UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ORIHUELA

INGENIERÍA INDUSTRIAL

"DISEÑO DE HARDWARE Y SOFTWARE DE UN CONTROLADOR DE ENTRADAS Y SALIDAS DIGITALES PROGRAMABLE A TRAVÉS DE USB"

PROYECTO FIN DE CARRERA

SEPTIEMBRE – 2013

AUTOR: Máximo Vicente Amorós

DIRECTORA: María Asunción Vicente Ripoll

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

DISEÑO DE HARDWARE Y SOFTWARE DE UN CONTROLADOR DE ENTRADAS Y SALIDAS DIGITALES PROGRAMABLE A TRAVÉS DE USB

Proyectante: MÁXIMO VICENTE AMORÓS

Director/es: MARÍA ASUNCIÓN VICENTE RIPOLL

VºBº director/es del proyecto:

Lugar y fecha: _____

CALIFICACIÓN NUMÉRICA ☐ MATRÍCULA DE HONOR ☐

Méritos justificativos en el caso de conceder Matrícula de Honor:

_____

_____

_____

_____

Conforme presidente:	Conforme secretario:	Conforme vocal:

Lugar y fecha: _____

ÍNDICE

CAPÍTULO 1 – INTRODUCCIÓN AL PROYECTO IO5-USB

1.1 – INTRODUCCIÓN Y MOTIVACIÓN

El siglo XXI se ha convertido en un reto para la humanidad. Los cambios sociales y tecnológicos han gestado lo que hoy día llamamos globalización. Cada día que pasa crece la rivalidad comercial y se ha creado la sensación de que siempre hay alguien que hace lo mismo que tú pero más barato.

En el ámbito tecnológico se están alcanzando unos niveles de competencia que estresan incluso a la empresa mejor posicionada mercantilmente. Son numerosos los casos de grandes empresas que parecían invencibles en su campo y que con el tiempo han perdido el liderazgo, víctimas de la falta de nuevas ideas y de errores estratégicos como subestimar a la competencia o sobreexplotar una moda.

Country/Region	Expenditures on R&D (billions of US$, PPP)	% of GDP PPP
Israel	9.4	4.2%
South Korea	55.8	3.74%
Japan	160.3	3.67%
Sweden	11.9	3.3%
Finland	6.3	3.1%
United States	405.3	2.7%
Austria	8.3	2.5%
Denmark	5.1	2.4%
Germany	69.5	2.3%
Taiwan	19.0	2.3%
Switzerland	7.5	2.3%
Iceland	0.3	2.3%
Singapore	6.3	2.2%
China	296.8	1.97%
France	42.2	1.9%
Canada	24.3	1.8%
United Kingdom	38.4	1.7%
Australia	15.9	1.7%
Belgium	6.9	1.7%
Luxembourg	0.67	1.62%
Netherlands	10.8	1.6%
Norway	4.2	1.6%
Czech Republic	3.8	1.4%
Ireland	2.6	1.4%
Slovenia	0.8	1.4%
Spain	17.2	1.3%

Figura 1: Lista de inversión en I+D por países en 2011.

Es trivial que España no se caracteriza por su empuje tecnológico (la Figura 1 representa uno de tantos aspectos a mejorar) pero algunos profesionales de la tecnología como un servidor sabemos que eso se puede cambiar a través del esfuerzo y la concienciación. A pesar de que la competencia acecha, no son pocos los ejemplos de empresas de ingeniería que son un tesoro intelectual para la sociedad que las acoge y una fuente de bienestar social. Con esa mentalidad está forjado este proyecto final de carrera, en el que desglosaré de una manera comprensible y sencilla la tarea de crear un producto industrial competitivo encontrando un compromiso entre los medios disponibles y las múltiples limitaciones que se presentan.

1.2 – OBJETIVOS

Puesto que se va a proceder al diseño de un producto comercial el objetivo del equipo como conjunto es disponer de un alto grado de competitividad mercantil y para ello será necesario buscar una relación óptima entre prestaciones y costes. Las prestaciones requeridas y los costes máximos ya definen a grandes rasgos el tipo de tecnologías y métodos que se emplearán en el diseño.

En los dos siguientes sub-apartados se desglosan los objetivos para el hardware y el software, los cuales han sido creados partiendo de las características de productos similares que ya existen en el mercado (bibliografía **[8]**).

1.2.1 – OBJETIVOS EN EL DISEÑO DE HARDWARE

Para el hardware se usará una filosofía bastante común dentro de este campo de la ingeniería: un sistema compacto y altamente integrado. Minimizar la cantidad de componentes discretos en pro de soluciones embebidas en circuitos integrados suele ser la estrategia ganadora en términos de precio, dimensiones y fiabilidad. La Figura 2 muestra un regulador lineal de tensión típico junto con su diseño esquemático y sus dimensiones físicas, ilustrando la ventaja en términos de espacio de decantarse por soluciones altamente integradas.

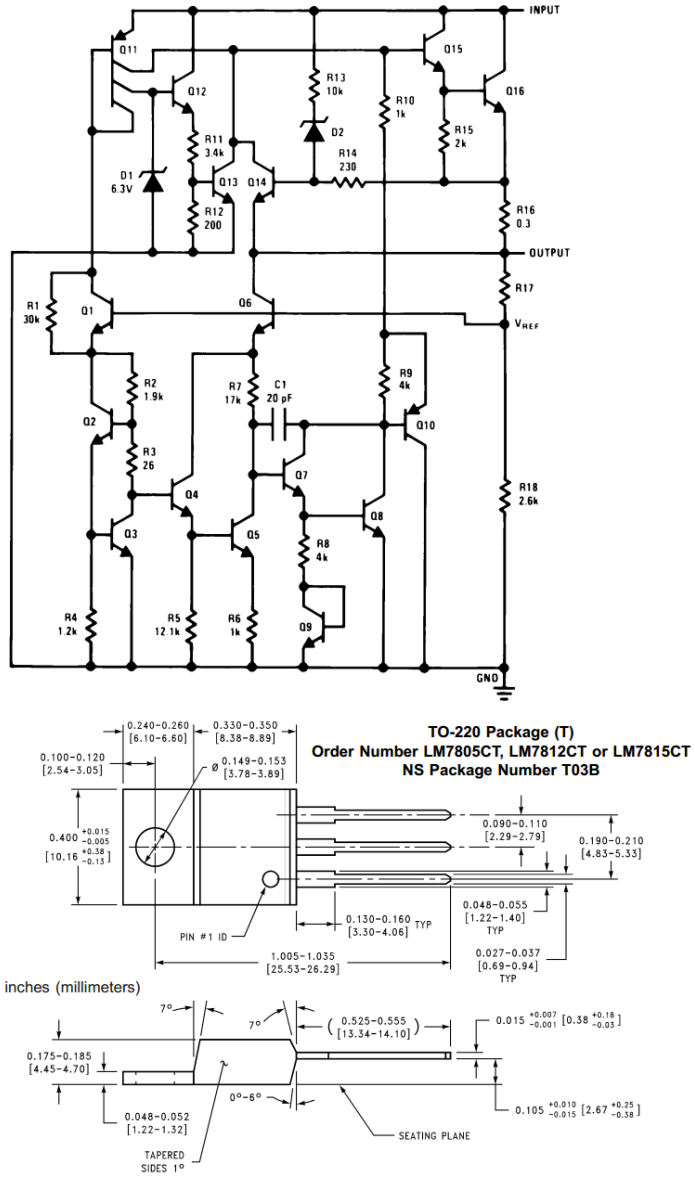


Figura 2: circuito integrado de tipo LM78XXCT.

Sin embargo y como se expone a medida que se desglosa el diseño del hardware en los siguientes apartados, el hardware se enfrenta al reto de estar integrado dentro de una carcasa. Esto añade cierta complejidad a su diseño debido a que todos los componentes electrónicos tienen tolerancias en sus dimensiones físicas y bastan unas décimas de milímetro de error en la relación entre un conector que sobresale de la carcasa y su hueco en la misma para que la carcasa y la PCB sean incompatibles.

1.2.2 – OBJETIVOS EN EL DISEÑO DE SOFTWARE

Como en el caso del hardware, la complejidad del software depende de la aplicación para la que es desarrollado. Sin embargo en este proyecto se va a tratar el diseño de software para un sistema embebido en tiempo real, lo que implica que son necesarios ciertos conocimientos de la plataforma sobre la que se ejecuta el software y la complejidad añadida de un sistema que depende (además significativamente) del tiempo.

Debido a esas razones se pondrá especial énfasis en el diseño de un diagrama de flujo de software que ilustre de manera secuencial el funcionamiento del sistema y las tareas que funcionarán en modo “background” ya que no será posible que todas las funciones del sistema operen bajo los parámetros de diseño si no se crea un sistema “multitarea” aunque en la práctica se use un solo núcleo computacional y por las instrucciones se ejecuten de manera unitaria y secuencial.

CAPÍTULO 2 – REQUISITOS A CUMPLIR POR EL HARDWARE

2.1 – CARACTERÍSTICAS FUNDAMENTALES DEL SISTEMA

El sistema estará embebido dentro de una carcasa, la cual es especificada en el apartado 2.1.2. Por tanto sus dimensiones deberán satisfacer las de la carcasa con arreglo a la distribución de conectores en la misma. Sin embargo el número de PCBs y su naturaleza (tecnología de fabricación y formato de componentes electrónicos) serán las que mejor satisfagan las características requeridas.

Todos los conectores empleados deberán ser estandarizados y deberán ser amigables con el futuro usuario, esto es, se evitarán conectores que no sigan la filosofía de "plug & play". Además deberán ser conectores que sólo puedan ser conectados en una posición, limitándose el resto de posiciones de manera física. Debido a esto no se empleará en el hardware ningún sistema de protección contra malas conexiones, será responsabilidad del usuario que el sistema sea dañado si se hace una mala conexión forzando el conector o usándolo indebidamente.

Esta filosofía obedece a una máxima eficiencia energética y abaratamiento de costes: no es necesario invertir en seguridad si por características intrínsecas del sistema no se pueden producir situaciones peligrosas.

Por otra parte, todos los periféricos de la PCB convergerán en un microcontrolador que será el encargado de ejecutar las tareas necesarias. Se da por hecho que se seleccionará un microcontrolador para esta tarea dada la naturaleza del sistema ya que no es necesaria una gran capacidad computacional ni es necesario almacenar gran cantidad de datos. Esto hace inviable desde el punto de vista económico implementar un SBC para gobernar el sistema a través de algún sistema operativo embebido.

2.1.1 – COSTE UNITARIO

El coste es un factor fundamental dado que es un producto que se comercializará en el competitivo mercado de los equipos industriales. Por tanto se buscará un adecuado compromiso entre prestaciones y el precio del sistema final. Los costes del equipo pueden estimarse en base a la fabricación mínima de 500 unidades dado que existen descuentos significativos para adquirir componentes electrónicos en función de la cantidad de los mismos y se recogen en los Anexos A y B.

2.1.2 – CARCASA Y DISTRIBUCIÓN DE ELEMENTOS EN LA MISMA

Para el diseño del mecanizado de la carcasa se partirá de la carcasa del fabricante WÖHR mostrada en la Figura 3, que corresponde a un modelo de carcasas de carril DIN cuyas dimensiones, características técnicas y precio resultan interesantes para SEL16.

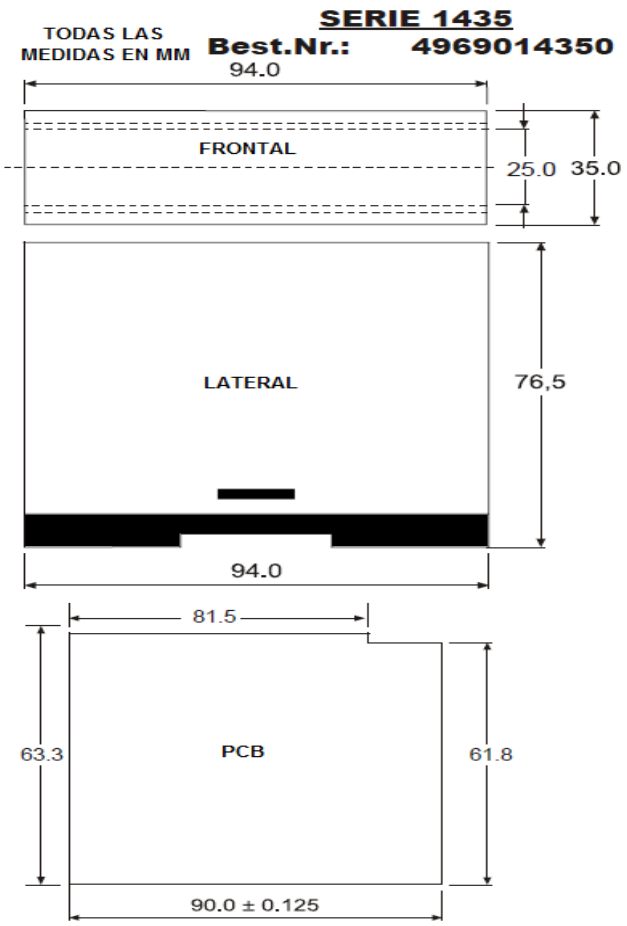


Figura 3: carcasa para SEL16 y FLX01.

Como la carcasa debe tener una distribución de elementos lógica y bajo esa premisa se ha diseñado su mecanizado completo y la etiqueta correspondiente a IO5-USB, expuestos en las Figuras 4, 5 y 6. Además las indicaciones luminosas mediante LEDs siguen un orden que respeta una estética adecuada y que son fácilmente interpretables con un rápido vistazo.

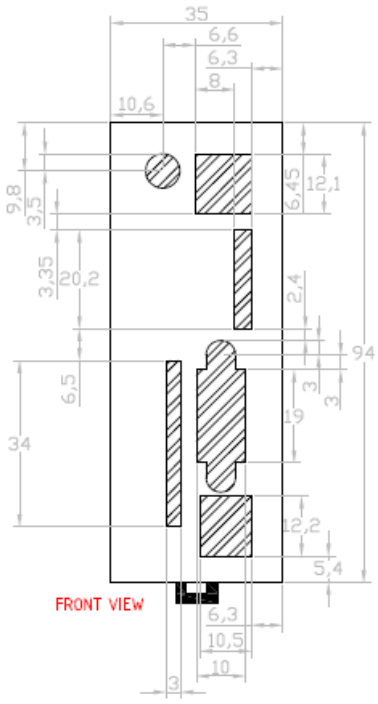


Figura 4: medidas del mecanizado frontal de la carcasa.

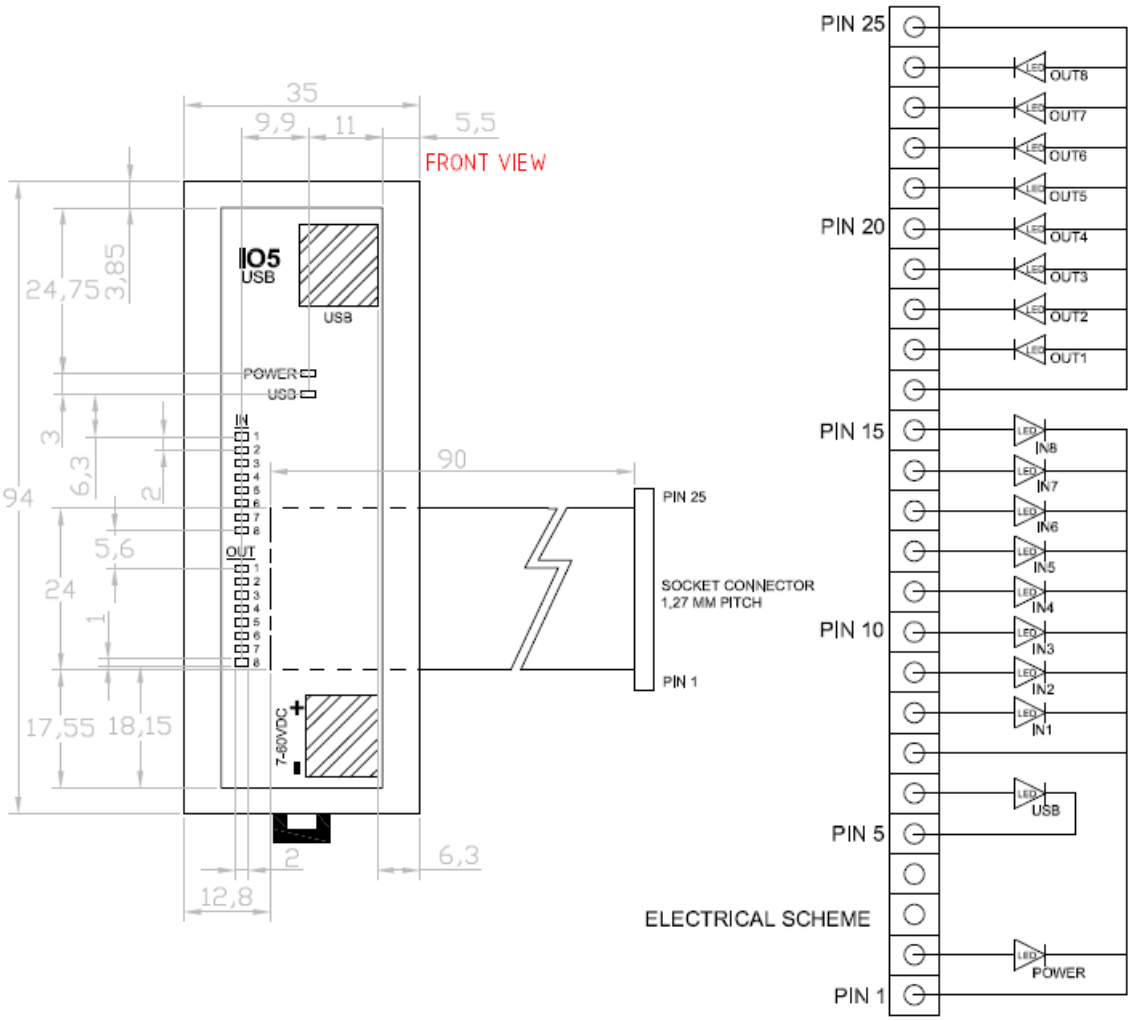


Figura 5: esquema del frontal de IO5-USB con FLX01.

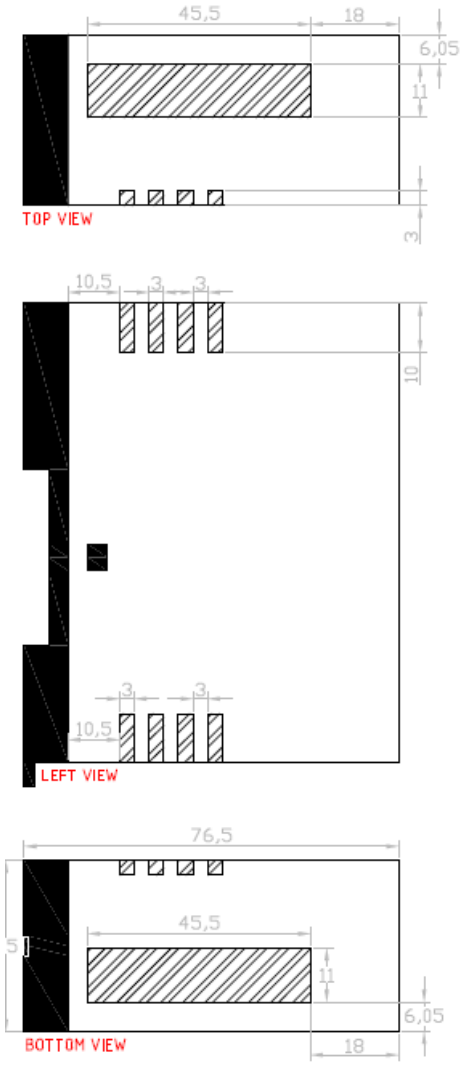


Figura 6: mecanizado de los laterales de la carcasa.

2.1.3 – PARÁMETROS DE FUNCIONAMIENTO

En los sucesivos sub-apartados se irán exponiendo los distintos parámetros sobre los que se va a diseñar el sistema. Los productos equivalentes actualmente existentes en el mercado sirven para trazar las características generales, las cuales deberán ser igualadas o mejoradas respetando los costes.

Todas las condiciones expuestas representan siempre los mínimos para que el producto cumpla con su cometido dentro del mercado industrial. Si el coste de la mejora es razonable se considera adecuado mejorar las características mínimas exigidas.

2.1.3.1 – CARACTERÍSTICAS DE LA ALIMENTACIÓN ELÉCTRICA

El sistema se podrá alimentar a partir de las tensiones de corriente continua más comunes en el ámbito industrial: 12, 24 y 48 voltios. De ello se deduce que el rango mínimo de tensiones será de 12 a 48 voltios, respetando un pequeño margen que situará como requisito que se pueda suministrar al sistema como mínimo de 10 a 50 voltios en corriente continua. La corriente de salida del sistema de regulación será la necesaria para cubrir las necesidades del circuito cuando esté funcionando en el modo más exigente energéticamente hablando.

La eficiencia energética del sistema será otro parámetro a tener en cuenta. El consumo eléctrico de sistemas similares puede servir como una referencia a la hora de encontrar un compromiso adecuado entre el rendimiento del sistema de alimentación eléctrica y su coste.

En cuanto a la naturaleza del sistema de alimentación no será imprescindible que el sistema esté aislado galvánicamente ya que es un sistema de alimentación de corriente continua de nivel industrial. Se prevé que el sistema que ofrece la tensión de entrada esté aislado y por tanto no procede aumentar el coste del sistema y reducir espacio en la PCB para conseguir un segundo aislamiento con un transformador eléctrico, obligando al sistema a funcionar con una fuente de alimentación conmutada de topología aislada.

2.1.3.2 – CARACTERÍSTICAS DE LAS ENTRADAS DIGITALES

Las entradas digitales serán de 12 a 24 voltios de corriente continua como mínimo. Además el sistema podrá diferenciar entre flancos con una precisión mínima de un milisegundo, luego el diseño de las entradas estará fuertemente influenciado por esta característica.

No es imprescindible que las entradas estén aisladas eléctricamente del exterior aunque se podría tener en cuenta una versión con estas características para satisfacer necesidades específicas de los clientes que lo soliciten, aunque a cambio se sacrifiquen otros parámetros de las entradas.

2.1.3.3 – CARACTERÍSTICAS DE LAS SALIDAS DIGITALES

Las salidas digitales ofrecerán una tensión que el usuario elegirá desde el exterior, comprendida como mínimo entre 12 y 24 voltios al igual que las entradas. Que las salidas y las entradas tengan niveles similares no es estrictamente necesario siempre que se cumplan los requisitos mínimos.

2.1.4 – CONECTIVIDAD

El sistema podrá ser programado y gestionado por tres sistemas de conexión diferentes: una conexión USB, una RS-422/485 y a través de la red de telefonía móvil. La conexión USB deberá cumplir al menos con USB 2.0 (naturalmente todas las versiones anteriores de USB deberán ser soportadas). La interfaz USB deberá ser compatible con Windows XP, Vista, 7 y 8. La gestión del sistema desde el ordenador se hará a través de un entorno de software que usará los comandos que se definan para el equipo.

La conexión RS-422/485 será complementaria con la de USB, no siendo implementadas ambas en el mismo equipo. Por tanto se podrá elegir a nivel de hardware mediante selectores y según los circuitos integrados en cada PCB cuál será el sistema de comunicación “in situ” del sistema: USB o RS422/485.

Para la conexión a través de la red de telefonía móvil deberá existir como mínimo la posibilidad de interactuar con el sistema a través de SMS y el estándar de comunicación GPRS. La comunicación a través de 3G sería deseable si bien la conexión a través de la red móvil del sistema no es necesaria para IO5-USB y por tanto sólo se necesita su soporte a nivel hardware para una implementación futura en otros productos.

2.1.5 – LEDS INDICADORES PARA EL USUARIO

El sistema contará como mínimo con un LED indicador de suministro eléctrico, otro indicador de comunicación a través de la red de telefonía móvil, otro indicador de comunicación USB y un LED para cada entrada y salida digital. En total son, como mínimo, 19 LEDs.

El LED indicador de suministro eléctrico seguirá un sistema simple de funcionamiento: si el equipo está alimentado estará encendido y si no está alimentado estará apagado. El LED indicador de comunicación a través de la red de telefonía móvil tendrá distintos modos de parpadeo según la tarea. Esto es de libre configuración siempre que sea intuitivo y sencillo de interpretar por el usuario. El LED indicador de comunicación USB parpadeará cuando haya comunicación USB efectiva y los LEDs de las entradas y salidas se iluminarán cuando una entrada o salida esté a nivel lógico alto, estando apagado cuando sea nivel lógico bajo.

2.1.5.1 – PCB FLEXIBLE CONECTADA A SEL16: FLX01

Situar los LEDs en la carcasa representa un problema ya que las posiciones para los mismos están alejadas de la PCB. Sin embargo existe la posibilidad de solucionarlo usando una PCB flexible que, a modo de bus de comunicaciones, vaya desde la placa hasta la carcasa, llevando sólo los cableados de los LEDs. Esta solución es muy práctica y tecnológicamente interesante ya que progresivamente se van incorporando al mundo de los sistemas electrónicos las PCBs flexibles, consiguiendo solucionar problemas como el que se ha presentado en este diseño y que sólo podían ser solucionados soldando los LEDs con cables largos que comunicaran la posición final del LED con su correspondiente PCB.

2.1.6 – CONSIDERACIONES DE CARA A APLICACIONES POTENCIALES

Determinados requisitos expuestos como las conexiones a través de RS-422/485 y a través de la red de telefonía móvil son sólo posibilidades futuras de uso que deben estar previstas en el hardware. Por tanto las mismas estarán habilitadas o no en función del número de circuitos integrados que se usen en la fabricación de la PCB y de la configuración de jumpers o sistemas de selección de conexiones equivalentes cuyo propósito sería el de conectar el hardware correspondiente con el microcontrolador empleado. Esta solución está enfocada en aprovechar la condición de que no se usarán en la misma PCB el puerto RS-422/485 y el USB a la vez: son complementarios.

Si el puerto de programación del microcontrolador empleado debiera usarse para otra función, también se podrá seleccionar entre programar el sistema y la utilidad alternativa a través de jumpers.

CAPÍTULO 3 – DISEÑO, SELECCIÓN Y MODELADO DEL HARDWARE

3.1 – DIAGRAMA DE HARDWARE DE LA SOLUCIÓN

Para diseñar el hardware es adecuado basarse en un esquema general que represente en bloques los distintos módulos del sistema y cómo interaccionan con los demás. Dado que en este caso se trata de un sistema que va a tener dos PCBs (para lidiar con el problema de los LEDs en la carcasa), la PCB auxiliar (llamada FLX01 y que será flexible) aparecerá representada dentro del bloque "CIRCUITO DE LEDs" del esquema de la Figura 7. El conjunto de todos los bloques representa la solución de hardware para IO5-USB.

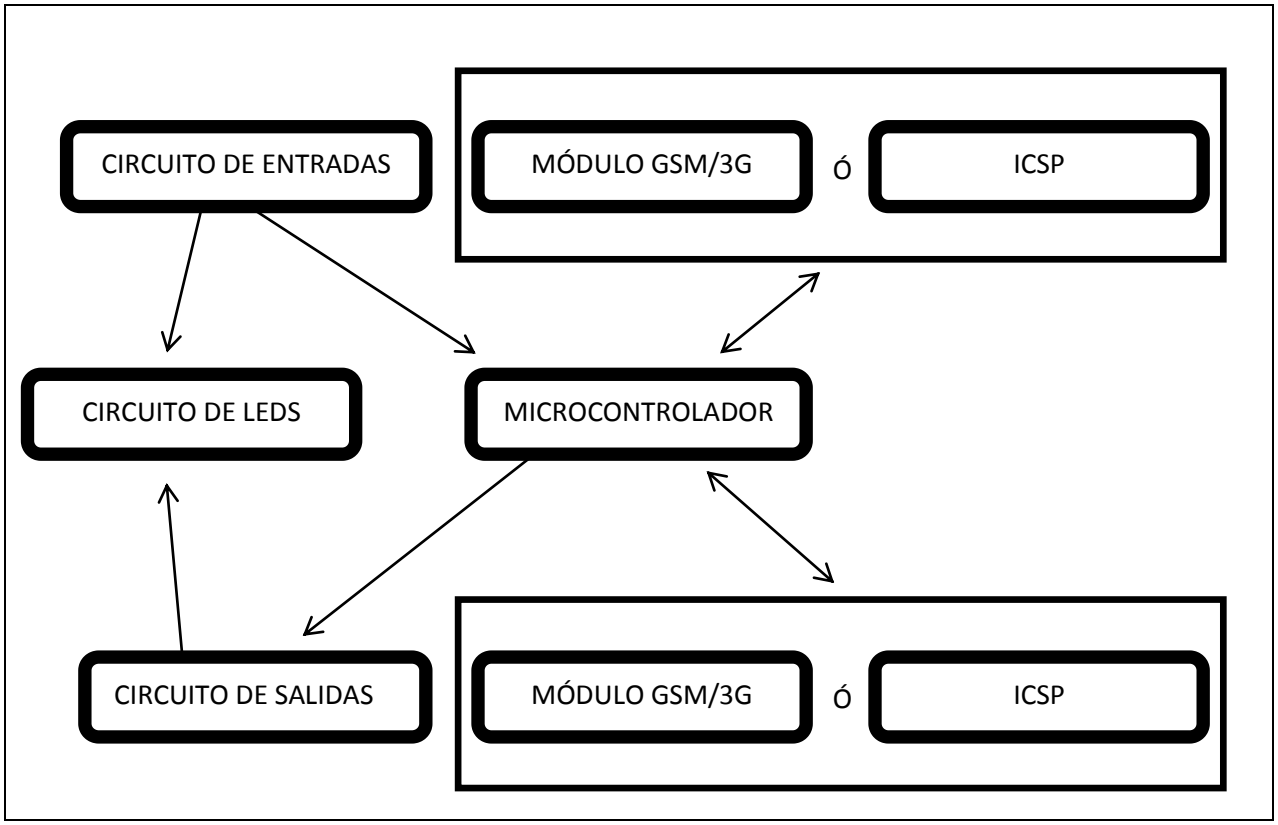


Figura 7: interacción entre módulos de la PCB de IO5-USB.

3.2 – SOFTWARE PARA EL DISEÑO DE HARDWARE: CADSOFT EAGLE 4.16r2

El paquete de software empleado para el diseño del hardware es CADSOFT EAGLE, versión 4.16r2. Si bien la versión ya es relativamente antigua (dado que la más moderna es la versión 6.4.2 que data del 30 de abril de 2013) es la versión de la que se dispone la licencia para el diseño profesional. Aunque la versión original era 4.1, CADSOFT ha ido sacando actualizaciones para las versiones antiguas, haciéndolas compatibles con los nuevos sistemas operativos y parcheando los fallos más evidentes hasta alcanzar a fecha de este proyecto la versión 4.16r2.

Que la versión no sea muy moderna no es en el fondo un problema dado que las capacidades de todas las versiones de EAGLE son muy similares y lo que progresa de una a otra es la facilidad para ejecutar ciertas tareas (creando botones que simplifican varias tareas que se hacían manualmente, habilitando la edición de coordenadas de elementos en tiempo real…). Por otra parte, en todas sus versiones EAGLE es un software de diseño electrónico muy conocido, que comparte tanto usuarios aficionados como profesionales.

3.3 – REGLAS DE SELECCIÓN DE COMPONENTES DISCRETOS PARA LA PCB

Como en la mayoría de casos del diseño de hardware (con la notable excepción de la industria aeroespacial) se busca un equilibro entre costes y prestaciones la hora de seleccionar componentes electrónicos. Interesa priorizar el uso de componentes que se ajustan a las condiciones de funcionamiento del sistema y que ofrezcan precios ventajosos, no usar componentes que, aunque sean económicamente baratos en comparación con sus prestaciones, salga más caro que el que tenga menos prestaciones pero se ajuste a las necesidades del sistema.

Se priorizarán los componentes que estén disponibles en formato SMT, que favorecen el aprovechamiento de la PCB. Además se tratará de usar ICs que integren varias funciones, de modo que se ahorre tanto en espacio de la PCB como en coste.

Aunque existen múltiples fuentes para tener una idea de lo que cuesta un componente electrónico en función del número de unidades a comprar como los distribuidores con páginas web de búsqueda de componentes FARNELL o RS COMPONENTS (por mencionar ejemplos), la realidad suele ser distinta a la hora de la fabricación, sobre todo si ésta se hace en China. Esta clase de fabricantes suelen trabajar con grandes stocks y reducidos márgenes de beneficio, por tanto son capaces de conseguir componentes a precios muy ventajosos que suelen rondar, a modo informativo, la mitad de los precios ofrecidos por los distribuidores mencionados.

3.4 – DISEÑO, SELECCIÓN Y MODELADO DEL HARDWARE DE SEL16

En este apartado se procede a la selección de componentes y a su modelado para usarlos en el diseño. El modelado del hardware seguirá unas directrices conservadoras: en caso de que el fabricante no proporcione un footprint recomendado éste se diseñará a partir del tamaño máximo de las dimensiones que aporte el fabricante (incluyendo tolerancias). De esta forma aunque se use más espacio en la PCB tendremos la garantía de que no habrá problemas a la hora de colocar los componentes. En cualquier caso esta pérdida de espacio en la PCB es bastante irrisoria.

3.4.1 – SELECCIÓN DEL MICROCONTROLADOR

Se da por hecho que el componente que permite una relación óptima en cuanto a prestaciones y precio para el manejo de un sistema de la complejidad de IO5-USB es un microcontrolador. Recurrir a un microprocesador o incluso un SBC sería ofrecer una solución sobredimensionada que repercutiría en el coste y en el consumo eléctrico del equipo sin ofrecer mejores prestaciones que un microcontrolador para las tareas de IO5-USB.

Se va a optar por un microcontrolador de 8 bits dado que es suficiente para las tareas que hay que ejecutar en el sistema. Existen en la actualidad varias familias de microcontroladores de 8 bits, si bien los fabricantes más conocidos son ATMEL, INFINEON, MICROCHIP TECHNOLOGY, NXP SEMICONDUCTORS y STMICROELECTRONICS. De ellos se elegirá un microcontrolador tipo PIC de MICROCHIP TECHNOLOGY, que son los más conocidos dada la sencillez y robustez que presentan.

Además MICROCHIP TECHNOLOGY cuenta con un enorme abanico de microcontroladores PIC de 8 bits que permite elegir el más adecuado en función de las necesidades del sistema. Además son microcontroladores con un alto grado de interoperabilidad ya que el software de funcionamiento es igual para una misma familia, pudiendo elegir entonces el microcontrolador más barato que se ajuste a las necesidades de la aplicación sin necesidad de aprender una nueva arquitectura o lenguaje de programación.

En lo referente a la selección del modelo de microcontrolador, se pueden usar diversos criterios, los cuales también están condicionados por el tipo de proyecto y la fase en la que se encuentra. Si es imperativo hace la selección del hardware para desarrollar en paralelo el hardware y el software del futuro producto, hay desarrolladores que usan la experiencia que tienen en programación para elegir el modelo de microcontrolador a usar, dependiendo de los recursos que estiman necesarios en el microcontrolador como memoria de programa o de datos. Los microcontroladores PIC son adecuados para este tipo de selección ya que con el mismo footprint hay distintos modelos con distintas capacidades, facilitando esta tarea y minimizando la posibilidad de que se presente el problema de que hay que cambiar de microcontrolador una vez ya se ha diseñado el hardware.

Si el diseño del hardware puede esperar a que el software esté casi completo, un criterio que deja menos a la experiencia y más a los hechos es que cuando el software se ha completado se sabe cuánta memoria es necesaria. Es entonces cuando se puede seleccionar con criterio el microcontrolador.

En este proyecto, en el momento de seleccionar el microcontrolador me encontraba en el primer supuesto descrito: para minimizar el tiempo de desarrollo total del producto, era preciso diseñar el hardware y enviarlo a fabricación para que mientras era fabricado poder desarrollar el software. Por tanto, haciendo uso de mi experiencia en microcontroladores y teniendo en cuenta los requisitos de IO5-USB la selección más adecuada de las familias de microcontroladores de 8 bits de MICROCHIP TECHNOLOGY era la familia PIC18(L)F2X/4XK22 y en concreto el modelo PIC18LF26K22 (ilustrado en la Figura 8).

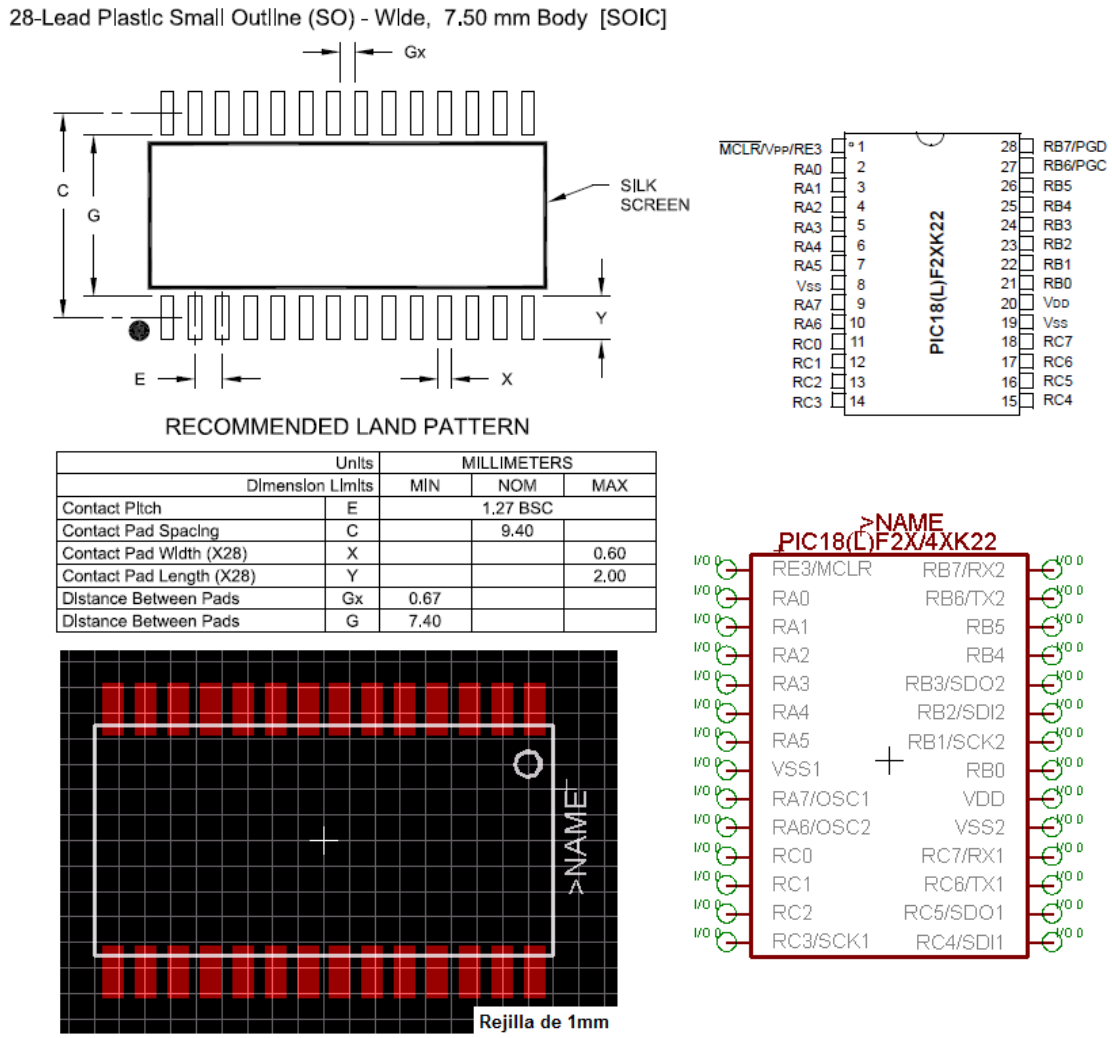


Figura 8: modelado de PIC18LF26K22.

Esta familia de microcontroladores satisfacía la necesidad de poder cambiar la cantidad de memoria EEPROM, memoria de programa y memoria de datos sin necesidad de cambiar el footprint, sólo cambiando el modelo. Además la familia dispone de 2 EUSART, funcionamiento a 3,3 o 5 voltios (letra L en el nombre del modelo), oscilador interno programable y otras características relevantes que se muestran en la tabla. En el Anexo D se encuentran las características más significativas de este PIC.

3.4.1.1 – OSCILADOR DEL MICROCONTROLADOR

La selección de un oscilador externo para el microcontrolador obedece a la precisión temporal que se espera del microcontrolador en su funcionamiento. Usar el oscilador interno es posible pero el oscilador interno cuenta con una precisión menor que el oscilador externo y es mucho más sensible a los cambios de temperatura.

Debido a que SEL16 es la base para otros productos y no sólo IO5-USB, el microcontrolador será usado con un oscilador externo. De hecho, aunque el oscilador interno ofrezca unas prestaciones adecuadas para muchos usos, es recomendable usar osciladores externos cuando las tareas dependan críticamente del tiempo.

3.4.1.2 – CIRCUITO DE PROGRAMACIÓN DEL MICROCONTROLADOR

Los microcontroladores de la familia PIC18(L)F2X/4XK22 usan el sistema ICSP para programarse y depurarse (bibliografía **[3]**). Este sistema emplea como puerto de programación y depuración una EUSART del microcontrolador junto con las señales de VDD, VSS y MCLR (en la Figura 9 se representa el esquema de dicha implementación en SEL16). Por tanto esas señales deben estar accesibles en SEL16 como puerto de programación y depuración, si bien la EUSART empleada se trata de la empleada para la comunicación con el SOC TELIT GL-865 QUAD.

Aunque a veces no hay problema con que dicha EUSART esté conectada a otros dispositivos que ignorarán los datos de programación del PIC (es el caso de dispositivos que esperan órdenes en silencio y no responden ante comandos erróneos) es muy recomendable que haya una separación eléctrica entre ambas EUSART dado que el TELIT puede interferir en el proceso de programación del PIC.

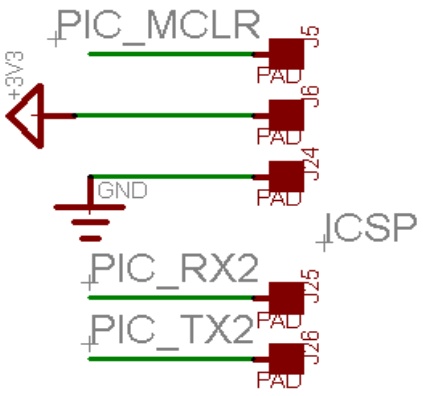


Figura 9: esquema del puerto ICSP.

3.4.2 – CIRCUITO DE ENTRADAS DIGITALES

Para conseguir unas entradas que admitan hasta 24 voltios con un coste razonable lo más adecuado es buscar algún transistor bipolar que admita una tensión entre base y emisor (o entre puerta y surtidor si se trata de un FET). De hecho, dado que la precisión de las entradas del sistema final ha de ser como mínimo de 1 milisegundo es adecuado descartar los transistores bipolares a favor de los FET.

Buscando en distintos fabricantes es posible encontrar a precios razonables transistores MOSFET que admiten entre puerta y surtidor hasta 20 voltios luego con un simple divisor resistivo a la entrada (que de paso hará de pull-down en la puerta del MOSFET y limitará la corriente de entrada) que divida su tensión por la mitad se conseguiría que la entrada soportara hasta 40 voltios (en realidad 43,3 menos la caída entre drenador y surtidor en saturación, dado que cuando la puerta sea capaz de saturar el MOSFET y en la configuración mostrada en el surtidor hay 3,3 voltios menos la caída entre drenador y surtidor). El transistor finalmente escogido es el Si4936BDY, modelado según se muestra en la Figura 10 y el esquema de las entradas resultante se ilustra en la Figura 11.

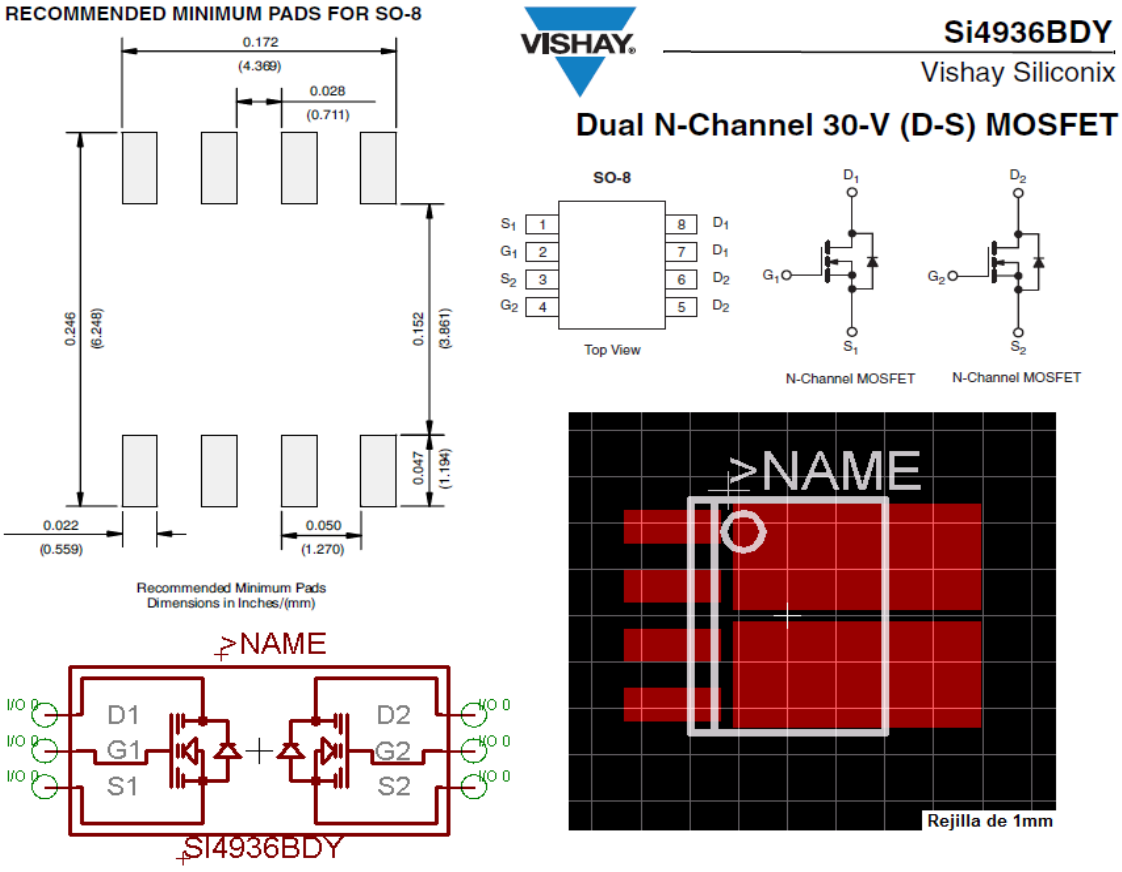


Figura 10: modelado de Si4936BDY.

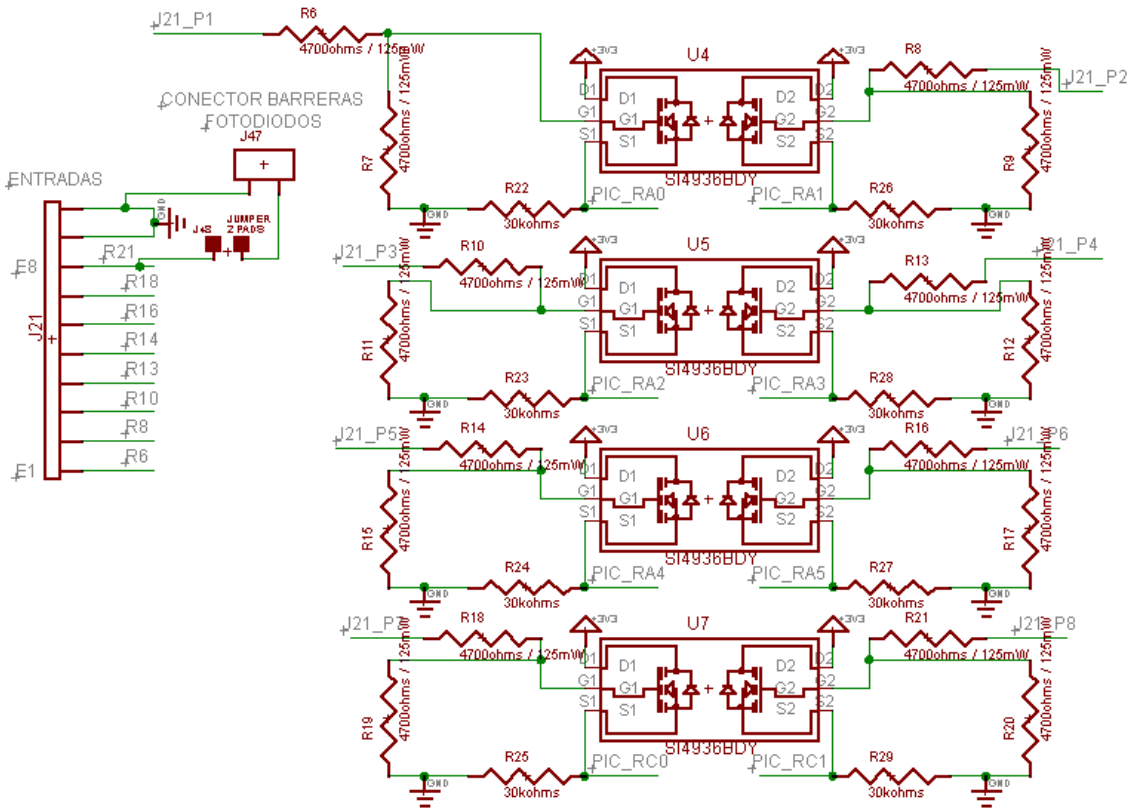


Figura 11: esquema de las entradas de SEL16.

J47 es un conector como el de la alimentación eléctrica (de dos contactos) y está conectado a la entrada 8 si se suelda J48 y su propósito está expuesto en el apartado 7.1.1. Los conectores empleados para las entradas y las salidas digitales son dos conectores iguales. El modelo escogido es mostrado en la Figura 12, que corresponde a un conector de fácil conexión y adecuado nivel de aislamiento.

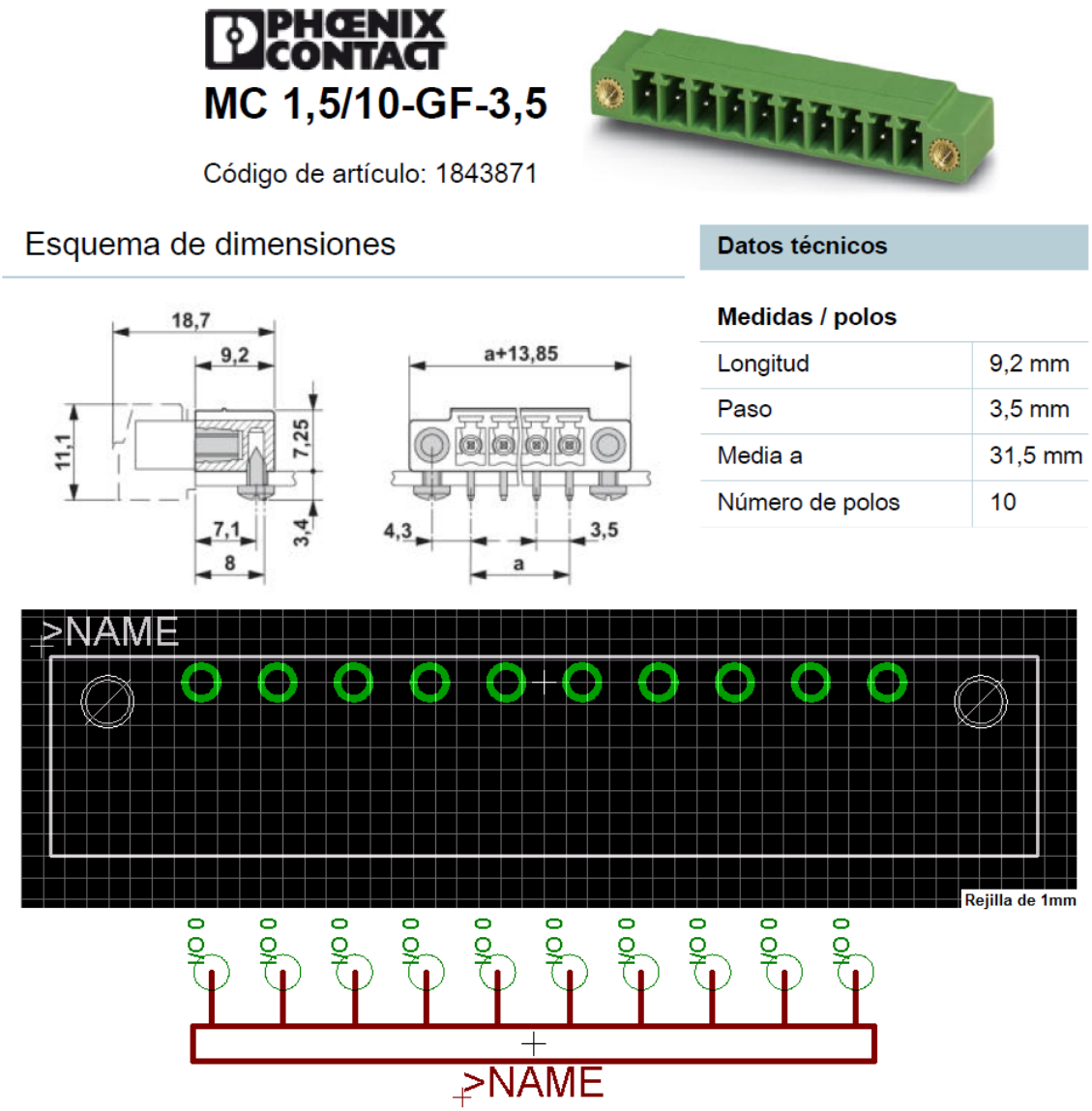


Figura 12: modelado del conector de entradas y salidas.

3.4.3 – CIRCUITO DE SALIDAS DIGITALES

Dado que las salidas deben ofrecer la tensión que prefiera el usuario (con el mínimo al igual que las entradas de 12 a 24 voltios) lo más adecuado es usar un IC del tipo high side driver (bibliografía **[6]**). De este modo, con una pequeña excitación por un lado, éste conectará la tensión seleccionada para las salidas en la salida correspondiente. Para este propósito se ha usado el IC de STMicroelectronics VN808CM-E, cuyas características están recogidas en el Anexo F. En la Figura 13 se ilustra su modelado y en la Figura 14 el esquema de hardware de gestión de las salidas de SEL16.

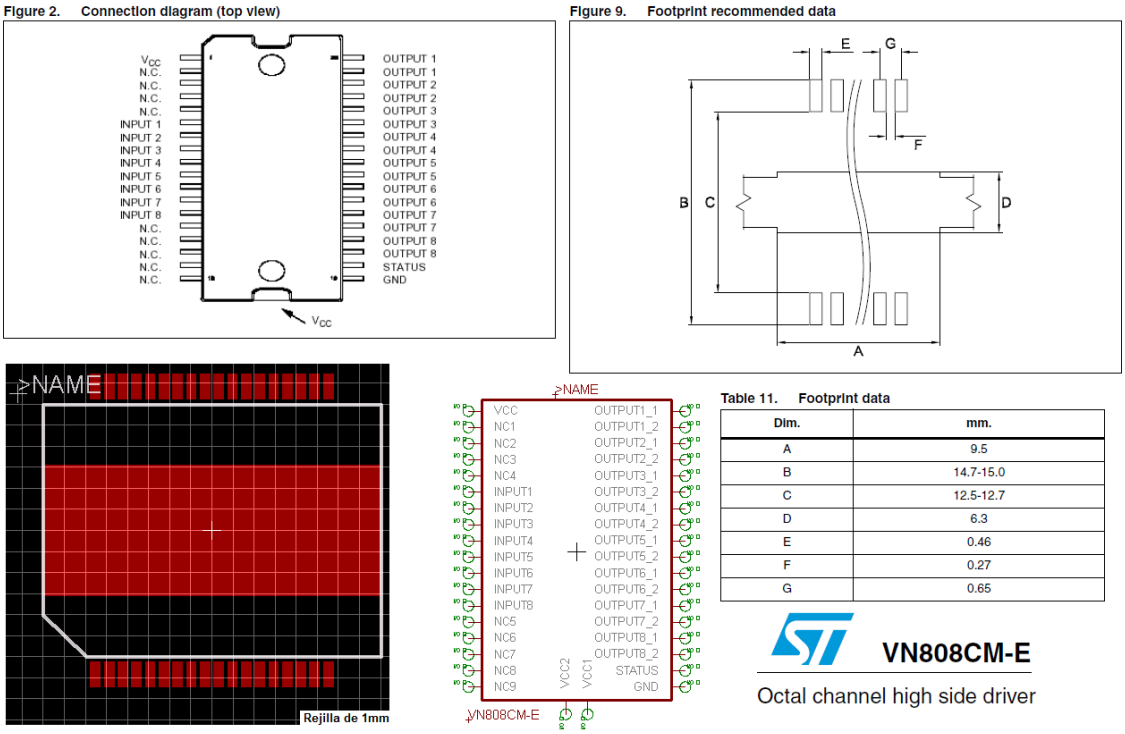


Figura 13: modelado de VN808CM-E.

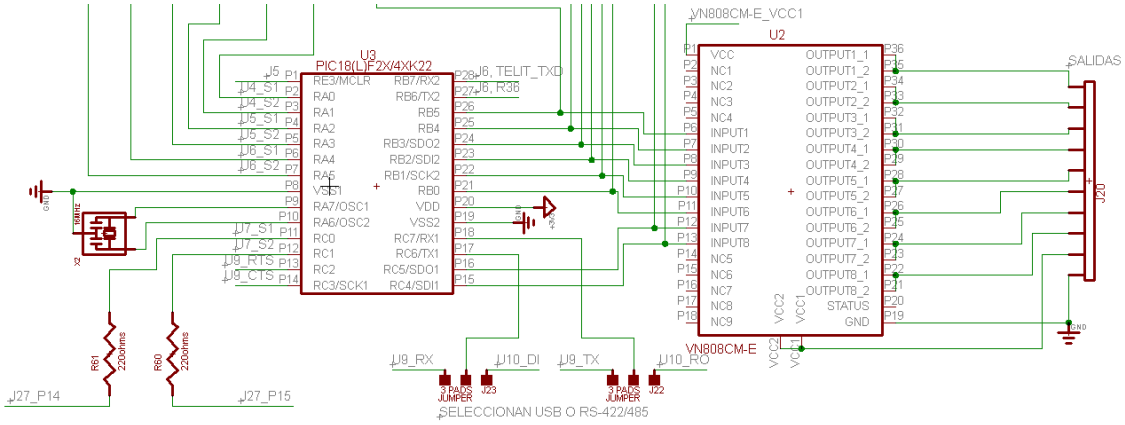
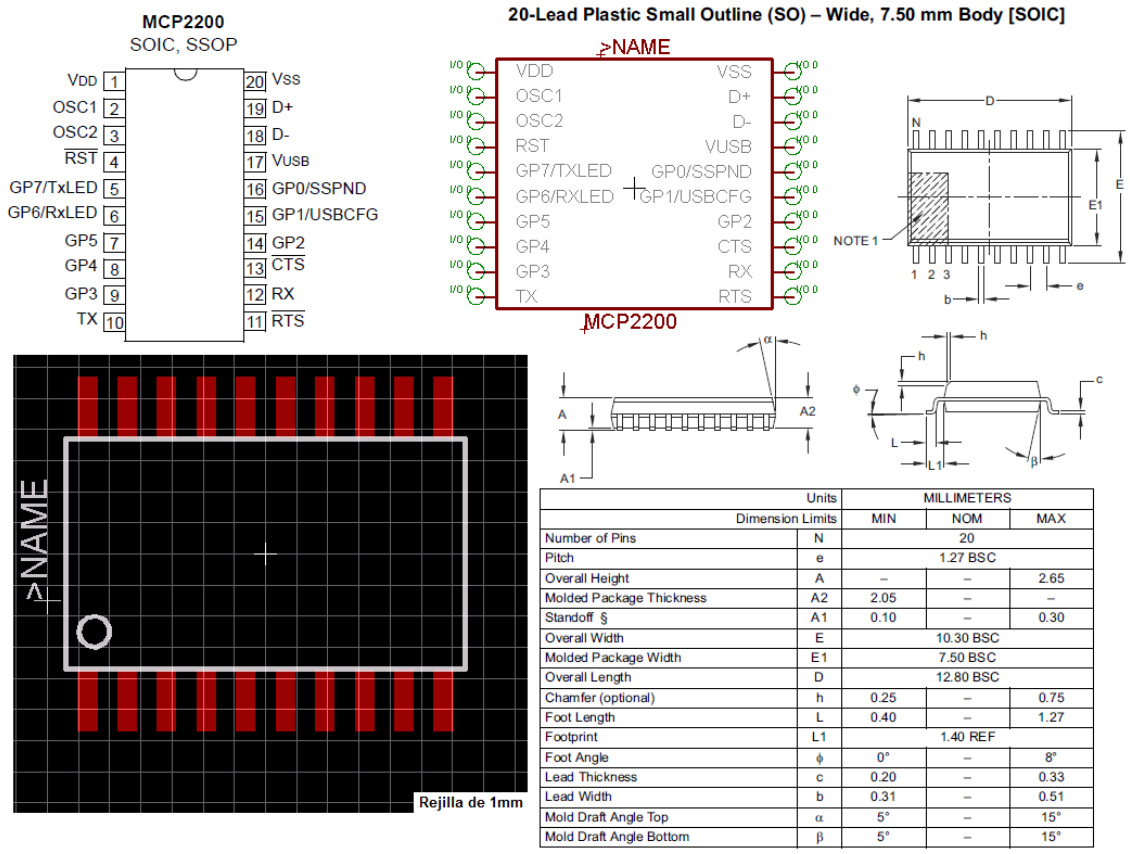


Figura 14: esquema de las salidas de SEL16.

3.4.4 – CIRCUITO DE COMUNICACIÓN USB

Los requisitos establecen que la conexión USB debe ser compatible con todos los estándares de velocidad hasta USB 2.0 luego se usará un IC encargado de transformar la EUSART correspondiente del microcontrolador de SEL16 en una conexión USB. MICROCHIP TECHNOLOGY ofrece el IC MCP2200, una solución económica y sencilla que será la empleada, sus características pueden ser consultadas en el Anexo G. En la Figura 15 se recoge el modelado de dicho IC y en la Figura 16 se muestra el esquema diseñado para el sistema USB de SEL16.



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		20	
Pitch	e		1.27 BSC	
Overall Height	A	–	–	2.65
Molded Package Thickness	A2	2.05	–	–
Standoff §	A1	0.10	–	0.30
Overall Width	E		10.30 BSC	
Molded Package Width	E1		7.50 BSC	
Overall Length	D		12.80 BSC	
Chamfer (optional)	h	0.25	–	0.75
Foot Length	L	0.40	–	1.27
Footprint	L1		1.40 REF	
Foot Angle	ϕ	0°	–	8°
Lead Thickness	c	0.20	–	0.33
Lead Width	b	0.31	–	0.51
Mold Draft Angle Top	α	5°	–	15°
Mold Draft Angle Bottom	β	5°	–	15°

Figura 15: modelado de MCP2200.

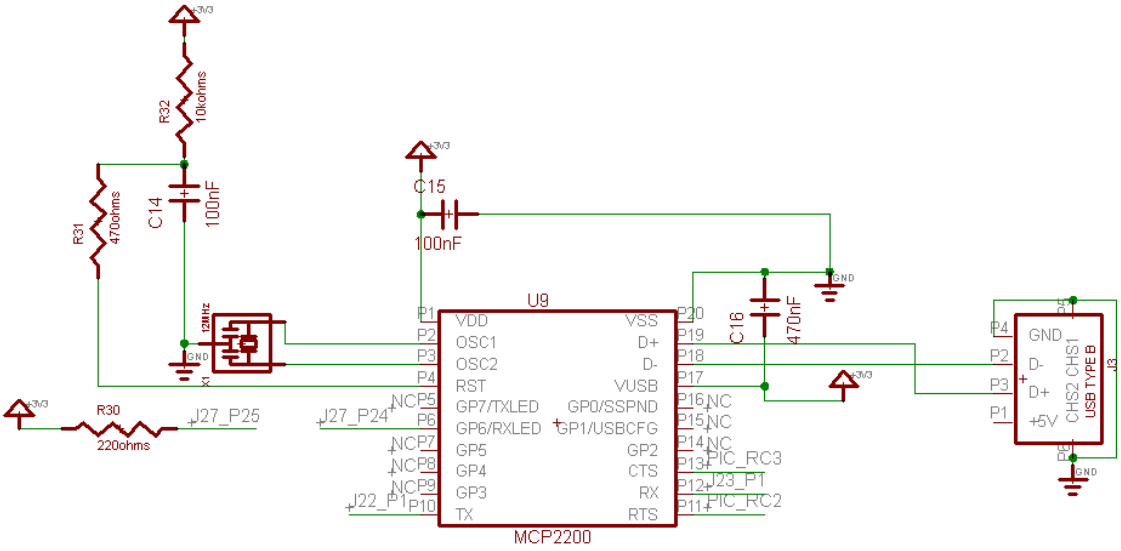
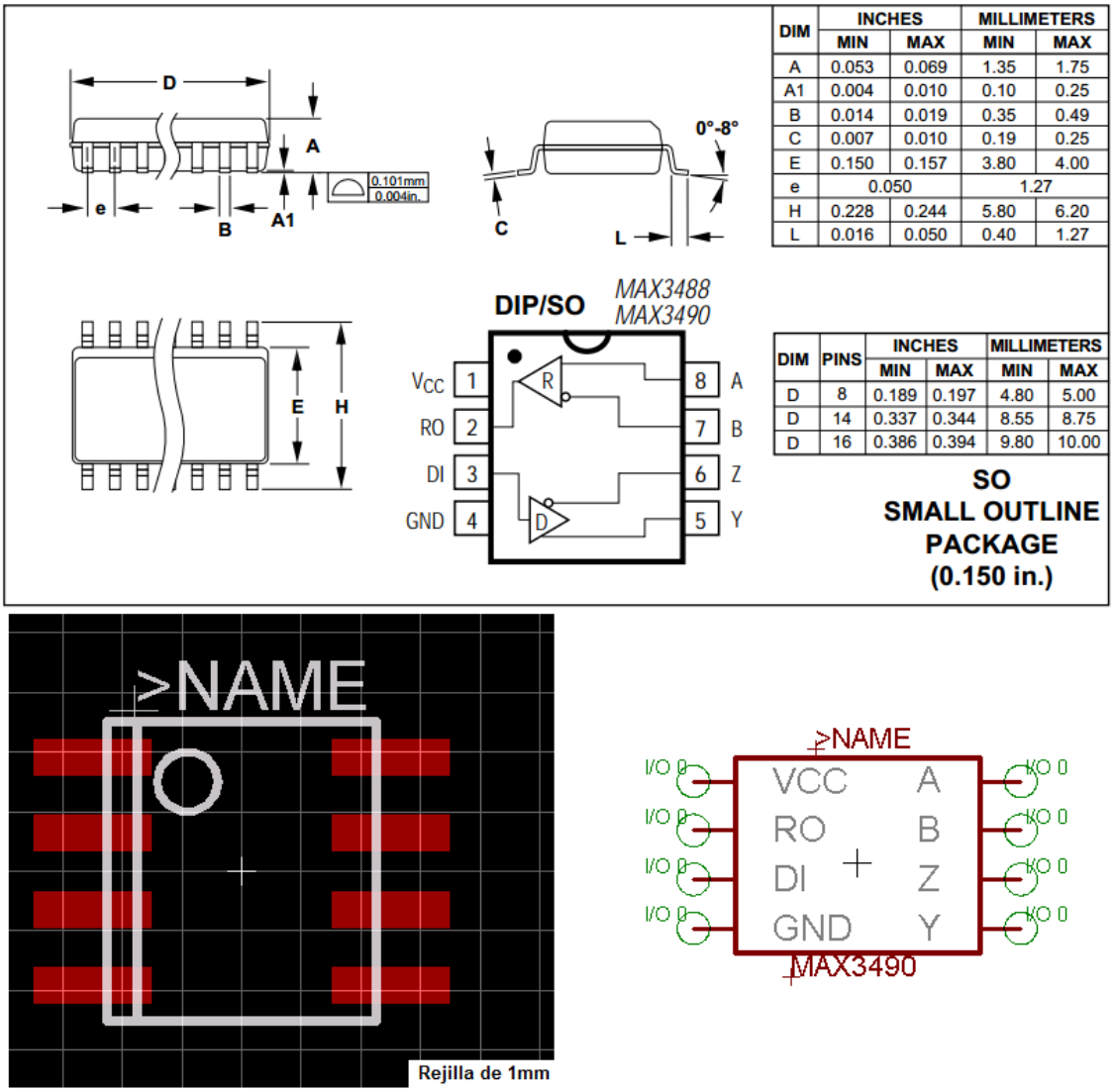


Figura 16: esquema del sistema USB de SEL16.

3.4.5 – CIRCUITO DE COMUNICACIÓN RS-422/485

Para la comunicación según los estándares RS-422/485 debemos usar un IC dedicado a convertir el formato EUSART usado por el PIC. El fabricante MAXIM INTEGRATED dispone de una amplia gama de ICs dedicados a la conversión entre UART y distintos estándares de comunicación en serie. Por tanto, debido a la fiabilidad que han demostrado en distintos diseños propios es adecuado decantarse por el uso de un IC de este fabricante.

El IC seleccionado es el MAX3490CSA+, sus parámetros de funcionamiento más relevantes se encuentran recogidos en el Anexo H (bibliografía **[5]**) y su modelado en EAGLE y el circuito diseñado se ilustran en las Figuras 17 y 18 respectivamente.



DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.053	0.069	1.35	1.75
A1	0.004	0.010	0.10	0.25
B	0.014	0.019	0.35	0.49
C	0.007	0.010	0.19	0.25
E	0.150	0.157	3.80	4.00
e	0.050		1.27	
H	0.228	0.244	5.80	6.20
L	0.016	0.050	0.40	1.27

DIM	PINS	INCHES		MILLIMETERS	
		MIN	MAX	MIN	MAX
D	8	0.189	0.197	4.80	5.00
D	14	0.337	0.344	8.55	8.75
D	16	0.386	0.394	9.80	10.00

Figura 17: modelado de MAX3490.

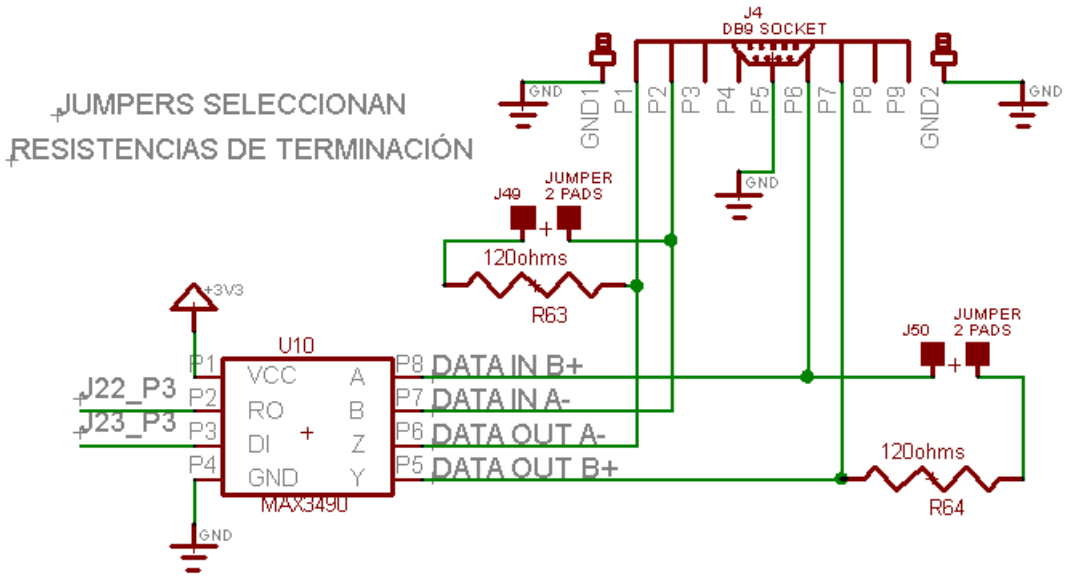


Figura 18: esquema del sistema RS-422/485 de SEL16.

3.4.6 – CIRCUITO DE COMUNICACIÓN GSM/3G

Para la comunicación a través de la red de telefonía móvil se usará el módulo GL865 QUAD del fabricante italiano TELIT ya que ofrece un sólido soporte técnico en España para resolver problemas y cuenta con varias notas de aplicación que facilitarán el diseño, sin dejar de lado que su distribuidor en España (Rutronik) ofrece precios muy competitivos y asesoramiento técnico a la hora de usar sus productos.

Gracias a las notas de aplicación de hardware (anexo I) el circuito básico para usar el GL865 QUAD a través de UART está prácticamente definido (y representado en la Figura 20) y sólo hay que eliminar los componentes discretos exteriores al módulo que se ocupan de funcionalidades innecesarias. Además, en estas notas de aplicación vienen las recomendaciones para la implementación de la comunicación (bibliografía **[4]**) entre GL865 QUAD y el conector de tarjetas SIM (que será el modelo 9002-M06 del fabricante OUPIIN).

Un problema que surge con este módulo es que su UART funciona con una tensión límite de 3,1 voltios, siendo 2,8 voltios la tensión nominal. Para que el PIC reciba datos desde la línea de transmisión del módulo TELIT no hay problema dado que 2,8 voltios siempre es tomado como nivel lógico alto por el PIC pero el PIC no puede transmitir con un nivel de 3,3 voltios al módulo TELIT ya que lo dañaría. Para sortear esto basta con situar un sencillo divisor resistivo (dado que la velocidad de transmisión de datos no es muy elevada se puede recurrir a este tipo de solución) que está implementado en la línea de transmisión desde el PIC hacia la recepción del módulo TELIT.

Con una resistencia en serie de 4700 ohmios (R36) y una en paralelo de 26100 ohmios (R37) usando la fórmula del divisor resistivo para una tensión de transmisión del PIC de 3,3 voltios se obtiene al otro lado del divisor resistivo 2,796 voltios, el nivel requerido por el módulo TELIT. En la Figura 19 se recoge el modelado de dicho módulo.

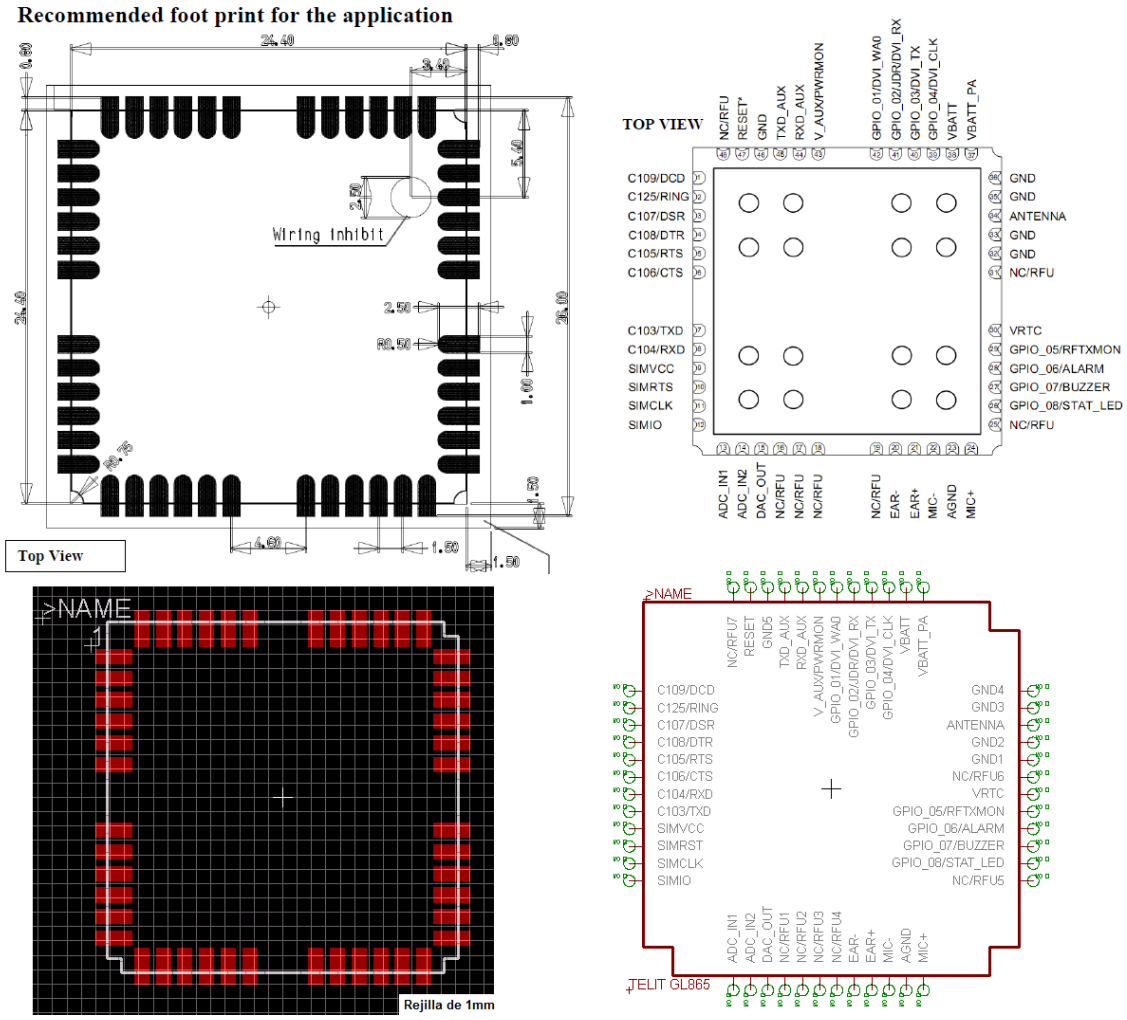


Figura 19: modelado de GL-865 QUAD.

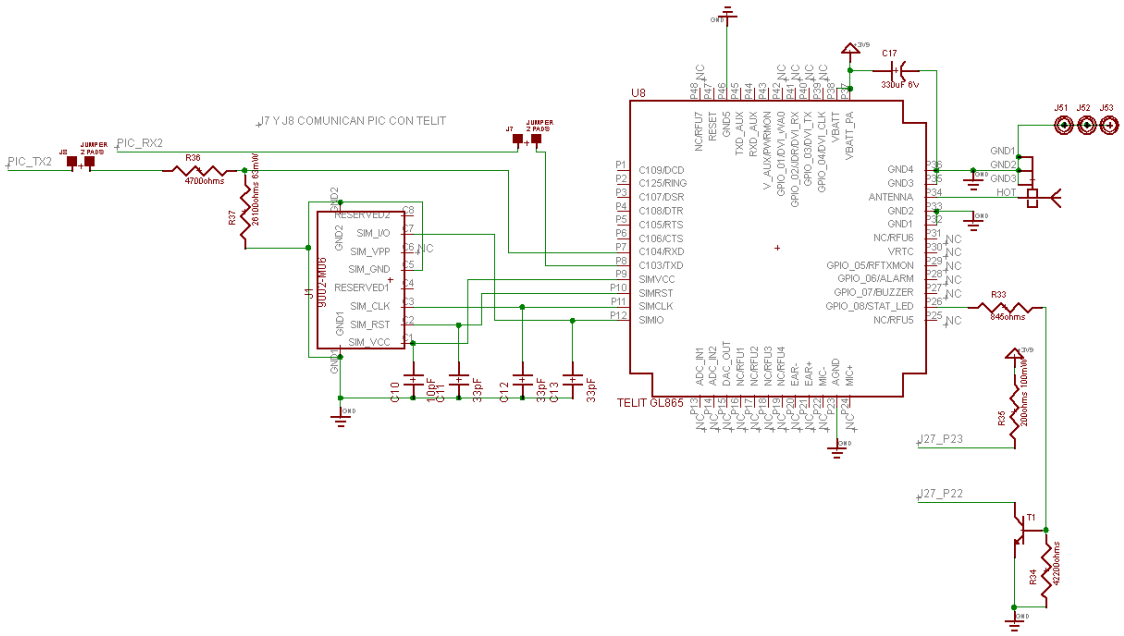


Figura 20: esquema del sistema GSM/3G de SEL16.

3.4.7 – CIRCUITO DE ALIMENTACIÓN ELÉCTRICA

Regular la tensión eléctrica del circuito hasta los niveles de funcionamiento de cada componente es una tarea fundamental del diseño de circuitos electrónicos. Sin embargo estos circuitos no son triviales cuando la eficiencia energética es un factor importante o cuando son necesarios más de 500 miliamperios en la salida y/o la caída de tensión entre la entrada y la salida es de más de 30 voltios.

Esos umbrales los da la experiencia en el diseño de circuitos de alimentación eléctrica y son precisamente los umbrales entre un regulador de tipo lineal y uno conmutado. Aun así ya empiezan a ser asequibles reguladores de tensión conmutados de bajo perfil y bajo consumo eléctrico como el IC MICROCHIP TECHNOLOGY MCP1603, que en sus versiones de tensión de salida prefijada sólo necesitan un condensador de tipo X7R en la entrada, otro en la salida y un pequeño inductor de potencia. Estas soluciones empiezan a competir incluso con los reguladores lineales en sistemas de bajo consumo, ya que su complejidad y su coste económico dejan de ser su desventaja frente a los reguladores de tensión lineales en estas aplicaciones, manteniendo la gran ventaja de la eficiencia energética (y por ende un menor tamaño), especialmente en los modelos síncronos.

Sin embargo en el caso de SEL16 el rango de tensión de alimentación es de 10 a 50 voltios respetando los márgenes de funcionamiento, lo que requerirá de un sistema un tanto más complejo al ofrecido por el IC MCP1603. Además se requieren dos tensiones de salida: 3,3 y 3,9 voltios de corriente continua. Esto es una dificultad añadida dado que la caída entre ambas tensiones es relativamente pequeña pero suficiente como para sobrepasar los umbrales de funcionamiento de los IC presentes: ni los de 3,3 voltios soportan 3,9 voltios ni el IC GL865 QUAD (el único que funciona a 3,9 voltios) soporta el funcionamiento a 3,3 voltios. Como la caída es tan pequeña no es posible usar un primer regulador hasta 3,9 voltios y de esa tensión obtener los 3,3 voltios ya que ni los reguladores de tipo "low dropout" toleran caídas tan bajas de una manera adecuada.

Existen dos soluciones de tipo "chapuza": la primera es emplear un diodo (o varios en serie) para que, con su caída de tensión en polarización directa (que ronda precisamente la caída de tensión necesaria en el caso de muchos diodos rectificadores estándar) se disponga en el cátodo de la tensión necesaria pero la caída de tensión en los diodos depende directamente de la corriente que circula a través de ellos y su temperatura luego la regulación de tensión sería muy pobre. La segunda (algo análogo y ligeramente más técnico) es emplear un diodo zener en polarización inversa para el mismo propósito, disponiendo en el ánodo de la tensión necesaria pero ambas soluciones cuentan con la desventaja de que el diodo disipa mucha energía (toda la corriente de consumo circula a través de él). Esto hace que sea muy mala opción energéticamente y finalmente también económicamente y de espacio en la PCB: los diodos necesarios para esa tarea son caros y grandes.

La solución más adecuada es emplear dos reguladores de tensión conmutados de tipo Buck (ya que una topología aislada no tiene sentido dado que directamente se emplea una corriente continua ni se especifica ese requisito). Para conseguir un alto grado de integración en la PCB, existen modelos que integran dos reguladores tipo Buck en el mismo encapsulado (incluyendo los transistores MOSFET de conmutación), siendo ésta una muy buena solución. Sin embargo no son comunes los reguladores de tipo Buck que soporten hasta 50 voltios de corriente continua (para esas tensiones se empiezan a valorar las topologías aisladas como la Flyback) pero existen.

Por tanto la solución más adecuada por precio y prestaciones es el IC LINEAR TECHNOLOGY LT3988IMSE, un regulador de tipo Buck dual asíncrono que admite hasta 60 voltios de entrada, ofreciendo dos salidas regulables de hasta 1 amperio de corriente cada una. En la Figura 21 se muestra el modelado de dicho IC.

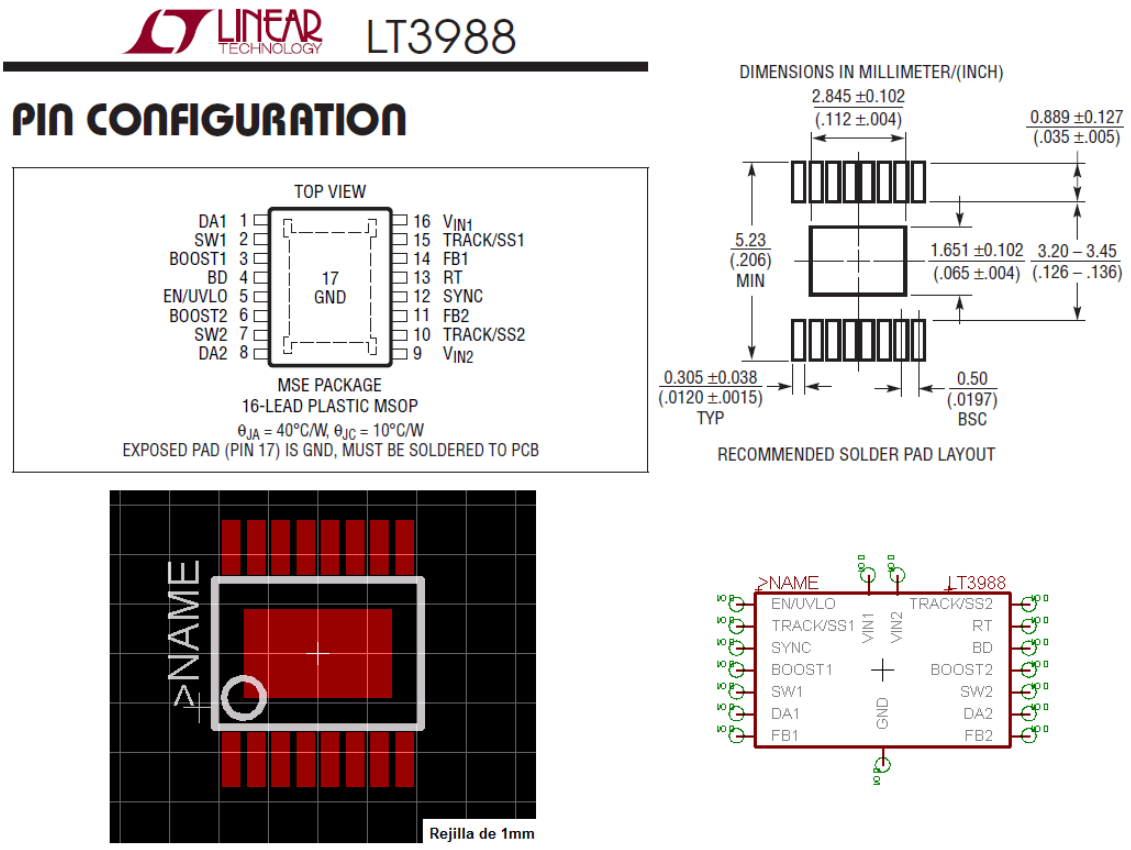


Figura 21: modelado de LT3988.

Para el diseño del circuito nos basaremos en el modelo de aplicación estándar proporcionado en las hojas características. A él, le añadiremos un condensador electrolítico a la entrada para evitar el efecto "avalancha de voltaje" que se genera cuando se conecta el circuito a través de cables largos y donde hay condensadores de característica dieléctrica X5R, X7R o similares (bibliografía **[2]**). Este efecto viene descrito en la nota de aplicación de LINEAR TECHNOLOGY. En el Anexo E se encuentran las ecuaciones que proporciona el fabricante para el dimensionamiento de componentes y las características de los mismos.

Dado que aprender a calcular esta clase de componentes conlleva una cierta dedicación suele ser conveniente crear una tabla en una hoja de cálculo para poder calcular nuevos diseño en el futuro con el mismo IC, evitando tener que comenzar a buscar de nuevo cómo calcular componentes.

En las Figuras 22, 23 y 24 se muestran las tablas empleadas para el cálculo (tanto para la salida de 3,3V como de 3,9V) donde se incluyen las ecuaciones empleadas, junto al diseño esquemático de referencia proporcionado por LINEAR TECHNOLOGY. Los cálculos proporcionados son conservadores y suponiendo siempre que se pretende usar hasta la máxima corriente de salida que puede proporcionar el IC (1 amperio por salida). Los valores en color rojo son la solución del diseño mientras que las celdas con fondo amarillo son los datos que aporta el diseñador. La Figura 25 muestra el diseño final, fiel a la recomendación del fabricante.

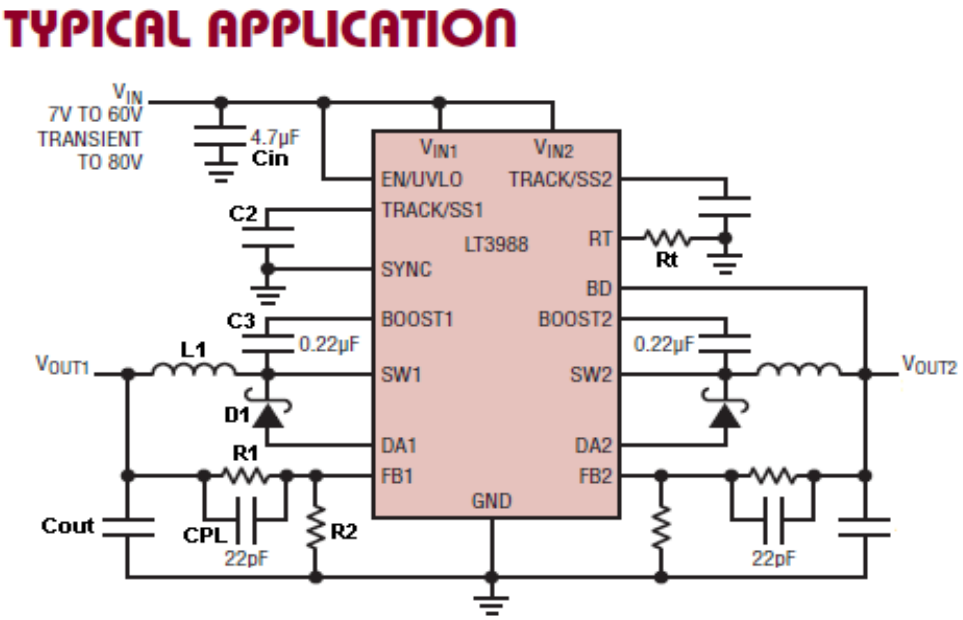


Figura 22: nomenclatura para cálculo de LT3988.

DATOS		CONDICIONES	RESULTADOS			FÓRMULAS
Vout (V) =	3,30	4VDC<=Vin<=60VDC	R1 (Ω) ≈	34000	=	C4*((C3/0,75)-1)
R2 (Ω) =	10000	1,8V<=Vout<=(Vin-2V)	R1//R2 (Ω) =	7727	=	(F3*C4)/(F3+C4)
Vin (V) <=	60	R1//R2<=20kΩ				
f (MHz) =	0,25	250kHz<=f<=2,5MHz	Rt (kΩ) ≈	200	=	(1,31/C6^2)+(46,56/C6)-7,322
		ILrms>1A, Ilsat>1,3A, Ldcr<0,1Ω	L1 (µH) ≈	15	=	0,6*(C3+0,4)/(0,6*C6)
			Cout (µF) >	16	=	10*F7*(1,1/C3)^2
L1 (µH) =	15		IL1r,pp (A) ≈	0,93	=	(1-((C3+0,4)/(C5+0,1)))*((C3+0,4)/(C9*C6))
SI Cout CERÁMICO		Dejar Vripple a 0 si no cerámico				
Vripple (V) ≈	0,02	X5R o X7R	Cout (µF) >=	23	=	F9/(C11*8*C6)
SI Cout ELECTROLÍTICO		Dejar ESR a 0 si no electrolítico				
ESR (Ω) =	0	Elegir C en función del Vripple deseado	Vripple (V) ≈	0	=	F9*C13
Cout (µF) =	47		C2 (µF) >=	0,0047	=	C14/10000

BILL OF MATERIALS		
REFERENCIA	VALOR	DETALLES
C3	0,22 µF	X5R ó X7R.
CPL	22 pF	Condensador de estabilidad del bucle de realimentación.
D1	Schottky	Referencia del fabricante MBRS190T3.
Cin	4,7 µF	X5R ó X7R si cerámico.
C2	10 nF	X5R ó X7R.
R1	34 kΩ	Resistencia calculada del divisor de tensión.
R2	10 kΩ	Resistencia fijada del divisor de tensión.
Rt	200 kΩ	Resistencia de configuración de frecuencia de conmutación.
Cout	22 µF	X5R ó X7R si cerámico.
L1	15 µH	Inductor de potencia, elegir uno de la serie WE-PD2-S de Würth.

Figura 23: Cálculo de la salida de 3,3V 1A de LT3988.

DATOS		CONDICIONES	RESULTADOS			FÓRMULAS
Vout (V) =	3,90	4VDC<=Vin<=60VDC	R1 (Ω) ≈	42000	=	C4*((C3/0,75)-1)
R2 (Ω) =	10000	1,8V<=Vout<=(Vin-2V)	R1//R2 (Ω) =	8077	=	(F3*C4)/(F3+C4)
Vin (V) <=	60	R1//R2<=20kΩ				
f (MHz) =	0,25	250kHz<=f<=2,5MHz	Rt (kΩ) ≈	200	=	(1,31/C6^2)+(46,56/C6)-7,322
		ILrms>1A, Ilsat>1,3A, Ldcr<0,1Ω	L1 (µH) ≈	17	=	0,6*(C3+0,4)/(0,6*C6)
			Cout (µF) >	14	=	10*F7*(1,1/C3)^2
L1 (µH) =	15		IL1r,pp (A) ≈	1,06	=	(1-((C3+0,4)/(C5+0,1)))*((C3+0,4)/(C9*C6))
SI Cout CERÁMICO		Dejar Vripple a 0 si no cerámico				
Vripple (V) ≈	0,02	X5R o X7R	Cout (µF) >=	27	=	F9/(C11*8*C6)
SI Cout ELECTROLÍTICO		Dejar ESR a 0 si no electrolítico				
ESR (Ω) =	0	Elegir C en función del Vripple deseado	Vripple (V) ≈	0	=	F9*C13
Cout (µF) =	47		C2 (µF) >=	0,0047	=	C14/10000

BILL OF MATERIALS		
REFERENCIA	VALOR	DETALLES
C3	0,22 µF	X5R ó X7R.
CPL	22 pF	Condensador de estabilidad del bucle de realimentación.
D1	Schottky	Referencia del fabricante MBRS190T3.
Cin	4,7 µF	X5R ó X7R si cerámico.
C2	10 nF	X5R ó X7R.
R1	42 kΩ	Resistencia calculada del divisor de tensión.
R2	10 kΩ	Resistencia fijada del divisor de tensión.
Rt	200 kΩ	Resistencia de configuración de frecuencia de conmutación.
Cout	22 µF	X5R ó X7R si cerámico.
L1	18 µH	Inductor de potencia, elegir uno de la serie WE-PD2-S de Würth.

Figura 24: cálculo de la salida de 3,9V 1A de LT3988.

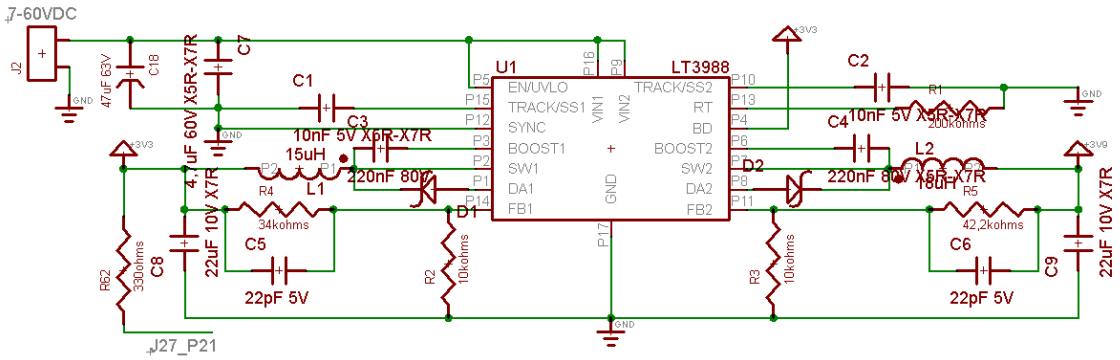


Figura 25: esquema de la fuente de alimentación de SEL16.

3.4.8 – CONEXIÓN CON FLX01

Dado que se va a emplear una PCB flexible conectada a SEL16 es necesario definir dicha conexión. El número total de puntos de conexión es 25 y se necesita un conector compacto que pueda ser usado tanto en SEL16 como en FLX01 (teniendo en cuenta las limitaciones de las PCBs flexibles).

El conector J38 mostrado en la figura del diseño esquemático corresponde a un conector de tipo macho para conectar un LED de indicación de alimentación adicional, por si el producto con el que se ha configurado SEL16 no necesita a FLX01. El conector empleado para FLX01 es el mostrado en la Figura 26 y el diseño esquemático de esta conexión correspondiente a los LEDs con resistencias limitadoras de intensidad en las resistencias se muestra en la Figura 27.

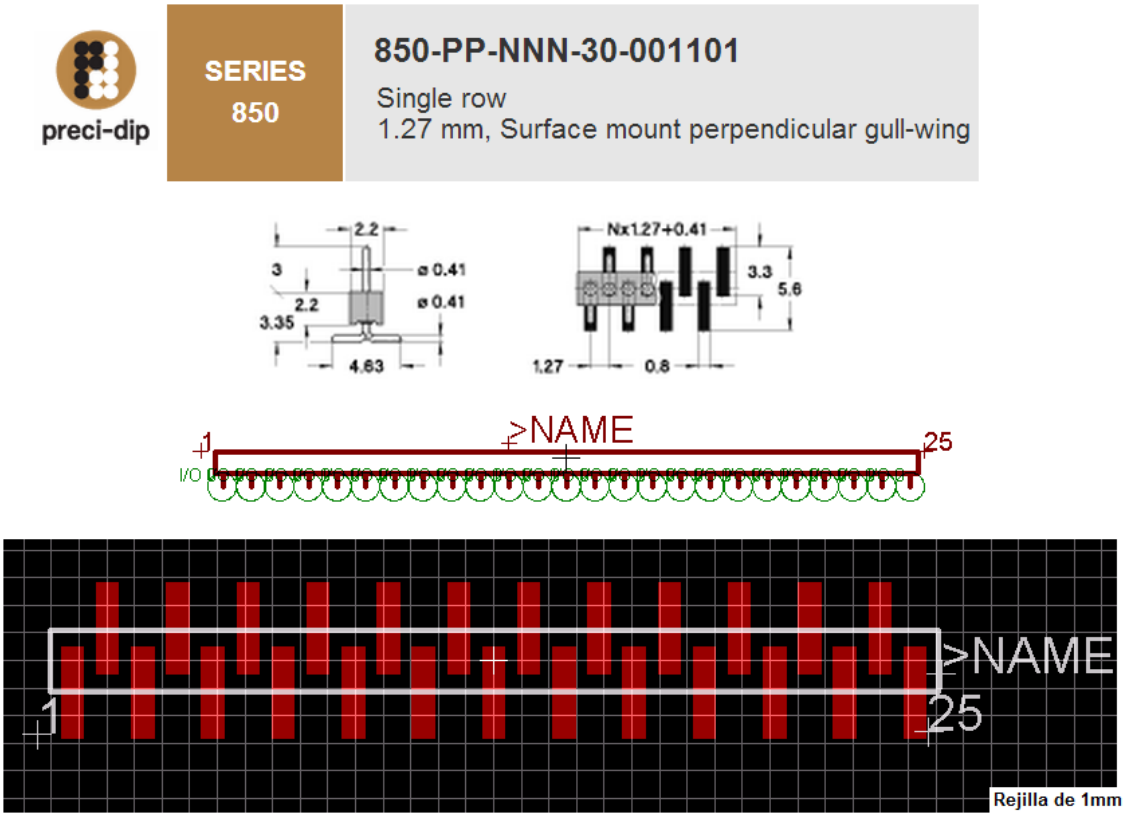


Figura 26: modelado de conector PRECIDIP tipo 850.

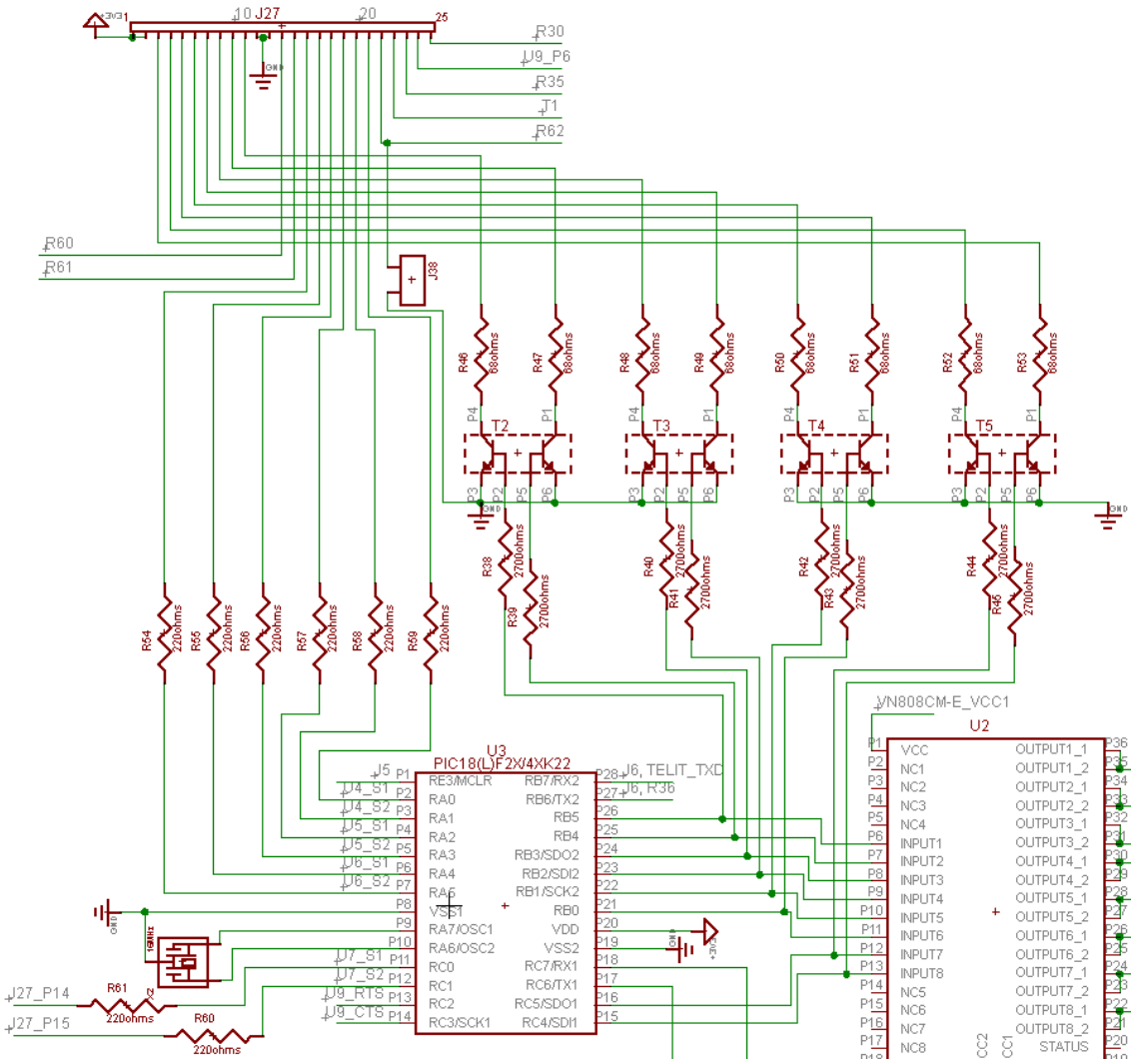


Figura 27: Esquema de la conexión de los LEDs de SEL16.

3.4.9 – DISEÑO ESQUEMÁTICO FINAL DE SEL16

Teniendo en cuenta todas las consideraciones previas analizadas en este capítulo, el diseño esquemático de SEL16 queda perfectamente definido. En el Anexo C se encuentra el diseño esquemático completo.

3.4.10 – LISTA DE MATERIALES DE SEL16

Una vez concluido el diseño esquemático ya se dispone de la preceptiva lista de materiales a facilitar al fabricante de la PCB. En el Anexo A se encuentra la lista donde se detallan la referencia del fabricante y la referencia en el diseño esquemático de SEL16.

3.5 – DISEÑO, SELECCIÓN Y MODELADO DEL HARDWARE DE FLX01

Una vez finalizado el diseño esquemático de SEL16 hay que proceder con el diseño esquemático de la PCB flexible que funcionará conectada a SEL16: FLX01. Aunque la técnica de fabricación de esta PCB es diferente de las PCBs sólidas su diseño sigue el mismo procedimiento. Sin embargo el diseño de FLX01 es bastante más sencillo que el de SEL16 pues sólo consta de 19 LEDs y un conector. La dificultad del diseño de esta PCB radica en el diseño de su circuito impreso ya que los LEDs y la propia PCB deben encajar en la carcasa.

Los LEDs empleados son los indicados en la Figura 28, de formato 0603 (código estándar para desginar en pulgadas el largo y el ancho de componentes con dos conexiones en formato SMT) y el conector es el complementario del usado en SEL16 para conectar con FLX01, mostrado en la Figura 29.

Kingbright • 0603 1.6x0.8mm > KPG-1608 **SMD LED**

Part No.	Material	λD (nm)	Lens Type	Iv(mcd) @20mA Min.	Iv(mcd) @20mA Typ.	Viewing Angle 2θ1/2
KPG-1608SURKC-T	AlGaInP	630	water clear	55	110	120º
KPG-1608SEKC-T	AlGaInP	601	water clear	55	100	120º
KPG-1608SYKC-T	AlGaInP	590	water clear	55	120	120°
KPG-1608CGKC-T	AlGaInP	570	water clear	20	50	120°
KPG-1608ZGC	InGaN	525	water clear	200	400	120°
KPG-1608QBC-D	InGaN	465	water clear	40	100	120°
KPG-1608VBC-D	InGaN	470	water clear	120	200	120º

Figura 28: LEDs de formato 0603 seleccionados para FLX01.

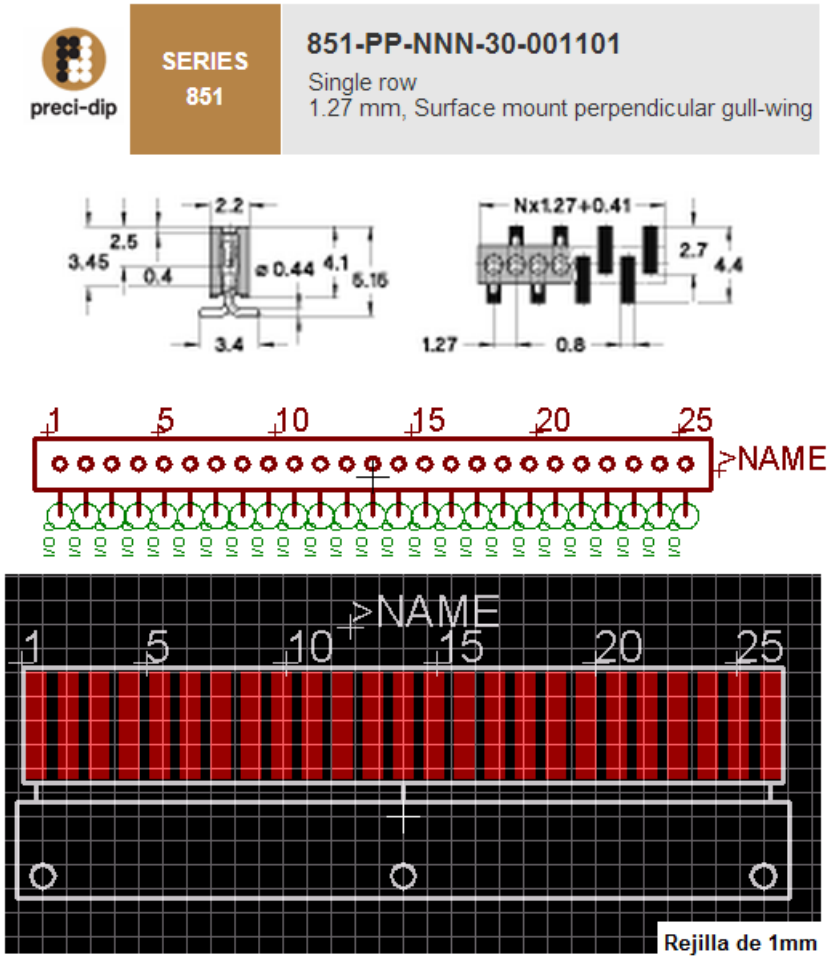


Figura 29: modelado de conector PRECIDIP tipo 251.

3.5.1 – DISEÑO ESQUEMÁTICO FINAL DE FLX01

El diseño esquemático de FLX01 se muestra en la Figura 30. Hay que tener especial cuidado a la hora de conectar los LEDs al conector, de forma que haya una correcta correspondencia entre los conectores que unen SEL16 y FLX01.

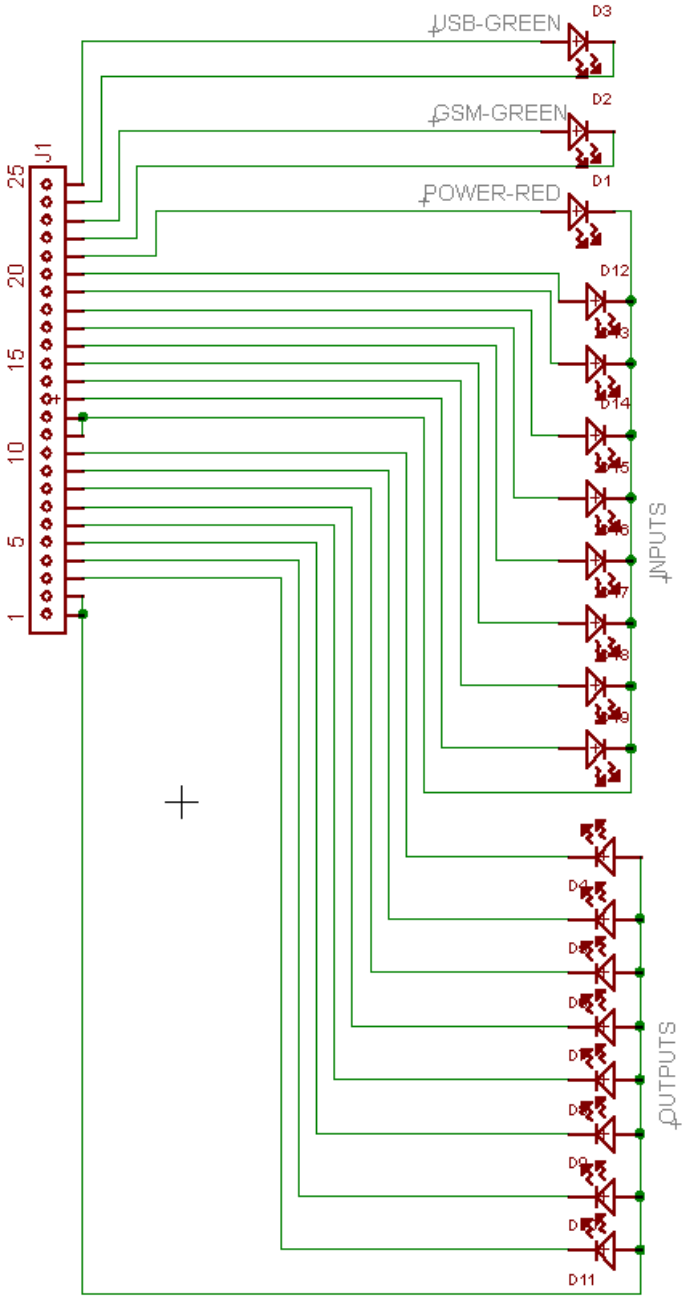


Figura 30: esquema de FLX01.

3.5.2 – LISTA DE MATERIALES DE FLX01

En Anexo B se encuentra la lista de materiales de FLX01 junto a sus costes. Esta placa es sencilla y su lista de materiales obedece a ello siendo escueta.

CAPÍTULO 4 – DISEÑO DE LOS CIRCUITOS IMPRESOS

4.1 – CIRCUITO IMPRESO DE SEL16

Una vez finalizados el diseño esquemático y la lista de materiales, el siguiente paso es el diseño de la PCB. Para el diseño de la PCB se tendrán en cuenta ciertas normas (recogidas en las reglas de diseño) para respetar unas separaciones mínimas adecuadas entre distintos tipos de señales y partes mecanizadas de la PCB.

Aunque no se recojan en las reglas de diseño de EAGLE, se respetarán las reglas generales para el diseño de PCBs que se resumen en no trazar pistas muy cerca de los extremos de la PCB ni hacer giros con las mismas de más de 45º dado que las esquinas propician que esa zona actúe como un condensador parásito. Así mismo se minimizará la longitud de las pistas y se maximizará el ancho de las mismas siempre que los pads, vías o drills que conectan con la correspondiente pista lo permitan y naturalmente sin interferir en el trazado de otras pistas. Esta última regla obedece a la ecuación de la resistividad de un material, donde se trata de minimizar la resistencia de la pista.

4.1.1 – REGLAS DE DISEÑO PARA DISEÑAR EL CIRCUITO IMPRESO DE SEL16

La base de la PCB será de FR-4 (Flame Retardant 4), un material bastante común para este propósito. Las características preliminares de esta resina epoxy de fibra de vidrio reforzada se muestran en el Anexo J. El circuito eléctrico constará de cuatro capas conductoras de cobre de 0,035 milímetros de espesor, divididas por 3 capas aislantes de ensamblaje de 0,2, 0,63 y 0,63 milímetros respectivamente, como se muestra en la Figura 32. Se escogen cuatro capas conductoras dada la cantidad de elementos a conectar y la superficie de la placa, definida de acuerdo a la carcasa.

La razón de que la capa aislante que separa la primera y la segunda capa conductora sea de 0,2 milímetros obedece a respetar la impedancia diferencial entre las líneas de transmisión y recepción USB exigida para cumplir con la normativa de USB 2.0 High Speed (bibliografía **[1]**). No se usarán vías enterradas ni vías ciegas (incrementan notablemente el precio de fabricación y no son necesarias en este diseño).

La separación mínima de una señal con respecto a otra señal, a taladros o a bordes de la PCB se muestran en la Figura 31. Por la tecnología empleada en la fabricación no se recomiendan pistas ni aislamientos con una anchura inferior a 0,2 milímetros aunque 0,1 milímetros sea el límite absoluto. En todas las Figuras de los sub-apartados de este punto se mostrará la rejilla de un milímetro (salvo que se especifique otra distancia) con el fin de proveer una referencia de distancia con la que evaluar el diseño visualmente.

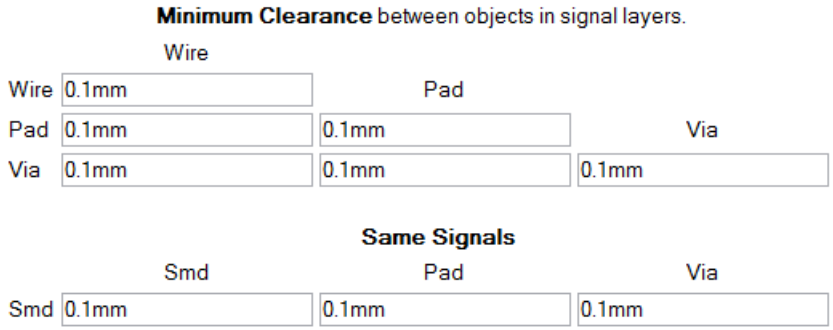


Figura 31: separación mínima entre pistas en SEL16

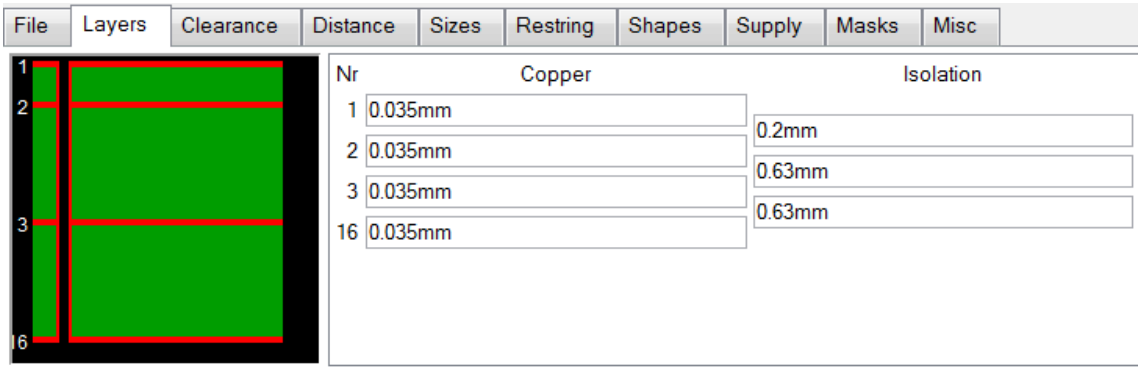


Figura 32: distribución de capas en SEL16.

Para la capa conductora TOP se ha escogido el color rojo y su plano de recubrimiento está conectado a la señal de 0V del circuito, la segunda capa conductora (debajo de la capa TOP) es de color violeta y su plano de recubrimiento también está conectado a la señal de 0V (estos dos requisitos son necesarios tanto para TELIT como para USB). La tercera capa, en color gris, tiene su plano de recubrimiento conectado a la señal de 3,3V y la última capa (la capa BOTTOM) es de color azul y tiene su plano de recubrimiento conectado a la señal de 0V.

4.1.2 – DISEÑO DEL CIRCUITO IMPRESO DE SEL16

Una vez definidas las reglas de diseño se procede a diseñar la PCB. Sin embargo hay que tomar un orden adecuado dado que algunos componentes necesitan una conexión más adecuada que una simple pista que pueda ser larga, sinuosa y con una o varias vías.

4.1.2.1 – UBICACIÓN DE CONECTORES SEGÚN DISTRIBUCIÓN DESEADA PARA LA CARCASA

La primera tarea siempre debe ser crear la dimensión de la PCB (que se ha optimizado añadiendo unas muescas en los laterales ya que mejora la inserción de la PCB en la carcasa) y situar los elementos que ya tienen asignado un emplazamiento debido a su utilidad y/o características geométricas. En el caso de SEL16 estos elementos sólo son los conectores, dado que el resto de elementos son de libre ubicación siempre y cuando se respeten las reglas para su correspondiente conexión con otros elementos.

Siguiendo las medidas de la carcasa, se sitúan los conectores de acuerdo con la Figura 33 (J47 que es un conector como J2 está sobre J3 que es USB para elegir uno u otro dependiendo de la aplicación, lo cual está expuesto en el apartado 7.1.1). Todos los conectores sobresalen en mayor o menor medida según necesidad, ya que deben ser accesibles desde el exterior de la carcasa y ofrecer espacio y comodidad para ser usados.

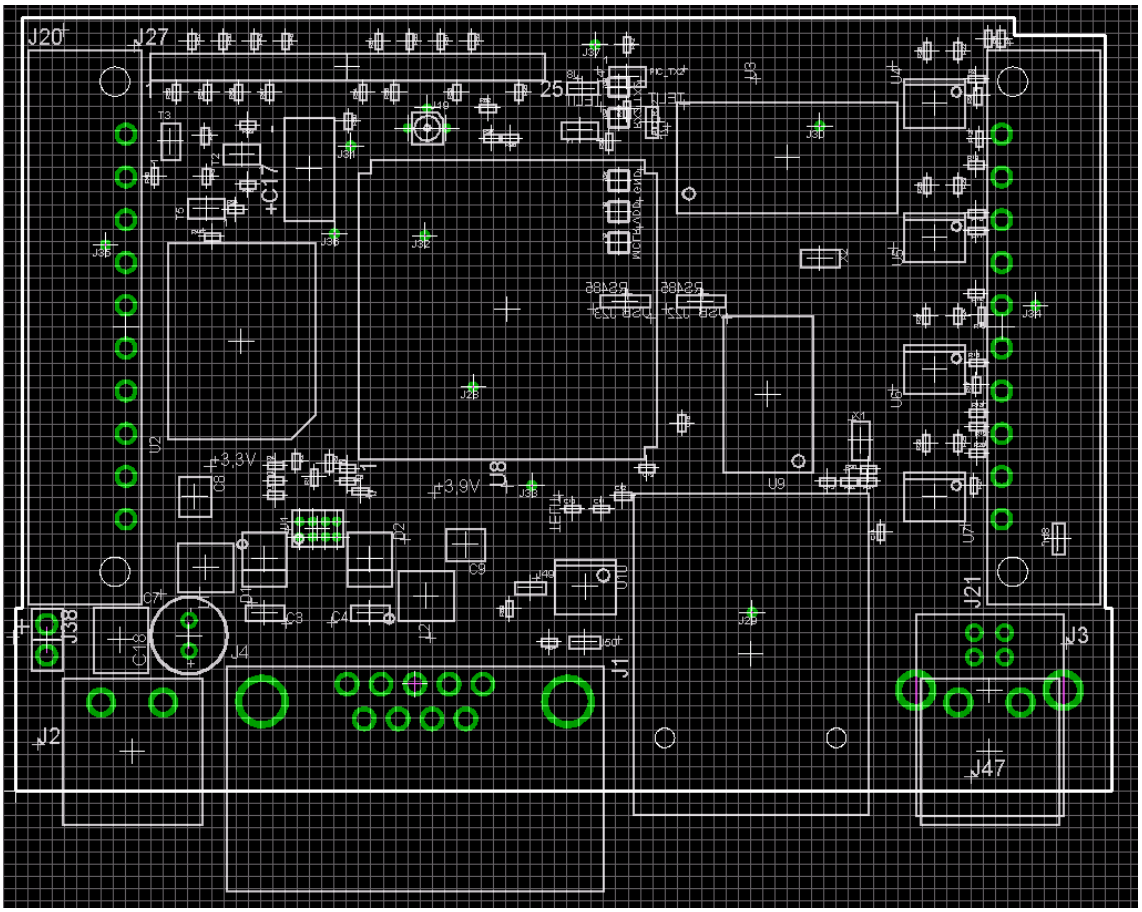


Figura 33: distribución de componentes en SEL16.

4.1.2.2 – UBICACIÓN DE CIRCUITOS PRIORITARIOS

A continuación se distribuirán en la PCB los circuitos que tienen necesidades especiales de conexionado con otros elementos. En la figura anterior se puede ver la distribución de los componentes de la PCB.

4.1.2.2.1 – CIRCUITO DE ALIMENTACIÓN ELÉCTRICA

Todas las fuentes de alimentación conmutadas son sensibles a una mala distribución y conexión de los componentes electrónicos que la conforman. Usar pistas estrechas y/o muy largas para conectar los distintos elementos son errores que se pagan con una mala regulación y la generación de ruido electrónico. Aparte de los inconvenientes electrónicos del ruido radiado, éste excita a la bobina a frecuencias audibles, la cual puede generar el típico pitido que se oye en este tipo de circuitos cuando surgen esa clase de problemas.

En el caso de usar un IC encargado de la conmutación y la regulación (como en SEL16) es habitual que el fabricante aporte, bien en las hojas de características del IC o bien en alguna nota de aplicación, la forma de situar los elementos para mejorar el funcionamiento del circuito. En nuestro caso LINEAR TECHNOLOGY proporciona el siguiente esquema ejemplo (Figura 34) en las hojas de características del IC LM3988IMSE.

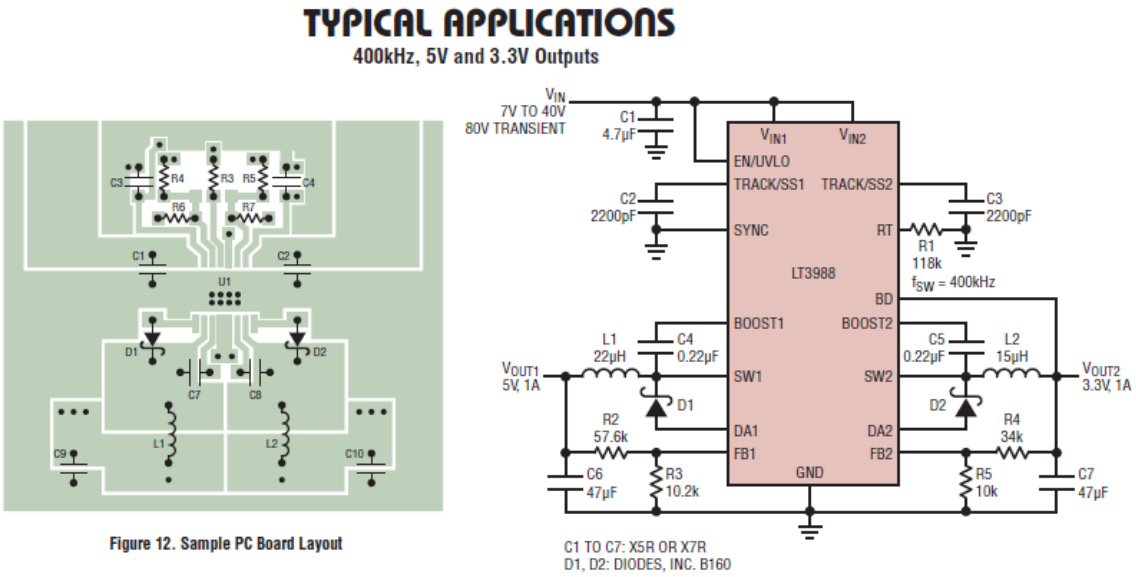


Figura 34: sugerencia de conexión de LT3988.

Siguiendo estas directrices para distribuir componentes y trazar sus respectivas conexiones en la PCB se ha llegado a la distribución mostrada en la Figura 35 (sin los planos de recubrimiento para facilitar la inspección visual).

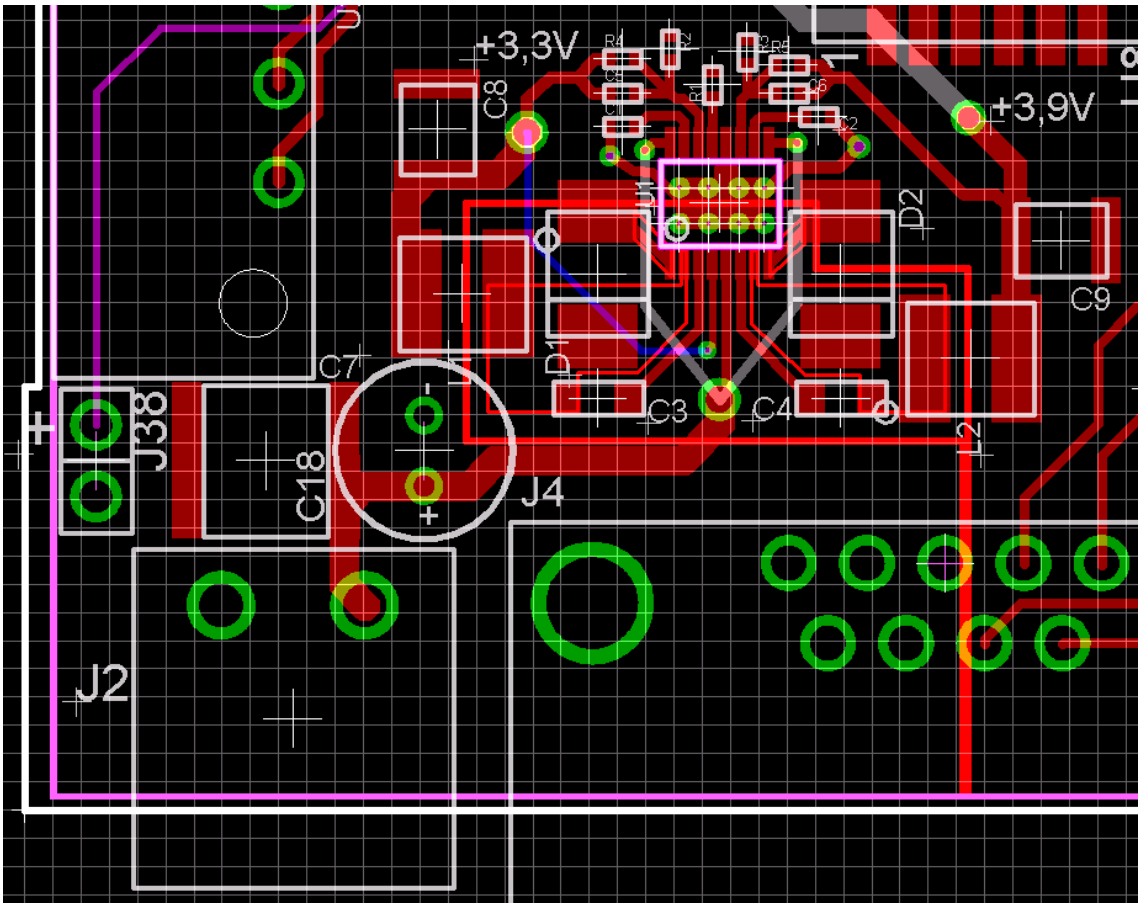


Figura 35: circuito de alimentación de SEL16.

4.1.2.2.2 – CIRCUITO USB

El estándar USB da las directrices para esta tarea. Las pistas pueden trazarse sin mayores preocupaciones si no se pretenden alcanzar velocidades de transmisión iguales o superiores a USB Full Speed pero SEL16 está preparada hasta High Speed luego si queremos que este tipo de comunicación sea posible hay que seguir de manera estricta las directrices de la normativa. En la Figura 36 se muestra en detalle las pistas correspondientes a datos en el circuito USB (sólo se muestra la primera capa conductora), con una rejilla de 0,1 milímetros y con el plano de recubrimiento.

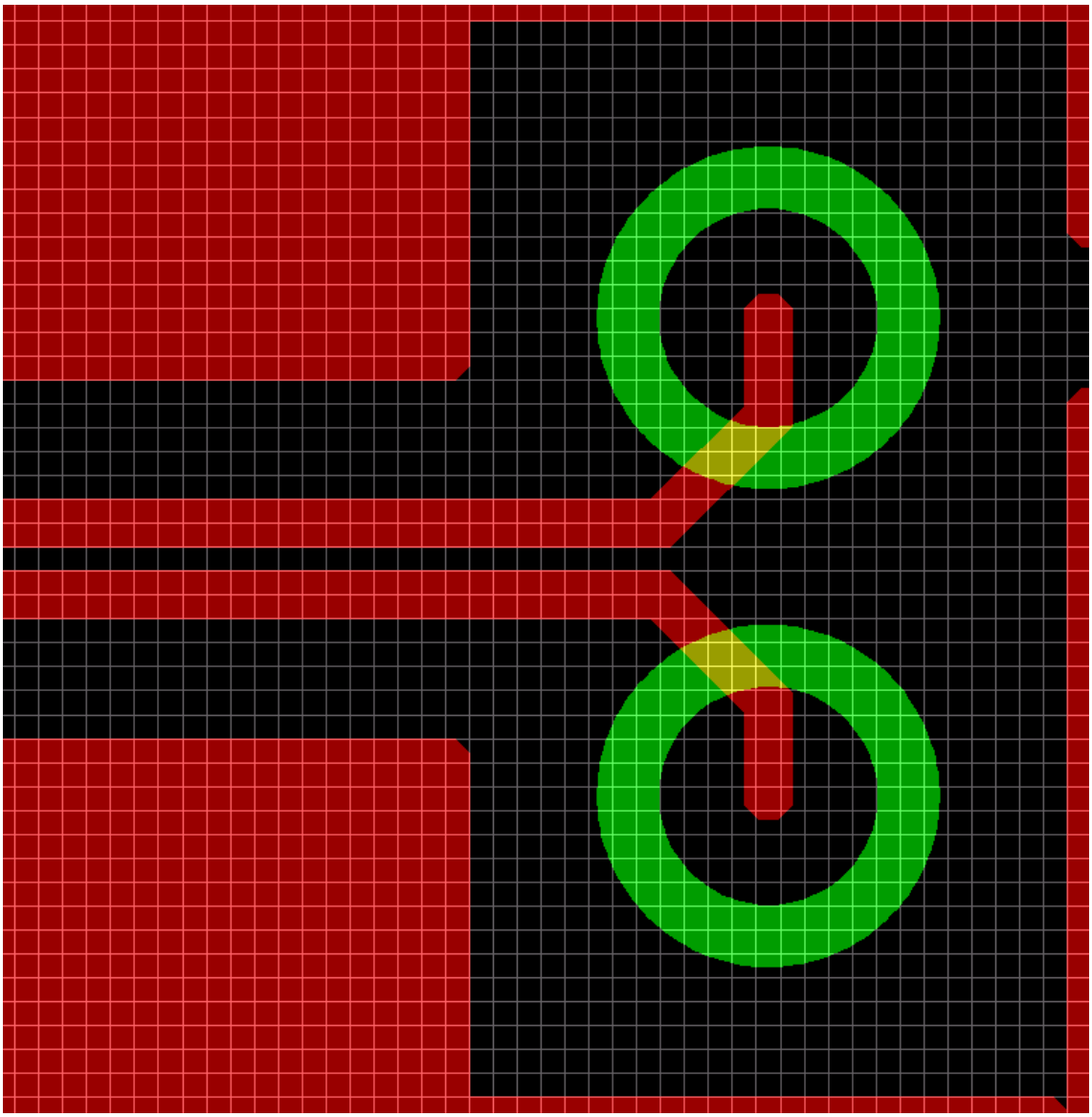


Figura 36: detalle de pistas USB de SEL16.

4.1.2.2.3 – OTROS CIRCUITOS PRIORITARIOS

Aunque el resto de componentes de la PCB pueden ser situados "libremente" sigue habiendo componentes que guardan una relación especial entre ellos y deben estar muy cerca físicamente unos de otros. Es el caso de la antena del IC TELIT GL865 QUAD, ya que aunque TELIT en las hojas características de dicho IC da directrices para trazar las pistas de la antena, especifica que cuando más cortas sean mejor así que lo adecuado es que el conector de la antena esté directamente al lado de su pad correspondiente.

La Figura 37 muestra el circuito correspondiente al sistema de comunicación con la red de telefonía móvil, donde se aprecia la conexión de la antena (evitando ángulos rectos) y que se ha respetado la zona libre de conductor en la capa exterior según se especifica en las hojas de características de TELIT.

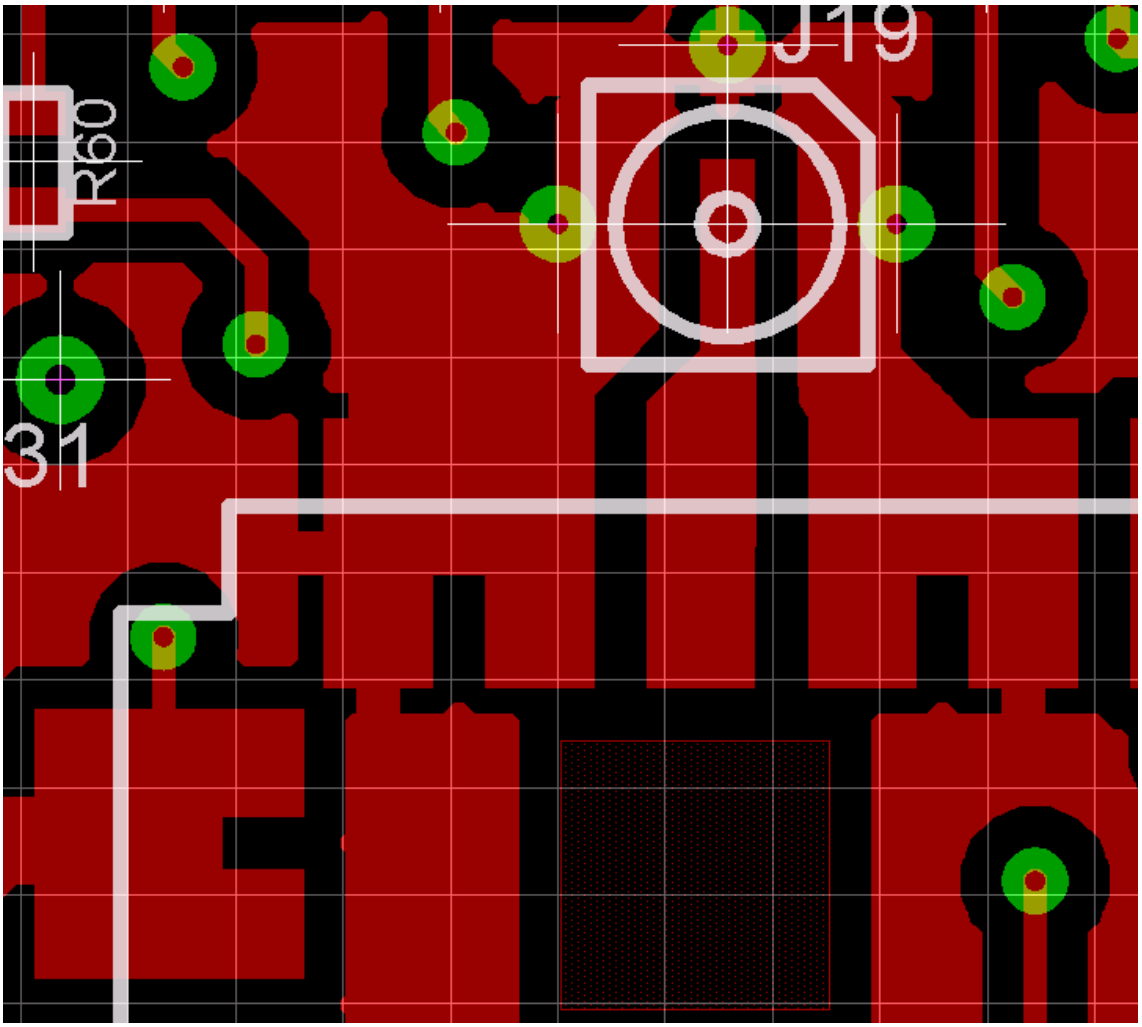


Figura 37: conexión del conector de antena de TELIT GL-865 QUAD.

Otros componentes sensibles son los osciladores, que deben estar lo más cerca posible de sus respectivos ICs ya que en caso contrario radiarán ruido electrónico, alterando el circuito y afectando a su eficacia como señales de reloj.

4.1.3 – DISEÑO FINAL DEL CIRCUITO IMPRESO DE SEL16

Una vez situados todos los componentes en la PCB y correspondientemente conectados, la PCB queda tal como se muestra en las Figuras 38, 39, 40 y 41. Los pasos finales para saber que todo está correcto es, primeramente, pulsar el botón RATSNEST para que se solidifiquen los planos conductores puestos y comprobar si están realizadas todas las conexiones. Si ya no quedan conexiones que realizar hay que usar el botón CHECK dentro del cuadro de diálogo de las reglas de diseño de la PCB. De este modo EAGLE hará una comprobación de si se respetan las reglas de diseño en el circuito y notificará los errores existentes para que sean corregidos (o aprobados por el diseñador ya que la interpretación de los errores queda siempre a juicio del diseñador).

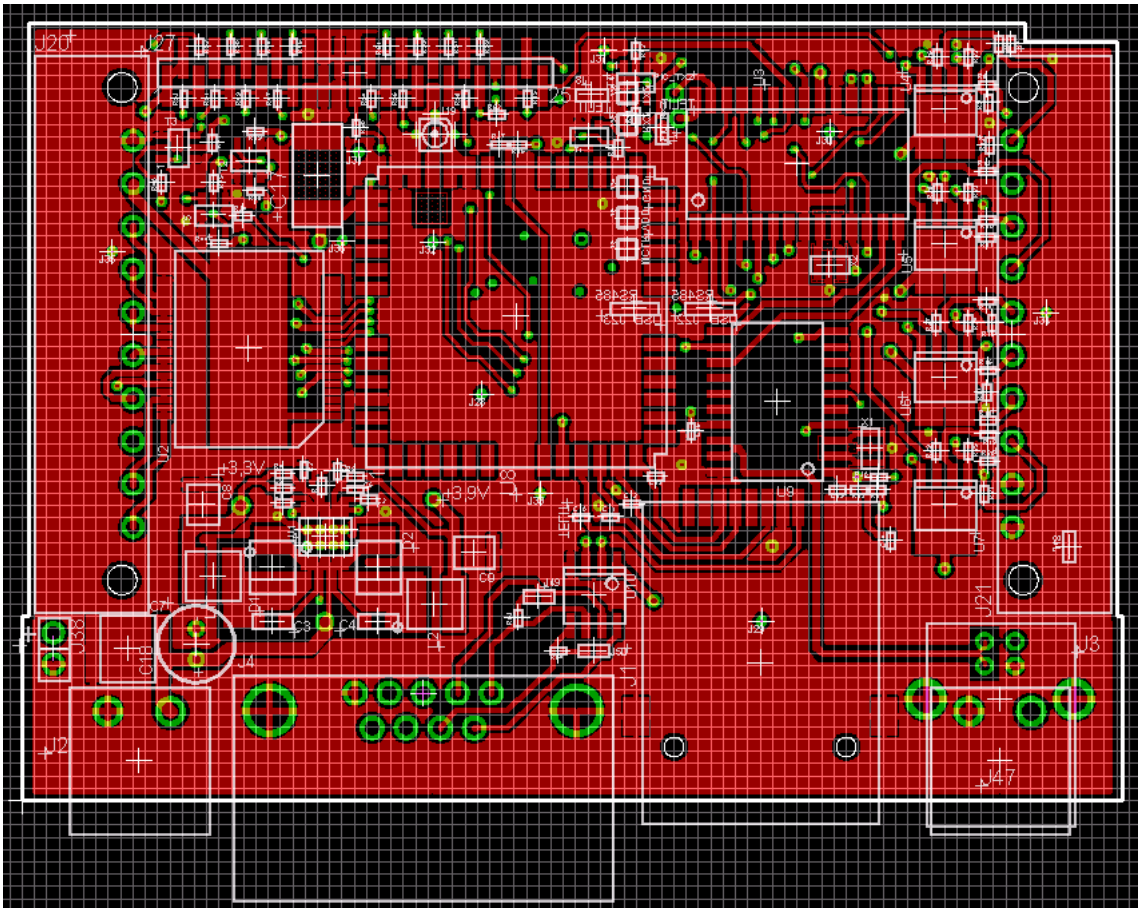


Figura 38: primera capa conductora de la PCB de SEL16.

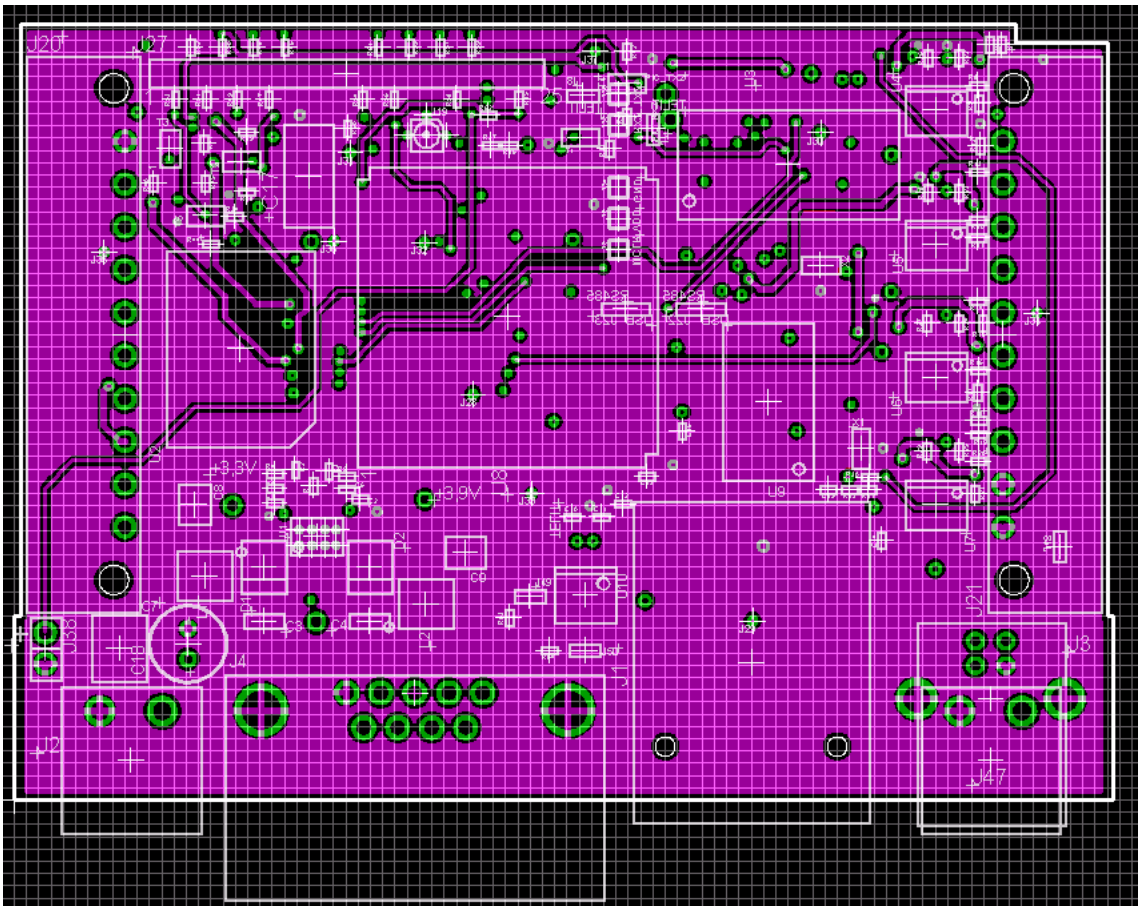


Figura 39: segunda capa conductora de la PCB de SEL16.

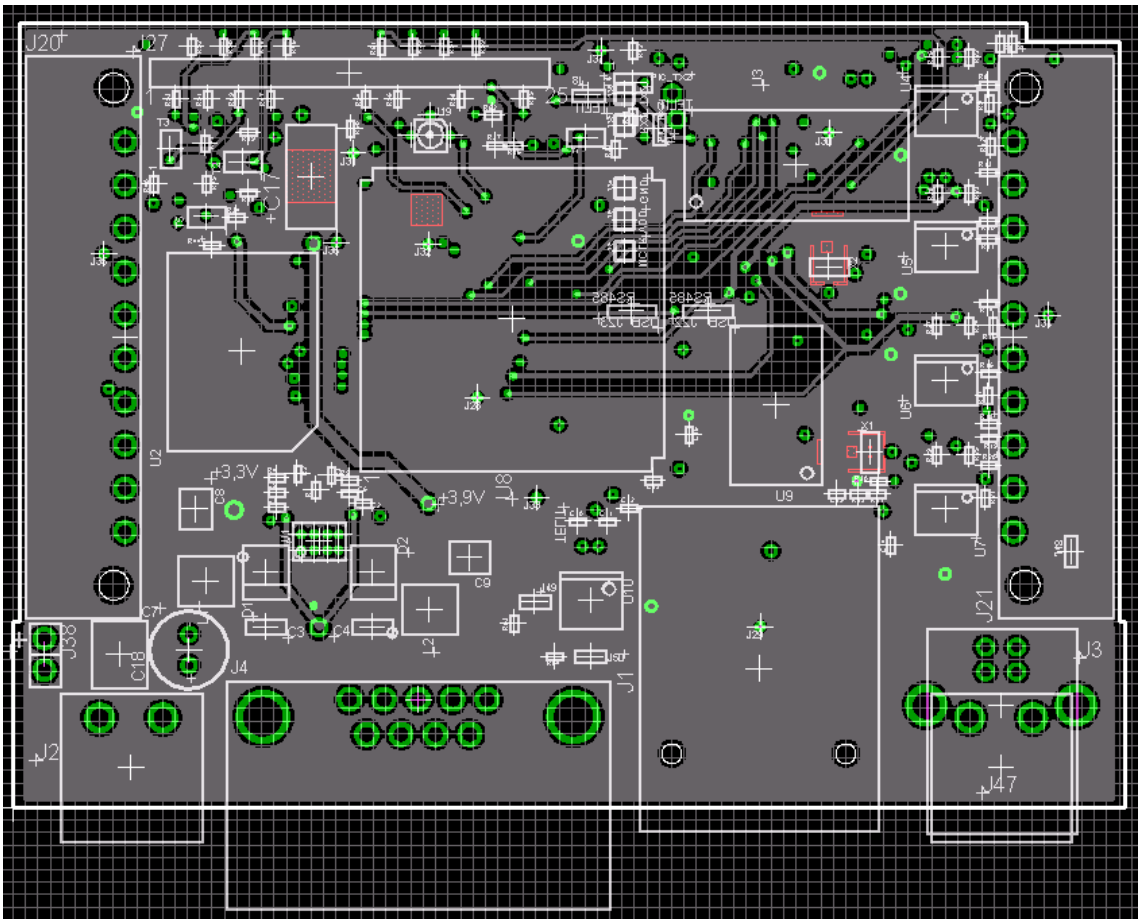


Figura 40: tercera capa conductora de la PCB de SEL16.

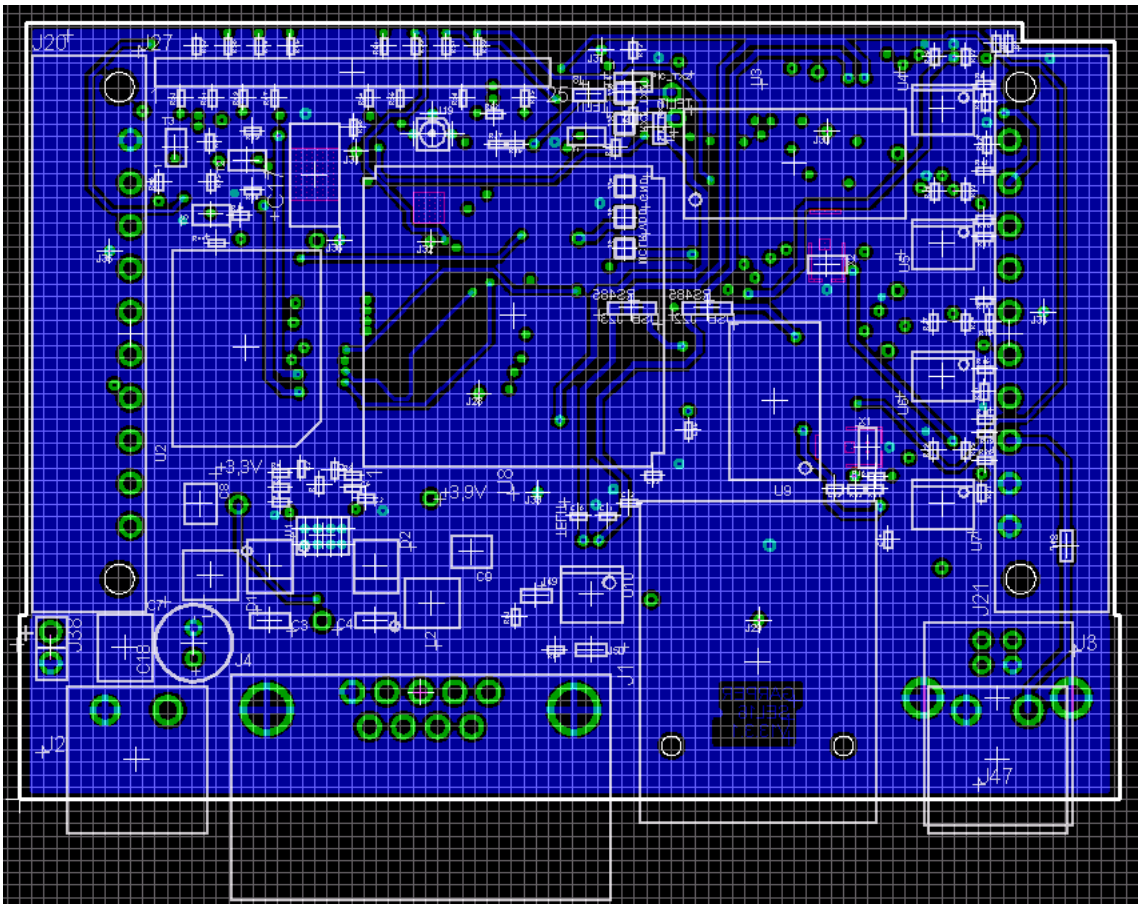


Figura 41: cuarta capa conductora de la PCB de SEL16.

4.1.4 – GENERACIÓN DE ARCHIVOS DE FABRICACIÓN DE SEL16 EN FORMATO GERBER

Para la fabricación de la PCB será necesario crear los archivos en formato GERBER que incluyen las capas de serigrafía, conductoras, aislantes y posición de taladros y componentes. Este formato es estándar para los sistemas de fabricación de PCBs y permite automatizar la mayoría del proceso de fabricación (con la excepción de ciertos componentes electrónicos que no son aptos para el posicionamiento y soldadura automatizados).

En la Figura 42 se muestra el perfil del CAM PROCESSOR creado para PCBs con cuatro capas conductoras y serigrafía y componentes sólo en el lado TOP de la PCB, es decir, el perfil adecuado para SEL16 (en cada pestaña se aprecia la capa a realizar, donde están incluidas las SOLDER STOP MASK para las capas exteriores). El procedimiento para obtener los ficheros GERBER consta de dos pasos:

1 – Ejecutar el comando DRILLCFG en la línea de comandos: se crea información sobre los taladros de la PCB.

2 – Ejecutar en el CAM PROCESSOR un perfil previamente creado por el diseñador que defina todos los aspectos de la PCB desde el punto de vista de la fabricación.

Una vez finalizado el proceso se obtienen los archivos necesarios para la fabricación, que se encontrarán en la carpeta del proyecto.

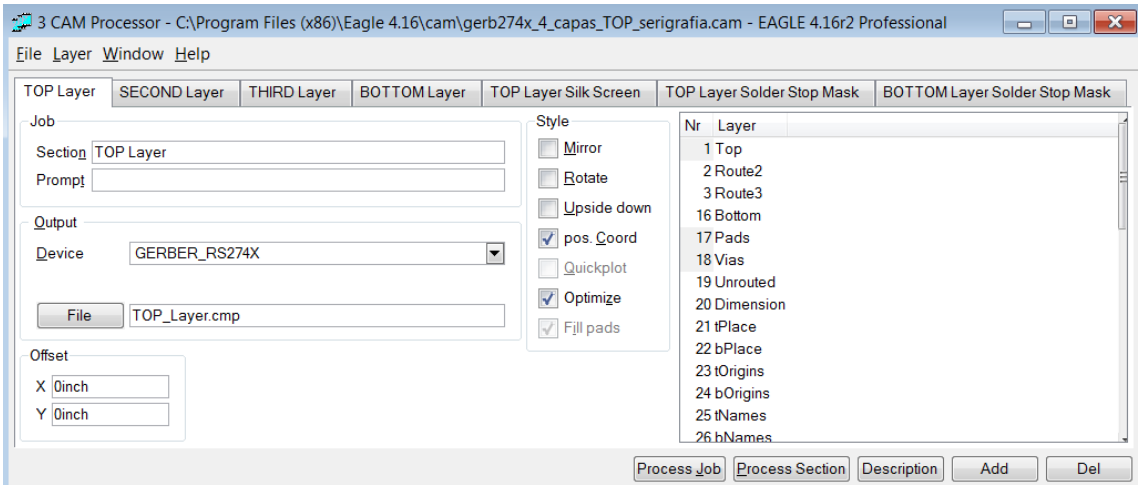


Figura 42: configuración del procesador CAM para SEL16.

4.2 – CIRCUITO IMPRESO DE FLX01

Ahora se va a proceder a diseñar FLX01, la PCB flexible que lleva los LEDs indicadores de IO5-USB.

4.2.1 – REGLAS DE DISEÑO PARA DISEÑAR EL CIRCUITO IMPRESO DE FLX01

Para las reglas de diseño se pueden emplear las mismas que para SEL16, teniendo en cuenta que FLX01 no tiene taladros ni vías luego sólo hay que cuidar la distancia entre distintas pistas y la distancia de éstas a los bordes de la PCB, respetando también una anchura adecuada. Sólo se empleará una capa conductora, la capa TOP que será de color rojo y sin plano de recubrimiento (no es necesario para esta clase de circuitos).

4.2.2 – DISEÑO DEL CIRCUITO IMPRESO DE FLX01

Para diseñar esta PCB hay que tener en cuenta la posición exacta de cada LED en la parte frontal de la carcasa y la correspondencia entre el conector de FLX01 y su homólogo en SEL16. Dado que la PCB consta de una sola capa hay que tener especial cuidado situando las pistas, de forma que se evite encerrar pistas dentro del trazado de otras o entre una pista y el borde de la PCB.

4.2.2.1 – UBICACIÓN DE LEDs

Los LEDs deben coincidir con la carcasa luego su ubicación está definida. Siguiendo las acotaciones de la carcasa, la distribución de los LEDs en la PCB es la que se muestra en la Figura 43, con una rejilla de 1 milímetro de distancia. En la figura también se aprecia el “cuello de botella” que se ha diseñado para añadir flexibilidad a la parte superior de la PCB flexible, para facilitar su inserción en la carcasa y se ha añadido una versión en miniatura del tamaño final de la PCB (se han añadido líneas discontinuas representando un acortamiento en la longitud de la PCB) para hacer contraste con el detalle de la posición de los LEDs.

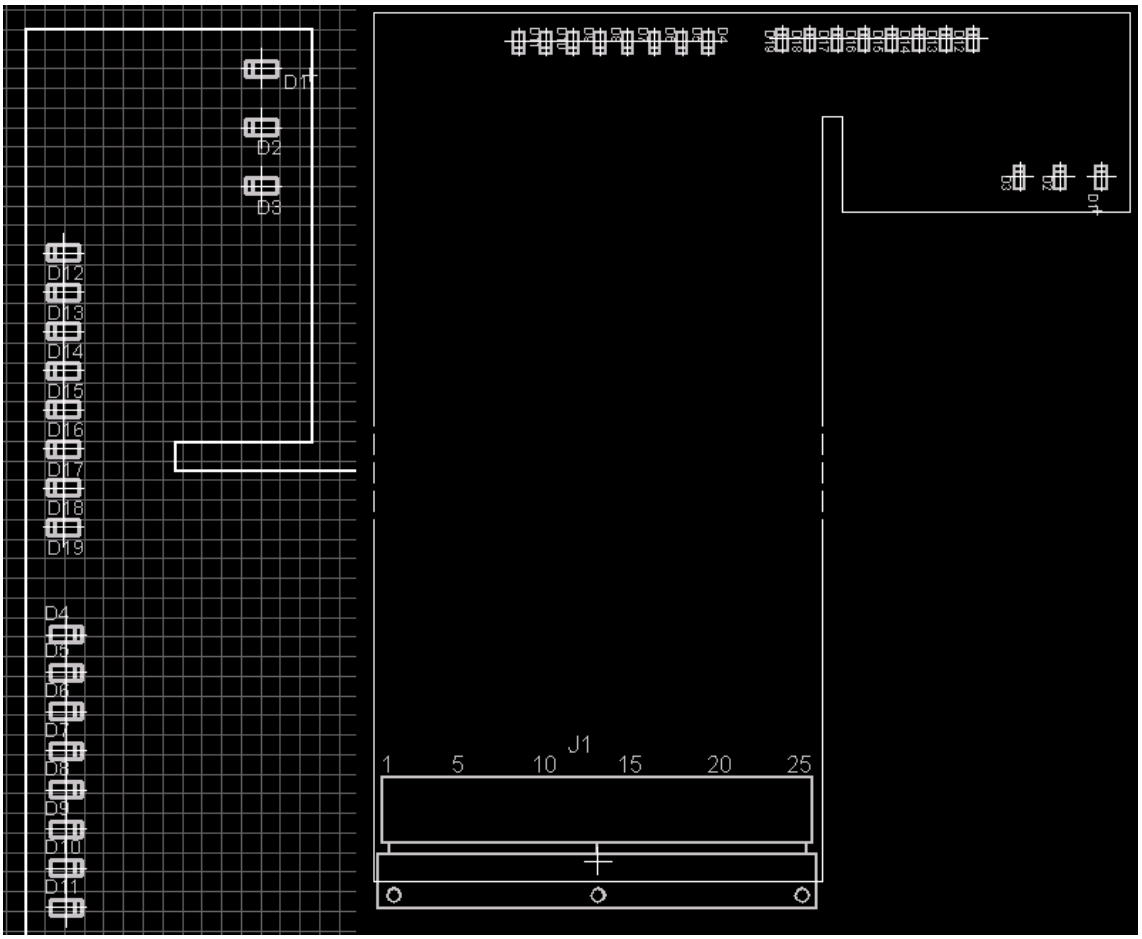


Figura 43: distribución de elementos en FLX01.

4.2.3 – DISEÑO FINAL DE FLX01

La PCB queda como se presenta en la Figura 44. Las dimensiones corresponden a las mediciones hechas para que la PCB flexible coincida con la parte frontal de la carcasa de manera adecuada. Se ha cuidado que la conexión entre el conector de FLX01 y su correspondiente en SEL16 coincidan adecuadamente.

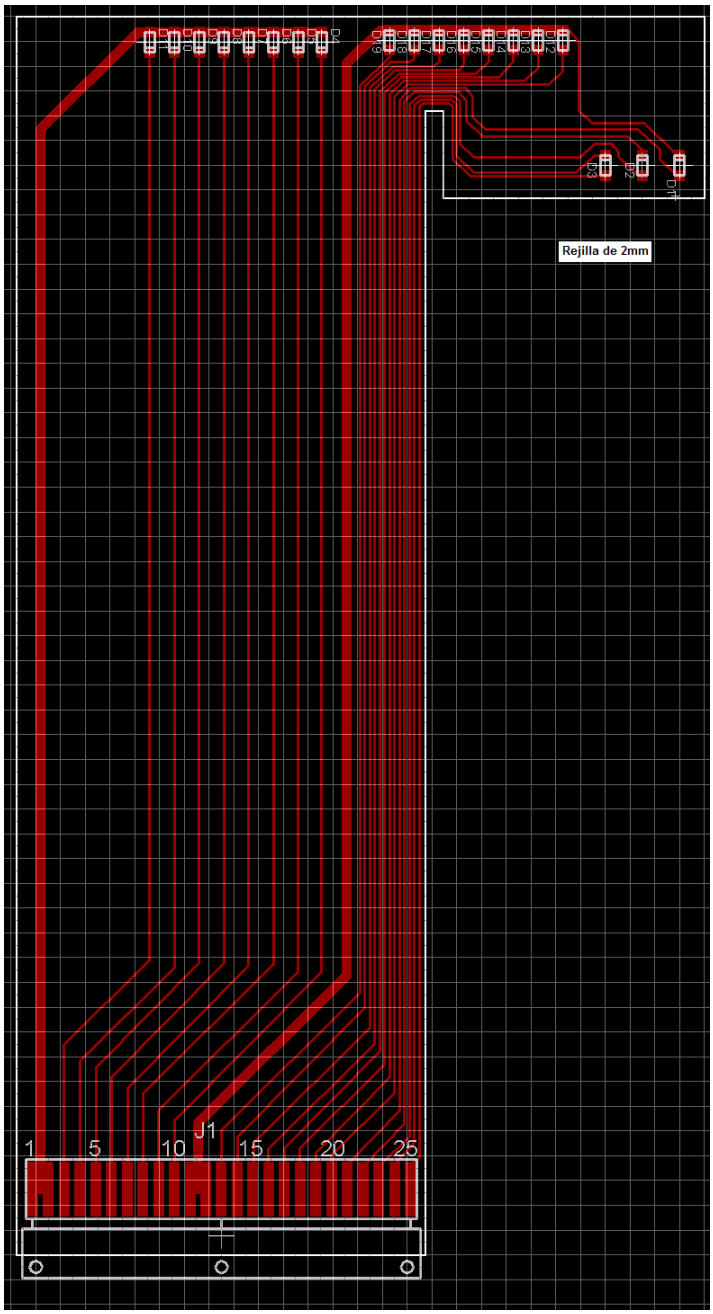


Figura 44: capa conductora en PCB de FLX01.

4.2.4 – GENERACIÓN DE ARCHIVOS DE FABRICACIÓN DE SEL16 EN FORMATO GERBER

Dado que la PCB sólo tiene una capa y carece de taladros y vías se puede emplear directamente en el CAM PROCESSOR un perfil que sólo genere los datos de serigrafía, capa conductora TOP y máscara de fin de soldadura TOP. En la Figura 45 se muestra el perfil creado para esta tarea (en las pestañas se puede comprobar que sólo se ejecutan las tareas propias de la capa TOP).

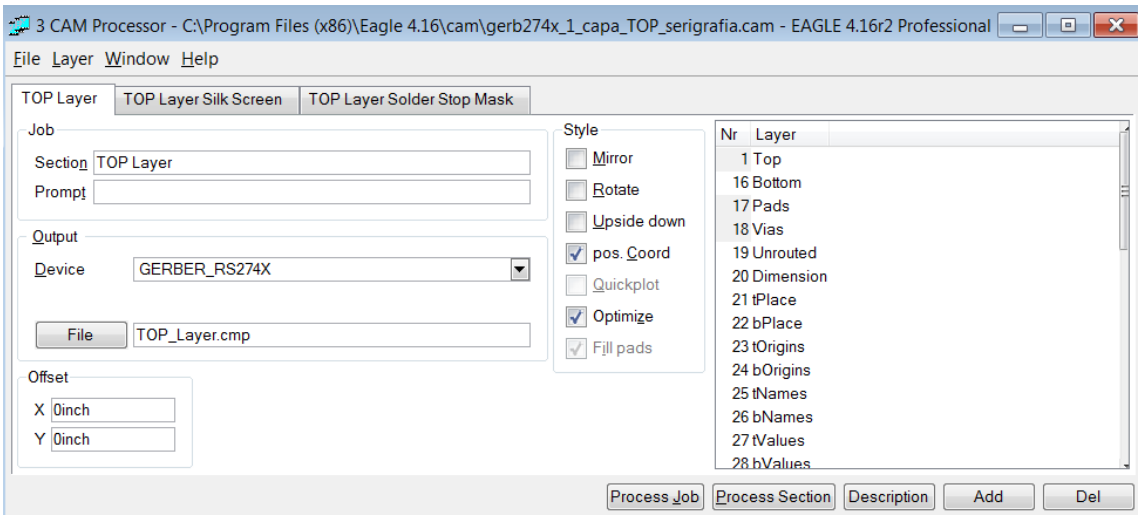


Figura 45: configuración de procesador CAM para FLX01.

CAPÍTULO 5 – DISEÑO DEL SOFTWARE DE IO5-USB

5.1 – ENTORNO DE DESARROLLO DEL SOFTWARE: MICROCHIP TECHNOLOGY MPLAB IDE V8.9

Para el desarrollo del software se empleará esta herramienta dado que es gratuita y proporcionada por MICROCHIP TECHNOLOGY para ser usada en el desarrollo de software para su familia de microcontroladores. Se usará la versión 8.9, que es la más reciente a fecha de redacción de esta memoria.

Esto entorno de desarrollo permite interactuar con paquetes de herramientas de software, pudiendo desarrollarse el software en distintos lenguajes de programación y con distintas funciones. MPLAB IDE ya incorpora de manera predeterminada la interfaz para la programación y depuración de software en lenguaje ensamblador pero este lenguaje resulta demasiado complejo y poco intuitivo para el desarrollo de aplicaciones complejas, en pro de un dominio preciso de las capacidades del microcontrolador en cuestión.

Por esa razón actualmente los microcontroladores suelen ser programados en lenguaje C aunque en muchos casos estos programas incluyan tramos de programación en ensamblador embebido en lenguaje C. Para el caso de IO5-USB se recurre a un compilador de microchip para programar el microcontrolador en lenguaje C.

5.1.1 – COMPILADOR EMPLEADO: MICROCHIP TECHNOLOGY MPLAB C18 V3.45

MICROCHIP TECHNOLOGY proporciona este compilador para el desarrollo de software en lenguaje C para su familia de microcontroladores PIC18, como es el caso de IO5-USB. Este compilador cuenta con una versión gratuita que no cuenta con los mismos mecanismos de optimización que la versión de pago pero para la programación de IO5-USB se empleará la gratuita y con todas las optimizaciones desactivadas, confiando en el diseñador la creación del programa más adecuado funcional y computacionalmente, minimizando ineficiencias.

5.2 – ESPECIFICACIONES A CUMPLIR POR EL SOFTWARE

El software desarrollado deberá cumplir una serie de requisitos mínimos que obedecen a las especificaciones de cara a la venta. Los productos similares de otras empresas dan una idea de las características a igualar o mejorar y serán una buena base para el desarrollo del software.

5.2.1 – MODOS DE FUNCIONAMIENTO DE IO5-USB

El sistema podrá funcionar en dos modos diferentes en analogía con los autómatas a los que emula: RUN y STOP. El modo RUN ejecutará el programa que haya guardado en la memoria de IO5-USB y el modo STOP detiene IO5-USB. Cada modo tiene ciertas peculiaridades: en el modo RUN es posible a través de USB leer el estado de las entradas, los contadores y las salidas, además de poder modificar contadores mientras que en STOP además de esas funcionalidades también es posible modificar el estado de las salidas y cambiar el programa de IO5-USB.

Todas las variables involucradas en el sistema pueden ser leídas en ambos modos de funcionamiento y "en tiempo real" si se realizan peticiones de manera continua por parte del software de PC gestor de IO5-USB.

5.2.2 – COMUNICACIÓN USB

La EUSART del PIC estará configurada a 115200 bits por segundo, en configuración de 8 bits de datos, 1 bit de parada y sin paridad. Aunque la conexión UART entre el PIC y el MCP2200 cuenta con las líneas CTS y RTS y teóricamente se pueden enviar datos desde el PIC al MCP2200 de manera continua siempre que esas líneas indiquen que es posible, existe en la práctica una especie de "bug" a la hora de monitorear datos en el PC que son enviados de manera continua desde el PIC: existen datos que, aunque existe evidencia de que han sido recibidos, no son mostrados por la aplicación de monitorización del puerto serie virtual creado por MCP2200.

A mayor velocidad de transmisión de datos más bytes no son mostrados, lo que lleva a la conclusión experimental de que todos los programas de monitorización empleados (DIGI X-CTU entre otros) muestran todos los datos si se respeta un tiempo entre bytes transmitidos de al menos 2 milisegundos. Aunque esto reducirá la tasa de transmisión de datos entre IO5-USB y el PC, servirá para conseguir una transmisión fiable y es un parámetro que podrá ser corregido conforme se compruebe el rendimiento del sistema según los distintos sistemas operativos en los que se use y las capacidades del software de PC gestor de IO5-USB.

Por otra parte hay que definir un protocolo de transmisión y recepción de mensajes que disponga de algún sistema de seguridad que detecte mensajes erróneos. Además habrá distintos tipos de mensajes según la tarea a realizar.

Para el protocolo de transmisión se empleará uno derivado de los módulos transceptores de radiofrecuencia DIGI XBEE. Este protocolo es útil por su sencillez y fiabilidad y la única modificación que se hará sobre él es que en vez de dos bytes indicadores del número de bytes del mensaje sólo habrá uno (dado que no se prevén tramas de más de 255 bytes seguidos, aparte de que el buffer de MCP2200 es de 64 bytes y ya es más que suficiente para IO5-USB).

Por tanto la estructura para el envío de datos será: primer byte de cabecera (0x7E, cualquier byte distinto cuando se espera una cabecera es automáticamente descartado), segundo byte longitud (número de bytes desde después de longitud y hasta el penúltimo inclusive), el resto de bytes son datos (que a su vez tienen una clasificación propia pero que desde el punto de vista global del mensaje son los datos o payload como se conoce en el argot de las telecomunicaciones) y el último byte siempre es el CRC (resultante de restar a 0xFF la suma de un único byte del valor numérico de todos los bytes de payload del mensaje).

La clasificación de los bytes de payload es como se explica a continuación.

5.2.3 – MODOS LÓGICOS DE GESTIÓN DE LAS SALIDAS

Las salidas dispondrán de dos modos lógicos de funcionamiento: AND y OR. Para estas funciones el software de PC gestor de IO5-USB ofrecerá casillas seleccionables para cada entrada (así como cada entrada negada) y el resultado en la salida correspondiente será dependiendo del cumplimiento de la combinación hecha entre AND de las salidas seleccionadas y de manera análoga para el caso de OR. Nótese que si para una misma entrada en el modo lógico OR se selecciona tanto la entrada como la entrada negada se estará forzando siempre a que se active la salida correspondiente.

Se elige de manera única o combinación OR o AND porque usarlas de manera mixta implicaría aumentar mucho la complejidad del sistema y esta clase de equipos no suele ofrecer tal funcionalidad (más propia de equipos mucho más caros como los autómatas).

5.2.4 – CONTADORES

IO5-USB dispondrá de 8 contadores, uno asociado a cada salida del sistema (lo que significa que cada contador podrá, si se configura para ello, accionar una salida pero sólo la salida a la que está asociado). Los contadores podrán ser accionados por cualquier entrada y funcionar como contador de flancos (tanto de subida como de bajada) como contador de pulsos (sólo de nivel alto).

El contador de flancos dispondrá de un modo de alta sensibilidad de detección (detección continua, si presentara ruido electrónico o un "rebote" al pulsar un hipotético pulsador para incrementar el contador correspondiente el contador se incrementaría rápidamente) y un modo de baja sensibilidad configurable, que descarte flancos cuya duración en el nivel de detección sea inferior al valor de configuración, entre 0 y 255 milisegundos.

Todos los contadores funcionarán incrementalmente desde 0 hasta 4294967295 (correspondiente a nivel de programación a una variable entera sin signo de 32 bits) y si se incrementa tras alcanzar esa cifra final el contador volverá a 0.

5.2.5 – PROGRAMACIÓN DE IO5-USB

El programa de IO5-USB deberá almacenarse en la EEPROM del PIC y por tanto deberá tener una estructura propia con tal de maximizar la utilidad de la memoria disponible. La distribución de la memoria se hará como se indica en la figura.

5.3 – DISEÑO DEL SOFTWARE

A continuación se abordará el diseño del software. Será diseñado de manera secuencial, respetando un formato amigable con su interpretación y maximizando el empleo de comentarios aclaratorios que favorecen no sólo su comprensión sino también su futura revisión dado que IO5-USB es susceptible de ampliaciones y mejoras en su funcionamiento. La codificación del programa de IO5-USB que se guarda en la memoria EEPROM se encuentra documentado en el Anexo K.

5.3.1 – DIAGRAMA DE FLUJO DEL SOFTWARE

Para comenzar a diseñar el software conviene partir de un diagrama de flujo general, que ilustre los pasos a seguir. En la Figura 46 se muestra el diagrama de flujo correspondiente al programa de IO5-USB (se trata de un diagrama escueto con propósitos ilustrativos dado que la cantidad de código es grande).

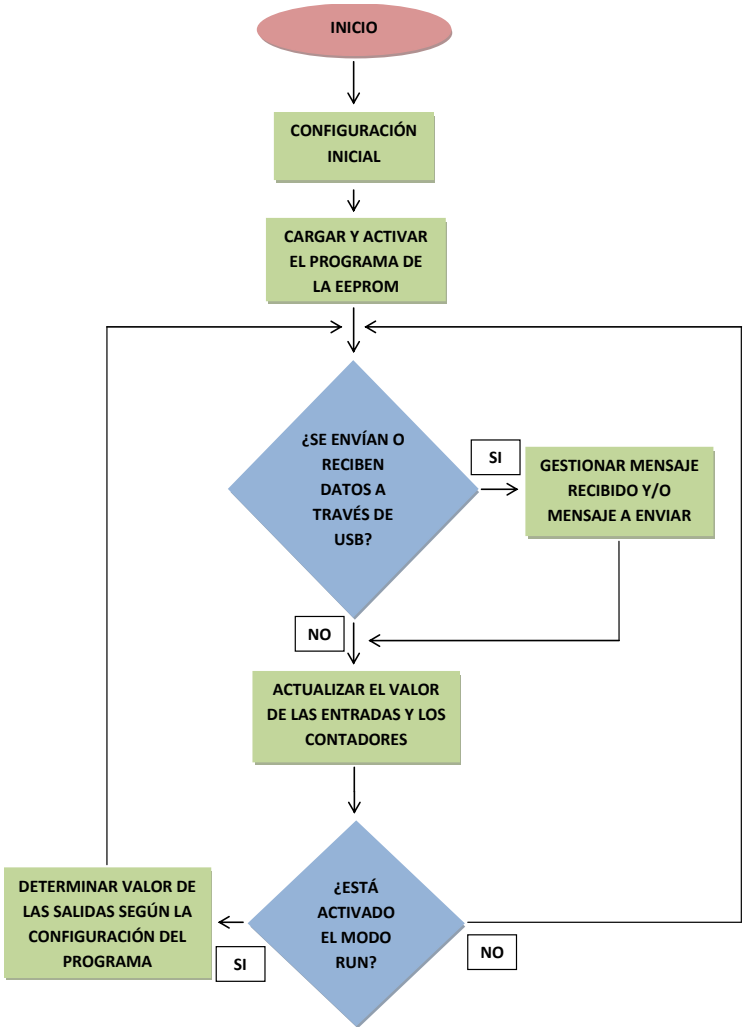


Figura 46: diagrama de flujo de programa.

El código será explicado de una manera secuencial, siguiendo "de arriba abajo" el fichero del código pero sin hacer hincapié en el código redundante dado que una parte significativa del código se basa en multiplicar para las otras siete salidas el código que se ha implementado para la primera, con modificaciones mínimas (simplemente modificando las entradas o salidas que interfieren de manera secuencial).

5.3.2 – LIBRERÍA Y AJUSTES DE CONFIGURACIÓN DEL MICROCONTROLADOR

El código debe comenzar especificando las librerías a usar y realizando los ajustes de la configuración del microcontrolador. En la Figura 47 se muestra el código correspondiente, donde los eventos más significativos son el uso de la librería correspondiente al PIC18LF26K22, la configuración del microcontrolador para funcionar con un oscilador de 16 megahercios y que se usará el PLL interno para multiplicar por 4 la frecuencia de reloj del microcontrolador.

```
//AUTOR: MAXIMO VICENTE AMOROS
//FECHA: JUNIO 2013
//MPLAB IDE V8.9, MPLAB C18 V3.45
//C18 PROJECT BUILD OPTIONS:TREAT CHAR AS UNSIGNED CHAR, MULTI-BANK STACK MEMORY MODEL, OPTIMIZATION DISABLE

#include <p18f26k22.h>

#pragma config FOSC = HSMP //HIGH SPEED MEDIUM POWER OSCILLATOR 16MHZ
#pragma config PLLCFG = ON // OSC X4: F=64MHZ
#pragma config PRICLKEN = ON   // OSC ON
#pragma config FCMEN    = OFF   // NO FAIL SAFE CLOCK MONITOR
#pragma config IESO     = OFF   // NO CLOCK CHANGE
#pragma config PWRTEN = OFF // NO POWER-UP TIMER
#pragma config BOREN = OFF //NO BROWNOUT RESET
#pragma config BORV     = 190   // BROWN-OUT VOLTAGE RESET
#pragma config WDTEN    = OFF   // NO WATCHDOG-TIMER
#pragma config WDTPS  = 32768  // WATCHDOG TIMER POSTSCALE
#pragma config CCP2MX  = PORTC1  // CCP2 MUX
#pragma config PBADEN = OFF //PORTB DIGITAL ON RESET
#pragma config CCP3MX  = PORTB5  // P3A/CCP3 MUX
#pragma config HFOFST = OFF // OUTPUT AND READY STATUS ARE DELAYED BY THE OSCILLATOR STABLE STATUS
#pragma config T3CMX  = PORTC0  // TIMER3 CLOCK INPUT
#pragma config P2BMX  = PORTC0  // ECCP2 B OUTPUT MUX
#pragma config MCLRE = INTMCLR //MASTER-CLEAR OFF
#pragma config STVREN = ON //STACK UNDERFLOW/OVERFLOW WILL RESET DEVICE
#pragma config LVP = OFF //ONE POWER SUPPLY ICSP OFF
#pragma config XINST = OFF //NO EXTENDED INSTRUCTIONS
#pragma config DEBUG = OFF //NO BACKROUND DEBUG

#pragma config CP0=ON,CP1=ON,CP2=ON,CP3=ON
#pragma config CPB      = ON
#pragma config CPD      = ON
#pragma config WRT0=ON,WRT1=ON,WRT2=ON,WRT3=ON
#pragma config WRTC  = ON
#pragma config WRTB  = ON
#pragma config WRTD = OFF //EEPROM
#pragma config EBTR0=ON,EBTR1=ON,EBTR2=ON,EBTR3=ON,EBTRB=ON     //TODAS LAS PROTECCIONES ACTIVADAS SALVO LA
                                                                 //DE ESCRITURA EN EEPROM
```

Figura 47: código de inicialización.

5.3.3 – DEFINICIÓN DE MACROS

Una tarea importante cuando se trabaja con varias entradas y/o salidas que además están numeradas es definirlas como macros, simplificando su uso en el programa y facilitando mucho el entendimiento del mismo. En este caso se definirán las entradas y salidas de una manera muy intuitiva: entradas de IN1 a IN8 y salidas de OUT1 a OUT8. También se definirán las máscaras lógicas (lo que contribuirá a que sean más fácilmente interpretados los algoritmos de decisión lógica) y las líneas de control de datos entre PIC y MCP2200 (y que son tratadas como I/O por el PIC). En la Figura 48 se muestra el fragmento correspondiente a la definición de macros en el código.

```
#define IN1 PORTAbits.RA0
#define IN2 PORTAbits.RA1
#define IN3 PORTAbits.RA2
#define IN4 PORTAbits.RA3
#define IN5 PORTAbits.RA4
#define IN6 PORTAbits.RA5
#define IN7 PORTCbits.RC0
#define IN8 PORTCbits.RC1
#define USB_CTS PORTCbits.RC2 //ENTRADA, SI ESTÁ A 0 SE PUEDEN ENVIAR DATOS A MCP2200
#define USB_RTS PORTCbits.RC3 //SALIDA, SI SE PONE A 0 SE LE INDICA A MCP2200 QUE PUEDE ENVIAR AL PIC
#define OUT1 PORTBbits.RB5
#define OUT2 PORTBbits.RB4
#define OUT3 PORTBbits.RB3
#define OUT4 PORTBbits.RB2
#define OUT5 PORTBbits.RB1
#define OUT6 PORTBbits.RB0
#define OUT7 PORTCbits.RC5
#define OUT8 PORTCbits.RC4
#define MASK1 0b00000001
#define MASK2 0b00000010
#define MASK3 0b00000100
#define MASK4 0b00001000
#define MASK5 0b00010000
#define MASK6 0b00100000
#define MASK7 0b01000000
#define MASK8 0b10000000
```

Figura 48: macros.

5.3.4 – DECLARACIÓN DE VARIABLES GLOBALES

A continuación se declaran las variables globales del programa, cuya naturaleza dependerá de su cometido. Al ser variables globales deben ser empleadas con cuidado dado que son una fuente típica de errores de programación pero la mayoría de ellas necesitan un valor estático a lo largo del programa. La declaración de variables globales está dividida en dos secciones debido a que MPLAB C18 no es capaz de asignar por sí sólo variables que, de manera única o en conjunto, ocupan más de 256 bytes (correspondiente a un banco de un PIC18). E los correspondientes sub-apartados se desglosará el cometido de cada una.

5.3.4.1 – GESTIÓN DE ENTRADAS

Las variables correspondientes a cada entrada son dos. Una guarda el valor recién leído (V_IN) mientras que la otra (V_ANT_IN) almacena ese mismo valor tras haberse comprobado la relación entre ambas. Esto permite detectar flancos y pulsos, en la Figura 49 se muestra la declaración de estas variables para las ocho entradas. Nótese que se emplea una variable de tipo byte para almacenar un parámetro almacenable en un bit (el nivel lógico de una entrada) pero no es adecuado hacer operaciones lógicas con bytes (máscaras de 8 bits, por ejemplo) y la dirección de una entrada: es una fuente típica de errores dado que se está haciendo una operación lógica entre elementos de distinta naturaleza.

```
#pragma udata seccion_1
//VARIABLES DE ENTRADAS
volatile char V_IN1,V_IN2,V_IN3,V_IN4,V_IN5,V_IN6,V_IN7,V_IN8; //VALORES "ACTUALES" DE LAS ENTRADAS
//VALORES "ANTERIORES" DE LAS ENTRADAS
volatile char V_ANT_IN1,V_ANT_IN2,V_ANT_IN3,V_ANT_IN4,V_ANT_IN5,V_ANT_IN6,V_ANT_IN7,V_ANT_IN8;
```

Figura 49: variables de entradas.

5.3.4.2 – GESTIÓN DE CONTADORES

Los contadores son declarados como números enteros sin signo de 32 bits. El valor actual de un contador se guarda en CONTADOR mientras que el umbral si lo hubiere para accionar su correspondiente salida se guarda en UMBRAL_CONT. Para el modo de pulsos n_MS_PULSO cuenta el número de milisegundos que un contador está registrando un pulso a nivel alto en la entrada seleccionada, para incrementar el contador al compararse con UMBRAL_CONT. La entrada para incrementar un contador o para volver su cuenta a 0 corresponden respectivamente a IN_CONT e IN_CONT_RESET respectivamente.

La configuración de cada contador corresponde a los bytes CONFIG_CONT_B1/B2 y las variable BAJA_SENS corresponde a la cuenta que se hace respecto de UMBRAL_BAJA_SENSIB (una variable global que recoge el umbral de baja sensibilidad entre 0 y 255 milisegundos). PULSO_CONT_OK actúa como variable auxiliar en la cuenta de pulsos. En la Figura 50 se muestra el código correspondiente a las variables descritas.

```
//VARIABLES DE CONTADORES
volatile unsigned long
CONTADOR1=0,UMBRAL_CONT1,CONTADOR2=0,UMBRAL_CONT2,CONTADOR3=0,UMBRAL_CONT3,CONTADOR4=0,
UMBRAL_CONT4,CONTADOR5=0,UMBRAL_CONT5,CONTADOR6=0,UMBRAL_CONT6,CONTADOR7=0,UMBRAL_CONT7,
CONTADOR8=0,UMBRAL_CONT8;
volatile unsigned short n_MS_PULSO1=0, UMBRAL_MS_PULSO1,n_MS_PULSO2=0, UMBRAL_MS_PULSO2,n_MS_PULSO3=0,
UMBRAL_MS_PULSO3,n_MS_PULSO4=0, UMBRAL_MS_PULSO4;
volatile unsigned short n_MS_PULSO5=0, UMBRAL_MS_PULSO5,n_MS_PULSO6=0, UMBRAL_MS_PULSO6,n_MS_PULSO7=0,
UMBRAL_MS_PULSO7,n_MS_PULSO8=0, UMBRAL_MS_PULSO8;
volatile char
IN_CONT1,IN_RESET_CONT1,CONFIG_CONT1_B1,IN_CONT2,IN_RESET_CONT2,CONFIG_CONT2_B1,IN_CONT3,IN_RESET_CONT3,
CONFIG_CONT3_B1,IN_CONT4,IN_RESET_CONT4,CONFIG_CONT4_B1;
volatile char
IN_CONT5,IN_RESET_CONT5,CONFIG_CONT5_B1,IN_CONT6,IN_RESET_CONT6,CONFIG_CONT6_B1,IN_CONT7,IN_RESET_CONT7,
CONFIG_CONT7_B1,IN_CONT8,IN_RESET_CONT8,CONFIG_CONT8_B1;
volatile char
PULSO_CONT1_OK=0xFF,PULSO_CONT2_OK=0xFF,PULSO_CONT3_OK=0xFF,PULSO_CONT4_OK=0xFF,PULSO_CONT5_OK=0xFF,
PULSO_CONT6_OK=0xFF,PULSO_CONT7_OK=0xFF,PULSO_CONT8_OK=0xFF;
volatile char
CONFIG_CONT1_B2,CONFIG_CONT2_B2,CONFIG_CONT3_B2,CONFIG_CONT4_B2,CONFIG_CONT5_B2,CONFIG_CONT6_B2,
CONFIG_CONT7_B2,CONFIG_CONT8_B2;
volatile char
BAJA_SENS_1=0xFF,BAJA_SENS_2=0xFF,BAJA_SENS_3=0xFF,BAJA_SENS_4=0xFF,BAJA_SENS_5=0xFF,BAJA_SENS_6=0xFF,
BAJA_SENS_7=0xFF,BAJA_SENS_8=0xFF;
volatile char UMBRAL_BAJA_SENSIB;
```

Figura 50: variables de contadores.

5.3.4.3 – CONFIGURACIÓN DE LAS SALIDAS Y DEL MODO DE FUNCIONAMIENTO

La configuración de cada salida se guarda en CONFIG_OUT y para los modos lógicos tanto las entradas como las entradas negadas forman una máscara guardada en OUT_MASK_INPUTS y OUT_MASK_INPUTS_N respectivamente y de manera complementaria entre sí. DEFAULT_OUTS recoge del programa que haya en memoria el estado predeterminado de las salidas cuando el programa arranca. En la Figura 51 se muestran estas variables para las ocho salidas y la variable que define el modo en que funciona el programa (RUN o STOP son los únicos modos disponibles en esta versión de IO5-USB).

```
//VARIABLES DE CONFIGURACION DE SALIDAS
volatile char
CONFIG_OUT1,CONFIG_OUT2,CONFIG_OUT3,CONFIG_OUT4,CONFIG_OUT5,CONFIG_OUT6,CONFIG_OUT7,CONFIG_OUT8;
volatile char OUT1_MASK_INPUTS,OUT1_MASK_INPUTS_N,OUT2_MASK_INPUTS,OUT2_MASK_INPUTS_N,OUT3_MASK_INPUTS,
OUT3_MASK_INPUTS_N,OUT4_MASK_INPUTS,OUT4_MASK_INPUTS_N;
volatile char OUT5_MASK_INPUTS,OUT5_MASK_INPUTS_N,OUT6_MASK_INPUTS,OUT6_MASK_INPUTS_N,OUT7_MASK_INPUTS,
OUT7_MASK_INPUTS_N,OUT8_MASK_INPUTS,OUT8_MASK_INPUTS_N;
//VARIABLES DEL MODO DE FUNCIONAMIENTO
volatile char MODO=0xFF;//PREDETERMINADO EL MODO STOP
volatile char DEFAULT_OUTS;//ESTADO PREDETERMINADO DE LAS SALIDAS
```

Figura 51: variables de salidas y de modo.

5.3.4.4 – BUFFERS PARA LA COMUNICACIÓN USB Y VARIABLES DE USO GENERAL

Dado que es necesario almacenar cada byte que se recibe hasta completar un mensaje y poder analizarlo (o descartarlo) son necesarios buffers tanto de transmisión como de recepción. De momento no se prevé que pueda haber mensajes de más de 64 bytes (que es el tamaño del buffer de MCP2200), n_uart1_rx almacena la cuenta de bytes recibidos, n_uart1_tx la cuenta de número de bytes a enviar y uart1_timeout maneja un sistema de caducidad de mensajes, que se activa cuando llega un mensaje y cuenta el número de segundos que tarda en completarse un mensaje, descartando los mensajes que tardan más de dos segundos en completarse.

Las variables de uso general son bytes empleados para bucles y almacenamiento temporal dentro del mismo bucle, en la Figura 52 se muestran las variables descritas.

```
#pragma udata seccion_2
//VARIABLES DE BUFFERS TX/RX DE USB
volatile char uart1_rx[65];
volatile char uart1_tx[65];
volatile char n_uart1_rx=0,n_uart1_tx=0,uart1_timeout=255;      //SI TIMEOUT ES 255 ESTÁ DESACTIVADO, SINO SE
                                                                //INCREMENTA CADA SEGUNDO CON TMR0
//VARIABLES DE USO GENERAL
volatile char clean,clean2,i,j,k;
```

Figura 52: variables de USB y de propósito general.

5.3.5 – FUNCIONES

Se usan un total de once funciones en el programa aunque nueve de ellas están declaradas como funciones para hacer más legible y entendible el programa. Las siete funciones que corresponden a UART1TX son respuestas que deben enviarse por USB cuando se presentan los eventos que forman parte de su nombre: timeout (se acabó el tiempo de transmisión de un mensaje que se ha recibido de manera incompleta), crc_error (su propio nombre indica la incidencia), leer_modo (envía el modo de funcionamiento actual de IO5-USB), tx_mismo_rx (respuesta automática de enviar el mismo mensaje que se ha recibido por USB, se emplea a la hora de configurar y programar IO5-USB) msj_desconocido (se ha recibido un mensaje no conocido por la actual versión del software de IO5-USB), estado_act_e_s (envía el nivel lógico de las entradas y salidas) y run_no_modif_s (se ha intentado modificar las salidas con IO5-USB en modo RUN y se notifica que eso no es posible). En la Figura 53 se muestran los prototipos de todas las funciones del programa.

```
//FUNCIONES
void UART1TX_TIMEOUT(void);
void UART1TX_CRC_ERROR(void);
void UART1TX_LEER_MODO(void);
void UART1TX_MISMO_RX(void);
void UART1TX_MSJ_DESCONOCIDO(void);
void UART1TX_ESTADO_ACT_E_S(void);
void UART1TX_RUN_NO_MODIF_S(void);
void MODIFICA_VALOR_CONTADOR(void);
void LEER_CONTADORES(void);
char ReadEEPROM(char dirH,char dirL);
void WriteEEPROM(char dirH,char dirL,char data);
```

Figura 53: prototipos de funciones.

Las funciones MODIFICA_VALOR_CONTADOR y LEER_CONTADORES tienen un cometido bien descrito por su nombre mientras que las funciones ReadEEPROM y WriteEEPROM se encargan de gestionar la EEPROM del PIC: ReadEEPROM admite como parámetros el byte alto y bajo de la posición a leer y devuelve el byte que haya en esa posición mienras que WriteEEPROM recoge la posición en la que se va a escribir un byte y dicho byte para proceder a su escritura. En las siguientes Figuras se recoge el código correspondiente a cada función, las cuales se encuentran por motivos de organización al final del fichero de programa, una vez cerrado el bucle MAIN.

```
//FUNCTIONS
void WriteEEPROM(char dirH,char dirL,char data)
{
	clean2=INTCON;
	INTCONbits.GIE=0;
	do
	{
		EECON1=0;		//ASEGURARSE DE QUE CFGS=0 Y EEPGD=0
		EECON1bits.WREN = 1;	//HABILITAR ESCRITURA DE EEPROM
		EEADRH = dirH;			//CONFIGURAR DIRECCIÓN HIGH
		EEADR = dirL;	//CONFIGURAR DIRECCIÓN LOW
		EEDATA = data;		//INTRODUCIR BYTE A ESCRIBIR
		INTCON=0;
		EECON2 = 0x55;		//SECUENCIA 1
		EECON2 = 0xAA;		//SECUENCIA 2
		EECON1bits.WR = 1;	//SECUENCIA 3 - ESCRITURA
		while(EECON1bits.WR);			//ESPERAR A FIN DE ESCRITURA
		while(!PIR2bits.EEIF);	//ESPERAR A QUE SE LEVANTE EL FLAG DE FIN DE ESCRITURA
		//if(EECON1bits.WRERR) Nop();  //NO SE COMPRUEBA SI HAY ERROR PORQUE SI NO SE HA ESCRITO BIEN EL
						//DATO SE REPITE LA ESCRITURA
		EECON1bits.WREN = 0;	//DESHABILITAR ESCRITURA EN LA EEPROM
		PIR2bits.EEIF = 0;				//SE BAJA EL FLAG DE FIN DE ESCRITURA PARA HABILITAR
						//FUTURAS ESCRITURAS
		EECON1=0; //ASEGURARSE DE QUE CFGS=0 Y EEPGD=0
		EEADRH = dirH;
		EEADR = dirL;
		EECON1bits.RD = 1;   //LA LECTURA SOLO TOMA UN CICLO, NO HAY QUE ESPERAR NADA DESPUES DE ESTO
					//PARA RECOGER EL VALOR LEIDO
		clean=EEDATA;
	} while(clean!=data);  //SE LEE DE NUEVO EL DATO PARA COMPROBAR QUE SE HA ESCRITO BIEN
	INTCON=clean2;
}
```

Figura 54: función WRITEEEPROM.

```
char ReadEEPROM(char dirH,char dirL)
{
	EECON1=0; //ASEGURARSE DE QUE CFGS=0 Y EEPGD=0
	EEADRH = dirH;
	EEADR = dirL;
	EECON1bits.RD = 1;//LEER EEPROM SOLO TOMA UN CICLO
	return(EEDATA);
}
```

Figura 55: función READEEPROM.

```
void UART1TX_CRC_ERROR(void)
{
	uart1_tx[1]=0x01;
	uart1_tx[2]=0xAA;
	uart1_tx[3]=0x55;
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 56: función UART1TX_CRC_ERROR.

```
void UART1TX_TIMEOUT(void)
{
	uart1_tx[1]=0x01;
	uart1_tx[2]=0xAB;
	uart1_tx[3]=0x54;
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 57: función UART1TX_TIMEOUT.

```
void UART1TX_LEER_MODO(void)
{
	uart1_tx[1]=0x02;
	uart1_tx[2]=0x00;
	uart1_tx[3]=MODO;//ESTE ES EL MODO ACTUAL
	uart1_tx[4]=(0xFF-clean);//COMO SOLO HAY DOS BYTES DE DATOS Y UNO ES 0x00 EL CRC ES ASI
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 58: función UART1TX_LEER_MODO.

```
void UART1TX_MSJ_DESCONOCIDO(void)
{
	uart1_tx[1]=0x01;
	uart1_tx[2]=0xAC;
	uart1_tx[3]=0x53;
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 59: función UART1TX_MSJ_DESCONOCIDO.

```
void UART1TX_MISMO_RX(void)
{
	for(i=1;i<n_uart1_rx;i++) //SE DEVUELVE EL MISMO MENSAJE RECIBIDO, SE USA n_uart1_rx COMO REFERENCIA, ESTO
				  //PODRIA SER ERRONEO SI LLEGARA ALGUN BYTE UNA VEZ SE ESTÁ ANALIZANDO ESTO
	{
		uart1_tx[i]=uart1_rx[i];
	}
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 60: función UART1TX_MISMO_RX.

```
void UART1TX_ESTADO_ACT_E_S(void)
{	//LA VARIABLE j SE USA PARA LAS ENTRADAS, LA k PARA LAS SALIDAS Y LA i PARA EL CRC
	j=0;k=0;
	if(IN1) j|=MASK1;
	if(IN2) j|=MASK2;
	if(IN3) j|=MASK3;
	if(IN4) j|=MASK4;
	if(IN5) j|=MASK5;
	if(IN6) j|=MASK6;
	if(IN7) j|=MASK7;
	if(IN8) j|=MASK8;
	if(OUT1) k|=MASK1;
	if(OUT2) k|=MASK2;
	if(OUT3) k|=MASK3;
	if(OUT4) k|=MASK4;
	if(OUT5) k|=MASK5;
	if(OUT6) k|=MASK6;
	if(OUT7) k|=MASK7;
	if(OUT8) k|=MASK8;
	i=0xFE-((char)(j+k));//COMO EL TIPO DE MENSAJE ES 1 YA SE LO HE DESCONTADO AL CRC, USANDO 0xFE
	uart1_tx[1]=0x03;
	uart1_tx[2]=0x01;
	uart1_tx[3]=j;//ENTRADAS
	uart1_tx[4]=k;//SALIDAS
	uart1_tx[5]=i;//CRC
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 61: función UART1TX_ESTADO_ACT_E_S.

```
void UART1TX_RUN_NO_MODIF_S(void)
{
	uart1_tx[1]=0x03;
	uart1_tx[2]=0x02;
	uart1_tx[3]=0xFF;
	uart1_tx[4]=0xFF;
	uart1_tx[5]=0xFF;
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 62: función UART1TX_RUN_NO_MODIF_S.

```
void MODIFICA_VALOR_CONTADOR(void)
{
	if(uart1_rx[3]==(char)0)//MODIFICAR CONTADOR 1
	{
		CONTADOR1=(unsigned long)uart1_rx[4];CONTADOR1*256;
		CONTADOR1+=(unsigned long)uart1_rx[5];CONTADOR1*256;
		CONTADOR1+=(unsigned long)uart1_rx[6];CONTADOR1*256;
		CONTADOR1+=(unsigned long)uart1_rx[7];
	}
	else if(uart1_rx[3]==(char)1)//MODIFICAR CONTADOR 2
	{
		CONTADOR2=(unsigned long)uart1_rx[4];CONTADOR2*256;
		CONTADOR2+=(unsigned long)uart1_rx[5];CONTADOR2*256;
		CONTADOR2+=(unsigned long)uart1_rx[6];CONTADOR2*256;
		CONTADOR2+=(unsigned long)uart1_rx[7];
	}
	else if(uart1_rx[3]==(char)2)//MODIFICAR CONTADOR 3
	{
		CONTADOR3=(unsigned long)uart1_rx[4];CONTADOR3*256;
		CONTADOR3+=(unsigned long)uart1_rx[5];CONTADOR3*256;
		CONTADOR3+=(unsigned long)uart1_rx[6];CONTADOR3*256;
		CONTADOR3+=(unsigned long)uart1_rx[7];
	}
	else if(uart1_rx[3]==(char)3)//MODIFICAR CONTADOR 4
	{
		CONTADOR4=(unsigned long)uart1_rx[4];CONTADOR4*256;
		CONTADOR4+=(unsigned long)uart1_rx[5];CONTADOR4*256;
		CONTADOR4+=(unsigned long)uart1_rx[6];CONTADOR4*256;
		CONTADOR4+=(unsigned long)uart1_rx[7];
	}
	else if(uart1_rx[3]==(char)4)//MODIFICAR CONTADOR 5
	{
		CONTADOR5=(unsigned long)uart1_rx[4];CONTADOR5*256;
		CONTADOR5+=(unsigned long)uart1_rx[5];CONTADOR5*256;
		CONTADOR5+=(unsigned long)uart1_rx[6];CONTADOR5*256;
		CONTADOR5+=(unsigned long)uart1_rx[7];
	}
	else if(uart1_rx[3]==(char)5)//MODIFICAR CONTADOR 6
	{
		CONTADOR6=(unsigned long)uart1_rx[4];CONTADOR6*256;
		CONTADOR6+=(unsigned long)uart1_rx[5];CONTADOR6*256;
		CONTADOR6+=(unsigned long)uart1_rx[6];CONTADOR6*256;
		CONTADOR6+=(unsigned long)uart1_rx[7];
	}
	else if(uart1_rx[3]==(char)6)//MODIFICAR CONTADOR 7
	{
		CONTADOR7=(unsigned long)uart1_rx[4];CONTADOR7*256;
		CONTADOR7+=(unsigned long)uart1_rx[5];CONTADOR7*256;
		CONTADOR7+=(unsigned long)uart1_rx[6];CONTADOR7*256;
		CONTADOR7+=(unsigned long)uart1_rx[7];
	}
	else	//MODIFICAR CONTADOR 8, YA SE HA ENTRADO A ESTA RUTINA DE MODIFICA_VALOR_CONTADOR
		//COMPROBANDO QUE EL VALOR MAXIMO ES 7
	{
		CONTADOR8=(unsigned long)uart1_rx[4];CONTADOR8*256;
		CONTADOR8+=(unsigned long)uart1_rx[5];CONTADOR8*256;
		CONTADOR8+=(unsigned long)uart1_rx[6];CONTADOR8*256;
		CONTADOR8+=(unsigned long)uart1_rx[7];
	}
}
```

Figura 63: función MODIFICA_VALOR_CONTADOR.

```
void LEER_CONTADORES(void)
{
	uart1_tx[1]=0x26;
	uart1_tx[2]=0x04;
	uart1_tx[3]=0xFE;
	uart1_tx[4]=(char)(CONTADOR1>>24);
	uart1_tx[5]=(char)(CONTADOR1>>16);
	uart1_tx[6]=(char)(CONTADOR1>>8);
	uart1_tx[7]=(char)CONTADOR1;
	uart1_tx[8]=(char)(CONTADOR2>>24);
	uart1_tx[9]=(char)(CONTADOR2>>16);
	uart1_tx[10]=(char)(CONTADOR2>>8);
	uart1_tx[11]=(char)CONTADOR2;
	uart1_tx[12]=(char)(CONTADOR3>>24);
	uart1_tx[13]=(char)(CONTADOR3>>16);
	uart1_tx[14]=(char)(CONTADOR3>>8);
	uart1_tx[15]=(char)CONTADOR3;
	uart1_tx[16]=(char)(CONTADOR4>>24);
	uart1_tx[17]=(char)(CONTADOR4>>16);
	uart1_tx[18]=(char)(CONTADOR4>>8);
	uart1_tx[19]=(char)CONTADOR4;
	uart1_tx[20]=(char)(CONTADOR5>>24);
	uart1_tx[21]=(char)(CONTADOR5>>16);
	uart1_tx[22]=(char)(CONTADOR5>>8);
	uart1_tx[23]=(char)CONTADOR5;
	uart1_tx[24]=(char)(CONTADOR6>>24);
	uart1_tx[25]=(char)(CONTADOR6>>16);
	uart1_tx[26]=(char)(CONTADOR6>>8);
	uart1_tx[27]=(char)CONTADOR6;
	uart1_tx[28]=(char)(CONTADOR7>>24);
	uart1_tx[29]=(char)(CONTADOR7>>16);
	uart1_tx[30]=(char)(CONTADOR7>>8);
	uart1_tx[31]=(char)CONTADOR7;
	uart1_tx[32]=(char)(CONTADOR8>>24);
	uart1_tx[33]=(char)(CONTADOR8>>16);
	uart1_tx[34]=(char)(CONTADOR8>>8);
	uart1_tx[35]=(char)CONTADOR8;
	//LAS POSICIONES DE LA 36 A LA 39 SE HAN ESCRITO ANTES DE LLAMAR A ESTA FUNCION
	j=0;
	for(i=2;i<(char)40;i++) j+=uart1_tx[i];
	i=0xFF-j;
	uart1_tx[40]=i;//CRC
	n_uart1_tx=0;
	T1CONbits.TMR1ON=1;//CONECTAR TIMER1, ENVIO EN MARCHA
}
```

Figura 64: función LEER_CONTADORES.

5.3.6 – RUTINA DE GESTIÓN DE INTERRUPCIONES

Tras la declaración de los prototipos de las funciones del programa se emplaza la rutina de gestión de interrupciones (sólo se emplea el vector de interrupciones de alta prioridad), a la que se le han reservado las tareas que requieren atención inmediata por parte del microcontrolador. Estas tareas son la gestión de la recepción de datos a través de USB y el funcionamiento del reloj segundero del equipo. Dicha rutina vuelve a habilitar las interrupciones al salir de la misma luego no es necesario ocuparse explícitamente de ello a la hora de salir de la rutina, la cual se muestra en la Figura 65 junto con su declaración en C.

```
#pragma interrupt H_ISR
void H_ISR(void)
{
	if(PIR1bits.RC1IF) //¿SE HA RECIBIDO ALGO POR UART1?
	{
		if(n_uart1_rx==(char)0) uart1_timeout=0;	//EN EL PRIMER BYTE RECIBIDO SE ACTIVA EL
								//SISTEMA DE TIMEOUT
		uart1_rx[n_uart1_rx]=RCREG1;
		if(n_uart1_rx<(char)64) n_uart1_rx++;		//SE INCREMENTA MIENTRAS NO SE SOBREPASE LA LONGITUD
								//MAXIMA
	}

	if(INTCONbits.TMR0IF) //¿HA PASADO UN SEGUNDO EN TIMER0?
	{
		T0CONbits.TMR0ON=0; //TIMER0 OFF
		INTCONbits.TMR0IF=0; //HAY QUE BAJAR EL FLAG DE DESBORDAMIENTO DEL TMR0
		TMR0H=0x0B;TMR0L=0xDC; //1S@64MHz
		T0CONbits.TMR0ON=1; //TIMER0 ON
		if(uart1_timeout<(char)254) uart1_timeout++; //254 ES EL LIMITE INCREMENTANDOSE AQUI, 255 ES
								//DESACTIVADO
	}

}
#pragma code H_VECTOR=0x08
void H_VECTOR (void)
	{
	_asm GOTO H_ISR _endasm
	}
#pragma code
```

Figura 65: rutina de gestión de interrupciones.

5.3.7 – SOFTWARE DEL BLOQUE DE INICIALIZACIÓN

Llegados a este punto, el siguiente paso es inicializar el programa, que consta de tres pasos fundamentales: configurar inicialmente al PIC (timers, EUSART, entradas y salidas), leer el programa (si lo hay) de la EEPROM y cargarlo en las variables de funcionamiento del programa (formado por una parte por las salidas y por otra por los contadores) y por último habilitar la comunicación de IO5-USB con el exterior, los timers y las interrupciones. En los cuatro sub-apartados que hay a continuación se desglosan dichas tareas.

5.3.7.1 – CONFIGURACIÓN DE EUSART, E/S Y TIMERS

Las configuraciones tras el inicio de la función main tienen un carácter especial ya que son absolutas: timers, EUSART y puertos digitales son configurados sin que vuelvan a serlo en el resto del programa. El TIMER0 se encarga de mantener una referencia de tiempo en el sistema (teóricamente sería posible implementar a partir de él un reloj en tiempo real para IO5-USB). TIMER1 gestiona un pequeño retardo en las transmisiones vía USB debido a lo explicado en el punto 5.2.2. El TIMER2 se encarga del modo de baja sensibilidad de los contadores de flancos y el TIMER3 es exclusivo para los contadores de pulsos. En la Figura 66 se expone el código correspondiente a este bloque inicial.

```
void main(void)
{
	LATA=0;
	LATB=0;
	LATC=0;

	ANSELA=0;
	ANSELB=0;
	ANSELC=0;

	TRISA=0b00111111;//RA0 ... RA5 INPUTS
	TRISB=0;//OUTPUTS (EL MODELO USB NO USA EL TELIT QUE ESTÁ EN UART2)
	TRISC=0b11000111;//RC3 (USB RTS), RC4 (OUT8) AND RC5 (OUT7) OUTPUTS, EVERYTHING ELSE INPUTS

	//UART1 (USB) TX ON RX ON (SYNC=0 BRGH=1 BRG16=1)
	TXSTA1=0b00100100;
	RCSTA1=0b10010000;
	BAUDCON1=0b00001000;
	SPBRG1=138;//115200 @ 64MHZ
	SPBRGH1=0;//115200 @ 64MHZ
	//SPBRG1=0x40;//19200 @ 64MHZ
	//SPBRGH1=0x03;//19200 @ 64MHZ

	//TIMER0 CONFIGURATION(16BITS PRESC=1:256) (INTERRUPCION CADA 1 SEGUNDO)
	T0CON=0b00000111;
	TMR0H=0x0B;TMR0L=0xDC; //1S@64MHz

	//TIMER1 CONFIGURATION (16 BITS PRESC=1:8) (ES EL TIMER DE LOS ENVIOS POR USB)
	T1CON=0b00110100;
	TMR1H=0xFF;TMR1L=0xFF;//A PUNTO DE DESBORDAR CUANDO SE CONECTE, PARA QUE EMPIECE PRONTO UN ENVIO

	//TIMER2 CONFIGURATION (8 BITS PRESC=1:16 POSTSC=1:4) (ES EL TIMER DE LOS RETARDOS PARA POLLING DE LOS
	//CONTADORES DE FLANCO DE BAJA SENSIBILIDAD)
	T2CON=0b00011011;
	TMR2=0x06;//1mS@64MHz

	//TIMER3 CONFIGURATION (16 BITS PRESC=1:8) (ES EL TIMER DE LOS CONTADORES DE PULSOS)
	T3CON=0b00110100;
	TMR3H=0xF8;TMR3L=0x58;//980uS@64MHz
```

Figura 66: configuración de EUSART, E/S Y timers.

5.3.7.2 – CONFIGURACIÓN DE LA SALIDA 1

De acuerdo con los bits de configuración se configura la salida 1 leyendo de la EEPROM, el método empleado se muestra en la Figura 67.

```
//CONFIGURACION DE LA SALIDA 1
CONFIG_OUT1=ReadEEPROM(0,2);
if(CONFIG_OUT1<(char)3)        //¿LA SALIDA ESTA CONFIGURADA EN UN MODO CONOCIDO?
{
        OUT1_MASK_INPUTS=ReadEEPROM(0,3);
        OUT1_MASK_INPUTS_N=ReadEEPROM(0,4);
}
if(CONFIG_OUT1==(char)2)       //¿SE HA SELECCIONADO EL MODO CONTADOR?
{
        //SE ANOTA EL UMBRAL, ES EL MISMO PARA AMBOS MODOS
        UMBRAL_CONT1=(unsigned long)OUT1_MASK_INPUTS;
        UMBRAL_CONT1*=256;//ASI PASAMOS AL TERCER BYTE LO QUE ESTABA EN EL BYTE BAJO
        UMBRAL_CONT1+=(unsigned long)OUT1_MASK_INPUTS_N;
        UMBRAL_CONT1*=256;//ASI SUBIMOS AL CONJUNTO OTRO BYTE
        UMBRAL_CONT1+=(unsigned long)ReadEEPROM(0,5);
        UMBRAL_CONT1*=256;//ASI SUBIMOS AL CONJUNTO OTRO BYTE
        UMBRAL_CONT1+=(unsigned long)ReadEEPROM(0,6);
}
```

Figura 67: configuración de la salida 1.

5.3.7.3 – CONFIGURACIÓN DEL CONTADOR 1

La configuración del contador 1 depende de la configuración ya descrita. Una vez se ha finalizado de configurar se suceden secuencialmente la configuración de la salida 2 y del contador 2, hasta acabar con el 8. No existe diferencia en el código salvo la sustitución del 1 por el número de salida o contador que corresponda en el código y la posición de EEPROM de donde se toma la configuración de acuerdo con la distribución de la memoria EEPROM en el apartado 5.2.5 y el código es mostrado en la Figura 68.

```
//CONFIGURACION DEL CONTADOR 1
CONFIG_CONT1_B1=ReadEEPROM(0,0x2A);//LEER PRIMER BYTE DE CONFIGURACION DEL CONTADOR1
i=CONFIG_CONT1_B1>>5;
if(i<(char)3)//¿CONTADOR ACTIVADO EN UN MODO CONOCIDO?
{
        CONFIG_CONT1_B2=ReadEEPROM(0,0x2B);//SE CARGA EL OTRO BYTE DE CONFIGURACION YA QUE SE VA A
                                                //USAR EL CONTADOR
        //CONFIGURAR LA ENTRADA QUE HACE RESET DEL CONTADOR SI ESTO ESTA CONFIGURADO
        if(CONFIG_CONT1_B1&MASK4) IN_RESET_CONT1=(CONFIG_CONT1_B1&0b00000111);
        IN_CONT1=(CONFIG_CONT1_B2&0b11100000)>>5;//CONFIGURAR LA ENTRADA DEL CONTADOR, 0 ES IN1 Y
                                                //7 ES IN8
        if(i==(char)2)//¿MODO CONTADOR DE PULSOS DE NIVEL ALTO?
        {
                j=CONFIG_CONT1_B2&0b00011111;//BITS ALTOS DEL Nº DE MS A NIVEL ALTO PARA CONTAR UN
                                                //PULSO
                UMBRAL_MS_PULSO1=(unsigned short)j*256;//ASI PASAMOS AL BYTE ALTO LO QUE ESTABA EN
                                                //EL BYTE BAJO
                UMBRAL_MS_PULSO1+=(unsigned short)ReadEEPROM(0,0x2C);
                //YA SE HA CARGADO EL //NUMERO DE MS DE UMBRAL PARA CONTAR UN PULSO
        }
}
```

Figura 68: configuración del contador 1.

5.3.7.4 – AJUSTES FINALES E INICIALIZACIÓN DEL BUCLE DE PROGRAMA

A continuación se procede a preparar el resto de variables que aún no han sido actualizadas para que el equipo pueda entrar dentro del bucle de programa, donde estará funcionando el resto del tiempo hasta que sea desconectado o programado de nuevo. La primera tarea es leer del programa grabado el umbral de baja sensibilidad, después se actualiza la variable correspondiente al estado "antiguo" de las entradas para que si está activo algún contador por flanco no tome como flanco que esa variable esté a un valor diferente al valor actual de la entrada sólo por inicialización del programa.

Posteriormente las salidas se colocan en el estado predeterminado que dicte el programa grabado en el PIC, se lee el modo de funcionamiento y finalmente se habilitan timers, interrupciones y la comunicación USB. A partir de estas tareas, ilustradas en la Figura 69, IO5-USB está listo para operar.

```
UMBRAL_BAJA_SENSIB=ReadEEPROM(0,0x42);          //SE LEE EL UMBRAL DE BAJA SENSIBILIDAD

//SE ACTUALIZA EL ESTADO "ANTERIOR" DE LAS ENTRADAS, PARA QUE LOS MODOS DE FLANCOS Y PULSOS NO
//CUENTEN AL CONECTAR SEL16
if(IN1) V_ANT_IN1=0xFF; else V_ANT_IN1=0;
if(IN2) V_ANT_IN2=0xFF; else V_ANT_IN2=0;
if(IN3) V_ANT_IN3=0xFF; else V_ANT_IN3=0;
if(IN4) V_ANT_IN4=0xFF; else V_ANT_IN4=0;
if(IN5) V_ANT_IN5=0xFF; else V_ANT_IN5=0;
if(IN6) V_ANT_IN6=0xFF; else V_ANT_IN6=0;
if(IN7) V_ANT_IN7=0xFF; else V_ANT_IN7=0;
if(IN8) V_ANT_IN8=0xFF; else V_ANT_IN8=0;


//ACTUALIZAR ESTADO PREDETERMINADO DE LAS SALIDAS
DEFAULT_OUTS=ReadEEPROM(0,1);//LEER EL ESTADO PREDETERMINADO
if(DEFAULT_OUTS&MASK1) OUT1=1; else OUT1=0;
if(DEFAULT_OUTS&MASK2) OUT2=1; else OUT2=0;
if(DEFAULT_OUTS&MASK3) OUT3=1; else OUT3=0;
if(DEFAULT_OUTS&MASK4) OUT4=1; else OUT4=0;
if(DEFAULT_OUTS&MASK5) OUT5=1; else OUT5=0;
if(DEFAULT_OUTS&MASK6) OUT6=1; else OUT6=0;
if(DEFAULT_OUTS&MASK7) OUT7=1; else OUT7=0;
if(DEFAULT_OUTS&MASK8) OUT8=1; else OUT8=0;

//LEER EL MODO DE FUNCIONAMIENTO
MODO=ReadEEPROM(0,0);



uart1_tx[0]=0x7E;     //ESTO ES SIEMPRE ASI
while(PIR1bits.RC1IF) clean=RCREG1;          //CLEAN UNDESIRED BYTES CAPTURED UART1
PIE1bits.RC1IE=1;//UART1 RX INTERRUPT ON
INTCON = 0b11100000; // HABILITAR INTERRUPCIONES, ACTIVADA LA DEL TMR0
T0CONbits.TMR0ON=1; //TIMER0 ON
T2CONbits.TMR2ON=1; //TIMER2 ON
T3CONbits.TMR3ON=1; //TIMER3 ON
USB_RTS=0; //MCP2200 YA PUEDE ENVIAR AL PIC
```

Figura 69: fase final de arranque.

5.3.8 – SOFTWARE DEL BUCLE DE PROGRAMA

Ahora dentro de un bucle infinito, el programa ha de repetir secuencialmente una serie de tareas que mantengan operativo a IO5-USB. Estas tareas serán desglosadas igualmente de manera secuencial en los sucesivos sub-apartados.

5.3.8.1 – ACTUALIZACIÓN DE ESTADO DE ENTRADAS

En esta primera parte del bucle de programa se actualizan las entradas, esto es: el nivel lógico que tienen las entradas pasa a la variable V_IN correspondiente. A partir de aquí se hará el análisis correspondiente de esta variable con respecto a la que representa el estado anterior de dicha entrada (V_ANT_IN) y entonces el valor de V_IN pasará a V_ANT_IN para repetir el procedimiento indefinidamente. En la Figura 70 se muestra el inicio del bucle de programa y la actualización de las entradas.

```
while(1) //MAIN PROGRAM LOOP
{
        //ACTUALIZACION DEL ESTADO DE LAS ENTRADAS
        if(IN1) V_IN1=0xFF; else V_IN1=0;
        if(IN2) V_IN2=0xFF; else V_IN2=0;
        if(IN3) V_IN3=0xFF; else V_IN3=0;
        if(IN4) V_IN4=0xFF; else V_IN4=0;
        if(IN5) V_IN5=0xFF; else V_IN5=0;
        if(IN6) V_IN6=0xFF; else V_IN6=0;
        if(IN7) V_IN7=0xFF; else V_IN7=0;
        if(IN8) V_IN8=0xFF; else V_IN8=0;
```

Figura 70: actualización de estado de entradas.

5.3.8.2 – GESTIÓN DE COMUNICACIÓN USB

Aunque los mensajes por vía USB son recogidos byte a byte en la rutina de gestión de interrupciones debe haber una rutina en el bucle de programa que analice la naturaleza de ese mensaje y cuando haya suficiente información se decida si el mensaje está completo (y a continuación se compruebe su integridad) o si hay algún problema durante su recepción (tarda demasiado en llegar o la cabecera y/o longitud del mismo son erróneos).

Para esta tarea existe una rutina en el bucle de programa que será convenientemente desglosada en los sucesivos sub-apartados. En la Figura 71 se muestra el código empleado para identificar un mensaje completo y en caso afirmativo calcular su CRC. Si el mensaje está incompleto y han pasado más de 2 segundos desde el último byte recibido, la función solicitada es desconocida o el CRC es erróneo el mensaje será descartado, lo cual es gestionado al final de esta subrutina. En caso contrario el mensaje será interpretado y obtendrá la contestación por parte de IO5-USB que proceda.

```
//GESTIÓN DE LA LLEGADA DE MENSAJES POR USB
if(n_uart1_rx>(char)0) //¿HA LLEGADO AL MENOS UN BYTE POR UART1?
{
        if((n_uart1_rx>(char)1)&&(n_uart1_rx==(uart1_rx[1]+(char)3)))
        //¿YA HA LLEGADO EL Nº DE BYTES QUE DICE LA LONGITUD RECIBIDA?
        {
                //CALCULO DE CRC
                clean2=(char)0;
                for(i=0;i<uart1_rx[1];i++) clean2+=uart1_rx[2+i];
                clean=0xFF-clean2; //CRC
                if(clean==(uart1_rx[uart1_rx[1]+(char)2])) //¿CRC CORRECTO?
                {
                        //INTERPRETAR MENSAJE
                        USB_RTS=1;
                        //MCP2200 YA NO PUEDE ENVIAR AL PIC, ESTO DEBE REHABILITARSE AL
                        //ACABAR DE CONTESTAR A LA PETICION
```

Figura 71: análisis de integridad de mensajes USB.

5.3.8.2.1 – GESTIÓN DE MODOS DE FUNCIONAMIENTO DE IO5-USB

Tanto la función de leer el modo de funcionamiento como la de modificarlo son siempre accesibles y devuelven como respuesta el mismo mensaje que se envió desde el PC. En la Figura 72 se muestra el software que cumple con este cometido. Se ha tenido en cuenta que RUN y STOP son sólo dos modos básicos y que en el futuro pueda haber otros con distintas funcionalidades.

```
                        if(uart1_rx[2]==(char)0)
                        //¿LEER O MODIFICAR EL MODO DE FUNCIONAMIENTO?
                        {
                                if(uart1_rx[3]==(char)254) UART1TX_LEER_MODO();
                                //¿LEER EL MODO DE FUNCIONAMIENTO?
                                else if((uart1_rx[3]==(char)0)||(uart1_rx[3]==(char)255))
                                //MODIFICAR MODO, DE MOMENTO SOLO HAY 0(RUN) Y
                                //255(STOP)
                                {
                                //SI SE PASA AL MODO RUN DESDE EL MODO RUN SE
                                //CONTESTA PERO NO SE HACE NADA
                                        WriteEEPROM(0,0,uart1_rx[3]);
                                        //SE GUARDA EN LA
                                        //EEPROM EL NUEVO MODO
                                        MODO=uart1_rx[3];
                                        UART1TX_MISMO_RX();
                                }
                                else UART1TX_MSJ_DESCONOCIDO();
                                //SE HA USADO UN MODO DESCONOCIDO, ENVIAR "MENSAJE
                                //DESCONOCIDO"
                        }
```

Figura 72: gestión de modos de funcionamiento.

5.3.8.2.2 – GESTIÓN DE ESTADO DE ENTRADAS Y SALIDAS

Desde USB siempre se puede solicitar el estado de las entradas y las salidas (el código correspondiente se expone en la Figura 73). Sin embargo, las salidas sólo se pueden modificar cuando IO5-USB está en modo STOP. En caso contrario se notifica que no es posible cambiar el estado de las salidas mientras se ejecuta el programa guardado en IO5-USB dado que se trata de una acción inconsistente con el modo de funcionamiento.

```
else if(uart1_rx[2]==(char)1) UART1TX_ESTADO_ACT_E_S();
//¿LEER ESTADO ACTUAL DE ENTRADAS Y SALIDAS?
else if(uart1_rx[2]==(char)2)
//¿MODIFICAR ESTADO ACTUAL DE LAS SALIDAS?
{
        clean=ReadEEPROM(0,0);
        //LEER EL MODO DE FUNCIONAMIENTO
        if (clean==(char)255)
        //MODO STOP, SE MODIFICAN LAS SALIDAS COMO SE HA
        //PEDIDO
        {
                if(uart1_rx[3]&MASK1) OUT1=1; else OUT1=0;
                if(uart1_rx[3]&MASK2) OUT2=1; else OUT2=0;
                if(uart1_rx[3]&MASK3) OUT3=1; else OUT3=0;
                if(uart1_rx[3]&MASK4) OUT4=1; else OUT4=0;
                if(uart1_rx[3]&MASK5) OUT5=1; else OUT5=0;
                if(uart1_rx[3]&MASK6) OUT6=1; else OUT6=0;
                if(uart1_rx[3]&MASK7) OUT7=1; else OUT7=0;
                if(uart1_rx[3]&MASK8) OUT8=1; else OUT8=0;
                UART1TX_MISMO_RX();
        }
        else if(clean==(char)0) UART1TX_RUN_NO_MODIF_S();
        //MODO RUN, NO SE PUEDEN MODIFICAR LAS SALIDAS DESDE EL
        //PC DE MANERA DIRECTA, SI SE AÑADE ALGUN MODO HAY QUE
        //METERLO AQUI
}
```

Figura 73: gestión de entradas y salidas.

5.3.8.2.3 – GESTIÓN DE PROGRAMACIÓN

Cuando se recibe un nuevo programa el sistema se reinicia tras haber guardado el programa en la EEPROM (el contador de programa del microcontrolador vuelve a la posición inicial en la memoria de programa). Esto obedece a la necesidad de actualizar la configuración de funcionamiento de las salidas. Es remarcable que la configuración de los contadores no cambia al introducir un programa, sólo se puede configurar una salida para ser conectada tras sobrepasar el umbral del contador pero la configuración del contador debe hacerse después. La Figura 74 muestra el código correspondiente a esta importante tarea de IO5-USB.

```
else if((uart1_rx[2]==(char)3)&&(uart1_rx[1]==(char)43))
//¿AÑADIR UN NUEVO PROGRAMA? EN ESTE CASO SE COMPRUEBA SI LA
//LONGITUD ES LA CORRECTA
{
        //GRABAR EL PROGRAMA, RESPONDER CON EL MISMO MENSAJE
        //RECIBIDO Y HACER RESET PARA ACTUALIZAR CONFIGURACIÓN
        for(i=1;i<(char)42;i++) //42 ES LA LONG. DEL MENSAJE
        WriteEEPROM((char)0,i,uart1_rx[i+(char)3]);
        UART1TX_MISMO_RX();
        Reset();
}
```

Figura 74: gestión de programación.

5.3.8.2.4 – GESTIÓN DEL VALOR DE LOS CONTADORES Y DEL NÚMERO DE SERIE DEL EQUIPO

Los ocho contadores pueden ser leídos y modificados en cualquier momento a través de USB, lo que se hace a través de un comando que devuelve el valor de todos los contadores y del número de serie del equipo. La opción de volver a cero un contador está implícita en la posibilidad de darle cualquier valor al contador dentro de su rango, lo cual se hace de manera individual para cada contador.

La modificación del número de serie del equipo se trata de una gestión propia de la empresa fabricante y no estará accesible para el usuario (el cual sólo podrá saber el número de serie del equipo pero no modificarlo ni siquiera saber que eso es posible). A través de ella es posible mantener la trazabilidad del equipo y teóricamente esa función sólo debe ser usada una vez, antes de vender cualquier unidad de IO5-USB. El código correspondiente a estas dos funciones se ilustra en la Figura 75.

```
else if(uart1_rx[2]==(char)4)//¿PROGRAMAR CONTADORES?
{
        if(uart1_rx[3]==(char)255)
        //¿GUARDAR CONFIGURACION EN EEPROM?
        {
                for(i=4;i<(char)29;i++)
                WriteEEPROM((char)0,i+(char)38,uart1_rx[i]);
                UART1TX_MISMO_RX();
                Reset();
        }
        else if (uart1_rx[3]==(char)254)
        {
                //ANTES DE LEER_CONTADORES() SE ESCRIBE EL
                //NUMERO DE SERIE EN EL MENSAJE A ENVIAR, ASI
                //NO SE LLAMA A UNA FUNCION DESDE OTRA
                //FUNCION
                uart1_tx[36]=ReadEEPROM(0x03,0xFC);
                uart1_tx[37]=ReadEEPROM(0x03,0xFD);
                uart1_tx[38]=ReadEEPROM(0x03,0xFE);
                uart1_tx[39]=ReadEEPROM(0x03,0xFF);
                LEER_CONTADORES();//LEER VALOR DE LOS
                                  //CONTADORES Y NUMERO DE
                                  //SERIE DE SEL16
        }
        else//¿MODIFICAR EL VALOR DE UN CONTADOR?
        {
                //COMPROBAR QUE CONTADOR ES, SI ES UN VALOR
                //INVALIDO DEVOLVER MENSAJE DESCONOCIDO
                if(uart1_rx[3]<(char)8)
                {
                        MODIFICA_VALOR_CONTADOR();
                        UART1TX_MISMO_RX();
                }
                else UART1TX_MSJ_DESCONOCIDO();
                //FUNCION DESCONOCIDA, ENVIAR "MENSAJE
                //DESCONOCIDO"
        }
}
else if(uart1_rx[2]==(char)255)//¿MODIFICAR EL NUMERO DE SERIE?
{
                if((uart1_rx[3]==0xFF) && (uart1_rx[4]==0xFF) &&
                (uart1_rx[9]==0xFF) && (uart1_rx[10]==0xFF) &&
                (uart1_rx[11]==0xFF))
                //¿TRAMA VALIDA?
        {
                WriteEEPROM(0x03,0xFC,uart1_rx[5]);
                WriteEEPROM(0x03,0xFD,uart1_rx[6]);
                WriteEEPROM(0x03,0xFE,uart1_rx[7]);
                WriteEEPROM(0x03,0xFF,uart1_rx[8]);

                UART1TX_MISMO_RX();
        }
        else UART1TX_MSJ_DESCONOCIDO();
        //FUNCION DESCONOCIDA, ENVIAR "MENSAJE DESCONOCIDO"
}
```

Figura 75: gestión de valor de contadores.

5.3.8.2.5 – GESTIÓN DE MENSAJES DESCARTADOS

Para el caso en el que un mensaje ha resultado descartado debe enviarse por USB una respuesta que permita conocer la naturaleza del descarte, esta funcionalidad está enfocada fundamentalmente a la acotación de problemas en la comunicación entre IO5-USB y el PC. La Figura 76 muestra el código encargado de esta tarea.

```
                                        else UART1TX_MSJ_DESCONOCIDO();
                                        //FUNCION DESCONOCIDA, ENVIAR "MENSAJE DESCONOCIDO"
                                        uart1_timeout=(char)255;
                                        n_uart1_rx=(char)0;
                                }
                                else //ENVIAR MENSAJE DE CRC INCORRECTO
                                {
                                        uart1_timeout=(char)255;
                                        n_uart1_rx=(char)0;
                                        UART1TX_CRC_ERROR();
                                }
                        }
                        else if (uart1_rx[0]!=(char)0x7E) //¿CABECERA INVALIDA?
                        {
                                uart1_timeout=(char)255;
                                n_uart1_rx=(char)0;
                        }
                        else if ((uart1_timeout!=(char)255)&&(uart1_timeout>=(char)2))
                        //¿TIMEOUT DE RECEPCION POR UART1?
                        {
                                uart1_timeout=(char)255;
                                n_uart1_rx=(char)0;
                                UART1TX_TIMEOUT(); //ENVIAR MENSAJE DE TIMEOUT
                        }
                }
```

Figura 76: gestión de mensajes descartados.

5.3.8.2.6 – GESTIÓN DE ENVÍO DE RESPUESTAS

Como está expuesto en el punto 5.2.2 puede haber problemas monitoreando el puerto virtual que crea MCP2200 según el sistema operativo empleado, su nivel de actualización y el software de monitoreo empleado. Para evitar esto, se emplea la rutina de envío de datos por USB mostrada en la Figura 77, la cual se limita a retardar el envío de cada byte hasta el punto donde no se han observado incidencias en ninguno de los sistemas operativos Windows probados con varios monitores de puertos serie.

```
if(PIR1bits.TMR1IF) //¿HA DESBORDADO EL TIMER1? HAY UN ENVÍO POR UART1 EN MARCHA
{
        T1CONbits.TMR1ON=0;//DETENER TIMER1
        PIR1bits.TMR1IF=0;//BAJAR FLAG DE DESBORDAMIENTO DEL TIMER1
        while(!PIR1bits.TX1IF);while(USB_CTS);//ESPERAR A QUE TODO ESTÉ LISTO PARA ENVIAR
        TXREG1=uart1_tx[n_uart1_tx];
        n_uart1_tx++;
        if(n_uart1_tx==(uart1_tx[1]+(char)3)) //¿FIN DE ENVIO?
        {
                TMR1H=0xFF;TMR1L=0xFF;//A PUNTO DE DESBORDAR CUANDO SE CONECTE, PARA
                                        //QUE EMPIECE PRONTO UN ENVIO
                USB_RTS=0; //MCP2200 YA PUEDE ENVIAR AL PIC DE NUEVO
        }
        else //ENVIO NO COMPLETADO, SEGUIR
        {
                TMR1H=0xF0;TMR1L=0x60;//2mS@64MHz, EL LIMITE ENTRE BYTES DE ENVIO DE
                                        //MCP2200 MEDIDO FUE DE 1,15mS
                T1CONbits.TMR1ON=1;//CONECTAR TIMER1, QUE CONTINUE EL ENVIO
        }
}
```

Figura 77: gestión de respuestas.

5.3.8.3 – FUNCIONAMIENTO DE LOS CONTADORES

La parte del código encargada de los contadores es voluminosa en relación al resto del código de IO5-USB ya que estos disponen de una gran flexibilidad, pudiendo configurar para un mismo contador las entradas, los flancos (incluyendo su sensibilidad) y la duración de los pulsos en el caso que corresponde a este modo, a lo que hay que añadir que se trata de ocho contadores idénticos (por tanto para explicar el código sólo es necesario explicar un contador dado que el código para el resto de los contadores es igual, sólo cambiando el número del contador en cuestión). En los siguiente sub-apartados se explica el código correspondiente a estas tareas.

5.3.8.3.1 – MODO CONTADOR DE FLANCOS

Para el modo contador de flancos se tiene que comprobar, usando las variables V_IN y V_ANT_IN correspondientes el tipo de flanco detectado, dependiendo esto también de si el modo de funcionamiento de los contadores de flancos es de alta o de baja sensibilidad. Con una cadena de instrucciones condicionales se comprueba el tipo de entrada que está encargada de actuar sobre un contador y el flanco correspondiente, lo cual está ilustrado en la Figura 78.

```
//CONTADORES
//CONTADOR1
i=CONFIG_CONT1_B1>>5;
j=CONFIG_CONT1_B1&MASK5;//SI ESTO ES DISTINTO DE 0 ESTA ACTIVADO EL MODO DE BAJA SENSIBILIDAD
                        //PARA LOS MODOS DE FLANCOS DEL CONTADOR1
if(i==(char)0)          //¿ACTIVADO EL CONTADOR1 POR FLANCO DE BAJADA?
{
        if((IN_CONT1==(char)0)&&(V_ANT_IN1!=(char)0)&&(V_IN1==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        //CONTAR SI FLANCO DE BAJADA EN ENTRADA 1 SI ES LA SELECCIONADA
        else if((IN_CONT1==(char)1)&&(V_ANT_IN2!=(char)0)&&(V_IN2==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        //CONTAR SI FLANCO DE BAJADA EN ENTRADA 2 SI ES LA SELECCIONADA Y ASI SUCESIVAMENTE
        else if((IN_CONT1==(char)2)&&(V_ANT_IN3!=(char)0)&&(V_IN3==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)3)&&(V_ANT_IN4!=(char)0)&&(V_IN4==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)4)&&(V_ANT_IN5!=(char)0)&&(V_IN5==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)5)&&(V_ANT_IN6!=(char)0)&&(V_IN6==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)6)&&(V_ANT_IN7!=(char)0)&&(V_IN7==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)7)&&(V_ANT_IN8!=(char)0)&&(V_IN8==(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
}
else if(i==(char)1)     //¿ACTIVADO EL CONTADOR1 POR FLANCO DE SUBIDA?
{
        if((IN_CONT1==(char)0)&&(V_ANT_IN1==(char)0)&&(V_IN1!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)1)&&(V_ANT_IN2==(char)0)&&(V_IN2!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)2)&&(V_ANT_IN3==(char)0)&&(V_IN3!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)3)&&(V_ANT_IN4==(char)0)&&(V_IN4!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)4)&&(V_ANT_IN5==(char)0)&&(V_IN5!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)5)&&(V_ANT_IN6==(char)0)&&(V_IN6!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)6)&&(V_ANT_IN7==(char)0)&&(V_IN7!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
        else if((IN_CONT1==(char)7)&&(V_ANT_IN8==(char)0)&&(V_IN8!=(char)0)) { if(j)
        BAJA_SENS_1=(char)0; else CONTADOR1++; }
}
```

Figura 78: modo contador de flancos.

5.3.8.3.2 – MODO CONTADOR DE PULSOS

Según el modo activado por programa, el contador correspondiente se incrementará cuando se presente un pulso de la polaridad adecuada en la entrada seleccionada, teniendo en cuenta el modo de sensibilidad que se haya empleado: si está en modo alta sensibilidad esa detección será inmediata (toma sólo el tiempo de la realización de un bucle de programa) mientras que si está en modo de baja sensibilidad dependerá del número de milisegundos de la configuración (de 0 a 1023 milisegundos). Las Figuras 79 y 80 muestran el código de esta tarea.

```
else if(i==(char)2)     //¿ACTIVADO EL CONTADOR1 COMO CONTADOR DE PULSOS?
{
        if(PIR2bits.TMR3IF)//¿HA PASADO 1MS?
        {
                if(IN_CONT1==(char)0)//SI LA ENTRADA SELECCIONADA ES LA 1
                {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN1!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN1==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                }
                else if(IN_CONT1==(char)1)//SI LA ENTRADA SELECCIONADA ES LA 2
                {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN2!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN2==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                }
                else if(IN_CONT1==(char)2)//SI LA ENTRADA SELECCIONADA ES LA 3
                {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN3!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN3==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                }
                else if(IN_CONT1==(char)3)//SI LA ENTRADA SELECCIONADA ES LA 4
                {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN4!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN4==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                }
```

Figura 79: modo contador de pulsos (parte 1/2).

```
                    else if(IN_CONT1==(char)4)//SI LA ENTRADA SELECCIONADA ES LA 5
                    {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN5!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN5==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                    }
                    else if(IN_CONT1==(char)5)//SI LA ENTRADA SELECCIONADA ES LA 6
                    {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN6!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN6==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                    }
                    else if(IN_CONT1==(char)6)//SI LA ENTRADA SELECCIONADA ES LA 7
                    {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN7!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN7==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                    }
                    else if(IN_CONT1==(char)7)//SI LA ENTRADA SELECCIONADA ES LA 8
                    {
                        if((PULSO_CONT1_OK==(char)255)&&(V_IN8!=(char)0)) n_MS_PULSO1++;
                        else if (V_IN8==(char)0)
                        {       //ENTRADA A 0 LOGICO
                                n_MS_PULSO1=(short long)0;
                                PULSO_CONT1_OK=(char)255;   //YA SE PERMITE CONTAR
                        }
                    }

                    if(n_MS_PULSO1>=UMBRAL_MS_PULSO1) //¿INCREMENTAR EL CONT. DE PULSOS?
                    {
                        CONTADOR1++;
                        n_MS_PULSO1=(short long)0;    //RESET DE LA CUENTA DE MS DE PULSOS
                        PULSO_CONT1_OK=(char)0;       //YA NO SE PERMITE CONTAR HASTA QUE
                                                      //VENGA UN FLANCO DE BAJADA, ESTO SE
                                                      //REACTIVA AL FINAL DEL PROGRAMA SI
                                                      //HAY UN FLANCO DE BAJADA
                    }
                }
            }
```

Figura 80: modo contador de pulsos (parte 2/2).

5.3.8.3.3 – TIMER DE PULSOS Y RESET DE CONTADORES

Si el contador está en modo contador de pulsos el contador se incrementará siempre que un pulso a nivel alto supere el umbral programado. El contador no volverá a tomar en cuenta otro pulso hasta que no detecte un nivel lógico bajo en la entrada que esté configurada para dicho contador.

Tras el código del timer de pulsos está el correspondiente al reset de contadores: el contador correspondiente volverá a cero si la entrada que se haya habilitado para ello está a nivel lógico alto. La Figura 81 muestra el código de estas tareas.

```
//SI HA DESBORDADO, COMO YA SE HA EVALUADO, HAY QUE VOLVER A ACTIVAR EL TIMER DE LOS PULSOS
if(PIR2bits.TMR3IF)
{
	T3CONbits.TMR3ON=0;//DETENER TIMER3
	PIR2bits.TMR3IF=0;//BAJAR FLAG DE DESBORDAMIENTO DEL TIMER3
	TMR3H=0xF8;TMR3L=0x58;//980uS@64MHz
	T3CONbits.TMR3ON=1;//DE NUEVO EN MARCHA EL TIMER3
}

//RESET DE CONTADORES
if(CONFIG_CONT1_B1&MASK4)//¿SE USA UNA ENTRADA PARA VOLVER EL CONTADOR1 A 0?
{
	if((IN_RESET_CONT1==(char)0)&&(V_IN1==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)1)&&(V_IN2==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)2)&&(V_IN3==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)3)&&(V_IN4==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)4)&&(V_IN5==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)5)&&(V_IN6==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)6)&&(V_IN7==(char)255)) CONTADOR1=(unsigned long)0;
	else if((IN_RESET_CONT1==(char)7)&&(V_IN8==(char)255)) CONTADOR1=(unsigned long)0;
}//IDEM CON EL RESTO DE CONTADORES
```

Figura 81: timer de pulsos y reset de contadores.

5.3.8.4 – GESTIÓN DE SALIDAS

Una vez actualizados los contadores hay que comprobar a qué nivel lógico corresponde poner cada salida. Como cada salida tiene tres formas de ser comandada (modo lógico AND, modo lógico OR y umbral del contador correspondiente a la entrada) las tres formas están redactadas de manera secuencial en el código mostrado.

Los algoritmos correspondientes a los modos lógicos son extensos dado que se comprueban varias variables al mismo tiempo. El algoritmo para decidir en qué nivel lógico ha de estar la salida se basa en dos comprobaciones para cada entrada de IO5-USB: la primera comprobación comprueba si la entrada correspondiente es relevante para dicha salida y si tiene el nivel lógico deseado y la segunda se ocupa de que el valor de la primera operación lógica no sea tomada en cuenta en el resto de operaciones para dicha salida dependiendo de si dicha entrada es relevante para la salida o no.

Para la primera salida y en el caso del algoritmo correspondiente al modo lógico AND, éste comprueba primeramente si coincide el valor lógico actual de la primera entrada con el bit correspondiente de las máscaras de nivel lógico alto y bajo. Esta comprobación se une a la comprobación posterior de si dicha entrada es relevante para este modo lógico ya que si no lo fuera por parte del análisis de la primera entrada el resultado sería positivo dado que en el modo lógico AND basta que no se cumpla una condición para que el resultado lógico sea falso.

Por ejemplo, si para accionar la salida 1 sólo se emplea la entrada 1 en nivel lógico bajo, la primera comprobación daría falso pero la segunda daría verdadero luego el resultado de ambas es verdadero y el resultado de la segunda comprobación es irrelevante dado que se trata de una unión lógica OR. Al comprobar la segunda salida la primera comprobación sería falsa para ambos niveles lógicos pero en la segunda comprobación se verificaría que dicha entrada no es relevante y por tanto el resultado sería positivo. Sucesivamente, el resultado sería positivo para las 8 entradas, dando lugar a un resultado positivo hacia la salida.

Por otra parte, para la salida 1 en el caso del modo lógico OR la primera comprobación es análoga al modo lógico AND pero la segunda cambia dado que en este modo lógico si una entrada no es relevante su resultado debe ser negativo en vez de positivo. El proceso es análogo teniendo en cuenta las diferencias intrínsecas entre ambos sistemas lógicos y se muestra el código correspondiente a la salida 1 en la Figura 82.

```
if(MODO==(char)0)//FUNCIONAMIENTO DEL PROGRAMA GUARDADO EN EEPROM SI ESTA EN MODO RUN
{
        //SALIDA1
        clean=(char)0;//CON ESTA VARIABLE SE DETERMINA AL FINAL SI HA DE INVERTIRSE EL ESTADO
                //PREDETERMINADO DE LA SALIDA1 (SI ESTÁ A 255) O NO (SI ESTÁ A 0)
        if(CONFIG_OUT1==(char)0)        //¿SALIDA 1 MODO AND?
        {
                if(((V_IN1&&(OUT1_MASK_INPUTS&MASK1))||(!V_IN1&&(OUT1_MASK_INPUTS_N&
                MASK1))||(!(OUT1_MASK_INPUTS&MASK1)&&!(OUT1_MASK_INPUTS_N&
                MASK1)))&&((V_IN2&&(OUT1_MASK_INPUTS&MASK2))||(!V_IN2&&
                (OUT1_MASK_INPUTS_N&MASK2))||(!(OUT1_MASK_INPUTS&MASK2)&&!
                (OUT1_MASK_INPUTS_N&MASK2)))&&((V_IN3&&(OUT1_MASK_INPUTS&MASK3))||
                (!V_IN3&&(OUT1_MASK_INPUTS_N&MASK3))||(!(OUT1_MASK_INPUTS&MASK3)&&!
                (OUT1_MASK_INPUTS_N&MASK3)))&&((V_IN4&&(OUT1_MASK_INPUTS&MASK4))||
                (!V_IN4&&(OUT1_MASK_INPUTS_N&MASK4))||(!(OUT1_MASK_INPUTS&MASK4)&&!
                (OUT1_MASK_INPUTS_N&MASK4)))&&((V_IN5&&(OUT1_MASK_INPUTS&MASK5))||
                (!V_IN5&&(OUT1_MASK_INPUTS_N&MASK5))||(!(OUT1_MASK_INPUTS&MASK5)&&!
                (OUT1_MASK_INPUTS_N&MASK5)))&&((V_IN6&&(OUT1_MASK_INPUTS&MASK6))||
                (!V_IN6&&(OUT1_MASK_INPUTS_N&MASK6))||(!(OUT1_MASK_INPUTS&MASK6)&&!
                (OUT1_MASK_INPUTS_N&MASK6)))&&((V_IN7&&(OUT1_MASK_INPUTS&MASK7))||
                (!V_IN7&&(OUT1_MASK_INPUTS_N&MASK7))||(!(OUT1_MASK_INPUTS&MASK7)&&!
                (OUT1_MASK_INPUTS_N&MASK7)))&&((V_IN8&&(OUT1_MASK_INPUTS&MASK8))||
                (!V_IN8&&(OUT1_MASK_INPUTS_N&MASK8))||(!(OUT1_MASK_INPUTS&MASK8)&&!
                (OUT1_MASK_INPUTS_N&MASK8)))) clean=(char)255;
        }
        else if(CONFIG_OUT1==(char)1)  //¿SALIDA 1 MODO OR?
        {
                if(((V_IN1&&(OUT1_MASK_INPUTS&MASK1))||(!V_IN1&&(OUT1_MASK_INPUTS_N&
                MASK1)))||((V_IN2&&(OUT1_MASK_INPUTS&MASK2))||(!V_IN2&&
                (OUT1_MASK_INPUTS_N& MASK2)))||((V_IN3&&(OUT1_MASK_INPUTS&
                MASK3))||(!V_IN3&& (OUT1_MASK_INPUTS_N&MASK3)))||
                ((V_IN4&&(OUT1_MASK_INPUTS& MASK4))||(!V_IN4&&(OUT1_MASK_INPUTS_N&
                MASK4)))||((V_IN5&&(OUT1_MASK_INPUTS&MASK5))||(!V_IN5&&
                (OUT1_MASK_INPUTS_N&MASK5)))||((V_IN6&&(OUT1_MASK_INPUTS&
                MASK6))||(!V_IN6&&(OUT1_MASK_INPUTS_N&MASK6)))||
                ((V_IN7&&(OUT1_MASK_INPUTS& MASK7))||(!V_IN7&&(OUT1_MASK_INPUTS_N&
                MASK7)))||((V_IN8&&(OUT1_MASK_INPUTS&MASK8))||(!V_IN8&&
                (OUT1_MASK_INPUTS_N&MASK8)))) clean=(char)255;
        }
        else if((CONFIG_OUT1<(char)3)&&(CONTADOR1>=UMBRAL_CONT1)) clean=(char)255;
        //¿SALIDA 1 USANDO ALGUN CONTADOR Y SE HA SOBREPASADO EL UMBRAL PROGRAMADO DEL
        //CONTADOR?
        if(clean==(char)255)//INVERTIR ESTADO ORIGINAL DE LA SALIDA
        {
                if(DEFAULT_OUTS&MASK1) OUT1=0; else OUT1=1;
        }
        else        //MANTENER ESTADO ORIGINAL DE LA SALIDA
        {
                if(DEFAULT_OUTS&MASK1) OUT1=1; else OUT1=0;
        }
```

Figura 82: gestión de salidas.

5.3.8.5 – ESTADO DE EUSART Y ACTUALIZACIÓN DE ESTADO DE ENTRADAS

El microcontrolador PIC18LF26K22 cuenta con un buffer de dos bytes de recepción de datos a través de EUSART. Si estando esos dos bytes ocupados llega alguno más dicha EUSART se bloqueará indicando OVERRUN, lo que significa que esa EUSART no recibirá más datos aunque se vacíen los buffers. Por tanto es conveniente (y casi obligatorio) monitorear el estado de las EUSART para solucionar un eventual desbordamiento de las mismas, que aunque se gestionen las recepciones a través de interrupciones no son raros los casos donde una interferencia en la línea de recepción o algún error en el cálculo de la velocidad de configuración de la EUSART acabe por desbordarla. El código expuesto en la Figura 83 se encarga de gestionar estas incidencias.

```
                    //¿UART1RX DESBORDADO?
                    if(RCSTA1bits.OERR) //PONER A 0 Y DESPUES A 1 CREN
                    {
                              while(PIR1bits.RC1IF) clean=RCREG1;//CLEAN UNDESIRED BYTES CAPTURED UART1
                              uart1_timeout=(char)255;
                              n_uart1_rx=(char)0;
                              RCSTA1bits.CREN=0;
                              RCSTA1bits.CREN=1;
                    }

                    //UNA VEZ TODO HECHO, LOS ESTADOS "ACTUALES" DE LAS ENTRADAS PASAN A SER LOS "ANTERIORES"
                    V_ANT_IN1=V_IN1;
                    V_ANT_IN2=V_IN2;
                    V_ANT_IN3=V_IN3;
                    V_ANT_IN4=V_IN4;
                    V_ANT_IN5=V_IN5;
                    V_ANT_IN6=V_IN6;
                    V_ANT_IN7=V_IN7;
                    V_ANT_IN8=V_IN8;

          }//WHILE(1)

}//MAIN
```

Figura 83: estado de EUSART y de entradas.

5.4 – RESUMEN DE CARACTERÍSTICAS DEL SOFTWARE DISEÑADO

Una vez finalizado el software sólo queda compilarlo para crear el correspondiente archivo HEX formado por mnemónicos para programar el PIC. En el bucle de programa el tiempo que transcurre entre bucle y bucle es variable aunque se estima que en ningún caso alcanza medio milisegundo, estimación basada en múltiples pruebas de simulación usando MPLAB SIM monitoreando los tiempos con la función STOPWATCH.

El número total de líneas del código en C es de 2454, en la Figura 84 se muestra el tiempo que toma el bucle de programa cuando tanto las salidas como los contadores están desactivados y el uso de recursos del PIC18LF26K22 con el programa diseñado.

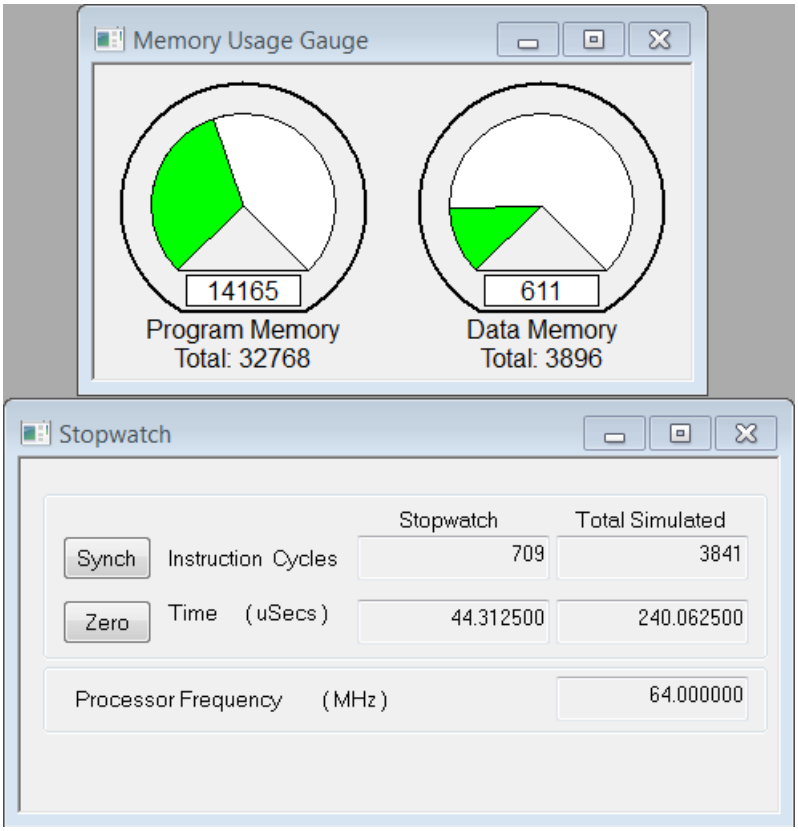


Figura 84: uso de memoria del programa y tiempo de bucle de programa.

CAPÍTULO 6 – PRUEBAS DE FUNCIONAMIENTO DE SEL16 Y FLX01

6.1 – DESCRIPCIÓN DEL MÉTODO DE PRUEBAS

Una vez fabricadas SEL16 y FLX01 deben pasar una serie de pruebas que garanticen su operatividad. Estas pruebas además sirven como realimentación del proceso de diseño, dado que las desviaciones que se detecten entre lo calculado y el sistema real permiten mejorar la metodología de diseño.

Todas las pruebas del hardware relacionado con IO5-USB se harán usando el software correspondiente al mismo mientras que las pruebas del hardware ajeno a IO5-USB (comunicación a través de la red de telefonía móvil y a través de RS-422/485) se harán usando otro método que permita probar su operatividad.

A nivel de hardware se hará hincapié en el consumo eléctrico del equipo y la comparación entre el rendimiento esperado y el real. Para FLX01 bastará una inspección visual para verificar su funcionamiento (dado que es sólo una PCB compuesta de LEDs) y para el resto del hardware habrá que comprobar su funcionamiento usando el software diseñado para gobernar el equipo.

Para el software se probarán todas las capacidades de IO5-USB mientras que para el hardware ajeno a IO5-USB se podrá diseñar otro software que permita probar su utilidad, lo cual será empleado para probar el TELIT GL865 QUAD dado que para probar la comunicación a través de RS-422/485 bastará con seleccionar a través de los terminales de selección con soldadura que en vez de USB se empleará dicho sistema para la comunicación y por tanto podrá ser probado con el software de IO5-USB (digamos que el producto pasaría a ser IO5-RS-422/485).

6.2 – PRUEBA DEL CIRCUITO DE ALIMENTACIÓN ELÉCTRICA

Para comprobar el circuito de alimentación eléctrica se introducirá una tensión admisible por diseño en la entrada (hasta 60 voltios de corriente continua de máximo y el mínimo real se comprobará cuán por debajo de 10 voltios se encuentra) y se medirán las tensiones y corrientes de cada salida y la corriente de entrada al circuito. De esta manera se podrá calcular el rendimiento, el consumo estacionario del equipo (programado con IO5-USB en modo RUN y al mismo tiempo con el TELIT GL-865 QUAD conectado) y la calidad de la regulación de las salidas. El esquema de las mediciones es el mostrado en la Figura 85.

En esta figura se muestra la tabla con las mediciones realizadas. La corriente de salida en la salida de 3,3V es de unos 50 miliamperios y en la salida de 3,9V unos 150 miliamperios. Dado que esas corrientes apenas varían sin importar la tensión de entrada al circuito, se han tomado como fijas para el cálculo del rendimiento.

Tensión entrada (V)	Corriente entrada (A)	Tensión salida 3,3V (V)	Tensión salida 3,9V (V)	Rendimiento (%)
10	0,09	3,31	3,92	85,51
24	0,035	3,31	3,91	83,99
48	0,017	3,32	3,92	81,12
60	0,013	3,3	3,92	80,26

Figura 85: mediciones del sistema de alimentación para IO5-USB.

El rendimiento se ha calculado multiplicando la tensión de cada salida por su corriente de salida correspondiente y sumándolas, obteniendo la potencia total de salida del circuito. Si se divide esta potencia entre la potencia de entrada del circuito (tensión de entrada multiplicada por corriente de entrada) se obtiene el rendimiento en tanto por uno. Por motivos visuales el rendimiento se ha representado en tanto por cien.

Se puede apreciar que la regulación es bastante buena (las tensiones de salida apenas varían con el cambio de tensión de entrada) y aunque el rendimiento baja conforme aumenta la tensión de entrada se mantiene en unos niveles buenos dada la corriente de carga (que es relativamente pequeña en comparación con la capacidad del circuito). En el peor de los casos se ha obtenido un rendimiento superior al 80% lo que hace al equipo muy eficiente energéticamente.

6.3 – PRUEBA DEL CIRCUITO DE PROGRAMACIÓN DEL MICROCONTROLADOR

Al conectar las conexiones de programación del PIC de IO5-USB debe ser posible tanto programar como depurar el software del PIC a través del MICROCHIP TECHNOLOGY MPLAB ICD 3. Previamente hay que comprobar que no esté conectado a través de las conexiones de soldadura el TELIT GL-865 QUAD dado que podría interferir en la programación. Tras una serie de pruebas se llega a la conclusión de que dicho sistema funciona correctamente y como demostración se adjunta la Figura 86 mostrando la programación del PIC de IO5-USB con el software diseñado para su funcionamiento.

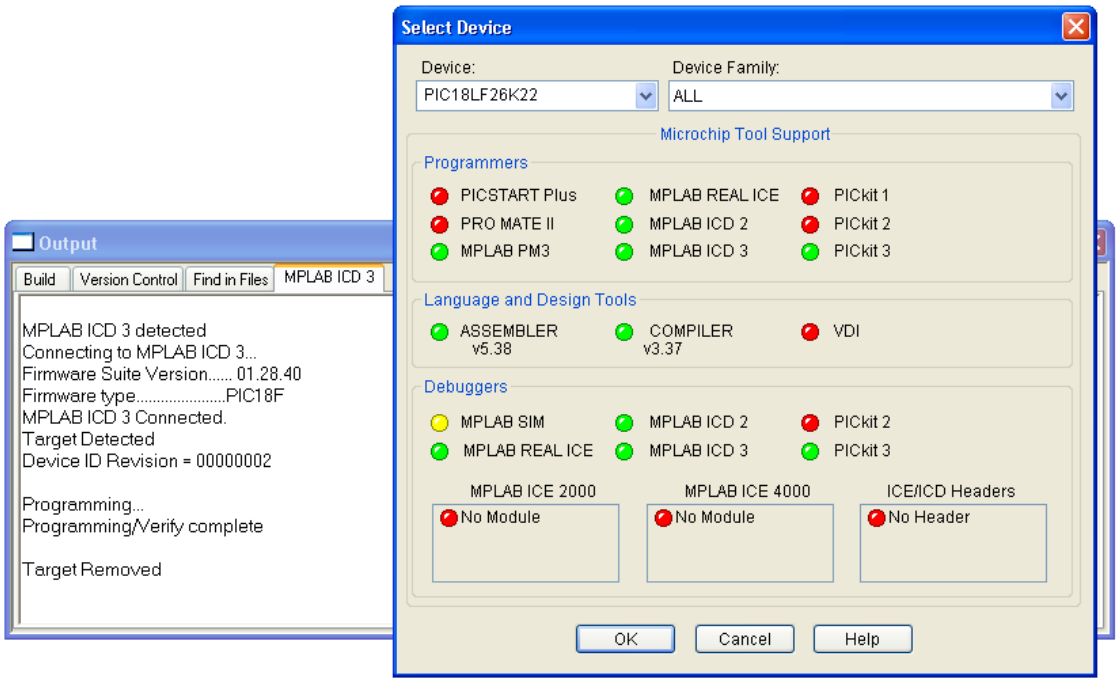


Figura 86: prueba del sistema ICSP implementado en SEL16.

6.4 – PRUEBA DE COMUNICACIÓN USB

A continuación se prueba con la utilidad de configuración de MICROCHIP TECHNOLOGY para el MCP2200 si hay comunicación efectiva USB tal como se muestra en la Figura 87. Con esta aplicación se pueden personalizar los descriptores para que cuando IO5-USB se conecte a un ordenador muestre el tipo de dispositivo conectado.

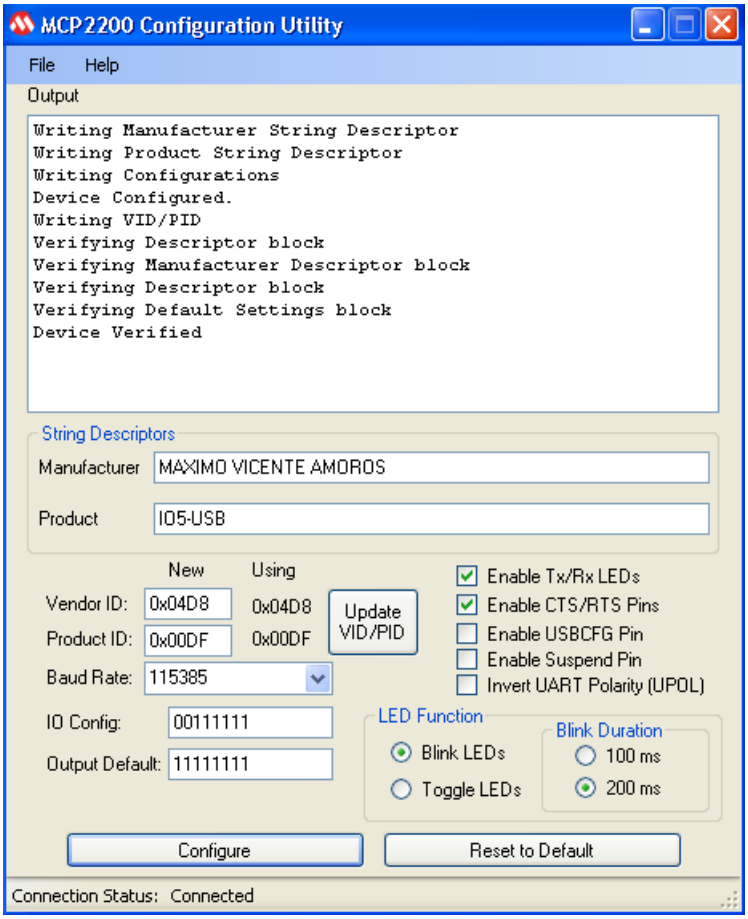


Figura 87: configuración de descriptores USB.

6.5 – FUNCIONAMIENTO DE FLX01

FLX01 es la PCB más sencilla y también la más fácil de comprobar: basta con echar un vistazo al funcionamiento de los LEDs. Como se aprecia en la Figura 88 los LEDs funcionan correctamente aunque es complicado tomar imágenes con varios LEDs encendidos dado que se trata de LEDs de alta luminosidad y hacen difícil interpretar en imágenes cuales están encendidos.

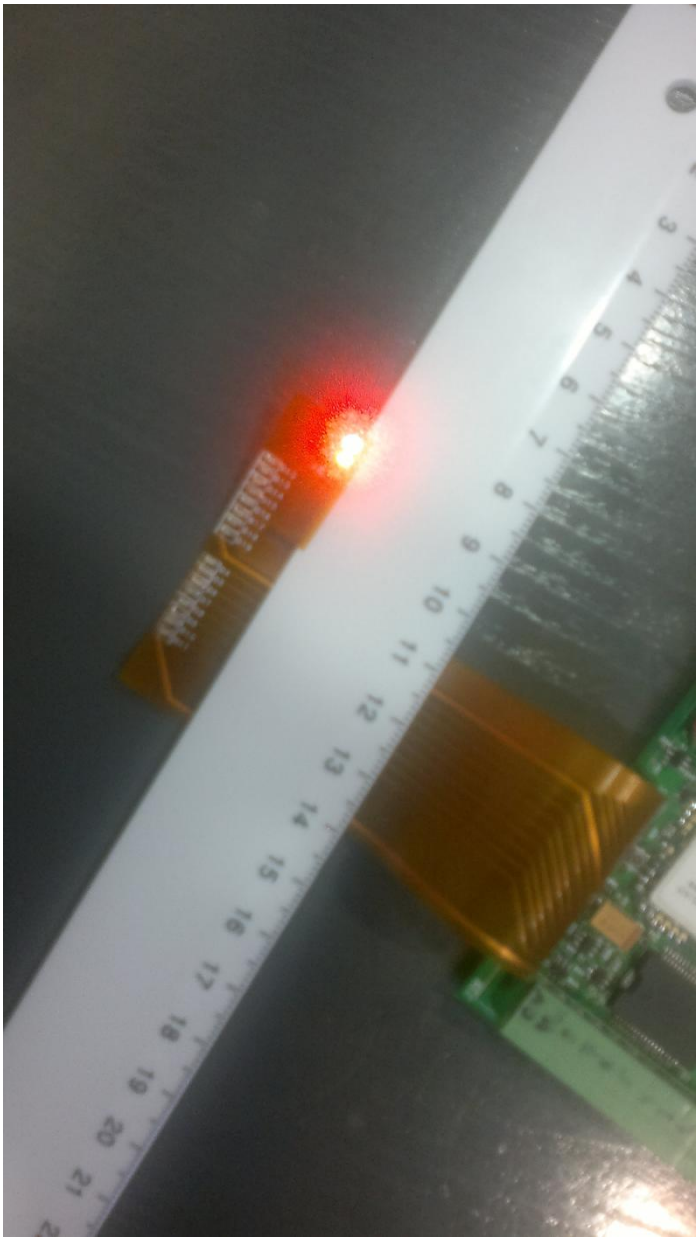


Figura 88: imagen de FLX01 en funcionamiento.

6.6 – PRUEBAS DEL HARDWARE CON EL SOFTWARE DE IO5-USB

En los siguientes sub-apartados se detallarán las pruebas ejecutadas para comprobar la funcionalidad del sistema. Aunque se han realizado muchas pruebas se documentan sólo las más ilustrativas que permiten conocer la naturaleza de IO5-USB sin caer en la redundancia dado que hay muchas formas de usar el equipo.

Para estas pruebas se empleará como interfaz de comunicación en el que sería el ordenador de gestión de IO5-USB el terminal X-CTU versión 5.2.8.6 de DIGI, en un ordenador con Windows XP Service Pack 3. Este terminal es amigable para la transmisión de datos en formato hexadecimal por lo que resulta práctico para estos fines, además de ser gratuito (bibliografía **[7]**). En la consola de X-CTU los datos que se envían desde el PC aparecen en color azul y los que se reciben en color rojo usando sólo dos cifras con un espacio entre byte y byte en configuración hexadecimal, que será la usada en las pruebas), esto permitirá seguir en las figuras correspondientes la comunicación entre el IO5-USB y el PC (es decir, el usuario final).

6.6.1 – PROGRAMACIÓN DE IO5-USB

Para hacer pruebas se va a programar en IO5-USB un modo que permita explotar todas las capacidades del sistema. Para ello primero se comprobará si el equipo está en modo STOP. Después se programarán los contadores para poder probarlos con las siguientes características:

CONTADOR 1: entrada 2, modo flanco de bajada, entrada 8 de reset, alta sensibilidad.

CONTADOR 2: desconectado.

CONTADOR 3: entrada 3, modo pulso mayor de cinco segundos, entrada 7 de reset.

CONTADOR 4: desconectado.

CONTADOR 5: entrada 5, modo flanco de subida, entrada 8 de reset, alta sensibilidad.

CONTADOR 6: entrada 8, modo pulso mayor de un segundo, sin entrada de reset.

CONTADOR 7: desconectado.

CONTADOR 8: desconectado.

Una vez programados los contadores sólo queda programar las salidas. Primero se decide el estado predeterminado de las salidas y después cuáles cambiarán de estado cuando se cumpla una de las tres condiciones posibles para ello: una combinación de entradas evaluadas mediante la función lógica AND, ídem con OR o si la cuenta del contador correspondiente a la salida que se desea cambiar sobrepasa determinado umbral. Para las pruebas se empleará el siguiente programa:

SALIDA 1: entrada 1 AND entrada 2 AND entrada 3.

SALIDA 2: entrada 1 OR entrada 3 OR entrada 4 negada.

SALIDA 3: contador 3 mayor o igual a 5.

SALIDA 4: entrada 2.

SALIDA 5: entrada 1 AND entrada 5 negada.

SALIDA 6: contador 6 mayor o igual a 10.

SALIDA 7: entrada 2 negada AND entrada 3 negada.

SALIDA 8: desactivada (permanece siempre en el estado predeterminado).

Una vez diseñado el programa se procede a traducirlo al idioma de IO5-USB, esto es, formato hexadecimal usando las reglas explicadas en el diseño del software. Hecho esto, se procede primeramente a comprobar si IO5-USB está en modo STOP, después se envía la programación de los contadores y después la de las salidas. En la Figura 89 se recogen estas acciones.

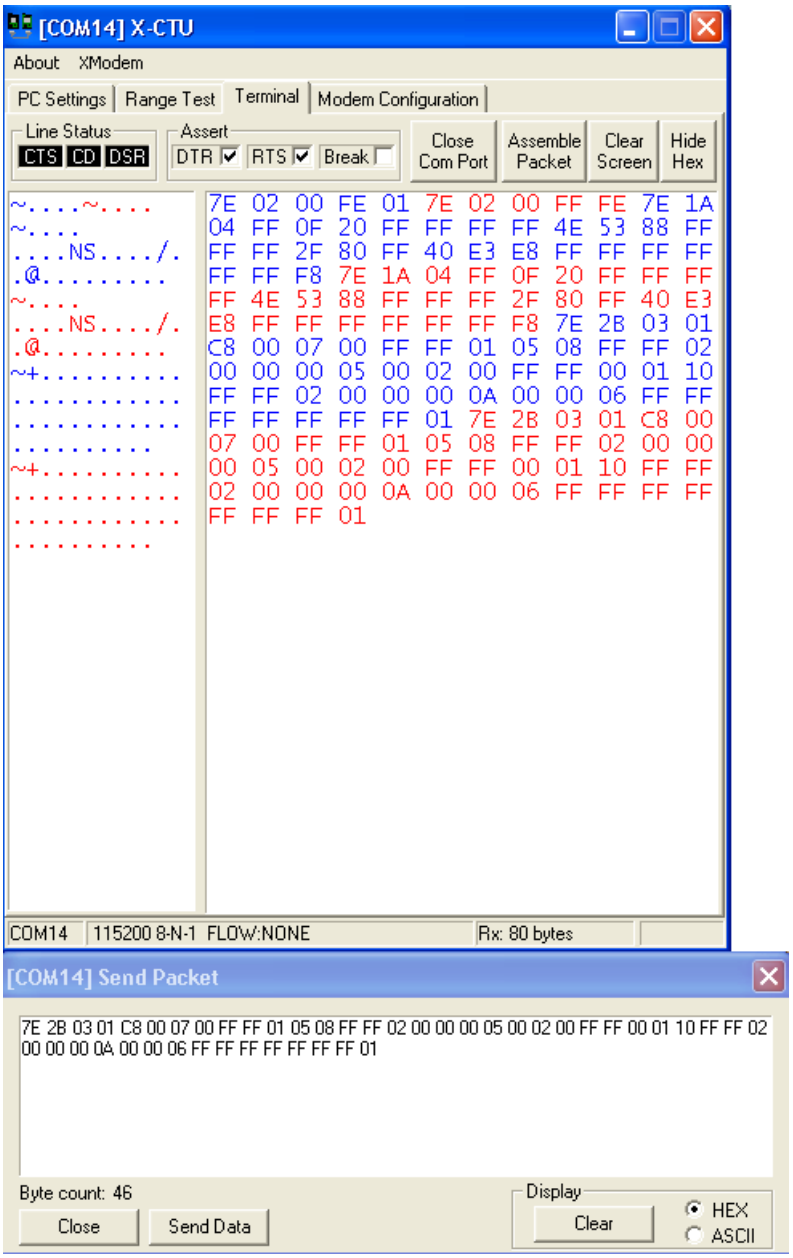


Figura 89: programación de IO5-USB.

La primera respuesta de IO5-USB confirma que el está en modo STOP y para cada programación la respuesta correcta debe ser el mismo mensaje enviado desde el PC, como confirmación de la integridad del programa recibido, lo que se comprueba que funcionó bien. Llegados a este punto, en modo depuración del PIC18LF26K22 de IO5-USB se podría detener el programa y comprobar en la EEPROM el programa enviado y su distribución en dicha memoria.

6.6.2 – MODO STOP

Se tiene la certeza de que el sistema está en modo STOP y recién programado. A continuación se comprueba el estado actual de las entradas y salidas y el valor actual de los contadores. Desde X-CTU el resultado de dichas operaciones es el mostrado en la Figura 90.

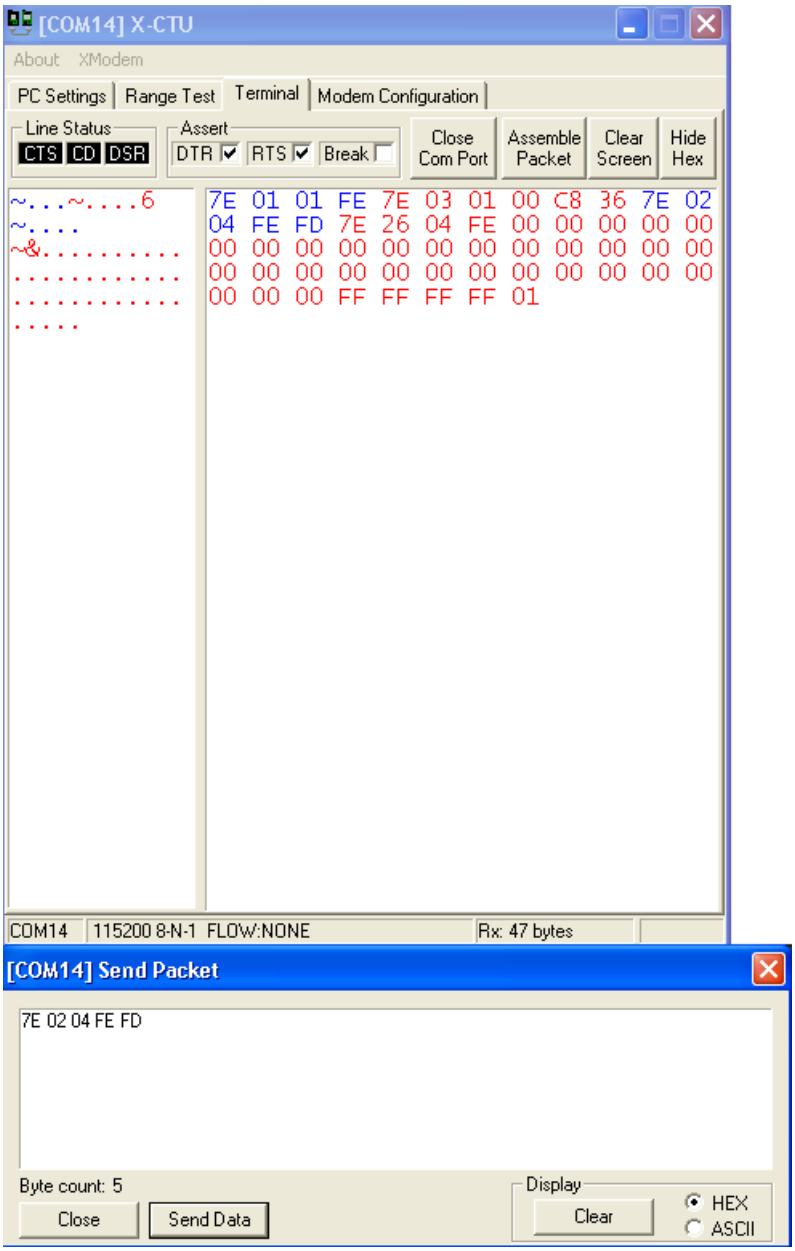


Figura 90: comprobación de E/S en modo STOP de IO5-USB.

La primera respuesta indica que las entradas están todas a nivel lógico bajo mientras que las salidas obedecen al número 0xC8, que en binario es 11001000, lo que significa que las salidas 8, 7 y 4 están a nivel lógico alto, esto es precisamente el estado predeterminado de las salidas que se ha programado.

Aunque se cambiaran las entradas las salidas nunca cambiarían de nivel lógico mientras IO5-USB permanezca en modo STOP. La única manera de cambiar las salidas en modo STOP es especificarlo a través del comando 0x02, que está pensado precisamente para gobernar las salidas manualmente a través del PC siempre y cuando IO5-USB esté en modo STOP.

6.6.3 – MODO RUN

A continuación se ensayará el modo RUN, en el cual las salidas son susceptibles de cambiar de acuerdo a las reglas que se hayan programado de manera automática. En los sucesivos sub-apartados se documentan los distintos modos de funcionamiento de las salidas.

6.6.3.1 – MODOS LÓGICOS

Para probar los modos lógicos se han hecho varias pruebas modificando aleatoriamente el nivel lógico de las entradas para posteriormente comprobar si las salidas corresponden con las operaciones lógicas programadas según el programa del apartado 6.6.1. En la Figura 91 se puede observar una muestra de estas pruebas tras configurar IO5-USB en modo RUN (primer comando empleado).

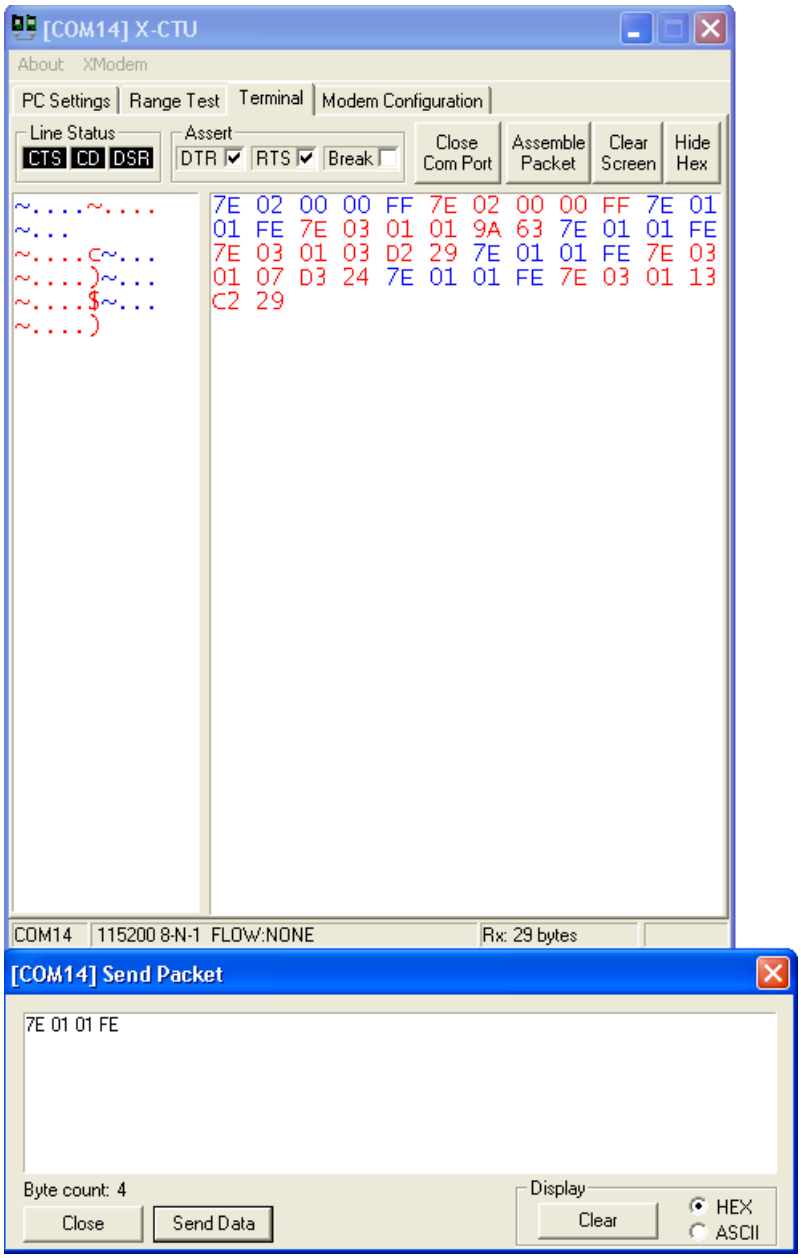


Figura 91: prueba de modos lógicos de IO5-USB.

Se solicita varias veces el estado de las entradas y las salidas para contrastar los resultados, para excitar con un nivel lógico alto las entradas se emplearon múltiples tensiones dentro del rango admisible de las tensiones de entrada. En el primer caso sólo está a nivel lógico alto la primera entrada (correspondiente a 0x01) mientras que las salidas a nivel lógico alto son la 8, 5, 4 y 2 (correspondiente a 0x9A). En el segundo caso se activan las dos primeras entradas, obteniendo a nivel lógico alto las salidas 8, 7, 5 y 2 (ahora pasa a cumplirse la función lógica correspondiente a las salidas 7 y 4, dejando de cumplirse la de la salida 5).

Para el tercer caso se activa una salida más (estando las tres primeras a nivel lógico alto) y en ese caso el byte correspondiente a las salidas (0xD3) indica que las únicas salidas a nivel bajo son las salidas 3, 4 y 6. En el cuarto caso están activas las entradas 5, 2 y 1, dando como resultado que estén activas a nivel alto las salidas 8, 7 y 2. De esta manera se ha comprobado que se han respetado las operaciones lógicas de todos los modos: cuando alguna se ha cumplido ha invertido el nivel lógico predeterminado de dicha salida (el cual era nivel lógico bajo en todas las salidas salvo la 4, la 7 y la 8).

6.6.3.2 – CONTADORES DE FLANCOS Y PULSOS

Para esta funcionalidad de IO5-USB se han hecho una serie de pruebas de acuerdo al programa del punto 6.6.1. Antes de hacer estas pruebas es conveniente devolver los contadores 1 y 3 al inicio de su cuenta ya que al probar los modos lógicos estos contadores se habrán incrementado ya que usan las entradas 2 y 3 respectivamente. Para esto se emplea la entrada 8, cuya finalidad es la de devolver a 0 los contadores 1, 3 y 5 aunque esto incrementará el contador 6 (que está configurado en modo contador de pulsos de más de un segundo).

Por tanto inicialmente se harán dos pulsos de nivel lógico alto de más de un segundo de duración en la entrada 8 con el doble propósito de devolver los contadores 1 y 3 a 0 y al mismo tiempo incrementar el contador 6. Una vez hecho esto se hace una comprobación del valor de los contadores, donde se comprueba que todos son cero (los cuatro bytes con valor 0xFF previos al CRC de cada mensaje corresponden al número de serie del equipo, el cual está configurado con ese número por defecto) salvo el contador 6 que tiene el número 2 en su cuenta.

Posteriormente se procede a ensayar el funcionamiento del resto de contadores que se han programado, aplicando manualmente una tensión de 12 voltios en la entrada en cuestión para analizar el resultado. Este método permitirá apreciar el efecto rebote y la necesidad del modo de baja sensibilidad para esos casos. La Figura 92 corresponde con las pruebas de los contadores.

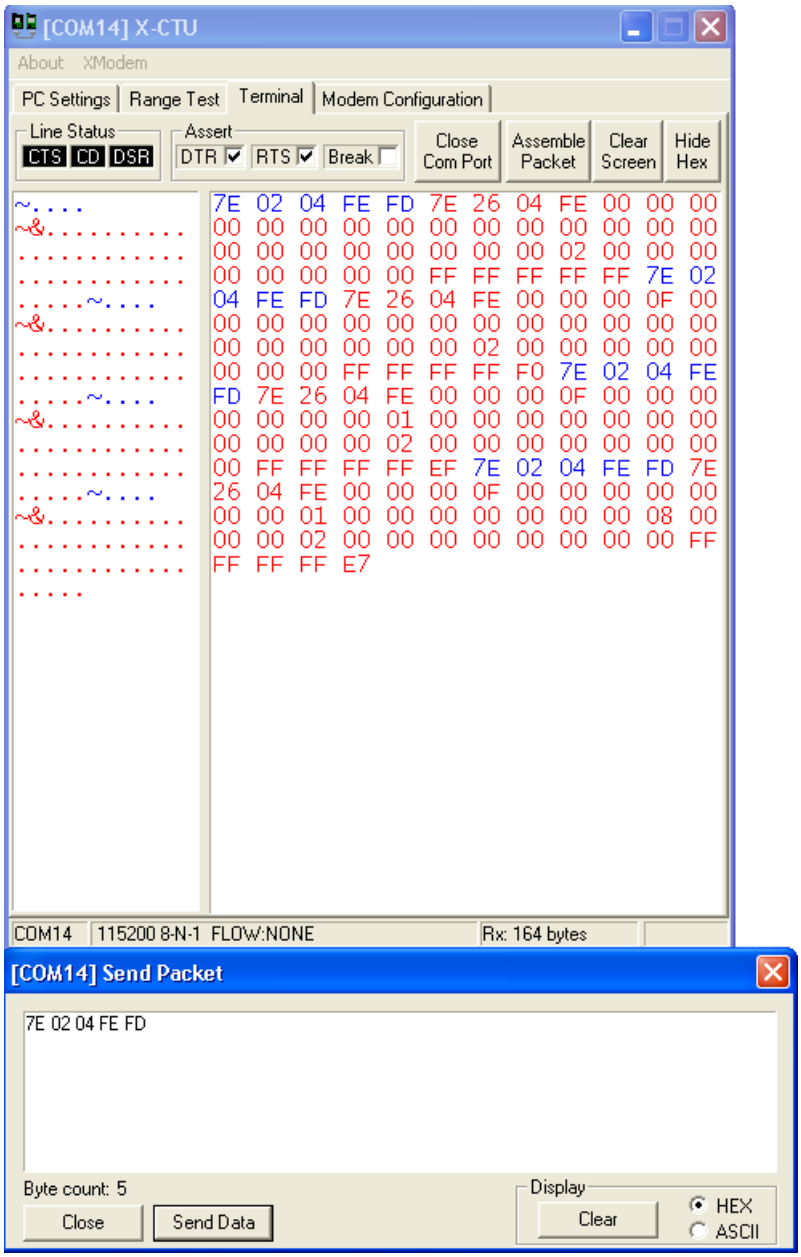


Figura 92: prueba de contadores de IO5-USB.

Lo primero que se ha hecho es hacer tres contactos en la entrada 2 con el cable a 12 voltios. Como el contador 1 está configurado como flanco de bajada en modo de alta sensibilidad, debe haber contado dichos contactos luego lo siguiente es solicitar a IO5-USB el estado de los contadores. Se aprecia que el primer contador está a 0x0F, esto es, 15 en formato decimal. Tres contactos han derivado en contar 15 (dado que han sido contactos manuales es normal que haya ocurrido esto por haber un contacto imperfecto entre cable con tensión y la entrada). Si se hubiera usado el modo de baja sensibilidad configurado por ejemplo con 200 milisegundos esto habría sido evitado y la cuenta sería 3 como se desearía.

A continuación se mantiene el cable en la entrada 3 durante 15 segundos y se procede de nuevo a comprobar el valor de los contadores. Entonces se aprecia que el contador 3 se ha incrementado para indicar una cuenta de 1. Como prueba final se hace un contacto a la entrada 5, correspondiente al contador 5 que está configurado como contador en flanco de subida en modo de alta sensibilidad y al comprobar de nuevo los contadores dicho contador indica una cuenta de 8, lo que pone de manifiesto el peligro de usar los contadores en modo de alta sensibilidad (este modo debe ser usado sólo en sistemas que gobiernan perfectamente y sin rebotes el nivel de tensión en la entrada de IO5-USB).

6.7 – PRUEBAS DEL RESTO DEL HARDWARE NO UTILIZADO POR IO5-USB

Como SEL16 y FLX01 son PCBs pensadas para ampliar las capacidades de IO5-USB u ofrecer otras soluciones se hace necesario comprobar la funcionalidad de dichas capacidades, las cuales son los sistemas de comunicación alternativos.

6.7.1 – SISTEMA DE COMUNICACIÓN GSM/3G

La segunda EUSART del PIC empleado se usa exclusivamente para programar el PIC y para comunicarlo con TELIT GL-865 QUAD. Para probar sus capacidades conviene usar una consola para comunicarse con dicho IC de interfaz con la red de telefonía móvil y poder comprobar que es funcional el circuito anexo a él, esto es, el sistema de la tarjeta SIM, el sistema de alimentación eléctrica, la conexión EUSART con el PIC a través del divisor resistivo y la conexión con la antena.

Por tanto para probar esto se programará el PIC para que haga de puente entre la consola que se empleará del PC (que será HyperTerminal dado que X-CTU no es práctico para la comunicación de comandos ASCII como sí lo es HyperTerminal), esto es: todo lo recibido a través de EUSART1 (que vendrá del USB desde el PC) pasará a EUSART2 (hacia TELIT) y viceversa. De esta forma habrá, desde el punto de vista del PC, una conexión tipo serie entre el PC y TELIT. Hecho esto, se harán algunas pruebas con una tarjeta SIM para comprobar el funcionalidad de TELIT (es decir, si se conecta a la red y si es capaz de enviar y recibir datos a través de ella). En la Figura 93 se muestran las pruebas ejecutadas.

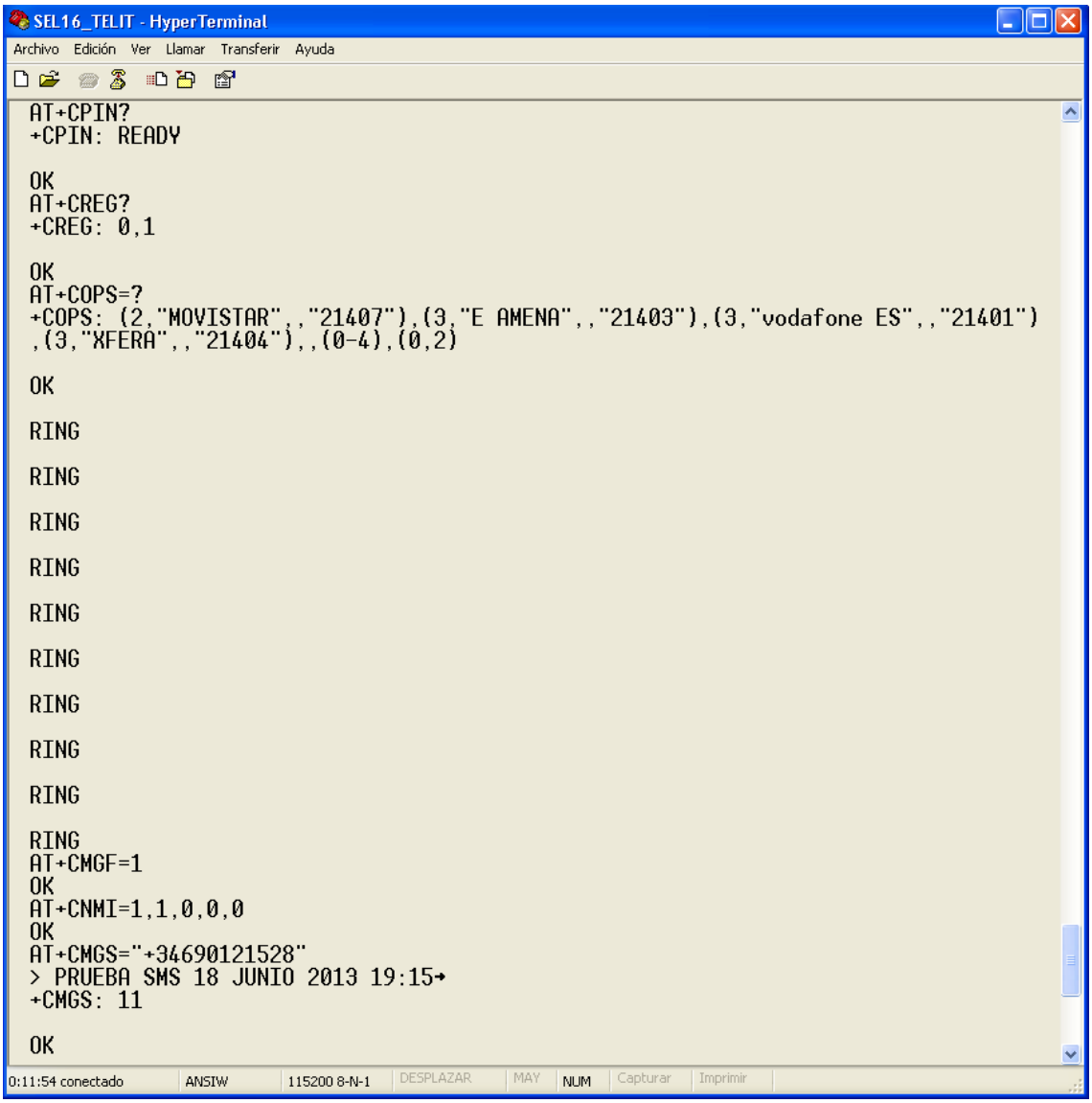


Figura 93: prueba de módulo GL-865 QUAD de SEL16.

Se han usado una serie de comandos reunidos en la guía de software de TELIT GL-865 QUAD (anexo I) que sirven para comprobar la detección de la tarjeta SIM (AT+CPIN?), si el módulo está registrado en la red telefónica móvil (AT+CREG?), detección de operadores de red (AT+COPS=?) y de envío de un SMS (usando AT+CMGF=1 para configurar el modo texto, AT+CNMI para configurar cómo serán tratados los SMS que se reciban en el módulo y AT+CMGS=”+XXXXXXXX” para enviar un SMS al número que ocupe las X).

Como se puede comprobar en la figura el módulo TELIT es completamente funcional (las notificaciones “RING” indican la detección de una llamada entrante al módulo) en SEL16, abriendo múltiples posibilidades como la comunicación a través de GPRS convirtiendo IO5-USB en un sistema inalámbrico de gestión de entradas y salidas.

6.7.2 – SISTEMA DE COMUNICACIÓN RS-422/485

Para probar el sistema de comunicación RS-422/485 a través del IC MAXIM INTEGRATED MAX3490CSA+ bastará con configurar adecuadamente las conexiones de soldadura de comunicaciones (J22 y J23 en el diseño esquemático de SEL16) para que en vez de conectar al sistema USB conecte con el sistema RS-422/485. Esto convierte a IO5-USB en "IO5-RS-422/485" (nótese que podría ser también RS-232 usando el IC adecuado dado que MAXIM INTEGRATED cuenta con multitud de ICs para estos propósitos).

Una vez hecho esto y a través del convertidor RS-422/485 a RS-232 de WUT (modelo RS422/RS485 interface compact mostrado en la Figura 94) que a su vez se conectó al puerto RS-232 del PC se comprobó que dicho sistema funciona perfectamente, de manera análoga al sistema USB de SEL16. Esto permite expandir las posibilidades de conectividad de SEL16.

Figura 94: convertidor RS-422/485 a RS-232 empleado.

CAPÍTULO 7 – OTRAS POSIBILIDADES DEL SISTEMA DISEÑADO Y CONCLUSIONES

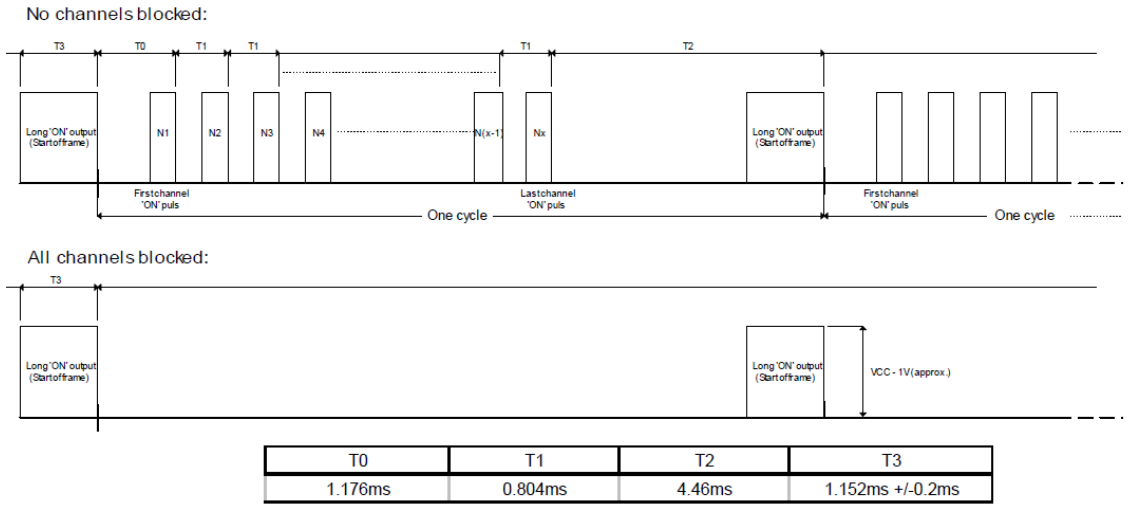
7.1 – OTRAS FUNCIONES QUE PUEDE DESEMPEÑAR EL SISTEMA

SEL16 se ha diseñado como una plataforma polivalente y según el programa que haya en el PIC y en combinación con el hardware disponible se pueden realizar multitud de soluciones comerciales. En los sucesivos sub-apartados se enumeran una serie de funciones que realiza SEL16 como solución propia en otra clase de productos distintos a IO5-USB.

7.1.1 – SENSOR DE BARRERAS DE FOTODIODOS MEDIANTE RS-422

Algunos tipos de barreras de fotodiodos ofrecen una salida de tipo tren de pulsos y por tanto en algunas aplicaciones es necesario convertir dicha salida en algún estándar de comunicación. Para la lectura de la salida de estas barreras se ha situado el conector J47 (conectado a la entrada 8 a través de los conectores de soldadura J48) sobre el conector J3 (USB) en SEL16. De esta manera, uniendo dicha soldadura y colocando ese conector en lugar del conector USB se puede conectar a dicha entrada una barrera de fotodiodos (y usar RS-422/485 como salida). Además no se usa FLX01 en este caso, recurriendo al conector auxiliar J38 para el LED de alimentación eléctrica como único indicador visual del funcionamiento del equipo, ya que su funcionamiento se comprueba a través de RS-422/485.

En la Figura 95 se muestra un tren de pulsos típico de este tipo de barreras de fotodiodos, donde se aprecia que se trata de pulsos de una duración muy pequeña (cada pulso representa el bloqueo de un fotodiodo) y que requieren de un programa eficiente en términos de tiempo.



T0	T1	T2	T3
1.176ms	0.804ms	4.46ms	1.152ms +/-0.2ms

Figura 95: tren de pulsos típico de barrera de fotodiodos.

Con el programa adecuado SEL16 se encarga de convertir este tipo de pulsos en señales del estándar industrial de comunicación RS-422, ofreciendo entre un byte de cabecera y uno de final una serie de bytes donde cada bit representa si un fotodiodo está bloqueado o no. Se trata de una aplicación real de SEL16 de la que no se pueden dar más detalles por motivos comerciales.

7.1.2 – CONTADOR DE PULSOS DE ALTA SENSIBILIDAD MEDIANTE USB

Una segunda aplicación alternativa real de SEL16 es contar en tramos de un milisegundo el tiempo que pasa entre un flanco de subida en una entrada y el siguiente, que vuelve esa cuenta de milisegundos a 0. Desde USB se monitorean estos tiempos y se hace uso de las salidas a conveniencia, luego es un programa que tiene escasa modificación sobre el de IO5-USB.

También se obtiene provecho del sistema de alimentación de SEL16 dado que se trata de una aplicación que funciona a 48 voltios de tensión de alimentación mientras que los pulsos son de 24 voltios. Por esta clase de niveles de tensión tan usuales en la industria se definieron los requisitos de diseño de sistema de alimentación, entradas y salidas de SEL16.

7.1.3 – MODIFICACIÓN SOBRE IO5-USB PARA SER GESTIONADO VÍA COMUNICACIÓN 3G

Habilitar a IO5-USB para funcionar en tiempo real a través de la red 3G sería una ampliación mayor de las prestaciones de IO5-USB que implicaría la implementación de la pila de software TCP/IP en el PIC, una tarea nada simple pero tampoco imposible. Microchip da en una nota de aplicación las directrices para implementar de manera funcional dicha pila de software en la familia PIC18 luego se puede recurrir a ella como punto de apoyo. Por otra parte existe dicha pila de software a nivel comercial, una solución razonable dado que se requiere cierta especialización en gestión de protocolos de transmisión de datos para implementarla.

7.2 – CONCLUSIONES

Qué he aprendido como autor del proyecto y qué puede aprender un lector que se ha tomado la molestia de leer esta memoria con interés es la mejor forma de abordar este apartado. IO5-USB no es un producto revolucionario en el sentido estricto de la palabra: se trata de un producto apto para competir en un segmento definido del mercado de la electrónica industrial. Ofrece una versión tecnológica del archiconocido “bueno, bonito y barato”.

Como autor del proyecto lo más valioso para mí ha sido el enriquecimiento en términos de “know-how” sobre este tipo de productos electrónicos. Es un gran aliciente trabajar en un campo donde siempre hay algo que aprender y donde la excelencia es especialmente determinante sobre qué producto triunfa y cual se marchita en el mercado.

Por otra parte, alguien que haya leído con interés esta memoria se habrá percatado de otras cosas aparte del conocimiento expuesto en la misma, por ejemplo:

- Hay razones para creer que España puede progresar significativamente en términos tecnológicos sin necesidad de inversiones multimillonarias ni pensando en competir con gigantes tecnológicos: como demuestra este proyecto se puede competir a nivel internacional con una escasa inversión siempre y cuando se disponga del conocimiento, la ilusión y las ganas de trabajar que ello requiere.

- Para competir con un producto electrónico en la actual coyuntura (globalización) es necesario definir un adecuado equilibrio entre prestaciones y precio. El secreto de esa primera frase es que el término "prestaciones" tiende a la ambigüedad y en el caso de SEL16 y FLX01 una de sus mejores prestaciones que el cliente no percibe es que se trata de un producto de bajo costo altamente configurable en el proceso de fabricación. Esto permite competir contra múltiples productos con el mismo sistema, explotando al máximo la ventaja de la economía de escala.

- La creatividad juega un papel fundamental en la ingeniería, hasta el punto donde en SEL16 se decide superponer sobre el conector USB otro conector como el de alimentación, para ser usado como una entrada de propósito especial simplemente sustituyendo un conector por otro a la hora del montaje.

Como apunte final cabe destacar que, en la dinámica de la mejora continua, ya se está investigando cómo mejorar SEL16 dado que aun se pueden ampliar tanto sus prestaciones eléctricas como sus características de personalización, como son entradas aisladas eléctricamente (sustituyendo los transistores MOSFET de las entradas por fototransistores) y buscar la manera de hacer compatible sólo cambiando de circuito integrado que la comunicación a través de RS-422/485 pueda serlo también a través de RS-232. Para concluir se muestra en la Figura 96 el conjunto del hardware formado por SEL16 y FLX01 con todos los elementos que los conforman.

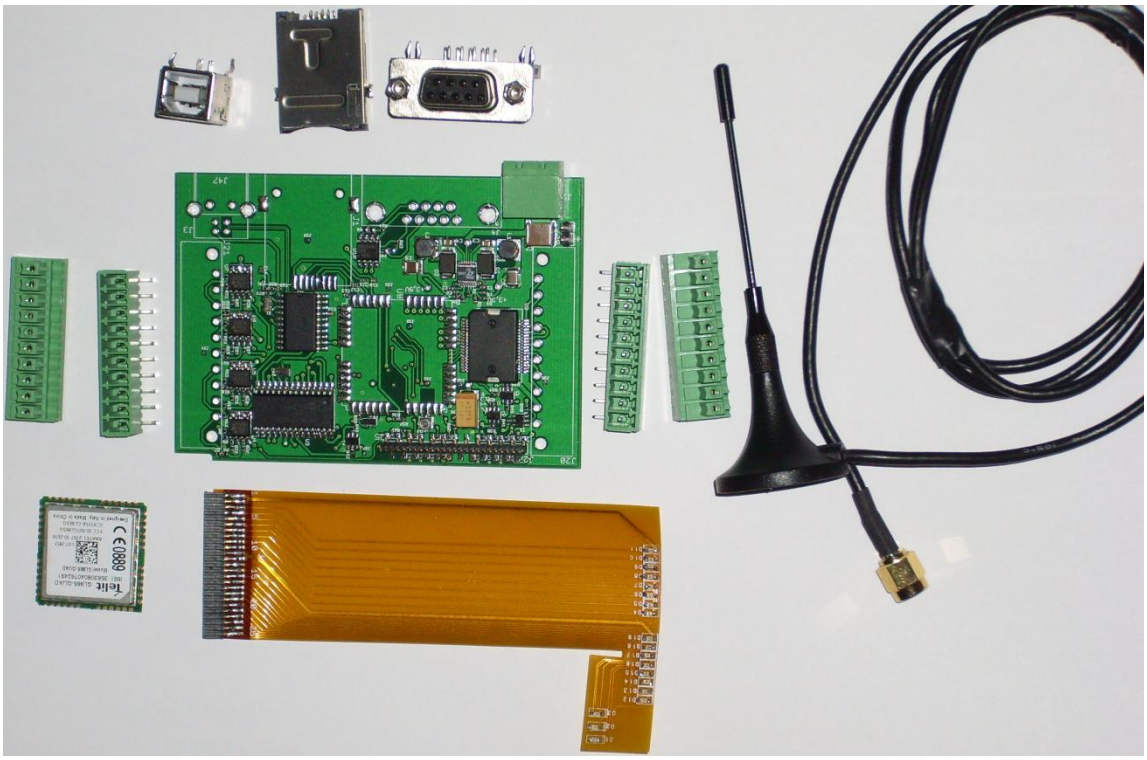


Figura 96: conjunto de todos los elementos de SEL16 y FLX01.

ANEXO A – LISTA DE MATERIALES Y COSTE DE SEL16

MANUFACTURER REFERENCE	DESCRIPTION	QUANTITY	SEL16 REFERENCE	TOTAL COST (€)
C0402C103K5RACTU	KEMET capacitor 10nF 50V X7R 0402	2	C1, C2	0,014
C0805C224K1RACTU	KEMET capacitor 220nF 100V X7R 0805	2	C3, C4	0,147
C0402C220J5GACTU	KEMET capacitor 22pF 50V 0402	2	C5, C6	0,11
C5750X7R2A475K	TDK capacitor 4,7µF 100V X7R C5750	1	C7	1,31
C3225X7R1C226K	TDK capacitor 22µF 16V X7R C3225	2	C8, C9	0,64
C0402C100J5GACTU	KEMET capacitor 10pF 50V 0402	1	C10	0,009
C0402C330J5GACTU	KEMET capacitor 33pF 50V 0402	3	C11, C12, C13	0,033
C0402C104K4RACTU	KEMET capacitor 100nF 16V 0402	2	C14, C15	0,021
C1005X5R1C474K	TDK capacitor 470nF 16V 0402	1	C16	0,022
T491D337K006AT	KEMET capacitor 330µF 6V 7343-31	1	C17	0,57
MBRS190T3G	ON SEMICONDUCTOR schottky power rectifier	2	D1, D2	0,362
CRCW0402120RFKED	VISHAY DRALORIC resistor 120Ω 0402	2	R63, R64	0,006
9002-M06	OUPIIN 6 contacts SIM card connector	1	J1	1,12
ST 08 02	HARTMANN horizontal header connector	2	J2, J47	0,88
USB-B-S-RA	ADAM TECH USB socket type B	1	J3	0,24
1734354-1	TYCO ELECTRONICS DB9 serial socket	1	J4	0,67
61301011121	WUERTH ELEKTRONIK 2 pin header 2,54mm.	1	J38	0,28
850-10-025-30-001101	PRECI-DIP 25 contact connector	1	J27	0,55
MM9329-2700	MURATA antenna connector	1	J19	0,9
1843871	PHOENIX CONTACT 3,5mm pitch connector	2	J20, J21	0,77
744773115	WUERTH ELEKTRON. 15µH power inductor	1	L1	0,92
744773118	WUERTH ELEKTRON. 18µH power inductor	1	L2	0,98

MANUFACTURER REFERENCE	**DESCRIPTION**	**QUANTITY**	**SEL16 REFERENCE**	**TOTAL COST (€)**
CRCW0402200KFKED	VISHAY DRALORIC resistor 200kΩ 0402	1	R1	0,001
CRCW040210K0FKED	VISHAY DRALORIC resistor 10kΩ 0402	3	R2, R3, R32	0,021
CRCW040234K0FKED	VISHAY DALE resistor 34kΩ 63mW 50V 0402	1	R4	0,011
CRCW040242K2FKED	VISHAY DALE resistor 42,2kΩ 63mW 50V 0402	2	R5, R34	0,02
CRCW04024K70FKEAHP	VISHAY DRALORIC resistor 4,7kΩ 0402	17	R6 … R21, R36	0,0976
CRCW040230K0FKED	VISHAY DALE resistor 30kΩ 63mW 50V 0402	8	R22 … R29	0,012
CRCW0402470RFKEAHP	VISHAY DRALORIC resistor 470Ω 0402	1	R31	0,008
CRCW0402845RFKED	VISHAY DALE resistor 845Ω 63mW 50V 0402	1	R33	0,009
ERJ2GEJ201X	PANASONIC resistor 200Ω 100mW 50V 0402	1	R35	0,006
CRCW040226K1FKED	VISHAY DALE resistor 26,1kΩ 63mW 50V 0402	1	R37	0,004
CRCW04022K70FKEAHP	VISHAY DRALORIC resistor 2,7kΩ 0402	8	R38 … R45	0,022
CRCW040268R0FKEAHP	VISHAY DRALORIC resistor 68Ω 0402	8	R46 … R53	0,027
CRCW0402220RFKEAHP	VISHAY DRALORIC resistor 220Ω 0402	9	R30, R54 … R61	0,0193
MMBT3904K	FAIRCHILD SEMICON. transistor NPN	1	T1	0,008
IMX17T110	ROHM dual NPN BJT transistor	4	T2 … T5	0,252
LT3988IMSE	LINEAR TECH. switching regulator	1	U1	5,01
VN808CM-E	STMICRO. high side driver PowerSO-36	1	U2	3,31
PIC18LF26K22-I/SO	MICROCHIP microcontroller SOIC	1	U3	2,59
SI4936BDY	VISHAY SILICONIX MOSFET transistor	4	U4 … U7	0,324
GL865-QUAD	TELIT GSM/GPRS module	1	U8	13,11
MCP2200-I/SO	MICROCHIP USB2.0-TO-UART module SOIC	1	U9	1,51
MAX3490CSA+	MAXIM RS-422/485 transceiver SOIC	1	U10	2,19
CSTCE12M0G15L	MURATA ceramic resonator 12MHz 33pF	1	X1	0,88
CSTCE16M0V53	MURATA ceramic resonator 16MHz 15pF	1	X2	0,97
MAL203858479E3	VISHAY BC COMP. capacitor 47uF 63V	1	C18	0,121
CRCW0402330RFKEAHP	VISHAY DRALORIC resistor 330Ω 0402	1	R62	0,015
NO	Printed Circuit Board 4 layers	1	NO	12,12
4969014350	WÖHR plastic enclosure type 1435	1	NO	9,91

El coste de materiales de SEL16 si se implementara con todos sus componentes (nunca se colocan todos en un modelo que se venda) e incluyendo la carcasa asciende a 63,13 euros, en el caso de encargar al menos 500 unidades. A este coste faltaría añadir el coste de transporte y aduana tras la fabricación, lo que finalmente rondaría 80 euros por cada placa.

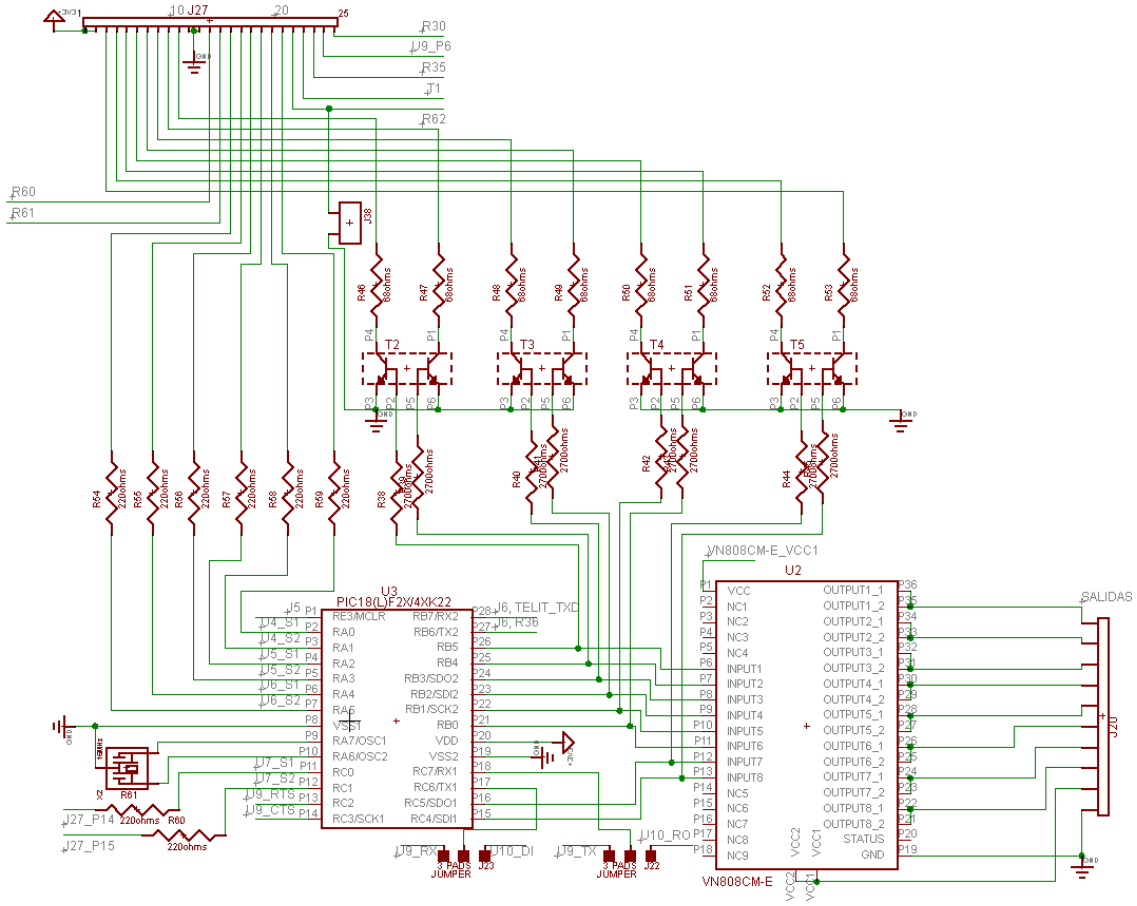
ANEXO B – LISTA DE MATERIALES Y COSTE DE FLX01

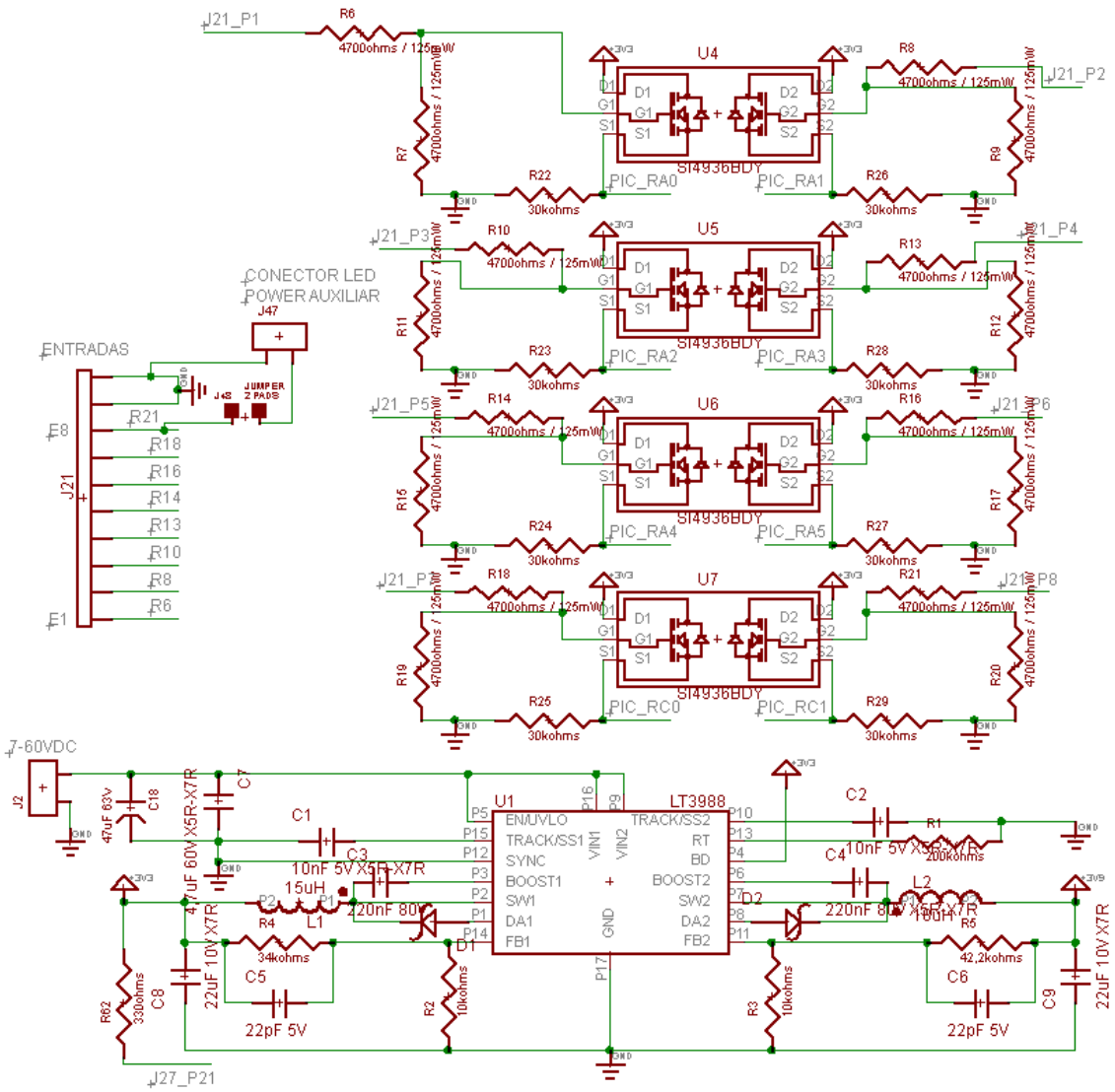
MANUFACTURER REFERENCE	DESCRIPTION	QUANTITY	FLX01 REFERENCE	TOTAL COST (€)
851-10-025-30-001101	PRECI-DIP 25 contact connector	1	J27	0,49
KPG-1608SEKC-T	KINGBRIGHT red LED 0603	1	D1	0,071
KPG-1608CGKC-T	KINGBRIGHT green LED 0603	18	D2 … D19	1,296
NO	Flexible Printed Circuit Board 1 layer	1	NO	8,7

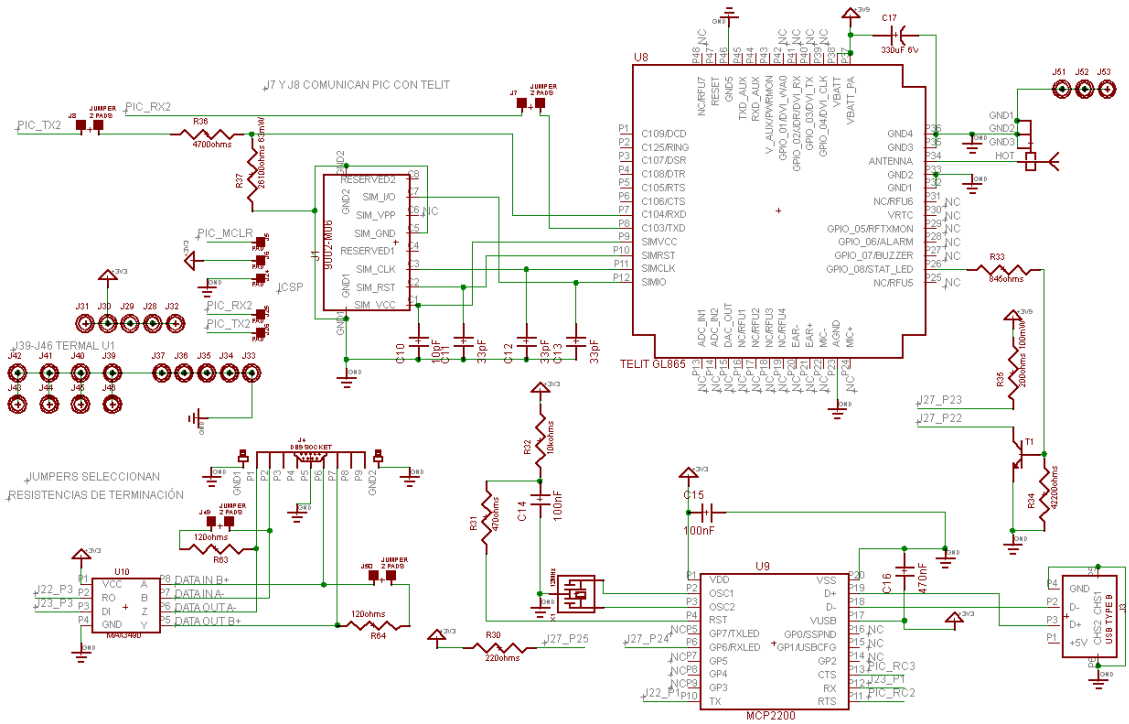
El coste de materiales de FLX01 asciende a 10,56 euros por unidad si se encargan al menos 500 unidades. El coste después de transporte y aduana rondaría los 18 euros, nótese que el grueso del coste de FLX01 es precisamente la PCB flexible.

ANEXO C – DIAGRAMA ESQUEMÁTICO DE SEL16

En este anexo se encuentra el diagrama esquemático de SEL16 en conjunto con fines ilustrativos, dado que se fraccionó para ser explicado en el capítulo 3.




J21_P1
R6
4700ohms / 125mW
U4
SI4936BDY
R8
J21_P2
R7
R9
R22
30kohms
PIC_RA0
PIC_RA1
R26
J21_P3
R10
U5
R13
J21_P4
R11
R12
R23
PIC_RA2
PIC_RA3
R28
CONECTOR LED
POWER AUXILIAR
J47
ENTRADAS
JUMPER
2 PADS
J48
J21_P5
R14
U6
R16
J21_P6
R15
R17
R24
PIC_RA4
PIC_RA5
R27
J21_P7
R18
U7
R21
J21_P8
R19
R20
R25
PIC_RC0
PIC_RC1
R29
7-60VDC
J2
C18
47uF 63v
C7
47uF 60V X5R-X7R
C1
C3
10nF 5V X5R-X7R
15uH
L1
220nF 80V
R4
34kohms
C5
22pF 5V
C8
22uF 10V X7R
R62
330ohms
J27_P21
U1
LT3988
EN/UVLO
TRACK/SS1
SYNC
BOOST1
SW1
DA1
FB1
TRACK/SS2
RT
BD
BOOST2
SW2
DA2
FB2
VIN1
VIN2
GND
D1
R2
10kohms
C2
R1
200kohms
C4
10nF 5V X5R-X7R
D2
220nF 80V X5R-X7R
L2
100uH
R5
42,2kohms
C6
22pF 5V
R3
10kohms
C9
22uF 10V X7R


J7 Y J8 COMUNICAN PIC CON TELIT
PIC_RX2
PIC_TX2
R36
4700ohms
R37
2610ohms
PIC_MCLR
CSP
J39-J46 TERMAL U1
JUMPERS SELECCIONAN
RESISTENCIAS DE TERMINACIÓN
120ohms
R83
R64
U10
MAX3490
DATA IN B+
DATA IN A-
DATA OUT A-
DATA OUT B+
U8
TELIT GL865
C10
10pF
C11
33pF
C12
33pF
C13
33pF
C17
330uF 6v
R33
845ohms
R35
200ohms
T1
R34
4220ohms
R32
18kohms
R31
470ohms
C14
100nF
C15
100nF
R30
220ohms
J27_P25
J27_P24
J27_P23
J27_P22
J22_P1
U9
MCP2200
C16
470nF
J22_P3
J23_P3
PIC_RC3
PIC_RC2
J23_P1
USB TYPE B

ANEXO D – CARACTERÍSTICAS DEL MICROCONTROLADOR MICROCHIP TECHNOLOGY PIC18LF26K22

A continuación se detallan las características más relevantes de este PIC de acuerdo a la versión DS41412F de sus hojas características.

High-Performance RISC CPU:

- C Compiler Optimized Architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- Up to 1024 Bytes Data EEPROM
- Up to 64 Kbytes Linear Program Memory Addressing
- Up to 3896 Bytes Linear Data Memory Addressing
- Up to 16 MIPS Operation
- 16-bit Wide Instructions, 8-bit Wide Data Path
- Priority Levels for Interrupts
- 31-Level, Software Accessible Hardware Stack
- 8 x 8 Single-Cycle Hardware Multiplier

Flexible Oscillator Structure:

- Precision 16 MHz Internal Oscillator Block:
  - Factory calibrated to ± 1%
  - Selectable frequencies, 31 kHz to 16 MHz
  - 64 MHz performance available using PLL – no external components required
- Four Crystal modes up to 64 MHz
- Two External Clock modes up to 64 MHz
- 4X Phase Lock Loop (PLL)
- Secondary Oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if peripheral clock stops
  - Two-Speed Oscillator Start-up

Analog Features:

- Analog-to-Digital Converter (ADC) module:
  - 10-bit resolution, up to 30 external channels
  - Auto-acquisition capability
  - Conversion available during Sleep
  - Fixed Voltage Reference (FVR) channel
  - Independent input multiplexing
- Analog Comparator module:
  - Two rail-to-rail analog comparators
  - Independent input multiplexing
- Digital-to-Analog Converter (DAC) module:
  - Fixed Voltage Reference (FVR) with 1.024V, 2.048V and 4.096V output levels
  - 5-bit rail-to-rail resistive DAC with positive and negative reference selection
- Charge Time Measurement Unit (CTMU) module:
  - Supports capacitive touch sensing for touch screens and capacitive switches

Extreme Low-Power Management PIC18(L)F2X/4XK22 with XLP:

- Sleep mode: 20 nA, typical
- Watchdog Timer: 300 nA, typical
- Timer1 Oscillator: 800 nA @ 32 kHz
- Peripheral Module Disable

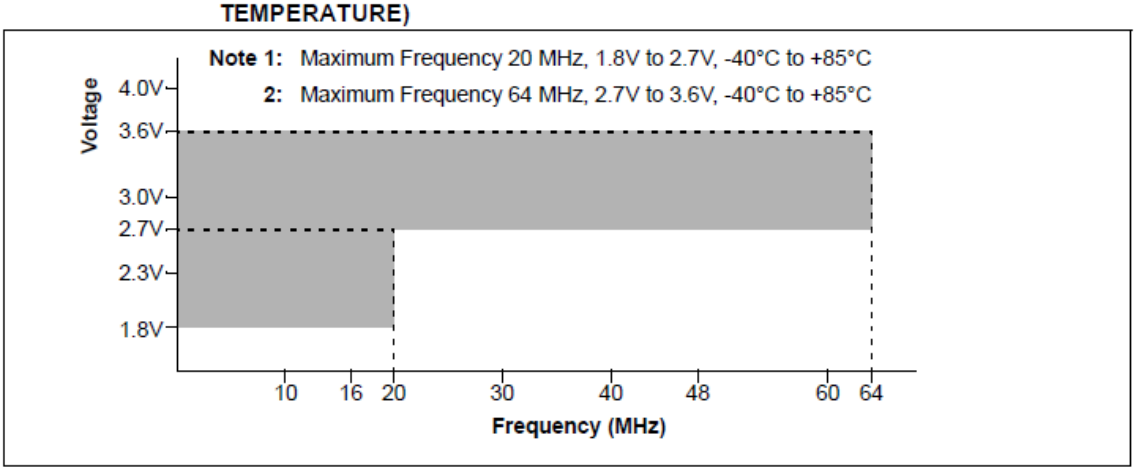
Special Microcontroller Features:

- 2.3V to 5.5V Operation – PIC18FXXK22 devices
- 1.8V to 3.6V Operation – PIC18LFXXK22 devices
- Self-Programmable under Software Control
- High/Low-Voltage Detection (HLVD) module:
  - Programmable 16-Level
  - Interrupt on High/Low-Voltage Detection
- Programmable Brown-out Reset (BOR):
  - With software enable option
  - Configurable shutdown in Sleep
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- In-Circuit Serial Programming™ (ICSP™):
  - Single-Supply 3V
- In-Circuit Debug (ICD)

Peripheral Highlights:

- Up to 35 I/O Pins plus 1 Input-Only Pin:
  - High-Current Sink/Source 25 mA/25 mA
  - Three programmable external interrupts
  - Four programmable interrupt-on-change
  - Nine programmable weak pull-ups
  - Programmable slew rate
- SR Latch:
  - Multiple Set/Reset input options
- Two Capture/Compare/PWM (CCP) modules
- Three Enhanced CCP (ECCP) modules:
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart
  - PWM steering
- Two Master Synchronous Serial Port (MSSP) modules:
  - 3-wire SPI (supports all 4 modes)
  - I^2C™ Master and Slave modes with address mask

FIGURE 27-1: PIC18LF2X/4XK22 FAMILY VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL TEMPERATURE)



Device	Program Memory		Data Memory		I/O(1)	10-bit A/D Channels(2)	CCP	ECCP (Full-Bridge)	ECCP (Half-Bridge)	MSSP		EUSART	Comparator	CTMU	BOR/LVD	SR Latch	8-bit Timer	16-bit Timer
	Flash (Bytes)	# Single-Word Instructions	SRAM (Bytes)	EEPROM (Bytes)						SPI	I²C™							
PIC18(L)F23K22	8K	4096	512	256	25	19	2	1	2	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F24K22	16K	8192	768	256	25	19	2	1	2	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F25K22	32K	16384	1536	256	25	19	2	1	2	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F26K22	64k	32768	3896	1024	25	19	2	1	2	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F43K22	8K	4096	512	256	36	30	2	2	1	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F44K22	16K	8192	768	256	36	30	2	2	1	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F45K22	32K	16384	1536	256	36	30	2	2	1	2	2	2	2	Y	Y	Y	3	4
PIC18(L)F46K22	64k	32768	3896	1024	36	30	2	2	1	2	2	2	2	Y	Y	Y	3	4

Note 1: One pin is input only.

2: Channel count includes internal FVR and DAC channels.

27.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings (†)

Ambient temperature under bias ..... -40°C to +125°C
Storage temperature ..... -65°C to +150°C
Voltage on any pin with respect to V_{SS} (except V_{DD}, and $\overline{MCLR}$) ..... -0.3V to (V_{DD} + 0.3V)
Voltage on V_{DD} with respect to V_{SS}
PIC18LF2X/4XK22 ..... -0.3V to +4.5V
PIC18F2X/4XK22 ..... -0.3V to +6.5V
Voltage on $\overline{MCLR}$ with respect to V_{SS} **(Note 2)** ..... 0V to +11.0V
Total power dissipation **(Note 1)** 1.0W
Maximum current out of V_{SS} pin (-40°C to +85°C) ..... 300 mA
Maximum current out of V_{SS} pin (+85°C to +125°C) ..... 125 mA
Maximum current into V_{DD} pin (-40°C to +85°C) ..... 200 mA
Maximum current into V_{DD} pin (+85°C to +125°C) ..... 85 mA
Input clamp current, I_{IK} (V_I < 0 or V_I > V_{DD}) ..... ±20 mA
Output clamp current, I_{OK} (V_O < 0 or V_O > V_{DD}) ..... ±20 mA
Maximum output current sunk by any I/O pin ..... 25 mA
Maximum output current sourced by any I/O pin ..... 25 mA
Maximum current sunk by all ports (-40°C to +85°C) ..... 200 mA
Maximum current sunk by all ports (+85°C to +125°C) ..... 110 mA
Maximum current sourced by all ports (-40°C to +85°C) ..... 185 mA
Maximum current sourced by all ports (+85°C to +125°C) ..... 70 mA

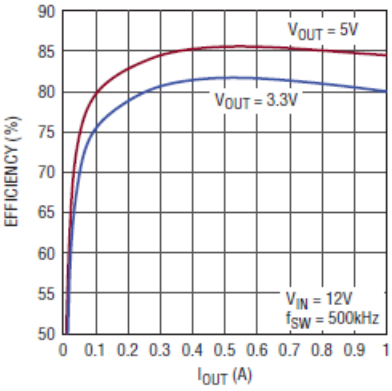
Note 1: Power dissipation is calculated as follows:
$P_{dis} = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD} - V_{OH}) \times I_{OH}\} + \sum (V_{OL} \times I_{OL})$

2: Voltage spikes below V_{SS} at the $\overline{MCLR}$/V_{PP}/RE3 pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the $\overline{MCLR}$/V_{PP}/RE3 pin, rather than pulling this pin directly to V_{SS}.

† **NOTICE:** Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

ANEXO E – CARACTERISTICAS DEL CIRCUITO INTEGRADO LINEAR TECHNOLOGIES LT3988

En este anexo se adjuntan los parámetros más reseñables del IC LT3988 de acuerdo con las hojas de características del mismo, versión LT3988f. Se incluyen las ecuaciones empleadas para el cálculo de circuitos con este regulador.



ABSOLUTE MAXIMUM RATINGS

V_{IN} (Notes 7, 8)	–0.3V to 80V
BOOST	80V
EN/UVLO (Note 7)	80V
BOOST above SW	30V
EN/UVLO above V_{IN1}	6V
RT, SYNC	6V
TRACK/SS, FB	5V
BD	20V
Operating Junction Temperature Range (Note 2)	
LT3988E	–40°C to 125°C
LT3988I	–40°C to 125°C
LT3988H	–40°C to 150°C
Storage Temperature Range	–65°C to 150°C
Lead Temperature (Soldering, 10 sec)	300°C

STEP-DOWN CONSIDERATIONS

FB Resistor Network

$$R1 = R2\left(\frac{V_{OUT}}{750mV} - 1\right)$$

Switching Frequency

$$R_T = \frac{1.31}{f^2} + \frac{46.56}{f} - 7.322$$

250kHz ≤ f ≤ 2.5MHz

where f is in MHz and R_T is in kΩ.

Input Capacitor Selection

Bypass the input of the LT3988 circuit with a 4.7µF or higher ceramic capacitor of X7R or X5R type. A lower value or a less expensive Y5V type will work if there is additional bypassing provided by bulk electrolytic capacitors, or if the input source impedance is low. The following paragraphs describe the input capacitor considerations in more detail.

Inductor Selection

$$L_{MIN} = \frac{V_{OUT} + V_F}{1.25A \bullet f}$$

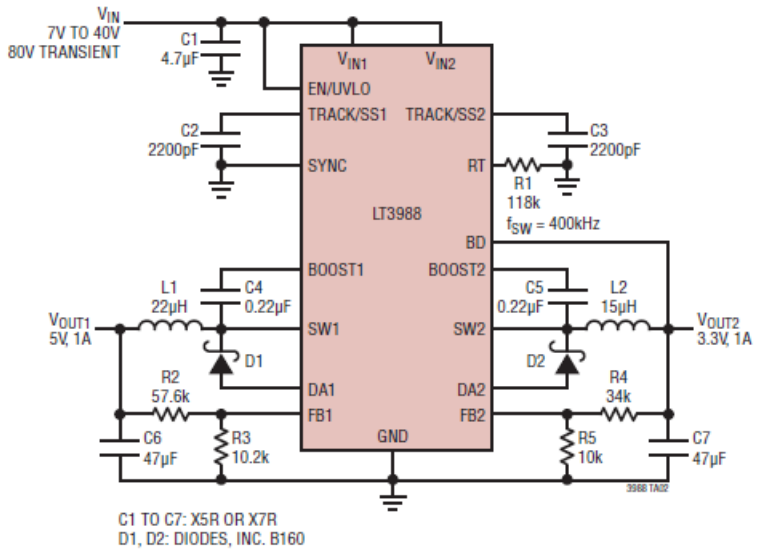
with L_{MIN} in µH and f in MHz.

Output Capacitor Selection

$$C_{OUT} > 10 \bullet L \bullet \left(\frac{I_{LIM}}{V_{OUT}}\right)^2$$

TYPICAL APPLICATIONS

400kHz, 5V and 3.3V Outputs



C1 TO C7: X5R OR X7R
D1, D2: DIODES, INC. B160

ANEXO F – CARACTERÍSTICAS DEL CIRCUITO INTEGRADO ST MICROELECTRONICS VN808CM-E

Este anexo recoge las características más relevantes de este high side driver de acuerdo con el documento 11456 revisión 8 de STMicroelectronics.

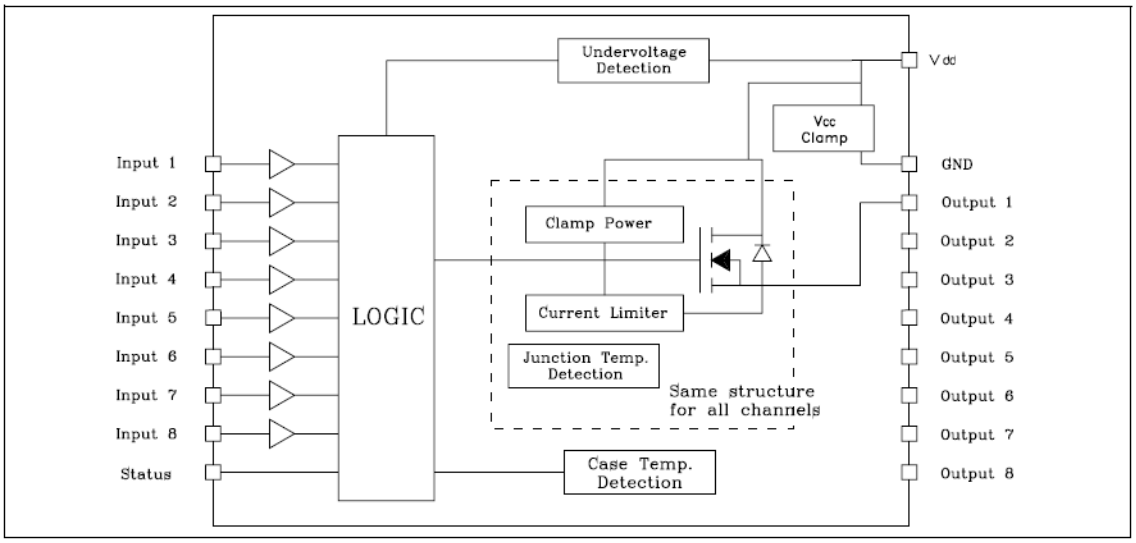


Table 1. Absolute maximum rating

Symbol	Parameter	Value	Unit
V_{CC}	DC supply voltage	45	V
$-I_{GND}$	DC ground pin reverse current TRAN ground pin reverse current (pulse duration < 1ms)	-250 -6	mA A
I_{OUT}	DC output current	Internally limited	A
$-I_{OUT}$	Reverse DC output current	-2	A
I_{IN}	DC Input current	± 10	mA
V_{ESD}	Electrostatic discharge (R = 1.5 kΩ; C = 100 pF)	2000	V
P_{TOT}	Power dissipation at T_c = 25 °C	96	W
L_{MAX}	Max inductive load (V_{CC} = 24 V, R_{LOAD} = 48 Ω, T_A = 100 °C)	2	H
T_J	Junction operating temperature	Internally limited	°C
T_C	Case operating temperature	Internally limited	°C
T_{STG}	Storage temperature	-40 to 150	°C

Table 2. Thermal data

Symbol	Parameter		Value	Unit
R_{thJC}	Thermal resistance junction-case	Max	1.3	°C/W
R_{thJA}	Thermal resistance junction-ambient (1)	Max	50	°C/W

1. When mounted on FR4 printed circuit board with 0.5 cm^2 of copper area (at least 35 µm think) connected to all TAB pins.

Table 3. Power section

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
V_{CC}	Operating supply voltage		10.5		45	V
V_{USD}	Undervoltage shutdown		7		10.5	V
R_{ON}	On state resistance	I_{OUT} = 0.5 A; T_J = 25 °C I_{OUT} = 0.5 A;			160 280	mΩ mΩ
I_S	Supply current	OFF state; V_{CC} = 24 V; T_{CASE} = 25 °C ON state (all channels ON); V_{CC} = 24 V, T_{CASE} = 100 °C			150 12	µA mA
I_{LGND}	Output current at turn-off	V_{CC} = V_{STAT} = V_{IN} = V_{GND} = 24 V V_{OUT} = 0 V			1	mA
$I_{L(off)}$	OFF state output current	V_{IN} = V_{OUT} = 0 V;	0		5	µA
$V_{OUT(off)}$	OFF state output voltage	V_{IN} = 0 V, I_{OUT} = 0 A			3	V
$t_{d(Vccon)}$	Power-on delay time from V_{CC} rising edge	*Figure 7 on page 12*		1		ms

Table 4. Switching (V_{CC} = 24 V)

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
t_{ON}	Turn-on time	R_L = 48 Ω from 80% V_{OUT} *Figure 4.*	-	50	100	µs
t_{OFF}	Turn-off time	R_L = 48 Ω to 10% V_{OUT} *Figure 4.*	-	75	150	µs
$dV_{OUT}/dt_{(on)}$	Turn-on voltage slope	R_L = 48 Ω from V_{OUT} = 2.4 V to V_{OUT} = 19.2 V *Figure 4.*	-	0.7		V/µs
$dV_{OUT}/dt_{(off)}$	Turn-off voltage slope	R_L = 48 Ω from V_{OUT} = 21.6 V to V_{OUT} = 2.4 V *Figure 4.*	-	1.5		V/µs

ANEXO G – CARACTERÍSTICAS DEL CIRCUITO INTEGRADO MICROCHIP TECHNOLOGY MCP2200

A continuación se detallan las características del IC MCP2200, extraídas del documento de características DS222288B.

Features

Universal Serial Bus (USB)

- Supports full-speed USB (12 Mb/s)
- Implements USB protocol composite device:
  - Communication Device Class (CDC) for communications and configuration
  - Human Interface Device (HID) for I/O control
- 128-byte buffer to handle data throughput at any UART baud rate:
  - 64-byte transmit
  - 64-byte receive
- Fully configurable VID and PID assignments, and string descriptors
- Bus powered or self-powered
- USB 2.0 Compliant: TID 40001150

USB Driver and Software Support

- Uses standard Windows® drivers for Virtual Com Port (VCP): Windows XP (SP2 or later), Vista, 7
- Configuration utility for initial configuration

Universal Asynchronous Receiver/Transmitter (UART)

- Responds to `SET LINE CODING` commands to dynamically change baud rates
- Supports baud rates: 300-1000k
- Hardware flow control
- UART signal polarity option

General Purpose Input/Output (GPIO) Pins

- Eight general purpose I/O pins

EEPROM

- 256 bytes of user EEPROM

Other

- USB activity LED outputs (TxLED and RxLED)
- SSPND output pin
- USBCFG output pin (indicates when the enumeration is completed)
- Operating voltage: 3.0-5.5V
- Oscillator input: 12 MHz
- ESD protection: > 4 kV HBM
- Industrial (I) Operating Temperature: -40°C to +85°C

Block Diagram

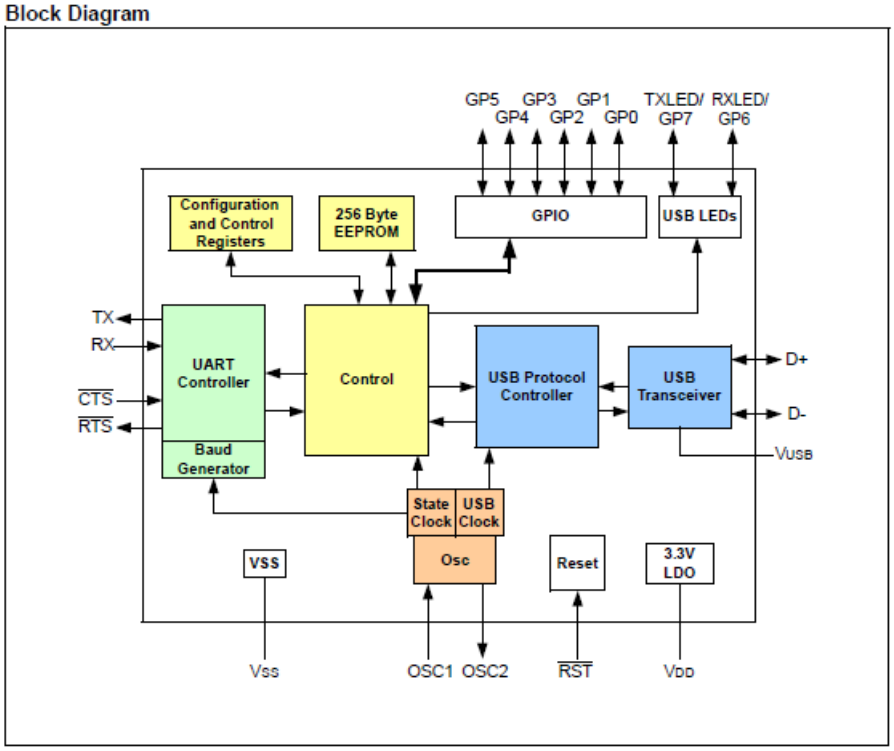


FIGURE 1-4: TYPICAL POWER SUPPLY OPTION USING THE 5V PROVIDED BY THE USB

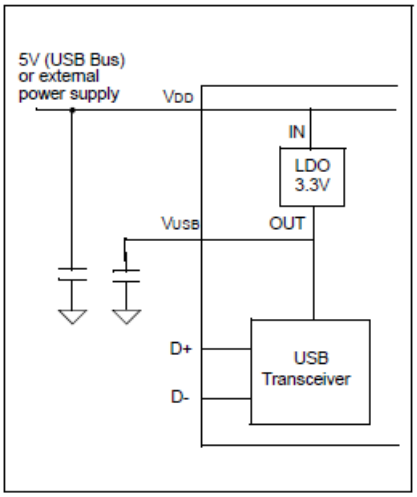
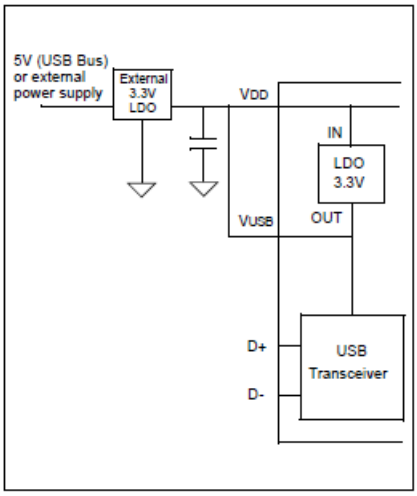


FIGURE 1-5: USING AN EXTERNALLY PROVIDED 3.3V POWER SUPPLY



1.3 UART Interface

The MCP2200 UART interface consists of the TX and RX data signals and the $\overline{RTS}$/$\overline{CTS}$ flow control pins.

The UART is configurable for several baud rates. The available baud rates are listed in Table 1-3.

FIGURE 1-1: $\overline{RTS}$/$\overline{CTS}$ CONNECTIONS EXAMPLE

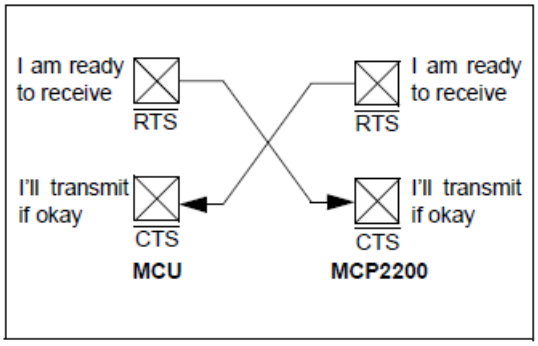


TABLE 1-3: UART PRIMARY BAUD RATES

Desired Rate	Actual rate	% Error
300	300	0.00%
1200	1200	0.00%
2400	2400	0.00%
4800	4800	0.00%
9600	9600	0.00%
19200	19200	0.00%
38400	38339	0.16%
57600	57692	0.16%
115200	115385	0.16%
230400	230769	0.16%
460800	461538	0.16%
921600	923077	0.16%

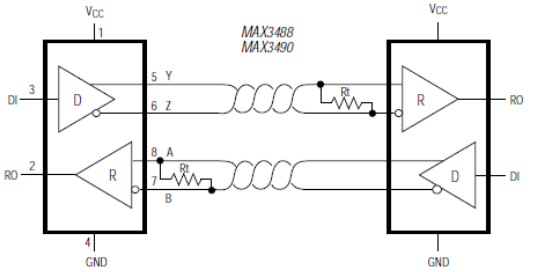
ANEXO H – CARACTERÍSTICAS DEL CIRCUITO INTEGRADO MAXIM INTEGRATED MAX3490

Detalle de las características más significativas del IC MAX3490 extraídas del documento 19-0333 revisión 0.

- **Operate from a Single 3.3V Supply—No Charge Pump!**
- **Interoperable with +5V Logic**
- **8ns Max Skew (MAX3485/MAX3490/MAX3491)**
- **Slew-Rate Limited for Errorless Data Transmission (MAX3483/MAX3488)**
- **2nA Low-Current Shutdown Mode (MAX3483/MAX3485/MAX3486/MAX3491)**
- **-7V to +12V Common-Mode Input Voltage Range**
- **Allows up to 32 Transceivers on the Bus**
- **Full-Duplex and Half-Duplex Versions Available**
- **Industry Standard 75176 Pinout (MAX3483/MAX3485/MAX3486)**
- **Current-Limiting and Thermal Shutdown for Driver Overload Protection**

ABSOLUTE MAXIMUM RATINGS

Supply Voltage (V_{CC}) 7V
Control Input Voltage ($\overline{RE}$, DE) ..... -0.3V to 7V
Driver Input Voltage (DI) ..... -0.3V to 7V
Driver Output Voltage (A, B, Y, Z) ..... -7.5V to 12.5V
Receiver Input Voltage (A, B) ..... -7.5V to 12.5V
Receiver Output Voltage (RO) ..... -0.3V to (V_{CC} + 0.3V)
Continuous Power Dissipation (T_A = +70°C)
8-Pin Plastic DIP (derate 9.09mW/°C above +70°C) 727mW
8-Pin SO (derate 5.88mW/°C above +70°C) 471mW



PART NUMBER	GUARANTEED DATA RATE (Mbps)	SUPPLY VOLTAGE (V)	HALF/FULL DUPLEX	SLEW-RATE LIMITED	DRIVER/ RECEIVER ENABLE	SHUTDOWN CURRENT (nA)	PIN COUNT
MAX3483	0.25	3.0 to 3.6	Half	Yes	Yes	2	8
MAX3485	10		Half	No	Yes	2	8
MAX3486	2.5		Half	Yes	Yes	2	8
MAX3488	0.25		Full	Yes	No	—	8
MAX3490	10		Full	No	No	—	8
MAX3491	10		Full	No	Yes	2	14

Devices without Receiver/Driver Enable (MAX3488/MAX3490)

Table 3. Transmitting

INPUT	OUTPUTS	
DI	Z	Y
1	0	1
0	1	0

Table 4. Receiving

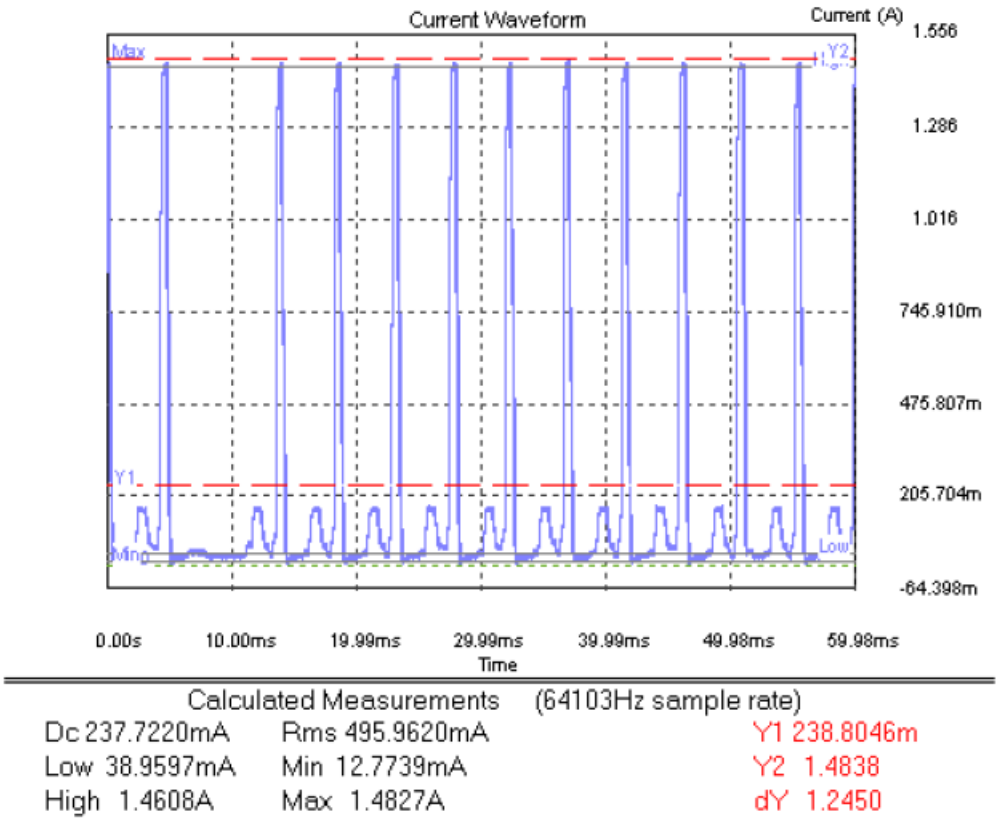
INPUTS	OUTPUT
A, B	RO
≥ +0.2V	1
≤ -0.2V	0
Inputs Open	1

ANEXO I – CARACTERÍSTICAS DEL MÓDULO TELIT GL-865 QUAD

Se presentan en este anexo las características de hardware y software básicas del módulo TELIT GL-865 QUAD. Los datos de hardware son extraídos de la guía de hardware de dicho módulo, revisión 6 mientras que los comandos de software expuestos (que se emplearon para las pruebas documentadas en el apartado 6.7.1) han sido extraídos de la guía de software del módulo, versión 13.

6.2.1. Power consumption Plots

GSM900 – Voice Call – Power level 5

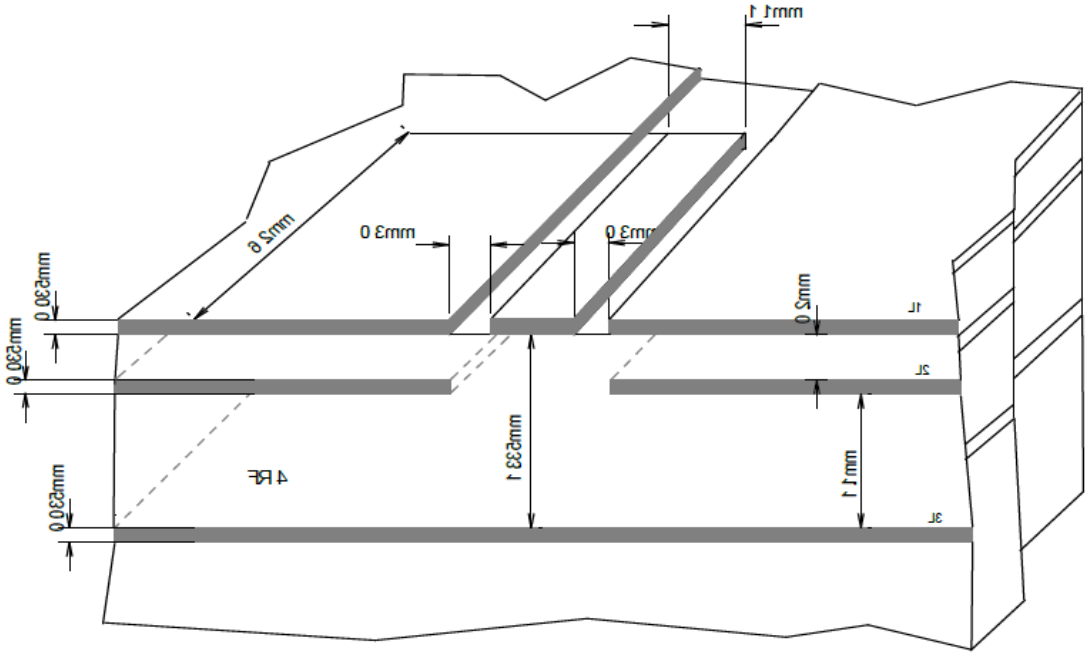


ANTENNA LINE ON PCB REQUIREMENTS	
Impedance	50 ohm
Max Attenuation	0,3 dB
No coupling with other signals allowed	
Cold End (Ground Plane) of antenna shall be equipotential to the GL865 **ground pins**	

This transmission line should be designed according to the following guidelines:

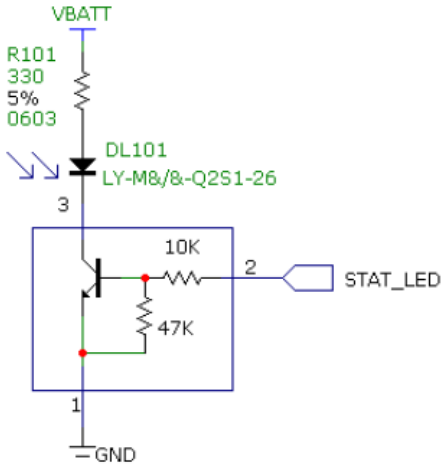
- Ensure that the antenna line impedance is 50 ohm;
- Keep the antenna line on the PCB as short as possible, since the antenna line loss shall be less than 0,3 dB;
- Antenna line must have uniform characteristics, constant cross section, avoid meanders and abrupt curves;
- Keep, if possible, one layer of the PCB used only for the Ground plane;
- Surround (on the sides, over and under) the antenna line on PCB with Ground, avoid having other signal tracks facing directly the antenna line track;
- The ground around the antenna line on PCB has to be strictly connected to the Ground Plane by placing vias once per 2mm at least;
- Place EM noisy devices as far as possible from GL865 antenna line;
- Keep the antenna line far away from the GL865 power supply lines;
- If you have EM noisy devices around the PCB hosting the GL865, such as fast switching ICs, take care of the shielding of the antenna line by burying it inside the layers of PCB and surround it with Ground planes, or shield it with a metal frame cover.
- If you don't have EM noisy devices around the PCB of GL865, by using a strip-line on the superficial copper layer for the antenna line, the line attenuation will be lower than a buried one;

7.2. PCB Guidelines in case of FCC certification



11.9. STAT LED Indication of network service availability

A schematic example could be:



Operating Range - Interface levels (2.8V CMOS)

Level	Min	Max
Input high level	2.1V	3.1V
Input low level	0V	0.5V
Output high level	2.2V	3.1V
Output low level	0V	0.35V

ANEXO J – CARACTERÍSTICAS TÉCNICAS DEL MATERIAL FR-4

General Properties 一般特性

Item 测试项目	Unit 单位	Test Method (IPC-TM-650) 测试方法	Test Condition 处理条件	Specification （IPC-4101B） 规格值	Typical Value 典型值
Peel Strength (1 oz.) 铜箔剥离强度	N/mm	2.4.8	125℃	AABUS	1.30
			Float 288℃/ 10 Sec	AABUS	1.25
Flammability 燃烧性	Rating	UL94	E-24/23	UL94 V-0	V-0
Glass Transition (Tg) 玻璃化转变温度	℃	2.4.25	E-2/105 DSC	≥170	175
Surface Resistivity 表面电阻	MΩ	2.5.17.1	C-96/35/90	$\geq 1.0\times10^{4}$	1.0×10^{7}
Volume Resistivity 体积电阻	MΩ-cm	2.5.17.1	C-96/35/90	$\geq 1.0\times10^{6}$	1.0×10^{9}
Dielectric Breakdown 介质击穿	kV	2.5.6	D-48/50+D0.5/23	≥40	70
Dielectric Strength 介质强度	kV/mm	2.5.6.2	D-48/50+D0.5/23	≥29	44
Dielectric Constant 介电常数	—	2.5.5.2	Etched/@1 MHZ	≤5.4	4.65
Loss Tangent 介质损耗	—	2.5.5.2	Etched/@1 MHZ	≤0.035	0.018
Moisture Absorption 吸水率	%	2.6.2.1	D-24/23	≤0.35(min0.51)	0.15
				≤0.80(max0.51)	0.25
Flexural Strength 抗弯强度	N/mm^2	2.4.4	Warp	≥415	589
			Fill	≥345	456
Arc Resistance 耐电弧性	Sec	2.5.1	D-48/50+D0.5/23	≥60	125
CTE/Z-Axis Expansion	ppm/℃	2.4.24	E-2/105 TMA	≤60/300	46/267
Z-轴热膨胀系数	%（50-260℃）			≤3.0	2.8
CTI	V	IEC 60112	Etched/0.1% NH_4CL	≥175	175
TD	℃	2.4.24.6	TGA	≥340	357
CAF	H	-	85%,85℃,50V DC	≥1000	1000
T-260	min	2.4.24.1	TMA	≥30	50
T-288	min	2.4.24.1	TMA	≥15	30

Remarks: Specimen Thickness: 1.6 mm 样品厚度：1.6 mm

ANEXO K – DETALLES DE PROGRAMACIÓN DE IO5-USB

1-GENERALIDADES

Parámetros de conexión: 115200bps 8-N-1 no control de flujo por hardware y mismo formato de intercambio de mensajes que XBEE pero con un solo byte de longitud de datos en vez de dos como en XBEE, ejemplo:

7E 01 01 FE

Donde 7E es la cabecera, el primer 01 es el número de bytes desde el que hay después de este byte de longitud y hasta el anterior al byte de CRC (la longitud máxima de un paquete son 61 bytes y por tanto de una trama completa 64 bytes), el 01 siguiente es la función (la cual después podría tener parámetros, en este caso no) y el último es el CRC, calculado como en XBEE: FF-(8bits)(suma de todos los bytes desde el de después de longitud hasta el anterior del CRC).

Las mismas funciones tienen una interpretación distinta en IO5-USB y en el PC y cuando el CRC de un envío del PC es erróneo IO5-USB responde con la función "AA": 7E 01 AA 55. Si el primer byte de cualquier envío no es 7E es automáticamente ignorado por IO5-USB y si un mensaje tarda más de un segundo en llegar desde la cabecera (siendo ésta válida) IO5-USB responde con la función "AB": 7E 01 AB 54. Aunque un mensaje sea válido en cabecera, longitud y CRC, si la función que se envía es desconocida por IO5-USB y/o alguno de los posibles parámetros de esa función, IO5-USB contesta con la función "AC": 7E 01 AC 53.

Esto se hace por si se añaden funcionalidades conforme se actualice el software de funciones y no hay concordancia entre una funcionalidad y la versión del software de IO5-USB. En la comunicación IO5-USB siempre es esclavo del PC, respondiendo a lo que envía el PC y siempre ha de esperarse la respuesta por parte de IO5-USB (o un timeout si tarda más de un segundo en contestar) para hacer un nuevo envío desde el PC.

2-FUNCIONES

00 - LEER O MODIFICAR EL MODO DE FUNCIONAMIENTO

Es una función con un parámetro que permite solicitar a IO5-USB el modo actual de funcionamiento o modificarlo. Los parámetros disponibles son FE para solicitar el modo actual, 00 para activar el modo RUN y FF para activar el modo STOP (el cual permite manejar las salidas manualmente desde el PC). Para solicitar el modo actual se debe enviar:

7E 02 00 FE 01

Y la respuesta de IO5-USB sería:

7E 02 00 XX YY

Donde YY es el CRC y XX es el modo actual de funcionamiento: FF significa STOP y 00 RUN. En el caso de activar el modo RUN o STOP la respuesta de IO5-USB será el mismo mensaje enviado por el PC. El modo de funcionamiento se mantiene aunque se desconecte el equipo. El modo programa tiene una opción para establecer el estado predeterminado de las salidas en modo RUN, sin embargo las salidas siempre están a 0 si está en modo STOP. Cada vez que se envía a IO5-USB que pase al modo RUN, sin importar el estado en el que esté, todos los contadores vuelven a 0, es como si se volviera a conectar el sistema desde una desconexión.

Ejemplos:

7E 02 00 00 FF activar modo RUN.

7E 02 00 FF 00 activar modo STOP.

01 - LEER ESTADO ACTUAL DE LAS ENTRADAS/SALIDAS

Es una función sin parámetros que devuelve en el mensaje de respuesta dos bytes de datos, uno correspondiente a las entradas y otro a las salidas. No depende del modo de funcionamiento de IO5-USB, siempre se puede solicitar esta información, la trama de envío desde el PC es:

7E 01 01 FE

Y la respuesta de IO5-USB, con valores aleatorios de entradas y salidas a modo de ejemplo, sería:

7E 03 01 AB 11 42

Donde AB (10101011) es el estado de las entradas (siendo el bit más significativo la entrada 8 y en orden decreciente el resto de entradas hasta que el bit menos significativo indica el estado de la entrada 1) y 11 es el estado de las salidas, siguiendo la misma analogía descrita para las entradas.

02 - MODIFICAR EL ESTADO ACTUAL DE LAS SALIDAS

Esta función sólo tiene un parámetro cuando se envía desde el PC, el cual debe indicar el valor que deben tomar las salidas siguiendo un orden descendente desde el bit de más peso. Sin embargo, devuelve distintos mensajes dependiendo del modo de funcionamiento que esté en vigor en IO5-USB. Si el modo de funcionamiento es RUN devuelve 7E 03 02 FF FF FF donde los dos parámetros con FF significan que no se pueden modificar las salidas mientras está habilitado el modo RUN. Si está en modo STOP se devuelve el mismo mensaje que se recibió, indicando que la operación se ha ejecutado exitosamente.

Ejemplos de modificación de las salidas: 7E 02 02 AA 53, 7E 02 02 00 FD, 7E 02 02 55 A8, 7E 02 02 FF FE.

03 - PROGRAMACIÓN DE SALIDAS

Esta función permite introducir un programa en IO5-USB, el cual será ejecutado al activar el modo RUN. Su longitud es fija y su primer parámetro, para la primera versión del software de IO5-USB es 01, que indica el número de tramas de programación que se van a recibir, lo cual de momento es sólo una. El siguiente parámetro indica el valor inicial de las salidas al conectarse IO5-USB (sin importar el modo de funcionamiento). Por supuesto, poner una salida con un valor inicial y luego no implementar ningún modo de funcionamiento en ella implica que esa salida se mantenga de manera permanente en ese valor.

Tras ese parámetro se suceden, para cada una de las 8 salidas y por orden ascendente entre ellas, 5 bytes de configuración de cada una de ellas, donde se configura tanto el modo de salida como el umbral del contador que está asociado a cada una. Aunque cada salida tiene un contador asociado a ella, éste no tiene porqué accionarla, pudiendo usarse únicamente para contar flancos o pulsos de una entrada y luego leerse su valor usando la función 04 (CONFIGURAR/LEER CONTADORES).

El proceso completo de programar IO5-USB es la secuencia de poner el sistema en modo STOP usando la función 00, programar las salidas usando la función 03, programar los contadores usando la función 04, poner un valor inicial en los contadores si procede con la función 04 y de nuevo poner el sistema en modo RUN con la función 00 para que funcione el nuevo programa. Este proceso debe ser automático desde el punto de vista del usuario, aunque éste no use los contadores o alguna salida, todos los parámetros deben ser enviados a IO5-USB. Ahora nos ocupa la función 03.

Un ejemplo de trama de programación de salidas sería la siguiente:

7E 2B 03 01 00 VV WW XX YY ZZ VV WW XX YY ZZ …

Donde 2B es la longitud, 03 la función y 01 el número de trama. Estos tres parámetros son iguales para la primera versión de IO5-USB y se envía una única trama para programar el modo de las salidas. El siguiente 00 es el estado inicial de las salidas (no importando el modo en el que funcione IO5-USB), donde el bit más significativo es la salida 8 y el menos significativo la salida 1. A partir de este byte se suceden secciones de 5 bytes conteniendo la configuración para cada salida (y el umbral del contador asociado a ésta) en orden ascendente: desde el primer VV hasta ZZ corresponde a la configuración de la primera entrada, después la segunda y así sucesivamente. Una vez han pasado 8 secciones de 5 bytes sólo faltaría el byte de CRC para completar el mensaje de programación para enviarlo. Siempre que algún bit no sea significativo para la configuración elegida para una salida y/o contador, debe dejarse a nivel alto.

El byte VV contiene la configuración de la salida en cuestión. Los valores posibles de este byte son: 0 modo AND, 1 modo OR y 2 modo CONTADOR. Los modos desde el 3 hasta el 255 no están implementados en esta versión y debe usarse el modo 255 si no se usa ninguno de los 3 modos implementados (lo cual significa salida deshabilitada). La salida en el modo 2 depende directamente del uso que se haga del contador correspondiente a dicha salida, mientras que en el resto de modos usar el contador para contar es opcional.

El byte WW se define de manera diferente en función de la configuración elegida en el byte VV. Si el modo de la salida es AND u OR, este byte es la máscara de las entradas para realizar la función lógica correspondiente. Si el modo es un CONTADOR este byte son los 8 bits de mayor peso en el umbral de desbordamiento de ese contador. El byte XX, de manera análoga al byte VV, es la máscara de las entradas negadas para realizar las funciones lógicas y en el caso de un modo que usa contador, este byte es el segundo de mayor peso en el umbral. Los bytes YY y ZZ son la parte baja del umbral en el modo contador y son irrelevantes en los modos lógicos (AND y OR). Una salida se accionará por contador siempre que la cuenta sea mayor o igual al umbral.

La acción sobre una salida siempre es invertir el estado original de ésta. Por ejemplo, si se configura la salida 1 como estado predeterminado a nivel alto y se la programa para que se accione cuando su contador llegue a 5, cuando se cumpla esa condición la salida 1 se pondrá a nivel bajo y volverá a nivel alto si la condición deja de cumplirse. El funcionamiento es idéntico para el caso opuesto: si se configura como estado predeterminado el nivel bajo, se pondrá a nivel alto cuando se cumpla la condición correspondiente.

04 – CONFIGURAR/LEER CONTADORES

Esta función permite leer y configurar el estado de los contadores, además de leer el número de serie del equipo. Los contadores cuentan números positivos de 4 bytes de tamaño, volviendo el contador a 0 cuando se sobrepasa la cuenta. Para configurar los contadores se usa el parámetro FF, enviándolo en el siguiente tipo de mensaje a IO5-USB:

7E 1B 04 FF XX YY ZZ XX YY ZZ… XX YY ZZ MM AA

Donde 1B es la longitud fija en el caso de la función 04 con el parámetro FF (configurar contadores) y cada secuencia de 3 bytes "XX YY ZZ" es la configuración para cada uno de los 8 contadores, de manera ascendente. Al final de la secuencia de 3 bytes durante 8 veces está el byte MM, que es el umbral en número de milisegundos para el modo de baja sensibilidad de los modos que cuentan flancos y AA es el byte de CRC (el cual habría que calcular, naturalmente).

El byte XX indica la configuración del contador correspondiente de esta manera: los tres bits de mayor peso indican la configuración del contador (0 flanco de bajada, 1 flanco de subida, 2 pulso de duración mayor al umbral de pulso a nivel alto y 255 contador desactivado). El siguiente bit indica si el contador se usa en modo de baja sensibilidad (si está a nivel alto el bit), lo cual sólo es relevante en los modos de flanco de subida y bajada. El modo de baja sensibilidad descarta pulsos menores de 10 milisegundos mientras que si este modo está desactivado cualquier flanco será contabilizado.

El siguiente bit indica, si está a nivel alto, que una entrada es capaz de devolver el contador correspondiente a 0 con lo que ello implica dependiendo del umbral que se haya establecido para ese contador previamente mediante la función 03, al programarla. Los bits restantes, los tres de menor peso, codifican la entrada que vuelve el contador a 0, donde en esa codificación el 0 indica que es la entrada 1 y de manera ascendente hasta que el 7 indica que es la entrada 8. Naturalmente esto es irrelevante si no se ha habilitado mediante el bit que habilita esta posibilidad.

El byte YY está dividido en dos secciones: los tres bits de mayor peso codifican la entrada que interpreta el contador, de nuevo siendo 0 la entrada 1 y 7 la entrada 8. Los bits restantes forman los bits de mayor peso a la hora de evaluar el umbral en el que un pulso a nivel alto es anotado por el contador, en número de milisegundos. El byte ZZ son los 8 bits bajos de este número, siendo el número máximo de este umbral en formato decimal 8191 milisegundos. La respuesta a este mensaje desde IO5-USB es el mismo mensaje que envió el PC.

Si con esta función se usa el parámetro FE IO5-USB devuelve, en tramos de 4 bytes y de manera ascendente el valor actual de los 8 contadores y por último otros 4 bytes correspondientes al número de serie de la placa, el cual es único y no se puede modificar. Un ejemplo sería este:

Envío desde el PC: 7E 02 04 FE FD (esta trama es fija)

Respuesta de IO5-USB: 7E 26 04 FE WW XX YY ZZ WW XX YY ZZ…

Donde los primeros cuatro bytes “WW XX YY ZZ” son el valor actual del contador 1 y así sucesivamente hasta el contador 8 y después, otros 4 bytes correspondientes al número de serie del equipo.

Los otros parámetros posibles para esta función son del 0 al 7, donde el 0 significa modificar el valor actual del contador 1 y así sucesivamente hasta que el 7 es modificar el valor del contador 8. Esto básicamente sirve para volver a 0 (o al número que se desee) un contador. Esto se puede hacer tanto en modo RUN como en STOP y como en todas las solicitudes donde se exige a IO5-USB modificar parámetros, la contestación si la operación ha sido exitosa es el mismo mensaje que envió el PC. Un ejemplo sería enviar:

7E 06 04 00 00 00 00 00 FB

En esa instrucción se pone a 0 el contador asociado a la salida 1 y IO5-USB contesta con el mismo mensaje.

FF – MODIFICAR NÚMERO DE SERIE

Esta función no es accesible para el usuario, sólo lo es para nosotros como fabricante. Permite introducir los 4 bytes que representan el número de serie del equipo. La trama para esto es:

7E 0A FF FF FF WW XX YY ZZ FF FF FF MM

En esa trama se compone de una longitud fija, donde “WW XX YY ZZ” son el número de serie y MM el CRC, de manera análoga al resto de funciones. El sistema debe devolver el mismo mensaje enviado si el cambio de número de serie ha sido correcto.

BIBLIOGRAFÍA

[1] http://www.usb.org, especificación USB.

[2] http://www.linear.com/, detalles sobre diseño de convertidores conmutados.

[3] http://www.microchip.com/, uso de PIC18LF26K22 y MCP2200.

[4] http://www.telit.com/, uso del módulo TELIT GL-865 QUAD.

[5] http://www.maximintegrated.com/, uso de MAX3490 y detalles sobre RS-422/485.

[6] http://www.st.com/web/en/home.html, gestión electrónica de salidas digitales.

[7] http://www.digi.com/es/, soporte para X-CTU.

[8] http://wut.de/e-wwwww-ww-hpus-000.php, fabricante de productos similares a IO5-USB.