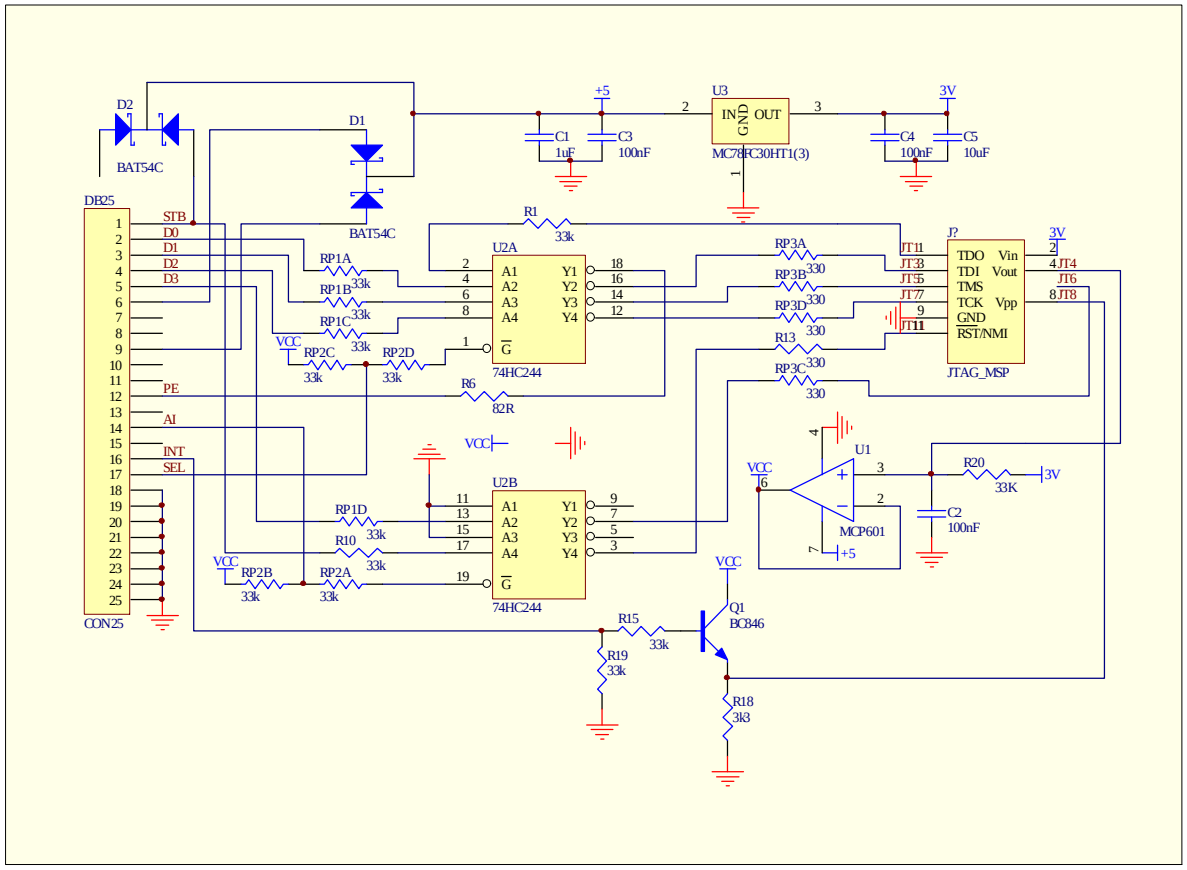


Figura 7. Circuito esquemático del programador JTAG.

En la figura 7 se presenta el diagrama esquemático del JTAG. Éste consta básicamente de un buffer que hace la interfaz de voltajes entre el puerto paralelo y la MSP, ya que esta última puede ser alimentada externamente entre 2.7 y 3.6 Volt y los voltajes de las señales JTAG no deben superar el voltaje de alimentación. Además posee un regulador de voltaje de *low-dropout* o baja caída, de 3.0 Volt salida, el cual obtiene su poder de 3 líneas del puerto paralelo. El voltaje de alimentación del buffer y por lo tanto el voltaje de salida de

las señales se establece mediante un amplificador operacional de bajo voltaje y bajo consumo, cuya referencia es el voltaje de alimentación de la MSP.

En la Figura 8 se encuentran fotos de ambas caras del circuito impreso armado y montado en una carcaza plástica para conectores DB25. Una vez hecho, el circuito impreso se ve relativamente simple, ya que se ve sólo una cara a la vez y quedan ocultas las pistas que van por debajo de las componentes. Sin embargo, el circuito fue extremadamente difícil de rutear y fueron necesarios muchos intentos y enroques de componentes para llegar al resultado final aquí presentado. La Tabla 2 contiene el listado de componentes del JTAG.

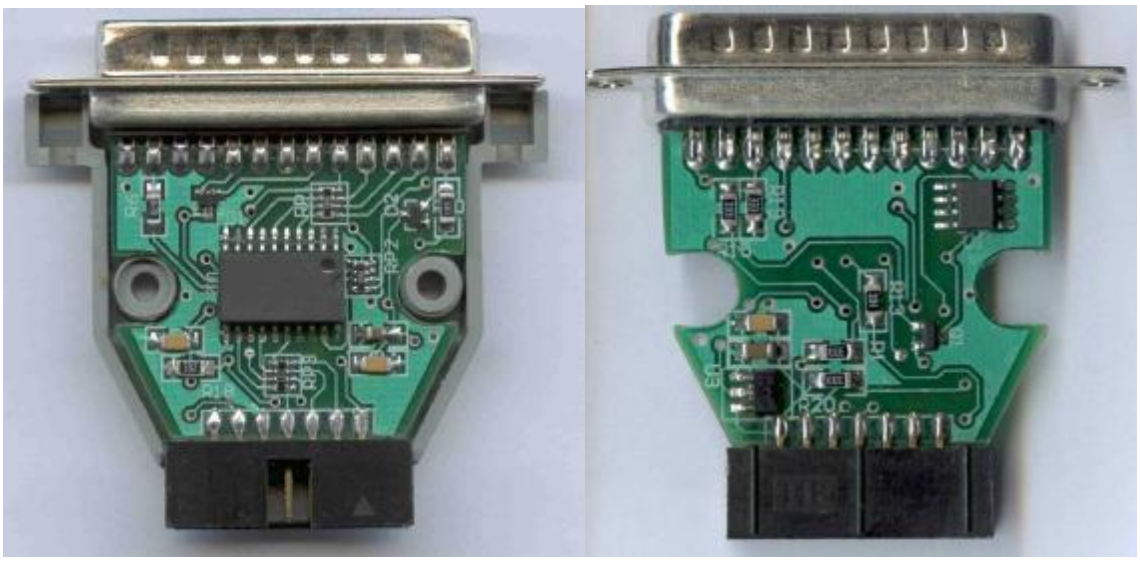


Figura 8. Ambas caras del circuito impreso con componentes.

Ya que los primeros prototipos del Header MSP430 y el JTAG alcanzaron a estar listos a principios del 2º Semestre de 2004, los profesores encargados del ramo "Laboratorio de Estructura de Computadores" construyeron 10 pares Header + JTAG para ser utilizados en las últimas 2 experiencias del laboratorio durante ese mismo semestre.

Finalmente se creó un documento en PDF llamado "MSP 430 Quickstart" que explica cómo comenzar a utilizar el Header, JTAG y Software. Los detalles incluyen cómo conectar la MSP al computador a través del JTAG, cómo se utilizan los jumpers, cómo se configura el software de emulación "IAR Electronic Workbench for MSP430", un programa de ejemplo que configura los registros de la MSP para inicializar correctamente los cristales y las señales de reloj, y finalmente una lista de problemas comunes al compilar, junto con sus explicaciones y soluciones. Este documento se encuentra en el Anexo B.

Tabla 2. Lista de Componentes programador/emulador JTAG

Cant.	Valor	Designador	Empaque	Descripción
1	1uF	C1	1206	Cond. Cerámico
1	3k3	R18	1206	Resistencia
1	10uF	C5	1206	Cond. Cerámico
5	33K	R1 R10 R15 R19 R20	1206	Resistencia
2	33k	RP1 RP2	4x0603A	Pack 4 resistencias
1	74HC244	U2	SOL-20	Buffer octal
1	82R	R6	1206	Resistencia
3	100nF	C2 C3 C4	1206	Cond. Cerámico
1	330R	RP3	4x0603A	Pack 4 resistencias
1	330R	R13	1206	Resistencia
2	BAT54C	D1 D2	SOT-23N	Pack 2 diodos Anodo común
1	BC846	Q1	SOT-23	Transistor NPN
1	CON25	DB25	DB-25M/EDGE	Conector DB25 volante
1	JTAG_MSP	JTAG	HDR2X7P	Conector IDC 2x7,macho,PCB, polarizado
1	MC78FC30HT1	U3	SOT89	Regulador Low-Dropout 3.0V
1	MCP601	U1	SO-8	Amplificador Operacional, bajo voltaje, bajo consumo
1				Carcaza plástica para DB25
1				Cable plano 1m 14 conductores
2				Conectores IDC 2x7 para cable plano.

3 Módulo de Red: uLAN

3.1 Introducción

Una de las grandes ventajas del kit de desarrollo EasyWeb II consiste en su conectividad a internet o a una red LAN. Construir un módulo para conectividad a ethernet fue la primera prioridad luego de la construcción de los módulos básicos.

Existen sólo 2 chips de red a considerar, puesto que son los únicos disponibles para mercado minorista y ambos son igualmente difíciles de obtener: el CS8900A de Crystal y el RTL8019 de Realtek, pero sólo el primero existe en versión 3.3V, razón por la cual es utilizado en la EasyWeb II y por lo mismo fue utilizado en este diseño. Además, utilizar el CS8900A-CQ3 tiene la ventaja de que existe en internet una implementación básica del protocolo TCP/IP para la familia de microcontroladores MSP430, llamada uIP (micro-IP). Este protocolo es gratuito y existen varias tarjetas de desarrollo que utilizan este microcontrolador con este chip de red, todos utilizando la misma conexión que la EasyWeb I y II.

En internet pueden encontrarse varios módulos de red que utilizan este chip, pero todos tienen uno de los siguientes problemas: O sólo poseen conexión a unos pocos pines, impidiendo por ejemplo el uso de interrupciones, ó la forma de conectarlos es muy poco amigable y no pueden ser montados en un protoboard.

Los requerimientos para el diseño de la tarjeta de red, bautizada uLAN (micro-LAN), son los siguientes:

- Operación a 3.3V
- En lo posible, utilizar el mismo chip de red que la EasyWeb II, para poder utilizar el mismo código.
- Montable en protoboard, en lo posible en una, o sino en 2 protos interconectados mediante una sola regleta de poder.
- Acceso a todos los pines que tengan señales útiles, incluyendo los que no son absolutamente necesarios pero que agregan funcionalidad.
- Porte compacto, ojalá que entre en un solo protoboard.
- En lo posible, componentes sólo por el lado superior, para evitar que sean dañados al utilizar alguna herramienta como un destornillador para sacar la placa del protoboard.

Luego del estudio del datasheet del CS8900 y los Aplication Notes asociados, el detalle de la funcionalidad de los pines de este chip se agruparon en 3 grupos: Señales mínimas de comunicación con el procesador, conexiones básicas de poder y componentes externas; y señales opcionales de comunicación con el procesador. Éstas se detallan a continuación:

Señales de comunicación con el procesador:

- SD0-SD7
  Bus de datos de 8 bit, ó byte par del bus de datos de 16 bit.
- SA0-SA3
  Bus de direcciones para operación en modo IO.

- IOR, IOW (activo bajo)
  Señales de Lectura y Escritura en modo de acceso IO. La escritura y lectura se hace de forma indirecta a través de un puntero y 16 registros básicos.

Conexiones básicas:

- DVDD, AVDD, DGND, AGND.
  Alimentación para las porciones análoga y digital internas del CS8900A. No deben tirarse pistas separadas para la alimentación análoga y digital; deben alimentarse de la misma forma. Debe ir un condensador de 100nF entre todos los VDD y tierra, lo más cercano posible a los pines de VDD. Solo habrá un pin externo para VDD y otro para tierra.
- XTAL1, XTAL2.
  Pines para conectar un cristal de 20MHz. A diferencia de la mayoría de los chips, éste posee condensadores internos, por lo cuál no es necesario conectar condensadores externos entre estos pines y tierra. No habrá acceso externo a estos pines.
- RES
  Debe conectarse una resistencia de 4.99kΩ, 1% entre este pin y tierra. Ésta irá en el circuito impreso y no habrá acceso externo a este pin.
- TXD+, TXD-, RXD+ y RXD-.
  Estas señales se conectan al conector RJ45 mediante un transformador de pulsos conforme a 10BaseT. La razón de trasformación es:
  - Tx 1CT:2.5
  - Rx 1CT:1

  CT indica que el transformador posee un terminal central.
  No debe ir un plano de tierra entre la salida del transformador de pulsos y el conector RJ45

Señales adicionales:

- SA4-SA19
  En modo IO, todos estos pines deben ir a tierra, excepto los pines A8 y A9 que van conectados a Vcc. Esto corresponde a la dirección base del bus ISA, que para tarjetas de red es por defecto 0300h. En el módulo sólo se dejaron disponibles SA4 a SA12, dejando abierta la posibilidad de direccional los 4kB de memoria RAM interna en el modo de direccionamiento directo. Los demás, SA13-SA19 se encuentran cableados a tierra, pues no son necesarios.
- SD8-SD15
  Byte impar del bus de datos en modo de comunicación de 16 bits. En modo 8 bits puede ser dejado al aire. Éstos se encuentran disponibles para conexión en el módulo uLAN, dejando abierta la posibilidad de utilizar el modo de 16 bits.
- ELCS, EECS, EESK, EEDataOut, EEDataIn
  Señales para EEPROM externa que carga automáticamente la configuración inicial de los registros. Utilizado en tarjetas de red para PC para almacenar configuraciones como dirección MAC, dirección ISA base, puerto DMA y puerto INTR, etc.
- CHIPSEL (activo bajo)
  Selección del Chip. Se dejará disponible.

- DMARQx (activo bajo), DMACKx
  Señales para la transferencia directa a memoria. Existen 3 canales, pero sólo 1 de ellos puede utilizarse a la vez y no hay diferencia entre ellos. Esto es para poder elegir cuál canal DMA del puerto ISA utilizará la tarjeta de red. Para conexión a microcontroladores no es necesario, pero se dejará disponible el canal 0, ya que todos los canales son equivalentes y sólo puede utilizarse uno de ellos.
- CSOUT (activo bajo)
  Chip select para memoria PROM externa de booteo.
- MEMW, MEMR (activos bajo)
  Señales para acceso directo a memoria.
- INTRQx
  Señal para petición de interrpución. Existen 4 canales, al igual que con la señal DMA. Sólo funciona en el modo 16 bits. Se dejara disponible únicamente el canal 0.
- IOCS16, MEMCS16 (activo bajo)
  Señales de salida de colector abiero. Se activan cuando el chip detecta una dirección válida de IO o de Memoria. No son necesarias pero se dejarán disponibles.
- SBHE (activo bajo)
  Indica que se están transfiriendo datos en el byte alto de datos (SD8-SD15). Debe cambiar al menos una vez de estado para pasar al modo 16 bits.
- REFRESH (activo bajo)
  Le indica al CS8900A que se está refrescando la memoria DRAM. Cuando está activo, se ignora el estado de todas las demás señales. No tiene utilidad en conexión con microcontroladores y se dejará deshabilitado.
- AEN
  Señal de reloj para la transferencia DMA. Puede utilizarse como bit de chip select (activo bajo) si no se está utilizando DMA.
- IOCHRDY
  Salida de colector abierto. Indica cuando el canal IO está listo. Extiende los ciclos de escritura y lectura, tanto en modo IO como el modo Memoria. Se dejará disponible.
- RESET
  Entrada activa alta, resetea el CS8900A. Se incluirá un circuito RC para lograr un reset automático al polarizar el Módulo. De todas formas este pin se dejará disponible.
- TEST (activo bajo)
  Pone el chip en modo de diagnóstico. Se dejará deshabilitado
- SLEEP (activo bajo)
  Pone el chip en modo de bajo consumo. Se dejará disponible.
- BSTATUS/HC1 (activo bajo)
  Salida de colector abierto. Puede ser controlado por software ó puede indicar cuando una recepción causa un acceso al bus ISA. Se dejará disponible.
- DI+, DI-, CI+, CI-, DO+,DO-
  Señales para interfaz AUI. No se utilizará.

- LINKLED(HC0)
  Salida de colector abierto. Indica que se ha establecido una conexión ethernet. También puede ser controlado por software. Se pondrá un led en este pin y no habrá conexión exterior
- LANLED
  Salida de colector abierto. Indica comunicación ethernet. Se pondrá un led en este pin y no habrá conexión al exterior.

Luego de estudiar los Aplication Notes y numerosos circuitos en internet, se llegó diseño el circuito esquemático de la Figura 9. Este deja disponibles todos los pines que pueden ser de utilidad con un microcontrolador. Las señales redundantes sólo tienen sentido en un bus ISA por lo que no se dejaron disponibles.

3.2 El transformador de pulsos

Todos los fabricantes que producen transformadores de pulso con la relación de transformación necesaria tienen listado al CS8900A-CQ3 como el único chip que utiliza esa relación. Por esta razón son pocos los modelos y fabricantes que lo producen, y sólo muy de vez en cuando algún distribuidor de los EEUU tiene stock.

El transformador de pulsos utilizado en la EasyWeb II es el e2023, del fabricante Pulse (www.pulseeng.com). Pero este mismo fabricante produce un conector RJ45 que incluye el transformador de pulsos en la misma carcaza. Produce 4 modelos cuya única diferencia es si poseen led o no, y si poseen aletas de conexión con la carcaza metálica para evitar interferencias electro-magnéticas (EMI) o no:

- J00-0062
- J00-0063 con LEDs
- J00-0025 con aletas EMI
- J00-0051 con aletas EMI y LEDS

La ventaja de estos conectores es que tienen el mismo precio o menor que el transformador de pulsos e2023. Otros fabricantes que producen conectores RJ45 con transformador de pulsos para 3V son:

- HALO (www.haloelectronics.com): HFJ11-1041E
- ABRACON (www.abracon.com): ARJ-108
- EPJ9056

El circuito impreso posee pads para la conexión de los leds de conectores RJ45 que los incluyan, así como también para leds externos de 5mm. La experiencia mostró que utilizar los leds del conector es poco práctico, ya que son ocultados por el cable de red conectado al módulo. Es preferible utilizar los LEDs de 5[mm], pero la opción final queda abierta.

La diferencia entre los conectores de PULSE de 3V y 5V radica, además de la relación de transformación, en cuáles pines están conectados a los puntos medios de los transformadores. Mientras que en las versiones 3V los pines son el 7 (TXD) y el 8 (RXD), en las versiones 5V los pines son el 4 y 5 respectivamente.

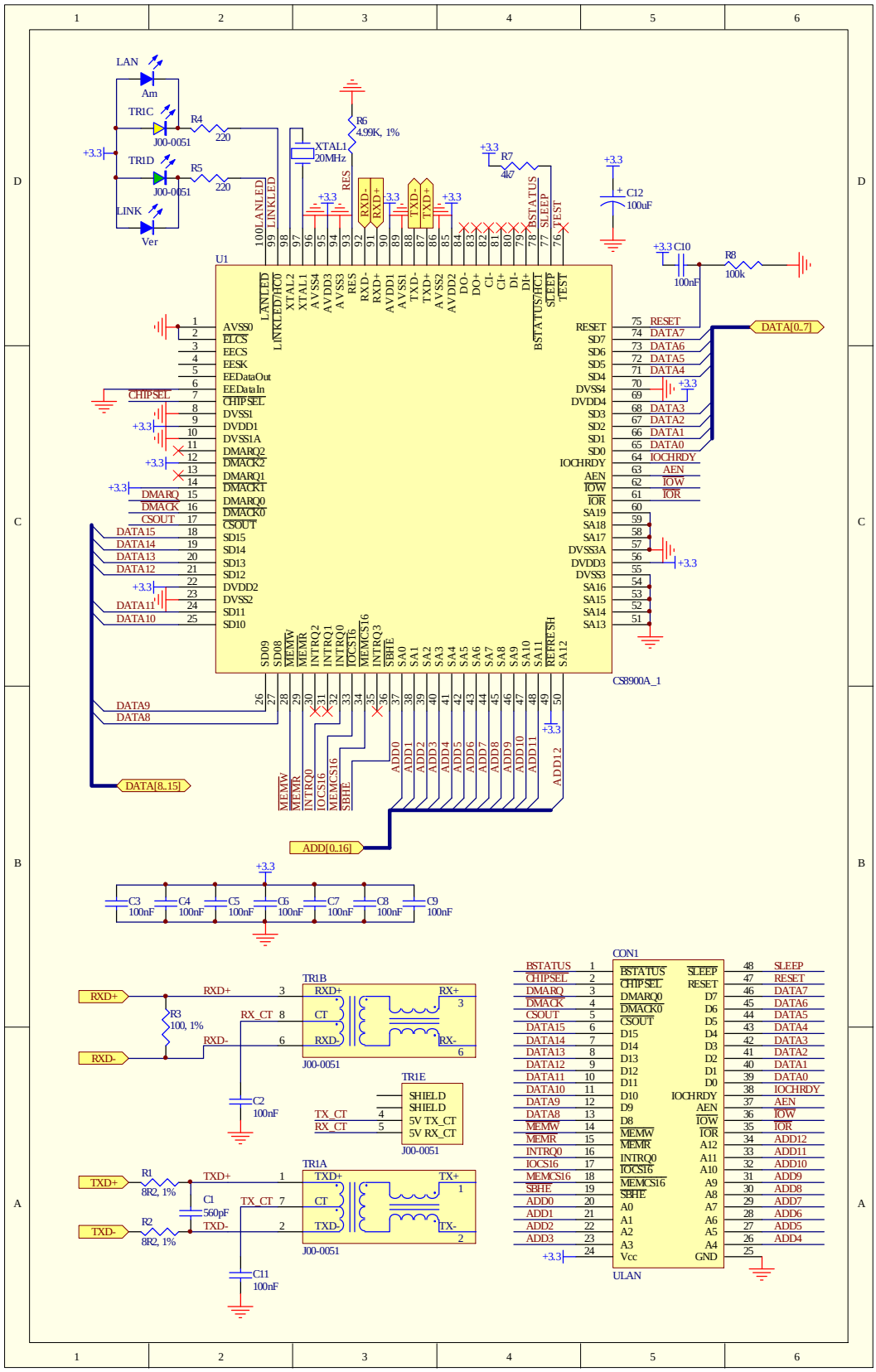


Figura 9. Circuito esquemático del módulo uLAN.

El conector HALO HFJ11-1041E (de 3V) posee los puntos medios en los pines 4 (TXD) y 5(TXD), igual que las versiones de 5V de PULSE. Por esta razón los pines 7 y 4 han sido cortocircuitados, al igual que los pines 8 y 5. De esta manera el mismo circuito impreso sirve para cualquiera de los conectores.

Por otro lado la versión de 5V del CS8900A tiene la misma distribución de pines. Así el diagrama de conexiones es el mismo para 3V y 5V, pudiendo utilizarse entonces el mismo circuito impreso.

Para hacer un módulo de 5V, las únicas otras componentes que cambian, además del chip y del transformador, son las resistencias R1 y R2 a 24.3Ω, 1% y C1 a 68pF. La Tabla 3 muestra las componentes dependientes del voltaje de operación, mientras que la lista de componentes para la tarjeta uLAN de 3V se presenta en la Tabla 4.

Tabla 3. Componentes que cambian con 3V o 5V.

		Relación de transformación			
	Chip	TX	RX	R1,R2	C1
3V	CS8900-CQ3	1CT:2.5	1CT:1	8.0Ω ó 8.2Ω 1%	560pF
5V	CS8900-CQ	1CT:1	1CT:1	24.3Ω 1%	68pF

Tabla 4. Lista de componentes módulo uLAN.

Cant.	Valor	Designador	Tipo	Descripción
1	560pF	C1	1206	Condensador cerámico
10	100nF	C2 C3 C4 C5 C6 C7 C8 C9 C10 C11	1206	Condensador cerámico
1	100uF	C12	Axial-0.1	Condensador electrolítico
1	Regleta 46 pines	uLAN	SIP	Regleta 2.54mm
1	LED Amarillo	LAN	5mm	Light Emitting Diode LED
1	LED Verde	LINK	5mm	Light Emitting Diode LED
2	8R0 ó 8R2, 1%	R1, R2	1206	Resistencia de precisión
1	100, 1%	R3	1206	Resistencia de precisión
2	220R	R4, R5	1206	Resistencia
1	4.99K, 1%	R6	1206	Resistencia de precisión
1	4k7	R7	1206	Resistencia
1	100k	R8	1206	Resistencia
1	J00-0051	TR1	RJ45	RJ45 con trafo, TX 2.5CT:1, RX 1CT:1
1	CS8900A_1	U1	LQFP-64	Chip de red (Packet Whaquer)
1	20MHz	XTAL1	XTAL-0.2	Cristal

Algunas de las consideraciones al rutear la PCB fueron:

- Poner planos de tierra donde sea posible, exceptuando bajo el conector RJ45.
- Poner todas las componentes por la cara de arriba.
- Rutear la pista de Vcc de manera de que pase desde el pin de alimentación del módulo primero por el condensador y luego sólo hacia el pin correspondiente a ese condensador. Esto disminuye la variación de voltaje en los pines de alimentación.
- Hacer la PCB lo más angosta y corta posible.

La Figura 10 presenta el ruteo del circuito impreso. De izquierda a derecha: Cara superior, cara inferior y guía de componentes, todas vistas desde arriba. La Figura 11 es una foto del módulo uLAN armado con todas sus componentes soldadas.

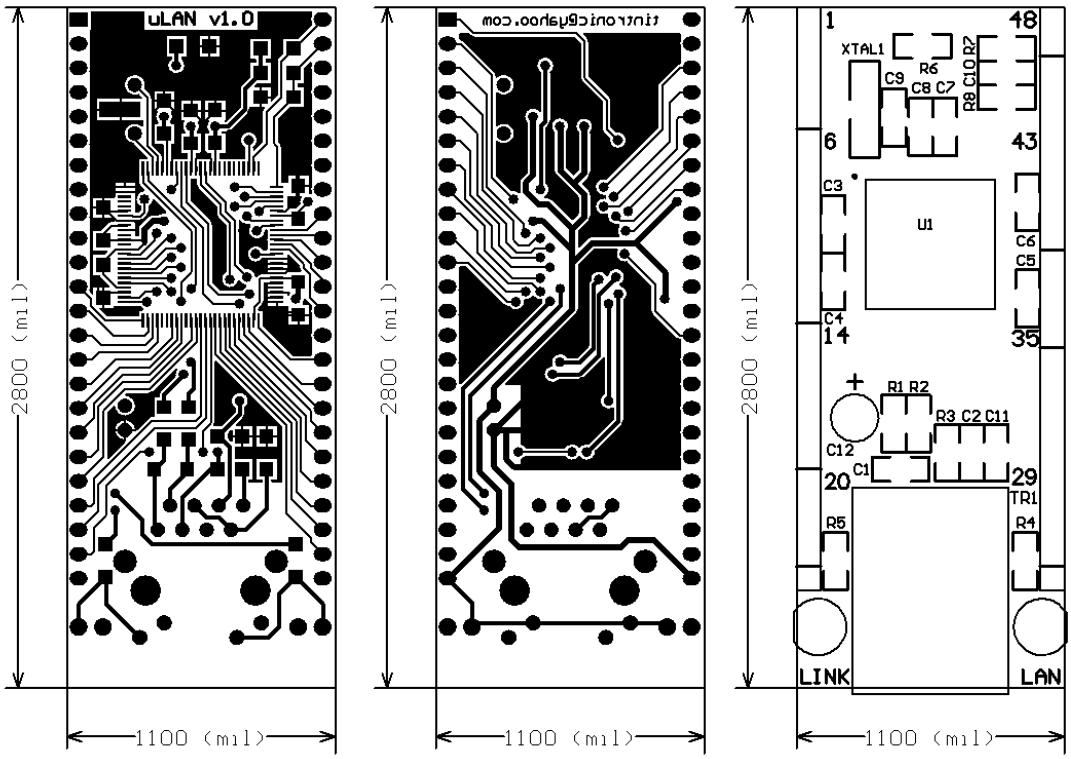


Figura 10. Caras superior, inferior y serigrafía de la tarjeta uLAN.

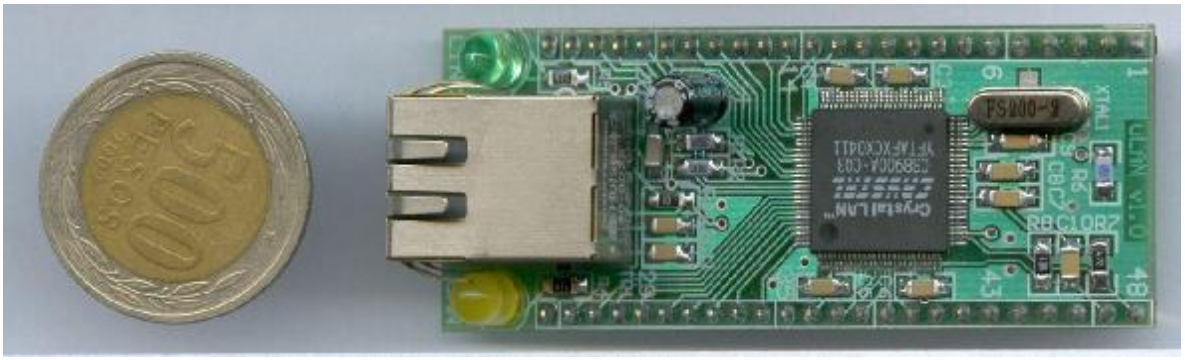


Figura 11. Módulo uLAN.

3.3 Driver uLAN para MSP430

El stack TCP/IP para microcontroladores llamado uIP es el utilizado en los programas de ejemplo proveídos por Olimex para su tarjeta EasyWeb II. En internet aparecen otras compilaciones de este stack, pero todos utilizan la configuración de hardware de la EasyWeb II. El driver es absolutamente rígido y no hay otra forma de cambiar algún bit o bus de puerto más que revisar todo el código del driver. Además, el driver asume que todos los pines a los que se encuentra conectado el chip de red, son únicamente utilizados por éste y no pueden ser multiplexados con ningún hardware externo. Por esta razón se ha modificado este código para poder definir y cambiar fácilmente todas las señales de un puerto a otro.

En primer lugar fue removido todo el código que no tiene directa relación con la comunicación con el CS8900A, como es la inicialización de los osciladores que se encuentra dentro de la rutina de inicio del CS8900A y la cuál por ejemplo no sirve para los módulos Header MSP430 diseñados en este trabajo.

3.3.1 Definiciones de puertos y bits:

Los bits IOW e IOR pueden ser cualesquiera de un mismo puerto. He agregado un bit más, para utilizar una señal de ENABLE del chip de red. Así las señales de escritura y lectura también pueden ser multiplexadas, por ejemplo con las señales RS y RW de un display LCD, o con las señales WR y RD de una RAM externa. Si no desea utilizarse el bit de enable, puede borrarse esa línea y el código va a ignorar todas las referencias a ese bit. Estos 3 bits siempre son salidas.

```
#define IO     P3OUT     //IO port
#define IODIR  P3DIR
#define IOSEL  P3SEL
#define IOR    BIT6      //IOR bit, active LOW
#define IOW    BIT7      //IOW bit, active LOW
#define CS8900_EN BIT1   //AEN bit, active LOW
```

Inicialmente se quiso poder elegir entre el nibble alto (4msb) y bajo (4 lsb) de cualquier puerto para el bus de direcciones A0-A3. El código resultante permite no sólo esto, sino que también elegir cualesquiera 4 bits seguidos, gracias al `#define ADOFF`, el cuál establece cuál es el bit correspondiente a A0. En el ejemplo, A0 a A3 están conectados a P6.4 a P6.7 respectivamente.

Sólo deben modificarse los primeros 4 `#define`. Los últimos 2 son resultado del 4º. Estos 4 bits sólo son salidas durante la comunicación con el chip, luego se establecen como entradas para asegurar una fácil multiplexación.

```
#define ADOUT  P6OUT
#define ADDIR  P6DIR
#define ADSEL  P6SEL
```

```
#define ADOFF  4    //=0 for low nibble, =4 for high nibble.
#define ADMASK (0x0F<<ADOFF)  //Only using 4 pins for address
#define ADBIT0 (0x01<<ADOFF)  //lsb pin of address bus
```

El byte bajo del bus de datos también puede ser cualquier puerto.
La dirección de este puerto depende si se están leyendo o escribiendo datos, pero al final siempre vuelve a establecerce como entrada, para una fácil multiplexación.

```
#define DLOUT  P4OUT
#define DLIN   P4IN
#define DLDIR  P4DIR
#define DLSEL  P4SEL
```

Las rutinas modificadas fueron probadas con varios puertos y grupos de bits exitosamente. También funcionó exitosamente la multiplexación del bus de direcciones con el bus de datos de un display LCD en modo 4 bits.

En la Figura 12 se muestra la forma de conectar el módulo uLAN y un display LCD a un Header MSP430 de la misma forma en que están conectados en una EasyWeb II.

En el CD adjunto puede encontrarse el código completo del driver.

3.3.2 Advertencia.

La versión del uIP compilado para MSP430 y CS8900A que fue utilizada como base, que es la misma que se ha estado utilizando en el departamento, tiene un error en la función `u8_t cs8900a_poll(void)` la cuál corrompe los datos desde el byte Nº 118 del paquete (app byte Nº 68 de `uip_data[]`) en adelante. Paquetes con menos de 118 bytes no sufren alteraciones. El problema radica en que en el modo 8 bits, dependiendo de la operación, hay que accesar primero el byte par o el impar para evitar o forzar el incremento automático del puntero. La forma correcta de leer los datos del paquete es primero el byte par y luego el impar. Los bytes 0 a 117 pueden accederse de forma “random”, por eso se leían correctamente, pero una vez leído el 118, todos los bytes restantes deben leerse en forma continua, ya que el chip de Red descarta cada palabra de 16 bits luego de leerse el byte impar. Ésta es la razón por la cuál llegaban bien los primeros bytes, y desde el 119 en adelante, todos los bytes impares eran reemplazados por el impar siguiente, obteniéndose la secuencia 116,117,118,121,120,123,122, etc, dando la apariencia de que los bytes de cada palabra han sido invertidos.

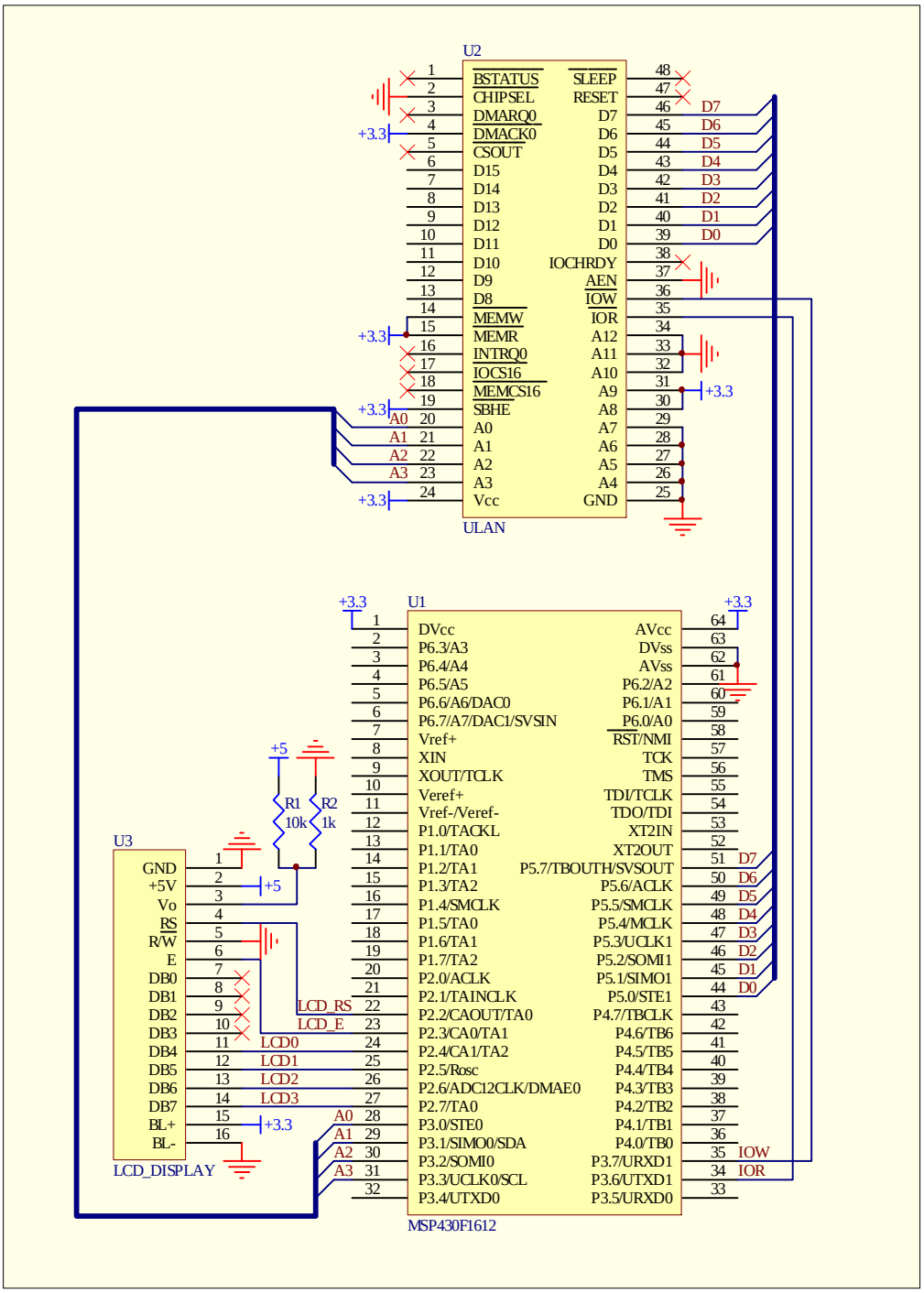


Figura 12. Conexión equivalente a EasyWeb II.

4 Módulo RS-232.

4.1 Introducción

Muchos microcontroladores poseen una interfaz UART o USART para comunicación serial asincrónica, tipo RS-232, que en un PC se denomina puerto "COM". Si bien los microcontroladores poseen hardware para generar la secuencia de bits en los tiempos correctos, no son capaces de generar el voltaje especificado por el estándar RS-232, por lo cuál requieren de un chip externo que haga esta conversión de voltajes.

Si bien el circuito de conversión puede armarse fácilmente en un protoboard, disponer del circuito en un módulo tiene las siguientes ventajas:

- Ahorro de espacio en el protoboard.
- Ahorro de tiempo en el armado y testeo del circuito.
- Evita conexiones erróneas del chip conversor.
- Pueden incluirse LEDs monitores.
- Puede incluirse un conector DB9 macho (el cuál no es compatible con protoboards) para la fácil conexión a otro dispositivo RS-232 a través de un cable *null-modem*.

4.2 El estándar RS-232.

El nombre oficial del estándar es EIA/TIA-232-E y es un estándar completo, puesto que no sólo especifica los niveles de voltaje y señal, sino que además especifica la configuración de pines de los conectores y una cantidad mínima de información de control entre equipos. También especifica la forma y características físicas de los conectores.

Este estándar fue definido en 1962, antes de la lógica TTL, razón por la cuál no utiliza los niveles lógicos de 5 volts y tierra. Un nivel alto a la salida del transmisor está definido como un voltaje entre +5 y +15 volts, mientras que un nivel bajo está definido como un voltaje entre -5 y -15 volts.

La lógica del receptor fue diseñada para permitir un nivel de ruido de 2 volts. Así, un nivel alto para el receptor está definido en el rango +3 a +15 volts, mientras que un nivel bajo va desde los -3 a los -15 volts.

Es importante notar que un nivel alto está representado por un valor lógico '0', históricamente llamado *spacing* (espacio), mientras que un nivel bajo representa un valor lógico '1', históricamente referenciado como *marking* (marca).

Este estándar también define un máximo *slew rate* o máxima variación de voltaje de 30[V/µs] para evitar el *crosstalk*, que es la inducción de las señales que viajan por un cable en los cables adyacentes. Inicialmente, el estándar limitaba la velocidad de transferencia de datos a 20[kbps] (kilo bits por segundo). Actualmente los circuitos integrados soportan velocidades mucho mayores, de hasta 350[kbps], manteniendo el *slew rate*.

La carga vista por el transmisor se especificó en 3 a 7 [kΩ]. En un principio se estableció un largo máximo del cable de 15 metros, pero luego fue modificado por la revisión D del estándar. Ésta especifica una máxima capacitancia de 2500[pF], en vez de establecer un

largo máximo. Así, el largo máximo depende de la capacitancia característica del cable utilizado.

El estándar estableció 4 grupos de señales: común, datos, control y temporizamiento, sumando en total 24 señales. También especifica un conector de 25 pines llamado DB25, el cuál es capaz de incluir todas estas señales. Afortunadamente sólo muy pocos equipos utilizan esta gran cantidad de señales. La mayoría, además de la señal de tierra de referencia, requiere sólo 2 para datos y 2 para control, ó sólo el par de datos. Estos últimos suelen utilizar un conector DB9S, de 9 pines, el cuál permite acomodar las mínimas señales utilizadas por equipos modernos. La figura 13 presenta las señales en un conector DB9 macho como el de un computador personal. Este conector está visto desde fuera del computador. Las señales que apuntan hacia la derecha son señales que salen del computador, mientras que las que apuntan a la izquierda son entradas al computador.

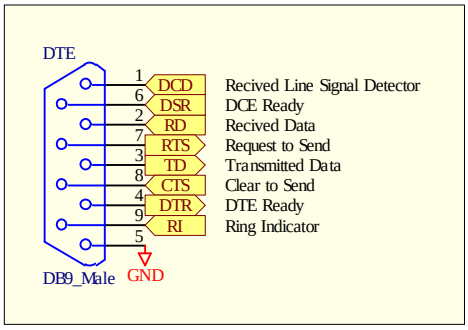


Figura 13. Señales RS-232 en un conector DB9 macho.

Las mínimas señales utilizadas en una comunicación bidireccional son TD para transmitir datos y RD para recibir datos. Asimismo, si desea utilizarse control de flujo por hardware, se utilizan las señales RTS (petición de transmisión) y CTS (habilitado para transmitir). El control de flujo impide que un transmisor rápido sature a un receptor lento. Normalmente el PC podrá transmitir datos ininterrumpidamente, pero el equipo receptor puede ser más lento y no alcanzar a procesar todos los datos que le envía el PC.

La interfaz RS-232 está pensada para conectar un terminal de datos (DTE) a un equipo tipo modem, llamado equipo de comunicación de datos (DCE). El DCE es un equipo que hace la interfaz entre el DTE y el medio por el cuál se transmitirán los datos. Un ejemplo de DCE es un modem, el cuál hace de interfaz entre un PC y la línea telefónica. También pueden conectarse 2 DTE directamente a través de un puerto RS-232. Para ello se emplea un cable denominado *null-modem*, haciendo referencia a una conexión sin modem. Este cable es especial, ya que posee líneas de datos y control invertidas entre sus 2 conectores. El detalle de la conexión interna en un cable null-modem se presenta en la figura 14.

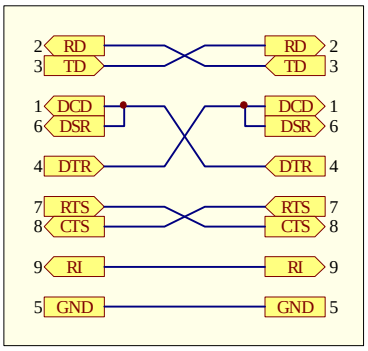


Figura 14. Conexión Null-Modem entre dos DTE.

Las señales RTS y CTS también pueden utilizarse para establecer la dirección de comunicación en un sistema *half-duplex*. Esto es necesario cuando se utilizan conversores RS-232 a RS-485, pues este último utiliza un mismo par trenzado tanto para transmitir como para recibir, convirtiéndolo en un protocolo *half-duplex*.

4.3 Diseño esquemático

Existe un circuito integrado muy popular para hacer la conversión de lógica TTL de 5V a lógica RS-232 llamado MAX232. El chip incluye inversores, ya que un 0 lógico se transforma en un nivel alto en el lado RS-232 y vice versa. Además, el chip es alimentado con una fuente simple de +5 volts, y a través de la conexión externa de 4 condensadores electrolíticos de 1 ó 10[µF] (dependiendo del fabricante del chip) genera el voltaje necesario para la transmisión RS-232. Cada chip posee 2 drivers y 2 receptores, con lo cuál pueden conectarse las 2 señales de datos y, de ser necesario, una señal de control de entrada y otra de salida.

La versión de 3V del MAX232 se llama MAX3232. Existen también modelos similares de otros fabricantes, como el ST3232E, los cuales son compatibles pin a pin. Las principales ventajas del ST3232E y MAX3232 sobre otros chips son:

- Voltaje de operación de 3 a 5V.
- Velocidad de hasta 250kbps
- 15kV de protección electroestática.
- Alta eficiencia, sólo 300uA de alimentación.
- No requiere condensadores electrolíticos, sólo cerámicos de entre 0.047 y 0.47 [µF], dependiendo del voltaje de alimentación.
- Posee una configuración de hardware para poder utilizar el mismo módulo tanto en 3[V] como en 5[V], sin tener que cambiar los valores de los condensadores.
- Disponible en 4 tipos de empaque SMD y un empaque DIP.

Este módulo fue diseñado para poder conectar un microcontrolador a cualquier equipo que utilice la interfaz RS-232 por medio de un cable serial tipo null-modem, con un conector DB9 común.

Los valores de los condensadores dependen del voltaje de alimentación, como muestra la Tabla 5, extraída de la hoja de datos del fabricante. Para poder utilizar el mismo módulo para microcontroladores de 3 y 5 volts, se eligió la configuración de la última fila (3.0 a 5.5V).

Tabla 5. Valores de condensadores en función de la alimentación.

Vcc	C1	C2	C3	C4
3.0 a 3.6	0.1	0.1	0.1	0.1
4.5 a 5.5	0.047	0.33	0.33	0.33
3.0 a 5.5	0.1	0.47	0.47	0.47

Utilizando un par driver/receptor para las señales de datos, queda otro par disponible para señales de control. Este par se conectó a las señales RTS y CTS. El host pone la señal RTS en 1 cuando quiere enviar datos por el pin TD y luego espera a que el otro equipo le dé el 'visto bueno', poniendo la señal CTS en 1 para iniciar la comunicación. Por esto a los pines se les llamó Req (***Request***) y Ack (***acknowledged***), respectivamente. Si no se usan estas señales, es recomendable puentear el pin Req con el pin Ack, puesto que el otro extremo puede necesitar estas señales para funcionar adecuadamente. Así, cuando el otro extremo ponga RTS en 1, indicando que desea transmitir, automáticamente se pondrá CTS en 1, indicando que puede iniciar la transmisión. De lo contrario el otro extremo estará esperando en vano que el CTS se ponga en 1. Por esta misma razón, se han puenteado las señales DCD, DSR y DTR, ya que se necesitaría un 2º chip y otros 4 condensadores para poder acceder a estas señales, las cuales difícilmente serán de utilidad en el laboratorio y sólo aumenta innecesariamente el costo de cada módulo.

Para conectar estos módulos a un PC se requiere de cables *null-modem*, los cuales se encuentran disponibles en pañol de electrónica.

El estado de las señales de comunicación puede monitorearse a través de los LEDs Tx y Rx. Debido a que la salida lógica del driver no puede manejar grandes corrientes, fue necesario utilizar un buffer para encender los LEDs. Para ello se utilizó un LM358 (o compatible), que es un amplificador operacional de bajo costo.

El diseño esquemático resultante se presenta en la Figura 15 y la lista de componentes en la Tabla 6.

El módulo posee un puerto de 6 pines:

- 2 para alimentación (Vcc y GND)
- 2 de datos (Tx y Rx) para las señales TD y RD.
- 2 de control: Req para RTS y Ack para CTS.

Tabla 6. Lista de componentes modulo RS-232.

Cantidad	Valor	Designador	Tipo	Descripción
2	220Ω	R5, R6	1206	Resistencia Carbón SMD
2	100nF	C1,C5	1206	Condensador Cerámico SMD
4	10kΩ	R1,R2,R3,R4	1206	Resistencia Carbón SMD
3	470nF	C2,C3,C4	1206	Condensador Cerámico SMD
1	CON6	J1	SIP-6	Regleta 1x6 Pines tipo Header
1	DB9	DB9_M	DB-9/M	Conector DB9 Macho para PCB 90º
1	LM358	U2	SO-8	Amplificador Operacional Doble SMD
1	LED Rojo	D2	5mm	LED
1	LED Verde	D1	5mm	LED
1	ST3232	U1	SO-16	Driver/Receptor RS-232 SMD

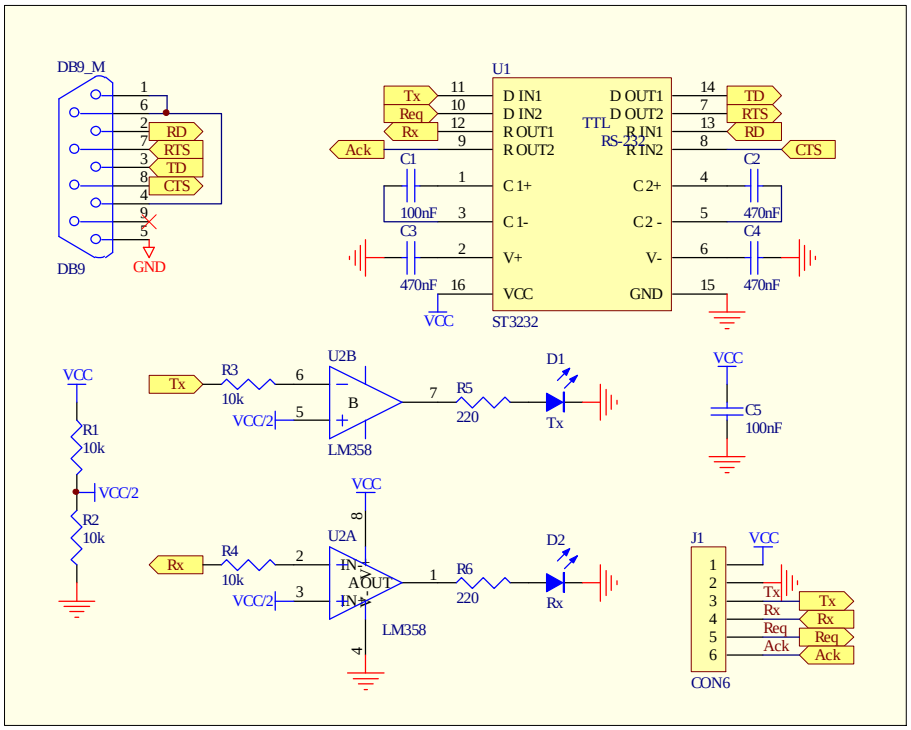


Figura 15. Circuito esquemático del módulo RS-232.

4.4 Diseño del circuito impreso

Para diseñar el circuito impreso se comenzó por ubicar el conector DB9 y la regleta de conexión al protoboard. Todas las componentes de montaje superficial se colocaron por la cara inferior. De esta manera se evita la utilización de una PCB de dos caras, reduciendo los costos. Todas las pistas fueron ruteadas a mano, excepto las conexiones de tierra. Esto porque es el plano de tierra el cual finalmente se encarga de conectar las componentes a tierra. Sólo debe tenerse cuidado de que el plano de tierra no quede separado en dos o más sectores, es decir, que todo el plano de tierra quede interconectado.

Finalmente se presenta en la Figura 16 el diseño del circuito impreso y la serigrafía de la cara superior, para indicar claramente donde van las señales en la regleta de conexión. La Figura 17 muestra una foto de la cara superior del módulo RS-232 montado.

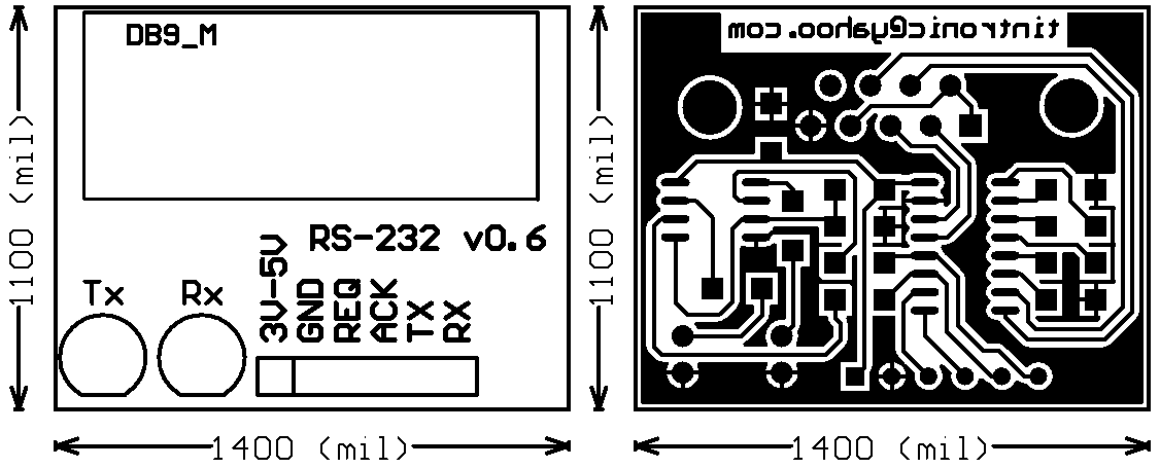


Figura 16. Diseño del circuito impreso.

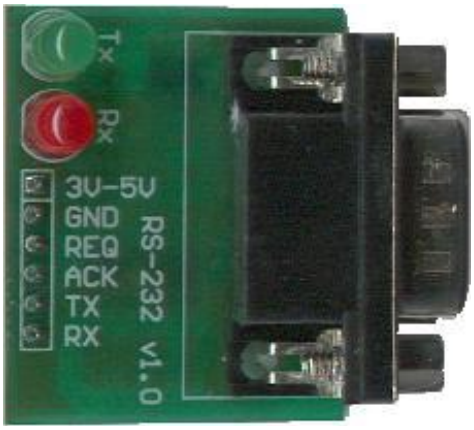


Figura 17. Módulo RS-232.

En el CD se encuentra un directorio con los documentos utilizados como referencia para este módulo, incluyendo la hoja de datos del chip utilizado. En el CD también se adjuntó la cotización de las componentes en Global Chile Eletrónica.

5 Módulo USB Host: uUSB-HS

5.1 Introducción

Ciertamente la interfaz más utilizada para conectar dispositivos externos a un PC hoy en día es el Universal Serial Bus o USB. Esta interfaz utiliza sólo un par trenzado para comunicación en modo *half duplex*. Además posee 2 cables de alimentación que proporcionan un mínimo de 4.4V y hasta 500mA para alimentar periféricos de bajo consumo sin la necesidad de una fuente externa.

Una comunicación USB requiere de un dispositivo maestro llamado HOST que normalmente es un PC, el cuál inicia absolutamente todas las transferencias, y uno o más dispositivos esclavos. Hasta 127 esclavos pueden ser comandados por una interfaz USB HOST.

Debido a que el HOST inicia todas las transferencias, no existe comunicación directa entre dos dispositivos esclavos. Debido a esto, todo el grueso del procesamiento se hace en el PC HOST, simplificando los controladores USB esclavos, reduciendo así enormemente el costo de los chips de interfaz USB. Gracias a esto, hoy en día se encuentran innumerables dispositivos USB a muy bajo costo, como lo son cámaras web, teclados, Mouse, scanner y los últimamente populares Pendrive, o memorias flash externas, cuyo costo va en rápido descenso a la vez que su capacidad aumenta exponencialmente. Incluso ya existen controladores tipo *bridge* entre USB e IDE, los cuáles poseen pines para el conector USB y pines para el conector IDE, y no requieren de ningún procesador adicional que haga la traducción entre ambos protocolos. Sólo requieren unas pocas componentes externas.

Resulta entonces lógico pensar que una forma barata y simple de agregar funcionalidades a un microcontrolador será utilizando una interfaz USB HOST. De esta manera puede crearse un *datalogger* con capacidad virtualmente ilimitada mediante el uso de un Pendrive o un disco duro USB, o agregarse una cámara de video sin la necesidad de buscar un chip digitalizador de video ni tener que construir una interfaz compleja. Conectar un disco duro o un Pendrive a un microcontrolador sólo requiere de programar las funciones mínimas de la especificación USB para dispositivos de almacenamiento masivo o *Mass Storage Device Class*.

Hoy en día existen muchos microcontroladores que incluyen una interfaz USB como si fuera un puerto serial más y existen muchos puentes USB-RS232 y USB-Paralelo, pero son todos para dispositivos esclavos, es decir, son para construir equipos periféricos para ser conectados a un PC. Por esta razón no pueden iniciar una transferencia USB, sólo pueden responder a comandos enviados por el HOST.

La mayor parte de los pocos controladores USB HOST que pueden encontrarse en Internet incluyen un microcontrolador, típicamente un 8051 modificado, el cuál se conecta internamente al controlador USB. Entonces, para que un microcontrolador externo pueda implementar un HOST, debe comunicarse con el controlador USB a través del 8051. La desventaja es que el 8051 es un microprocesador y no posee los periféricos normalmente disponibles en los microcontroladores actuales, lo que implica que no puede sustituir a un microcontrolador. Esto implica tener que programar 2 procesadores y sus lenguajes respectivos. Además, el hecho de poseer un 8051, aumenta el costo del chip y el costo total

del equipo en construcción. La única ventaja de este tipo de configuración radica en poder programar el *stack* USB en el 8051, liberando al microcontrolador de esta tarea.

Se encontró un único controlador USB HOST que no incluyera un procesador y que estuviera disponible para el mercado minorista a través de un distribuidor en EEUU. Corresponde al SL811HS de Cypress, el cuál puede ser configurado tanto en modo HOST como en modo esclavo.

El funcionamiento de este chip es similar al controlador de ethernet CS8900A en cuanto a que genera automáticamente los bits de preámbulo y los CRC correspondientes. Se podría decir que también es un *packet whacker* como el CS8900A, pero para USB.

5.2 Diseño esquemático

Las Figuras 18 y 19 corresponden a los circuitos esquemáticos proporcionados por el fabricante para el modo HOST y esclavo en el documento "SL811HS - Interfacing an External Processor". Con un rápido vistazo a ambos circuitos puede apreciarse las diferencias. Básicamente, el HOST debe ser capaz de proporcionar una alimentación de 5 volts a través de un switch de poder, mientras que el esclavo puede utilizar los 5 volts del puerto USB como fuente de alimentación. El estándar USB obliga a incluir un condensador de 120uF a la salida de la línea de +5 volts del puerto USB-A.

Otra diferencia, aparte del tipo de conector, está en las resistencias conectadas a las líneas D+ y D- del bus USB. El Host tiene 2 resistencias de 15[kΩ] a tierra, mientras que el esclavo tiene una resistencia de 1.5[kΩ] conectada a +3.3 volts. Dependiendo de la velocidad del esclavo, la resistencia se conecta a D+ (*full-speed*) o a D- (*low-speed*). Esto permite la autodetección por parte del host de la velocidad del esclavo.

El SL811HS se alimenta con 3.3 volts, lo que significa que puede ser alimentado directamente del USB en modo esclavo, a través de un regulador de voltaje lineal de *low dropout* o baja caída. Además, sus entradas son tolerantes a 5 volts, haciéndolo compatible con microcontroladores de 5V. Los bits de control de salida son de colector abierto, pudiendo entonces conectar una resistencia de *pull-up* al voltaje de alimentación del microcontrolador. Si bien las salidas de datos son de 3.3 volt, este voltaje alcanza a ser leído como un 1 válido para lógica TTL de 5V.

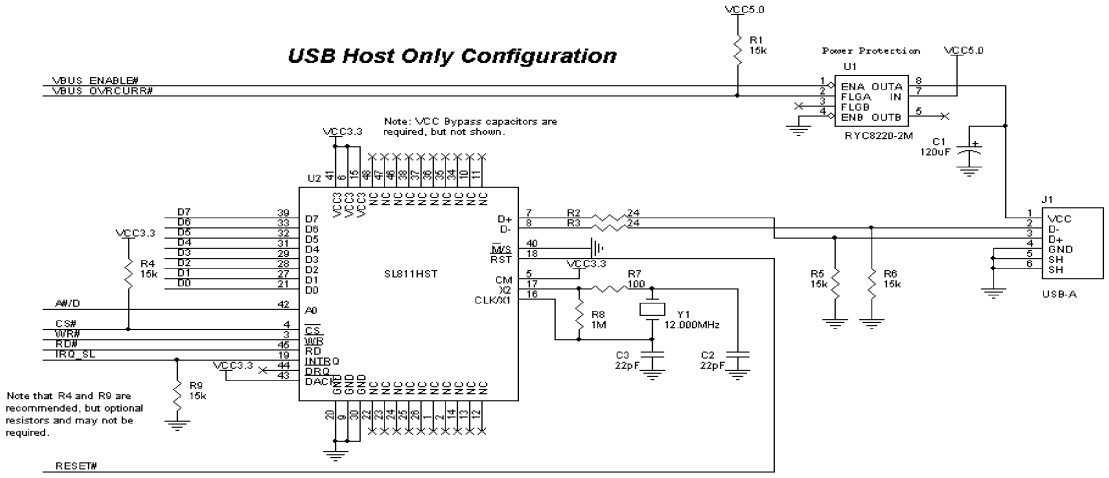


Figure 5. SL811HS in a Typical USB Host Configuration

Figura 18. Conexión en modo Host.

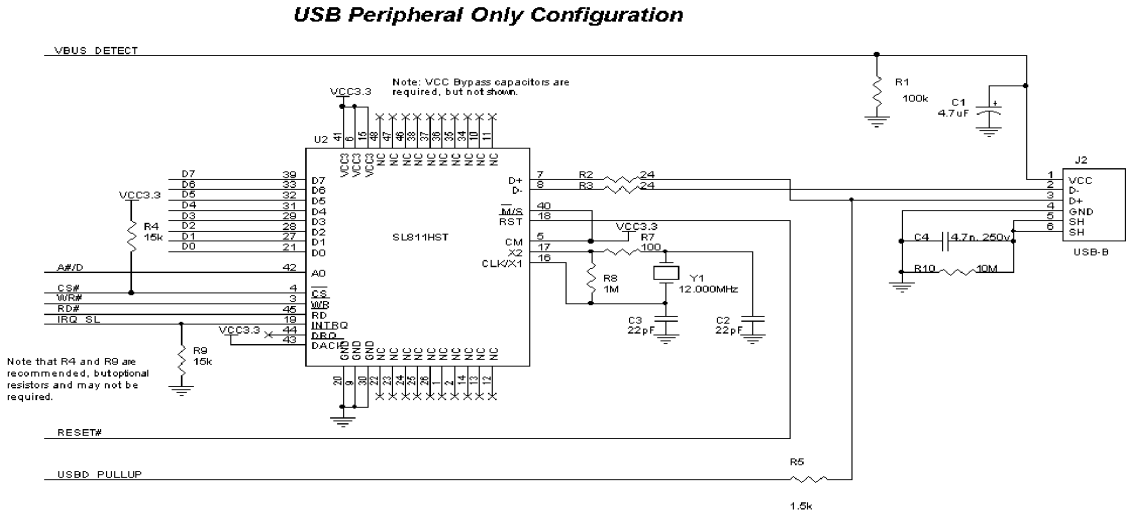


Figure 6. SL811HS/S in a Typical USB Peripheral Configuration

Figura 19. Conexión en modo esclavo.

Aprovechando la similitud entre las conexiones HOST y esclavo, se diseñó un módulo que permitiera la operación en cualquiera de los dos modos, seleccionables a través de 3 jumpers. La figura 20 presenta el circuito esquemático diseñado y la Tabla 7 contiene la lista de componentes.

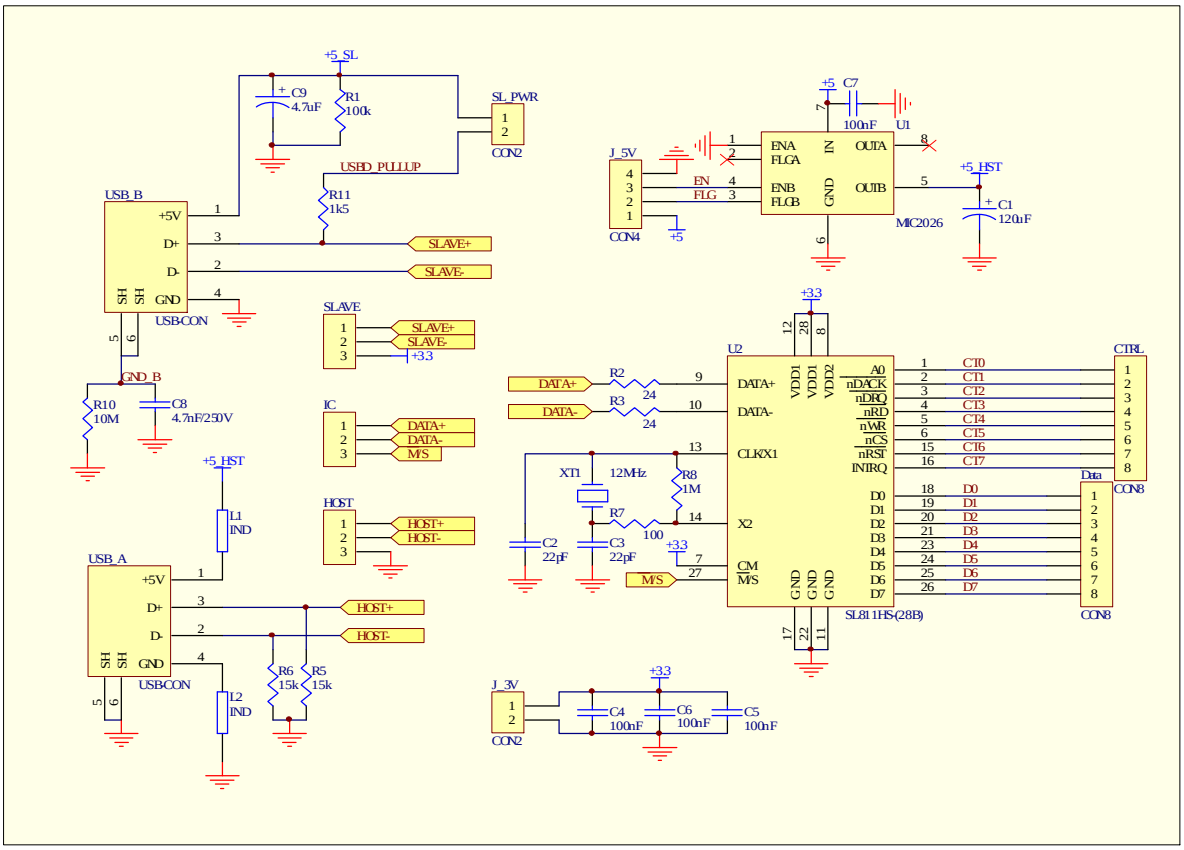


Figura 20. Circuito esquemático módulo uUSB.

Tabla 7. Lista de componentes módulo uUSB.

Cant.	Valor	Designador	Empaque	Descripción
1	1M	R8	1206	Resistencia
1	1k5	R11	1206	Resistencia
1	4.7nF/250V	C8	1206	Cond. Cerámico
1	4.7uF	C9	RB-.1/.2	Cond. electrolítico radial
1	10M	R10	1206	Resistencia
1	12MHz	XT1	XTAL-0.2D	Cristal
2	15k	R5 R6	1206	Resistencia
2	22pF	C2 C3	1206	Cond. Cerámico
2	24R	R2 R3	1206	Resistencia
1	100R	R7	1206	Resistencia
1	100k	R1	1206	Resistencia
4	100nF	C4 C5 C6 C7	1206	Cond. Cerámico
1	120uF	C1	2113_POL	Cond. Tantalio
2	IND	L1 L2	1206	Ferrita
1	MIC2026	U1	SO-8	Switch de poder 500mA
1	SL811HS-(28B)		PLCC-28	USB Host
1	USB-CON	USB_A	USB-A	Conector USB-A para PCB
1	USB-CON	USB_B	USB-B	Conector USB-B para PCB
1	PLCC-28	U2		Soquete PLCC-28 thru-hole.

5.3 Características del conector USB-A

En la Figura 21 se muestra el conector USB-A hembra para circuito impreso. Este conector de frente rectangular indica que el puerto es de tipo HOST. Los pines 1 y 4 proveen alimentación de 5 volt al periférico y suelen ser conectados a inductancias en serie para la supresión de EMI. Los pines 2 y 3 son D- y D+ respectivamente. Todo host posee una resistencia de 15[kΩ] entre estos pines y tierra. El blindaje va conectado directamente a la tierra del circuito.

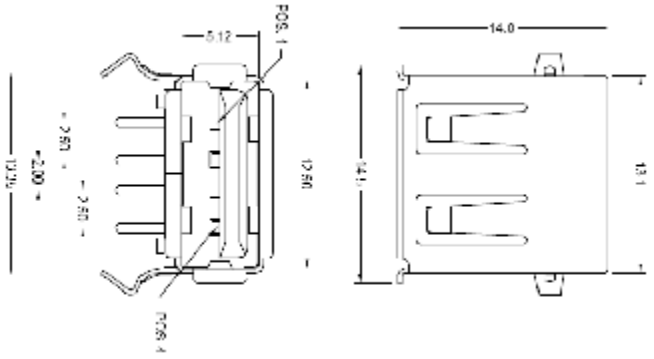


Figura 21. Conector USB-A hembra para circuito impreso.

5.4 Características del conector USB-B.

Este conector de frente cuadrado indica que es un puerto esclavo y se muestra en la Figura 22. El pin 1 corresponde a la alimentación de al menos 4.4[V], con la cuál puede ser alimentado el SL811HS y el resto del circuito. Si el circuito es alimentado de forma externa, este pin sirve para detectar la presencia de un HOST al otro extremo del cable. Este pin se encuentra disponible a través de un condensador y una resistencia conectados en paralelo cuya función es filtrar la línea para evitar la detección de falsos picos de voltaje que inducirían a una desconexión y reconexión del circuito esclavo. El pin 4 es tierra y va conectado directamente a la tierra del circuito esclavo.

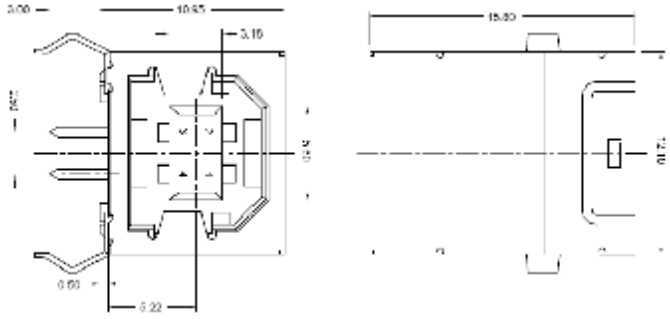


Figura 22. Conector USB-B hembra para circuito impreso.

Una resistencia de 1.5[kΩ] va conectada al pin D+. Al ser ésta conectada a +3.3[V], indica que el esclavo es un dispositivo de 12Mbps o “full speed”.
A diferencia del circuito maestro, el blindaje no va conectado directamente a tierra, sino que va conectado a través de una resistencia de 10[MΩ] en paralelo con un condensador

cerámico de 4.7[nF], como lo recomienda el fabricante. Esto porque se trata del blindaje y su función es proteger la línea de transmisión de fuentes externas de interferencia y no debe ser usado como línea de tierra de potencia.

5.5 Características del switch de potencia.

Existen en el mercado switches de potencia especialmente diseñados para el estándar USB. Son circuitos integrados que poseen 2 pines de alimentación y dos switches de potencia. La entrada a cada switch proviene directamente de la alimentación. Cada switch posee un pin de salida, un pin de habilitación, y un pin de error. Estos circuitos integrados poseen una limitación de corriente continua de 500[mA] y protección contra cortocircuito y sobrecalentamiento. Cualquier error polariza el transistor de colector abierto, conectando el pin FLG a tierra. Estos chips pueden alimentarse entre 3 y 5 [V].

La razón por la cuál poseen 2 switches se debe a que en un PC, todos los puertos HOST aparecen de a pares. Internamente, cada controlador HOST está conectado a un HUB de 2 puertos. Al conectar un nuevo esclavo USB, el HOST encuesta al dispositivo cuánta corriente consume. El esclavo retorna un byte con el consumo en unidades de 2[mA]. Si el consumo es mayor al que puede entregar ese puerto, el HOST puede desconectar la alimentación a ese dispositivo.

La elección del switch de poder se hizo buscando primero los que tenía disponibles el distribuidor MOUSER de EEUU, donde se han comprado la mayor parte de las componentes no adquiribles en Chile. El switch de potencia elegido es el MIC2026 del fabricante MICREL. Sus principales características son:

- Voltaje de operación: 2.7 a 5.5 [V].
- Resistencia máxima del switch: 140[mΩ].
- Corriente continua mínima: 500[mA] por canal.
- Protección ante cortocircuito y sobrecalentamiento.
- Disponible en empaque DIP-8 y SOIC-8.

Este módulo posee un único puerto USB-A, por lo cuál se utiliza sólo un switch. Se eligió el switch B por motivos de ruteo del circuito impreso. Los pines de alimentación y control se encuentran disponibles en la regleta de conexión. Debe recordarse conectar una resistencia de pull-up al pin FLG. También es recomendable conectar una resistencia de *pull-down* al pin ENB, ya que mientras el microcontrolador no inicialice el respectivo pin, éste estará como entrada que en la práctica es un estado de alta impedancia, quedando indeterminado el estado del switch.

La especificación USB requiere que todo puerto HOST tenga una capacitancia de salida de 120uF para la alimentación de 5V.

5.6 Características del controlador SL811HS.

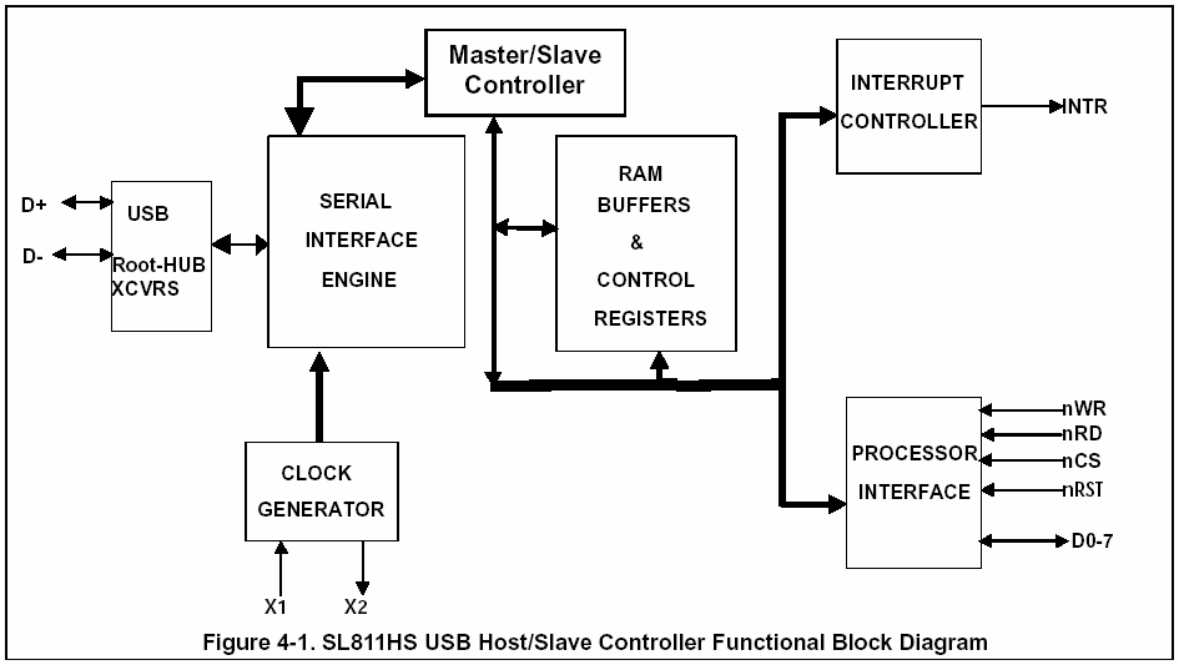


Figura 23. Diagrama interno del SL811HS.

La Figura 23, obtenida directamente del datasheet, presenta el diagrama interno del controlador USB. Los pines D+ y D- se encuentran ruteados hacia los conectores USB-A y USB-B. A través de 2 *jumper* se selecciona cuál conector se utilizará. Los pines X1 y X2 se encuentran conectados al circuito del cristal. Todas las señales de control restantes y de datos se encuentran disponibles en la regleta del módulo. Para modo HOST, las señales de DMA llamadas nDREQ y DACK no se utilizan. Existen otros 3 pines que no se ven en este diagrama: M/S, A0 y CM. Éstos serán explicados a continuación.

El pin A0 selecciona si lo que se está transfiriendo por el bus de datos es una dirección de memoria o un dato, parecido al pin RS de un display LCD inteligente. Las señales nRD y nWR son las de lectura y escritura. El pin nRST es de reset, activo en bajo. El pin nCS es de *chip select*, habilitando el funcionamiento de todas las demás señales y sacando al bus de datos de su estado de alta impedancia. Este chip posee un único pin de petición de interrupción para todas las señales internas de interrupción.

El pin M/S selecciona el modo de trabajo HOST (*master*) o esclavo (*slave*). Éste se conecta a Vcc o GND por medio de un jumper. Sin embargo, el funcionamiento como HOST o esclavo puede ser cambiado por software a través del bit 7 del registro CONTROL2 (0x0F).

El pin CM selecciona la frecuencia del cristal a utilizar como fuente de reloj. El SL811HS requiere de una señal de reloj de 48[MHz]. Ésta puede provenir de un cristal de 48[MHz] ó, gracias a un PLL interno, de un cristal de 12[MHz], que es más barato y fácil de encontrar. Además, de la Figura 24 se aprecia que el circuito recomendado para conectar un cristal de

48[MHz] es más complejo y requiere de una bobina, la cuál en empaque superficial es normalmente cara y más difícil de soldar.

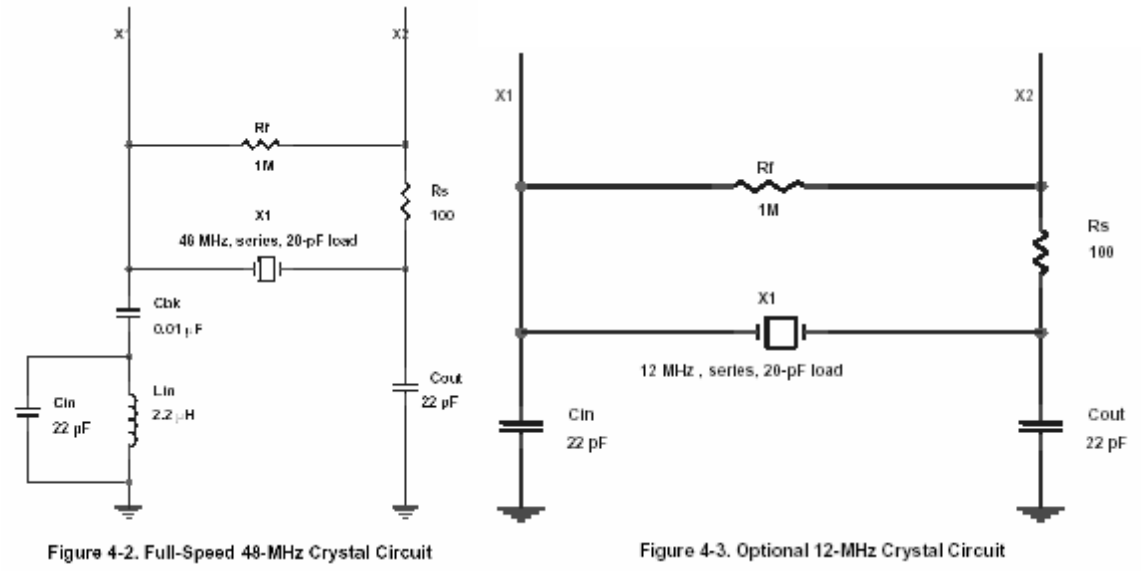


Figure 4-2. Full-Speed 48-MHz Crystal Circuit

Figure 4-3. Optional 12-MHz Crystal Circuit

Figura 24. Conexión del cristal.

Se han colocado 3 condensadores de 100[nF] lo más cercano posible a los pines de alimentación del SL811HS, ya que el PLL interno de éste es muy sensible al ruido de alta frecuencia en la alimentación. Esto está recomendado por el fabricante.

5.7 Diseño del circuito impreso.

Para hacer el módulo lo más compacto posible se utilizaron componentes de montaje superficial donde fuera posible, y se diseño el circuito impreso con componentes por ambas caras. La distancia entre las 2 regletas de conexión debía ser tal que el módulo pudiese conectarse a un protoboard, por lo cuál se partió utilizando las medidas del módulo *Header MSP430*. El *socket* PLCC-28 para el chip SL811HS impidió poder hacer el circuito impreso aún más angosto. Sin embargo, al igual que el módulo Header MSP430, éste cabe en un solo protoboard, dejando una fila a cada lado para conectar cables ó un bus de datos hacia el microcontrolador. También tiene el ancho justo para poder conectarlo entre 2 protoboards, teniendo así el máximo de 4 puntos libres por pin para el conexionado.

Debido al porte de los conectores USB, fue necesario ensanchar el circuito impreso. Esto no presenta ningún inconveniente, ya que los conectores USB se encuentran lejos de las regletas de conexión al protoboard. Además, el ensanchamiento de ese extremo del circuito impreso provee de espacio suficiente para poner las demás componentes.

Se puso especial énfasis en lograr un buen filtrado de la línea de alimentación. Para ello se trató de llevar la línea de +3.3[V] lo más directamente posible a cada pin de alimentación del SL811HS. Se posicionaron los condensadores de filtraje C4, C5 y C6 lo más cerca

posible de los pines de alimentación. Finalmente se ruteó la línea de alimentación de manera tal que una vez que pasara por el *pad* del condensador se fuera directamente al pin de alimentación y terminara en él. La idea es que ojalá cada pin de alimentación del chip estuviese conectado primero a un condensador y luego directamente al pin de alimentación de la regleta. Esto para que el condensador sólo filtre el ruido para el respectivo pin, y no deba además encargarse de filtrar el ruido de otras componentes que también estén conectadas a +3.3[V]. A continuación se muestra el ruteo de la línea de +3.3[V]. En rojo se muestra la cara superior, mientras que en azul se muestra la cara inferior.

El circuito impreso con las componentes montadas puede verse en la Figura 26. A la izquierda se encuentra la cara superior. En ella pueden verse los conectores USB-A (derecha) para la interfaz HOST y USB-B (izquierda) para la interfaz esclava. Los *jumpers* azules están puestos en configuración esclavo.

Se eligió un empaque PLCC (de 28 pines) para el chip SL811HS por dos motivos:

- El socket PLCC-28 es mucho más fácil de soldar que el chip de empaque TQFP de 48 pines, ya que este último posee un espaciamiento de apenas 10[*mil*](milésimas de pulgada) entre pines y un grosor de pines de también 10[*mil*]
- En caso de quemarse el SL811HS por conexionado, el chip es fácilmente reemplazable ya que no va soldado al circuito impreso. No existe un socket para el empaque TQFP, por lo que el chip tendría que haberse soldado directamente al circuito impreso.

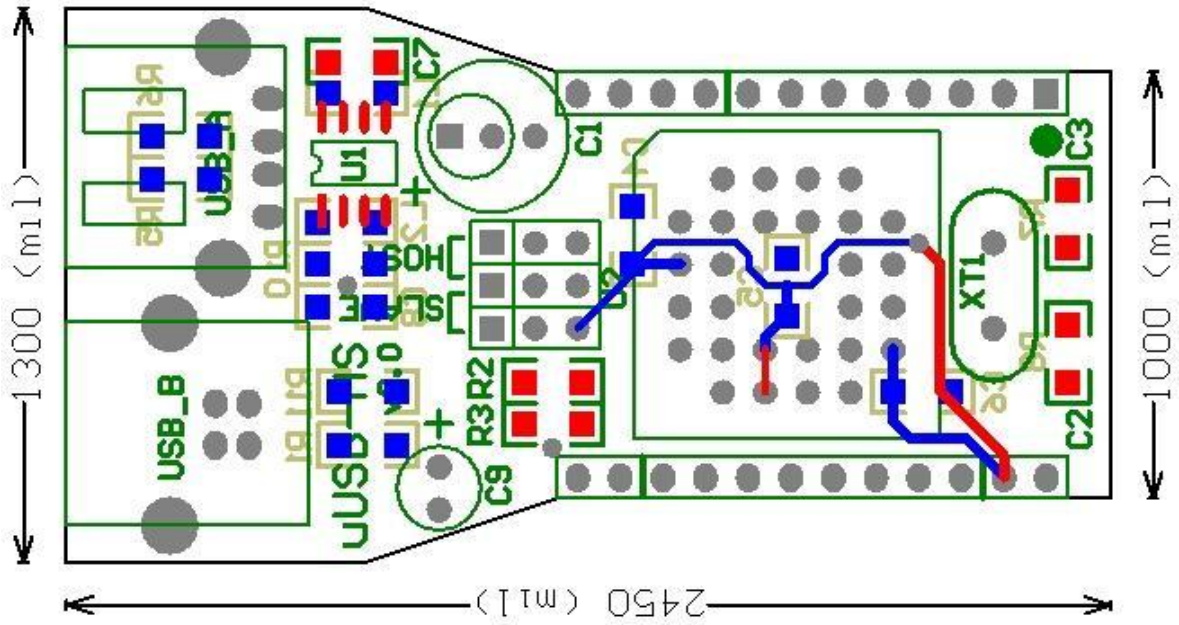


Figura 25. Línea de alimentación.

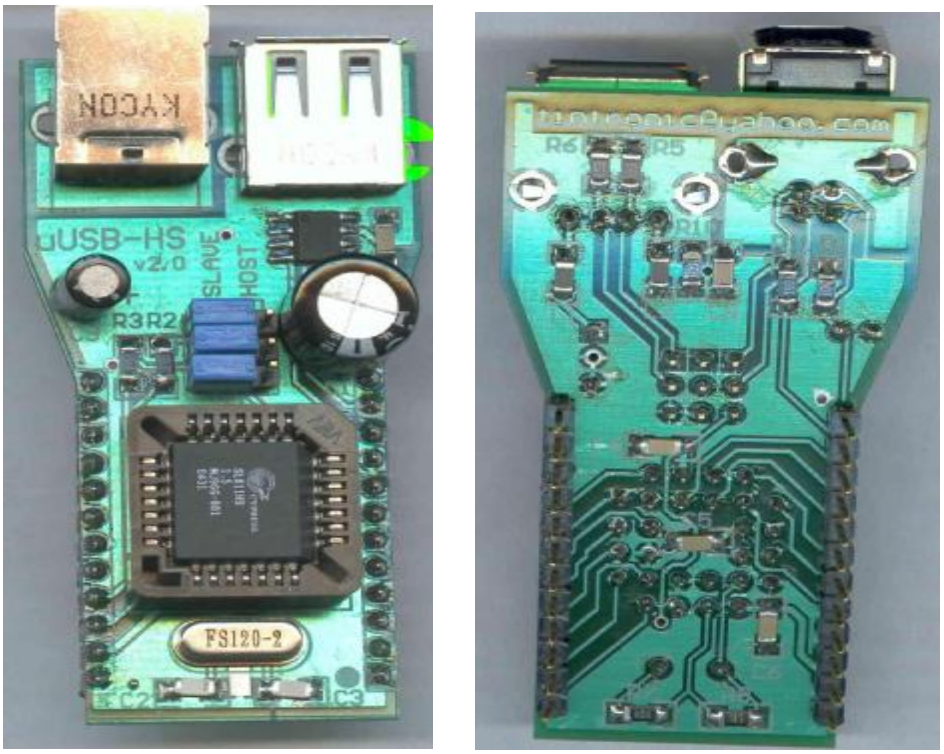


Figura 26. Módulo uUSB-HS.

5.8 Stack USB HOST

Implementar un stack USB HOST para un microcontrolador alcanza para una memoria por sí sólo y escapa al objetivo de esta Memoria. Sería el equivalente a desarrollar el stack TCP/IP *uIP*, compilado para la MSP. Se presenta un código de ejemplo que permite utilizar un teclado o mouse USB con un microcontrolador MSP430.

5.8.1 Conocimientos básicos del protocolo USB

El bus USB está pensado específicamente para la conexión de dispositivos periféricos a un computador. Por esta razón, todo bus USB se compone de un único dispositivo maestro llamado HOST, y de uno hasta 127 periféricos, incluidos Hubs. Al conectar un dispositivo esclavo a un bus USB, el Host inicia un proceso de configuración del dispositivo. En primer lugar se corre un proceso llamado enumeración, en el cuál a cada dispositivo conectado al bus le es asignado un único número que lo identifica. A diferencia de ethernet, un HUB USB no es sólo un repetidor de señal, sino un dispositivo periférico en sí que también debe ser enumerado. Al principio, todos los esclavos responden a la dirección #0.

Además de la dirección, cada dispositivo esclavo posee un número limitado de puertos, llamados *endpoints*, que pueden ser de entrada o de salida. Cada *endpoint* es visto como un buffer receptor o proveedor de datos. Entonces, la comunicación no se realiza sólo con un dispositivo, sino con un determinado *endpoint* del dispositivo.

Los tipos de paquetes que se transmiten por el bus USB son los siguientes:

- SETUP
  Indica la transferencia de un comando de configuración.
- IN
  Indica que el siguiente paquete es una lectura de datos que serán transferidos desde el esclavo hacia el host.
- OUT
  El siguiente paquete será de datos desde el host hacia el esclavo.
- SOF/EOF
  Start of Frame. Paquete de inicio de FRAME. Es transmitido una vez cada 1[ms] para evitar que los esclavos entren en modo de bajo consumo. Posee un índice de tiempo que cicla cada 2048[ms] para coordinar dispositivos de tiempo real. EOF es la versión de baja velocidad del SOF.
- Preaamble
  Es transmitido para indicar que la siguiente transferencia será con un dispositivo de baja velocidad.
- ACK
  Indica que el último comando fue reconocido y ha comenzado a ser ejecutado. *No* indica que ya se terminó de procesar.
- NAK
  El destinatario no pudo procesar el paquete debido a que se encuentra ocupado. La operación deberá ser efectuada nuevamente. Esto no se hace de forma automática.
- STALL
  El comando no es reconocido. Es equivalente a un STOP, pero el dispositivo esclavo puede seguir funcionando.
- DATA0 ó DATA1
  Estos paquetes son transmitidos después de un paquete de comando IN, OUT o SETUP y contiene los datos referentes a esos comandos. Al enviar o recibir datos, se transmiten alternadamente como paquete DATA0 ó DATA1, para así saber si se ha perdido un paquete entremedio.

Toda transferencia es iniciada por el HOST, el cuál envía un paquete llamado TOKEN. Éste contiene el tipo de transferencia (SETUP/IN/OUT/SOF/EOF), la dirección y puerto (***endpoint***) de destino y un CRC5 del paquete. Luego se transfiere un paquete de datos, ya sea desde (OUT) o hacia (IN) el HOST. Finalmente se envía un paquete de ***handshake*** (ACK,NACK,STALL) que contiene el resultado de la operación.

Es importante hacer notar que el NACK no se envía en respuesta a un paquete que ha llegado con errores, sino que indica que el paquete no pudo ser procesado porque el dispositivo se encuentra ocupado. Si un paquete llega con errores, el dispositivo simplemente lo descarta y no envía respuesta alguna.

El HOST transmite un TOKEN SOF (***Start of Frame***) cada 1[ms] indicando el comienzo de cada ***frame***. Por lo tanto, cada ***frame*** dura 1[ms]. Toda transferencia debe realizarse dentro de un mismo frame y no puede pasarse a otro.

La revisión 1.1 del protocolo USB establece 2 velocidades de comunicación:

- Velocidad completa o *Full-Speed*: La comunicación se efectúa a 12Mbps. Utilizada para dispositivos rápidos como discos duros, scanners, dispositivos de audio y video.
- Velocidad baja o *Low Speed*: La comunicación se efectúa a 1.5Mbps y debe ir presidida por un *preamble* para indicar que el paquete a continuación es de baja velocidad. Es utilizada básicamente por dispositivos de interfaz hombre-maquina, como mouses y teclados.

Existen 4 tipos de transferencia:

1. *Bulk Transfers*

   Son transferencias tipo ráfaga, cuyo paquete de datos contiene 8, 16, 32 ó 64 bytes. Puede ser de tipo IN (slave a host) o de tipo OUT (host a slave). Cada transferencia se compone de 3 paquetes: *Token* (IN ó OUT), *Payload Data* (datos) y *Handshake* (ACK, NACK ó STILL). La Figura 27 detalla este tipo de transferencia.

   El protocolo USB tuvo muy en mente el concepto de errores de transmisión, tratándolos como un hecho de la vida en vez de un caso fortuito. Por esta razón, además del CRC16, se utiliza un mecanismo llamado *Data Toggle*, alternando el ID de paquete o *PID* entre DATA0 y DATA1, cada vez que se transfiere un paquete de datos.

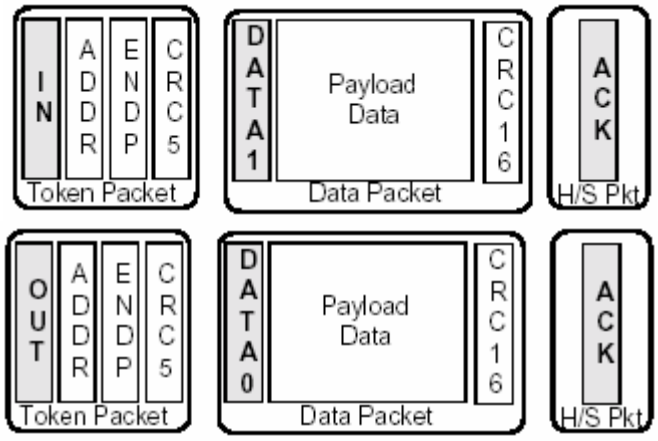


Figura 27. Transferencia tipo Bulk.

2. *Interrupt Transfers*

   Este tipo de transferencia fue agregado en la revisión 1.1 del protocolo USB. Es parecido al *bulk transfer*, pero sólo existe para paquetes tipo IN. Pueden transferirse de 1 a 64 bytes de datos. Debido a que toda comunicación es iniciada por el HOST, el esclavo no puede enviar una señal de interrupción. En vez de ello, los *endpoints* de tipo interrupción tienen asociado un intervalo de encuestamiento o *ping*. De esta forma, el Host efectúa un *polling* del estado de interrupción del paquete a intervalos fijos, simulando una interrupción desde el esclavo. Este paquete se detalla en la Figura 28.

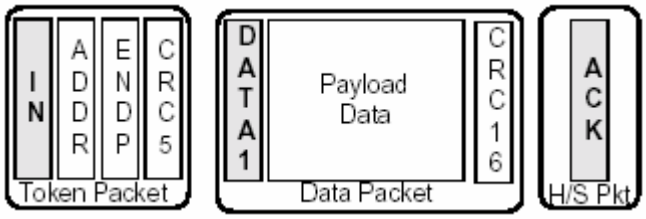


Figura 28. Transferencia tipo interrupción.

3. Isochronous Transfers

Transferencia isocrónica es utilizada para aplicaciones de tiempo real, como audio y video. El bajo retardo y un ancho de banda asegurado son las características más importantes en este tipo de aplicaciones. Para ello se reserva una cierta cantidad de ancho de banda para transferencias isocrónicas, las cuales se realizan al principio de cada frame (después de un paquete SOF). Para disminuir el *overhead*, este tipo de transferencias no posee un paquete de *handshake* (ACK,NACK,STALL), no implementa el mecanismo de *data toggle* (siempre es DATA0) ni posee retransmisiones. El único mecanismo de detección de errores que posee es el CRC16 intrínsico a un paquete de datos. El máximo largo son 1024 bytes de datos. Esta transferencia se detalla en la Figura 29.

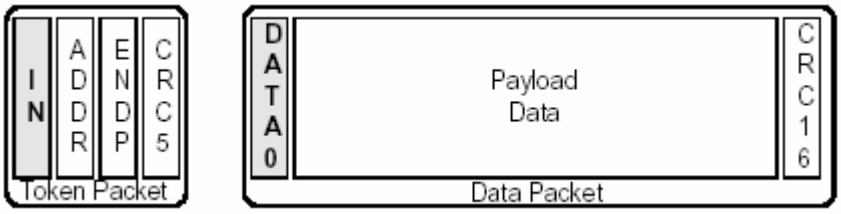


Figura 29. Transferencia tipo Isocrónica.

4. Control Transfers

Este tipo de transferencias se utiliza para controlar y enviar comandos a los dispositivos esclavos. Debido a su extrema importancia, este tipo de transacción emplea la mayor verificación de errores que el protocolo USB puede ofrecer. Las transferencias de control son ejecutadas de acuerdo a una "ley del mejor esfuerzo", proceso de 6 pasos definidos en la sección 5.5.4 de la revisión 1.1 del estándar USB.

Una transferencia de control consta de 2 ó 3 pasos. Primero se envía una *transferencia* de SETUP, en la cuál se envían 8 bytes de datos de control. En caso de que se necesiten enviar más datos, se ejecuta una nueva transferencia tipo IN. Finalmente se ejecuta una transferencia de estado, en la cual el dispositivo esclavo confirma la ejecución satisfactoria de un comando de control. Este set de transferencias se detalla en la Figura 30.

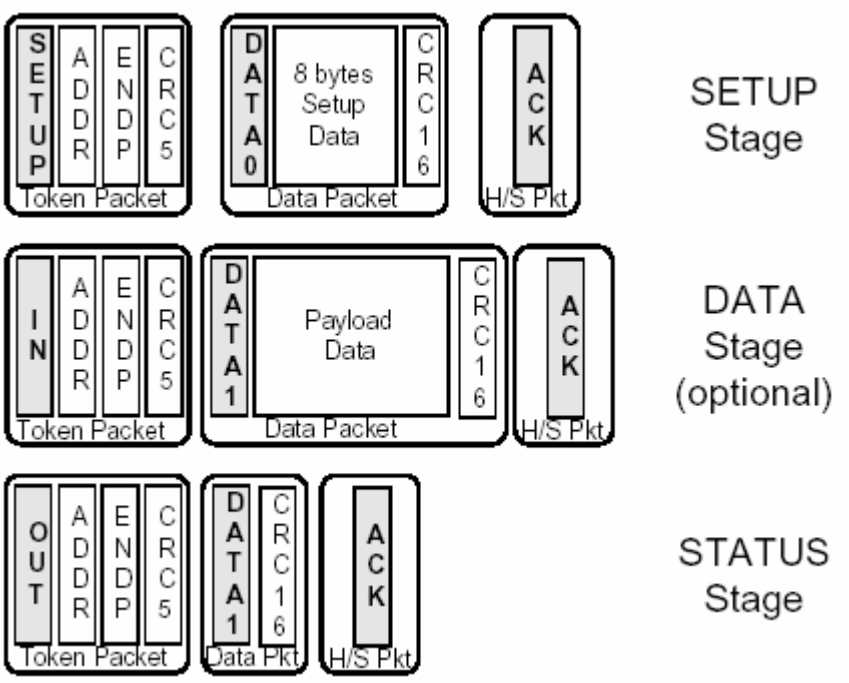


Figura 30. Transferencia de Control.

5.8.2 Funcionamiento del controlador Host/Slave SL811HS de Cypress.

El controlador SL811HS posee un bus de 8 bits compartido para direcciones y datos. El tipo de información a transferir se elige con el pin A0. Cuando A0=0, se accesa a un puntero el cuál contiene la dirección de memoria a escribir/leer. Cuando A0=1, se accesa a la información contenida en la dirección del puntero. Al leer o escribir cualquier dirección de memoria, el puntero se incrementa automáticamente. Sin embargo, un documento de Erratas del SL811HS publicado por Cypress, declara que el puntero podría no incrementarse, por lo cuál no es recomendable utilizar la propiedad de autoincremento del puntero.

Debido a que el puntero es de 8 bits, el tamaño de memoria del SL811HS es de 256 bytes. Los primeros 16 bytes corresponden a registros de configuración. Los siguientes 240 bytes corresponden al buffer de datos para la comunicación USB. Este buffer posee un puntero que indica la posición del primer byte dentro del buffer de los datos a enviar o recibir. Así pueden armarse varios paquetes a la vez, o armarse un paquete mientras otro se está transfiriendo, ya que la mayoría de los paquetes de datos son de largo máximo 64 bytes (de datos).

El SL811HS posee un único pin de petición de interrupción, el cuál se pone en uno cuando se activa cualquiera de sus banderas de interrupción, las cuáles pueden ser enmascaradas por software.

Además de la interfaz lógica, el SL811HS efectúa muchas de las funciones del protocolo USB por sí sólo:

- Generación automática de paquetes de inicio de frame (SOF/EOF).
- Generación automática de *preamble* para paquetes de baja velocidad.
- Cálculo automático del CRC5 para paquetes TOKEN.
- Cálculo automático de CRC16 para paquetes de datos.

5.8.3 Programación del SL811HS con un microcontrolador MSP430.

El conexionado utilizado incluye una MSP430F1612, un Módulo "uUSB-HS v2.0" y un display LCD inteligente de 4 líneas por 20 caracteres utilizado en modo bidireccional. El diagrama esquemático del conexionado se presenta en la Figura 31.

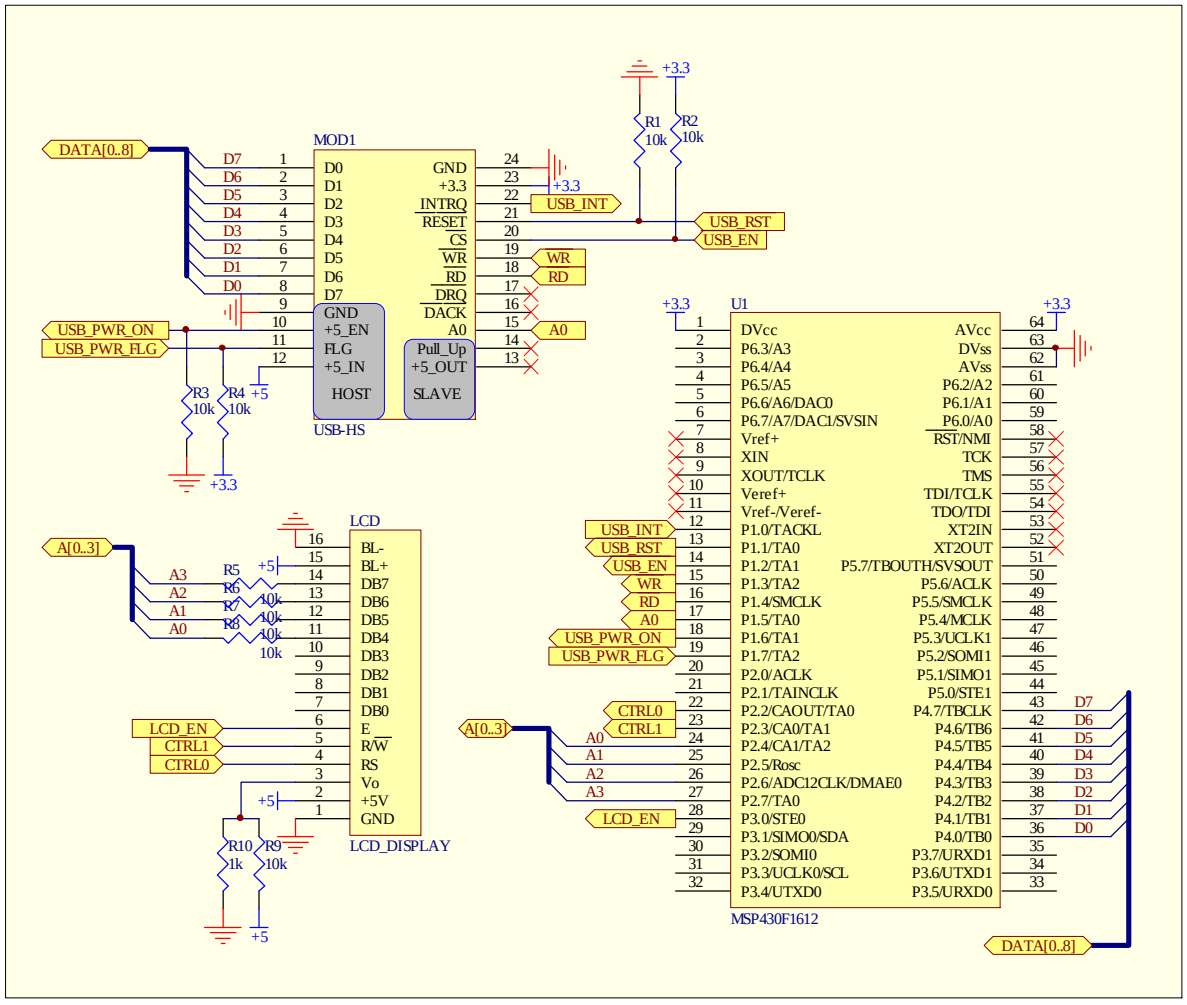


Figura 31. Conexión módulo uUSB-HS con MSP430 y LCD.

Primero deben definirse las funciones básicas que leen y escriben un registro del SL811HS. Para ello se han definido los pines de comunicación entre la MSP y el SL811HS de la siguiente forma:

```
//Bus de datos/control
#define USB_IN  P4IN
#define USB_OUT P4OUT
#define USB_SEL P4SEL
#define USB_DIR P4DIR
//Pin de interrupción. Asignación exclusiva.
#define USB_INT P1IN&BIT0  //Active High
#define USB_INT_B    BIT0  //Active High
#define USB_INT_init P1DIR&=~BIT0;P1SEL&=~BIT0;  // IN; DigitalIO;
#define USB_INT_EN   P1IE|=BIT0;P1IES&=~BIT0;
//pin de reset. Asignación exclusiva.
#define USB_RST_ON P1OUT|=BIT1
#define USB_RST_OFF P1OUT&=~BIT1
#define USB_RST_init P1DIR|=BIT1;P1SEL&=~BIT1;   // OUT; DigitalIO;
//pin de ENABLE o CHIP_SELECT. Asignación exclusiva.
#define USB_EN_ON P1OUT|=BIT2
#define USB_EN_OFF P1OUT&=~BIT2
#define USB_EN_init P1DIR|=BIT2;P1SEL&=~BIT2;    // OUT; DigitalIO;
//Pin de escritura. Multiplexable.
#define USB_WR_ON P1OUT|=BIT3
#define USB_WR_OFF P1OUT&=~BIT3
#define USB_WR_init P1SEL&=~BIT3;     // DigitalIO;
#define USB_WR_OUT P1DIR|=BIT3;
#define USB_WR_IN P1DIR&=~BIT3;
//Pin de lectura. Multiplexable.
#define USB_RD_ON P1OUT|=BIT4
#define USB_RD_OFF P1OUT&=~BIT4
#define USB_RD_init P1SEL&=~BIT4;     // DigitalIO;
#define USB_RD_OUT P1DIR|=BIT4;
#define USB_RD_IN P1DIR&=~BIT4;
//Pin de dirección. Selecciona DATO o REGISTRO. Multiplexable.
#define USB_A0_ON P1OUT|=BIT5
#define USB_A0_OFF P1OUT&=~BIT5
#define USB_A0_init P1SEL&=~BIT5;     //OUT; DigitalIO;
#define USB_A0_OUT P1DIR|=BIT5;
#define USB_A0_IN P1DIR&=~BIT5;
//Pin de habilitación del switch de poder hacia el esclavo
#define USB_PWR_ON P1OUT|=BIT6
#define USB_PWR_OFF P1OUT&=~BIT6
#define USB_PWR_init P1DIR|=BIT6;P1SEL&=~BIT6;   //OUT; DigitalIO;
//Pin de error del switch de poder
#define USB_PWR_FLG   P1IN&BIT7  //Active Low
#define USB_PWR_FLG_init P1DIR&=~BIT7;P1SEL&=~BIT7;    //IN;
DigitalIO;

//USB_SEL First makes bits WR RD and A0 outputs. Then it enables
SL811HS
#define    USB_SELECT    USB_WR_ON;USB_WR_OUT;USB_RD_ON;USB_RD_OUT;
USB_A0_OUT;USB_EN_OFF;

//USB_DES First disables SL811. Then it turns bits WR RD and A0
back to inputs (for multiplexing)
```

```
#define USB_DESELECT USB_EN_ON;USB_WR_IN;USB_RD_IN;USB_A0_IN;
```

Ahora la definición de los 16 registros:

```
#define    CTL   0x00  // write this register to kick off a transfer
#define    BUFADR     0x01  // start of internal data buffer
#define    BUFLEN     0x02  // length of internal buffer
#define    PID_EP     0x03  // name when written--PID and Endpoint
for next xfr
#define    PKTSTAT    0x03  //  name  when  read--status  of  last
transfer
#define    FNADDR     0x04  //  name  when  written--USB  function
address
#define    CTL1       0x05  // more control stuff
#define    INTSTATUS  0x0D  // Interrupt request status bits. We use
DONE and SOF.
#define    SOFCT_L    0x0E  // SOF (EOP) time constant low byte
#define    SOFCT_H    0x0F  // name when written--EOP time constant
high byte
```

Algunas definiciones para los datos a escribir en registros.

```
#define    IN_PID     0x90  // PID (Packet ID) constants
#define    SETUP_PID  0xD0  // for the 'set address' request
#define    SOF_PID    0x05  // constants for 811 CTL1 register
#define    USB_RESET  0x08  // SIERES=1
#define    USB_OPERATE     0x21  //Low Speed=1(b5),SOF(EOP)EN=1(b0)
```

Finalmente las funciones básicas.

```
void wr811(BYTE a, BYTE d)
  {
      USB_SELECT;      //Make SL811 ready to communicate with.

      USB_DIR=0xFF;    //Data port to outputs
      USB_OUT=a;
      USB_A0_OFF;      //Write address to pointer
      USB_WR_OFF;
      USB_WR_ON;

      USB_OUT=d;
      USB_A0_ON;       //Write data to register
      USB_WR_OFF;
      USB_WR_ON;
      USB_DIR=0x00;    //Data port to inputs (for multiplexing)

      USB_DESELECT;    //Liberate used signals
  }

BYTE rd811(BYTE a)
  {
      unsigned char d;
      USB_SELECT;      //Make SL811 ready to communicate with.
```

```
      USB_DIR=0xFF;    //Data port to outputs
      USB_OUT=a;
      USB_A0_OFF;      //Write address to pointer
      USB_WR_OFF;
      USB_WR_ON;

      USB_DIR=0x00;    //Data port to inputs
      USB_A0_ON;       //Read data from register
      USB_RD_OFF;
      d=USB_IN;        //Read DATA
      USB_RD_ON;

      USB_DESELECT;    //Liberate used signals
      return d;
  }

void addr811(BYTE a)
{
      USB_SELECT;      //Make SL811 ready to communicate with.

      USB_DIR=0xFF;    //Data port to outputs
      USB_OUT=a;
      USB_A0_OFF;      //Write address to pointer
      USB_WR_OFF;
      USB_WR_ON;

      USB_DIR=0x00;    //Data port to inputs
      USB_DESELECT;    //Liberate used signals
}
```

La rutina de inicialización del chip SL811HS retorna el número de versión de hardware del SL811HS:

- 00h: SL11
- 01h: SL811 rev. 1.2
- 02h: SL811 rev. 1.5

La primera función que debe ejecutarse es una inicialización de los pines de comunicación, ejecutar un reset del chip SL811HS y obtener el número de versión, para así asegurarse de que el SL811HS está correctamente conectado.

```
unsigned char USB_init(void)
  {
      unsigned char rev;
      USB_RST_OFF;     //SL811 Reset enabled.
      USB_RST_init;
      USB_EN_ON;       //SL811 Disabled.
      USB_EN_init;
      USB_PWR_OFF;     //Power to USB-A disconnected.
      USB_PWR_init;
```

```
    USB_WR_init;
    USB_RD_init;

    USB_INT_init;         //interrupt input;
    USB_PWR_FLG_init;     //USB_+5V Error (overload) input

    Delayx100us(250);     //Delay 25ms.
    USB_RST_ON;           //Disable Reset.

    USB_SEL=0x00;         //DigitalIO
    rev=( (rd811(0x0E)&0xF0) >>4 );  //Get HW rev. number
    USB_PWR_ON;           //Enable +5V to USB Slave.
    return rev;
  }
```

Para enviar un comando USB y esperar hasta que la operación haya sido terminada (comando + datos + ack), se utiliza la función `go()`, la cuál retorna el registro de estado STATUS.

```
BYTE go(BYTE cmd)                 // Launch an 811 operation.
  {
    wr811(INTSTATUS,0x01);// clear the DONE bit
    wr811(CTL,cmd);       // start the operation
    Delayx100us(1);
    while(rd811(INTSTATUS) & 0x01 == 0)
    {;}                   // spin while "done" bit is low
    return rd811(PKTSTAT);// return the status of this transfer
  }
```

El siguiente ejemplo asume que hay un teclado USB conectado directamente al módulo USB HOST al encender la MSP, y que no hay HUB de por medio. Antes de utilizar un dispositivo USB, éste debe ser ennumerado y de ser necesario, configurado. Para ello, primero se envía un reset del bus USB, que es distinto a un reset del SL811HS. Un reset USB se envía a un puerto específico de un HUB o en este caso al dispositivo conectado directamente al módulo USB. Un reset USB causa un reset en el dispositivo esclavo, el cuál responde ahora a la dirección #0 del bus USB.

Primero se reservan 8 bytes del buffer del SL811HS para datos. Esto se hace estableciendo primero la dirección donde se guardarán los datos a ser enviados. Como se vio anteriormente, el buffer comienza en la dirección 0x10, por lo que se eligió esta dirección, escribiendo el registro BUFADR(0x00). Luego se escribe el largo del paquete en el registro BUFLEN(0x02).

```
wr811(BUFADR,0x10);   // start of SETUP/IN internal data buffer
wr811(BUFLEN,0x08);   // reserve 8 bytes
```

Luego se genera una condición de reset del bus USB durante unos 25[ms], para dar tiempo al esclavo de resetearse.

```
// (1) Reset the USB device. This makes it listen to address 0.
wr811(CTL1,USB_RESET);          // Speed=1(L), JKFORCE=1, SOFEN=1
Delayx100us(250);               // about 18 milliseconds
wr811(CTL1,USB_OPERATE);
```

Luego se habilita la generación automática de paquetes SOF. Como ya se explicó, estos paquetes se envían una vez cada 1[ms] al inicio de cada frame. De esta manera se evita que el esclavo entre en modo suspendido. El valor de SOF se basa en una señal de reloj de 12MHz, por lo cuál el valor correcto es 12000, ó 0x2EE0. Debe tenerse cuidado al modificar el registro SOFCT_H (0x0F), ya que los bits 6 y 7 de éste configuran la inversión de polaridad D+/D- y el modo Master(HOST)/Slave, respectivamente. El pin de selección M/S configura este bit (bit 7 del registro 0x0F) cuando el SL811HS sale de reset. Luego el pin M/S es ignorado y sólo cuenta el estado del mencionado bit.

```
// Enable sending 1 msec EOP's (the low-speed version of SOF's)
wr811(SOFCT_L, 0xE0);       // Set the SOF generator time constant
wr811(SOFCT_H, 0x2E | 0xC0); // 1 ms SOF rate, b7=HOST, b6=POLSWAP
```

Luego se enumeran los dispositivos USB. Como se asume de antemano que ya existe un único teclado conectado directamente al Módulo USB, puede asignársele la dirección #1, sin necesidad de preguntar primero qué tipo de dispositivo es para buscar el driver adecuado. La enumeración se realiza enviando un paquete tipo SETUP, con el comando set_address, especificado en el capítulo 9 del estándar USB rev1.1. Para ello debe primero llenarse el buffer del SL811HS con los 8 bytes de datos con los campos del comando set_address de la siguiente forma:

```
//    (2) Issue a SET_ADDRESS USB request, setting the peripheral
//    address to 1 From the USB spec, Chapter 9, here is the data
//    for a "SET_ADDRESS" request:
wr811(0x10,0x00);     // bmRequestType (h->d,std request,device is
                      recipient)
wr811(0x11,0x05);     // bRequest (SET_ADDRESS)
wr811(0x12,0x01);     // wValueL   (device address)--we're setting
                      it to ONE
wr811(0x13,0x00);     // wValueH  (zero)
wr811(0x14,0x00);     // wIndexL  (zero)
wr811(0x15,0x00);     // wIndexH  (zero)
wr811(0x16,0x00);     // wLengthL (zero)
wr811(0x17,0x00);     // wLengthH (zero)
```

Debido a que el teclado acaba de ser reseteado, éste responde a la dirección #0, por lo tanto el comando set_address se envía a esa dirección. Una vez armado el paquete, éste es enviado a través de la función go, cuyo resultado debe ser un ACK, de lo contrario significa que el teclado no está conectado.

```
wr811(FNADDR,0x00);             // USB address zero
wr811(PID_EP,SETUP_PID | 0x00); // SETUP PID, EP0
result=go(0x07);                // DIREC=1(out), ENAB=1, ARM=1
```

Una vez que el teclado ha recibido satisfactoriamente el comando set_address, envía de vuelta un ACK y recién entonces comienza a configurarse para responder a la dirección #1. Es importante notar que el ACK no significa que haya terminado la ejecución del comando enviado, por el contrario, significa que ha reconocido el comando y va a comenzar a ejecutarlo.

Luego se le pide un paquete de datos al dispositivo, el cuál contiene el estado de éste. En la configuración de los dispositivos HID, éste es un paquete sin datos. Si el teclado responde con un NACK, significa que aún está procesando el comando anterior, por lo que el último comando debe ser reenviado hasta que el teclado responda con un ACK.

```
result = 0;
while( !(result & 0x01) )
  {
     // STATUS stage is a no-data IN to EP0
     wr811(PID_EP,IN_PID | 0x00);// IN PID, EP0
     result=go(0x03);// Don't sync to SOF, DIREC=0(in), ENAB, ARM
     //display(result);
  }
```

Ahora el teclado debe ser configurado. Si no se envía el comando de configuración, el teclado se encontrará en un estado “desconfigurado” y probablemente no retorne ningún dato.

```
//    (3) Send a CONTROL transfer to select configuration #1. Until
//    we do this the device is in an "unconfigured" state and
//    probably won't send data.
//    Again, from USB spec Chapter 9:
wr811(0x10,0x00);     // bmRequestType (h->d,std request,device is
                      recipient)
wr811(0x11,0x09);     // bRequest (SET_CONFIGURATION)
wr811(0x12,0x01);     // wValueL  (configuration = 1)
wr811(0x13,0x00);     // wValueH  (zero)
wr811(0x14,0x00);     // wIndexL  (zero)
wr811(0x15,0x00);     // wIndexH  (zero)
wr811(0x16,0x00);     // wLengthL (zero)
wr811(0x17,0x00);     // wLengthH (zero)

wr811(FNADDR,0x01);   // now talking to USB device at address 1
wr811(PID_EP,SETUP_PID | 0x00);  // OR in the endpoint (zero)
result=go(0x07);      // DIREC=1(out), ENAB=1, ARM=1
// STATUS stage is a no-data IN to EP0
wr811(PID_EP,IN_PID | 0x00);     // IN PID, EP0
result=go(0x03);                 // DIREC=0(in), ENAB=1, ARM=1
```

Ahora el teclado ya se encuentra configurado y podemos iniciar una lectura periódica. El SL811HS generará automáticamente un EOF cada 1[ms] y pondrá al mismo tiempo la bandera EOF del registro STATUS en 1. Esta bandera puede ser utilizada como una señal de reloj de 1[kHz]. Una lectura típica de una tecla se hace cada 4 a 20 [ms]. En este

ejemplo se hará una lectura cada 4 banderas EOF, o sea, cada 4 [ms]. Una vez leída la bandera, ésta debe ser sobreescrita con un 1 para ser borrada.

```
wr811(PID_EP,IN_PID | 0x01);  // set up for IN PIDS to endpoint 1
while(1)
  {
     addr811(0x10);//reset pointer to beginning of internal buffer
     waitframes(4);  // send the IN requests every n milliseconds
     result=go(0x03);           // DIREC=0(in), ENAB=1, ARM=1
     if (result & 0x01)         // look only for ACK
     decode();
  }
```

Se han creado 2 nuevas funciones. `waitframes(n)` espera a que pasen `n` frames. `decode()` hace la decodificación del paquete de 8 bytes retornado por el teclado.

```
void waitframes(BYTE num)
  {
     int j=0;
     BYTE result;
     while (j < num)
       {
          wr811(INTSTATUS,0xFF);// clear the int request flags
          while (1)
            {
               result = (rd811(INTSTATUS)); // hang while SOF flag
                                                 low
               result &= 0x10;             // SOF flag is bit 4
               if (result == 0x10) break;
               }
          j++;
       }
     Delayx100us(1);
  }

void decode(void)
  {
     unsigned char i,len;
     len=rd811(BUFLEN);    //number of bytes returned by keyboard
     for(i=0;i<len;i++)    //copy data to local buffer
          USB_BUF[i]=rd811(0x10+i);
     SEND_CMD(LINE3);
     for(i=2;USB_BUF[i]>0;i++)  //print decimal value
          printf("%02d ",(int)USB_BUF[i]);
     for(;i<8;i++)
          printf("   ");  //clear rest of line
     SEND_CMD(LINE4);
     for(i=2;USB_BUF[i]>0;i++)  //print table character
          printf(" %c ",tec2char(USB_BUF[i]));
     for(;i<8;i++)
          printf("   ");  //clear rest of line
```

```
    SEND_CMD(LINE2);
    printf(" ");
    for(i=0x80;i>0;i>>=1) //print flags of byte 0 from msb to lsb
      {
         if(USB_BUF[0]&i)
              SEND_CHAR(0xFF);//print black rectangle
         else
              SEND_CHAR(' '); //print blank space
      }
    printf("  ");

    for(i=0x80;i>0;i>>=1) //print flags of byte 0 from msb to lsb
      {
         if(USB_BUF[1]&i)
              SEND_CHAR(0xFF);//print black rectangle
         else
              SEND_CHAR(' '); //print blank space
      }
}
```

Un rápido vistazo al paquete enviado por el teclado durante un par de pruebas muestra que las teclas pulsadas aparecen desde el byte 2 al 7, dando un máximo absoluto de 6 teclas que pueden ser presionadas simultáneamente. Todas las teclas retornan un código de posición, exceptuando las teclas Ctrl, Alt, Shift y Windows, las cuáles aparecen como banderas en el byte 0, 4 para las izquierdas y 4 para las derechas. El orden en el que aparecen las hasta 6 teclas es aleatorio según la especificación para dispositivos HID. Quien haya trabajado con teclados PS/2 se dará cuenta que su funcionamiento es totalmente distinto a un teclado USB.

Recuérdese que un teclado de PC es leído de forma matricial por el controlador embebido en él, ya sea éste PS/2 ó USB. Por esta razón existe también un máximo de teclas por sector que pueden ser presionadas al mismo tiempo. Si el límite es excedido, el teclado USB responde con un código de error, poniendo los 6 bytes de teclas en 0x01.

La función `decode()` está hecha para un display de 4 líneas por 20 caracteres. En la 2ª línea se despliegan los ***flags*** del byte 0, representado las teclas Shift, Alt, Ctrl y Windows derecha e izquierda. La 3ª línea muestra los números de las teclas presionadas, mientras que la 4ª línea utiliza la función `tec2char()` para convertir el número de tecla en el caracter ASCII correspondiente. De esta manera puede implementarse rápidamente una conversión número de tecla a caracter, dependiendo de cuáles quiera implementarse en el microcontrolador. Además, debe tenerse en cuenta que la función `tec2char()` debe ser implementada para el teclado que se está utilizando. Un teclado inglés es distinto a uno en español.

A continuación se muestra una implementación matemática de conversión de tecla a caracter, la cual es preferible a utilizar una tabla; puesto que una conversión matemática requiere menos memoria y la diferencia en tiempo de procesamiento es poca para la

mayoría de las teclas. Podría implementarse una mezcla de conversión matemática para números y letras, y conversión por tabla para signos de puntuación.

Esta función `tec2char()` retorna el código ASCII para las letras mayúsculas y los números que se encuentran sobre las letras y en el pad numérico de la derecha. Esta función puede ser ampliada por ejemplo para incluir la tecla Shift y diferenciar entre letras mayúsculas y minúsculas.

```
char tec2char (unsigned char c)
  {
     if( (c>=4) & (c<=29) )
          return (c+61);
     else if( (c>=89) & (c<=97) )
          return (c-40);
     else if(c==98)
          return '0';
     else if( (c>=30) & (c<=38) )
          return (c+19);
     else if(c==39)
          return '0';
     else
          return 0;
  }
```

5.8.4 Otros Ejemplos

En el CD se suministra un 2º ejemplo el cual detecta la conexión/desconexión de un dispositivo esclavo al puerto USB. Sólo detecta el cambio en la conexión del dispositivo y configura la interfaz del SL811HS apropiadamente, pero no inicializa el dispositivo conectado. Este ejemplo utiliza las opciones de interrupción que provee el SL811HS. Al detectar un cambio en el estado de la conexión, la MSP muestra el nuevo estado en un display LCD. Los estados son:

- Waiting: Ningún dispositivo está conectado.
- Full Speed: Hay conectado un dispositivo de velocidad completa.
- Low Speed: Hay conectado un dispositivo de baja velocidad.

5.8.5 Conexión de Mouse

El mismo programa para leer un teclado USB puede utilizarse para leer un mouse USB. La diferencia radica exclusivamente en la interpretación de los bytes del paquete de respuesta. El mouse responde con 3 bytes que se describen a continuación:

- Byte 0: Reporte de botones:
  - Bit 0: Botón Primario
  - Bit 1: Botón Secundario
  - Bit 2: Botón Terciario
- Byte 1: Variación del eje X. Valor entre -127 y +127. Positivo hacia la derecha.
- Byte 2: Variación del eje Y. Valor entre -127 y +127. Positivo hacia abajo.

Tanto la lectura de un teclado como la de un mouse óptico fueron exitosas. Para probar el programa de detección del estado y velocidad de conexión se utilizó un mouse, un teclado, un HID de Microsoft llamado “SideWinder Strategic Commander”, un Hub y 2 modelos de pendrives. Todos estos dispositivos fueron detectados correctamente.

A la fecha, el egresado Alex Volantines se encuentra terminando su tesis de pregrado, la cual consiste en crear un driver para el MSP430 que utilizando este módulo uUSB-HS permite operaciones de escritura y lectura con un pendrive.

6 Módulo Decodificador MP3: MP3DEC

6.1 Introducción

El más popular método de compresión de música utilizado hoy en día es el *MPEG 2 Layer 3*, más conocido como MP3. Sus tasas de transferencia van desde los 24[kbps] hasta los 320[kbps], mientras que la frecuencia de muestreo puede ir desde los 8 hasta los 48 [kHz]. Para obtener calidad de radio FM se utilizan 64[kbps], mientras que a 128[kbps] ya es considerado calidad de CD.

6.2 Diseño Esquemático

Gracias a la alta tasa de compresión que se logra, sería ideal poder incluir archivos MP3 en un microcontrolador para reproducir mensajes cortos, pero el procesamiento necesario para decodificar un *stream* MP3 impide su implementación en un microcontrolador. Afortunadamente han aparecido en el mercado circuitos integrados cuya única función es decodificar un stream MP3. El más popular es el *STA013*, del fabricante *SGS-Thompson* también conocido como *ST*. Sus principales características son:

- Operación a 3.3 Volt.
- Decodificación de MP3 hasta los 320 [kbps].
- Frecuencias de muestreo de 8 a 48 [kHz]
- Autodetección de la tasa de datos.
- Decodificación de tasa fija (CBR) ó variable (VBR).
- Entrada serial sincrónica de datos MP3 a velocidad de hasta 20Mbps.
- Posee un buffer interno de datos MP3 de aproximadamente 220 bytes.
- Salida serial sincrónica de datos decodificados a velocidad real.
- Bus I2C para comandos.
- Salida de reloj para sobremuestreo del DAC.
- Extracción automática de *metadatos* dentro del stream MP3.
- Control digital de volumen.
- Control digital de tonos de bajos y altos.
- Puede operar con una alta variedad de cristales para generar su señal de reloj.
- Requiere algunos componentes externos.
- 3 tipos de empaque, todos de montaje superficial: PLCC-28, TQFP-44, LFBGA-64.

Una de las desventajas de este chip es que no está disponible en empaque DIP, siendo necesaria la construcción de un circuito impreso para poder conectarlo a un protoboard. La otra dificultad es que su salida de datos es digital y serial, haciendo necesaria la conexión de un chip conversor digital análogo (*DAC*) de 2 canales. Normalmente, los DACs tienen entrada paralela y son monocanal, requiriéndose 2 DACs y 2 conversores serial-paralelo. Afortunadamente existe un DAC serial que puede ser conectado directamente al STA013. Se trata del CS4334, del fabricante *Cirrus Logic* y cuya estructura interna se encuentra en la Figura 32.Las características principales de este DAC son:

- Operación a 5V.
- Una entrada digital serial sincrónica para ambos canales.
- 2 DACs de 24 bits.
- Entrada de reloj de sobremuestreo.

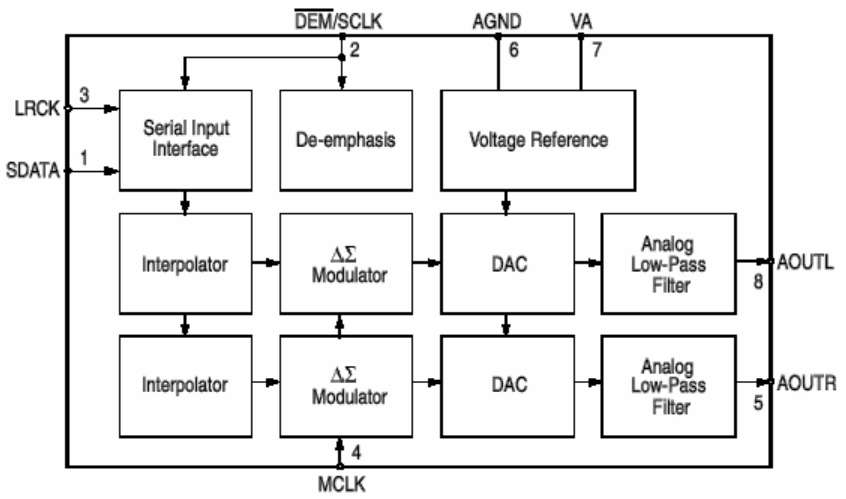


Figura 32. Diagrama interno CS4334.

Si bien este DAC opera a 5V, no existe incompatibilidad de voltajes con el STA013, ya que todos los pines de conexión entre ambos chips son salidas del STA013 y entradas para el CS4334. Las salidas de 3.3V del STA013 alcanzan a ser leídas como un 1 lógico por el CS4334.

Sería entonces útil disponer de un módulo que permitiese conectar el STA013 junto con el CS4334 a un protoboard, para ser utilizado por un microcontrolador como salida de audio MP3.

En base a los datasheets y a varios circuitos encontrados en internet que utilizan ambos chips, se diseñó el circuito esquemático de la Figura 33. En la mitad superior se puede identificar el conexionado básico del chip decodificador de audio STA013, mientras que en la mitad inferior se muestra la conexión básica del DAC CS4334. La alimentación del DAC está conectada a través de una inductancia y posee 2 condensadores para reducir al mínimo el ruido que pueda provocar la fuente de alimentación en la salida de audio. La Tabla 8 contiene la lista de componentes.

Si bien la conexión entre ambos chips es directa y no requiere de intervención alguna, se decidió conectarlos por medio de jumpers, y dejar disponibles los pines comunes en la regleta de conexión al protoboard de forma separada. Esto principalmente por 3 razones:

1. El DAC puede utilizarse para aplicaciones que requieran una conversión digital análoga de alta resolución, pero cuya fuente no es un stream MP3. Es decir, así el CS4334 queda disponible para aplicaciones que no requieran del STA013.
2. El STA013 puede utilizarse como un *coprocesador*, cuya única función es recibir un stream MP3 y entregar los datos decodificados a la salida. Así, el microcontrolador o DSP podría hacer una ecualización del audio antes de entregárselo al DAC, ó utilizar un DAC interno.
3. Existen chips que poseen las mismas entradas del CS4334, pero que tienen salidas ópticas y/o SPDIF. De esta manera se deja abierta la posibilidad de agregar salidas digitales para el audio MP3.

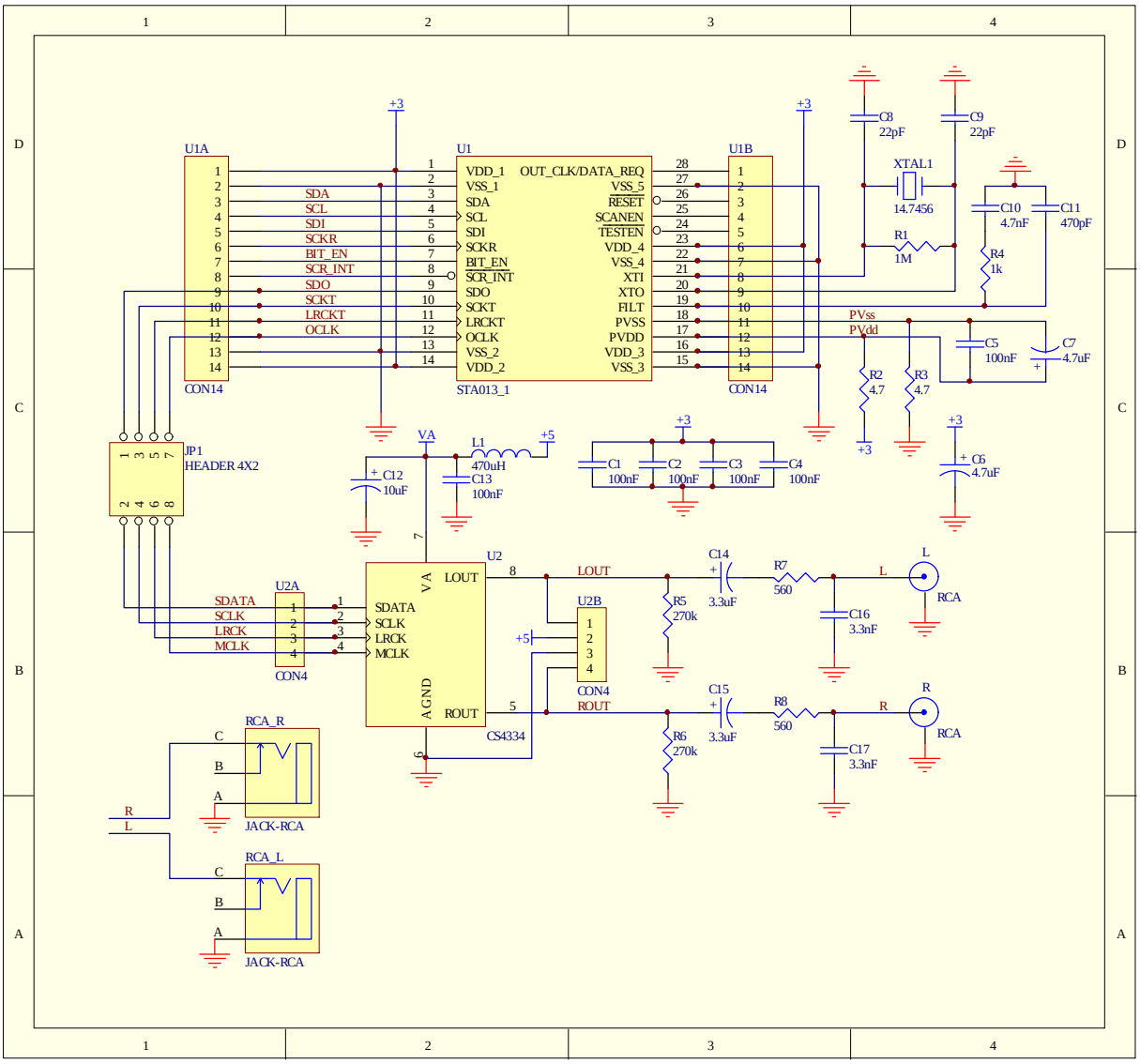


Figura 33. Circuito esquemático módulo MP3DEC.

El DAC provee una salida de audio de 1.75[V] pico a pico, sobre una continua 2.2[V] (Valores típicos). Esta salida se encuentra disponible en la regleta de conexión en dos formatos: directamente y a través de una red RC para filtrar el voltaje continuo. La señal filtrada también está disponible mediante conectores RCA, para ser utilizada como señal de línea en un equipo de audio. La resistencia mínima de carga son 3[kΩ], mientras que la capacitancia máxima de carga son 100[pF]. Eso implica que se requiere de un amplificador de audio para conectar parlantes o audífonos. La elección de un amplificador de audio depende de la aplicación específica en que desea utilizarse el módulo y puede no ser necesario, por lo cual no se ha incluido.

Tabla 8. Lista de componentes módulo MP3DEC.

Cant.	Valor	Designador	Forma	Descripción
1	1M	R1	1206	Resistencia
1	1k	R4	1206	Resistencia
2	3.3nF	C16 C17	1206	Cond. Cerámico
2	3.3uF	C14 C15	RB-.1/.2	Cond. Electrolítico
2	4.7	R2 R3	1206	Resistencia
1	4.7nF	C10	1206	Cond. Cerámico
2	4.7uF	C6 C7	RB-.1/.2	Cond. Electrolítico
1	10uF	C12	RB-.1/.2	Cond. Electrolítico
1	14.7456MHz	XTAL1	XTAL-0.2	Cristal
2	22pF	C8 C9	1206	Cond. Cerámico
6	100nF	C1 C2 C3 C4 C5 C13	1206	Cond. Cerámico
2	270k	R5 R6	1206	Resistencia
1	470pF	C11	1206	Cond. Cerámico
1	470uH	L1	1812	Bobina
2	560R	R7 R8	1206	Resistencia
2	CON4	U2A U2B	SIP-4	Connector
2	CON14	U1A U1B	SIP-14	Connector
1	CS4334	U2	SO-8	DAC serial 2 canales
1	HEADER 4X2	JP1	HDR2X4	pinheader macho 2x4 no polarizado
2	JACK-RCA	RCA_L RCA_R	JACK-RCA	Jack RCA para PCB
2	RCA	L R	SIP-2	BNC Connector
1	STA013_1	U1	SOL-28	Decodificador MP3
4				Jumper

Para el diseño del circuito impreso se tuvieron en consideración los siguientes aspectos:

- Ubicar condensadores de filtro lo más cerca posible de los pines de alimentación del chip.
- Utilizar un plano de tierra.
- Ruteo de la pista de +3.3V de manera de minimizar el ruido.
- No cruzar las pistas de datos seriales con las pistas de audio análogo, para evitar la inducción de ruido en las pistas de audio.
- Hacer el circuito lo más angosto posible.
- Dejar todos los pines de ambos chips disponibles en la regleta de conexión. De esta manera, pueden soldarse sólo los dos chips y utilizarse componentes thru-hole conectadas a través de un protoboard.
- Hacer el módulo lo más angosto posible, de manera de poder colocarlo en un protoboard.
- Utilizar en lo posible componentes de montaje superficial.
- En lo posible, no poner componentes por la cara inferior.

El circuito impreso resultante es el de la Figura 34.

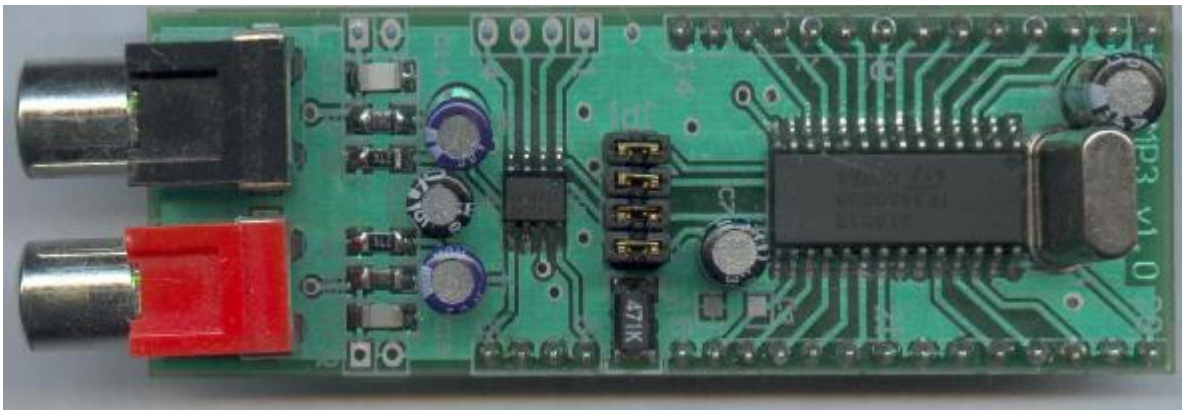


Figura 34. Módulo MP3DEC montado.

6.3 Driver para el STA013

Se creó un driver en C y dos archivos de audio de ejemplo para demostrar el funcionamiento del chip. Estos archivos tienen un tamaño aproximado de 32[kB], suficientemente pequeños para ser incluidos en la misma memoria flash de varios modelos de las MSP430.

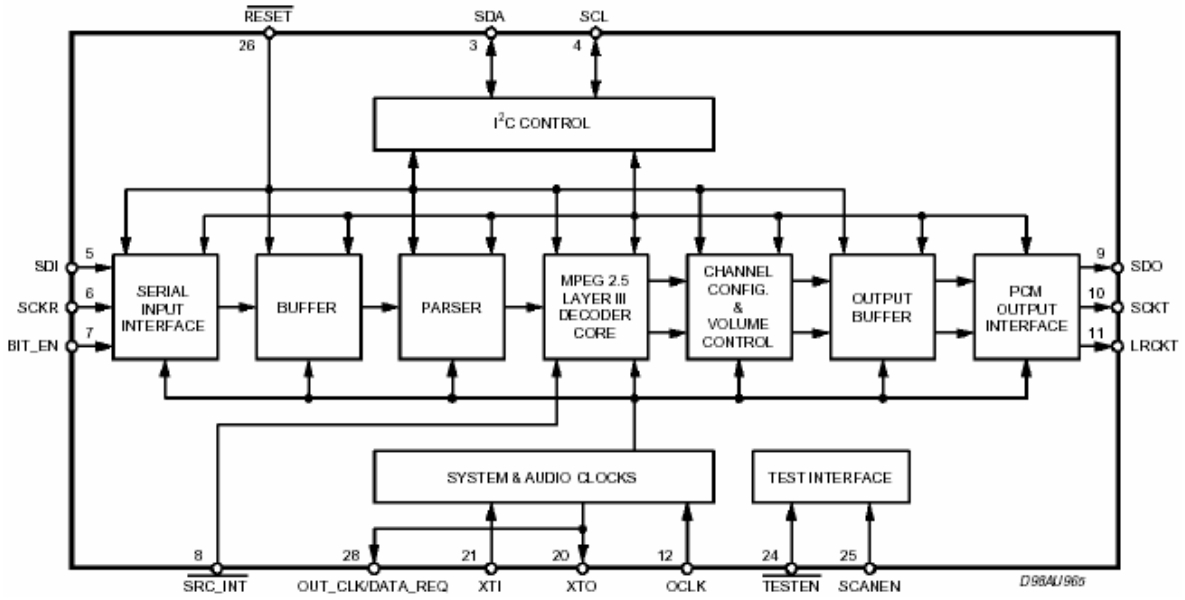


Figura 35. Diagrama interno STA013.

En primer lugar debe diseñarse la conexión entre el STA013 y el microcontrolador, en este caso una MSP430F1612. El diagrama interno del STA013 es el de la figura 35. Los microcontroladores de la familia MSP430F16xx poseen 2 puertos seriales, pero sólo el puerto 0 soporta la interfaz I2C por hardware. Por esta razón el puerto serial 0 será utilizado para la interfaz de comandos I2C, mientras que el puerto serial 1 será utilizado para la interfaz de datos, en modo sincrónico o SPI. El pin BIT_EN habilita la entrada de datos por la interfaz serial. Es equivalente a un *chip select* del puerto serial y suele ser utilizado para bytes, debido a que la interfaz SPI sólo posee sincronismo de bit y no de byte.

El pin RESET se conectó a la MSP en vez de conectarlo a una red RC, ya que al utilizar la MSP con el software para emulación, se puede resetear la MSP, pero no el STA013. El pin DATA_REQ se pone en 1 cuando el buffer interno de MP3 se está agotando, por lo que debe conectarse a una entrada de interrupción de la MSP, para avisarle que deben enviarse más datos al buffer. Los pines SRC_INT, TESTEN y SCANEN no se utilizarán. Todo esto se resume en la Figura 36, en la cuál aparece la conexión entre una MSP, el módulo MP3DEC y un display LCD.

La documentación que provee el fabricante de los MSP es muy deficiente y no pudo configurarse satisfactoriamente la USART0 en modo I2C, por lo que se optó por buscar en internet una aplicación para manejar memorias EEPROM I2C, y ese código fue adaptado para el STA013. La transferencia de datos I2C se hace mediante interrupciones.

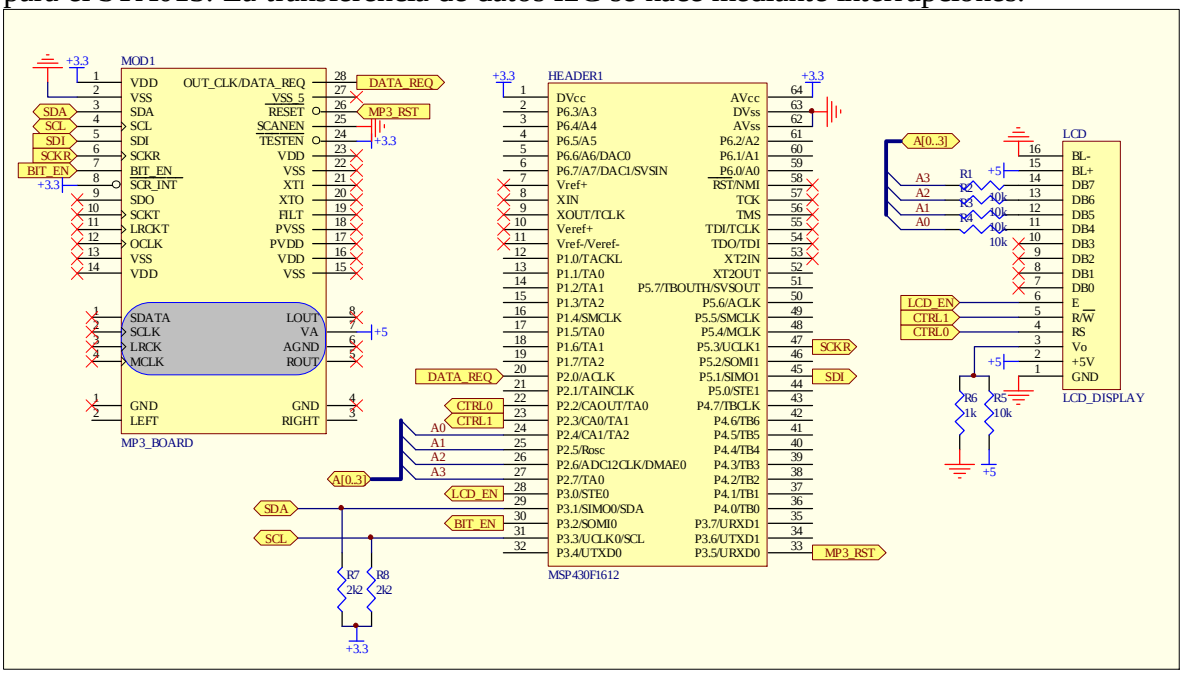


Figura 36. Conexión MSP, STA013 y LCD.

6.4 El protocolo I2C.

Este es un protocolo de transferencia serial que utiliza 2 señales (en base a una tierra en común) para una comunicación sincrónica *half-duplex*. La comunicación en un bus I2C se hace entre un maestro y un esclavo. Sin embargo, pueden existir múltiples maestros y múltiples esclavos compartiendo un único bus I2C. Las dos señales son llamadas SDA para la señal de datos y SCL para la señal de reloj. Todos los dispositivos del bus I2C escriben estas señales mediante un transistor a tierra. En estado inactivo, las señales SDA y SCL permanecen en alto por medio de dos resistencias de *pull-up* cuyo valor suele ser de 2[kΩ]. Estas resistencias no están presentes en ninguno de los chips conectados al bus, sino que se colocan externamente. De esta manera, cuando un dispositivo escribe un 1 lógico en una de las señales, lo que realmente hace es dejar el pin en 3er estado o estado de alta impedancia, mientras que cuando escribe un 0 lógico, conecta la respectiva señal a tierra mediante un transistor interno, es decir, son salidas de colector abierto. De esta manera, si 2 dispositivos

están escribiendo valores lógicos distintos, no existe un corto circuito, y el valor lógico resultante es un 0. Esta consideración es necesaria por características funcionales del protocolo que se indican a continuación.

Al transferirse datos por el bus I2C, la señal SDA sólo puede cambiar de estado cuando la señal de reloj SCL se encuentra en estado bajo. Un cambio en SDA mientras SCL se encuentra en estado alto, indica una condición de start o stop. La señal de reloj SCL es siempre controlada por el dispositivo Master y es éste quién inicia todas las transferencias.

Antes de enviar un comando, el Master crea una condición de *START*. Ésta es generada mediante un canto de bajada de la señal SDA mientras la señal SCL permanece establemente en un estado alto.

Para indicar el fin de una transmisión, el Master crea una condición de *STOP*. Ésta es generada mediante un canto de subida de la señal SDA mientras la señal SCL permanece establemente en un estado alto.

La transferencia de datos y comandos se hace mediante paquetes de 1 byte. Luego de cada byte, el transmisor libera la señal SDA para que el receptor (Master o Slave) responda con un ACK (*acknowledge*). El receptor genera el ACK tirando la señal SDA a tierra.

6.5 Rutinas I2C para la MSP430.

La inicialización del hardware I2C es la misma para el STA013 que para memorias EEPROM. Los pines P3.1 y P3.3 deben ser configurados como periféricos y entradas, ya que corresponden a las señales bidireccionales SDA y SCL del bus I2C, respectivamente. Luego se configura la USART0 en modo I2C. El bit SWRST protege la configuración I2C y debe setearse en 1 para poder hacer modificaciones. La dirección del dispositivo esclavo es parte de la configuración protegida (registro `I2CSA`) y es establecida durante la inicialización como 0x43 que corresponde al STA013, ya que éste es el único dispositivo esclavo conectado al bus I2C. Si desea conectarse una memoria EEPROM y el STA013, deberá actualizarse el registro `I2CSA` con SWRST=1 cada vez que se cambie el esclavo con el cuál desea realizarse la comunicación.

```
void InitI2C(void)
{
  P3SEL = 0x0A;   // select module function for the used I2C pins
  P3DIR &= ~0x0A;
  U0CTL |= I2C+SYNC;     // (1) Select I2C mode with SWRST=1
  U0CTL &= ~I2CEN;       // (2) disable the I2C module
                         // (3) Configure the I2C module with I2CEN=0 :
                         //     U0CTL default settings:
                         //     7-bit addressing, no DMA, no feedback
  I2CTCTL = I2CTRX+I2CSSEL_2; // Byte mode, repeat mode, SMCLK,
                              // transmit mode
  I2CSA = SlaveAddress; // Define Slave Address
                         // In this case the Slave Address defines the
                         // control byte that is sent to the EEPROM.
  I2COA = 0x01A5;        // own address.
```

```
  I2CPSC = 0x00;         // I2C clock = clock source/1
  I2CSCLH = 0x03;        // SCL high period = 5*I2C clock
  I2CSCLL = 0x03;        // SCL low period  = 5*I2C clock
  U0CTL |= I2CEN;        // (4) set I2CEN via software
}
```

Luego deben establecerse las funciones de escritura y lectura para el bus I2C. El primer byte que se envía por el bus I2C es un byte de control, que contiene la dirección de 7 bits del esclavo y un bit para el tipo de operación escritura o lectura.

6.5.1 Escritura aleatoria

Para escritura, luego del byte de comando debe enviarse la dirección interna del registro que desea escribirse, la cuál es almacenada en un puntero interno. Ya que el STA013 posee sólo 128 registros, se requiere un único byte para el direccionamiento interno. Memorias EEPROM de más de 128 bytes requieren 2 bytes de direccionamiento interno. Luego del byte de dirección interna se envían los datos a escribir. Cada byte de datos que se envía incrementa automáticamente un puntero interno lo que posibilita la escritura de varios bytes seguidos en una sola operación. Esto es muy útil para memorias EEPROM, ya que la escritura en memoria EEPROM o FLASH es muy lenta y puede llegar a los 10[ms]. Pero estas memorias están organizadas en páginas que pueden ser escritas al mismo tiempo gracias a que poseen un buffer interno cuyo tamaño varía entre una EEPROM y otra. Escribir una página completa demora lo mismo que escribir un solo byte. Durante una operación de escritura en EEPROM, los bytes se van almacenando en el buffer interno que suele ser circular por lo que debe tenerse cuidado. El proceso de grabado en la memoria se realiza luego de que llega una condición de STOP del bus I2C, indicando que se ha transferido el último byte. El diagrama temporal de las operaciones de escritura de byte y multibyte se presenta en la Figura 37.

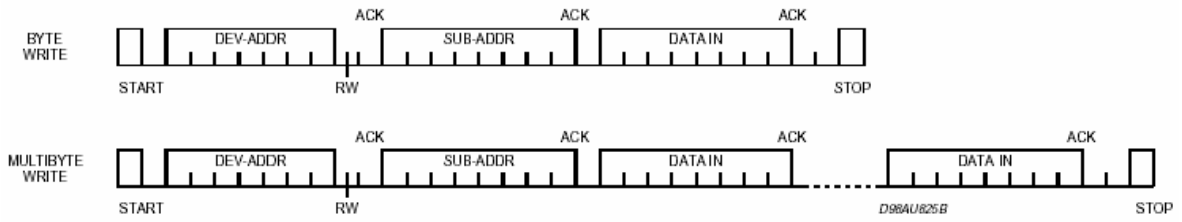


Figura 37. Diagrama temporal de operaciones de escritura.

El STA013 no posee memoria de estado sólido, sino que registros en RAM cuyo tiempo de escritura es ínfimo. Por esta razón no requiere de un buffer, lo que implica que no hay un límite de bytes que puedan ser escritos durante una sola operación. Debido a que en general se deseará accesar registros en un orden aleatorio, se utilizará una función para escribir un único byte.

A continuación se muestra la función de escritura de un solo byte. Primero que todo debe esperarse a que el bus I2C esté libre. Luego se guardan en un buffer la dirección interna y el dato a escribir. La variable `PtrTransmit` apunta a la posición del primer byte (dirección

interna) a enviar, mientras que el registro `I2CNDAT` contiene el número de bytes a enviar luego del byte de comando.

```
void STA013_ByteWrite(unsigned char Address, unsigned char Data)
  {
      while (I2CDCTL&I2CBUSY)
            ;     // wait until I2C module has finished all operations

      I2CBuffer[1] = Address;
      I2CBuffer[0] = Data;
      PtrTransmit = 1;        // set I2CBuffer Pointer

      I2CWriteInit();
      I2CNDAT = 2;      // 1 control byte + 2 bytes should be transmitted
      I2CTCTL |= I2CSTT+I2CSTP;     // start and stop condition generation
                                    // => I2C communication is started
  }
```

La función `I2CWriteInit()` prepara el I2C para el comando de escritura y se detalla a continuación.

```
void I2CWriteInit(void)
  {
      U0CTL |= MST;             // define Master Mode
      I2CTCTL |= I2CTRX;        // I2CTRX=1 => Transmit Mode (R/W bit = 0)
      I2CIFG &= ~TXRDYIFG;
      I2CIE = TXRDYIE;          // enable Transmit ready interrupt
  }
```

La rutina de interrupción que maneja el envío de los bytes desde el buffer es la siguiente:

```
#pragma vector=USART0TX_VECTOR
__interrupt void ISR_I2C(void)
  {
      switch (I2CIV)
        {
            case I2CIV_TXRDY:       // Transmit ready (TXRDYIFG)
            I2CDRB = I2CBuffer[PtrTransmit];
            PtrTransmit = PtrTransmit-1;
            if (PtrTransmit<0)
                  I2CIE &= ~TXRDYIE;      // disable interrupts
            break;
        }
  }
```

6.5.2 Lectura Aleatoria

La lectura en un bus I2C se hace sobre la dirección actual del puntero. Esto significa que para leer una dirección de memoria específica, primero debe actualizarse el puntero. Esto se hace mediante una operación de escritura sin datos y sin *STOP*, seguida de una operación de lectura. Los distintos tipos de lectura se muestran en la Figura 38.

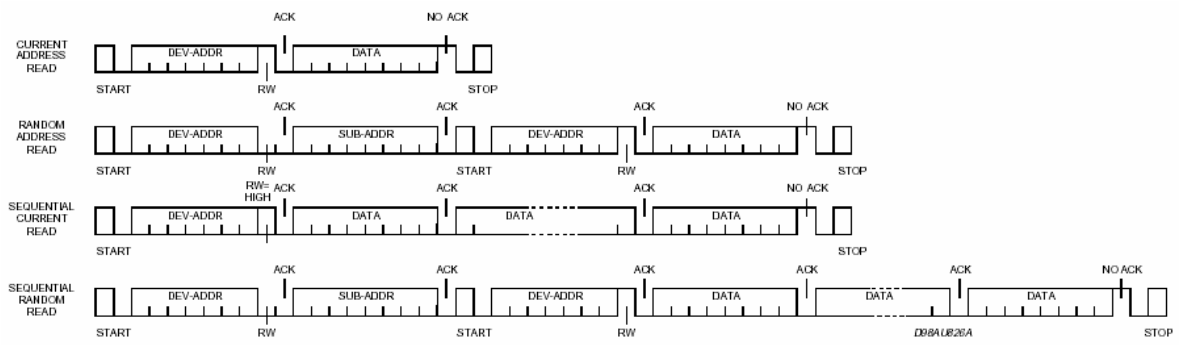


Figura 38. Diagrama temporal de operaciones de lectura.

Debido a que lo que se desea leer son registros aleatorios, se implementará el 2º tipo de lectura que aparece en la Figura 38, llamada `STA013_RandomRead()`. En ésta, se utiliza la función `I2CWriteInit()` anteriormente descrita, y su análoga `I2CReadInit()` que prepara el I2C para lectura.

```
unsigned char STA013_RandomRead(unsigned char Address)
{
      unsigned int k=0;
      while (I2CDCTL&I2CBUSY); // wait until I2C module has finished
      I2CBuffer[0] = Address; // in the I2CBuffer.
      PtrTransmit = 0;        // set I2CBuffer Pointer

      I2CWriteInit();
      I2CNDAT = 1;      // 1 control + 1 data byte should be transmitted
      I2CIFG &= ~ARDYIFG;     // clear Access ready interrupt flag
      I2CTCTL |= I2CSTT;      // start condition generation
                              // => I2C communication is started
      while ((~I2CIFG)&ARDYIFG)  // wait untill transmission is finished
            if( (I2CIFG&NACKIFG)!=0)
              {
                  SEND_CMD(LINE2);
                  printf("STA013 I2C ERROR");
                  while(1)
                        ;
              }
      I2CReadInit();
      I2CNDAT = 1;      // 1 byte should be received

      I2CIFG &= ~ARDYIFG;     // clear Access ready interrupt flag
      I2CTCTL |= I2CSTT+I2CSTP; // start receiving and finally generate
                               // re-start and stop condition
      while ((~I2CIFG)&ARDYIFG);   // wait untill transmission is finished
      return I2CBuffer[0];
}

void I2CReadInit(void)
  {
      I2CTCTL &= ~I2CTRX;     // I2CTRX=0 => Receive Mode (R/W bit = 1)
      I2CIE = RXRDYIE;        // enable Receive ready interrupt
  }
```

6.6 Inicialización

Después de desactivar la señal de reset del STA013, éste debe ser programado con un firmware que el fabricante entrega en un archivo que contiene pares de bytes que corresponden a dirección y dato a escribir, los cuales deben ser enviados en orden al STA013. El porqué de la necesidad enviar este archivo no está documentado y parece ser un *patch*. Si no se envía ese archivo, el STA013 no funciona. El archivo `sta013_data.c` contenido en el cd tiene un arreglo unidimensional con el archivo a enviar y contiene también un set de comandos al final para configurar apropiadamente la conexión de hardware del módulo MP3DEC. Esto incluye la frecuencia del cristal, la configuración para la conexión del DAC, etc. Las primeras lineas del arreglo son:

```
const unsigned char STA013_PROG[]={      58    ,     1
      ,     42    ,     4
      ,     40    ,     0
      ,     41    ,     0
      ,     32    ,     0
      ,     33    ,     0
...
```

Utilizando las funciones I2C anteriormente descritas, el envío de este archivo se hace de la forma:

```
for(i=0;i<2011;i++)
  {
      address = (unsigned char)STA013_PROG[2*i];
      data = (unsigned char)STA013_PROG[(2*i)+1];
      STA013_ByteWrite(address,data);
  }
```

6.7 Bus de datos

Ahora que ya está configurado el STA013 para operar, debe configurarse la USART1 en modo SPI para alimentar al decodificador con los datos MP3. La comunicación SPI está configurada para máxima frecuencia, que son 4[Mbps]. Una interfaz SPI consta de 3 señales:

- MOSI o SIMO: *Master Out Slave In*. Es la salida de datos desde el Maestro al Esclavo.
- MISO o SOMI: *Master In Slave Out*. Es la entrada de al Maestro desded el Esclavo.
- MCLK: Es la salida de reloj del dipositivo Maestro.

Es necesario configurar correctamente la polaridad de las señales de datos y de reloj tanto en el STA013 como en la MSP. Se decidió utilizar la polaridad por defecto del STA013 con SCLK_POL=0. Ésta puede verse en la Figura 39, en la cuál también puede apreciarse la función del pin BIT_EN. Para configurar correctamente el bus SPI debe compararse el diagrama temporal del STA013 de la Figura 39 con el diagrama de la Figura 40, correspondiente a la MSP.

A diferencia de un bus I2C, el bus SPI no posee un protocolo de selección de chip, debiendo seleccionarse mediante una señal externa de *Chip Enable*. Esto porque una interfaz SPI está pensada para máxima velocidad de transferencia y no se incluye ningún tipo de sincronización de byte ni destinatario para no agregar overhead. Así, la señal BIT_EN sirve además para coordinar la transmisión, ya que al ser desactivada, resetea la cuenta de bits.

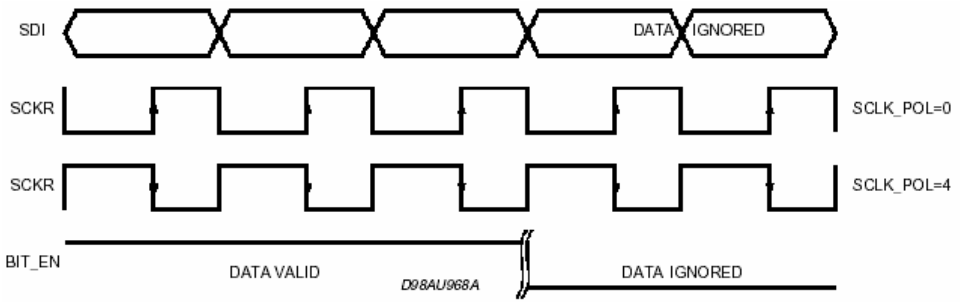


Figura 39. Diagrama temporal SPI STA013.

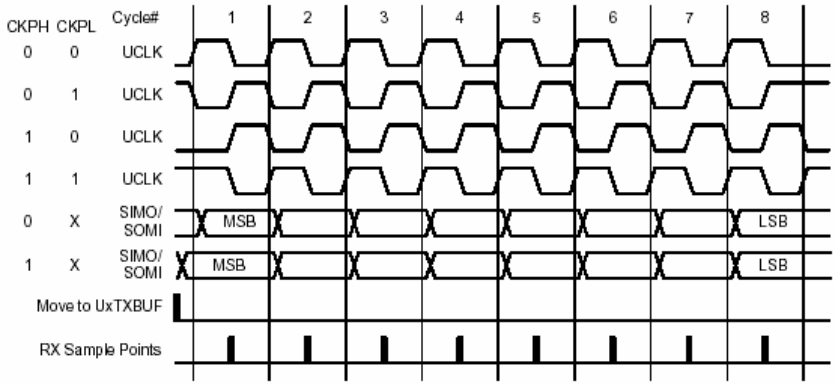


Figura 40. Diagrama temporal SPI MSP430 .

Debido a que el STA013 sólo recibe datos, el pin MISO de la MSP no es necesario y queda libre para ser usado como un pin de entrada/salida digital. La rutina de inicialización está listada a continuación.

```
void Init_SPI(void)
  {
      P5SEL |= (BIT1|BIT3);   // Bits 3 & 1 are peripherals

      ME2 |= USPIE1;          // Module Enable - SPI

      U1CTL &= ~SWRST;        // Make sure the RESET bit is off
      U1CTL |= CHAR + SYNC + MM; // USART1 module operation
                              // CHAR = 1 => 8-bit data
                              // SYNC = 1 => SPI selected
                              // MM = 1 => master mode,
                              //     MSP is the master

      U1TCTL |= SSEL0 + SSEL1 | STC; // USART0 Tranmit control register
```

```
                                // SSEL0 = 1 & SSEL1 = 1
                                // => SMCLK is used for baud-rate generation
                                // STC = 1 => 3-pin SPI mode selected
      U1BR0  = 0x02;            // Divide SMCLK by 2 => transfer clock
      U1BR1  = 0x00;            //
      U1MCTL = 0x00;            // Modulation control - not used. Ensure
                                // all these bits are reset
  }
```

6.8 Reproducción

Finalmente el módulo MP3DEC se encuentra configurado y listo para empezar a reproducir un *stream* MP3. Para ello se le envían 2 comandos al STA013 que le indican que comience a reproducir. Cuando el STA013 necesite más datos, pondrá su pin DATA_REQ en 1. En ese momento se le enviarán datos vía la interfaz SPI hasta que el pin DATA_REQ vuelva a 0, indicando que el buffer se encuentra lleno.

```
STA013_ByteWrite(114,1);      //Start Running;
STA013_ByteWrite(19,1);       //Start Playing
while(1)
  {
      while(DATA_REQ != 0)
        {
            while( (IFG2&UTXIFG1) == 0)
                  ;
            U1TXBUF=mp3_file[i];
            i++;
            if (i==mp3_file_size)   //vuelta a 0 del puntero del buffer
                  i=0;              //circular
        }
  }
```

Para generar un archivo de ejemplo se creó una función en matlab la cual crea un arreglo en C a partir de un archivo binario cualquiera. El programa, llamado mp32c (MP3 to C) se lista a continuación:

```
function mp32c(file);
%file
infile=sprintf('%s',file)
FID=fopen(infile);
if (FID>2)
    fprintf('Archivo abierto, ID = %d\n',FID)
else
    fprintf('Error abriendo archivo\n')
    exit
end
DATA=fread(FID);
fprintf('Tamaño de archivo es %d bytes\n\n',max(size(DATA)))
%outfilename=sprintf('%s.c',filename)
outfile=infile;
i=max(size(infile));
outfile(i-2)='c';
outfile(i-1)='';
outfile(i-1)='';
fout=fopen(outfile,'w');
fwrite(fout,sprintf('const unsigned char mp3_file[%d]={%3d',max(size( DATA)),DATA(1)));
i=2;
k=1+16;
per=0;
max=max(size(DATA));
```

```
%fprintf('Ultimos 3 datos: %3d,%3d,%3d\n',DATA(max-2),DATA(max-1),DATA(max) )
while i<max
    while ( (i<k) & (i<=max) )
        fwrite(  fout, sprintf(',%3d',DATA(i))  );
        i=i+1;
        if 100*i/
        printf('/b/b/b%3d',(100*i)/max)
    end
    fwrite(  fout, sprintf('\n')  );
    k=i+16;
end
fwrite( fout, sprintf('};\n') );
fclose(fout);
```

Las primeras líneas del archivo generado se muestran a continuación.

```
unsigned int mp3_file_size=32735;
const unsigned char mp3_file[32735]={
255,251, 48,192,  0,  0,  4,228,189,102, 24,114,128,  1, 33,139
,238,107,158, 48,  1,240, 15, 20,120,144, 79,207, 63,219,252, 14
,255,236,199,255,224,116, 33,  9,255,228, 62,127,255,236,  7,120
...
```

El tamaño máximo del archivo está dado por la memoria flash libre que quede en el microcontrolador, ó puede grabarse en una FLASH ó EEPROM externa. En el CD adjunto se encuentra un ejemplo que reproduce continuamente un archivo MP3 de pocos segundos. Al momento de compilar puede elegirse uno de los 2 archivos de ejemplo.

Existen memorias FLASH seriales SPI de hasta 8[Mbit]. Eventualmente pueden reproducirse archivos desde un pendrive utilizando el módulo USB descrito en otro capítulo de este trabajo. También podría bajarse uno o más archivos de hasta 512[kB] desde un servidor web o ftp utilizando el módulo de red uLAN y guardarlos temporalmente en una memoria RAM externa. Tanto el módulo de red uLAN como el módulo de RAM externa de 256k x 16bit se encuentran descritos en otros capítulos de este trabajo.

La hoja de datos del STA013 especifica todos los registros de uso público. Sus funciones son variadas, desde los valores del formato del *stream* MP3 (*bitrate*, frecuencia, etc), hasta los valores de atenuación de tonos altos, bajos y general para la ecualización de la salida de audio.

7 Hardware Relacionado

7.1 Memoria RAM 256k x 16bit.

El último módulo desarrollado consistió en un Header para una memoria RAM de 512kB. Debido a que las MSP430 trabajan con un voltaje máximo de 3.6[V] fue necesario buscar memorias RAM que trabajasen a este bajo voltaje. No fue posible encontrar en ningún distribuidor chileno o estadounidense una memoria RAM de 3[V] en empaque DIP. Las únicas memorias RAM de bajo voltaje que se encontraron en stock fueron la M68AW128 y la M68AW256 de la SGS-Thompson. Éstas son memorias con un bus de datos de 16 bits y de 2 y 4Mb de capacidad respectivamente. Ambos modelos utilizan el mismo empaque TSOP de 44 pines. Estos chips son memorias SRAM y no requieren circuitería adicional alguna. El Header RAM es sólo para poder conectar estos chips a protoboard.

Como puede apreciarse en el circuito esquemático de la Figura 41, los pines de estas memorias están desordenados. Aprovechando la necesidad de tener que construir el circuito impreso, en vez de rutear los pines en el mismo orden en que se encuentran en el chip, se decidió redistribuir los pines para que queden ordenados y de esta manera poder utilizar los buses de datos descritos en un comienzo. Además se aprovechó de agregar condensadores de filtro a la alimentación.

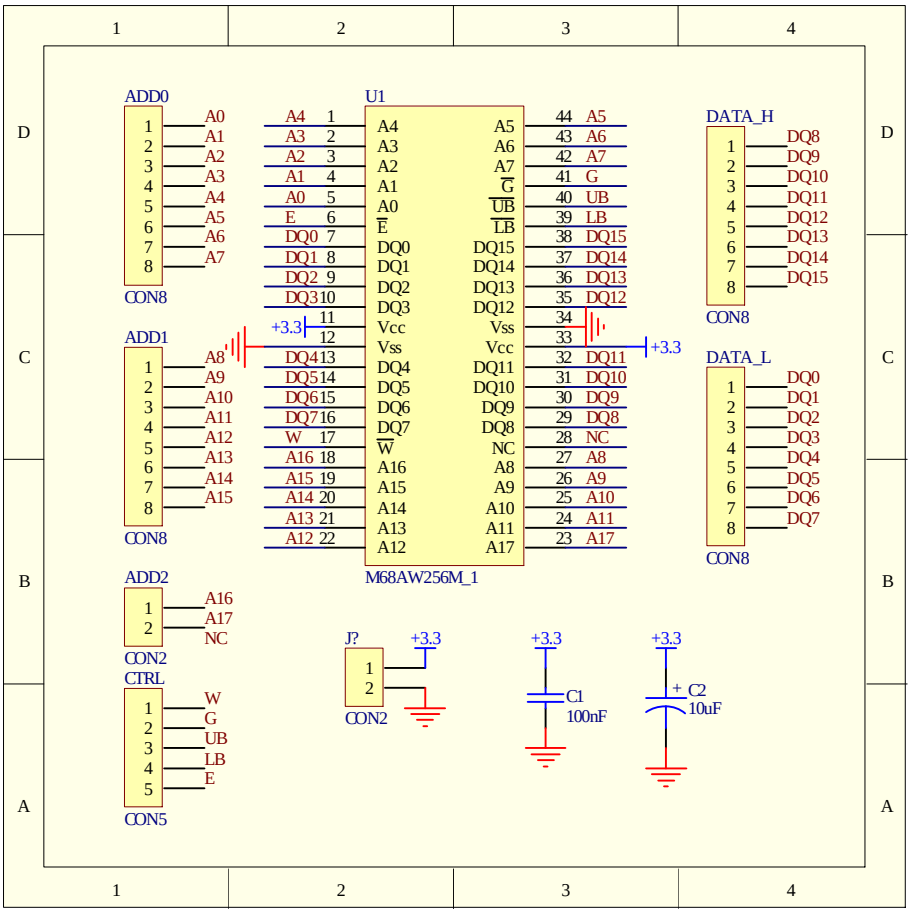


Figura 41. Circuito esquemático módulo RAM.

Como resultado se obtuvo el módulo RAM de la Figura 42. En éste no están soldados los condensadores, puesto que son opcionales. Puede apreciarse que cada pin está nombrado en la serigrafía, facilitando el conexionado.

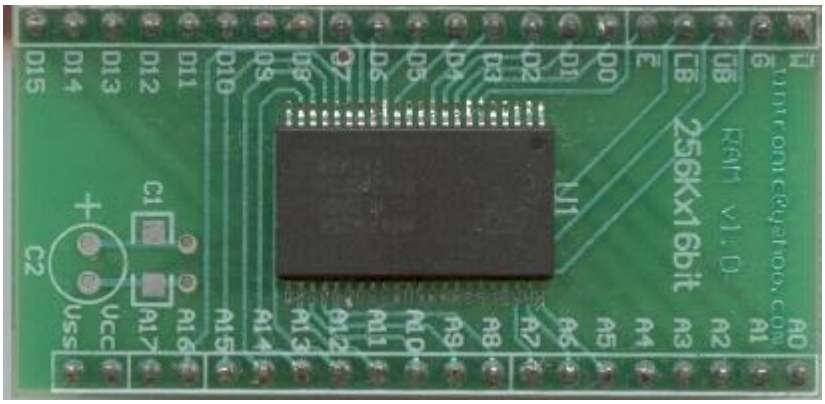


Figura 42. Módulo RAM.

7.2 Hardware adicional

7.2.1 Memorias no volátiles

En cuanto a memoria de datos no volátil para almacenar variables que no deben borrarse incluso cuando el circuito está sin poder, existen un sinnúmero de fabricantes de memorias seriales EEPROM en formato DIP. Las memorias seriales con interfaz I2C y de 3V de alimentación pueden encontrarse bajo los códigos 24LCxx y están disponibles en distribuidores locales. Estas mismas memorias, pero con interfaz serial SPI, tienen código 93LCxx. Los tamaños máximos para memorias EEPROM seriales suelen ser 256kb, es decir 64kB. Si se necesita una mayor capacidad, el distribuidor estadounidense mouser vende memorias FLASH seriales SPI con capacidad de hasta 8Mb. El código de esta memoria es SST25LF080A.

7.2.2 Displays LCD alfanuméricos.

Existe una gran cantidad de fabricantes de Displays LCD alfanuméricos. Todos utilizan el mismo formato. Además de tener una gran relación costo/beneficio en cuanto a la cantidad de información que es posible mostrar con un único display, éstos requieren un mínimo número de pines de comunicación. Sólo 6 pines bastan para mostrar cualquier información en estos displays. Esto ya es menor que el mínimo de 7 pines para un diplay de LEDs de 7 segmentos que sólo muestra un dígito. El más común de estoy Displays es el de 2x16, es decir, 2 líneas por 16 caracteres alfanuméricos. Éstos pueden encontrarse en distribuidores locales como globalelectronica.cl o victronics.cl y no es más barato importarlos.

En la Figura 12 aparece cómo conectar una MSP430 y un display LCD en modo 4 bits y sólo escritura. Ésta es la conexión típica y sólo requiere un total de 6 señales para utilizar el LCD: 4 bits de datos y 2 bits de control. En la Figuras 31 y 36 aparece cómo conectar un display LCD en modo bidireccional. Para ello es necesario agregar resistencias en serie en el puerto de datos, ya que el LCD retorna datos en 5V y la MSP430 sólo soporta hasta Vcc

+ 0.3V, es decir máximo 3.6V al ser alimentada con 3.3V. Gracias a unos diodos de protección que posee la MSP430 en todas sus entradas y que están conectados en antiparalelo GND y Vcc, basta con agregar resistencias en serie de unos 10[kΩ] para absorber la diferencia de voltaje.

En el CD adjunto se encuentra un driver para LCD en modo 4 bits y bidireccional. También hay un archivo de ejemplo tipo que demuestra varias características de estos displays.

7.2.3 Teclados Matriciales.

Teclados Matriciales son una útil herramienta para que un usuario pueda introducir datos a un microcontrolador, ya que con sólo 8 líneas de datos pueden manejarse 16 botones. Éstos también pueden encontrarse en distribuidores locales, y existen en una gran variedad de formas y tamaños. La Figura 43 muestra una forma de conectar un teclado matricial a una MSP430. En el CD adjunto se encuentra un driver para teclados matriciales. La función `Tec_Matrix()` que efectúa el barrido del teclado debe ejecutarse cada 5 a 20 [ms] y retorna un valor entre 0x00 y 0xF, indicando la posición de la tecla presionada.

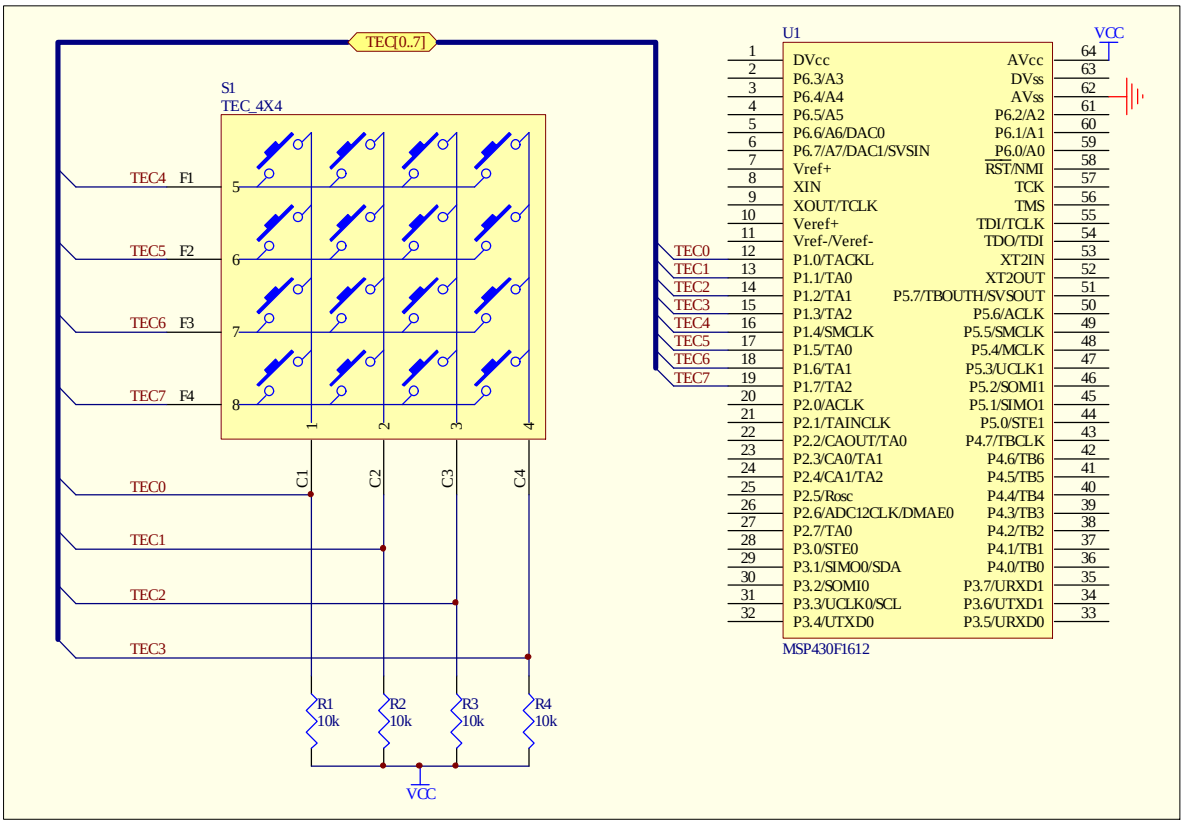


Figura 43. Conexión teclado matricial.

7.2.4 Interfaz 3V-5V.

Si es necesario conectar un dispositivo de 5V en modo bidireccional a una MSP y utilizar resistencias en serie no es una opción, existe un chip que permite hacer esta conversión. Se trata del 16211 y puede ser encontrado en mouser.com. Este chip posee 2 puertos bidireccionales de 12 bits cada uno. La particularidad de este chip radica en que sus puertos son switch bidireccionales. Es decir que no es necesario ir cambiando la dirección del puerto como sí ocurre con otros chips como la familia 74LS24x. Este chip es utilizado principalmente en tarjetas PCI para computadores, para hacer compatibles tarjetas con chips de 3V en sistemas PCI de 5V. En el CD adjunto se encuentran los datasheet del 16211 de 2 fabricantes distintos.

La Figura 44 muestra un módulo diseñado para conectar un chip 16211 a un protoboard. Este módulo sólo deja disponibles 8 bits de un puerto y la señal de Enable o activación del switch entre los pines A y B. Este módulo no fue construido, pero se deja su diseño a disposición. Por ejemplo puede servir para conectar el bus de datos de la tarjeta uLAN o el módulo RAM a un microcontrolador de 5V.

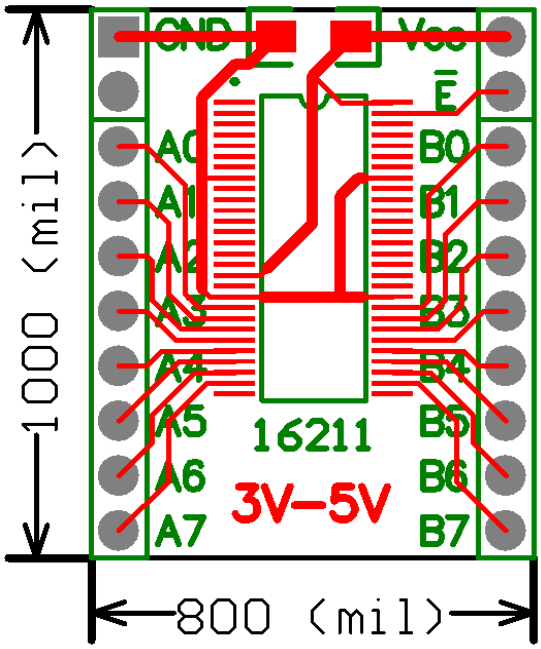


Figura 44. Módulo interfaz 3V-5V.

7.2.5 Interfaces de potencia

Finalmente, muchas veces se necesita controlar cargas de alto voltaje o potencia. Existe una gran variedad de circuitos que pueden armarse en protoboard utilizando distintos tipos de optoacopladores, según sea el caso. Por esta razón no se creó ningún módulo de este tipo.

Comentarios

Para el 2º semestre de 2005 se construyeron 35 kits, uno para cada grupo del laboratorio de estructura de computadores. La idea tras esto fue que cada grupo pudiese llevar consigo la plataforma para desarrollar las experiencias en casa, y no estar limitados a trabajar sólo durante la sesión de laboratorio, teniendo que armar y desarmar los circuitos cada semana. Cada kit constó de los siguientes módulos:

- Protoboard cuádruple.
- Header MSP430 más programador JTAG.
- Display LCD de 2x16.
- Teclado Matricial de 4x4.
- Módulo RS-232 (Este último no alcanzó a estar listo a tiempo por problemas con el distribuidor).

Además se construyeron 35 uLAN, 15 uUSB-HS, 15 MP3DEC y 15 RAM. Estos estuvieron a disposición de los alumnos para que desarrollaran su proyecto final.

En cuanto a la experiencia con la adquisición de componentes, esta fue frustrante. Los 2 distribuidores chilenos de componentes electrónicas no respetaron los plazos, tanto para componentes de lista como para importaciones especiales.

El importador internacional de componentes RS sí cumplió sus plazos, pero debido a los precios que maneja, sólo fueron unas pocas componentes las que se adquirieron por esta vía.

Finalmente, algunas componentes se adquirieron en EEUU vía casilla en Miami. Los tiempos de importación fueron parecidos a los de RS, pero los precios mucho más bajos en la mayoría de las componentes. Sumando fletes e impuestos, el costo de importación se encuentra alrededor del 50% (incluido IVA) del valor de las componentes en EEUU. Incluso así los precios de la mercadería puesta en chile puede llegar a ser muy parecida o incluso menor al precio de las otras alternativas. La experiencia con mouser.com en específico ha resultado ser tremendamente satisfactoria:

- Las componentes son enviadas al mismo día o al día siguiente.
- Buenos descuentos por cantidades, sobre todo en resistencias.
- Buen soporte por mail.
- Cualquier problema con el envío o la tarjeta de crédito es informada en menos de 24 horas.
- Los envíos los hacen en cajas lo más chicas posibles, lo que disminuye los costos de importación.
- Posee hojas de dato en línea de casi todas sus componentes.

Al hacer importaciones debe tenerse cuidado en no sobrepasar el límite de 500 dólares, incluido el flete dentro de EEUU. Esto debido a que desde 500 dólares, debe contratarse un agente de aduanas, lo que eleva el costo y el tiempo que demora en obtenerse la mercadería.

Conclusiones

El desarrollo de esta plataforma demostró que es absolutamente factible que la universidad construya sus propios Kits de desarrollo en vez de tener que importarlos. La empresa chilena CIGA, que se dedica a la fabricación de circuitos impresos, posee la tecnología necesaria para fabricar las PCBs con el detalle necesario para soldar componentes con pines de tan sólo 10 milésimas de pulgada de ancho y de separación entre pines. El taller de electrónica posee las herramientas y experiencia necesarias para el soldado de todos los módulos desarrollados durante esta memoria. Si bien la capacidad de montaje del taller no es de los cientos por mes, es satisfactorio para las necesidades institucionales.

La adquisición de componentes para los circuitos diseñados fue una tarea que demandó mucha investigación y tiempo. Definitivamente no puede confiarse en los plazos de los distribuidores chilenos de venta directa. Distribuidores como RS son una alternativa efectiva pero cara, aunque el ahorro en horas hombre lo puede hacer comercialmente viable en una empresa, mas no para el estudiante o memorista. La mejor opción sin embargo son distribuidores en EEUU como Newark y en especial Mouser. La compra en estos distribuidores puede hacerse por internet, pagando con tarjeta de crédito. El tiempo normal entre la orden de compra y el arribo de las componentes a una casilla en Miami es de 3 a 5 días. Luego la importación hacia Chile demora de 3 a 5 días más, incluyendo los trámites de aduana, los cuales son realizados por la misma agencia importadora como eShopEx. En cuanto al costo, basta con multiplicar el precio de las componentes por 1.5 para obtener un valor bastante aproximado de su costo puesto en Chile.

Si la lista de componentes incluye distribuidores chilenos y extranjeros, es recomendable comprar las componentes en Chile antes de hacer el pedido al extranjero. Una vez que se tienen en la mano las componentes compradas en distribuidores chilenos, recién entonces debe hacerse el pedido al extranjero. Con más de una componente sucedió que estaba en stock al momento de hacer la cotización o el depósito bancario, pero que al momento de tener que ser enviadas ya no estaban en stock, y sin que el distribuidor nos avisara de la situación. Simplemente las componentes pagadas no llegaron. En este punto en específico se quiere mencionar la experiencia con Victronics, quién no envió una gran cantidad de componentes cuyo stock se había consultado pocos días antes y cuya falta de preocupación al respecto fue sorprendente. No solo al no informar de la falta de stock y hacerse los desentendidos. Como si esto fuera poco, entre las componentes que luego repusieron, enviaron algunas que ellos consideraron como compatibles, sin dar aviso alguno, pero que no servían para el voltaje con que se estaba trabajando. Específicamente enviaron algunos 74LS244 entre los 74HC244 que se habían encargado. Este problema sólo fue descubierto luego de armar los JTAG y ver que una gran cantidad de ellos no funcionaba, sin tener explicación aparente.

La experiencia en el desarrollo de estros módulos demostró que el primer paso para la selección de qué componentes utilizar no pasa por buscar en las páginas de los fabricantes. Primero debe buscarse en las páginas de los distribuidores conocidos, cuáles son los chips disponibles y con stock y luego deben compararse los datasheets. Es muy importante buscar documentos de errata en la página de los fabricantes antes de decidirse por una determinada componente, para no encontrarse con sorpresas, como sucedió con el módulo uUSB-HS.

Cuando se trabaja con componentes que tienen un empaque que no se ha utilizado antes, es recomendable hacer una impresión láser del circuito impreso y hacer un chequeo con quién la va a soldar (en general el personal de taller). Muchas veces los footprints de los programas de diseño de PCB son demasiado pequeños para soldado a mano, o tienen pequeñas discrepancias.

La utilización de estos módulos en docencia con más de 90 alumnos fue comprobada en el Laboratorio de Estructura de Computadores, cuyas anteriores tarjetas de desarrollo, compradas en el extranjero, fueron reemplazadas por esta plataforma para el segundo semestre de 2005. La versatilidad de la plataforma permitió generar experiencias de acuerdo a los objetivos específicos del ramo y su estructura modular permitió la integración de otros módulos que habían sido adquiridos anteriormente. El abanico de proyectos desarrollados por los distintos grupos del laboratorio demostró que esta plataforma tiene una aplicación no sólo didáctica, sino que va más allá, permitiendo incorporar inteligencia a existentes y nuevos productos chilenos, haciendo un aporte a la innovación en Chile.

Por otro lado, la experiencia en este laboratorio demuestra que pueden desarrollarse en la Universidad otro tipo de tarjetas con DSP o FPGA, ya que en Chile existen fabricantes de circuitos impresos con el nivel tecnológico, el Taller ELO posee las herramientas y existen los canales de adquisición de componentes desde el extranjero.

Bibliografía

MSP430:

- *"MSP430x1xx Family User's Guide (Rev. E)"*, Texas Instruments, http://focus.ti.com/general/docs/lit/getliterature.tsp?literatureNumber=slau049e&fileType=pdf
- *"MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller"*, Texas Instruments, Oct/02, http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=msp430f1611

CS8900A:

- *"CS8900A Product Data Sheet"*, Cirrus Logic, Jan/2004 http://www.cirrus.com/en/pubs/proDatasheet/CS8900A_F3.pdf
- *"AN181: Using the Crystal 8900A in 8-Bit Mode"*, Cirrus Logic, 1/2000 http://www.cirrus.com/en/pubs/appNote/an181.pdf
- *"Crystal LAN CS8900A Ethernet Controller Technical Reference Manual"*, Cirrus Logic http://www.cirrus.com/en/pubs/appNote/An83-3.pdf
- *"CS8900A Frequently asked questions"*, Cirrus Logic, Aug/2002 http://crystal.com/en/pubs/appNote/an205-2.pdf

Conectores RJ45 con transformador de pulsos:

- *"Single Port FastJacks™ With Integrated Filters"*, Halo Electronics Inc., Rev. 7/03 http://www.haloelectronics.com/pdf/fastjack-10base-t.pdf
- *"STARJACKTM 1X1 Tab-DOWN RJ45"*, Pulse, J414.A (10/03) http://www.pulseeng.com/pdf/J414.pdf

RS-232:

- *"MAX3232 Datasheet"*, Rev06, 10/03, http://pdfserv.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf

SL811HS:

- *"SL811HS Embedded USB Host/Slave Controller"*, Cypress, Mar/02 http://www.cypress.com/portal/server.pt/gateway/PTARGS_0_2_1524_209_259_43/http%3B/sjapp20%3B7001/publishedcontent/publish/design_resources/datasheets/contents/sl811hs_5.pdf
- *"Basic Embedded Host Using the SL811HS"*, Cypress, Dec/02 http://www.cypress.com/portal/server.pt/gateway/PTARGS_0_2_1289_209_0_43/http;/sjapp20;7001/publishedcontent/publish/design_resources/application_notes/contents/basic_embedded_host_using_the_sl811hs_9.pdf
- *"Errata for SL811HS Embedded USB Host/Slave Controller"*, Cypress, Dec/04, http://www.cypress.com/portal/server.pt/gateway/PTARGS_0_2_1289_209_0_43/http;/sjapp20;7001/publishedcontent/publish/design_resources/errata_update/contents/errata_for_sl811hs_embedded_usb_host_slave_controller_9.pdf
- *"Interfacing an External Processor to the SL811HS/S"*, Cypress, Dec/02, http://www.cypress.com/portal/server.pt/gateway/PTARGS_0_2_1289_209_0_43/http;/sjapp20;7001/publishedcontent/publish/design_resources/application_notes/contents/interfacing_an_external_processor_to_the_sl811hs_s_9.pdf

Estándar USB:

- *"Universal Serial Bus Specification"*, USB.org, Revision 1.1, Sep/98, http://www.usb.org/developers/docs/usbspec.zip
- *"Device Class Definition for Human Interface Devices (HID)"*, USB.org, Revision 1.11, Sep/98, http://www.usb.org/developers/devclass_docs/HID1_11.pdf

Switch de poder:

- *"MIC2026 DATASHEET"*, MICREL, Rev.11/04, http://www.micrel.com/_PDF/mic2026.pdf

MP3Dec:

- *"ST013 Datasheet"*, SGS-Thompson, Nov/99, http://www.st.com/stonline/products/literature/ds/6399.pdf
- *"CS4334/5/8/9 Product Data Sheet"*, Cirrus, Jul/04, http://www.cirrus.com/en/pubs/proDatasheet/CS4334-35-38-39_F3.pdf

RAM:

- *"M68AW256M Datasheet"*, SGS-Thompson, Apr/04, http://www.st.com/stonline/products/literature/ds/7996.pdf

Interfaz 3V-5V:

- *"FSTD16211 Datasheet"*, Fairchild Semiconductor, Nov/02, http://www.fairchildsemi.com/ds/FS%2FFSTD16211.pdf
- *"SN74CBTD16211 Datasheet"*, Texas Instruments, Jul/02, http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=sn74cbtd16211

Anexos

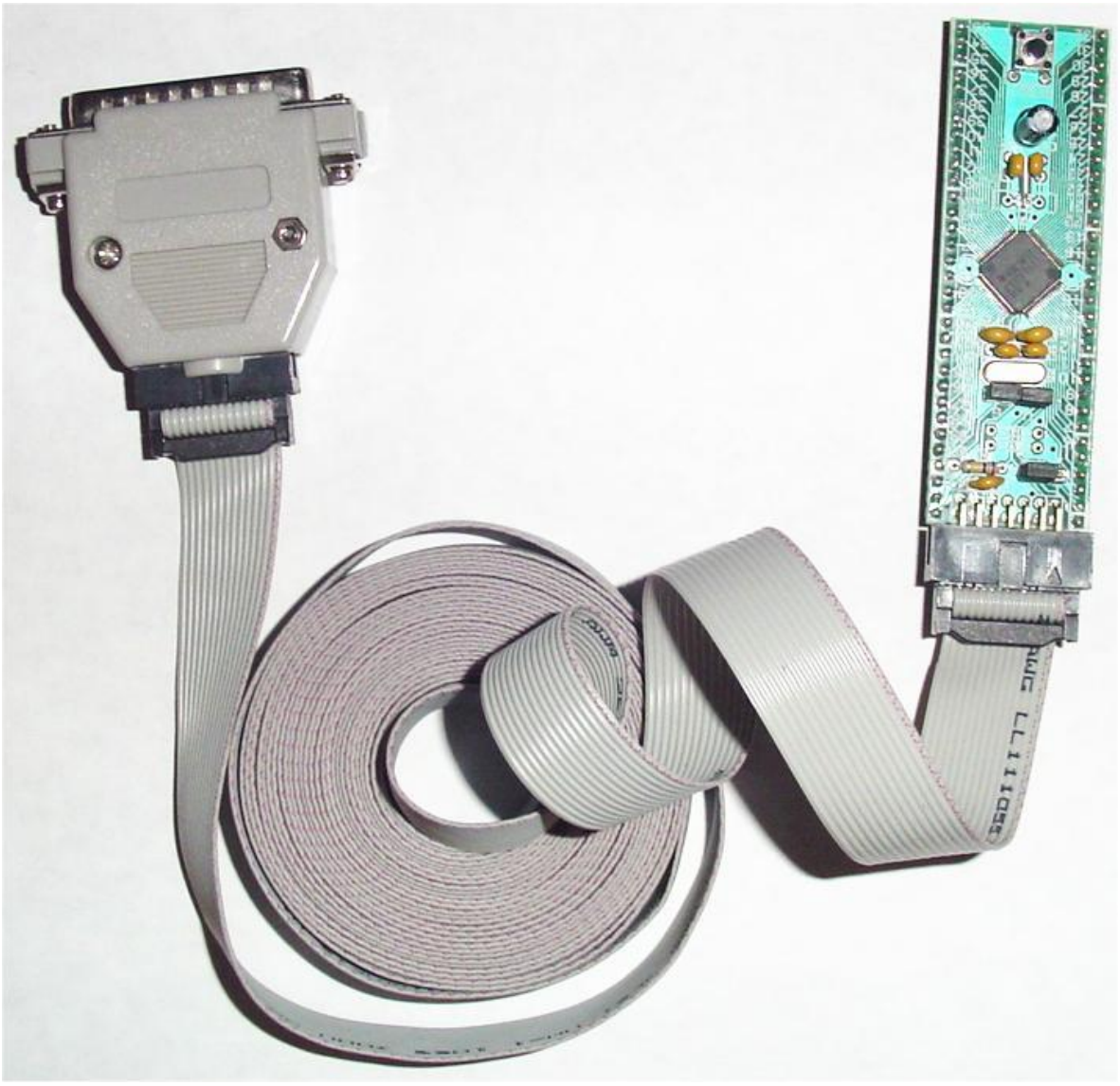
MSP430 Quickstart

Para Kits del Taller
ELO de la UTFSM

Por Michael Kusch
tintronic@yahoo.com
Versión 0.3.
Diciembre de 2005

Paso 1: Conexión

Conectar el programador JTAG al puerto paralelo del PC y al microcontrolador.



Paso2: Jumpers

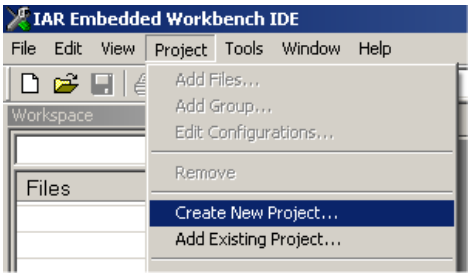
Poner el jumper de selector de alimentación JVE (JTAG – Vcc – Externo) en J-V para alimentar el microcontrolador (uC) desde el JTAG. De esta manera no es necesario conectar una fuente de poder para alimentar el uC. El JTAG prevee 3.0V, pero una corriente muy baja, ya que es obtenida de los buses de datos del puerto paralelo.

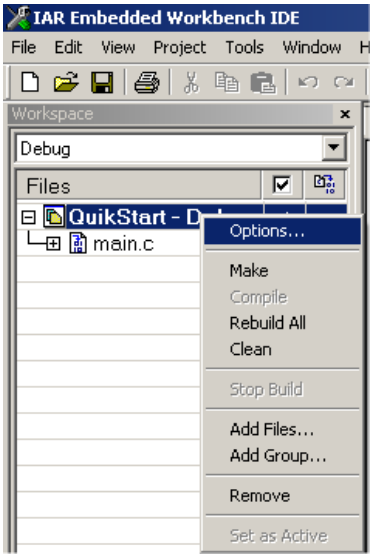
Dejar puestos los jumpers V (puentea DVcc y AVcc) y G (puentea DVss y AVss), ya que éstos interconectan el voltaje análogo y digital. Sólo deben sacarse si se utiliza una fuente externa ***y se desea una alimentación especial para la circuitería análoga*** interna del uC que minimice el ruido en el conversor análogo digital.

El botón de Reset es sólo para ser utilizado cuando el uC NO está conectado al PC. Si se presiona durante la emulación (cuando el programa corre en el PC y en el micro a la vez), se perderá la comunicación entre el PC y el uC y habrá que reinicializar la comunicación.

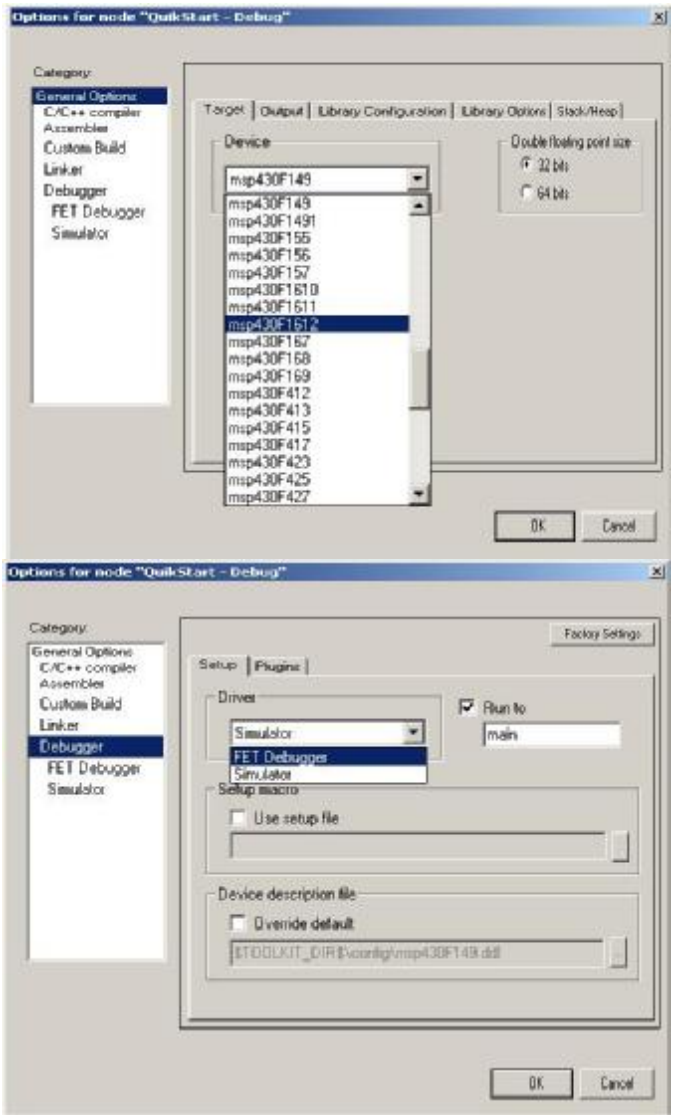
Paso 3: Crear y configurar proyecto.

Bajar el demo del software IAR desde http://www-s.ti.com/sc/techzip/slac050.zip.
Abrir el programa IAR Embedded Workbench y crear un nuevo proyecto tipo C -> main.





Clic derecho sobre el proyecto recientemente creado y seleccionar “Options”.
En opciones generales elegir el micro que se desea programar y se tiene conectado.



Luego en Debugger seleccionar “FET Debugger”. Aquí estamos seleccionando conexión al micro a través del JTAG. El puerto paralelo puede seleccionarse en la columna del lado izquierdo, bajo el menú FET Debugger. Presionar OK.

Paso 4: Programación básica.

Éste es un programa que contiene lo básico: el archivo de definiciones `msp430x16x.h` y la inicialización de los relojes.

```
#include <msp430x16x.h> //Definiciones básicas
void Init_Osc_X2(void);

int main( void ) //Programa principal
  {
      Init_Osc_X2();    //configuración básica del micro

      //Inicializaciones de puertos y recuros. Ejemplo:
      P1DIR = 0x00;     //Puerto 1 como entrada
      P2DIR = 0xFF;     //Puerto 2 como salida

      while(1)                //Loop Infinito
        {                     //Aquí se programa el "Super-Loop"
            P2OUT = P1IN;     //La salida del Puerto 2 es igual
                              // a la entrada al Puerto 1
        }
  }

void Init_Osc_X2(void)
  {
//    Al enceder la MSP, ésta queda operando a aprox 800kHz para MCLK y
SMCLK, y a XT1 en modo LF para ACLK

      BCSCTL1 &= ~XT2OFF;     // XT2 encendido (8MHz)
      do                      // wait in loop until crystal is stable
            IFG1 &= ~OFIFG;
      while (IFG1 & OFIFG);
      IE1 &= ~WDTIE;          // disable WDT int.
      IFG1 &= ~WDTIFG;        // clear WDT int. flag

//Cambiar el siguiente comando dependiendo de la configuración de reloj
desead. No cambiar nada más.
      BCSCTL2 = SELM_2 | DIVM_0 | SELS | DIVS_3 ;
// MCLK <- XT2 | MCLK = 1:1 | SMCLK <- XT2 | SMCLK = 1:8 ;

      WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDTIS1 ;
// use WDT as timer, flag each 512 pulses from SMCLK
      while (!(IFG1 & WDTIFG));// count 1024 pulses from XT2(until XT2's
                               // amplitude is OK)
      IFG1 &= ~OFIFG;          // clear osc. fault int. flag
  }
```

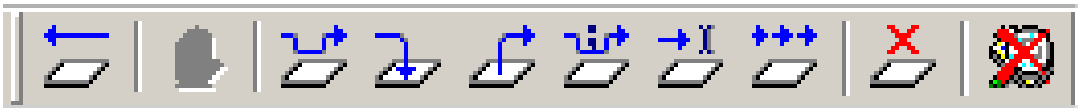
Ahora tenemos 3 señales de reloj: ACLK a 32.768Hz, MCLK (reloj de instrucciones) a 8MHz y SMCLK a 1MHz.

Paso 5: Emulación.

Para compilar el programa presionar el icono:

Para compilar, grabar y comenzar la emulación, presionar el icono:

Ahora ha aparecido una nueva barra de herramientas:



1 2 3 4 5 6 7 8 9 10

De izquierda a derecha los íconos son:

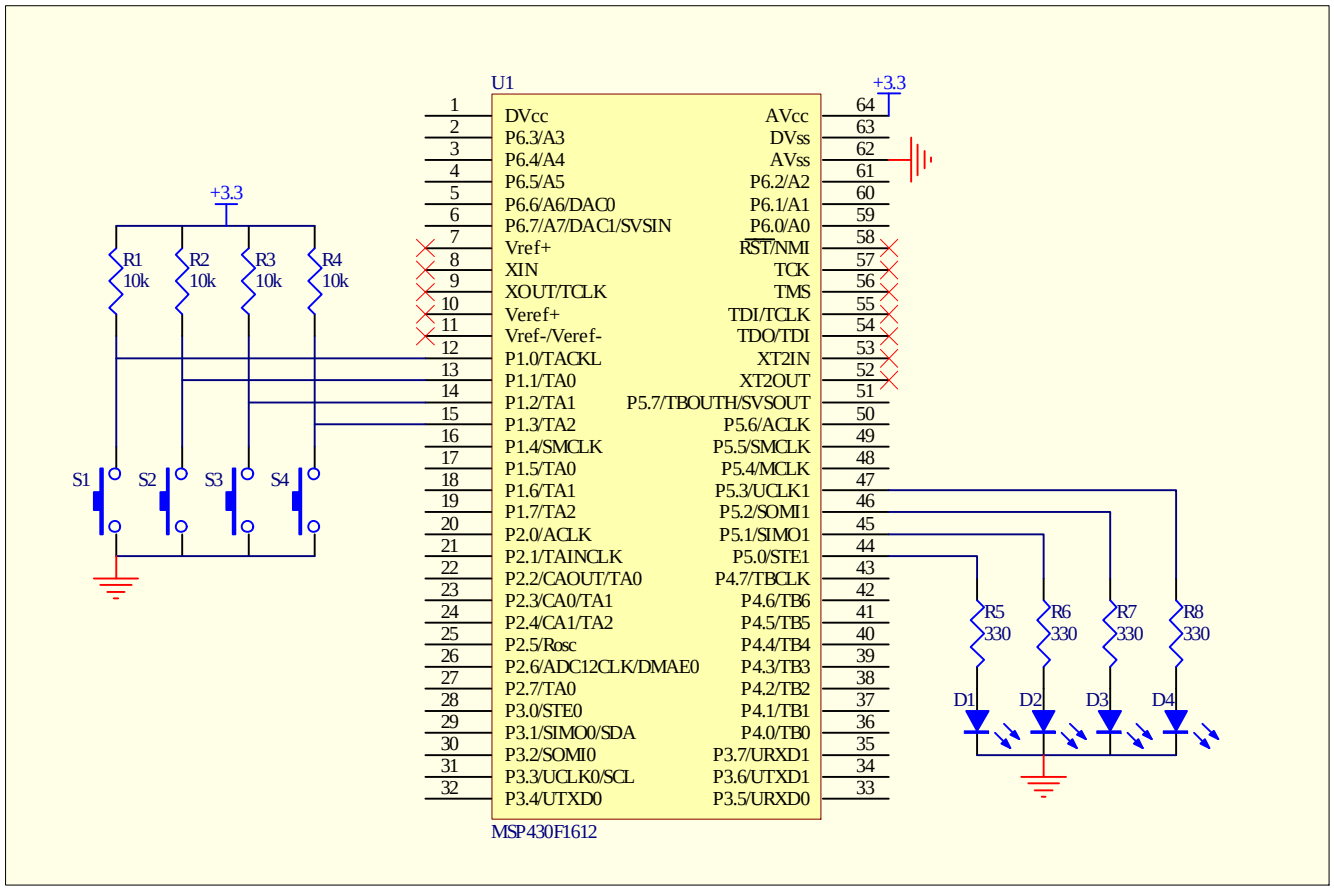
1. Reset. Utilizar éste botón en vez del que se encuentra en la placa.
2. Stop, detener ejecución del programa.
3. Ejecutar instrucción o subrutina
4. Entrar a subrutina
5. Salir de subrutina
6. (moya)
7. Ejecutar programa hasta posición del cursor
8. Ejecutar programa (libremente)
9. Activar/Desactivar Breakpoint
10. Salir del modo Emulación.

Para ver el estado de cualquier registro del micro, debe detenerse la ejecución del programa con el botón Stop.
Luego en el menú View se selecciona Registers.
Ahora puede verse el estado de cualquier registro.

Para ver el estado de variables de programa, utilizar View -> Watch ó hacer doble clic sobre la palabra en cualquier parte del programa y luego clic derecho y seleccionar “Add to Watch”.

Paso 6: Entradas y Salidas básicas: Botones y Leds.

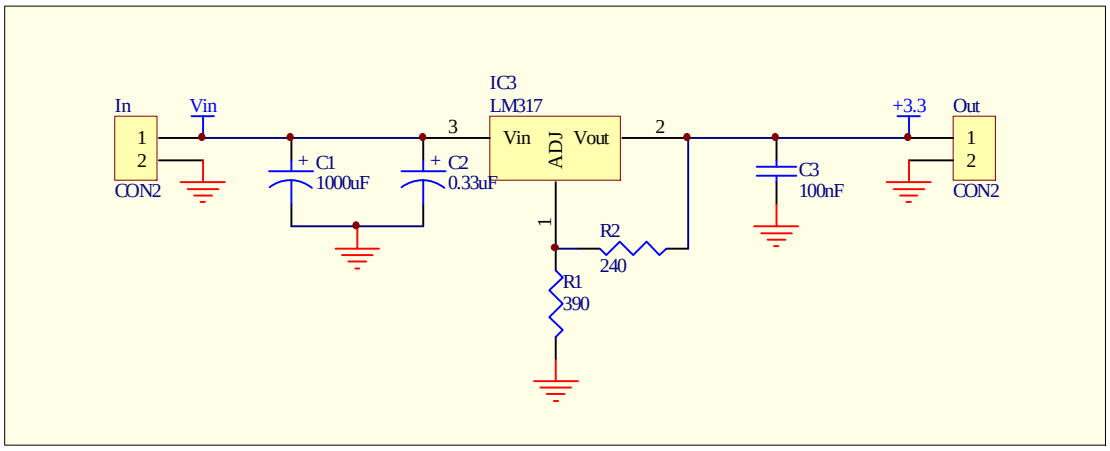
Ya puedes comenzar a programar tu propio proyecto!!!!
Para ver el estado de los pines puedes conectar un LED a través de una resistencia de 220 a 560 ohms. Mientras más alto el valor, menor consumo de corriente y menor brillo del LED. Recuerda que el JTAG sólo puede suministrar unos pocos miliampere.
Para conectar interruptores, conéctalos entre el pin y tierra. Luego conecta una resistencia de valor alto (típicamente 10k) a Vcc.



Paso 7: Utilización de fuente externa.

Si se requiere manejar más LEDs o chips que la corriente que puede suministrar el JTAG, entonces se debe conectar una fuente externa. Para ello se debe cambiar el jumper JVE a la posición V-E. Luego conectar una fuente de voltaje de 2.7 a 3.6V (típicamente de 3.3V) a los pines de alimentación: El positivo al pin 1 ó 64 y la tierra al pin 62 ó 63 (sólo es necesario conectar uno de los Vcc y un Vss, ya que los otros 2 están puenteados a través del jumper V y del G).

El siguiente es el esquemático de un regulador de voltaje de salida 3.3V/1A y entrada 5.5 a 30V (a mayor voltaje y mayor consumo, mayor calentamiento y más grande debe ser el disipador) utilizando el popular LM317.



Consultas, Comentarios y Aportes son Bienvenidos
Enviarlos a tintronic@yahoo.com.

Ejemplos: Seminario de Computadores, Semestre 1 de 2004:
http://www.elo.utfsm.cl/~lsb/elo325/elo325.html

Datasheet de la Familia MSP430:
http://www.elo.utfsm.cl/~lsb/elo325/datos/msp430x1xx.pdf

Datasheet de la MSP430F1612:
http://www-s.ti.com/sc/ds/msp430f1612.pdf

Esquemático del Circuito Impreso del Kit MSP430