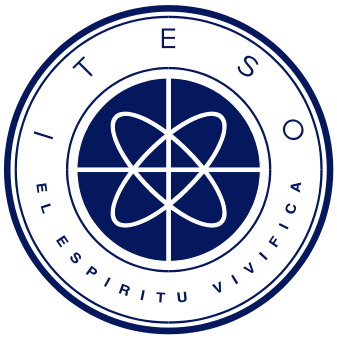
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

MAESTRÍA EN DISEÑO ELECTRÓNICO



REPORTE DE FORMACIÓN COMPLEMENTARIA EN ÁREA DE CONCENTRACIÓN EN SISTEMAS EMBEBIDOS Y TELECOMUNICACIONES

Trabajo recepcional que para obtener el grado de

MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: Cástulo Javier Martínez Ramírez

San Pedro Tlaquepaque, Jalisco. 24 de Mayo del 2017.

Contenido

Introducción

El siguiente documento se presenta como parte del proceso de titulación de la maestría en diseño electrónico. La opción de titulación elegida y presentada en este documento corresponde a aquella llamada “Formación complementaria y proyectos de impacto en área de concentración”.

La formación técnica del alumno comenzó con el estudio de la carrera de Ingeniería en Electrónica en la misma universidad, ITESO, de la cual egresó en el año 2004. Sin embargo desde su salida de la universidad su experiencia profesional se ha desarrollado en el área de software, y más específicamente el área de aseguramiento de la calidad de software, teniendo ya al día de hoy más de 12 años de experiencia en esta área. El área de concentración que seleccionó de la maestría de diseño electrónico fue la de “Sistemas embebidos y telecomunicaciones” principalmente por dos razones; primero, una de las mayores áreas de interés del alumno es la programación, por lo que el desarrollo de software para sistemas embebidos captó su atención de inmediato además de ser el área que el alumno consideró más afín al área en el que se desenvuelve profesionalmente, y segundo, desde que estudiaba la licenciatura, siempre ha tenido gusto por los sistemas embebidos, y una prueba de ello es que su proyecto de tesis de licenciatura fue relacionado con un sistema embebido.

El área de concentración de sistemas embebidos y telecomunicaciones es la más extensa de todas y cuenta con ocho materias posibles de las que el estudiante debe elegir al menos tres. Las materias seleccionadas por el estudiante son las siguientes:

1. Taller de diseño de tarjetas de circuito impreso.
   Impartida por: José Guadalupe Guzmán
   Razón para la selección de este curso: Como se mencionó anteriormente, la mayor parte de la experiencia profesional del alumno ha sido con software, y uno de los factores que para él son de lo más atractivos en el campo del software es la facilidad y factibilidad de llevar un proyecto desde su concepción hasta la

implementación, ya que generalmente no se requiere de mucha infraestructura o equipo, en su forma más básica solo una computadora y acceso a internet, con eso es posible implementar proyectos completos listos para ir a producción. En contraste, para que un proyecto de hardware pueda ser implementado a partir de su conceptualización y diseño, en algo real y comercializable, es necesario también saber cómo plasmar ese diseño dentro de una tarjeta de circuito impreso. Por esta razón este curso complementa perfectamente los conocimientos de diseño de circuitos electrónicos desarrollados durante la licenciatura y maestría.

2. Ingeniería de software en ambientes embebidos.
   Impartida por: Héctor Rivas
   Razón para la selección de este curso: el trabajo del estudiante como se mencionó anteriormente está íntimamente ligado con la ingeniería de software, razón por la cual el alumno ha ido aprendiendo en este tema, sin embargo nunca llevó una formación formal al respecto. Este curso fue capaz de ayudarle a aterrizar conceptos que ha aprendido empíricamente durante su trabajo con software, además le proporcionó al alumno un punto de comparación en cuanto a la ingeniería de software utilizada con un sistema embebido vs un software diseñado para correr sobre un sistema tradicional (en un ordenador común).

3. Diseño de sistemas analógicos basados en dispositivos comerciales.
   Impartida por: René Padilla – Esteban Martínez
   Razón para la selección de este curso: Al igual que con el curso de taller de diseño de tarjetas de circuito impreso, lo que le llamó la atención de este curso al alumno es la necesidad de poder llevar proyectos de la concepción a la implementación en la vida real. En la vida real, el diseño de un sistema electrónico se basa en la utilización de dispositivos comerciales, se deben considerar factores que no se consideran cuando se diseña un sistema que en realidad no se piensa construir o comercializar, cuestiones como costo de componentes, inventario, etc. Por este motivo se consideró este curso como una parte complementaria en la formación del estudiante.

1. Resumen de los proyectos realizados

1.1. Proyecto 1: Creación del *layout* de un sistema electrónico implementado en tecnología de tarjeta de circuito impreso

1.1.1 Introducción

El proyecto desarrollado durante esta asignatura fue la creación del *layout* de un sistema electrónico implementado en tecnología de Tarjeta de Circuito Impreso, desde la creación de librerías, la captura del esquemático hasta el diseño físico del PCB mismo. Durante el proceso de creación del *layout* el alumno debió verificar la validez de su diseño por medio de la aplicación de diversas reglas de diseño automatizadas en tecnologías de circuitos impresos, considerando especificaciones de procesos tecnológico comerciales disponibles, además de aplicar reglas heurísticas recomendadas de *layout* para manufacturabilidad, ensamble, pruebas, y para minimizar interferencia electromagnética y otros efectos de alta velocidad. Este proyecto fue realizado individualmente, no en equipo, se fue completando paulatinamente durante el transcurso del semestre, donde el trabajo asignado en casa sesión iba construyéndose sobre el trabajo anterior.

1.1.2 Antecedentes

Los PCBs (*Printed Circuit Board*) se han utilizado desde principios de los años 50s, aunque su concepción data desde aproximadamente cincuenta años atrás, cuando Frank Sprague tuvo la idea de eliminar el cableado punto a punto en 1904 y le confirió esta idea a su mentor, Thomas A Edison. Desde entonces los PCBs han cambiado enormemente y actualmente los PCBs son dispositivos usados no solo para interconectar componentes, sino que también son usados como estructura mecánica, conductor de calor, como blindaje del ruido, e incluso como un elemento de circuito, por lo que los PCBs deben pasar por un proceso de manufactura complicado. Existen además varios tipos de PCBs, con una cara, de doble lado, *multilayer*, con componenetes por un lado o ambos lados, etc. Además el tipo de producto final que se contendrá

en el PCB también debe ser considerado al momento de construir el PCB ya que podría haber diferentes reglas a seguirse dependiendo de este. Todos estos factores complican el diseño de estos circuitos impresos, el propósito del proyecto fue familiarizar al estudiante con todos estos conceptos relacionados con la creación de *layouts* de sistemas electrónicos en circuitos impresos, con las herramientas computacionales utilizadas para estos desarrollos y las técnicas de diseño para la posterior fabricación de un PCB que sigue las buenas prácticas establecidas por el IPC (anteriormente conocido como *Institute for Printed Circuits*).

1.1.3 Solución desarrollada

La solución desarrollada durante la materia consiste en la creación del layout para un PCB de un sistema electrónico el cual tiene como finalidad funcionar como PDA (*Personal Digital Assistant*). El circuito electrónico (esquemático) utilizado no fue diseñado por el alumno, sino que fue proporcionado por el profesor de la asignatura, ya que este sale del alcance de la materia, así mismo el BOM (*Bill Of Material*) del circuito se le proporcionó al alumno, y la labor del alumno fue desarrollar todo el diseño del *layout* tomando estos dos documentos de entrada, y el proyecto culmina con la elaboración de los archivos necesarios para la fabricación del PCB. La construcción física del PCB queda fuera del alcance debido a los costos monetarios relacionados con esto.

1.1.4 Análisis de resultados

Debido a lo laborioso y complejo que es el proceso de creación de un *layout* de PCB se observó que es propenso a cometer errores, y que un pequeño error en una etapa temprana del diseño traerá consigo problemas mucho mayores en etapas posteriores de la elaboración del PCB, los cuales serán muy costosos de reparar. Por ejemplo, suponiendo que al crear el *footprint* para un componente el diseñador interpreta incorrectamente las unidades de medición proporcionadas por el fabricante del componente, esto muy probablemente resulte en un problema de escala del *footprint*, el cual, si es detectado una vez que se ha hecho el *placement* de los componentes y se está trabajando en el ruteo, en una tarjeta con espacio limitado, provocará muchos dolores de cabeza al diseñador para arreglarlos, al punto que quizás tenga que empezar

de nuevo. De la misma manera se recomienda utilizar las herramientas de verificación de reglas del software de diseño en cada etapa del proceso donde estén disponibles, para minimizar cambios posteriores, costosos al proyecto. Como un ejemplo de esto sería el uso del "*Design Rules Check*" al terminar del capturar el esquemático y antes de comenzar con el *placement* de los componentes.

1.1.5 Conclusiones

La elaboración de este proyecto, sirvió para que el estudiante conociera los distintos tipos de PCBs existentes, además de los fundamentos para su elaboración. El estudiante logró identificar los pasos que conforman el proceso de diseño de un PCB, así como identificar que hay prácticas recomendadas en cada punto del diseño, con la finalidad de minimizar errores en el proceso, el cuál es propenso a ellos, creando con su reporte final (ver apéndice) una especie de guía de consulta para posteriores creaciones de PCBs. El estudiante además aprendió a utilizar las herramientas de validación contenidas en el software de diseño con las cuales se puede corroborar el cumplimiento de las reglas heurísticas recomendadas por la industria. Después de este proyecto el alumno será capaz efectivamente de crear un PCB para un proyecto de sistema electrónico si así lo desease.

1.2. Proyecto 2: Aplicación de la ingeniería de software en ambientes embebidos para el diseño de un SPI (*Serial Peripheral Interface*)

1.2.1 Introducción

El proyecto elaborado durante esta asignatura consistió en la aplicación de principios y metodologías de la ingeniería de software para el diseño de un interface serial o SPI. Este proyecto fue desarrollado de manera individual. Gran parte del interés del alumno era identificar las principales diferencias en cuanto a las metodologías de ingeniería de software para ambientes embebidos, con respecto a la misma en sistemas tradicionales, ya que el alumno tiene vasta experiencia práctica en la aplicación de estas metodologías en ambientes tradicionales.

1.2.2 Antecedentes

El concepto de ingeniería de software surgió en 1968 con el objetivo de resolver los problemas de las crisis del software. Básicamente, la crisis del software se refiere a la dificultad en escribir programas libres de defectos, fácilmente comprensibles, y que sean verificables. Las causas son, entre otras, la complejidad que supone la tarea de programar, y los cambios a los que se tiene que ver sometido un programa para ser continuamente adaptado a las necesidades de los usuarios. La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y el estudio de estos enfoques.

1.2.3 Solución desarrollada

El proyecto para esta asignatura consistió en el diseño de un sistema de interface serial para un sistema embebido. Los sistemas embebidos modernos cuentan ya generalmente con uno o varios módulos para interfaces seriales, sin embargo para términos del proyecto descrito aquí, el sistema SPI fue diseñado asumiendo que este dispositivo no estaba disponible en el sistema. La parte fundamental del proyecto fue el utilizar las metodologías de la ingeniería de software para la creación del diseño electrónico. El modelo de desarrollo seleccionado para el proyecto fue el "*V-Model*". Cabe mencionar que debido a los alcances de la materia, y el tiempo disponible, no fue necesaria la implementación real del SPI, por lo que la parte de codificación quedó fuera del proyecto. También es importante recalcar que el proyecto consistió en la entrega de diversos documentos independientes creados durante el proceso de diseño del SPI, ver anexo B para consultarlos.

1.2.4 Análisis de resultados

La aplicación de las metodologías de la ingeniería de software en sistemas embebidos es sencillo aunque puede ser laboriosa. Es importante dedicar el tiempo necesario en cada una de las etapas del modelo de desarrollo y completar los entregables desarrollados durante cada fase de manera que se puedan minimizar errores dentro de lo posible en etapas posteriores del

proyecto. Estas metodologías además comprenden una excelente guía para el desarrollo de una aplicación de software, dando herramientas para el desarrollo de software con alta calidad y que cumple satisfactoriamente las necesidades para el que fue concebido. La ingeniería de software no solo ayuda a identificar posibles problemas con el software que se está desarrollando sino que promueve la prevención de estos errores, entre más pronto se localicen errores dentro del proyecto, o se eviten, más barato y sencillo será arreglarlos.

1.2.5 Conclusiones

Aun cuando el alumno tenía experiencia previa en el desarrollo y pruebas de software, se obtuvieron algunas conclusiones importantes del estudio de la asignatura. Primero que nada se pudo identificar que las mismas metodologías utilizadas para desarrollar software en sistemas tradicionales (PCs) son válidas y aplicables a sistemas embebidos. También se logró aterrizar varios conceptos y metodologías de la ingeniería de software, varias de las cuales el estudiante solo conocía de manera empírica. Una cosa que vale la pena recalcar como conclusión, es que en área de la ingeniería de software es muy fácil caer en la trampa de no seguir los procesos o metodologías, ya que muchas veces el seguir los procesos pareciera que solo añaden más trabajo y no agregan valor. El alumno ha visto pasar esto decenas de veces en proyectos comerciales en empresas grandes, lo cual a final de cuentas ha llevado a varios de estos proyectos al fracaso. Se ha comprobado que sin estas metodologías se pone en riesgo tanto la calidad de los proyectos, como su presupuesto y/o el plazo acordado para su desarrollo.

1.3. Proyecto 3: Fuente de switcheo tipo Buck para un tablero de instrumentación automotriz

1.3.1 Introducción

El proyecto elaborado para esta asignatura consistió en el diseño de una fuente de alimentación conmutada tipo Buck para un tablero de instrumentación automotriz. El proyecto fue realizado por un equipo de dos integrantes. A grandes rasgos, el sistema cosiste en una fuente

de voltaje que provee consistentemente 5V de salida, con hasta 1.5A, la cual podría ser usada para alimentar diversos dispositivos digitales utilizados en los tableros automotrices, tomando en cuenta que la entrada de voltaje del sistema proviene de una batería que tiene un voltaje típico automotriz de 9V a 16V. Para la realización del proyecto, se definieron los requerimientos técnicos en base a la funcionalidad requerida, y se propuso el diseño del circuito seleccionando sus componentes en base a diversos factores como temperatura, características eléctricas, incluso si los componentes estaban autorizados dentro de la compañía que los utilizaría.

1.3.2 Antecedentes

Como se mencionó anteriormente, el proyecto fue realizado por un equipo de dos integrantes, uno de los cuales, trabajaba en una compañía que manufactura dispositivos automotrices, por lo que se aprovechó el proyecto de la asignatura para diseñar algo que pudiera ser de utilidad en la realidad. El voltaje suministrado típicamente en los automóviles por el alternador va de los 9V a los 16V, y habiendo algunos dispositivos que funcionan con 5V de corriente directa en el tablero automotriz es necesario proporcionar este nivel de voltaje. Las fuentes de alimentación convencionales usan transformadores, por lo que suelen ser costosas, pesadas, y voluminosas, además de dar una bajo rendimiento de conversión. Las fuentes conmutadas proveen rendimientos elevados, son de bajo costo y volumen además de utilizar muy pocos componentes, lo que las hace más confiables.

1.3.3 Solución desarrollada

Para poder proponer la solución fue importante primero analizar los requerimientos funcionales y técnicos que se tienen para el circuito, mostrados en la siguiente tabla.

TABLA I
REQUERIMIENTOS DE LA FUENTE DE VOLTAJE

Requerimientos	Rangos
Voltaje de entrada	9V – 16V
Voltaje de salida	5V ± 3%
Rizo máximo de voltaje	< 1%

Corriente de salida	Hasta 1.5A

Debido a los requerimientos y características de las fuentes conmutadas sobre las fuentes lineales se optó por el diseño de un convertidor de DC-DC step-down o también conocida como fuente de switcheo tipo Buck. Para la realización del proyecto se decidió utilizar el controlador de step-down DC/DC TLE 6389 de Infineon, el cual sirve para generar la señal que controla el encendido apagado del transistor de switcheo regulando así el voltaje de salida. Se utilizó el circuito y componentes recomendados por el fabricante. El esquemático utilizado puede ser consultado en el Anexo C.

1.3.4 Análisis de resultados

Con el desarrollo del proyecto se pudieron observar varias ventajas del convertidor Buck, por ejemplo la alta eficiencia a la que puede llegar, la poca variación al voltaje de salida, la simplicidad del circuito entre otras cosas. En la simulación del circuito se observó que el rizo de voltaje de la salida es pequeño, más pequeño del que se encontraba en los requerimientos de la fuente, por lo que el circuito es una buena opción para las necesidades especificadas.

1.3.5 Conclusiones

Durante el estudio de la ingeniería electrónica aprendemos a diseñar los circuitos de manera muy ideal, ya que la mayoría de las veces cuando diseñamos algo es con fines didácticos, por lo que pocas veces llega el circuito a su construcción, y cuando se hace se hace en pequeño volumen, generalmente solo un sistema. En la vida real es poco factible seguir esta mismo metodología de diseño, debido a varios factores, como disponibilidad de los componentes, costos, tamaños, etc. Por ejemplo digamos que al calcular un valor de resistencia encontramos un valor que no está disponible comercialmente, en este caso tendríamos que buscar un equivalente echando mano de dos o más resistencias, pero a mayor resistencia mayor costo, quizás en un proyecto donde solo habrá un sistema no importa, pero en uno donde se piensa producir en grandes cantidades se vuelve un factor crucial. Por esto mismo el diseñador debe considerar

diferentes aspectos al momento de diseñar los sistemas y seleccionar componentes. Algo que se observó durante el curso también, es que muchas veces los diseños electrónicos en la vida real, se basan mucho en las llamadas “reglas de dedo”, lo que en cierta manera facilita el diseño, ya que esto se basa mucho en aproximaciones.

2. Conclusiones

El desarrollo de los proyectos mencionados en este reporte aportaron principalmente dos puntos importantes a la formación del alumno. El primero es la realización de que las prácticas de ingeniería de software utilizadas para el desarrollo y validación de sistemas en una computadora son portables en su mayoría a los sistemas embebidos. Esto es importante ya que la experiencia laboral del alumno es en esta área, lo que le permite extender su campo de acción en la indutria donde labora, siendo capaz de colaborar tanto en equipos de software de computadora como de sistemas embebidos. El segundo punto importante fue el reforzamiento y complementación de conocimiento en sistemas embebidos, esta es un área de la industria que está creciendo rápidamente y que con el aumento en el interés en el IoT (Internet Of Things) en la industria solo puede significar que el uso de los sitemas embebidos verá un crecimiento exponencial en los proximos años, el material estudiado durante la maestría le servirá al alumno para estar preparado para poder formar parte de este crecimiento.

Las áreas de aplicación posibles de los conocimientos adquiridos durante el estudio de una licenciatura son incontables. Por esta razón, un egresado de licenciatura puede contribuir con la sociedad por medio de un trabajo, que puede ser sumamente diferente al realizado por otro compañero egresado de la misma licenciatura. El estudio de una maestría permite especializar los conocimientos de un egresado de licenciatura en un área más particular de manera que el impacto en la sociedad del egresado de maestría sea mayor. Algo que me pareció muy relevante de las materias cursadas durante la mestría fue el hecho de que muchas de ellas, abordaban los temas de manera más práctica, se hacía mucho uso de la experiencia real de los profesores, lo cual junto con la experincia laboral que el alumno de maestria ya posee, se refleja en mejoras inmediatas en las aportaciones del alumno a su área laboral.

Apéndices

A. REPORTE DEL PROYECTO DE CREACIÓN DE LAYOUT DE UN SISTEMA ELECTRÓNCIO IMPLEMENTADO EN TECNOLOGÍA DE TARJETA DE CIRCUITO IMPRESO

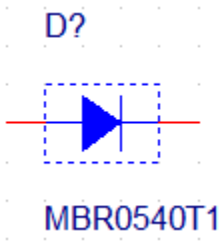
Reporte Final.- El propósito de este documento es el de servir de guía para realizar futuros proyectos donde se desee diseñar PCBs. Este documento describe además toda la metodología para realizar un PCB, de principio a fin, junto con puntos importantes a considerar y conclusiones.

Metodología para elaboración de un PCB

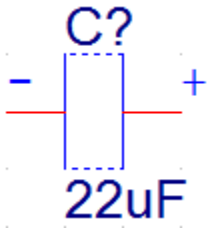
Parte 1. Elaboración de símbolos lógicos.

El primer paso consiste en la elaboración de los símbolos lógicos de todos los tipos de componentes que comprenden el diseño electrónico, incluyendo componentes pasivos, activos, conectores, etc. Es importante recalcar que en este punto todos los componentes deben tener su propio símbolo lógico, sin importar cuán parecidos sean a otro componente del mismo diseño. El símbolo lógico de una parte, indicará la cantidad de pines que tiene un componente, así como el tipo de cada uno de estos pines (salidas, entradas, bidireccionales, pasivos, de poder, etc.). Todo símbolo debe ir ligado a una huella o *footprint*, de la cual se hablará con más detalle más delante. Por último, es recomendable que cada símbolo lógico tenga una representación gráfica que sea representativa al componente al que se refiere, de manera que sea más fácil identificar los componentes al momento de capturar el esquemático. A continuación se muestran algunos símbolos lógicos para ejemplificar el punto anterior.

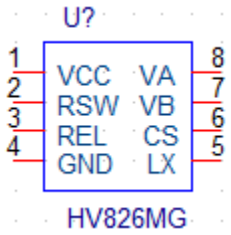
Este podría ser el símbolo lógico de un diodo:



De un capacitor:



De un circuito integrado:



Tip: Nótese que en el caso del capacitor, en vez de agregar el nombre del componente al símbolo lógico, como en el caso del circuito integrado, o del diodo, se le asignó el valor de su capacitancia, esto se recomienda en resistencias y capacitores, de nuevo, con el propósito de hacer el esquemático más entendible.

Tip: Se puede observar que en el símbolo lógico del diodo, el nombre de los pines no se muestra, ya que en realidad no es necesario mostrarlos para saber cuál pin es cuál, esto es bastante evidente gracias a la figura que se escogió para definir el diodo. Sin embargo, en el circuito integrado, si no se incluyera el nombre de los pines, la identificación de ellos sería imposible, por lo que se hace evidente que se requiere mostrar el nombre de los pines. Por último, aunque el símbolo del capacitor mostrado en el ejemplo, es un elemento pasivo que no tiene gran problema al conectarlo, debido a que este capacitor en particular tiene polaridad, entonces sí es necesario ya sea mostrar el nombre de los pines o poner algún indicador que nos ayude a determinar cuál es el pin positivo y cuál el negativo.

Tip: Se debe prestar mucha atención a que el pin del símbolo lógico corresponda exactamente con el mismo pin que muestra la hoja de especificaciones para cada componente. Por ejemplo, si en un componente el pin número 3 es de VCC, entonces en el símbolo lógico, el pin 3 deberá de ser VCC también, y así sucesivamente.

Tip: Se pueden agrupar los pines por funcionalidad aun cuando el componente físicamente no tenga los pines en ese orden siempre y cuando el número de pin y su función corresponda con el real de acuerdo a la hoja de especificaciones (de hecho es recomendable). De esta manera se facilita el uso de este símbolo cuando se utiliza en esquemáticos. Por ejemplo, una memoria, que tiene un bus de datos de 8 bits, podría no tener los pines físicamente colocados uno junto a otro, sin embargo se recomendaría hacer el símbolo lógico con estos pines agrupados para facilitar su conexión en esquemáticos.

A continuación se muestra el proceso para crear estos símbolos lógicos.

1. El primer paso es crear un proyecto nuevo, usando File -> New -> Project, y asignar un nombre al proyecto.

2. El siguiente paso es crear las librerías que contendrán las partes o símbolos lógicos requeridos para el diseño del circuito deseado. Esto se hace por medio de el menú File –> New -> Library. La elección de qué librerías crear, es libre, y está relacionado con la manera en que uno quiera organizar sus librerías de partes. Por ejemplo a continuación se muestra una manera en que se podrían organizar símbolos dentro de librerías:

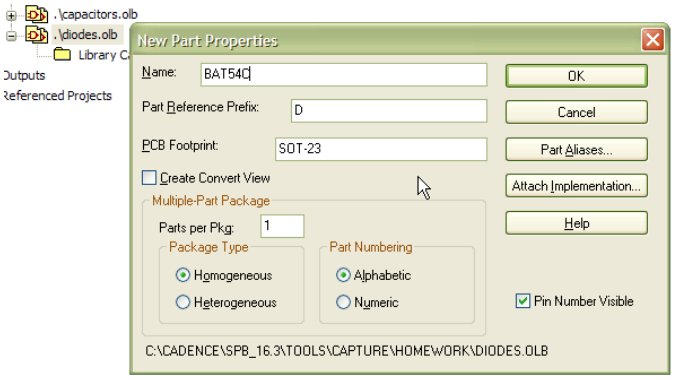
    - IC (Integrated Circuits)

- Capacitors
- Diodes
- Inductors
- Transistors
- Resistors
- Switches
- Connectors

3. Crear cada uno de los símbolos lógicos. El procedimiento para crear una nueva parte es el siguiente: click derecho sobre la librería que va a contener esa parte, y utilizar “New Part” o “*New Part From Spreadsheet*”. La opción de crear la parte desde una hoja de cálculo, nos da la flexibilidad de crear componentes de muchos pines, sin tener que definir manualmente pin por pin, sino que se pueden definir todos los pines, con su formato, tipo, si son de entrada, salida, etc. en una tabla, y después crear el componente completo de una sola vez.

   A continuación describo un poco la manera en que se crean las partes por los dos métodos.

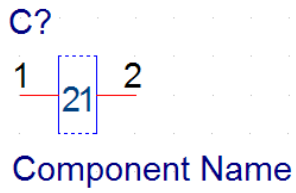
   - *New Part*: una vez abierta la ventana de *New Part Properties*, llenar los datos pertinentes, teniendo especial cuidado en incluir un nombre para la parte, el prefijo de la referencia de la parte (se sugiere seguir el estándar de U para circuitos integrados, R para resistencias, C para capacitores, etc.), y por último se debe seleccionar el footprint adecuado para la parte en construcción. El footprint requerido estará ligado al tipo de empaquetado del componente, este se puede obtener de la hoja de especificaciones de cada componente.



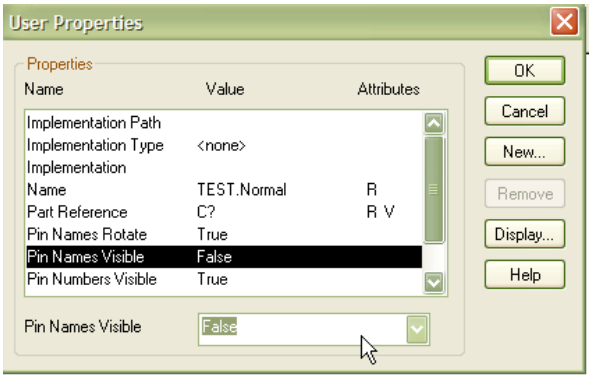
   - *New Part From Spreadsheet*: una vez abierta la tabla, se deberán vaciar los datos del componente considerando las columnas incluidas en la tabla. Este método solo se aconseja si la parte que se quiere crear contiene muchos pines.

C	Name	Type		Shape		Position
19	VCC	Power	1	Short		Left
2	V+	Input	1	Short		Left
6	V-	Input	1	Short		Left
7	TIN1	Input	1	Short		Left
8	TIN2	Input	1	Short		Left
9	TIN3	Input	1	Short		Left
14	RIN1	Input	1	Short		Left
13	RIN2	Input	1	Short		Left
12	VL	Input	1	Short		Left
20	S\H\D\N\	Input	1	Short		Left
1	C1+	Input	1	Short		Right
3	C1-	Input	1	Short		Right
4	C2+	Input	1	Short		Right
5	C2-	Input	1	Short		Right
17	TOUT1	Output	1	Short		Right
16	TOUT2	Output	1	Short		Right
15	TOUT3	Output	1	Short		Right
11	ROUT1	Output	1	Short		Right
10	ROUT2	Output	1	Short		Right
18	GND	Power	1	Short		Right

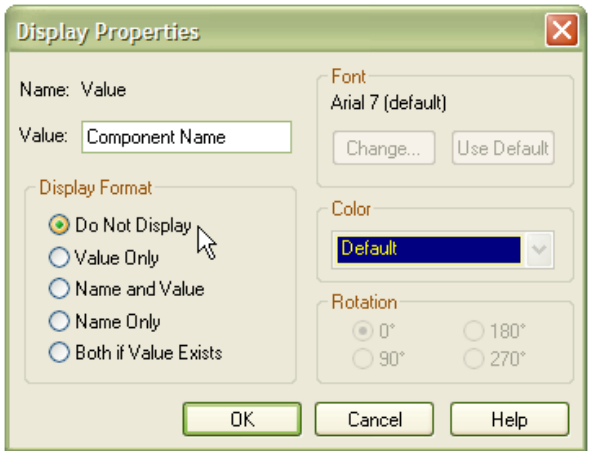
4. El último paso, es asignarle un símbolo adecuado a la parte, de manera que sirva como una ayuda visual al momento de hacer el esquemático. Tomaremos de ejemplo la creación de una parte simple, un capacitor. Al crear la parte del capacitor, se puede hacer el símbolo del capacitor fácilmente, de manera que la parte inmediatamente al verla se sepa qué tipo de componente es.



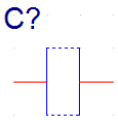
Para hacer aún más legible esta parte, se pueden esconder los números de los pines si así se desea, así como el nombre del componente, y el nombre del pin. En algunos casos esto será conveniente, en otros casos no. Para esconder nombres y números de pines se usa la siguiente herramienta:



Escondiendo nombre del componente:



Resultado final: una parte de componente más legible:

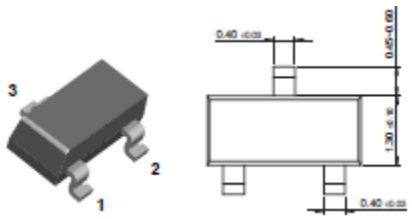


Parte 2. Elaboración de las huellas o *footprints*.

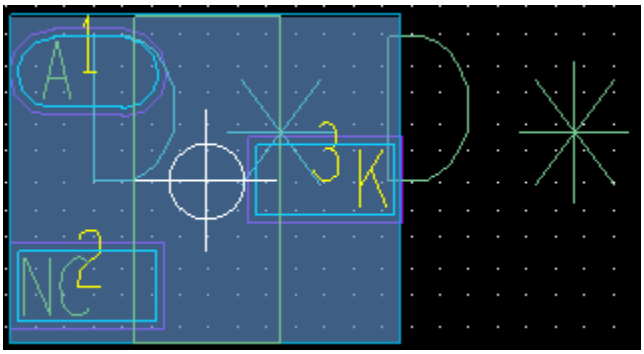
Cuando se coloca un componente en una tarjeta de PCB, es necesario no solo colocarlo, sino también soldar cada uno de sus pines a la tarjeta, de manera que haya conectividad entre las señales del componente y la tarjeta; aquí es donde entran los footprints. Los footprints están definidos por la ubicación física de cada uno de los pines en un componente así como su tamaño y superficie de contacto. Debido a que los footprints en realidad están relacionados con la forma física de los componentes, o sea con el empaquetado de este mismo, no es necesario crear un footprint para cada uno de los componentes en el diseño, sino que todos aquellos componentes que tengan el mismo

empaquetado pueden compartir el mismo footprint. Por ejemplo, un circuito podría utilizar varios tipos diferentes de diodos, pero si todos estos diodos están dentro de un empaquetado SOT-23, entonces solo se crea un footprint para el tipo de empaquetado SOT-23 y se le asigna a todos los símbolos lógicos de estos diodos. A continuación se muestran como ejemplo el footprint de un componente dentro de un empaquetado SOT-23:

Empaquetado físico del SOT-23:



Footprint del SOT-23:



En el footprint mostrado en la imagen anterior, se puede observar que los pines corresponden exactamente con los pines como se muestran en la hoja de especificaciones, tanto en acomodo, como en tamaño. Cuando digo que corresponden en tamaño no me refiero a que sean de el mismo tamaño, pero sí mantienen una proporción. En este momento es pertinente introducir un nuevo elemento de un footprint, este elemento es el *Padstack*. Un padstack es la superficie en la tarjeta en que quedará cobre para conectar un pin de un componente. Por lo tanto un footprint comúnmente estará formado de dos o más padstacks con posiciones relativas especificas entre ellos. En el ejemplo mostrado arriba, el footprint del empaquetado SOT-23 contiene 3 padstacks, uno por cada pin del empaquetado.

Tip: Los padstacks no son del mismo tamaño exacto de un pin, sino que su tamaño recomendado variará de acuerdo a varios factores: la superficie de contacto del pin, el tamaño del empaquetado, tipo de montaje del empaquetado, etc. En muchas ocasiones el fabricante de cada componente recomienda las dimensiones de footprint para su componente en las hojas de especificaciones de este. Si el fabricante proporciona estos valores, se recomienda ampliamente que se utilicen esos valores para la realización del padstack. Si el fabricante no especifica estos valores, entonces se recomienda utilizar alguna herramienta especializada en el cálculo de las dimensiones de estos padstacks y sus posiciones, por ejemplo la herramienta *LP Calculator* de la empresa *PCB Matrix*.

Tip: El Allegro tiene un wizard para asistir con la creación de footprints, se recomienda utilizarla siempre que sea posible (cuando el componente sea estándar).

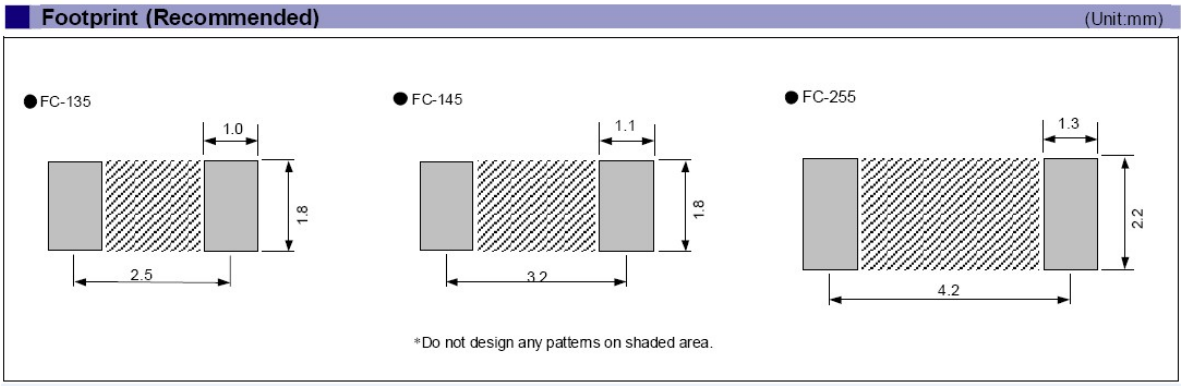
Tip: Si el footprint que se está realizando tiene múltiples pines, se recomienda identificar el pin 1 por medio de la utilización de un tipo diferente de padstack. Por ejemplo, si los padstacks utilizados son rectangulares, entonces se puede utilizar un padstack oblongo para identificar el pin 1.

Tip: Cuando se calculan y realizan los padstacks, se recomienda utilizar las unidades dominantes de las hojas de especificaciones, aun cuando presente más dimensiones. Por ejemplo, una hoja de especificaciones podría mostrar las dimensiones del componente en milésimas de pulgada, y en milímetros, pero si milímetros es la unidad dominante, entonces se deberá usar esta para calcular los padstacks.

A continuación se muestra el proceso para crear estos footprints.

1. El primer paso es la creación de los padstacks necesarios para cada tipo de paquete. Para la creación de estos padstacks, se recomienda el siguiente sub-proceso:
    A. Definir el tamaño del padstack.
        - En los casos en que el fabricante del componente proporciona dentro de las hojas de especificaciones, recomendaciones sobre las dimensiones de los padstacks, se deben utilizar estas dimensiones.

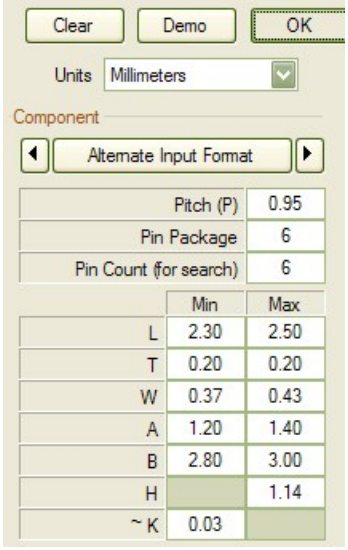
          Ejemplo: Para el componente Q13FC1350xxxx00 el cual usa un empaquetado FC-135, el fabricante proporciona las siguientes dimensiones.

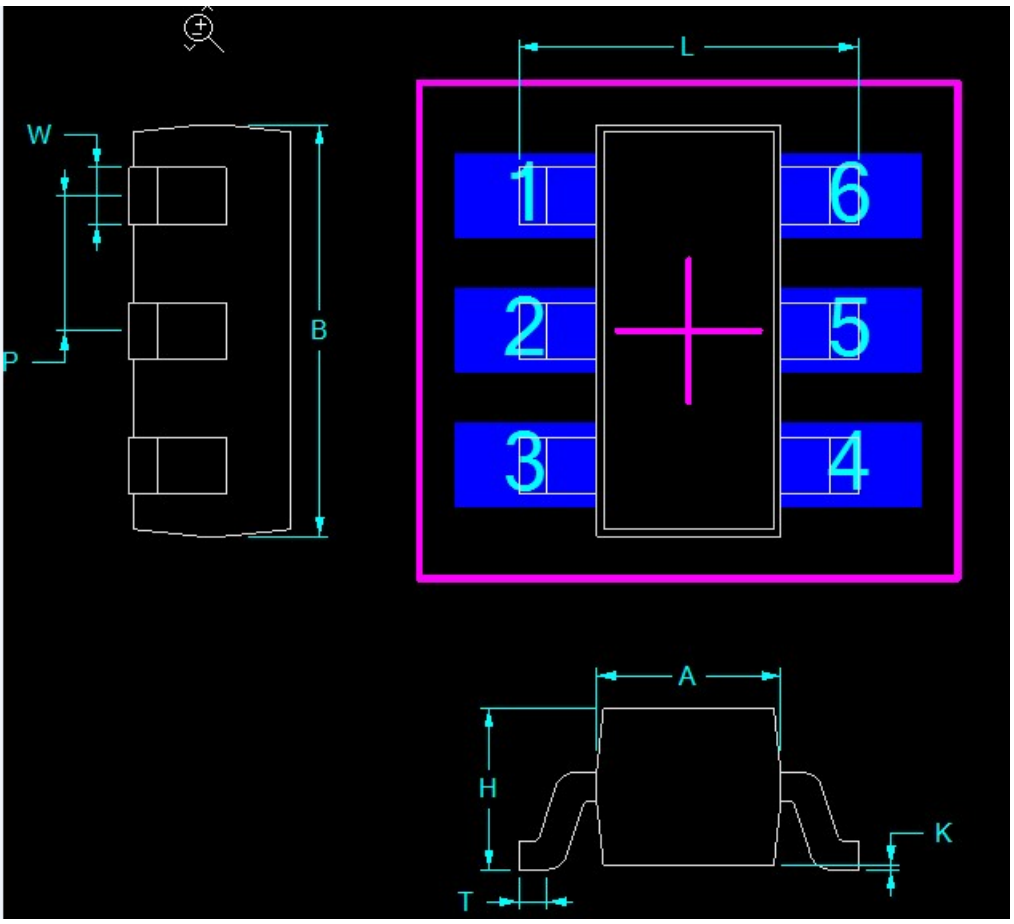


        - En aquellas ocasiones en que el fabricante no proporciona recomendaciones para las dimensiones de los padstacks, se deberá utilizar una herramienta, como LP Calculator para calcular estas dimensiones. Esta herramienta, permite seleccionar el tipo de componente al que se le quieren calcular los padstacks, se le introducen las dimensiones del componente como se proveen en las hojas del fabricante, y la herramienta nos proporciona las dimensiones recomendadas para los padstacks.

          Ejemplo:

Primero se proveen los valores de las dimensiones solicitadas:

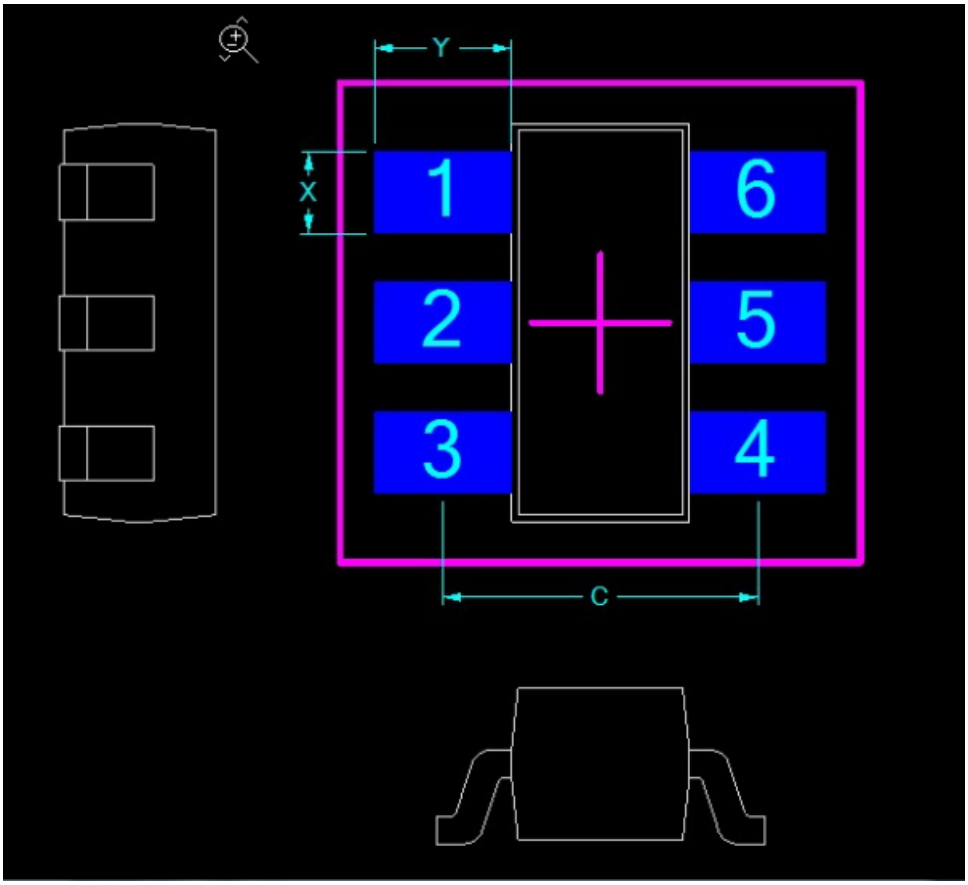




Y la herramienta nos proporciona los valores recomendados para los padstacks:

Dimensions

Land Space C	2.30
Land X	0.60
Land Y	1.00
Silkscreen R1	0.65
Silkscreen R2	2.90
Courtyard V1	3.80
Courtyard V2	3.50

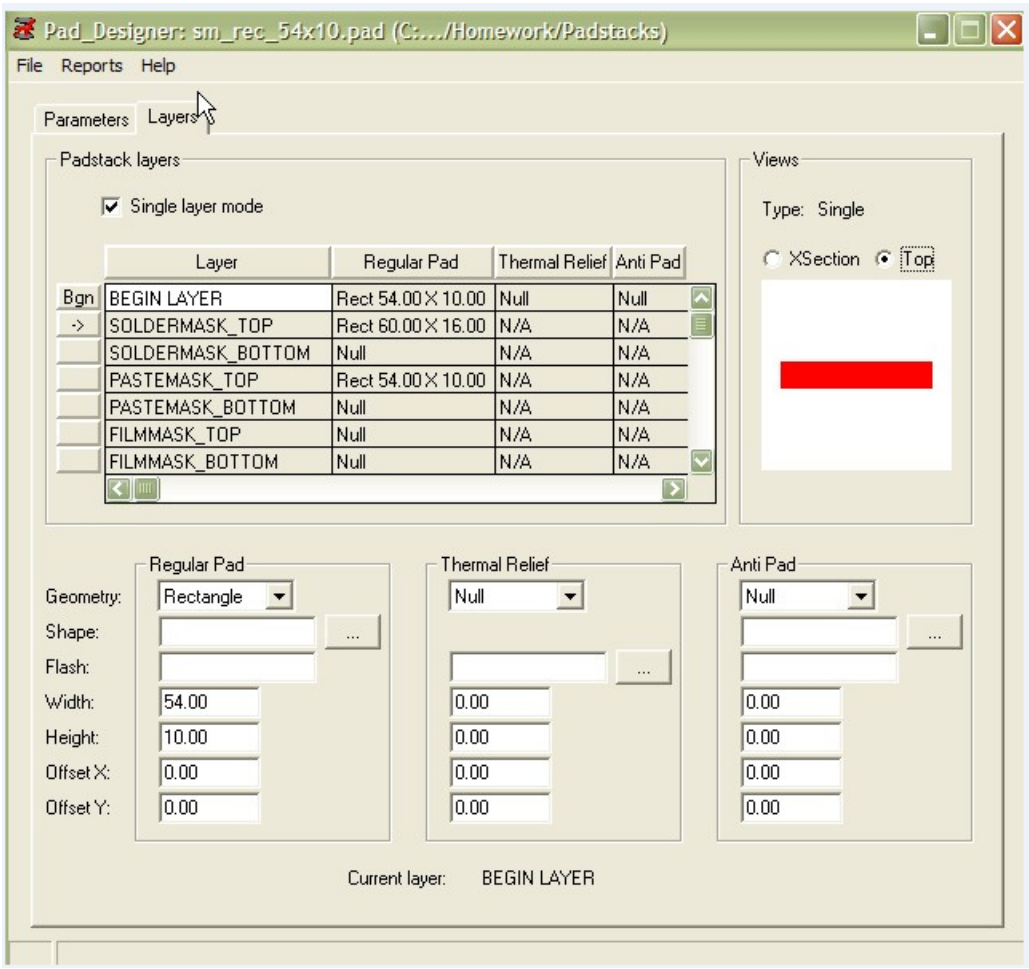


B. Creación del padstack.

Para la creación del padstack se utiliza la herramienta de Allegro llamada *Pad Designer*. Todos los padstacks contienen las siguientes capas:

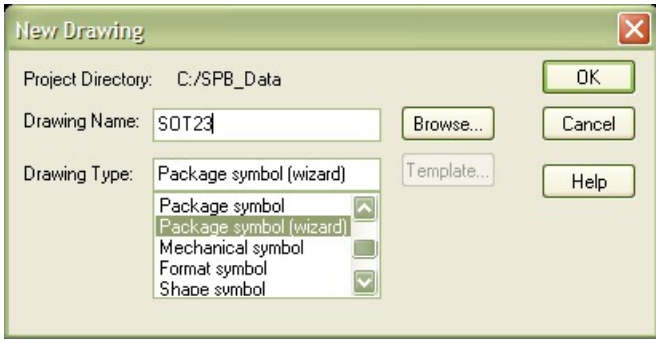
- Begin Layer (de las mismas dimensiones recomendadas por LP Calculator).
- Pastemask_Top (de las mismas dimensiones recomendadas por LP Calculator).
- Soldermask_Top (6 milésimas de pulgada más grande que la capa de begin_layer o que la de pastemask_top).

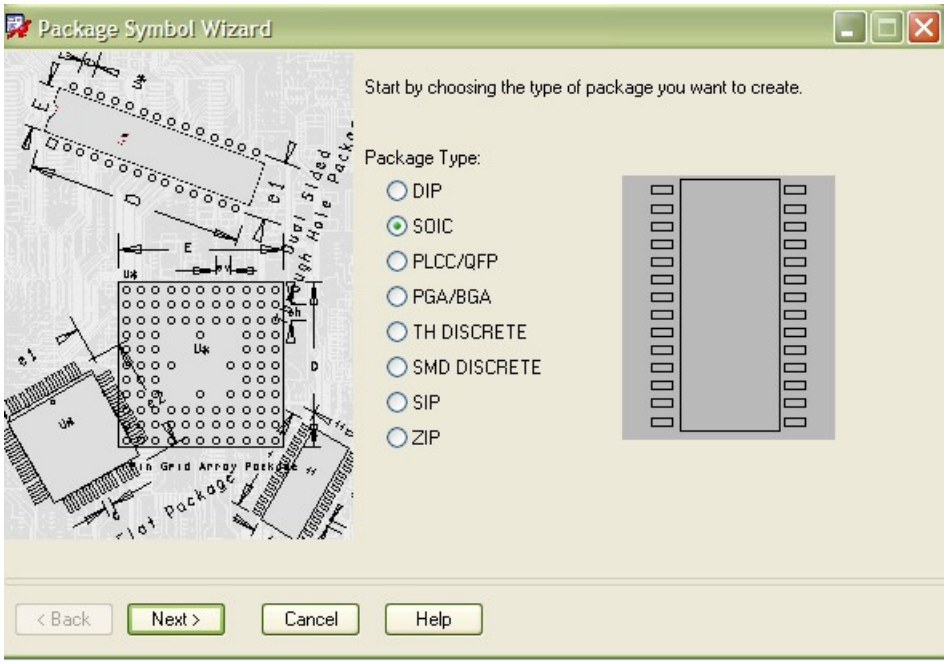
Ejemplo:

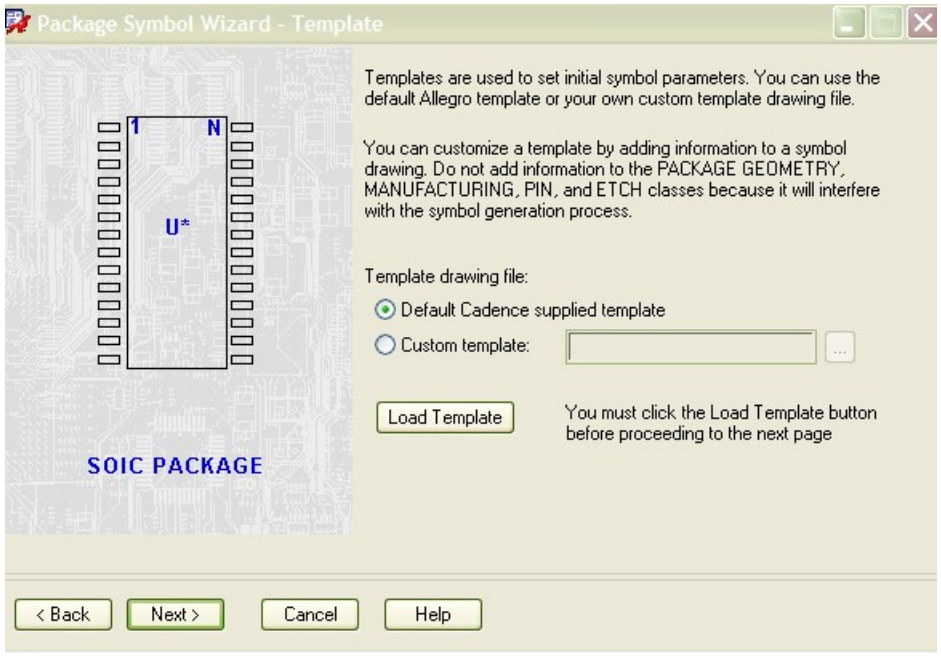


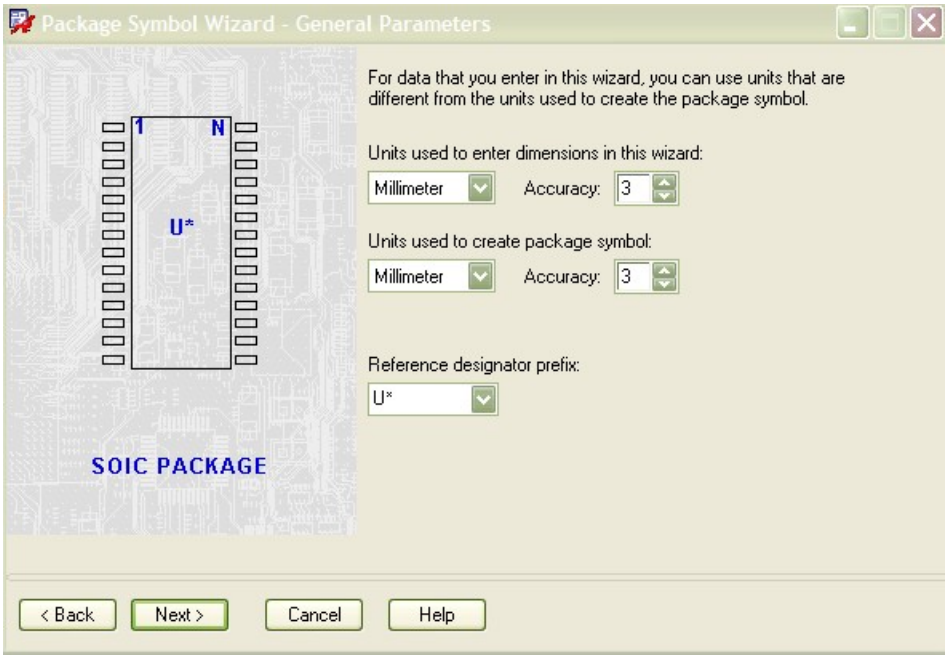
2. El siguiente paso es la creación del footprint para cada tipo de componente. Para la creación de footprints se utiliza la herramienta de Allegro llamada *PCB Editor*. Como se mencionó antes, cuando el tipo de componente es común, es recomendado utilizar el *New Part Wizard* de la herramienta, el cual nos permite crear el footprint, tan solo con introducir los datos y dimensiones que se solicitan, así como los padstacks que se quieren utilizar para esa parte (y que se crearon en el paso anterior).

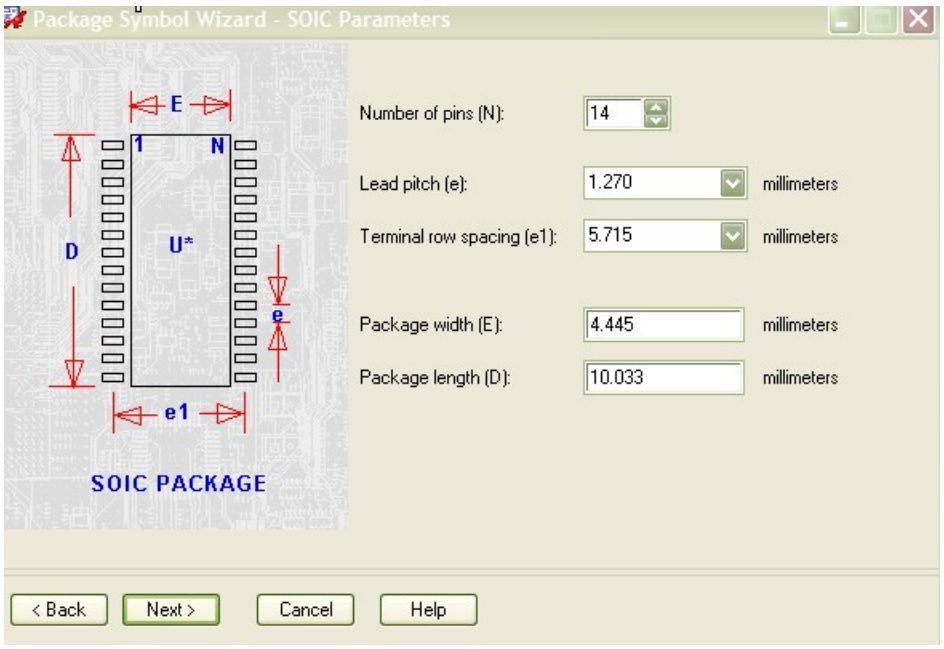
   Ejemplo:

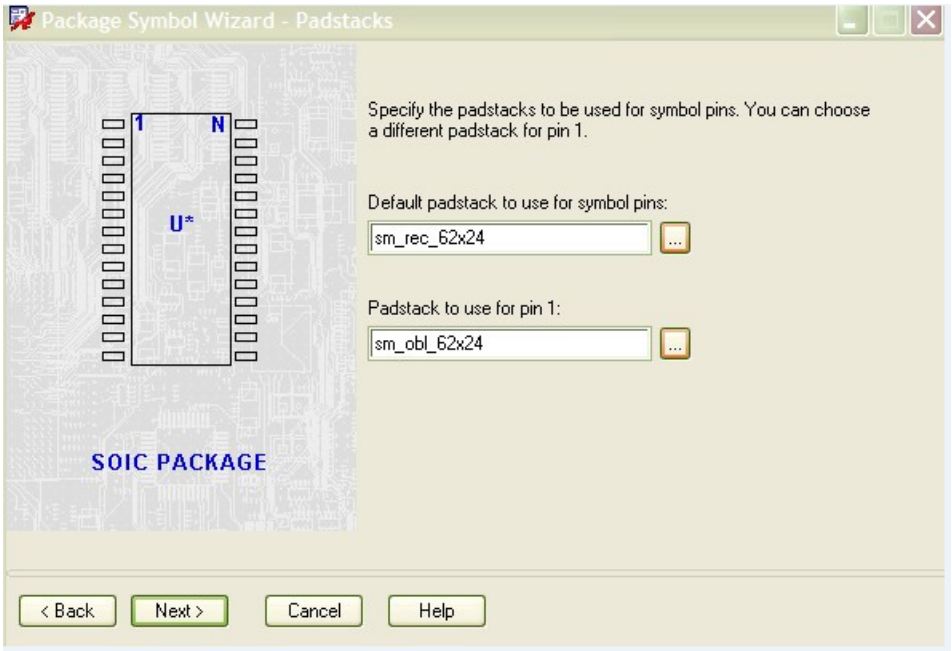

New Drawing
Project Directory: C:/SPB_Data
Drawing Name: SOT23
Drawing Type: Package symbol (wizard)
Package symbol
Package symbol (wizard)
Mechanical symbol
Format symbol
Shape symbol
Browse...
Template...
OK
Cancel
Help

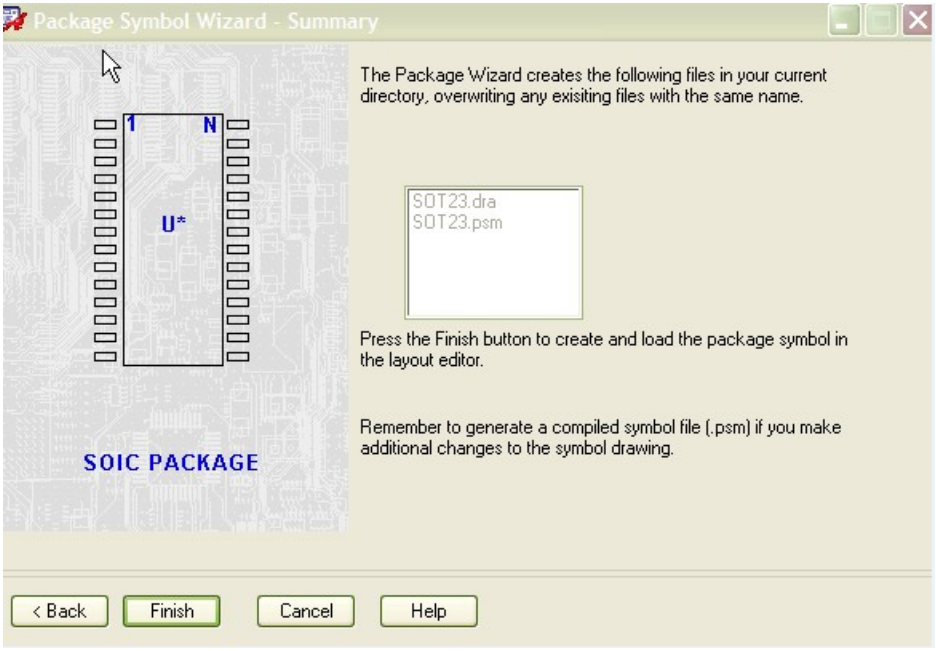

Package Symbol Wizard
Start by choosing the type of package you want to create.
Package Type:
DIP
SOIC
PLCC/QFP
PGA/BGA
TH DISCRETE
SMD DISCRETE
SIP
ZIP
< Back
Next >
Cancel
Help


Package Symbol Wizard - Template
1
N
U*
SOIC PACKAGE
Templates are used to set initial symbol parameters. You can use the default Allegro template or your own custom template drawing file.
You can customize a template by adding information to a symbol drawing. Do not add information to the PACKAGE GEOMETRY, MANUFACTURING, PIN, and ETCH classes because it will interfere with the symbol generation process.
Template drawing file:
Default Cadence supplied template
Custom template:
Load Template
You must click the Load Template button before proceeding to the next page
< Back
Next >
Cancel
Help


Package Symbol Wizard - General Parameters
1
N
U*
SOIC PACKAGE
For data that you enter in this wizard, you can use units that are different from the units used to create the package symbol.
Units used to enter dimensions in this wizard:
Millimeter
Accuracy:
3
Units used to create package symbol:
Millimeter
Accuracy:
3
Reference designator prefix:
U*
< Back
Next >
Cancel
Help


Package Symbol Wizard - SOIC Parameters
E
1
N
D
U*
e
e1
SOIC PACKAGE
Number of pins (N):
14
Lead pitch (e):
1.270
millimeters
Terminal row spacing (e1):
5.715
millimeters
Package width (E):
4.445
millimeters
Package length (D):
10.033
millimeters
< Back
Next >
Cancel
Help

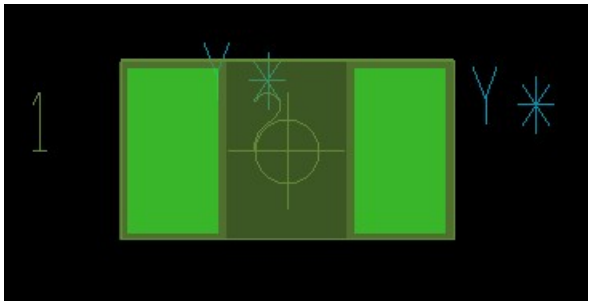

Package Symbol Wizard - Padstacks
1
N
U*
SOIC PACKAGE
Specify the padstacks to be used for symbol pins. You can choose a different padstack for pin 1.
Default padstack to use for symbol pins:
sm_rec_62x24
...
Padstack to use for pin 1:
sm_obl_62x24
...
< Back
Next >
Cancel
Help



Para aquellos casos en que la forma del componente no es típica, es necesario crear el footprint a mano. En estos casos, se pueden seguir estos pasos:

- Colocar manualmente los padstacks correspondientes para ese componente en las posiciones correctas de acuerdo a las hojas de especificaciones (utilizar el comando “x posicionX posición Y”.
- Crear un objeto con el siguiente tipo para delimitar la zona donde estará el componente: Package Geometry -> Silkscreentop -> type: unfilled
- Crear dos objetos con el siguiente tipo: Package Geometry -> Place Bound Top, Dfa_Bound_Top -> type: solid.
- Agregar los siguientes reference designators: Assembly top, Silkscreen top.

Ejemplo:



Parte 3. Captura del Esquemático.

El esquemático es el diagrama de interconexión de los componentes de un circuito, o en otras palabras, en el proceso de creación de un PCB, es el diagrama de interconexión de símbolos lógicos. Estos deben ser tomados de la librerías de símbolos lógicos creadas en la parte 1, y se deben interconectar entre sí para definir la manera en que interactúan. Este parte del proceso de diseño de PCB, es muy laboriosa pero es de las más sencillas.

Tip: Al capturar un esquemático se pueden utilizar tantas páginas de esquemático se requieran de manera que quede más modularizado y no tan amontonado. Las señales entre páginas se conectan unas con otras por medio de elementos llamados *off page connectors*.

Tip: Se recomienda nombrar todos los Nets del esquemático con un nombre significativo, si no se nombre específicamente, entonces la herramienta lo hace por nosotros pero utilizando nombres no tan fáciles de relacionar, por ejemplo N10025. Para nombrar los nets se utiliza la herramienta *Place Net Alias*.

Tip: Se recomienda llamar a los componentes en el esquemático de acuerdo a la página de esquemático en que están ubicados. Por ejemplo, todas las resistencias colocadas en la página 3 del esquemático deberán tener el siguiente formato de nombre: R301, R302, R303, etc. Donde el primer dígito indica la página en que esa resistencia está.

Tip: Se recomienda utilizar *Off Page Connectors* para interconectar señales entre páginas, en vez de utilizar *Ports*, ya que estos pueden causar problemas.

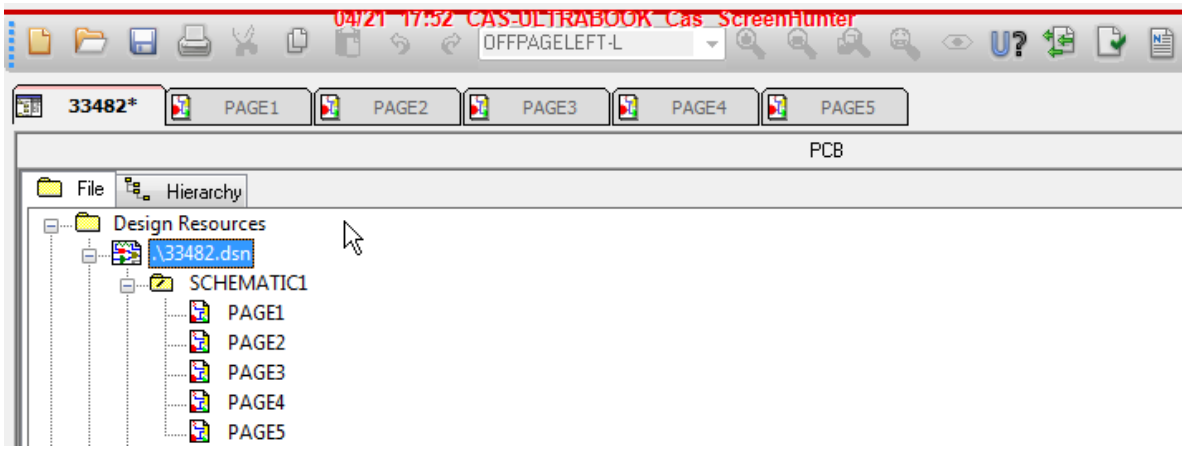
Tip: Si se hace algún cambio en algún símbolo lógico una vez que ya ha sido colocado en el esquemático, entonces es necesario actualizar la *Design Cache* de manera que el componente en el esquemático también sea actualizado.

A continuación se muestra el proceso para hacer la captura del esquemático.

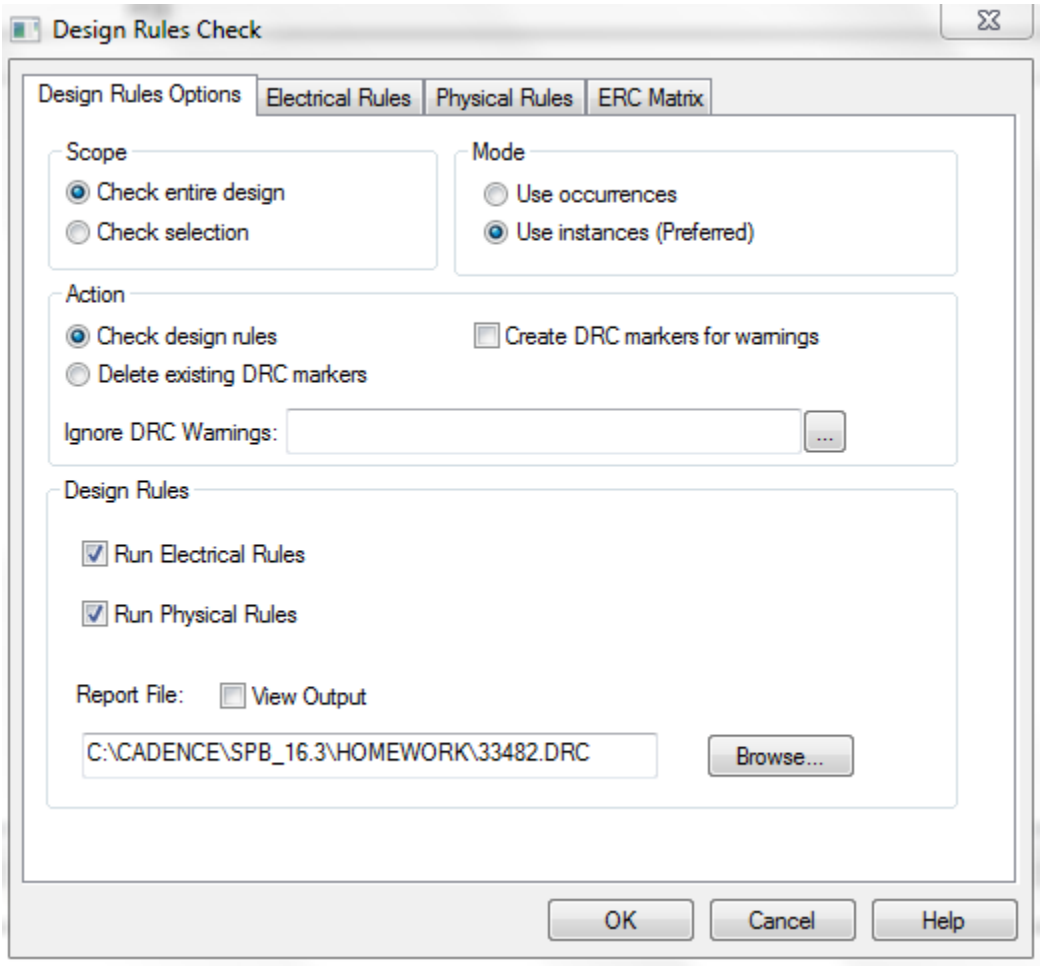
1. Si se tiene un esquemático de referencia, se empieza a capturar el esquemático tratando de hacerlo lo más parecido al esquemático de referencia, incluyendo las etiquetas de los componentes, de esta manera, si hay algún problema, es más fácil determinar dónde puede estar el problema.

2. Una vez que se terminó de capturar todo el esquemático, es necesario ejecutar la herramienta llamada *Design Rules Check*, la cual hace una revisión en todo el esquemático, basándose en varias reglas eléctricas. Esta herramienta nos indicará advertencias y errores que deben ser corregidos antes de que se pueda continuar con el diseño. Para ejecutar esta herramienta se sigue el siguiente proceso:

   Ir a la ventana principal del proyecto y seleccionar el archivo del proyecto.



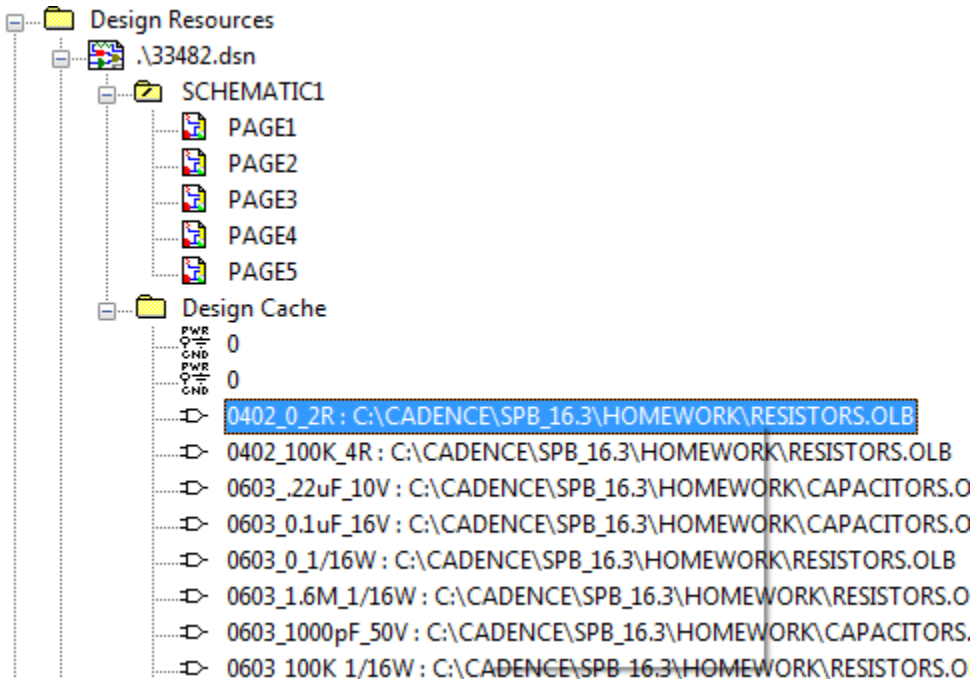
Y utilizar la herramienta siguiente:



Si el esquemático aun contiene errores el DRC lo mostrará:

```
********************************************************************************
*
*  Design Rules Check
*
********************************************************************************

-------------------------------------------------
Checking Schematic: SCHEMATIC1
-------------------------------------------------
Checking Electrical Rules

Checking For Single Node Nets

Checking For Unconnected Bus Nets

Checking Off-Page Connections

Checking Pin to Port Connections

Checking Physical Rules

Checking Pins and Pin Connections
```

3. Si al momento de correr la herramienta para checar las reglas de diseño, esta nos arroja que tenemos errores que requieran alguna modificación a algún símbolo lógico, entonces para corregir este problema será necesario hacer modificaciones al símbolo. Como se mencionó antes, el símbolo después de ser modificado, y se han guardado los cambios, es importante que si el símbolo ya fue usado en el esquemático, entonces se actualice la cache del proyecto para ese símbolo de manera que el símbolo anterior sea sustituido por el nuevo con los cambios.



Parte 4. Realización del *Layout* del PCB.

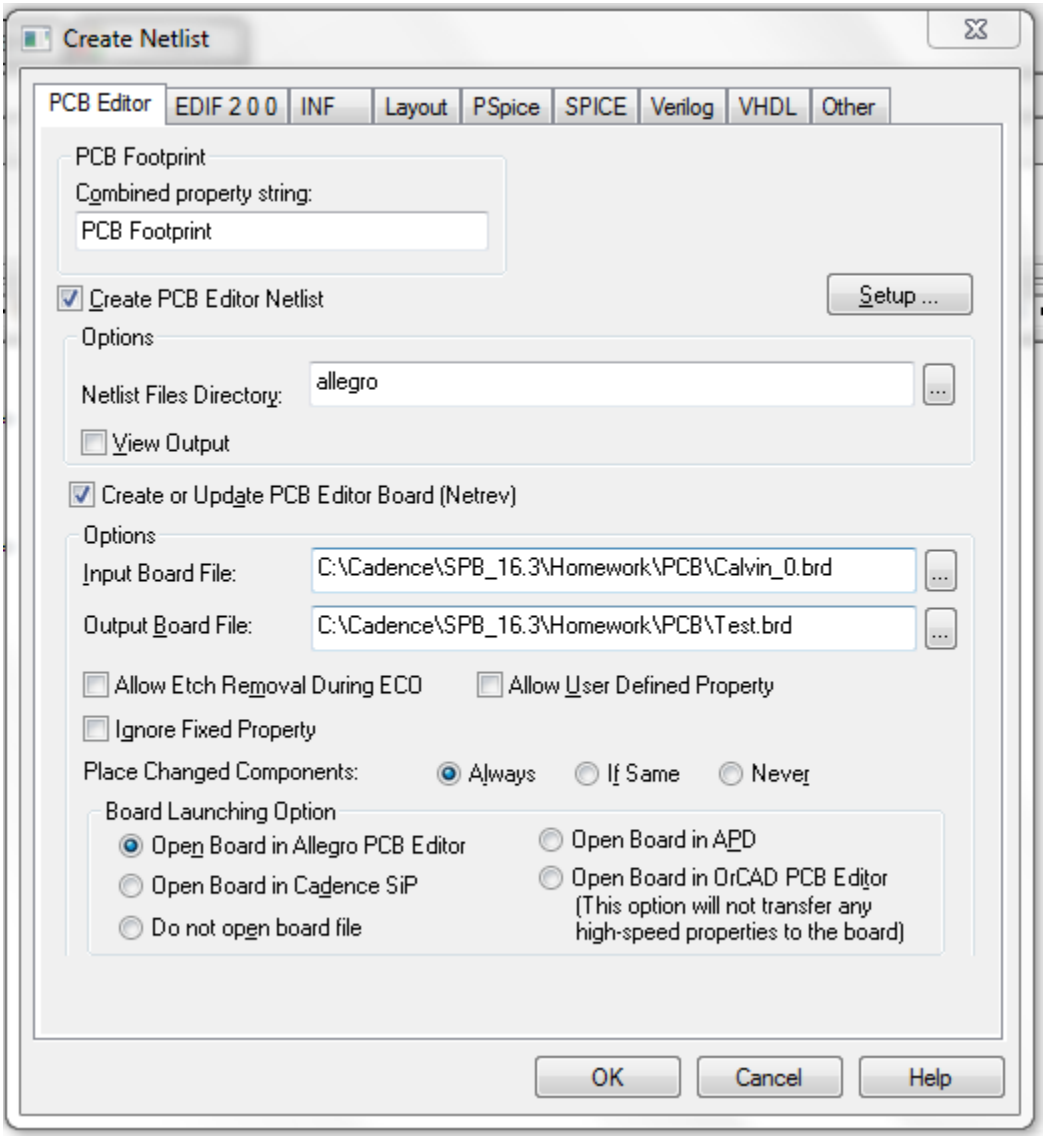
Las actividades que se han mostrado hasta este punto, son en cierta manera de “preparación” para lo que es ya propiamente dicho el diseño del PCB. A este proceso se le llama *layout,*y consiste en colocar los footprints de los componentes en la tarjeta, en la ubicación donde irán los componentes soldados, y realizar los trazos o líneas de transmisión que conectarán las diferentes señales del circuito. El proceso, básicamente se podría resumir en 4 partes principales: El placement, definición de constraints, el fanout de los componentes, y el ruteo.

Tip: Cuando se colocan los componentes (placement), es importante asegurarse de dejar espacio suficiente para poder hacer el ruteo para interconectar los componentes.

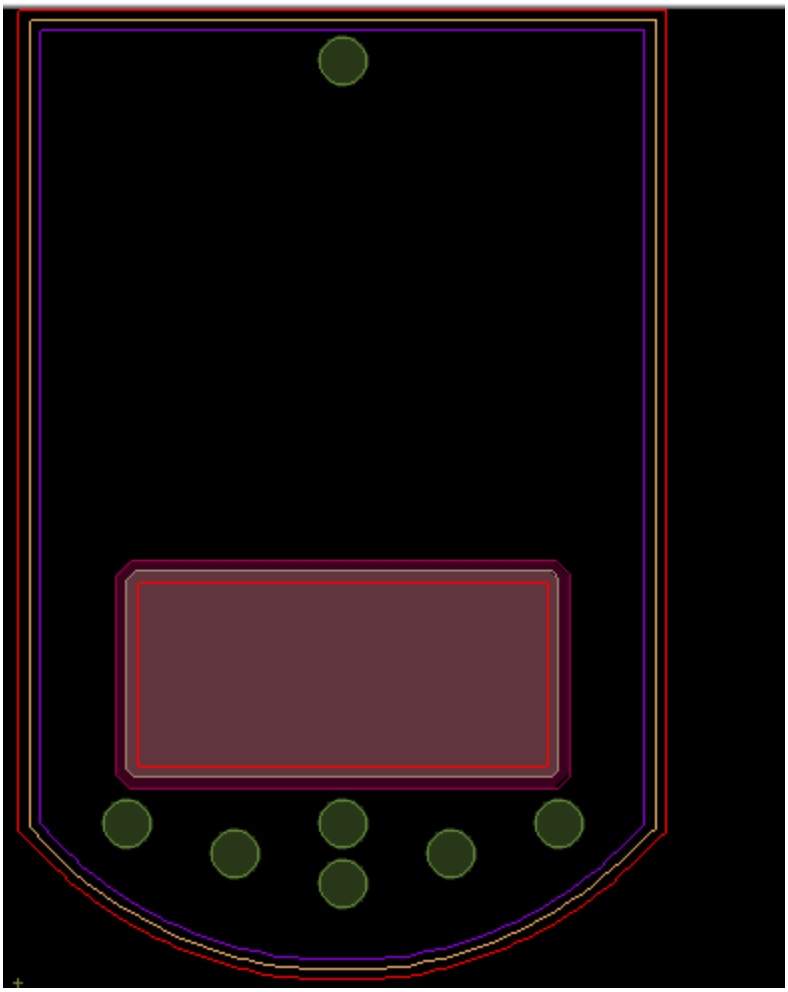
Tip: Cuando se realicen los trazos para interconectar componentes (routing), se recomienda comenzar por los componentes más delicados, con más señales o con más constraints.

A continuación se muestra el proceso para crear realizar el layout.

1. Una vez que completada la captura del esquemático, y que el DRC no encuentra errores se deberá usar la herramienta de *Create Netlist*, esta herramienta exportará los componentes y los nets del esquemático al board que se utilizará para desarrollar el PCB. En este momento se debe utilizar una plantilla de tarjeta como entrada, la cual contiene la forma de la trjeta y las capas, etc. Se generará un segundo archivo .brd mezclando la plantilla de entrada y los *footprints* de los componentes y nets del esquemático.

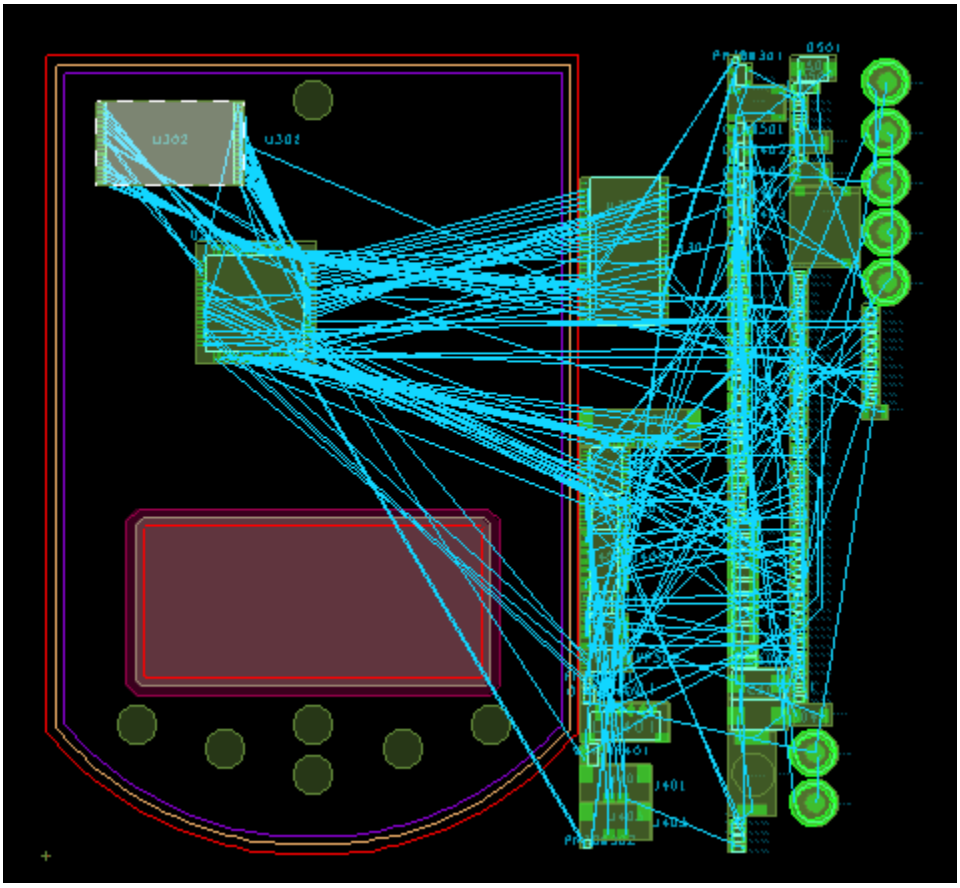


La plantilla de la tarjeta es cargada:



2. Ahora es necesario comenzar a colocar los componentes sobre la tarjeta, a este proceso se le conoce como *placement*. Se puede utilizar tanto la parte de arriba como la de abajo para colocar componentes. Se comienza por hacer un quickplace de todos los componentes en el área de trabajo para después comenzar a moverlos a sus posiciones sobre o debajo de la tarjeta. Para saber dónde colocar los componentes se utiliza algo llamado *ratsnest*, los cuales son solo líneas que nos indican qué pines se conectan con qué, y de esta manera tener una ayuda visual para la colocación..

   La imagen a continuación muestra una tarjeta y zona de trabajo después del *quickplace*, también se pueden ver los ratsnest, y la manera en que se comienzan a acomodar los componentes:



3. Una vez colocados todos los componentes, se deben crear los las reglas para el ruteo. A este set de reglas se les conoce como *constraints,* y para administrar estos constraints se usa una herramienta llamada *constraint manager.* Se pueden configurar reglas físicas, las cuales determinan los anchos de las pistas dependiendo de la capa donde se trazara la pista, o dependiendo de la señal que va a transmitir, ya que las señales requieren diferentes impedancias para no corromperse. También están las reglas de espaciamiento entre señales, esto también para evitar interferencia entre señales. Por último existen las reglas eléctricas, estas reglas indican los tamaños mínimos o máximos que deben tener las pistas con respecto a otras o con respecto a la señal que transmiten, de nuevo, esto para mantener la integridad de la señal.

   En la siguiente imagen se observa un ejemplo de reglas físicas, define el ancho de pista para las diferentes capaz del PCB, además define el tamaño de las VIAS que se usan de nuevo dependiendo de la señal que van a contener.

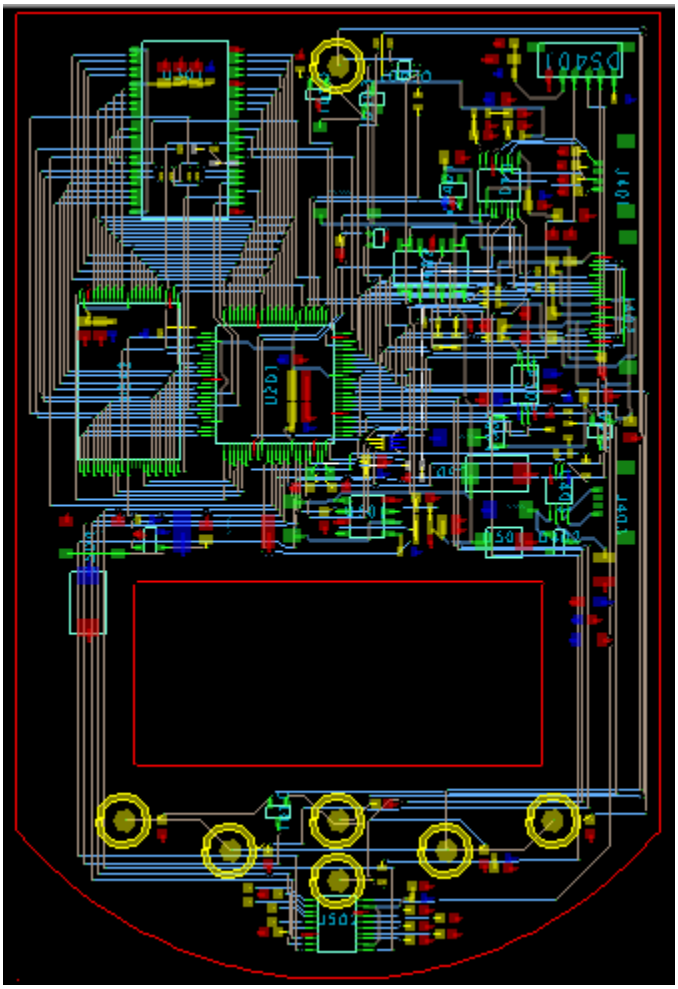
Type	Objects	Line Width		Neck		Differential Pair					Vias
		Min	Max	Min Width	Max Length	Min Line Spac	Primary Gap	Neck Gap	(+)Tolerance	(-)Tolerance	
		mil	mil	mil	mil	mil	mil	mil	mil	mil	
*	*	*	*	*	*	*	*	*	*	*	*
Dsn	⊟ Calvin_5	4.00:4.50:4.50...	4.00:4.50:4.50...	4.00:4.50:4.50...	4.00:4.50:4.50...	0.00	0.00	0.00	0.00	0.00	VIA_20-10
PCS	⊞ DEFAULT	4.00:4.50:4.5...	4.00:4.50:4.5...	4.00:4.50:4.5...	4.00:4.50:4.5...	0.00	0.00	0.00	0.00	0.00	VIA_20-10
PCS	⊞ DIFFERENTIAL	5.00	5.00	5.00	5.00	0.00	7.00:6.50:6.5...	7.00:6.50:6.5...	0.00	0.00	VIA_20-10
PCS	⊞ POWER	20.00	20.00	10.00	200.00	0.00	0.00	0.00	0.00	0.00	VIA_26-14
PCS	⊟ SINGLE_ENDED	6.50:8.00:8.0...	6.50:8.00:8.0...	6.50:8.00:8.0...	6.50:8.00:8.0...	0.00	0.00	0.00	0.00	0.00	VIA_20-10
Lyr	TOP	6.50	6.50	6.50	6.50	0.00	0.00	0.00	0.00	0.00	
Lyr	L2_GND	8.00	8.00	8.00	8.00	0.00	0.00	0.00	0.00	0.00	
Lyr	L3	8.00	8.00	8.00	8.00	0.00	0.00	0.00	0.00	0.00	
Lyr	L4	8.00	8.00	8.00	8.00	0.00	0.00	0.00	0.00	0.00	
Lyr	L5_PWR	8.00	8.00	8.00	8.00	0.00	0.00	0.00	0.00	0.00	
Lyr	BOTTOM	6.50	6.50	6.50	6.50	0.00	0.00	0.00	0.00	0.00	

.

4. Una vez que todos los costraints están listos, el siguiente paso es el trazar las pistas para interconectar todos los pines de los componentes con sus respectivos componentes de acuerdo al esquemático. Para hacer esto más sencillo se puede hacer uso de los *rats* los cuales, como ya se mencionó antes, son las líneas de interconexión que se muestran de ayuda para saber qué va conectado con qué. Algunas de las recomendaciones al momento de hacer este laborioso proceso son las siguientes:

   A) Comenzar primero con los componentes que tienen más constraints y más señales. En este caso por ejemplo serían las memorias y el procesador.
   B) Si se tienen varias capas, se puede utilizar una capa o layer para hacer los trazos que van en una misma dirección, por ejemplo horizontales, y otro layer para los trazos que van en otra dirección, por ejemplo verticales.
   C) Si se tienen varios layers, se recomienda de preferencia hacer los trazos por los layers interiores, de manera que se proteja la integridad de las señales del ruido exterior.

   La siguiente figura muestra un ejemplo de diseño después de terminar el ruteo.



Aprendizajes y Conclusiones

El proceso de diseño de un PCB de principio a fin, es complicado, pero sobre todo es un proceso muy laborioso. Es necesario cuidar hasta el más mínimo detalle durante el proceso de desarrollo ya que un error en etapas tempranas del proyecto puede ocasionar problemas más grandes en una parte avanzada, donde la reparación de ese problema podría significar mucho retrabajo. Por ejemplo, una selección equivocada de constraints, podría invalidar todo el ruteo que se hace en una tarjeta, forzando al diseñador a comenzar de nuevo. Todas las recomendaciones basadas en mis principales problemas durante el desarrollo del proyecto las fui tocando durante el dearrollo del reporte, por lo que no es necesario que las vuelva a mencionar en esta parte.

B. REPORTE DEL PROYECTO DE APLICACIÓN DE INGENIERIA DE SOFTWARE EN AMBIENTES EMBEBIDOS PARA EL DISEÑO DE UN SPI

Serial Peripheral Interface SPI

SPI Overview

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard, named by Motorola, which operates in full duplex mode. A SPI bus is a master slave multi node bus system, the master sets a Chip Select (CS) to select a slave for data communication. The SPI (Serial Peripheral Interface) has a 4-wire synchronous serial interface. Data communication is enabled with a Chip Select wire (CS). Data is transmitted with a 3-wire interface consisting of wires for serial data input (MOSI), serial data output (MISO) and Serial Clock (SCK).

Operation

The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. With multiple slave devices, an independent SS signal is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance (disconnected) when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

Applications

The full-duplex capability makes SPI very simple and efficient for single master/single slave applications. Some devices use the full-duplex mode to implement an efficient, swift data stream for applications such as digital audio, digital signal processing, or telecommunications channels, but most off-the-shelf chips stick to half-duplex request/response protocols.

SPI is used to talk to a variety of peripherals, such as:

- Sensors: temperature, pressure, ADC, touchscreens, video game controllers
- Control devices: audio codecs, digital potentiometers, DAC
- Camera lenses: Canon EF lens mount
- Communications: Ethernet, USB, USART, CAN, IEEE 802.15.4, IEEE 802.11, handheld video games
- Memory: flash and EEPROM
- Real-time clocks
- LCD displays, sometimes even for managing image data
- Any MMC or SD card (including SDIO variant)

Requirements Specification

The RS is a specification for a particular system, software product, program, or set of programs that performs certain functions in a specific environment. The requirements specification may be written by one or more representatives of the supplier, one or more representatives of the customer, or by both.

The basic issues that the SRS writer(s) shall address are the following:

a) Functionality. What is the system supposed to do?
b) External interfaces. How does the system interact with people, the systems hardware, other hardware, and other software?
c) Performance. What is the speed, availability, response time, recovery time of various system functions, etc.?
d) Attributes. What are the portability, correctness, maintainability, security, etc. considerations?
e) Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

Conventions used

In requirements, the following specific semantics are used:

The key words "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", and "CAN" in this document are to be interpreted as described below. Note that the requirement level of the document in which they are used modifies the force of these words.

- SHALL: This word means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHOULD: This word means that there may exists valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- CAN: This word means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the

product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

Functional Requirements of the SPI

R1. The SPI **shall** be able to transmit data between two devices (one "Master" and one "Slave") or more (commonly one "Master" and various "Slaves").
R2. The SPI communication **shall** always be initiated by the master.
R3. The slave device(s) **shall** always be waiting for the signal from the master to start communicating.
R4. The SPI communication **shall** be full duplex.
R5. The SPI communication **shall** be synchronous.
R6. The SPI communication **shall** be serial.
R7. The SPI **shall** communicate at a variable speed that is pre-defined before establishing the communication at a baud rate of up to 1 Mega baud (10 Mbps).
R8. The SPI shall offer the possibility of configuring the transmission data width in the range of 8 to 32 bits (8, 16 or 32 bits).
R9. The SPI **shall** have a configurable clock polarity and phase with respect to the data. This polarity and phase will determine the edges of the clock signal on which the data are driven and sampled.
R10. The SPI **shall** provide ability for the master to select to which slave to send the data and read the data from if more than one slave exist.
R11. The SPI **should** provide ability for the slave to drive its output to high impedance state (disconnected) when the device is not selected, so many slaves can share the same master bus without interfering with each other.
R12. The SPI **shall** support all controller peripherals, which are capable of performing the SPI functionality (data in/ data out/ clock+ optional chip select signal).
R13. The SPI **shall** allow the static configuration of all software and hardware properties related to SPI. The following list is a list of proposed properties:
- Chip select pin polarity high or low
- Baud rate
- Timing between clock and chip select
- data width (1 up to 32 bits)
- shift clock idle low or idle high
- data shift with leading or trailing edge

R14. The SPI **shall** support access to transferred data (read /write) related to a certain HW device independent of the HW configuration.

R15. If new data is attempted to be transmitted while the SPI is still transmitting/receiving the previous data, the communication **shall not** be interrupted, and the SPI **shall** inform an error has occurred.

Acronyms

Acronym:	***Description:***
CS	Chip Select
ECU	Electric Control Unit
MCU	Micro Controller Unit
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
Master	A device controlling other devices (slaves, see below)
Slave	A device being completely controlled by a master device
SS	Slave Select
SCK	SPI Clock

References

- Serial Peripheral Interface Bus (http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)
- Requirements on SPI Handler/Driver V2.0.5 R4.0 Rev 1 (http://www.autosar.org/download/R4.0/AUTOSAR_SRS_SPIHandlerDriver.pdf)
- SPI Block Guide V03.06 (http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf)
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requireme

Serial Peripheral Interface SPI

SPI Overview

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard, named by Motorola, which operates in full duplex mode. A SPI bus is a master slave multi node bus system, the master sets a Chip Select (CS) to select a slave for data communication. The SPI (Serial Peripheral Interface) has a 4-wire synchronous serial interface. Data communication is enabled with a Chip Select wire (CS). Data is transmitted with a 3-wire interface consisting of wires for serial data input (MOSI), serial data output (MISO) and Serial Clock (SCK).

Operation

The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. With multiple slave devices, an independent SS signal is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance (disconnected) when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

Data Transmission

To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1–100 MHz.

The master then transmits the appropriate chip select bit for the desired chip to a logic 0. A logic 0 is transmitted because the chip select line is active low, meaning its off state is a logic 1; on is asserted with a logic 0. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line
- The slave sends a bit on the MISO line; the master reads it from that same line

Not all transmissions require all four of these operations to be meaningful but they do happen.

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data and the process repeats.

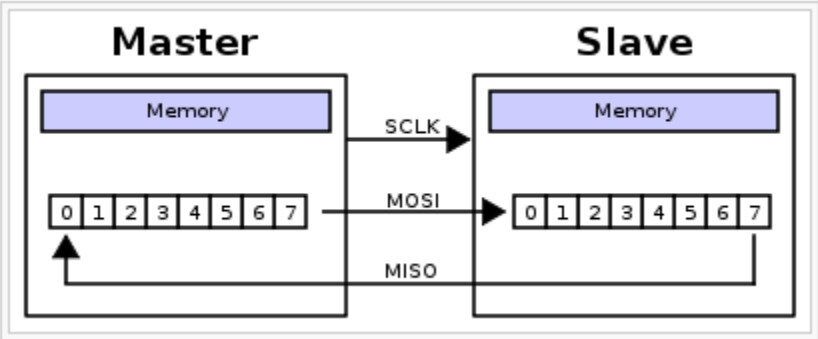


Figure1. A typical hardware setup using two shift registers to form an inter-chip circular buffer.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it wishes/needs. However, other word sizes are also common, such as 16-bit words.

Every slave on the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

Clock polarity and phase

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Motorola named these two options as CPOL and CPHA respectively, and most vendors have adopted that convention.

The timing diagram is shown below:

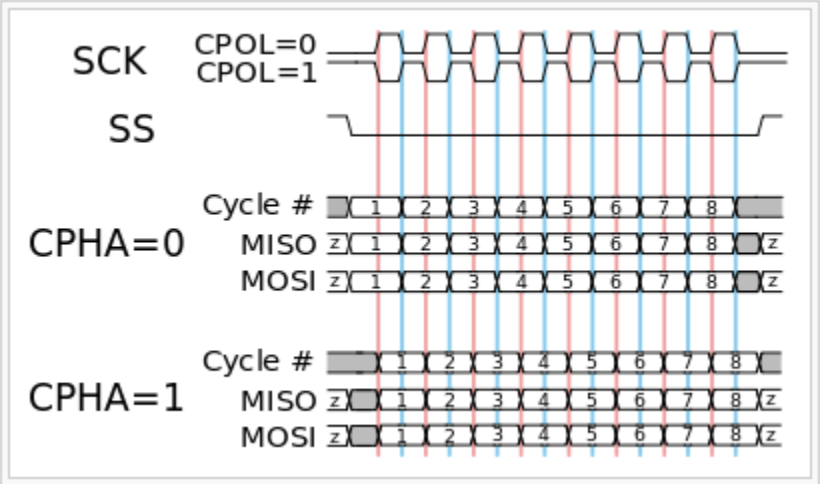


Figure 2. A timing diagram showing clock polarity and phase.

At CPOL=0 the base value of the clock is zero

- For CPHA=0, data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition).
- For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.

At CPOL=1 the base value of the clock is one (inversion of CPOL=0)

- For CPHA=0, data is captured on clock's falling edge and data is propagated on a rising edge.
- For CPHA=1, data is captured on clock's rising edge and data is propagated on a falling edge.

That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active.

The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle.

Mode numbers

The combinations of polarity and phases are often referred to as modes which are commonly numbered according to the following convention, with CPOL as the high order bit and CPHA as the low order bit:

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Figure 3. Possible modes for SPI.

Architecture and Design

Software Architecture

Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."
Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk. Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements. The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals. For each of these areas, you should outline key scenarios and identify important quality attributes (for example, reliability or scalability) and key areas of satisfaction and dissatisfaction. Where possible, develop and consider metrics that measure success in each of these areas.

Architecture focuses on how the major elements and components within an application are used by, or interact with, other major elements and components within the application. The selection of data structures and algorithms or the implementation details of individual components are design concerns. Architecture and design concerns very often overlap. Rather than use hard and fast rules to distinguish between architecture and design, it makes sense to combine these two areas. In some cases, decisions are clearly more architectural in nature. In other cases, the decisions are more about design, and how they help you to realize that architecture.

Consider the following high level concerns when thinking about software architecture:

- How will the users be using the application?
- How will the application be deployed into production and managed?
- What are the quality attribute requirements for the application, such as security, performance, concurrency, internationalization, and configuration?
- How can the application be designed to be flexible and maintainable over time?
- What are the architectural trends that might impact your application now or after it has been deployed?

The following image illustrates common software application architecture with components grouped by different areas of concern.

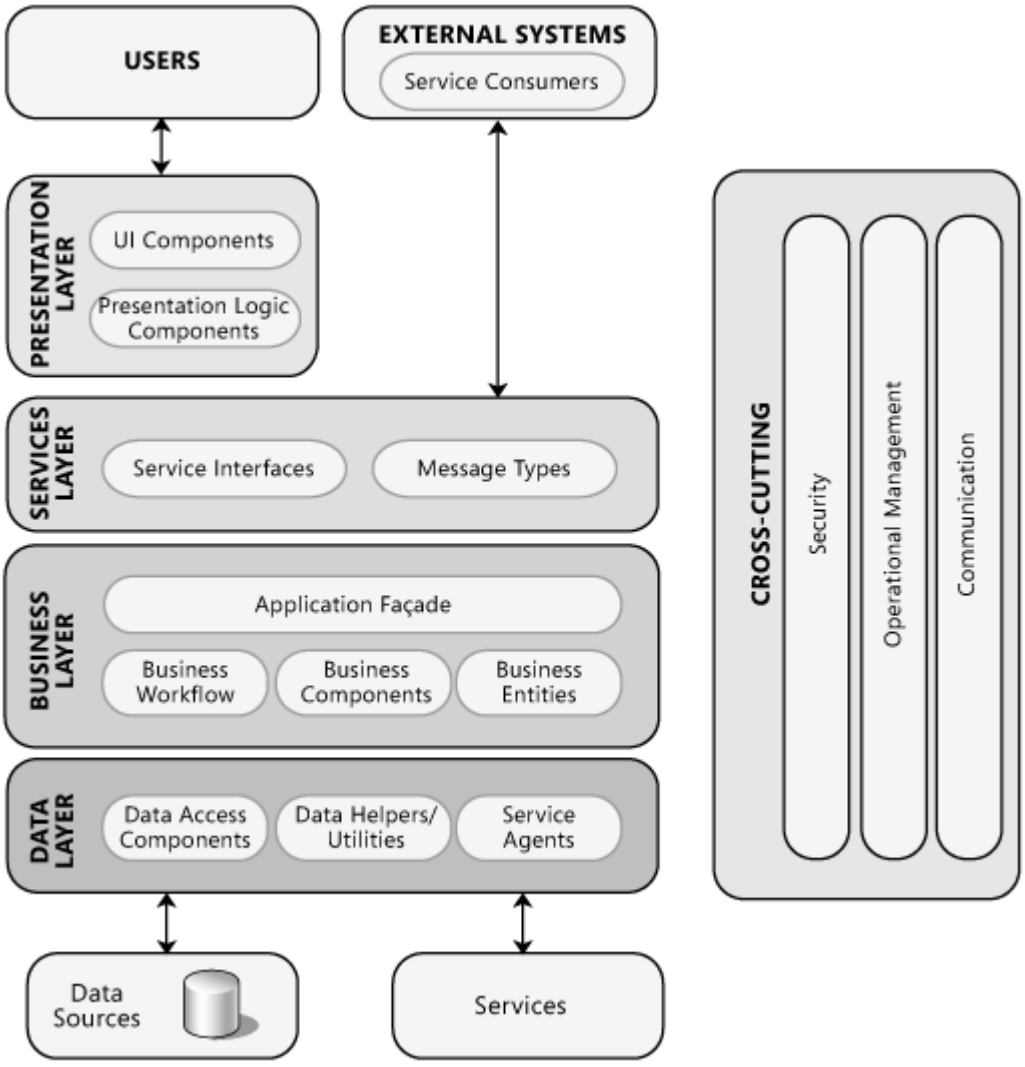


Figure 4. Common software application architecture.

The architecture shown above works great for pure software applications, but since the SPI module is an embedded software design, meaning that it is a hardware software design, some modifications to the suggested architecture needs to be done to fit with the kind of project.

The following layered software architecture structure is suggested by Autosar (Automotive Open System Architecture). This structure considers the hardware part of the system.

The AUTOSAR Architecture distinguishes on the highest abstraction level between three software layers: Application, Runtime Environment and Basic Software which run on a Microcontroller:

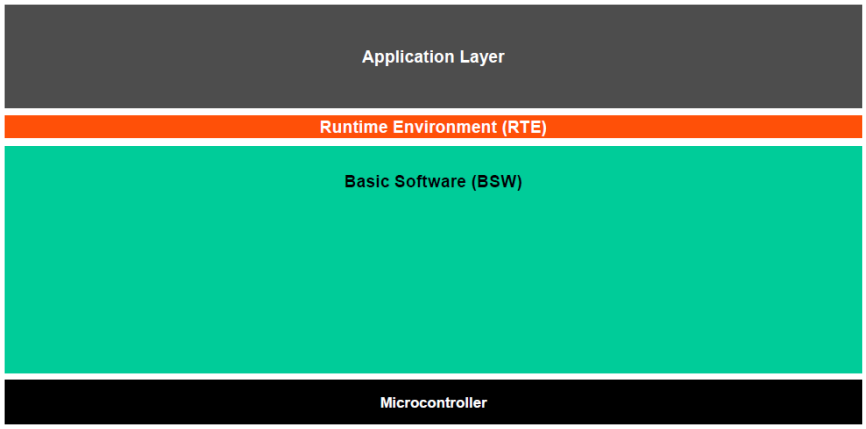


Figure 5. Layered software architecture with highest abstraction.

The AUTOSAR Basic Software is further divided in the layers: Services, ECU Abstraction, Microcontroller Abstraction and Complex Drivers.

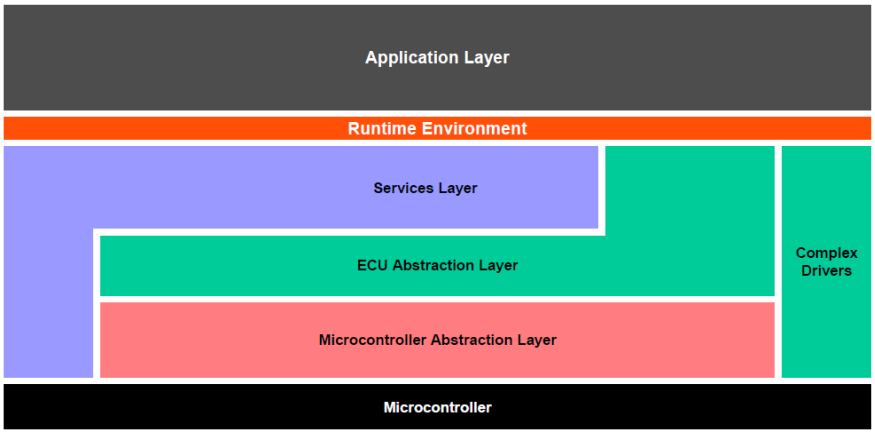


Figure 6. Basic software division.

The Basic Software Layers are further divided into functional groups. The SPI would fall in what the Autosar calls “The Communication Service” of the basic software (circled in red in the image below).

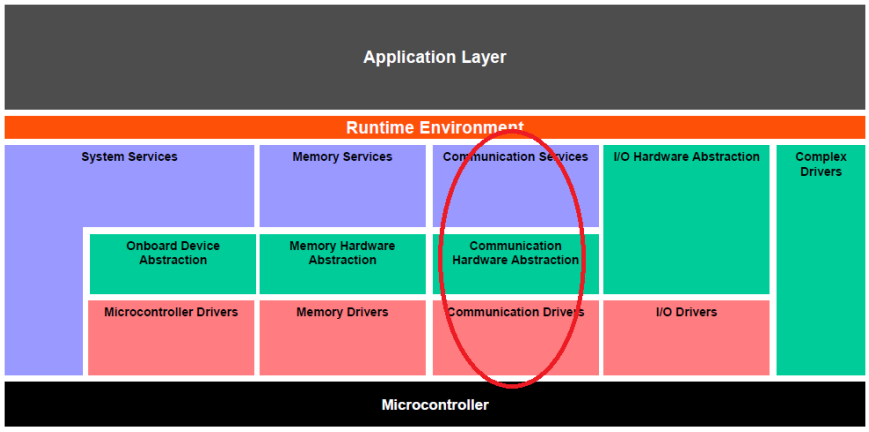


Figure 7. Functional groups division.

All the drivers and handlers needed for communications are included in this communication driver's functional group of the microcontroller abstraction layer. These drivers are the ones that directly interact with the communications modules in the microcontroller: SPI, LIN, CAN, etc. as shown in the image below:

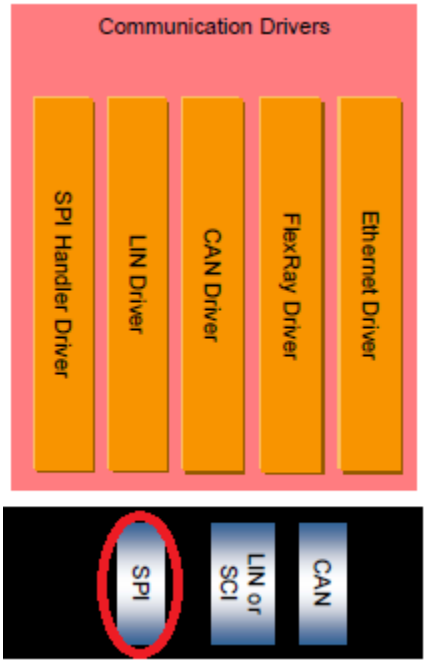


Figure 8. Location of the SPI module in the whole system architecture.

In this case we want to implement the SPI module, which is located in the lowest layer of the architecture, the microcontroller layer, this makes the SPI module completely hardware dependent (designed for one specific microcontroller).

The following image shows the architecture of the SPI MS (Master-Slave) module.

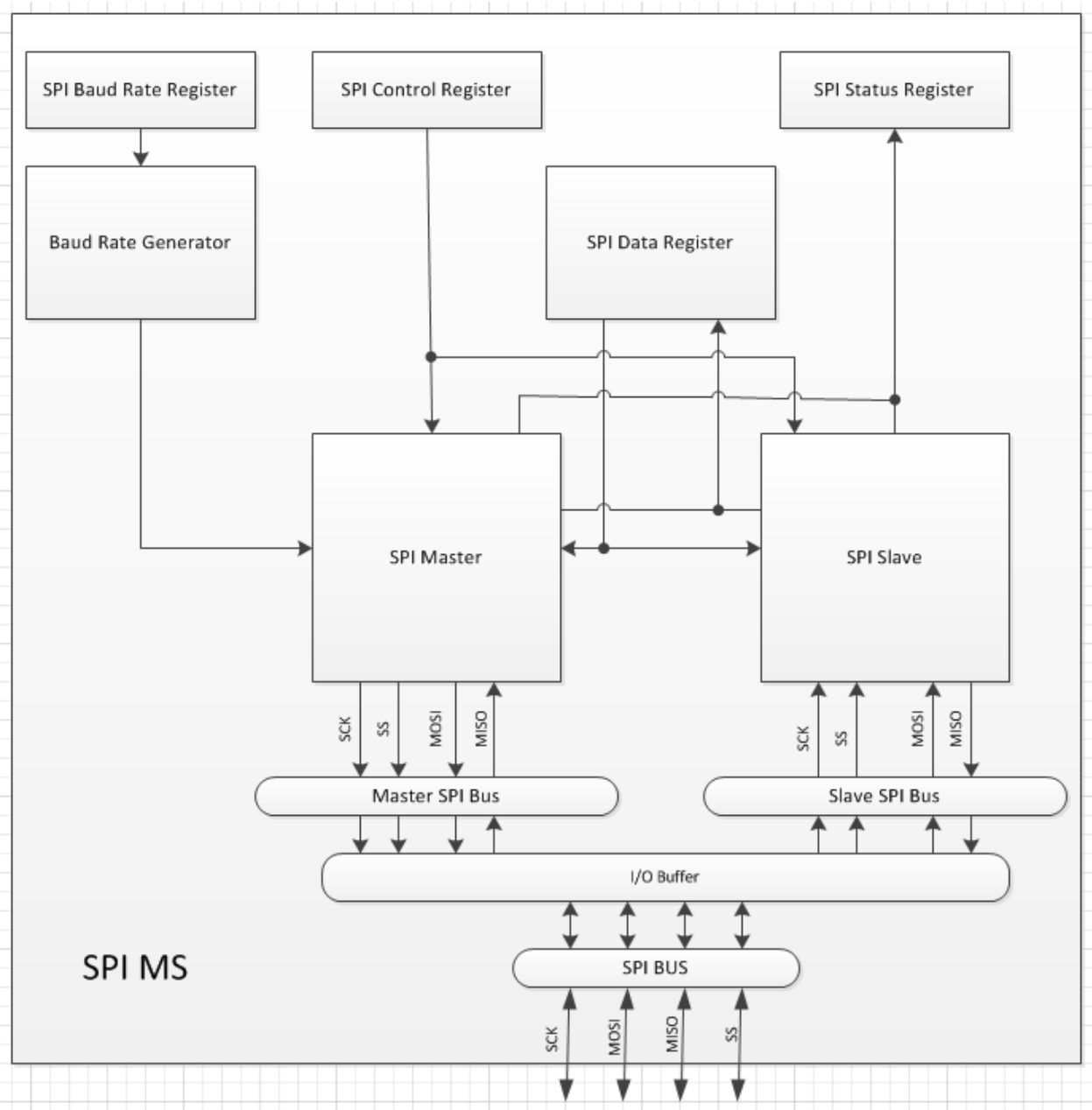


Figure 9. Architecture of the SPI Master-Slave module.

The desired behavior of the SPI master is described next. Data written to the SPI data register SPDR is automatically transferred to a shift register. SPI clock generator is enabled and serial data appears on the MOSI pin. After sending one byte, the SPI clock generator stops, the SPIF bit (flag) is set, the received byte is transferred to the SPDR register.

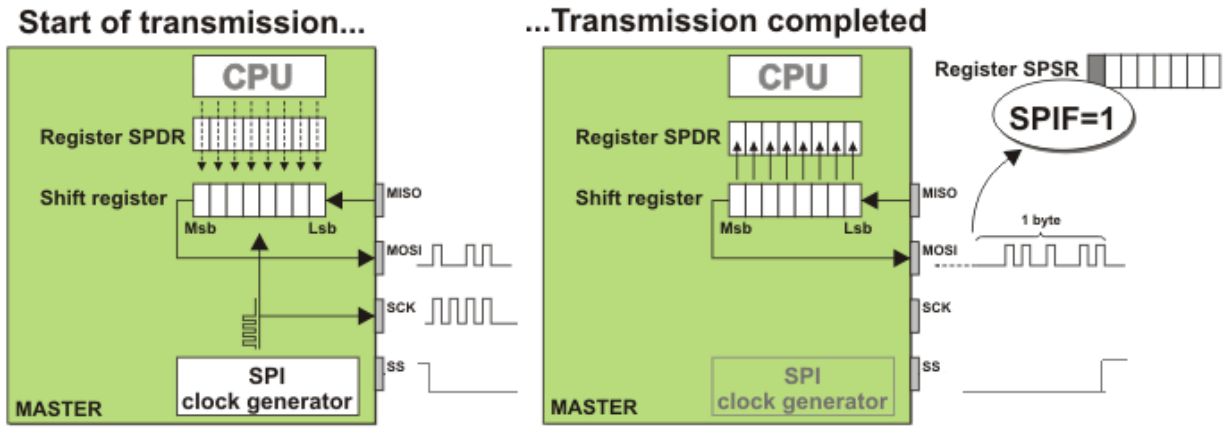


Figure 10. SPI Master behavior

The design of the SPI Master module is documented in the flow diagram shown below:

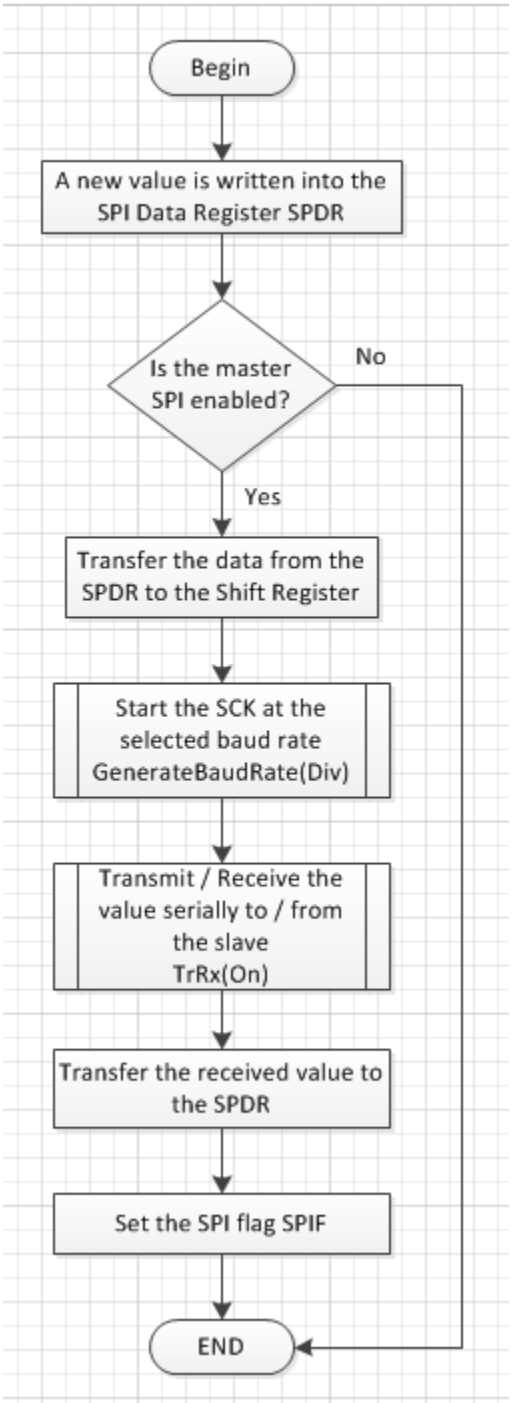


Figure 11. Design of the SPI Master module.

The design of the *GenerateBaudRate(Div)* module is shown in the state diagram from the following figure.

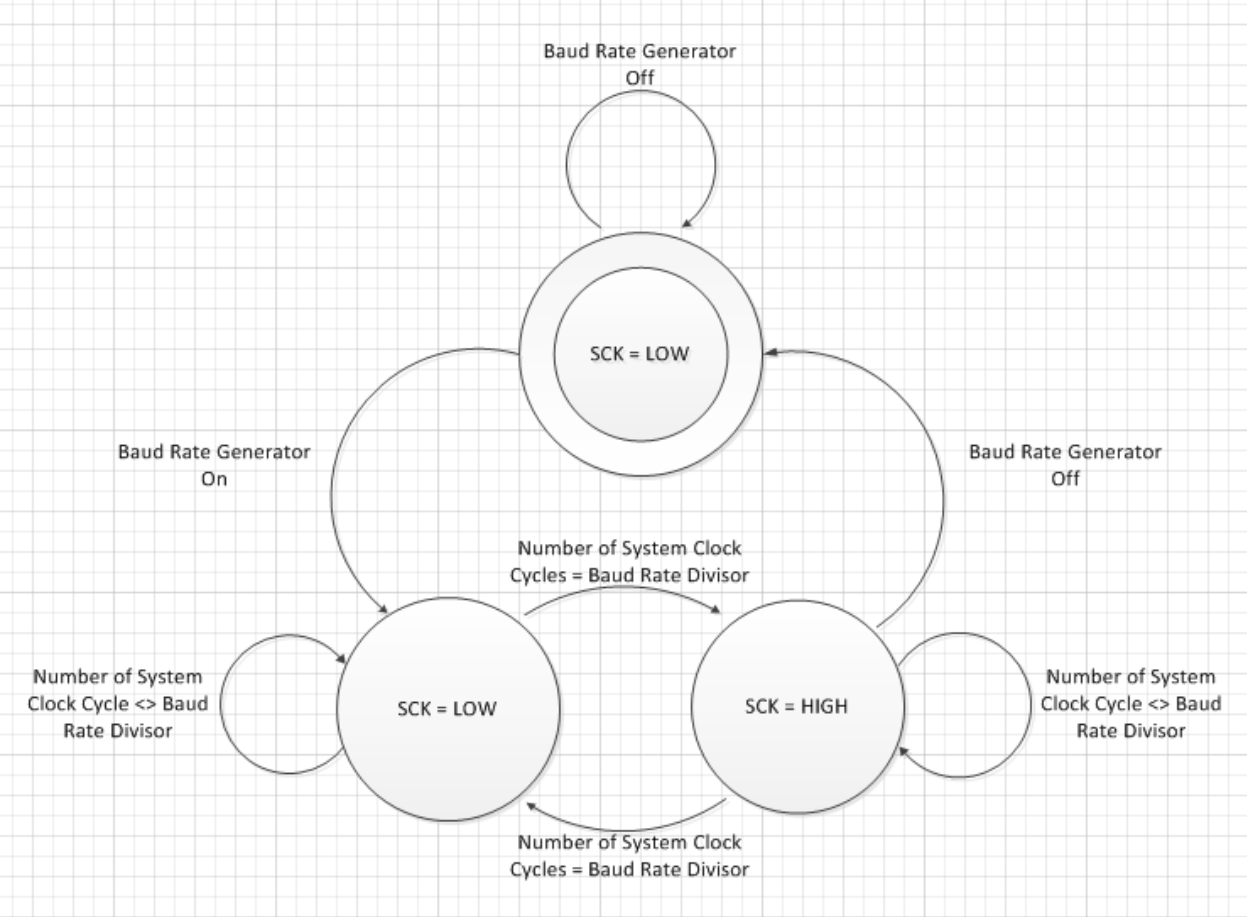


Figure 12. Design of the Baud Rate Generator module.

The design of the *TxRx(On)* module is shown in the next state diagram in figure 12.

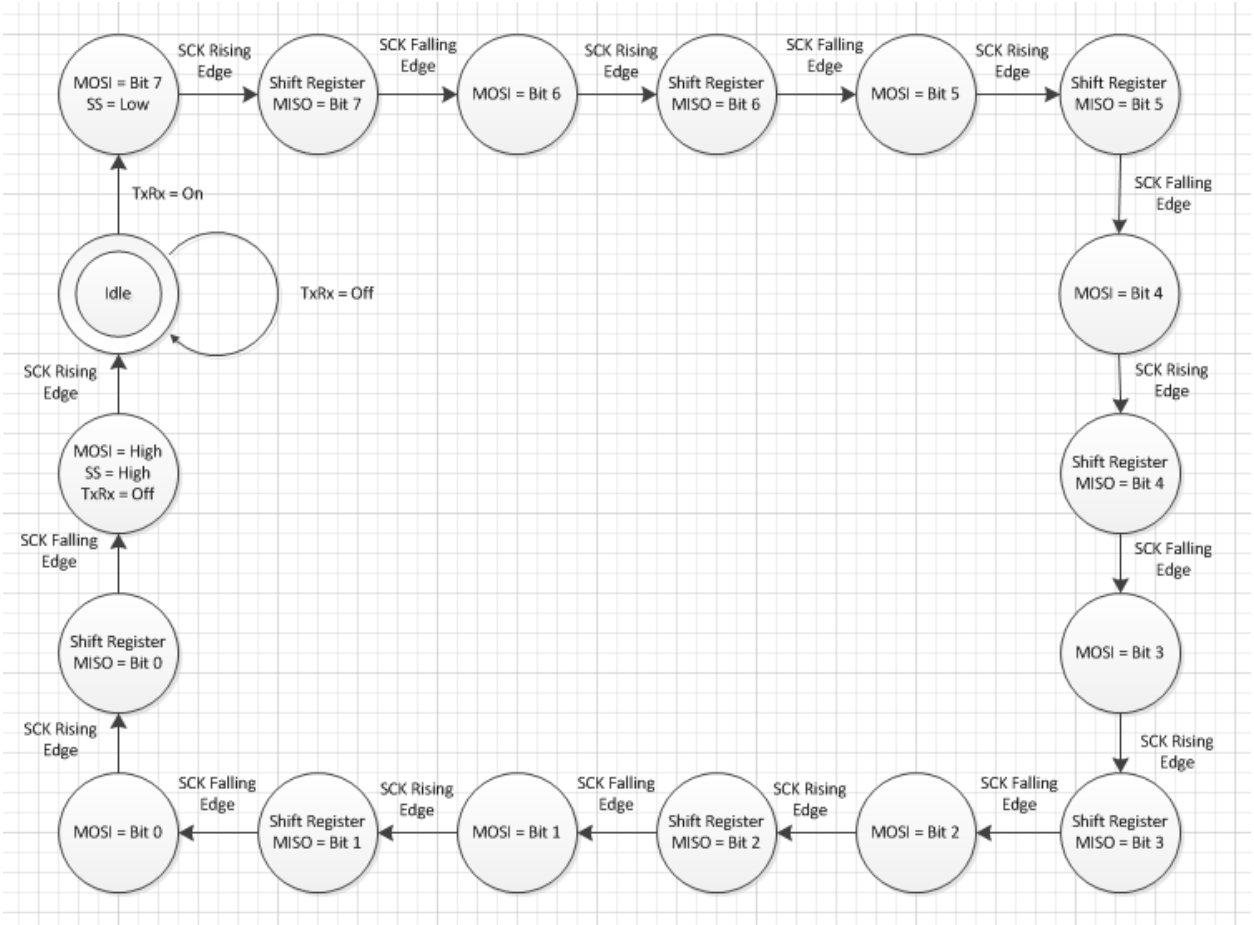


Figure 13. Design of the TxRx(On) module that transmits data to the slave through the MOSI and receives data from it through the MISO.

The signals that would be generated using the described design are shown in the following figure:

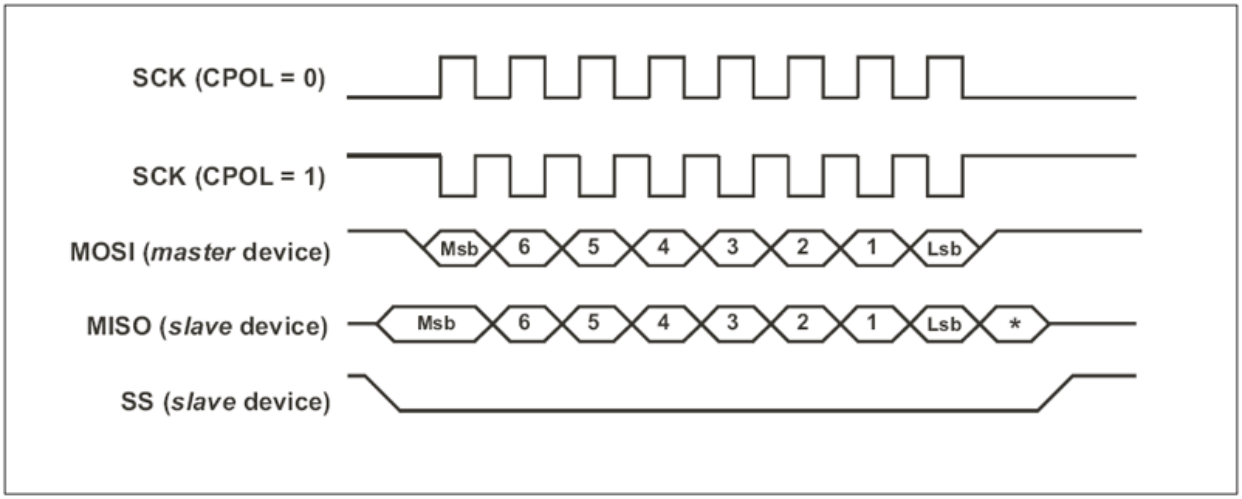


Figure 14. SPI Clock format 0.

Acronyms

Acronym:	Description:
CS	Chip Select
ECU	Electric Control Unit
MCU	Micro Controller Unit
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
Master	A device controlling other devices (slaves, see below)
Slave	A device being completely controlled by a master device
SS	Slave Select
SCK	SPI Clock

References

- Serial Peripheral Interface Bus (http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)
- SPI Block Guide V03.06 (http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf)
- Microsoft Application Architecture Guide, 2nd Edition (http://msdn.microsoft.com/en-us/library/ee658098.aspx)
- AUTOSAR Layered Software Architecture (http://www.autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf)

ITESO
Universidad Jesuita
de Guadalajara

Serial Peripheral Interface

Master Test Plan

SPI

Document Version: 1.0.0

Date: 12/01/2012
Document Status: Draft

Author(s): Castulo J. Martinez

Version History

V. No.	Details of Change	Changed Sections	Prepared by	Reviewed by	Date
V.1.0.0	First document draft	All	*Castulo J. Martinez*	N/A	12/02/2012

Review / Approvals

Content to be Approved	Approver	Role	Approved Date
All	Hector Rivas	Assignment Instructor	TBD

TABLE OF CONTENTS

1 INTRODUCTION

This document provides the overall testing strategy and approach required to ensure that the requirements of the Serial Peripheral Interface module are tested adequately, and that the required level of quality and reliability of the software deliverables is attained.

Master Test Plan is initiated in the Analysis phase and completed by the end of the Design phase. However, this document could be updated throughout the project.

1.1 PURPOSE AND SCOPE

The purpose of this document is to communicate activities related to the planning, staffing, managing and execution of testing activities for the Serial Peripheral Interface module project.

This document focuses on:

- Overall testing strategy
- Levels of testing to be performed
- Entry and Exit criteria for each test level
- Supporting testing tools
- Roles and Responsibilities of supporting testing resources
- System resources
- Assumptions and risks

1.2 OUT OF SCOPE

The SPI module includes both sub-modules, the master and the slave. All the testing that will be included in this test plan is only related to the Master and its functionality, using the SPI as a slave will not be considered for testing. Also, in SPI communication, it is possible to have different configurations when connecting a master with a slave. The only configuration covered by this test plan is the one master – one slave connection.

The following configurations for communication will not be covered by this test plan:

- Master with several slaves.
- Master with master.
- Several masters with one slave.

1.3 BACKGROUND

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard, named by Motorola, which operates in full duplex mode. A SPI bus is a master slave multi node bus system, the master sets a Chip Select (CS) to select a slave for data communication. The SPI (Serial Peripheral Interface) has a 4-wire synchronous serial interface. Data communication is enabled with a Chip Select wire (CS). Data is transmitted with a 3-wire interface consisting of wires for serial data input (MOSI), serial data output (MISO) and Serial Clock (SCK).

Operation

The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. With multiple slave devices, an independent SS signal is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance (disconnected) when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

Data Transmission

To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1–100 MHz.

The master then transmits the appropriate chip select bit for the desired chip to a logic 0. A logic 0 is transmitted because the chip select line is active low, meaning its off state is a logic 1; on is asserted with a logic 0. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line
- The slave sends a bit on the MISO line; the master reads it from that same line

Not all transmissions require all four of these operations to be meaningful but they do happen.

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data and the process repeats.

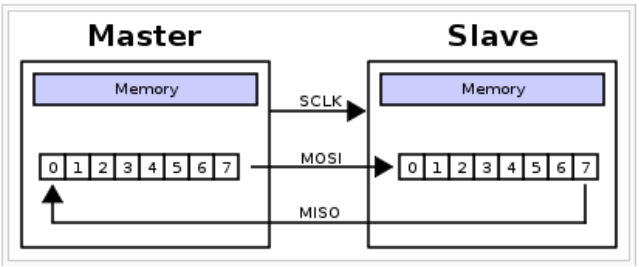


Figure1. A typical hardware setup using two shift registers to form an inter-chip circular buffer.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it wishes/needs. However, other word sizes are also common, such as 16-bit words.

Every slave on the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

Clock polarity and phase

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Motorola named these two options as CPOL and CPHA respectively, and most vendors have adopted that convention.

The timing diagram is shown below:

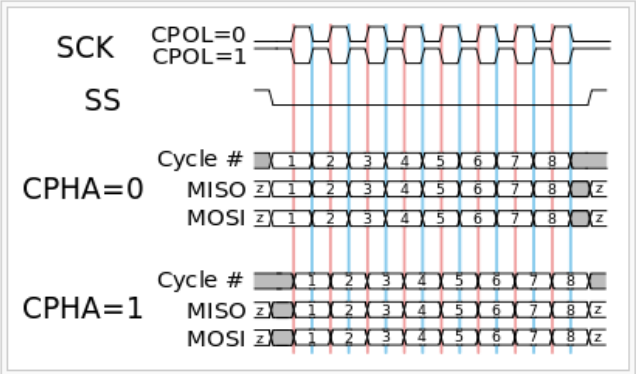


Figure 2. A timing diagram showing clock polarity and phase.

At CPOL=0 the base value of the clock is zero

- For CPHA=0, data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition).
- For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.

At CPOL=1 the base value of the clock is one (inversion of CPOL=0)

- For CPHA=0, data is captured on clock's falling edge and data is propagated on a rising edge.
- For CPHA=1, data is captured on clock's rising edge and data is propagated on a falling edge.

That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active.

The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle.

Mode numbers

The combinations of polarity and phases are often referred to as modes which are commonly numbered according to the following convention, with CPOL as the high order bit and CPHA as the low order bit:

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Figure 3. Possible modes for SPI.

2 TEST STRATEGY

This section addresses test level selection, characteristics and testing tools.

2.1 SOFTWARE DEVELOPMENT LIFECYCLE MODEL

The software development lifecycle model that will be followed for this product is **V model**.
The whole release of the product is planned to be completed in 3 weeks distributed in the following way:

- 2 weeks for the development activities including requirements specification, design, and code implementation.
- 1 week for the testing activities (all applicable test levels mentioned in section 2.3).

2.2 TEST COVERAGE STRATEGY

Coverage Strategy	Choose One (x)
100% Feature Coverage	X
Testing Risk Based Analysis	

2.3 TEST LEVEL COVERAGE

The ISTQB (International Software Qualification Board) identifies 4 basic test levels:

- Component test (or Unit Test): verifies whether each software component performs correctly according to its specification.
- Integration test: checks if groups of components collaborate in the way that is specified by the technical system design.
- System test: verifies whether the system as a whole meets the specified requirements.
- Acceptance test: checks if the system meets the requirements, as specified in the contract, from the customers point of view.

Since integration testing can mean component integration testing (integrating a set of components to form a system, testing the builds throughout that process), or it can mean system integration testing (integrating a set of systems to form a system of systems, testing the system of systems as it emerges from the con-glomeration of systems). It suggests breaking the

"Integration Test" level into two levels: "Component Integration Test" and "System Integration Test".

Finally, this project is a hardware-software project, so the ISTQB also recommends adding one more test level for this kind of projects: "Hardware-software integration testing"

Test Level	Applicable?	Rationale for omitting test level
Unit Test (UT)	☒ *Yes* ☐ *No*	
Component Integration Test (CIT)	☐ *Yes* ☒ *No*	Since the SPI module is small, there is no real value into executing the component integration testing. The tests executed during this level would be pretty much the same one that would be executed during system test. This test level can be skipped.
System Test (ST)	☒ *Yes* ☐ *No*	
System Integration Test (SIT)	☐ *Yes* ☒ *No*	Not Applicable - The system under test is not a system of systems.
Hardware-software integration testing (HSIT)	☒ *Yes* ☐ *No*	
User Acceptance Test (UAT)	☐ *Yes* ☒ *No*	There is no customer or end user for this project.

2.4 TEST LEVEL CHARACTERISTICS

Test Level	Owner	Entry Criteria	Exit Criteria
UT	Development Team	• Business Requirement or functional specification is completed • Development of component is completed • Development environment is stable • Code compiled cleanly.	• 100% of planned test scenarios for unit test level must be executed and passed. (No formal test cases will be created for this level)
CIT	Development Team	• N/A	• N/A
ST	Test Team	• All previous applicable test levels are completed • System test environment is established • Adequate Test data is available • Completed and reviewed	• All items in scope were tested • All test cases (100%) are executed: failed cases have a satisfactory resolution • Defects were documented and reported • All severity 1 (showstopper)

		test cases / test scripts	and 2 (major) defects are resolved and implemented • All severity 3 (minor) and 4 (cosmetic) defects are deferred or action planned
SIT	Test Team	• N/A	• N/A
HSIT	Test Team	• All previous applicable test levels are completed • The integration test environment is established • Adequate Test data is available • Completed and reviewed test cases / test scripts	• All items in scope were tested • All test cases (100%) are executed: failed cases have a satisfactory resolution • Defects were documented and reported • All severity 1 (showstopper) and 2 (major) defects are resolved and implemented • All severity 3 (minor) and 4 (cosmetic) defects are deferred or action planned
UAT	Business Partner or Customer	• N/A	• N/A

2.5 TEST SCOPE

Test Level	In Scope	Out of Scope
UT	• SPI Baud Rate Register • SPI Status Register • SPI Control Register • Baud Rate Generator module • SPI Master module	• SPI Slave module
CIT	N/A	N/A
ST	• All functions related to the SPI master module: Outputting values in the MOSI and reading values from the MISO • Transmission of 8 bit words	• All functionality related to the SPI slave • Transmission of 16 and 32 bit long words • Transmission of data at high speed • Testing the SCK with reverse polarity
SIT	N/A	N/A
HSIT	• All functions related to the SPI master module: Transmitting values to a Slave device and reading values from it • Transmission of 8 bit words	• All functionality related to the SPI slave • Transmission of 16 and 32 bit long words • Transmission of data at high speed • Testing the SCK with reverse polarity
UAT	N/A	N/A

2.6 PRIORITIZATION

2.6.1 PRIORITIZATION FOR TEST EXECUTION

All test execution will be prioritized based on a test case risk assessment. The test case risk will be a combination of likelihood of failure and impact if it fails. All test cases will receive an overall score that will be grouped into a high, medium, or low category. These categories will be used to determine the order that test cases should be executed.

Score will be done using:

Impact
1 = None - No noticeable impact to features
2 = Little – Low impact to features
3 = Moderate - Medium impact to features
4 = Severe - High impact to features
5 = Extreme - Critical impact to features

Likelihood of feature failure
1 = Somewhat Likely - Little chance that the feature will fail
2 = Likely – Feature will probably fail, but not certain
3 = Very Likely - Very high probability that the feature will fail

Overall score is the product of the impact value times the likelihood value
High = 9-15
Medium = 5-8
Low = 1-4

2.6.2 PRIORITIZATION FOR BUGS

Bug priority will be defined by getting stakeholder response during a daily meeting as to the importance, looking for any dependent activities (testing, other bug resolution, and other project activities). The priority will be recorded with the bug in the bug tracking tool.

Priority Values:

Critical = This would cause all testing activities to be halted until the bug is fixed. This must be fixed immediately.

High = Continued testing, bug resolution and/or project activities would be dependent on getting this bug fixed. This must be fixed immediately.

Medium = Continued testing, defect resolution and/or project activities could be slightly dependent on this bug getting fixed. The problem should be fixed before release of the current version in development, or a patch must be issued if possible.

Low = Continued testing, bug resolution and/or project activities are not dependent on getting this bug fixed. The bug should be fixed if there is time. But it can be deferred until the next release if the severity is medium or low.

2.7 TESTING TOOLS

Test Level	Tool(s)	Description
Unit Test, System Test	Code Composer	Used to execute the code in debugging mode for testing purposes.
Hardware-Software Integration Test	Oscilloscope, TI Launchpad	Will be used to measure signals when testing the modules.
All	Traceability Matrix	Used to link every test case with its requirement specification.
All	Bugzilla	Will be used as a defects management tool.

3 ROLES AND RESPONSIBILITIES

Role/ Group	Responsibilities	Name(s)
Project Manager	N/A	N/A
Test Manager *(or Test Lead, Solution Quality analyst)*	*Provides testing management oversight. Responsibilities:* • *provide technical direction* • *acquire appropriate resources* • *provide management reporting*	Castulo Martinez
Product Owner	*Represents customer's interest and represents the product to the outside world (Customer).* *Responsibilities:* • *Responsible for market, business case, and competitive analysis* • *Responsible for long and short term product vision* • *Prioritizes features for releases based upon expected ROI* • *Writes Acceptance Criteria* • *Writes user stories* • *Makes trade-off decisions between scope and schedule*	Hector Rivas

QA Team	*Executes the tests.* *Responsibilities:* • *decide on the scope of the testing in agreement with Project Manager* • *execute tests* • *log results* • *recover from errors* • *document change requests* • *open and verify bugs* • *Enforces QA Best Practices* • *Defining Quality* • *Measuring Quality*	Castulo Martinez
Development Team	*Responsible for creation of product.* *Responsibilities:* • *Estimates size of backlog items* • *Translation of backlog items into engineering design and logical units of work (tasks)* • *Evaluation of technical feasibility* • *Implementation of backlog items* • *Writes unit tests* • *Writes and verifies code which adheres to the acceptance criteria* • *Application of product development best practices*	Castulo Martinez

Architect	Leads the technical direction of overall system. Responsibilities: • Responsible for end-to-end cross functional system design and communication • Works with the PM to group features based upon the Architectural Elements which support them, an influence on priorities • Facilitates technical decision; incorporates feedback and emergent patterns from the team back in to the overall design • Produces alternate Design Concepts & detailed approach • Ensures the Design goals – Performance, Modularity, Reliability, Maintainability, Reusability, Internationalization and Accessibility – are met • Ensures technical cohesion and helps write the technical contract in interfaces and other abstract objects and data entities	Castulo Martinez
ET	Ensures test environment and assets are managed and maintained. Responsibilities: • administer test management system • install and manage access to test systems	N/A
Database Administrator, Database Manager	Ensures test data (database) environment and assets are managed and maintained. Responsibilities: • administer test data (database)	N/A
Business Partner	• Develops UAT test cases • Performs UAT • Reviews test results	N/A

4 TEST ENVIRONMENT AND RESOURCES

The following table is used to identify the system resources (hardware, software etc. required for the test environment needed to support each test level.

4.1 SYSTEM TEST ENVIRONMENT

4.1.1 HARDWARE

Resource	Type
Texas Instrument Launchpad	Examination (development) board
IC with SPI slave capabilities	SPI slave device
USB to mini USB cable	Cable

4.1.2 SOFTWARE

Testing Level	Environment	Component	Product/Application	Version
N/A	N/A	N/A	N/A	N/A

4.2 ENVIRONMENT ACCESS

Environment	Group	Reason for Access
N/A		

4.3 TEST DATA ACQUISITION

The following table is used to identify the approach for acquiring and securing the test date to be used for each test level.

Source of Test Data	Extraction approach	Type of test data (input or pre-existing)	Security controls
Will be created according to test plans	N/A	input	N/A

5 TEST ASSUMPTIONS AND RISKS

5.1 ASSUMPTIONS

This section lists assumptions that are specific to the test planning.

#	Assumption
1	Testing environments will be stable.
2	The lack of the slave module won't affect the functioning of the master module.
3	No data integrity testing is needed.
4	If the device works correctly with one test slave device, it will work fine with any other.
5	If the device works correctly with one master – slave configuration, it will work fine with the rest of the possible configurations.

6	If the device works fine with 8 bit words, it will work fine when the system is scaled to 16 or 32 words.
7	If the device works fine with 2 or 3 different values of common baud rates, it will work fine with all the baud rates up to 1 Mega baud.
8	If the device works fine with the SCK at normal polarity, it will work fine with SCK at reverse polarity.

5.2 RISKS

The following risks to the testing plan have been identified and the supporting contingency plans included to mitigate their impact on the project. The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered. The trigger is the milestone or event that would cause the risk to become an issue to be addressed.

#	Risk	Impact	Trigger	Mitigation/ Contingency Plan
1	Development and testing for this project is being executed by the same person.	Overlook bugs – Medium impact	Error in the application occurs in areas that were already tested	The bugs would have to be corrected, and a regression suite should be executed (by a 3rd party preferable).
2	Incorrect design that could introduce bugs when the slave module is implemented	The functionality could be compromised – Critical impact	Having to change already completed modules to fit the new functionality	Run all the test suite again
3	Not having the code implemented on time	Testing pushed or squeezed – High Impact	Development schedule almost up and code less than 90% complete	Overall schedule could be extended, or functionality could be deferred to a later release, whatever makes more sense.

6 TEST SCHEDULE

Testing Level	Test Activity	Timeframe
Unit Test	Execute unit test as soon as the component code is implemented, during the development schedule	Development cycle
System Test	Test Cases creation	TBD
System Test	Test Cases execution	TBD
Hardware-Software Integration Test	Test Cases creation	TBD
Hardware-Software Integration Test	Test Cases execution	TBD

7 TEST REPORTING

Following measurements will be collected and reported.

#	Metrics	Measurement Data	Frequency	Responsible	Reported To

8 REFERENCES

The references are separated into “external” references that are imposed external to the project and “internal” references that are imposed from within to the project.

8.1 EXTERNAL REFERENCES

This section lists references to the relevant policies or laws that give rise to the need for this plan.

a) IEEE Std 829-2008 IEEE Standard for Software and System Test Documentation

8.2 INTERNAL REFERENCES

This section lists references to documents such as other plans or task descriptions that supplement this plan.

a) SPI Requirements Specification.docx
b) SPI Architecture and Design.docx
b) Serial Peripheral Interface Bus (http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)
c) SPI Block Guide V03.06 (http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf)

9 GLOSSARY

Item	Description
Black box testing	Focus is on the external attributes and behavior of the software. Such testing examines the software from the user perspective. UAT is the classical example of this type of testing
Bug	A bug is a flaw, error or omission identified during the testing process. Bugs are typically classified by level of severity ranging from non-critical to “show stopper”
Negative Testing (destructive)	Testing attempts to prove that the software can be broken using invalid or erroneous input conditions. Both defined and undefined error conditions should be generated.
Positive Testing	Testing attempts to prove that the software satisfies the requirements
S&P testing	Stress and Performance testing
Test Case (TC)	A test case is a specific test designed to verify a particular condition or requirement. It identifies input data with predicted results and describes the testing objective.
Test Script	Provide the step by step procedures comprising the actions to be taken and the verification of the results
White-box testing	It tests software with knowledge of internal data structures, logical flow at the source code level. Unit testing is the classical example of this type of testing.
CS	Chip Select

ECU	Electric Control Unit
MCU	Micro Controller Unit
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
Master	A device controlling other devices (slaves, see below)
Slave	A device being completely controlled by a master device
SS	Slave Select
SCK	SPI Clock

Serial Peripheral Interface

Test Cases Design

Test Case Id	SPI01
Description	Verify the SPI Master module transmits serially the data from the SPI Data Register to the MOSI line.

Assumptions & Prerequisites:

- Make sure to examine the test closely so data is not overwritten in the SPI data register.
- The MISO line would need to be connected to ground.

Steps:	Expected Results:
1) Enter an 8 bit data into the SPI Data Register	• Value is saved in the register
2) In the SPI Control Register enable the SPI Master	• The data from the data register should be copied to the shift register. • The SCK is generated. • The SS signal should be set to logic 0 • The MOSI should show one bit of data at each rising edge of the SCK clock.
3) Examine the sequence of bits that are output in the MOSI line	• The bits should correspond to the bits in the data register starting by the MSB
4) Once the 8 bits of data have been transmitted	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the SPI data register should have been updated with a random value (we don't care about this value for this test)
5)	•
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	SPI02
Description	Verify the SPI Master module receives serially the data in the SPI Data Register from the MISO line.

Assumptions & Prerequisites:

- Make sure to examine the test closely so data is not overwritten in the SPI data register.

Steps:	Expected Results:
1) Enter any value in the SPI Data Register	• The value is saved in the register
2) In the SPI control register enable the SPI master module	• The SCK is generated. • The MISO should show one bit of data at each rising edge of the SCK clock.
3) Examine the sequence of bits that are input in the MISO line	• Since the MISO line is connected to ground, all the incoming bits should be 0
4) Once the 8 bits have been received, check the SPI data register in the master device	• The value in the register should be 0x00
5)	•
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	SPI03
Description	Verify the SPI Master module can exchange data with a SPI Slave device.

Assumptions & Prerequisites:

- The Master device should be connected to a SPI slave device; the slave device should have data loaded in its data register.

Steps:	Expected Results:
1) Enter an 8 bit data into the SPI Data Register	• The value is saved in the register
2) In the SPI Control Register enable the SPI Master	• The data from the data register should be copied to the shift register. • The SCK is generated.

	• The SS signal should be set to logic 0 • The MOSI should show one bit of data at each rising edge of the SCK clock. • The MISO line should show one bit of data at each rising edge of the SCK clock.
3) Examine the sequence of bits that are output in the MOSI line	• The bits should correspond to the bits in the data register of the Master device starting by the MSB
4) Examine the sequence of bits that are output in the MISO line	• The bits should correspond to the bits in the data register of the Slave device starting by the MSB
5) Once the 8 bits of data have been transmitted	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the data registers of each device should have been exchanged (swapped).
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	SPI04
Description	Verify the communication is always initiated by the SPI master device

Assumptions & Prerequisites:

- The Master device should be connected to a SPI slave device.

Steps:	Expected Results:
1) Enter data into the SPI data register in the slave device.	• The value is saved in the register.
2) Enter data into the SPI data register in the master device. Do not enable the SPI in the master device.	• The value is saved in the register.
3) Monitor the MOSI, MISO, SS and SCK signals.	• MISO and MOSI lines are steady (not changing) • SCK signal is off • SS signal is high
4)	•
5)	•

6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	SPI05
Description	Verify the Baud Rate Generator can work with different clock speeds

Assumptions & Prerequisites:

- N/A

Steps:	Expected Results:
1) In the SPI master device, set the system clock divisor in the SPI Baud Rate Register	• The register is updated with the value assigned by the user
2) Enter data into the SPI data register in the Master device	• The data is saved in the register
3) Enable the SPI master module in the SPI control register	• The SCK clock signal is generated at the frequency: System clock / Divisor
4) Repeat steps 1 to 3 for at least 5 different values for the divisor in the SPI Baud Rate Register	• The SCK signal frequency matches the value selected by the user in the SPI baud rate register
5)	•
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	SPI06
Description	Verify the user can choose the word length for the SPI communication between 8, 16 or 32 bits (Functionality not yet available)

Assumptions & Prerequisites:

- The Master device should be connected to a SPI slave device; the slave device should have data loaded in its data register.

Steps:	Expected Results:
1) In the SPI control register of the master device, select the word length to 8 bits	• Word length selected to 8 bits
2) Enter an 8 bit data into the SPI Data Register	• The value is saved in the register
3) In the SPI Control Register enable the SPI Master	• Communication should be started
4) After 8 bits of data have been transmitted the transmission should stop	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the data registers of each device should have been exchanged (swapped), and they should be of the correct size.
5) Repeat steps 1 through 4 but using the 16 and 32 bits word length option.	• All expected results are still working correctly for the new word legths.
6)	•
7)	•
8)	•
9)	•

Test Case Id	SPI07
Description	Verify the user can select reverse polarity for the SCK signal (functionality not yet available)

Assumptions & Prerequisites:

- Make sure to examine the test closely so data is not overwritten in the SPI data register.
- The Master device should be connected to a SPI slave device; the slave device should have data loaded in its data register.

Steps:	Expected Results:
1) Enter an 8 bit data into the SPI Data Register	• The value is saved in the register
2) In the SPI Control Register enable the SPI Master	• The data from the data register should be copied to the shift register. • The SCK is generated. • The SS signal should be set to logic 0

	• The MOSI should show one bit of data at each **falling** edge of the SCK clock. • The MISO line should show one bit of data at each **falling** edge of the SCK clock.
3) Examine the sequence of bits that are output in the MOSI line	• The bits should correspond to the bits in the data register of the Master device starting by the MSB
4) Examine the sequence of bits that are output in the MISO line	• The bits should correspond to the bits in the data register of the Slave device starting by the MSB
5) Once the 8 bits of data have been transmitted	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the data registers of each device should have been exchanged (swapped).
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	**SPI08**
Description	**Verify the SPI Master device can choose between different Slave devices if more than one are connected (functionality not yet available)**

Assumptions & Prerequisites:

- The Master device should be connected to a SPI slave device; the slave device should have data loaded in its data register.
- More than one slave should be connected to the master device, and each slave device should have a separate SS signal. For example, if there are 3 slave devices connected to the Master, the following SS lines should exist in the master: SS0, SS1 and SS2
- It is assumed that when no selected, the slaves disable their buses (set the lines to 3rd state), so they don't interfere with each other.

Steps:	Expected Results:
1) Enter an 8 bit data into the SPI Data Register of the master device	• The value is saved in the register
2) In the SPI Control Register, select the salve that you want to interact with and enable the SPI Master	• The data from the data register should be copied to the shift register. • The SCK is generated.

	• The correct SS signal should be set to logic 0, only that assigned to the desired Slave device. The rest of the SS lines should be kept at logic 1. • The MOSI should show one bit of data at each **rising** edge of the SCK clock. • The MISO line should show one bit of data at each **rising** edge of the SCK clock.
3) Examine the sequence of bits that are output in the MOSI line	• The bits should correspond to the bits in the data register of the Master device starting by the MSB
4) Examine the sequence of bits that are output in the MISO line	• The bits should correspond to the bits in the data register of the **selected** Slave device starting by the MSB
5) Once the 8 bits of data have been transmitted	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the data registers of each device should have been exchanged (swapped).
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	**SPI09**
Description	**Verify the Slave module has the high impedance capability when no selected (not yet available)**

Assumptions & Prerequisites:

- N/A

Steps:	**Expected Results:**
1) In the SPI control register, activate the Slave module.	• The Slave module is activated and the Master module is disabled.
2) Force the SS signal of the SPI to ground	• The MISO line should be set to logic 0
3) Force the SS signal of the SPI to VCC (logic 1) to disable it	• The MISO line should be set to high Impedance.
4)	•

5)	•
6)	•
7)	•
8)	•
9)	•
10)	•

Test Case Id	SPI10
Description	Verify the SPI Master can work with any IC with SPI slave integrated (Memories, ADCs, LCDs, etc)

Assumptions & Prerequisites:

- The ICs should have the SPI Slave mode enabled, and should have a data already loaded in its data register.

Steps:	Expected Results:
1) Connect an IC with SPI slave capabilities to the master bus	• Slave connected and ready to communicate
2) Enter an 8 bit data into the SPI Data Register of the master device	• The value is saved in the register
3) In the SPI Control Register enable the SPI Master	• The data from the data register should be copied to the shift register. • The SCK is generated. • The SS signal should be set to logic 0 • The MOSI should show one bit of data at each rising edge of the SCK clock. • The MISO line should show one bit of data at each rising edge of the SCK clock.
4) Once the 8 bits of data have been transmitted	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the data registers of each device should have been exchanged (swapped).
5) Replace the IC connected in step 1 with another SPI Slave capable IC. Repeat steps 2 to 4.	• The same expected results are gotten with the new device.
6)	•
7)	•
8)	•

Test Case Id	SPI11
Description	Verify that if a new value is written to the SPI data register before the last transaction finishes, the transaction doesn't fail.

Assumptions & Prerequisites:

- The Master device should be connected to a SPI slave device; the slave device should have data loaded in its data register.

Steps:	Expected Results:
1) Enter an 8 bit data into the SPI Data Register in the SPI master device	• The value is saved in the register
2) In the SPI Control Register enable the SPI Master	• The data from the data register should be copied to the shift register. • The SCK is generated. • The SS signal should be set to logic 0 • The MOSI should show one bit of data at each rising edge of the SCK clock. • The MISO line should show one bit of data at each rising edge of the SCK clock.
3) Examine the sequence of bits that are output in the MOSI line	• The bits should correspond to the bits in the data register of the Master device starting by the MSB
4) Examine the sequence of bits that are output in the MISO line	• The bits should correspond to the bits in the data register of the Slave device starting by the MSB
5) **Before** the 8 bits of data have been transmitted update the SPI data register in the master device with a new value	• The new value replaces the old one in the data register
6) Once the 8 bits finished transmitting check the data registers in both devices: master and slave	• The SS signal should be set to logic 1 • The SCK signal is turned off • The value in the data registers of each device should have been exchanged (swapped). • The value written in the data register of the master device while the transmission was in progress should have been overwritten and it should have not caused any difference in the result.
7)	•
8)	•

9)	•
10)	•

C. REPORTE DEL PROYECTO DE FUENTE DE SWITCHEO TIPO BUCK PARA UN TABLERO DE INSTRUMENTACIÓN AUTOMOTRIZ

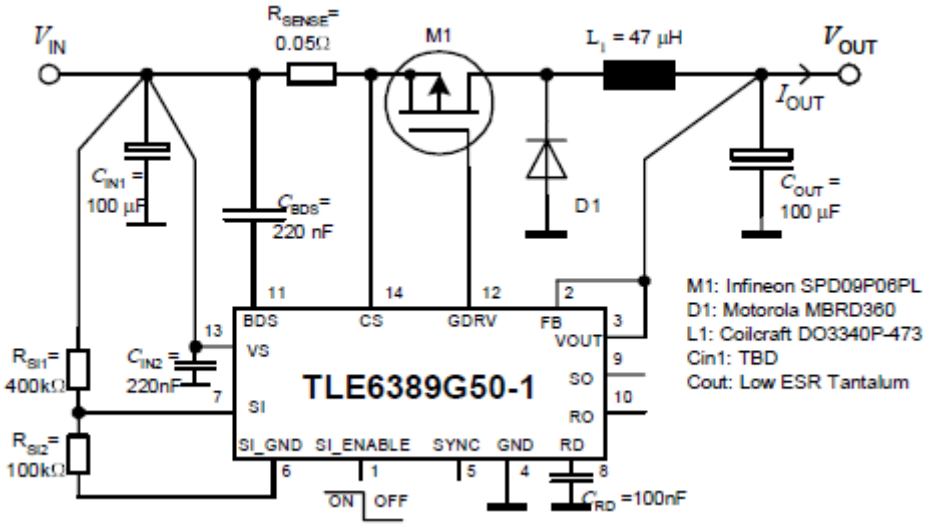
Requerimientos

- Voltaje de salida: 5V ± 3%
- Rizo máximo de voltaje: < 1%
- Corriente de salida: hasta 1.5A
- Voltaje de entrada de 9V a 16V (Rango típico automotriz)

Desarrollo

Para la realización del proyecto se decidió utilizar el controlador de step-down DC/DC TLE 6389 de Infineon que tiene las siguientes características:

- Rango de operación para voltaje de entrada: de 5V a 60V
- Eficiencia > 90 %
- Corriente de salida: hasta 3A
- Voltaje de salida ajustable de 3.3V y 5V.
- Rango de Operación: -40°C a 125 °C



Descripción

El controlador de switcheo TLE6389 step-down DC-DC provee alta eficiencia en cargas desde 1mA a 2.5A. Un esquema único de control PWM/PFM que puede operar hasta en un 100% de ciclo de trabajo, dando una baja caída de voltaje. El TLE 6389 step-down controller maneja un MOSFET canal P externo, dando flexibilidad al diseño para aplicaciones con una potencia de salida hasta 12.5W. Una alta frecuencia de switcheo y operación en modo de conducción continua permite el uso de inductores que pequeñas superficies de montaje. Los requerimientos de los capacitores de salida también son reducidos, minimizando el área de la tarjeta impresa y los costos. El voltaje de entrada puede ser hasta de 60V.

Se ha agregado una circuitería de protección de transientes para evitar daños en componentes.

Cálculos

El valor de la inductancia determina junto con el voltaje de entrada, el voltaje de salida y la frecuencia de switcheo, el rizo de corriente que ocurre durante la operación normal.

$$\Delta I = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1} \quad \text{eq.1}$$

En esta ecuación f_{SW} es la frecuencia de switcheo del dispositivo, dado por el oscilador interno.

Cuando se elige la inductancia final de algún proveedor, la corriente de saturación debe ser considerada. El valor de la corriente de saturación de la inductancia deseada debe ser mayor que la corriente pico máxima:

V_{intyp} = 13.8V / f = de 270kHz a 450kHz
V_{inmax} = 16V / f = de 270kHz a 450kHz
V_{inmin} = 9V / f = de 270kHz a 450kHz
V_{out} = 5V
L1 = 47µH (Sugerido por el fabricante) , $L1_{min}$ = 37.6µH, $L1_{max}$ = 56.4µH

Pos.	Inductions µH	Resistance	Part number	current	Temp.	Price
1	**47µF/±20%**	**86mΩ**	**Do5022P- 473**	**2.6A**	**85 °C**	**€**

$$\Delta I_{\max 270kHz} = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1_{\min}} = \frac{(16V - 5V) * 5V}{270kHz * 16V * 37.6\mu H} = 338.6mA \qquad \text{eq. 2}$$

$$\Delta I_{\max 420kHz} = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1_{\min}} = \frac{(16V - 5V) * 5V}{450kHz * 16V * 37.6\mu H} = 203.2mA \quad \text{eq. 3}$$

$$\Delta I_{typ270kHz} = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1} = \frac{(13.8V - 5V) * 5V}{270kHz * 13.8V * 47\mu H} = 251.3mA \quad \text{eq. 4}$$

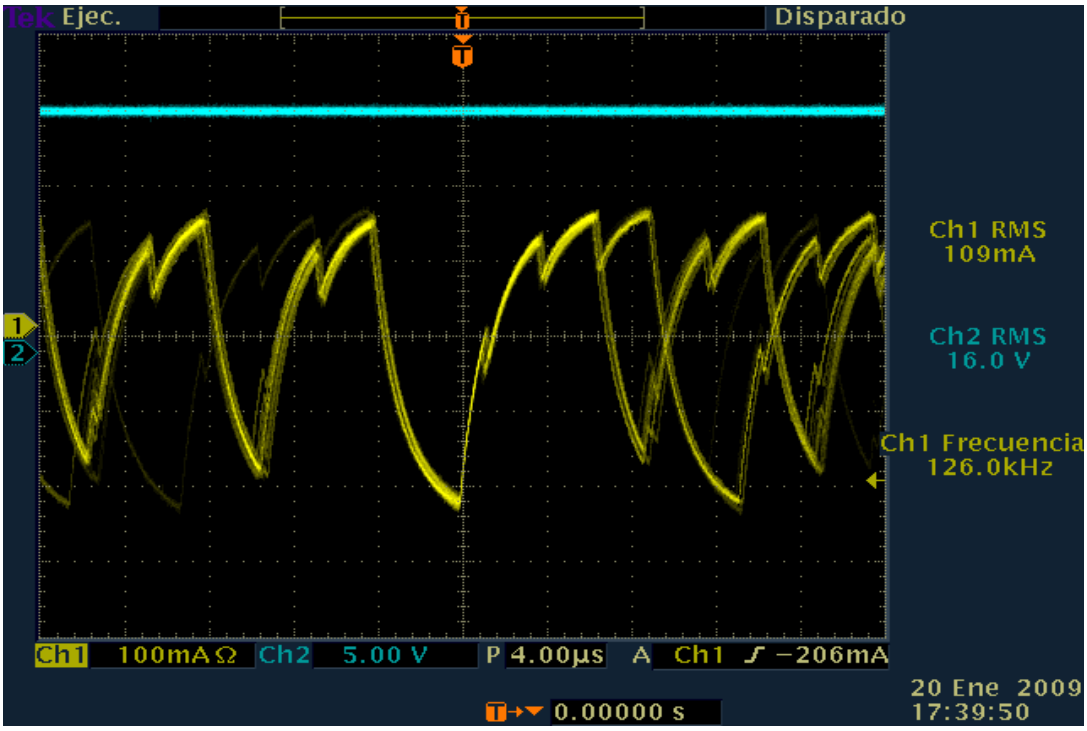
$$\Delta I_{typ420kHz} = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1} = \frac{(13.8V - 5V) * 5V}{450kHz * 13.8V * 47\mu H} = 150.8mA \quad \text{eq. 5}$$

$$\Delta I_{\min 270kHz} = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1_{\max}} = \frac{(9V - 5V) * 5V}{270kHz * 9V * 56.4\mu H} = 145.9mA \quad \text{eq. 6}$$

$$\Delta I_{\min 420kHz} = \frac{(V_{IN} - V_{out}) * V_{out}}{f_{SW} * V_{IN} * L1_{\max}} = \frac{(9V - 5V) * 5V}{450kHz * 9V * 56.4\mu H} = 87.6mA \quad \text{eq. 7}$$

El rizo de corriente estará entre **87.6 mA y 338.6 mA**.

La imagen siguiente muestra la prueba para medir el rizo de corriente:



Cálculo del límite de corriente y resistencia de sensado

La corriente pico que el convertidor buck puede proveer es determinada por el voltaje de umbral del límite del pico de corriente V_{LIM} y la resistencia de sensado R_{SENSE}.

La salida de corriente deseada de la fuente de switcheo es de 1.5 A, tomando el peor caso de rizo de corriente de la eq 2, calculamos el pico de corriente:

$$I_{PEAK,PWM\max} = I_{LOAD} + 0.5\Delta I = 1.5A + (0.5 * 338.6mA) = 1.67A \qquad \text{eq. 9}$$

Con la corriente pico calculamos la resistencia de sensado:

$$R_{Sense} = \frac{V_{LIM}}{2 * I_{PEAK,PWM}} m\Omega = \frac{80mV}{2 * 1.67A} m\Omega = 24m\Omega \quad \text{eq. 10}$$

R_{sense} Se utilizará de 25 mΩ ± 1%

Capacitor de salida

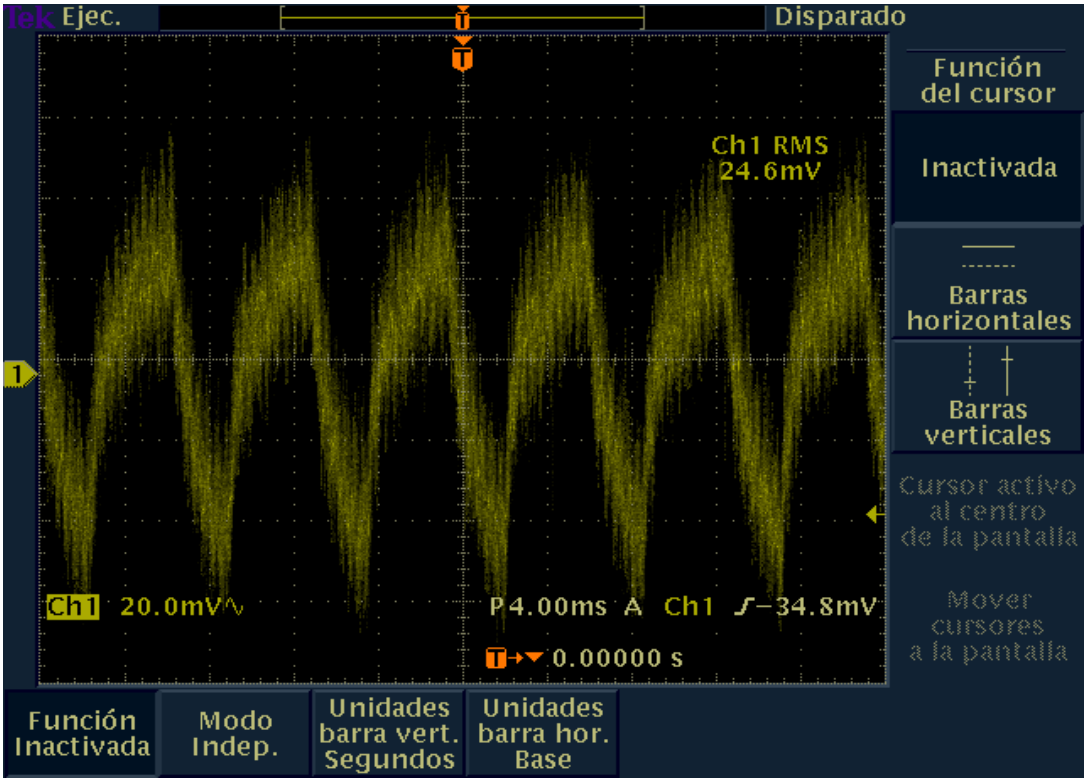
El ESR del capacitor de salida del buck tiene una gran influencia en el rizo de la salida, asi que los capacitores de tantalio son recomendados para esta aplicación. El capacitor elegido para esta aplicación es de 100uF@10V ±20% con 80mΩ ESR. Usando 2 capacitores en paralelo, reduce a la mitad el ESR y dobla la capacitancia, reduciendo el rizo de voltaje.

$$V_{(Ripple)} = \Delta I * \left(R_{ESRCOUT} + \frac{1}{8 * f_{sw} * C_{out}} \right) \qquad \text{eq. 11}$$

Tomando $I\Delta_{max290kHz}$ de la ecuación 2 como peor caso:

$$V_{(Ripple@16V,290kHz)} = 338.6mA * \left(40m\Omega + \frac{1}{8 * 270kHz * 160\mu F} \right) = 17.91mV \qquad \text{eq. 12}$$

El rizo de voltaje máximo será de **18.18 mV**

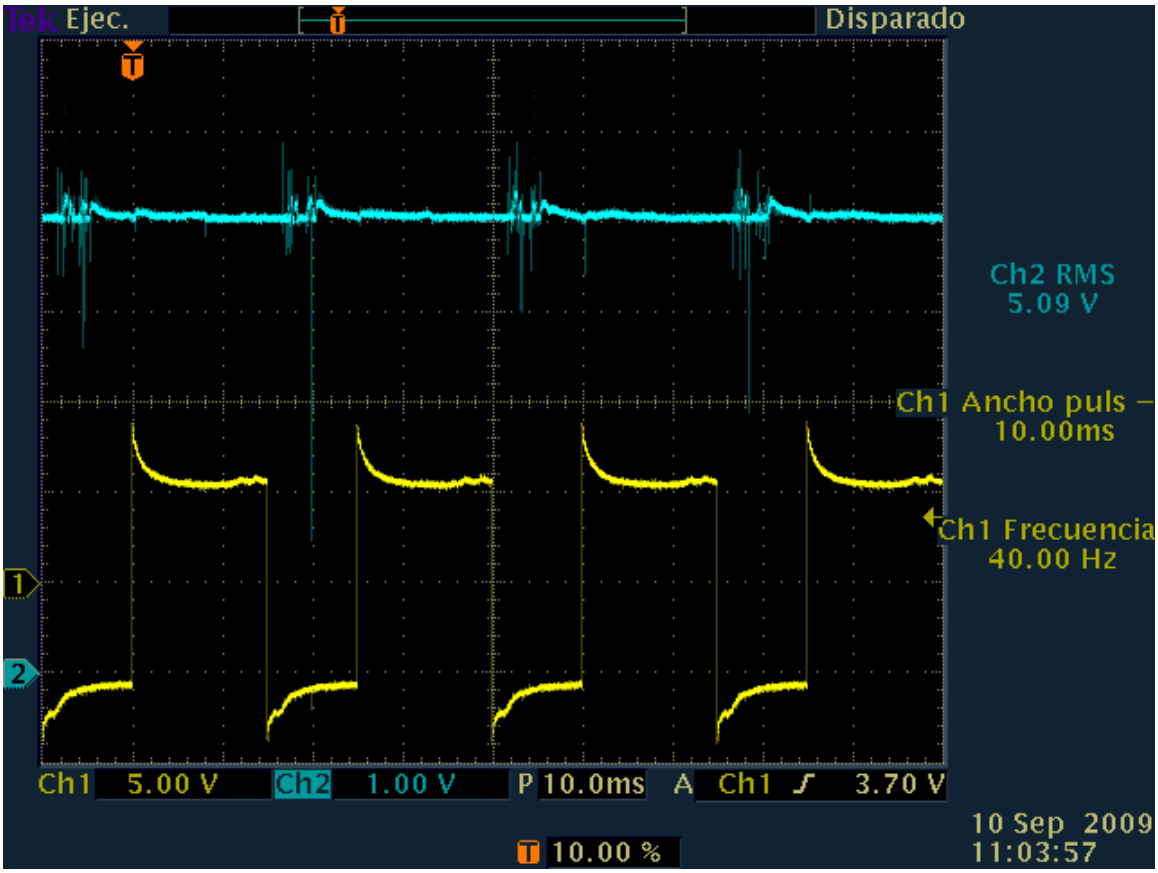


Signal: **Ch1** Ripple Voltage.

Se puede ver un poco más de rizo de voltaje, que puede ser por la precisión del equipo, pero aun así está más bajo que el requerimiento.

Capacitor de entrada

El fabricante recomienda un capacitor de entrada de 100 uF con uno en serie de 220nF, pero se decidió utilizar 2 capacitores de paralelo de 470uF y uno de 100n que la fuente soporte desconexiones breves de batería.



Señales Ch1 Señal activación del relevador, Ch2 Voltaje de salida.

MOSFET de switcheo

Para el MOSFET de switcheo se buscó uno que soportara al menos 3A, que fuera de grado automotriz y que estuviera aprobado por el departamento de calidad

Pos.	Type	R_{DSON}	I_D	Part number	V_{DS}	Temp.
1	IRFR9024NTR	175mΩ	-11A	IRFR9024N	-55V	150 °C

Ciclo de operación del 100% y dropout

El TLE6389 opera con un ciclo de operación de hasta el 100%. Esto permite que trabaje con el voltaje de dropout más bajo cuando el voltaje de batería es mínimo. El MOSFET es encendido de forma continua cuando el voltaje de alimentación se acerca al voltaje de salida, los reguladores de switcheo convencionales con menos del 100% de ciclo de trabajo podrían fallar en ese caso.

El voltaje de dropout es definido como la diferencia de voltaje entre la entrada y la salida cuando la entrada es suficientemente baja para que el voltaje de salida se caiga. El dropout depende de la resistencia de encendido de drain a source del MOSFET, la resistencia de sensado y la resistencia del inductor. El voltaje de dropout es proporcional a la corriente de carga.

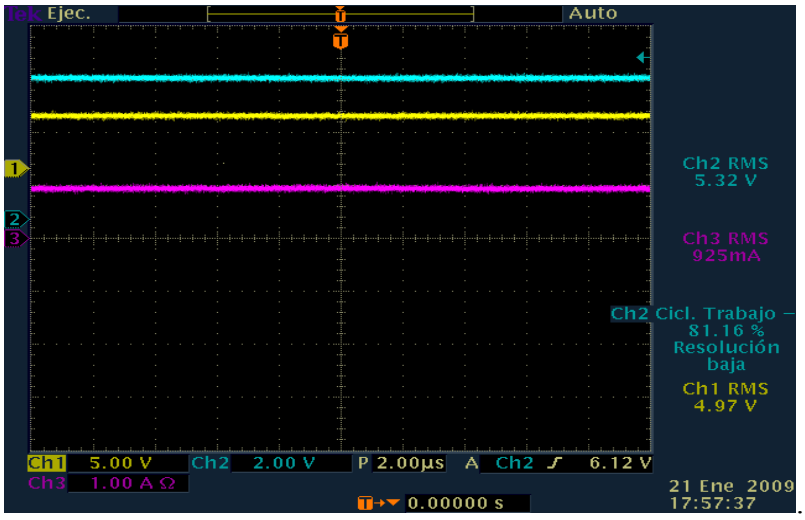
$R_{DS(ON)SW}$ = IRFR9024NTR = 175 mΩ, switch principal de la fuente de poder
R_{SENSE} = 25 mΩ
R_{BUCK_L} = 86mΩ con 47µH de inductancia, inductor de salida

$$V_{drop} = I_{LOAD} \cdot \left(R_{DS(ON)SW} + R_{SENSE} + R_{BUCK\_L}\right) \quad \text{eq. 18}$$

Con una corriente de carga de 1A:

$$V_{drop} = I_{LOAD} \cdot \left(R_{DS(ON)SW} + R_{SENSE} + R_{BUCK\_L}\right) = 0.29V$$

Esto nos dice que para que el circuito funcione debe haber por lo menos un voltaje de 5 + 0.29 = 5.29V @1A como voltaje de entrada.



Signals: **Ch1** Voltaje de Salida, **Ch2** Voltaje de entrada, **Ch3** Carga

Diodo de rueda libre

Los diodos schottky son recomendados para tener la pérdida más baja de energía como rueda libre por su alta velocidad entre sus estados de conducción directa e inversa.
Se recomienda utilizar un diodo de mayor capacidad que la operación nominal

Pos.	Diode Type	$I_{reverse}$	V_{RRM}	Part number	current	Temp.
1	**SS34 Schottky**	**20mA/40V/100°C**	**40**	**MBRS340-T3**	**3A/ 0.3V**	**125°C**

ANALISIS TÉRMICO

MOSFET de switcheo

Corriente pico: 1.67A.
R_{DSon} = 175mΩ.
$R_{\theta JA}$ = 50 ºC/W

La máxima temperatura de Junction es 150 ºC, por lo que el calor de la temperatura disipada por el componente + la temperatura ambiente (85 ºC en peor caso) debe ser menor.

$$T_J = \left(I_{pico}^2 * R_{DSon}\right) * R_{\theta JA} + T_{Amb} = (1.67^2 * 175m\Omega) * 50\frac{℃}{W} + 85℃$$

$$T_J = 109.4℃$$

Basado en los cálculos no debe haber ningún problema térmico con el componente, pero se recomienda usar suficiente superficie de disipación en el PCB para reducir la resistencia térmica.