Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Implantación de un Microcontrolador en un dispositivo FPGA.

Autor:

Vicente Herranz, Rodrigo

Tutor:

Andrés Rodríguez Trelles,
Francisco José de
Tecnología Electrónica

Valladolid, octubre de 2017

Resumen.

Este proyecto presenta el diseño completo, y la realización, de un sistema digital complejo, incluyendo un microcontrolador en un único dispositivo, tipo FPGA. Para conseguirlo, se ha utilizado las herramientas de Lattice Diamond y Lattice MicoSystem, así como técnicas innovadoras de diseño.

El diseño incorpora un sensor de ultrasonidos, que se ha utilizado para medir distancias, y una interface de usuario, basada en interruptores y visualizadores de 7 segmentos.

Se ha empleado una técnica de diseño original, en la que las interconexiones, entre el microcontrolador y los restantes elementos, se hace en base a esquemas, y se evita una descripción estructural, extremadamente compleja y con gran riesgo de errores en VHDL.

La programación del microcontrolador se realiza en lenguaje C, con lo que en el mismo diseño se emplean, esquemas, descripciones VHDL y verilog, herramientas de generación automática de bloques y lenguaje C.

El diseño se ha realizado de forma incremental, y se ha implantado en un prototipo completamente funcional.

Palabras Clave.

Palabras clave: FPGA - SoC – microcontrolador - VHDL – C

INDICE

Tabla de imágenes.

Tabla de gráficos.

1 Objetivos del TFG.

El objetivo fundamental es evaluar la posibilidad de realizar sistemas digitales completos en un único dispositivo, FPGA, económico.

La FPGA, evidentemente, no incluiría los sensores, pero debería incorporar un microcontrolador, programable por el usuario, y toda la circuitería auxiliar, incluyendo la lógica de manipulación de la señal proporcionada por los sensores.

La parte de acondicionamiento y pretratamiento de la señal se haría en la lógica de propósito general del dispositivo FPGA, de forma que el microcontrolador se dedicaría, exclusivamente, al post-procesamiento de la información.

Para evaluar las posibilidades, se va a realizar un proyecto complejo concreto, que consistirá en el desarrollo e implementación de un sistema que incorpore un sensor de ultrasonidos (que se usará para medir distancias), la lógica de interface de dicho sensor, y un microcontrolador de 8 bits, para el post procesamiento de la información.

El microcontrolador y la lógica se implementarán en un único dispositivo FPGA, de la familia MachXO2 de Lattice, disponible en el Departamento de Tecnología Electrónica.

Para conseguir el objetivo principal presentado, se deberían conseguir otros objetivos parciales:

-Realizar un estudio de las diferentes alternativas disponibles, dentro del campo de dispositivos tipo FPGA.

-Analizar los sensores de ultrasonidos disponibles, y seleccionar el más adecuado para esta aplicación.

-Usar técnicas modernas de diseño, que incluirán:

- Jerarquización del diseño.

-Lenguajes de descripción de hardware (Verilog y VHDL)

-Interconexión de bloques, en base a esquemas.

-Empleo de lenguajes de alto nivel (C) para la programación del microcontrolador.

-Empleo de herramientas de simulación y síntesis lógica, y análisis de los recursos empleados en el dispositivo FPGA.

-Empleo de la herramienta Lattice Mico

-Estudio del microprocesador Mico 8

2 Introducción.

En este trabajo se tratará de implementar un microcontrolador en un dispositivo FPGA.

Para ello se utilizará un sensor de ultrasonidos que, junto con el microcontrolador y otros bloques (que representan otros dispositivos electrónicos digitales), permitirá mostrar en unos visualizadores de 7 segmentos la distancia a la que se encuentra un objeto, con una precisión elevada y otros datos, que serán generados en el interior del microcontrolador.

Este trabajo demostrará que es posible implementar el microcontrolador en la FPGA junto con más lógica independiente del microcontrolador y que trabajen de forma conjunta y cooperativa.

Fomentará su uso en máquinas y dispositivos que requieran unas elevadas prestaciones (velocidad y fiabilidad), así como la versatilidad para poder implementar circuiterías adicionales al microcontrolador en un mismo dispositivo.

El uso de un pequeño procesador introduce ventajas como puede ser la mejora del cálculo y simplicidad de programación. Si realizásemos las operaciones aritméticas con bloques funcionales en la FPGA haría que el esquema creciese en tamaño y su diseño fuese más difícil.
Mientras que, si utilizamos un Microcontrolador, simplemente añadimos el bloque a nuestro esquema y lo programamos en lenguaje C con las operaciones a realizar.

Además, queda integrado todo en un solo Chip haciéndolo más fácil de manejar.

En este capítulo se seleccionan los sensores, FPGA y el resto de material empleado, así como el software de diseño.

2.1 Introducción a los sensores de ultrasonidos.

Es un tipo de sensor que emite una onda en un determinado rango de frecuencia y mide el tiempo que la señal tarda en regresar. La señal emitida por el sensor se refleja en un objeto y el eco se convierte en señales eléctricas, las cuales son elaboradas en el aparato de cuantificación.

Los materiales a detectar pueden ser sólidos, líquidos o polvorientos, pero han de ser reflectores de sonido.

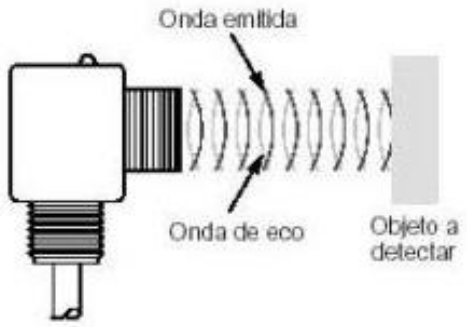


Figura 1 Funcionamiento Sensor de ultrasonidos

Pueden ser utilizados en interiores y exteriores, sin embargo, una desventaja que posee esta línea de sensores es que tienen un rango de medida limitado, no pudiendo medir distancias inferiores a 6 pulgadas.

2.1.1 Funcionamiento del sensor.

El sistema de medida se compone de dos módulos, el módulo electrónico, y el transductor.

El transductor, de tipo electrostático o piezoeléctrico, se usa tanto para emitir el pulso, como para recibir el eco.

La distancia del transductor al objetivo, lo determina el módulo electrónico que conoce la velocidad del sonido en el aire, y el intervalo de tiempo transcurrido entre la emisión y la recepción de la señal (proporcionado por el transductor).

La parte más importante del sistema es el transductor electrostático. Está compuesto por dos electrodos, negativo y positivo. El electrodo negativo es una membrana muy fina, recubierta de oro para formar un diafragma de

vacío. El electrodo positivo, es una lámina recubierta de aluminio que, además, sirve como estructura resonante para el diafragma. La señal no se transmite de forma lineal, a partir del transductor, sino que se genera una señal que se extiende con forma de cono.

En la parte electrónica, la señal se amplifica en un pequeño transformador incluido en la placa. La componente continua se mantiene en el transductor, por medio de un condensador, mientras éste actúa como micrófono para recibir el eco. El transductor se bloquea durante unos pocos microsegundos, para evitar la recepción del eco que se genera al salir la señal del propio transductor, y a partir de ese instante pasa a actuar como un micrófono.

Para compensar la pérdida de energía de la señal al recorrer mayores distancias, la ganancia del amplificador que trabaja en la recepción de la señal, se va incrementando en función del tiempo, además de disminuir en ancho de banda para disminuir los efectos del ruido. Cuando una señal es recibida por el sistema, y si esta señal supera un nivel umbral mínimo, se activa una fuente de corriente, que va cargando un condensador hasta que alcance éste los 1.2V. En este instante, se considera que la señal recibida es el eco, y se genera una señal lógica.

Para medir el intervalo de tiempo transcurrido entre las dos señales hay que recurrir a un circuito externo. Al multiplicar este valor por la velocidad del sonido, y teniendo en cuenta que la señal se ha recorrido dos veces, dividiendo entre dos, se obtiene la distancia a la que se encuentra el objeto del transductor.

2.1.2 Características de los sensores de ultrasonidos:

Los parámetros más importantes que caracterizan un sensor de ultrasonidos son los siguientes:

Zona muerta: Los sensores de ultrasonidos tienen una zona muerta en la cual no pueden detectar exactamente el objeto u obstáculo.

Máximo rango sensible: Es la distancia máxima en la que se puede detectar cada objeto y cada aplicación. Se determina mediante experimentación.

Angulo de emisión: Es el ángulo del cono que se emite. Está formado por los puntos del espacio en los que la señal del sensor es atenuada por lo menos 3 dB

Diámetro del cono de emisión: El sensor de ultrasonidos emite un haz de sonido en forma de cono que elimina los lóbulos laterales. Es importante el tamaño del objeto respecto del tamaño de la zona que abarca el haz. Teóricamente, el objeto más pequeño detectable es aquel que mide la mitad de la longitud de onda de la señal del sensor de ultrasonidos.

Frecuencia de disparo: Es la máxima frecuencia a la que un sensor es capaz de dispararse o pararse. Depende de varias variables, las más significativas son: el tamaño del objeto, el material del que está hecho y la distancia a la que se encuentra.

Inclinación del haz de ultrasonidos: Si un objeto liso es inclinado más de ± 3 grados con respecto a la normal al eje del haz de emisión de la señal de ultrasonidos, parte de la señal se desvía del sensor, y la distancia de detección disminuye. Sin embargo, para objetos pequeños situados cerca del sensor, la desviación respecto a la normal puede aumentar hasta ± 8 grados. Si el objeto está inclinado más de ± 12 grados respecto a la normal, toda la señal es desviada fuera del sensor y el sensor no recibiría el eco.

2.1.3 Ventajas de los sensores de ultrasonidos.

Los sensores de ultrasonidos miden y detectan distancias de objetos en movimiento. Detectan objetos pequeños a distancias grandes. Son resistentes frente a perturbaciones externas tales como vibraciones, radiaciones infrarrojas, ruido ambiente y radiación EMI. No son afectados por el polvo, suciedad o ambientes húmedos. No es necesario que haya contacto entre el objeto a detectar y el sensor.

Otra ventaja es que miden distancias sin necesidad de contacto midiendo el tiempo que tarda el sonido desde que deja el transductor hasta que vuelve a él, por lo que la medida de la distancia es sencilla y exacta con un margen de error de 0.05 %.

Además, los sensores ultrasónicos pueden cubrir tanto áreas anchas como estrechas.

Permiten detectar todo tipo de materiales sea cual sea su composición. El material detectado puede ser claro, sólido, líquido, poroso, blando y puede tener cualquier color.

Los sensores de ultrasonidos detectan a distancias superiores a los 6 metros mientras que otros tipos de sensores, como los inductivos, no pueden. Aunque, los sensores fotoeléctricos si son capaces, estos requieren ser calibrados periódicamente si existen vibraciones y, carecen de la habilidad de detectar sobre áreas grandes sin usar un número elevado de sensores.

2.1.4 Desventajas de los sensores de ultrasonidos.

Existen propiedades físicas que inducen errores de medida a los sensores ultrasonidos, y será necesario utilizar otro tipo de sensores como los que se contemplan en el apartado: *2.2 Alternativas a los sensores de ultrasonidos para medir distancia.*

Las causas de error en sensores de ultrasonidos más comunes son las siguientes:

La Temperatura: La velocidad del sonido en el aire depende de la temperatura.

La Presión del aire: Los cambios normales en la presión atmosférica del aire no tienen efectos sustanciales en la exactitud de la medida, pero si estos cambios son grandes, si podrían afectar a las mediciones

Turbulencias en el aire. Las corrientes de aire, turbulencias, y capas de distinta densidad causan refracción de la onda de sonido, que afecta a la medición.

2.2 Alternativas a los sensores de ultrasonidos para medir distancia.

Los sensores de ultrasonidos no son la única alternativa para medir distancias. También se puede emplear:

Sensores Laser: estos sensores se fundamentan en el fenómeno físico de la propagación de la luz y el tiempo empleado en el mismo. Se emite un haz de luz y se refleja en un objeto. Se mide el tiempo que el haz de luz necesita para realizar el recorrido del emisor al objeto y del objeto al receptor. Ya que la velocidad de la luz es constante, el tiempo de propagación permite calcular la distancia.

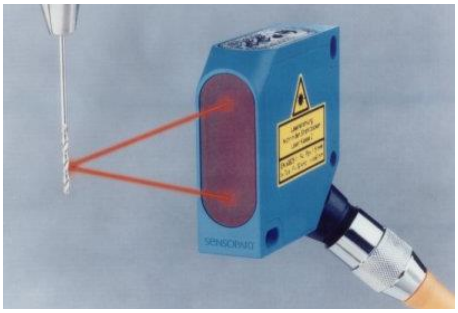


Figura 2 Sensor Laser

Sensores Infrarrojos: son sensores de medición de distancia, que se basan en un sistema de emisión/recepción de radiación lumínica en el espectro de los infrarrojos (menor que las ondas de radio y mayor que la luz). Estos sensores presentan el inconveniente de ser sensibles a la luz ambiente, como consecuencia de que los rayos de sol también emiten en el espectro de luz infrarroja. Por este motivo se utilizan habitualmente en entornos con iluminación artificial de forma predominante (interiores).

Figura 3 Sensor de Infrarrojos

Magnético e Inductivos: estos sensores trabajan generando un campo magnético y detectando las pérdidas de corriente de dicho campo generadas al introducirse en él los objetos de detección, férricos y no férricos. Al aproximarse un objeto metálico, se inducen corrientes de histéresis en el objeto, sin embargo, esto implica que hay que usar materiales metálicos y no

metálicos lo cual no es útil para esta aplicación, y además las distancias de medición son más pequeñas.

Figura 4 Sensor Inductivo

Debido a toda la información expuesta anteriormente se elige el sensor de ultrasonidos como el más adecuado para el proyecto.

El sensor se adecua perfectamente al proyecto porque funciona con solo un voltaje 3.3V haciéndolo idóneo para integrarlo en un circuito digital. La FPGA con la que trabajo no posee entradas analógicas (aunque se podrían añadir) y por ello el sensor me permite conectarlo directamente y trabajar con pulsos de distintas anchuras funcionando en su conjunto como un circuito digital más grande.

Dentro de la variedad de sensores existentes en el mercado se empleará el sensor de ultrasonidos, ya que se dispone de él, en el Laboratorio de Tecnología Electrónica, y se adecua perfectamente a las necesidades del proyecto.

2.3 Dispositivos FPGA.

En los siguientes subapartados se presentan los dispositivos FPGA, analizando brevemente sus características, y seleccionando el que se va ha emplear en este proyecto.

2.3.1 Introducción y fabricantes.

Un dispositivo FPGA (Field Programable Gate Array) es un dispositivo programable mediante bloques de lógica combinacional y secuencial, cuya interconexión y funcionalidad se puede describir mediante un lenguaje de programación de descripción especializado, VHDL o VERILOG, de tal manera que al ejecutar la herramienta de síntesis lógica y cargar el fichero de configuración, introducimos los bloques descritos en este lenguaje en el dispositivo. En este proyecto se utiliza VHDL debido a que es el que se imparte en la asignatura de Sistemas Electrónicos Reconfigurables.

La lógica que se puede implementar permite crear funciones tan sencillas como las llevadas a cabo por una puerta lógica, u otras mucho más complejas como un microcontrolador como el que se tratará de crear, que contiene un microprocesador, un bus de datos, memoria y periféricos.

Las FPGAs tienen la ventaja de ser reprogramables aumentando claramente su flexibilidad en cuanto a realizar modificaciones en poco tiempo.

Las FPGAs fueron creadas en el año 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx como una evolución de los CPLDs.

Los CPLDs y las FPGAs se basan en matrices de elementos lógicos programables.

Sin embargo, la densidad de los elementos lógicos programables en puertas lógicas (número de puertas NAND equivalentes que podríamos programar en un dispositivo) es muy superior en las FPGA que en los CPLDs. Mientras que en un CPLD hallaríamos del orden de decenas de miles de puertas lógicas programables, en una FPGA hallaríamos del orden de cientos de miles hasta millones de ellas.

La arquitectura de las FPGAs incorpora un mayor número de biestables con una gran libertad de interconexiones entre bloques.

Existen dispositivos con grados de complejidad muy diferentes, desde los que incorporan un pequeño número de LUTs, hasta los que incluyen elementos preconstruidos, funciones complejas como sumadores, multiplicadores, memorias, etc.

2.3.1.1 Fabricantes Principales de FPGAs.

Existen 4 principales fabricantes de FPGAs que llevan en el mercado desde las ultimas 2 décadas, las 4 compañías poseen buenos e innovadores productos y también similares (en cuanto a FPGAs), estas son las siguientes:

Xilinx, Altera, Actel y Lattice.

Todos ellos disponen de dispositivos que pueden satisfacer las especificaciones (y muchas más), dependerá del uso que se le vaya a dar a la FPGA.

Se ha optado por la última porque se dispone de él en el departamento de Tecnología Electrónica de esta Universidad.

2.3.2 Familia de FPGA MACHO2.

La familia de FPGAs MachXO2, es una gama de FPGAs del fabricante Lattice Semiconductor.

Esta familia de dispositivos contiene un gran número de LUTs (unidad que usa la FPGA para implementar lógica sin tener que realimentar a otro LUT, esta unidad permite establecer una tabla de verdad o configurar la salida deseada), variable en función del modelo, donde encontraremos desde 256 a 6864 LUTs

La arquitectura de las MachXO2 organiza los bloques funcionales en la siguiente jerarquía:

- PFU.
  - Slice.
    - LUT.

A continuación, se explica cada uno de ellos de menor a mayor. Cada vez que se sube en la jerarquía se incluye unidades de la jerarquía inferior, interconectadas junto con más elementos:

Una LUT es una memoria, un circuito básico que me permite implantar funciones mediante tablas.

Se puede representar una LUT3 por el siguiente circuito:

Truth Table - A + (C*B')			
Input A	Input B	Input C	Output Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

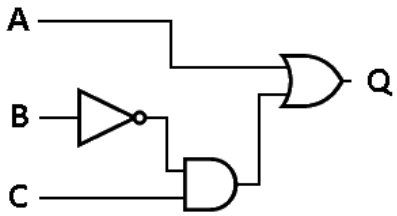


Figura 5 Realización de la función lógica indicada, mediante una LUT3

Una LUT4 sería una memoria de 16 palabras de 1 bit. Podré hacer cualquier función lógica de 4 variables y se podrá emplear si no hace falta como lógica, como memoria.

Dependiendo del número de entradas que posea la LUT será LUT3, LUT4, LUT5 o LUT6. De tal manera que si una LUT3 permite 8 posiciones de memoria (una función de 3 variables), una LUT4 permitirá una función de 4 variables, una LUT5 permitirá 5 …

La familia MACHXO2 contiene LUT4s que dependiendo como se agrupen se podrá optar a LUTs de mayor orden para ir creando lógica más compleja.

Asociando 2 LUT4 con dos biestables de tipo Latch junto con la lógica y las conexiones pertinentes, se constituye la siguiente unidad funcional, el Slice.

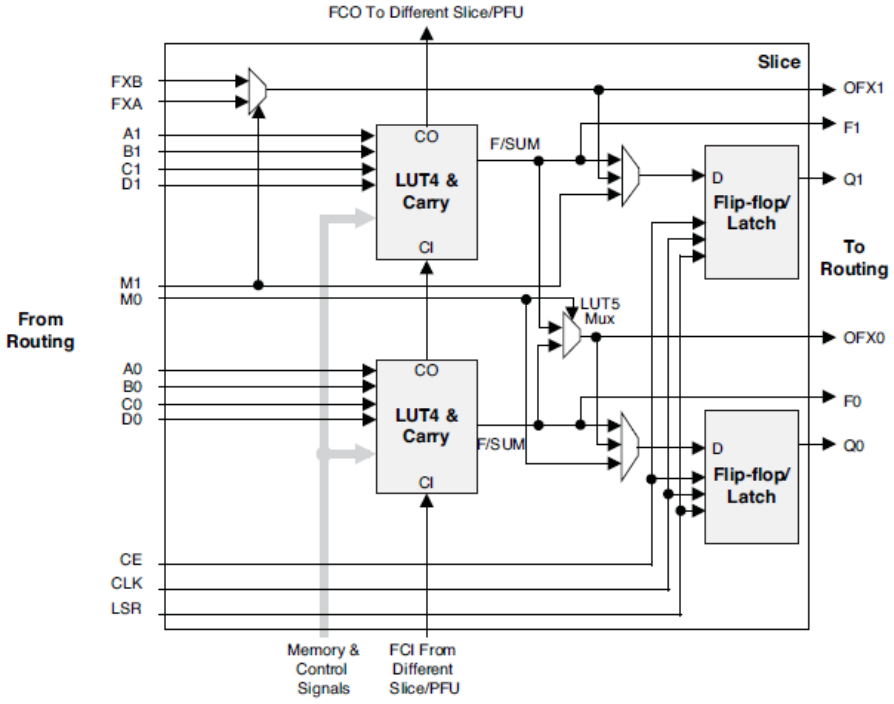


Figura 6 Contenido de un Slice de una FPGA MachXO2

La siguiente unidad funcional es el PFU que agrupa 4 Slices interconectados que ya permite la construcción de registros, memoria RAM y bloques aritméticos.

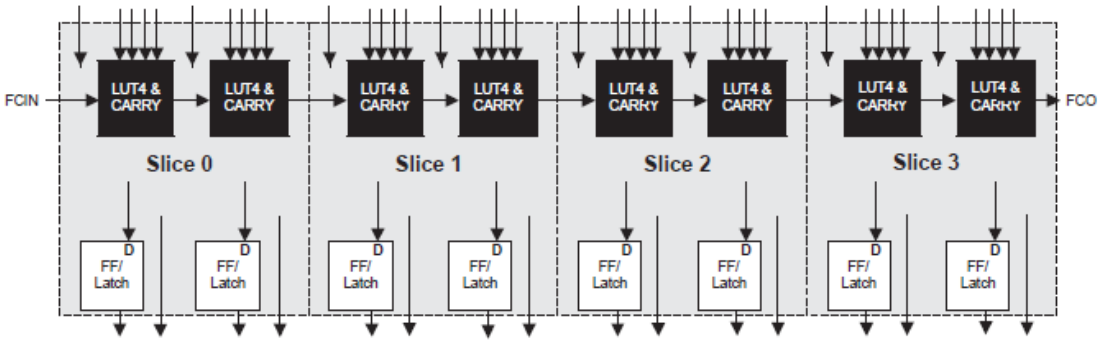


Figura 7 Contenido de un PFU de una FPGA MachXO2

En cada PFU los Slices 0-3 contienen dos LUT4s que alimentan dos registros. Los Slices 0-2 pueden ser configurados como memoria RAM. Además, cada PFU contiene lógica que permite combinar LUTs para aumentar su funcionalidad como LUT5, LUT6, LUT7 and LUT8. También contiene un control lógico que permite establecer los estados set/reset (que dependiendo de si se conecta a una señal de reloj, se puede hacer que la lógica funcione como síncrona o asíncrona), también tiene una señal de clock select, y funciones ocultas de RAM/ROM.

Además, esta familia de FPGAs incluye distintos bloques preconstruidos y listos para utilizar:

- Embeded Block RAM (EBR).
- Distributed RAM.
- User Flash Memory (UFM).
- Phase Locked Loops (PLLs).
- Controladores SPI.
- Controlador de I2C.
- Temporizadores y Contadores.

Y otros bloques como por ejemplo un Oscilador Interno o la lógica de distribución de señales de reloj,etc.

Dependiendo del modelo la arquitectura será distinta, en el caso de mi trabajo, se usará la MachXO2-1200-ZE, en la que podemos ver por ejemplo como se organizan los bloques:

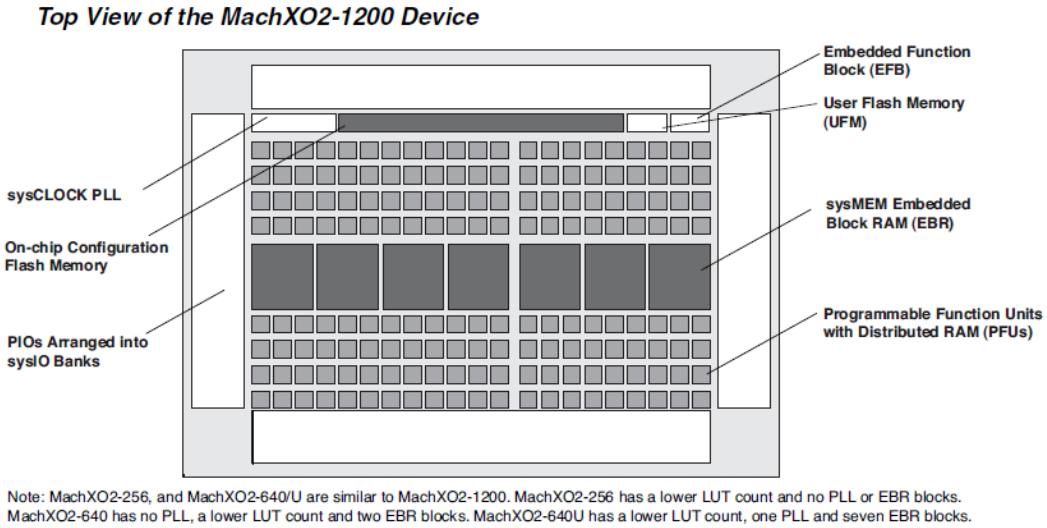


Figura 8 Interior MachXO2-1200

Dentro de la familia se encuentran FPGAs que consumen más o menos potencia, esto dependerá de la terminación de las mismas ZE (de baja velocidad), HE (alta velocidad). Dependiendo de esto, los voltajes de alimentación, entradas y salidas serán distintos.

Los dispositivos HE consumen mayor cantidad de energía que los ZE debido a que su funcionamiento es más rápido, dependiendo de la función a desempeñar será conveniente emplear unos u otros.

Todos y cada uno de los bloques nombrados anteriormente, quedan explicados perfectamente en la Hoja de características de la familia MachXO2, que se adjuntar en el apartado de Anexos.

2.4 Software empleado.

Se han empleado dos programas del Fabricante Lattice de dispositivos electrónicos reconfigurables

2.4.1 Lattice Diamond.

Lattice Diamond es un software de Lattice Semiconductor con distintas herramientas para el diseño e implementación de lógica en dispositivos reconfigurables.

Su Herramienta de síntesis (Synplify Pro) esta optimizado para reducir el consumo de Slices (celdas físicas reconfigurables del dispositivo) de las FPGAs y que la configuración sea rápida y sencilla.

Este software tiene múltiples herramientas, entre ellas destacan los entornos de programación y configuración, para crear el código que configure la FPGA, que puede ser tanto en VHDL como en Verilog. También incluye entornos para la configuración de las patillas a utilizar del dispositivo seleccionado, para el diseño a través de esquemas funcionales, entornos de simulación de señales "programadas", como reportes de recursos utilizados del dispositivo.

Ademas, contiene muchas ventanas y opciones de configuración avanzadas que permite implementar diseños complejos en muchos dispositivos de FPGAs diferentes, como la herramienta IPExpress, que se usará en el proyecto para generar elementos automáticamente, o librerias con elementos preconfigurados.

El software de Lattice Diamond es de licencia gratuita, basta con registrarse para obtener la licencia del programa.

Se utilizará este software porque es el utilizado por el Departamento de Tecnología Electrónica de la Escuela de Ingenierías Industriales de la UVa para estos fines.

2.4.2 Lattice MicoSystem.

Lattice MicoSystem es un software de Lattice Semiconductor que sirve para implementar un microcontrolador de 8 bits optimizado para la familia MachXO2 ™ de Dispositivos Lógicos Programables (PLD).

Se trata de un software gratuito (previo registro) y que complementa a Lattice Diamond (sin él no funciona).

Lattice MicoSystem se utilizará para implementar un sistema de microcontrolador, con componentes periféricos, memorias y otros elementos conectados entre sí, buscando el consumo de recursos mínimo del dispositivo, de tal manera que permite usar el resto de la FPGA para otras tareas.

El programa está creado en un entorno de Eclipse, este contiene distintas herramientas, entre ellas se utilizarán las siguientes:

La herramienta MSB perspective, es un entorno donde configurar el hardware de un microcontrolador.
En este entorno se deberán selecciona los componentes deseados (para constituir un microcontrolador a medida) y se interconectan entre si haciendo uso de los distintos buses que se nos ofrecen.

La herramienta C / C ++ Development Tools, es un entorno de desarrollo de software donde se creará y modificará la aplicación que controla el microcontrolador y los componentes. Este contiene funciones propias para utilizar los elementos ubicados en la herramienta MSB.

2.5 Material empleado.

En los siguientes apartados se presentan los elementos de tipo hardware, que se han empleado en el proyecto.

2.5.1 FPGAs y Breakout Board.

Se analizan por separado los dos elementos que constituyen el dispositivo.

2.5.1.1 Análisis de las FPGA

Análisis y estudio de los dispositivos FPGA a utilizar (solo del chip, sin la placa):

Product *MachXO2*

Series LCMXO2-7000HE-4TG y LCMXO2-1200ZE-4TG.

En el proyecto se utilizarán las dos series de FPGAs, y se compararan los recursos empleados de cada una de ellas, estos dos dispositivos tienen la misma huella y patillaje, por lo que, a continuación, solo se indicará la información que es distinta por separado.

Es importante asignar el dispositivo al proyecto. Existen muchos tipos de FPGAs y PLDs pero yo usaré concretamente las dos anteriormente mencionadas, cuyas características son las siguientes:

Package / Case *TQFP-144*

Number of I/Os *114 I/O → para MachXO2-7000HE*

Number of I/Os *107 I/O → para MachXO2-1200ZE*

Esto implica que los chips elegidos contienen 144 patillas, de las cuales 114, pueden ser configuradas como entradas y salidas, en el caso de la FPGA *MachXO2-7000HE*, y 107 en la FPGA *MachXO2-1200ZE* .

Dependiendo de cómo se configure el dispositivo, podré utilizar todas ellas o no, ya que algunas pueden quedar bloqueadas internamente.

Mounting Style *SMD/SMT*

Si pretende montar el chip sobre una placa sin utilizar el Breakout board, la soldadura, que es del tipo SMD, obliga a diseñar una PCB capaz de conectar los 144 pines dentro de una superficie muy pequeña. Tarea complicada para un prototipo.

Por lo tanto, se utilizará la Breakout Board propia del fabricante, sobre la que ya viene soldado el chip y se extenderán las salidas respecto de esta, ya que al tratarse de un prototipo, no tengo problemas de espacio físico.

Características en cuanto a la programación:

Habrá que tener en cuenta que el dispositivo con el que trabajo, aunque tiene grandes características, tiene limitaciones, no podré exceder el número de LUTs que indica el fabricante, ni incluir elementos que no contenga la FPGA (como, por ejemplo 300 entradas y salidas, teniendo 114 la FPGA en uso).

En el software Diamond Tool existe una herramienta para mostrar los recursos utilizados y así poder ir gestionando el espacio del dispositivo.

Además, el dispositivo tiene embebido bloques funcionales de uso general, que no tendré que describirlos mediante código VHDL si quiero utilizarlos. Con el programa Diamond Tool puedo acceder rápidamente a ellos e incorporarlos. En el caso de no usarlos, simplemente permanecerán inactivos.

Las características de funcionamiento de las dos FPGAs que se van a utilizar son las siguientes:

De la FPGA *LCMXO2-7000HE:*

Distributed RAM *54 kbit*

Embedded Block RAM - EBR *240 kbit*

Maximum Operating Frequency *269 MHz*

Number of Logic Array Blocks - LABs *858*

Number of Logic Elements *6864*

Total Memory *550 kbit*

De la FPGA *LCMXO2-1200ZE:*

Distributed RAM *10 kbit*

Embedded Block RAM - EBR *64 kbit*

Maximum Operating Frequency *104 MHz*

Number of Logic Array Blocks - LABs *160*

Number of Logic Elements *1280*

Total Memory *138 kbit*

Las características físicas de alimentación y "ambientales" (temperatura), son las siguientes para ambos dispositivos, estas deberán darse para obtener un buen funcionamiento.

Maximum Operating Temperature + 70 C

Minimum Operating Temperature 0 C

Operating Supply Current 189 uA

Operating Supply Voltage 2.5 V/3.3 V

El precio del chip Lattice MachXO2-7000HE comprado directamente al fabricante es el siguiente que (dependerá del número de unidades):

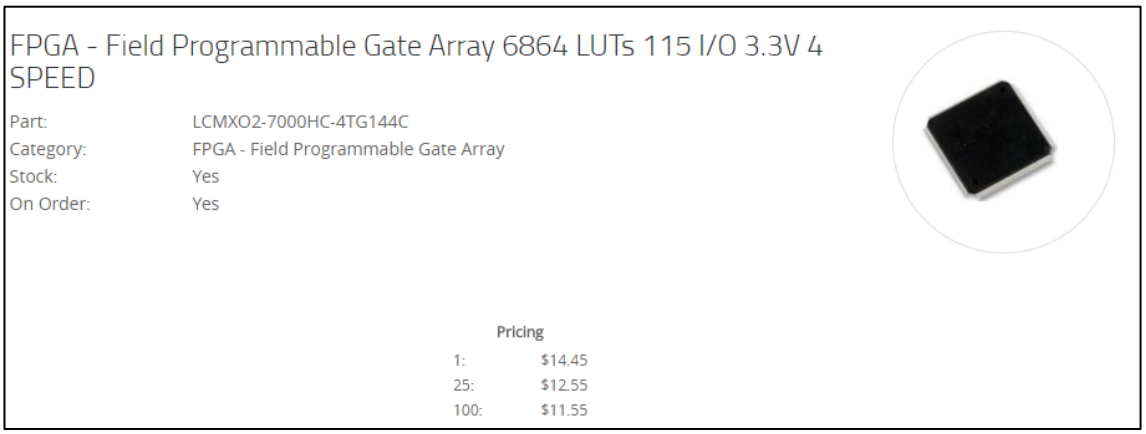


Figura 9 Precio MachXO2 7000

Este precio incluye únicamente la FPGA. Para proyectos de mayor envergadura al que yo voy a abordar, sería lo más conveniente.

En el caso de este proyecto, como se trata de un prototipo, se utilizará este chip embebido en una placa, la cual describo a continuación.

2.5.1.2 Análisis de la Breakout Board

Lattice MachXO2-1200ZE y 7000HE Breakout Board:

Se trata de los dispositivos explicados en el apartado: *2.5.1.1 Análisis de las FPGA*, pero con una PCB que me brinda una mayor comodidad a la hora de trabajar. El núcleo serán las FPGAs descritas anteriormente.

Figura 10 Breakout board

En la hoja de características del producto, encontramos todos los elementos que introduce la Breakout Board.

Con el siguiente dibujo y diagrama funcional, podremos identificar los elementos introducidos por esta placa.

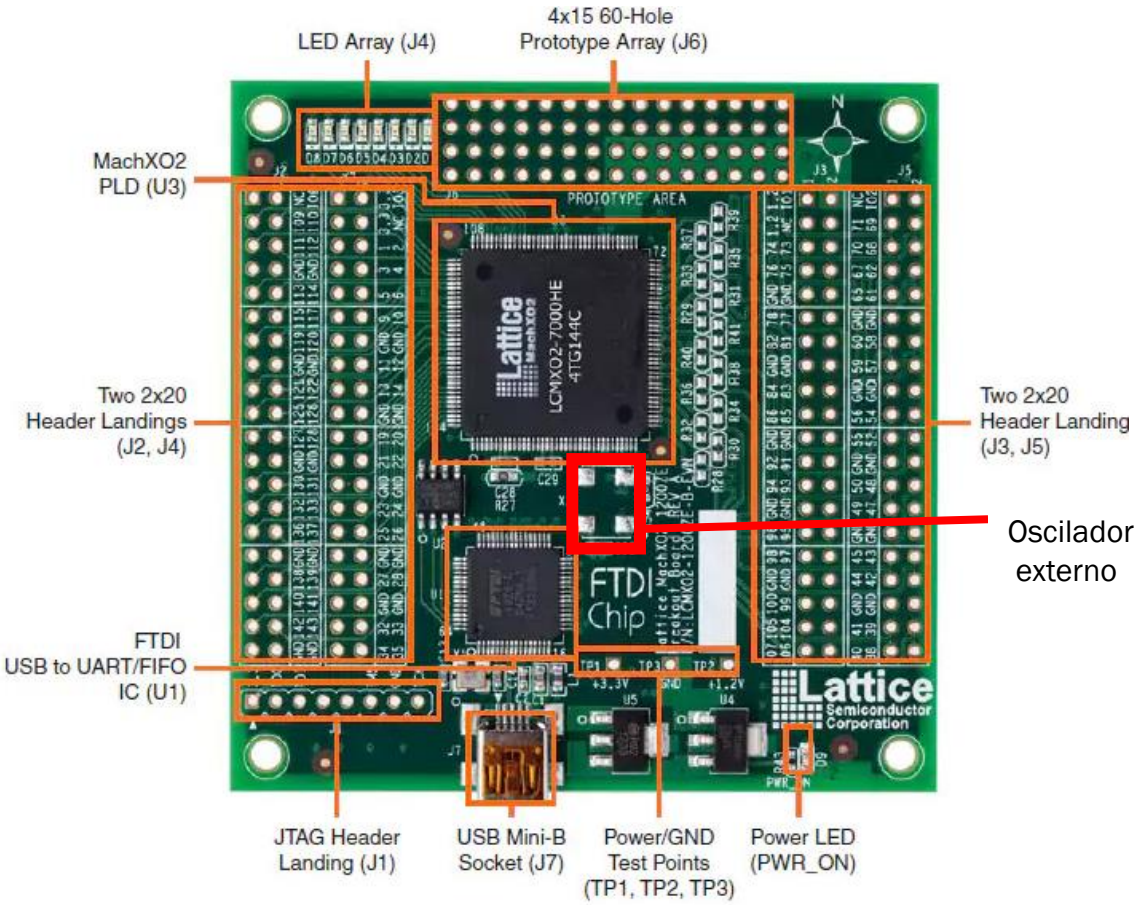


Figura 11 Elementos de la Breakout Board

Como vemos se incluyen elementos muy útiles como el Puerto USB, que me permitirá configurar y alimentar el dispositivo con facilidad desde un ordenador, mediante un cable USB.

Como vimos en las características de la FPGA, esta podía ser alimentada a 3.3V o a 2.5V, un voltaje poco usual en ordenadores o baterías (del tipo pilas alcalinas). Con este puerto y los dos reguladores lineales de tensión, se soluciona el problema y puedo alimentar el dispositivo a otras tensiones.

En la placa se distribuyen las entradas y salidas con un emplazamiento de fácil acceso para la realización de prototipos. Además, la huella (footprint) blanca, incluye la numeración de las patillas que se pueden utilizar de la FPGA.

Pin programado. ←→ Número en la huella de la Breakut board.

Como se puede observar en la PCB, existen pines de tierra al lado de cada pin de entrada/salida, con el objetivo de reducir interferencias.

También existen pines destinados a la comunicación serie con la FPGA, que podrían ser útiles en otros proyectos.

En la figura 11, se puede observar un hueco marcado en rojo, para posicionar un oscilador de mayor frecuencia que el incluido internamente en el chip. Con esto puedo hacer trabajar los biestables de la FPGA a mayor frecuencia, y, por lo tanto, lograré que mi microcontrolador trabaje más rápido. En este espacio se coloca un oscilador externo de 50 MHz.

La placa, además, contiene LEDs Rojos, con sus correspondientes resistencias limitadoras de corriente. Están unidas a pines de la FPGA. Estos funcionan en lógica negativa y serán utilizados en los diseños.

En la figura 12, se pueden ver los principales elementos de comunicación que permite la Breakot board.

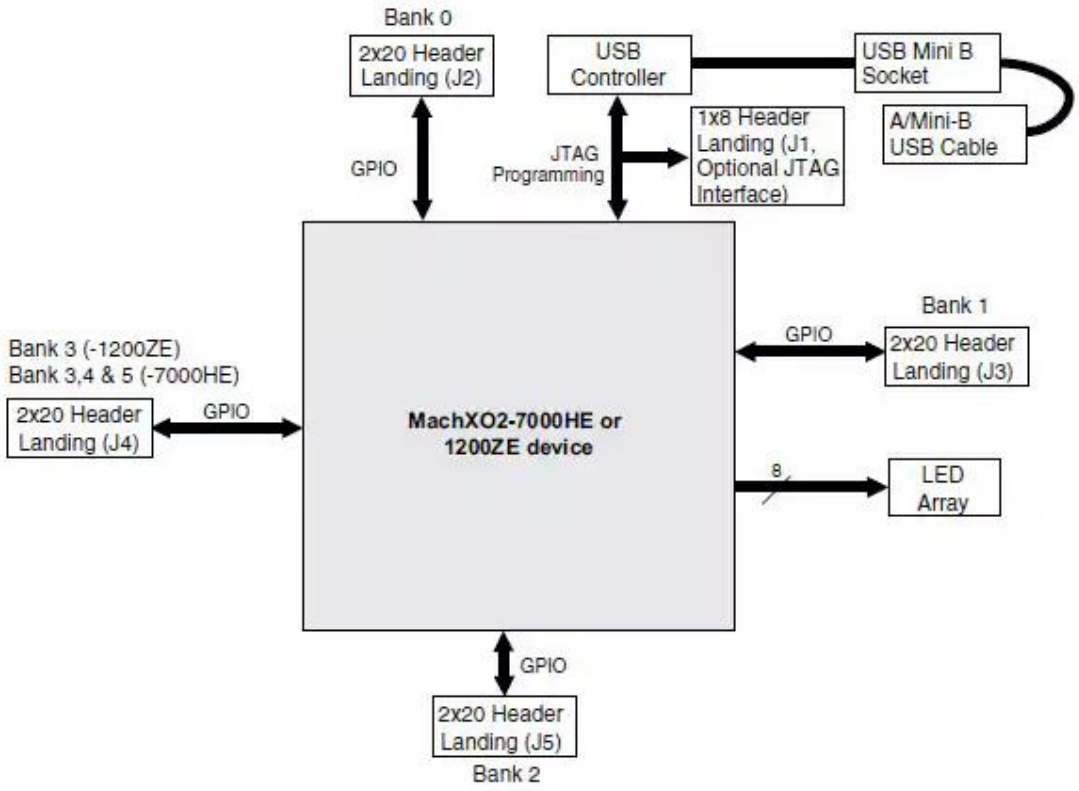


Figura 12 Esquema funcional de comunicación de la Breakout Board

Existen pines reservados para el chequeo del correcto funcionamiento del dispositivo. Mediante los métodos estudiados en la asignatura de Sistemas Electrónicos Reconfigurables, se puede realizar el diagnóstico de la FPGA.

Dichos pines, que van conectados a una lógica embebida y oculta en el dispositivo, son:

Description	MachXO2 Pin
Test Data Output	137:TDO
Test Data Input	136:TDI
Test Mode Select	130:TMS
Test Clock	131:TCK

Figura 13 Tabla pines de Chequeo

En el Anexo 2: Pines de entradas y salidas de las placas de expansión, se documenta la numeración de dichas patillas.

2.5.2 Entradas/Salidas y visualización.

Para la visualización y análisis de los resultados implementados en el dispositivo FPGA, será necesario poseer una serie de entradas y salidas que funcionen como interface hombre/maquina.

2.5.2.1 Visualización de E/S

A la Breakout board, descrita en el apartado anterior, se le añaden dos placas de expansión. Estos se unen mediante 2 buses de datos a una serie de pines que estaban libres de la placa.

Existe un documento que se adjuntará en el Anexo 2: Pines de entradas y salidas de las placas de expansión, en el que se observa la correspondencia entre los pines de la FPGA y la localización final en las placas de expansión.

Una de las placas contiene las entradas, formada por 8 interruptores (switch) y 8 pulsadores. La segunda placa está conectada a dos visualizadores de 7 segmentos con sus respectivas resistencias limitadoras de corriente.

Además, esta segunda placa con visualizadores contiene una serie de pines libres para conectar diferentes dispositivos electrónicos. En este proyecto se ubicará el sensor de ultrasonidos en estos pines.

2.5.2.2 Configuración de E/S en la FPGA

Será importante localizar las entradas y salidas previstas en nuestro diseño, a una entrada o salida del chip físico, que a su vez está conectado a un pin de la Breakout board, para que llegue a los pulsadores, switch, visualizadores de 7 segmentos, LEDs y al sensor de ultrasonidos.

Esta configuración se realiza en Lattice Diamond.

Tras abrir la ventana Process y hacer clic en “JEDEC File”, para que se ejecute la herramienta de síntesis, Synplify Pro. el proceso de creación del archivo de configuración se detendrá en el apartado Map Desing, antes de la creación del archivo Jedec (archivo de configuración de la FPGA).

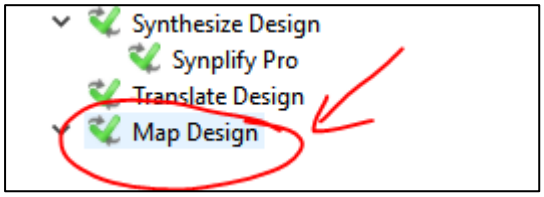


Figura 14 Map Design, para asignar pines

Cuando se haya sintetizado aparecerá (en la ventana Spreadsheet View) una lista de entradas y salidas correspondientes a las declaradas en el esquema o en el código VHDL. Se puede observar en la figura 16.

Con este suceso será necesario configurar las entradas y salidas (con la herramienta Spreadsheet View).

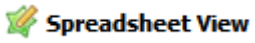


Figura 15 Spreadsheet View, para asignar pines

Al hacer clic sobre su icono (habiendo guardado y sintetizado antes), se abre una ventana con una tabla, donde podemos elegir los pines del dispositivo físico de la FPGA y configurarlos a elección.

Será necesario configurar las columnas:

- Pin (lo elegiremos de acuerdo con la salida que se quiera de la Beakout board unida a uno de los elementos mencionados en este apartado)
- IO_Type (que se configuraran a LVCMOS33 correspondiente a 3.3V de salida, ya que las resistencias de los LED están calculadas con este Voltage)
- PULLMODE (habrá que seleccionar DOWN)

	Name	Group By	Pin	BANK	BANK_VCC	VREF	IO_TYPE	PULLMODE	DRIVE	SLEWRATE	CLAMP	OPENDRAIN	DIFFRESISTOR	DIFFDRIVE
1	All Ports	N/A	N/A	N/A	N/A	N/A			N/A	N/A				
1.1	Input	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.1.1	Clock	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.1.1.1	sal_osc	N/A	27(27)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.2	e0	N/A	76(76)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.3	e1	N/A	75(75)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.4	e2	N/A	74(74)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.5	e3	N/A	73(73)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.6	e4	N/A	96(96)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.7	e5	N/A	95(95)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.8	e6	N/A	94(94)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.9	e7	N/A	93(93)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.1.10	pe	N/A	85(85)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2	Output	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.2.1	enc_osc	N/A	32(32)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.2	ps	N/A	9(9)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.3	s0	N/A	97(97)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.4	s1	N/A	98(98)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.5	s2	N/A	99(99)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.6	s3	N/A	100(100)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.7	s4	N/A	104(104)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.8	s5	N/A	105(105)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.9	s6	N/A	106(106)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)
1.2.10	s7	N/A	107(107)	1(1)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA)

Figura 16 Tabla de I/O Spreadsheet View

Tras la elección de las patillas se ejecuta de nuevo la herramienta de síntesis.

Si se han seleccionado correctamente las patillas se creará el fichero de configuración Jedec.

2.5.3 Sensor de Ultrasonidos.

El sensor que se va a utilizar en este trabajo es el LV-MaxSonar®-EZ

Figura 17 Sensor de Ultrasonidos

Se trata de un sensor de la línea de sensores ultrasónicos fabricados por MaxBotix.Inc.

La característica principal de este sensor es que está construido sobre pequeña PCB, que contiene un primer tratamiento de la señal del sensor, y ofrece una salida en forma de pulso variable, en función de la distancia que se detecte.

El origen de emisión se encuentra en el extremo del transductor

Figura 18 Rango alcance del sensor ultrasonidos

Este sensor ha sido elegido porque posee las siguientes ventajas:

- **Rango de detección elevado:** Es capaz de detectar objetos a 6.45 metros de distancia.
- **Gran resolución:** 1 pulgada (2.14 cm) dentro del rango de detección.
- **Múltiples Interfaces:** Puede ser leído de diferentes formas: Ancho de pulso, RS232 Serial y Voltaje Analógico (aunque se utilizará la opción de ancho de pulso).
- **Versatilidad de Alimentación:** Funciona con una alimentación desde 2.5V hasta 5V.
- **Tamaño:** Posee un diseño pequeño y muy ligero.

El sensor viene soldado a una PCB con siete pines, ya que puede ser utilizado en distintos modos. Haciendo uso de su hoja de características me informo de para qué sirve cada uno de ellos:

Figura 19 Cara posterior de Placa del sensor ultrasonidos

- **Pin 1 (BW):** Sirve para la comunicación serie la cual no voy a utilizar puesto que se usará la salida por PWM (ancho de pulso), y por lo tanto quedará al aire libre.

- **Pin 2 (PW):** Este pin emite una representación de ancho de pulso de rango. Es decir, da una señal (un voltaje) durante un tiempo

determinado. Si cuantifico el tiempo que ha estado la señal habilitada, se sabrá la distancia medida. La distancia se puede calcular utilizando el factor de escala de147uS por pulgada.

- **Pin 3 (AN):** Produce como salida un voltaje analógico, con un factor de escala de (Vcc / 512) por pulgada. Con una fuente de 5V produce ~ 9.8mV / pulgada mientras que con una de 3.3V produce ~ 6.4mV / pulgada.
  Esta sería la opción más ideal para otros microcontroladores que incluyen entradas analógicas, dependiendo del voltaje de entrada (al microcontrolador) se obtiene un valor numérico dentro de unos márgenes. Esta opción podría ser implementada también con la FPGA, pero implica añadir componentes al diseño y sería igual de valida que la opción utilizada en los diseños.

- **Pin 4 (RX):** Este pin está internamente forzado a 1. El LV-MaxSonar-EZ medirá continuamente distancias, y dará una salida si RX no se cambia de valor o está a 1. Si se deshabilita esta señal, el sensor dejará de medir y emitir salidas. Hay que habilitarlo durante al menos 20µs para que el sensor comience a medir.

  Por lo tanto, este pin es el que se utiliza de disparador o Trigger, para comenzar las mediciones o parar el sensor.

- **Pin 5 (TX):** Este pin, es una salida del sensor, comunica el valor de la medición, mediante un protocolo serie RS232. No se utilizará en este proyecto.

- **Pin 6 (+5V- Vcc):** Pin de alimentación del sensor, necesario para que el sensor trabaje. Funciona entre 2.5V – 5.5V. Capacidad de corriente recomendada de 3mA para 5V, y 2mA para 3V.

- **Pin 7 (GND):** Conexión de retorno a tierra. Sirve tanto para retornar el voltaje como para eliminar el ruido. El fabricante recomienda que sea libre de ruido y no se conecte con otras tierras para una operación óptima.

El modo de funcionamiento empleado es el siguiente:

Los pines 6 y 7 estarán conectados a Tensión y Tierra respectivamente, para alimentar el sensor.

Usaré el pin RX, ya descrito anteriormente, para iniciar los disparos en los que comienza las mediciones. Cada vez que quiera comenzar una medición pondré el pin a cero 32µS, ya que como mínimo tengo que deshabilitarlo 20µS y es conveniente dar un margen de seguridad para el correcto funcionamiento.

Cuando el sensor esté midiendo tras haber recibido el disparo RX por PW se emitirá un pulso (el sensor habilita el pin) un tiempo determinado equivalente a 147µS por pulgada.

Por ejemplo, si medimos una distancia de 3 metros.

$$3m \rightarrow \frac{300\ cm}{2.14\frac{cm}{inch}} = 140.2\ inch$$

$$147\frac{uS}{inch} \cdot 140.2\ inch = 20.6\ mS$$

Este tiempo se deberá cuantificar.

Por el pin PW recibiré el ancho de pulso que habilitará un contador de una frecuencia superior (17.28 KHz), este empezará a contar y al acabar el ancho de pulso el sensor lo deshabilitará. El contador parado mantiene a su salida un número, cuyo valor ya está en centímetros (se explica con más detalle en la Etapa 6 del sub apartado 3.1.3 Desarrollo del diseño 1,).

El pin RX, que dispara el contador, se activará de forma periódica para tomar una medida en cada uno de estos ciclos

Como podemos ver en la figura 20, tendremos que tener en cuenta los tiempos que debe mantenerse a 1 el pin 4 (Rx) para que se realice adecuadamente la medición. Este deberá estar habilitado el tiempo que dure la medicción

Ademas, como se puede observar poco después de habilitarlo el sensor comienza a medir.

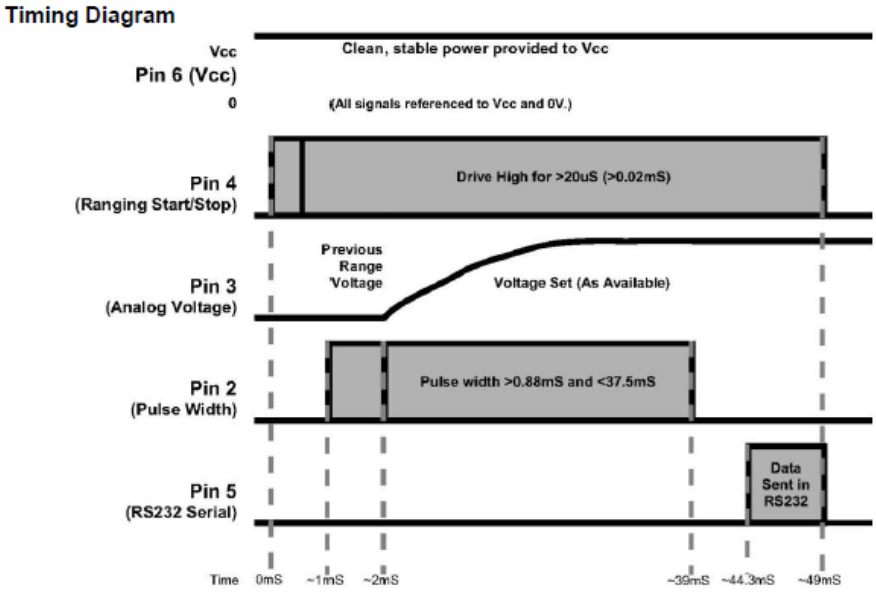


Figura 20 Señales del Sensor de Ultrasonidos

2.6 Introducción a los microcontroladores

Un microcontrolador es un circuito integrado programable, capaz de ejecutar órdenes grabadas en su memoria. En su interior incluye las tres principales unidades funcionales de una computadora: unidad central de procesamiento (microprocesador), memoria y periféricos de entrada/salida.

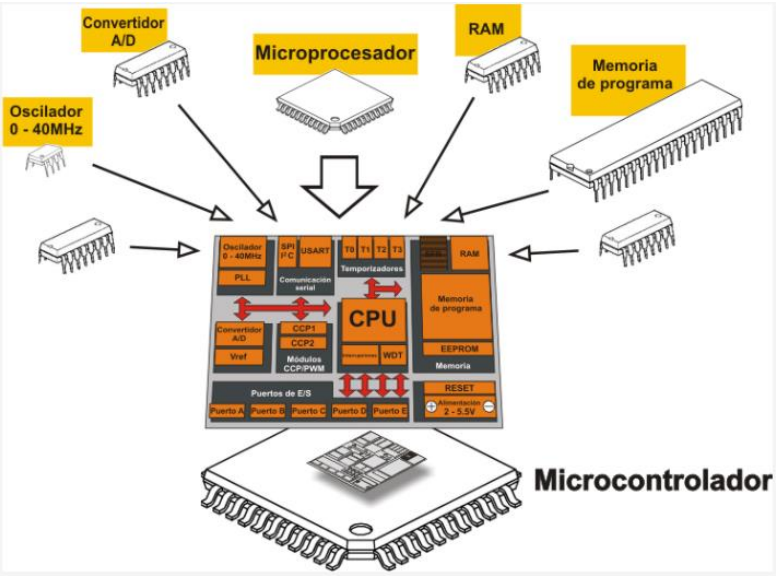


Figura 21 Elementos de un microcontrolador

Los elementos principales de un microcontrolador son:

- Procesador o UCP (Unidad Central de Proceso).
- Oscilador y generador de señal de reloj para la sincronización de todo el microcontrolador.
- Memoria RAM para contener las variables y datos.
- Memoria tipo ROM/PROM/EPROM donde se aloja el programa.
- Puertos de E/S de tipo GPIO.
- CAD: Conversores Analógico/Digital.
- CDA: Conversores Digital/Analógico, etc.).

Actualmente existe una gran variedad de microcontroladores disponibles para su integración en productos. Su programación puede llevarse a cabo mediante distintos lenguajes de programación tales como Assembly, C y C ++, u otras propias del fabricante del dispositivo.

La clasificación más importante que podemos hacer se basa en el número de bits (4, 8, 16 ó 32). Si bien los microcontroladores de 16 y 32 bits ofrecen prestaciones mayores a los de 4 y 8 bits, la realidad es que éstos últimos son válidos para la mayoría de aplicaciones y hoy en día dominan el mercado.

3 Desarrollo del TFG

Este apartado recoge los distintos diseños que se han realizado con el fin de implementar un microcontrolador en la FPGA, se observará que se realizan modificaciones progresivamente y se parte de los diseños anteriores para evolucionar.

La estructura de los apartados varía en función de las necesidades de cada uno.

3.1 Diseño 1: Implantación de la medida de distancia.

Implantación de un medidor de distancia empleando la placa de desarrollo "breakout board" que contiene la FPGA MACHX02-7000HE y el sensor de Ultrasonidos LV-MaxSonarEZ3.

La idea de este diseño es poder medir distancias, en centímetros, enfocando el dispositivo al objeto que se desee, y visualizando la distancia en los displays. Con los interruptores podremos detener las mediciones.

Se ha tomado este diseño porque contiene varios elementos que permitirán demostrar la utilidad de los dispositivos FPGA. Se implementarán elementos "sencillos" (en comparación con un microcontrolador), como pueden ser los, contadores, decodificadores... Y también se integrarán algunos más complejos como el bloque PLL (se explicará más adelante) o un bloque generado en VHDL llamado "control_disp" (también se explicará en este diseño).

3.1.1 Objetivos del diseño

Implantación de un medidor de distancia en la FPGA.

Descripción de elementos mediante código en VHDL, y generación de símbolos para incorporarlos a un esquema.

Trabajar con esquemas en Lattice Diamond. Permitirá crear proyectos grandes haciendo uso de símbolos y uniéndolos mediante líneas, como si se tratase de esquemas electrónicos.

Utilizar adecuadamente Lattice Diamond. Gestionar proyectos y los archivos que contienen, así como configurar las patillas de entradas y salidas de la FPGA, también a configurar el dispositivo.

3.1.2 Material utilizado

El material utilizado en este diseño es el siguiente:

Software:

Lattice Diamond 3.8.

Hardware:

FPGA MACH2OX-1200ZE junto con su breakout board.

Placa de extensión con pulsadores y SWITCHs.

Placa de extensión con dos displays.

Sensor de Ultrasonidos LV-MaxSonarEZ3.

3.1.3 Desarrollo.

El desarrollo, se ha dividido en las etapas (o partes) que se muestran a continuación:

3.1.3.1 Etapa 1

Lo primero se realiza la descripción VHDL de un decodificador BCD-7segmentos que se utilizará posteriormente. Se genera un símbolo que se empleará, posteriormente, en el proyecto global.
Este transformará las señales del bus de entrada procedente de contadores BCD de 4 bits (que está en binario) a otro destinado a salir a los visualizadores de7 segmentos que tienen 7 bits.

Su función será mostrar en los distintos displays el dígito numérico que proporcionan los contadores BCD en binario.

El visualizador de 7 segmentos consta de los siguientes diodos, ordenados de la siguiente forma:

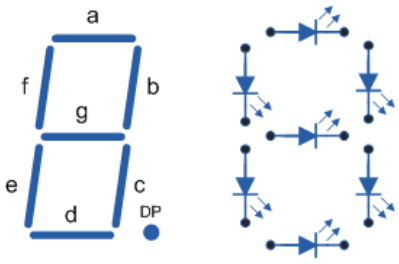


Figura 22 Configuración LEDs del visualizador de 7 segmentos

Para que los LEDs se iluminen formando un dígito en decimal dependiendo del número en binario que reciba como entrada, se debe implementar un

decodificador, que será un bloque en el esquema de Lattice Diamond como el siguiente:

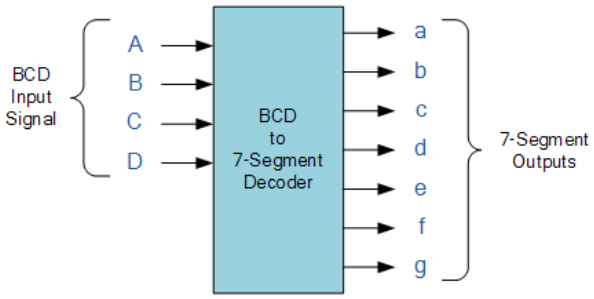


Figura 23 Decodificador BCD

La señal de entrada estará en binario y la salida en un código de 7 segmentos, tal y como se representa en la siguiente tabla:

Entradas BCD				Saídas Segmentos							display
D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9

Figura 24 Tabla Decodificador BCD

Dependiendo si es ánodo o cátodo común ceros y unos estarán invertidos, o aparecerán como en la tabla de la figura 24.

La descripción se podría haber hecho de manera secuencial o combinacional, haciendo uso o no de la función process en VHDL (son dos maneras de describir un mismo elemento igualmente válidas).

Se crea un nuevo proyecto en el que se realiza la descripción VHDL de un convertidor BCD-7segmentos. Se generará un símbolo que se empleará, posteriormente, en el proyecto global. Se salva todo y se cierra el proyecto.

El código VHDL que describe el comportamiento del bloque es el siguiente:

```
1 LIBRARY IEEE;
```

```
2 USE IEEE.std_logic_1164.all;
3 USE IEEE.numeric_std.all;
4 entity DEC_DISP is
5 PORT( BCD : in std_logic_vector(3 downto 0);
6 SEGMENTOS : out std_logic_vector(6 downto 0));
7 end DEC_DISP;
8 architecture DEC_DISP_arch of DEC_DISP is
9 BEGIN
10
11
12 WITH BCD SELECT
13
14 SEGMENTOS <=
15 "0111111" when "0000", --0
16 "0000110" when "0001", --1
17 "1011011" when "0010", --2
18 "1001111" when "0011", --3
19 "1100110" when "0100", --4
20 "1101101" when "0101", --5
21 "1111101" when "0110", --6
22 "0000111" when "0111", --7
23 "1111111" when "1000", --8
24 "1101111" when "1001", --9
25 "1111111" when others; --todo
26 end DEC_DISP_arch;
```

Para generar el símbolo hay que crear un proyecto nuevo. Todos los proyectos que se vayan a utilizar en un diseño deberán guardarse dentro de una carpeta concreta y arbitraria. Es aconsejable guardar todo en una carpeta cercana a la raíz del disco duro y con nombres cortos (todo, tanto proyectos, como archivos de esquemas, archivos VHDL...).

En mi caso trabajaré en la carpeta:

C:\TFG\exp1

En ella se generan todos los proyectos, además, es aconsejable llamar con el mismo nombre a un proyecto, junto con su archivo principal (si es un proyecto donde elaboramos un esquema, al esquema se le llamará con el nombre del proyecto, si es un proyecto donde generemos un archivo VHDL para crear un símbolo, al archivo VHDL se le llamará con el nombre del proyecto, etc) así como a la carpeta de implementación propia del proyecto.
Esto permitirá gestionar fácilmente los archivos, y cuando en un proyecto queramos incluir elementos de otros, será fácil localizar archivos y recordar sus nombres.

Se crea el proyecto y se añade el archivo VHDL.

Carpeta donde se aloja proyecto	
Archivos y carpeta de impl.:	C:\TFG\exp1
El proyecto será:	DEC_DISP.lpf
El archivo VHDL será:	DEC_DISP.vhd
La carpeta de implementación será:	C:\TFG\exp1\DEC_DISP

Una vez creado el código VHDL, se guarda, y se hace clic en la ventana de Process al "Boton" de Synplify Pro, que es la herramienta de síntesis utilizada en mis diseños por Lattice Diamond.
El chequeo de la sintaxis se realiza al salvar, y el chequeo del funcionamiento al simular.

Se observa que no hay fallos.

En este momento en la ventana Hierarchy aparecerá una línea con el nombre del archivo VHDL que he creado.

Se realizan las siguientes operaciones.

Botón derecho con el Ratón y Set as top Level Unit

Botón derecho con el Ratón Generate Schematic Symbol.

Se crea un nuevo proyecto para integrar el símbolo y chequear su funcionamiento.

Cuando se logra el objetivo prosigo con el diseño. En este nuevo proyecto se añade un archivo ".sch" se importa el símbolo creado anteriormente y el archivo ".vhd" generado.

En mi diseño el símbolo tendrá un aspecto como el siguiente.

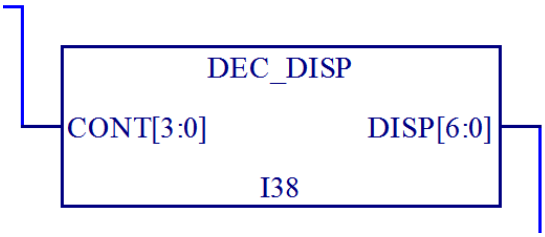


Figura 25 Simbolo Decodificador BCD

Se dibuja el circuito que se muestra a continuación en la ventana "schematics" y se configura la FPGA para observar su comportamiento.

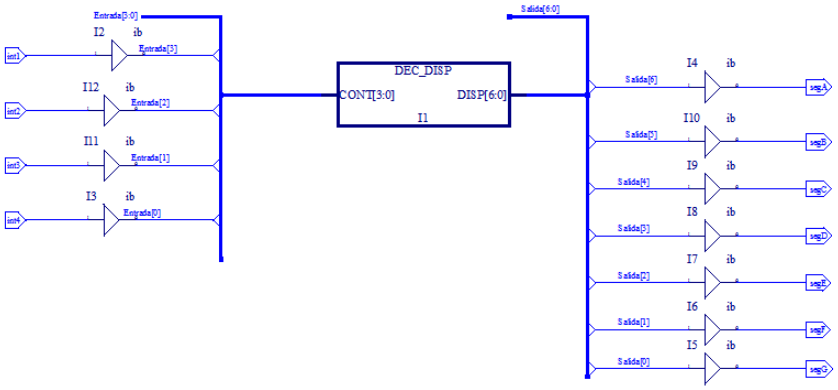


Figura 26 Esquema con Decodificador BCD

3.1.3.2 Etapa 2

Se genera un nuevo proyecto y, dentro de este, un nuevo esquema en el que se irá realizando, en pasos sucesivos, el sistema completo.

Carpeta donde se aloja proyecto	
Archivos y carpeta de impl.:	C:\TFG\exp1
El proyecto será:	proyecto_global.lpf
El archivo esquema será:	esquema_global.sch
La carpeta de implementación será:	C:\TFG\exp1\proyecto_global

En primer lugar, se construirá un circuito que genere una frecuencia de, aproximadamente, 1 Hz. Para ello se empleará el oscilador interno del dispositivo (frecuencia por defecto de 2.08 MHz.)
El valor exacto no es importante, ya que su utilidad es establecer el ritmo en el que se refrescan los visualizadores de 7 segmentos, que muestran la distancia.

Un contador, del tamaño adecuado, que se generará con la utilidad IPexpress y que se usará como divisor de frecuencia.

Para generar los bloques con IPexpress se deberá usar la herramienta y seguir los menús básicos que aparecen en ella, y para añadir los símbolos al esquema habrá que realizar las siguientes dos operaciones:

-Sobre el esquema, botón derecho → Add → Symbol → Buscamos el símbolo en la carpeta local.

-En la ventana files será necesario añadir el código VHDL generado por la herramienta. Para ello, clic en File → Add →Existing File → y buscamos en la carpeta donde tenemos el proyecto el nombre del símbolo generado.

El esquema generado en este paso se muestra en la figura 27. Como vemos el simbolo "osch" es el oscilador interno del dispositivo, que tendrá una señal de entrada que nos permitirá habilitar y deshabilitar el oscilador.

El segundo bloque llamado divfrec1 es un contador que actúa como un divisor de frecuencia. Este tiene 21 bits, y como queremos que pase de 2.08MHz a 1Hz tendrá 2.080.000 estados.
Se genera con la herramienta IPexpress, que permite describir bloques de elementos comunes (como es el caso de un contador), con un menú de configuración, y evita generar el código VHDL a mano, ahorrando mucho trabajo.

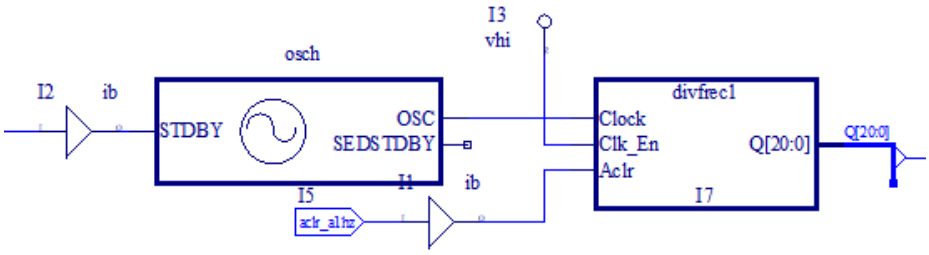


Figura 27 Esquema para señal de 1Hz

Nota:
Para añadir un símbolo generado por IPExpress, se realiza el procedimiento realizado anteriormente en esta misma etapa:

-Sobre el esquema, botón derecho → Add → Symbol → Buscamos el símbolo en la carpeta local.

-En la ventana files será necesario añadir el código VHDL generado por la herramienta para ello, clic en File → Add →Existing File → y buscamos en la carpeta donde tenemos el proyecto el nombre del símbolo generado.

Las señales de habilitación del oscilador, habilitación del contador y reset del contador se gobernarán desde los interruptores de la placa.
La señal de 1 Hz se visualizará en uno de los LEDs de la placa, concretamente, en el LED D1.

Una vez realizada la descripción (esquema), se salva, se genera la jerarquía y se realiza el chequeo.

Se genera el fichero de configuración “jedec”, se configura el dispositivo y se comprueba que el funcionamiento es el esperado.

Podemos observar que de su bus de salida solo nos interesa el bit 20, ya que es el que dará un pulso de ‘1’ cada segundo estando el resto de tiempo a ‘0’.

3.1.3.3 Etapa 3

En este paso se va a realizar un contador BCD de un dígito, al que se le aplica la señal de 1 Hz del paso anterior, esta frecuencia es independiente del contador y posteriormente se cambiará. Se parte del esquema generado en el paso 2. La señal que se tenía, de 1Hz, no solo va a gobernar el LED, sino que, además, se va a aplicar a un contador BCD de un dígito que se debe generar con la utilidad IPexpress. Las salidas del contador se llevarán a cuatro de los LEDs de la placa. Una vez generadas jerarquías y chequeado el diseño, se generará el fichero “jedec”, se configurará el dispositivo y se comprobará su funcionamiento.

Siguiendo los pasos realizados en el apartado anterior con IPexpress generamos un bloque llamado cont_BCD, que será un contador de 4 bits que contara de 0 a 9 y que cuando llegue al estado del 9, que es 1001(en binario) se resetea volviendo al 0000.

El símbolo generado para el contador BCD es el siguiente:

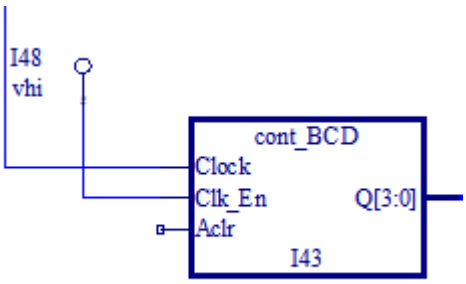


Figura 28 Símbolo contador BCD

Una vez generado el símbolo se realizan las conexiones oportunas, como se muestra en la figura 29:

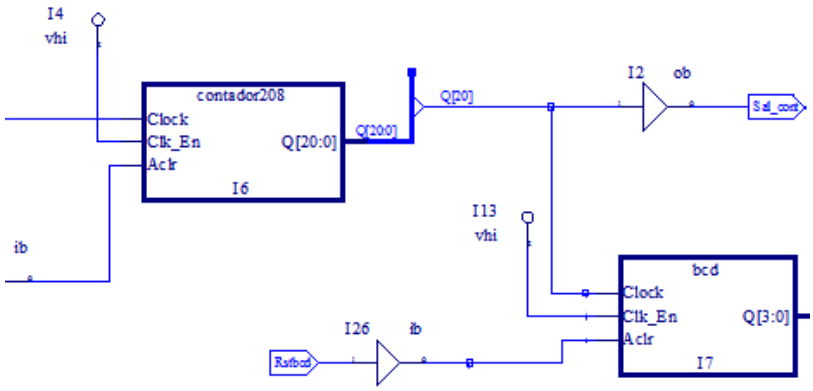


Figura 29 Alimentacion del contador BCD

3.1.3.4 Etapa 4

En este paso, se realizará la visualización del estado del contador en el display de la derecha.
Para ello, se incorporará al esquema el convertidor BCD-7segmentos realizado en el paso 1. La salida del contador BCD se llevará al convertidor BCD-7segmentos y la salida de éste a los segmentos adecuados del display.
Una vez generada la jerarquía, realizado chequeos y generado el fichero “jedec”, se configura el dispositivo y se comprueba su funcionamiento.

En este apartado manteniendo el resto de partes del esquema intactos realizamos las siguientes conexiones:

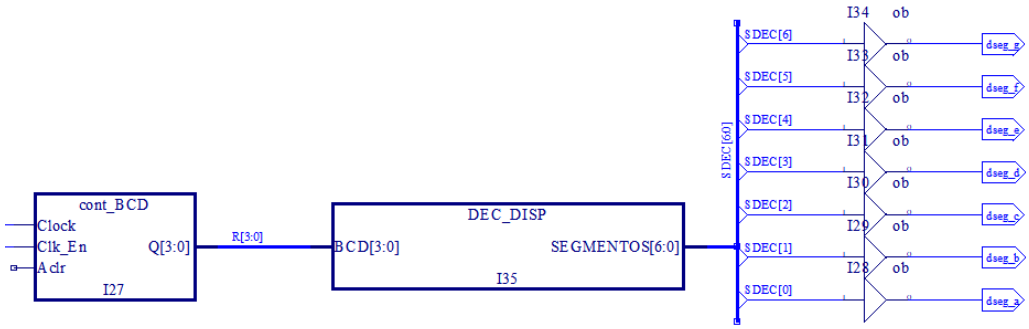


Figura 30 Unión Contador-Decodificador-Salidas

3.1.3.5 Etapa 5

En este paso, se debe modificar el diseño del paso anterior, de forma que el contador BCD sea, ahora, de 3 dígitos y, en consecuencia, la visualización se deberá realizar en los dos visualizadores de 7 segmentos y en los LEDs de la placa breakout board.

En los displays aparecerá los números en decimal y en la placa se dejarán en binario. Esto es debido a las limitaciones de las placas de expansión que solo poseen dos visualizadores de 7 segmentos.

- LEDs placa → Unidades
- Visualizador de 7 segmentos derecha → Decenas.
- Visualizador de 7 segmentos izquierda → Centenas

La circuitería necesaria para conectar los contadores en cascada implica incluir unas puertas AND y unos inversores, para que cuando el BCD de la derecha llegue a 9 habilite el reloj, para que el siguiente contador BCD cambie de estado (decenas), y otras dos para que cuando el estado sea 99, se habilite el reloj del contador de las centenas.

Queda reflejado en el siguiente esquema:

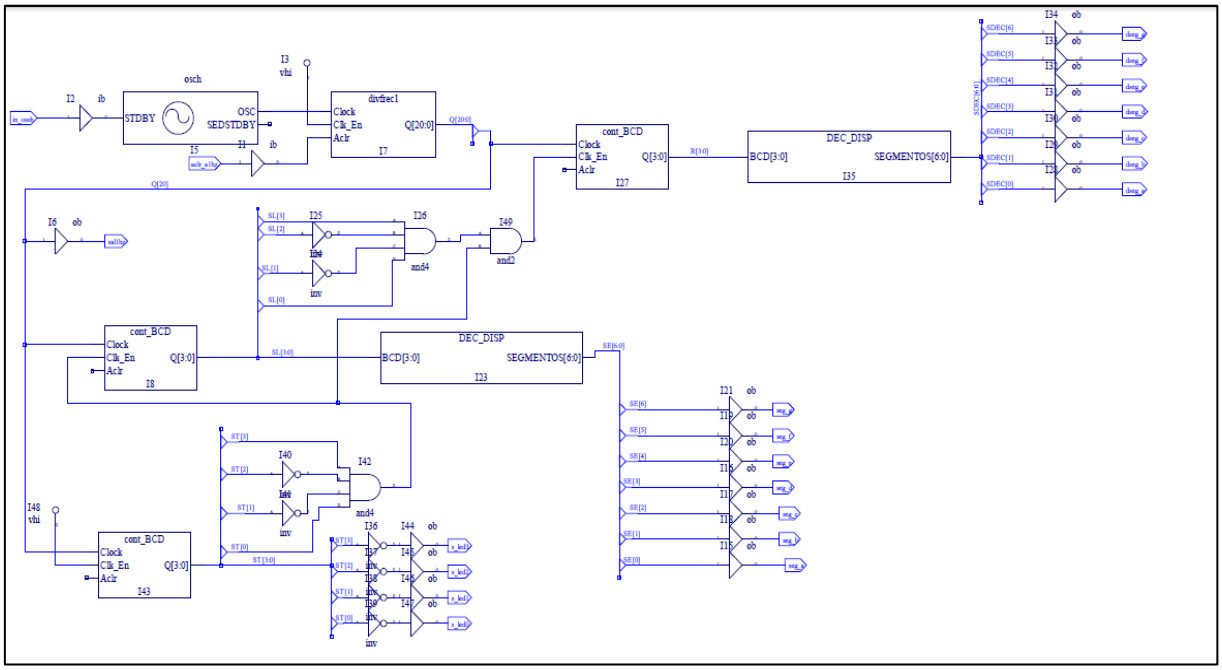


Figura 31 Esquema Contador en cascada diseño 1

Se comprueba el funcionamiento generando el archivo de configuración "Jedec" (".jed") y configurando en la FPGA.

En este paso vemos en el dispositivo varias acciones:

El dispositivo cuenta de 1 a 999 en intervalos de un segundo, mostrando las centenas y decenas por los displays, y mostrando las unidades en binario en los LEDs de la derecha.

El LED D8, el de abajo del todo, parpadea a la frecuencia de 1 Hz (periodo de 1 segundo).

Imagen del dispositivo en funcionamiento:

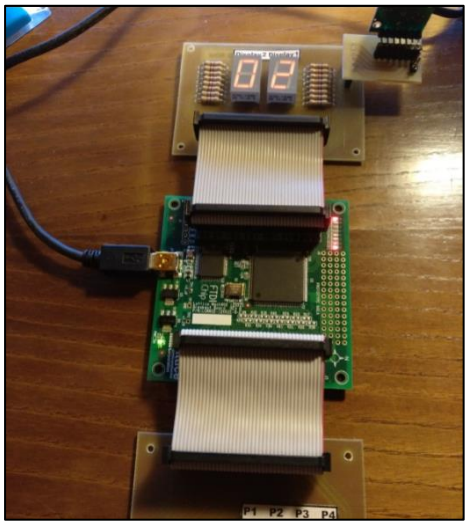


Figura 32 Funcionamiento Contador cascada Diseño 1

3.1.3.6 Etapa 6

Generar una frecuencia que permita al contador visualizar los resultados en centímetros.

Es decir, el sensor dará un ancho de pulso determinado, lo que es lo mismo, la salida del sensor PW estará un tiempo habilitada (a '1').

Se tiene un contador que cuenta de 1 a 999, que dependiendo de la frecuencia que se alimente su señal de reloj, contara más rápido o más despacio, mientras esté habilitado.

Para facilitar el procesamiento, se alimenta al contador con una frecuencia (CLK), de forma que la frecuencia de esta señal hará que el contador muestre la distancia en centímetros, cuando el sensor habilite la señal del contador(En).

El pulso emitido por el sensor es de $147\,\frac{\mu s}{inch}$.

Con esto se haya una frecuencia que en Hz por cada cm.

De tal manera:

$$147\,\frac{\mu s}{inch}\cdot\frac{1\;inch}{2.54\;cm}=57.874\;\frac{\mu s}{cm}$$

$$57.874\ \frac{\mu s}{cm} = 17278'9\frac{Hz}{cm}$$

$$17278'9\frac{Hz}{cm} = 17.28\ KHz/cm$$

Luego si se introduce esta frecuencia en el reloj del contador, dará un valor en centímetros en su salida.

Para generar esta frecuencia se debe usar el reloj externo de la FPGA, ya que el interno es poco preciso y tiene 2,08Mz. Crear un contador para que queden 17.28 KHz es complicado y puede resultar poco exacto.

Por lo tanto, se utiliza el reloj externo soldado a la breakout board de 50 MHz cuya señal de entrada es la patilla 32.

Se utiliza un bloque llamado PLL, que se genera desde la herramienta IPExpress, cuyo objetivo es crear un bloque donde puedo elegir las frecuencias de entrada y de salida dentro de unos límites y con unas tolerancias.

Este bloque es un componente que ya posee la FPGA, esta embebido físicamente en el chip y se accede a él mediante IPExpress

Las unidades de este bloque no aparecen en la ventana de dialogo, en la documentación encontraremos que son MegaHercios.

Se configurará como aparece en la figura 33, donde usaremos un reloj que no es la frecuencia principal ya que la primera salida se utiliza para frecuencias grandes 10MHz, por ejemplo.

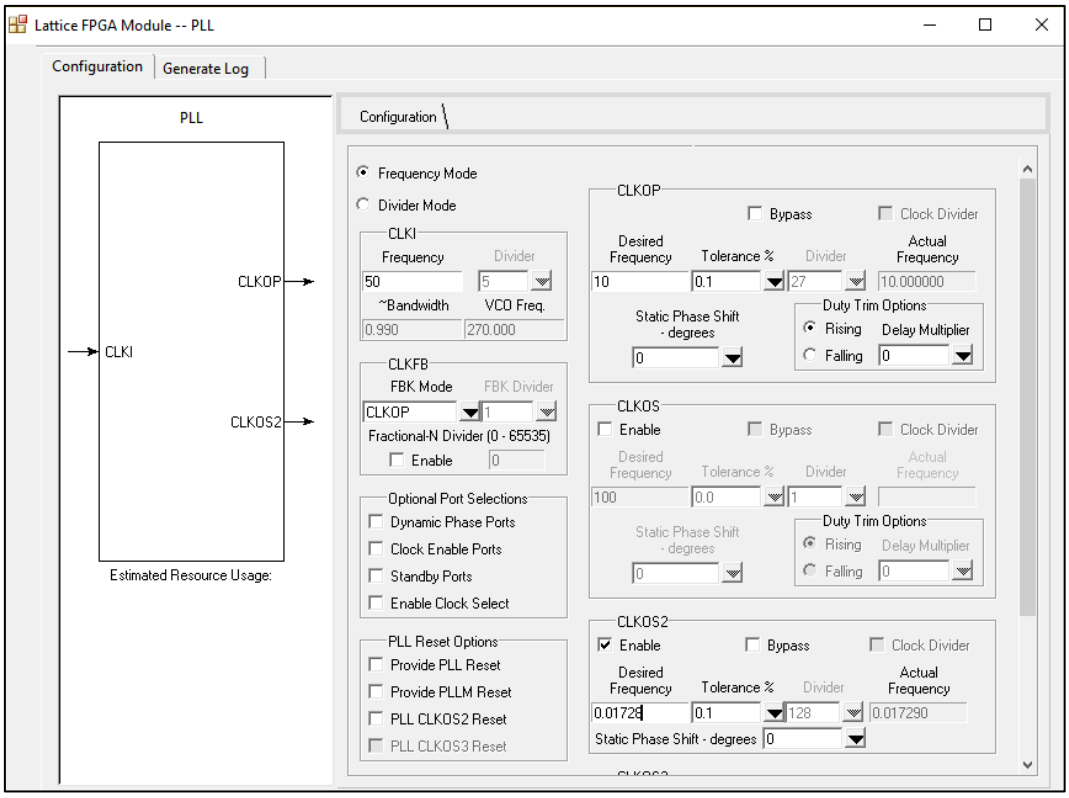


Figura 33 Configuración bloque PLL

Se habilita la tercera salida con el check box, llamada CLKOS2 (a medida que se haga check en "CLKOSs" con más numeración, se pueden calcular frecuencias más pequeñas), y se configura con una frecuencia de 0.01728 (17.28 KHz) que es la que se necesita.

La herramienta tiene un "botón" llamado "Calcular" que nos indica si la frecuencia elegida es válida. Al hacer clic en calcular, nos indica cual es la frecuencia que utilizará, de 0.01729 MHz. Esto es debido a que el bloque requiere unas tolerancias de cálculo.

3.1.3.7 Etapa 7

Generar el pulso de reset del contador y el pulso de lanzamiento de medición del sensor ultrasonidos EZ3. Esta señal se generará a partir de las salidas del contador, que se emplea para generar el reloj de 1 Hz.

La señal de reset será un pulso, de 2 µs de ancho, que se repetirá periódicamente cada 1 segundo, es por lo tanto que se usará el reloj de 1 Hz.

La señal para lanzar las "mediciones" (Rx del sensor) será un pulso de 32 µs, ya que como mínimo había que introducir una señal de 20 µs, aplicando un

margen de seguridad de 1.5, nos queda 30 µs y como funcionamos con números en binario se aproxima a 32 µs.

Ambas señales deben pasar de 0 a 1 simultáneamente.

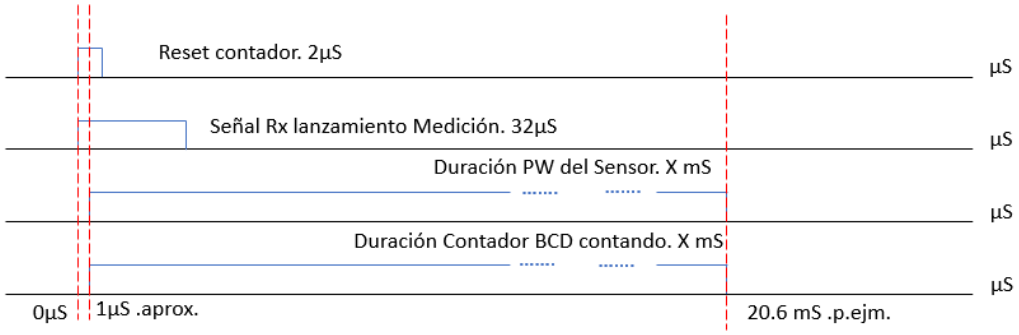


Figura 34 Dibujo señales, durante el funcionamiento

Para generar las señales de 2µs y 32µs se ha creado el siguiente esquema:

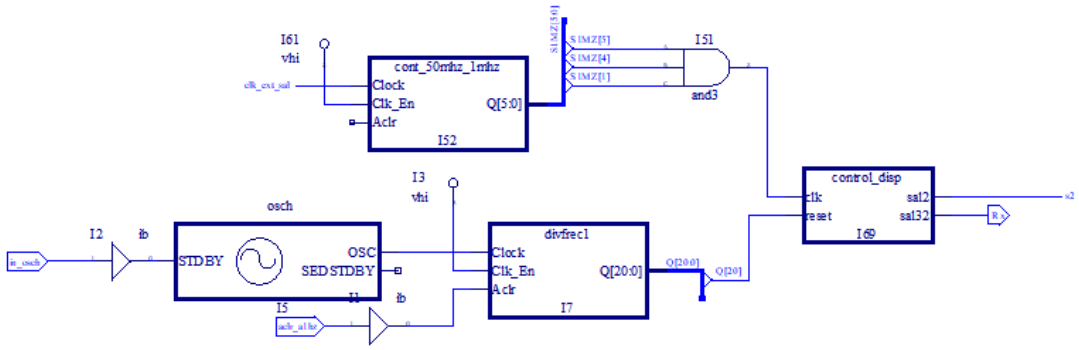


Figura 35 Esquema para señales s2 y s32

En él se incluyen 3 elementos:

El conjunto “osch” y divisor de frecuencia a 1 Hz que se utilizó en la Etapa 2, para que el contador de 0 a 999 contase. Ahora se utilizará para resetear las señales “Rx” y “s2” (lanzar medidas y refrescar contador de displays) a cero cada segundo.

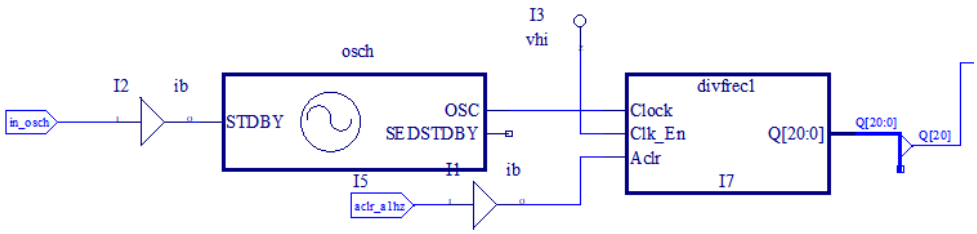


Figura 36 Esquema para señal 1Hz

Un bloque llamado “cont_50mhz_1mhz” cuya función junto con la puerta lógica AND se encargan de dar una frecuencia de 1 MHz, ya que esta es mucho más manejable a la hora de manipular periodos de un µs. Se trata de un contador de 1 a 50 alimentado por el oscilador externo que es de 50 MHz. Este bloque ha sido generado por la herramienta IPExpress y se le ha añadido la puerta AND para los bits ‘5’, ‘4’ y ‘1’ sin negar y los bits ‘0’ ‘2’ y ‘3’ negados ya que cuando estos estén a ‘1’ y ‘0’ respectivamente, marcará el número 50 (en binario).

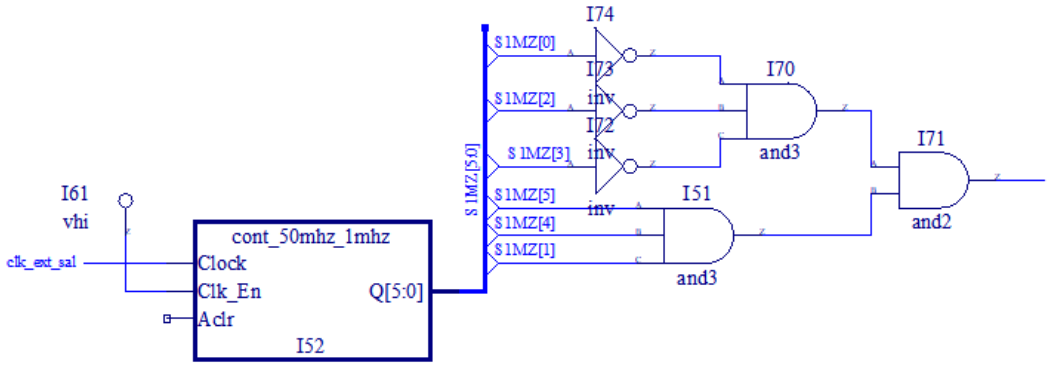


Figura 37 Esquema con contador para obtener 1Mz de 50Mz

50 → 110010

Y un tercer elemento que he llamado “control_disp”, quien a partir del reloj de 1 MHz y del reset cada segundo da las dos señales deseadas, este ha sido programado en código VHDL, ya que mediante esquemas es difícil de implementar.

Para ello se crea un proyecto nuevo, se añade un archivo de esquema nuevo y se programa en él, posteriormente se genera el símbolo y dentro del proyecto global se incorpora tanto el símbolo como el código VHDL.

Los directorios nuevos son:

Carpeta donde se aloja proyecto	
Archivos y carpeta de impl.:	C:\TFG\exp1
El proyecto será:	control_disp.lpf
El archivo VHDL será:	control_disp.vhd
La carpeta de implementación será:	C:\TFG\exp1\cotrol_disp

Como se puede observar se sigue manteniendo las reglas de organización anteriormente establecidas.

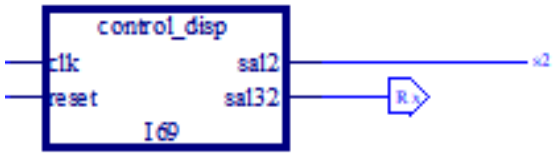


Figura 38 Bloque que genera s32 y s2

El código VHDL del bloque contiene un contador con un millón de estados, con el objetivo de que jamás se alcance este valor, ya que antes de que se alcance se deberá hacer un reset. El reset es de tipo asíncrono, lo que quiere decir que en cualquier momento que la señal reset tome valor 1, el contador independientemente de cuál sea su valor de cuenta, comenzará de nuevo empezando por cero.

Las salidas “sal2” y “sal32” solamente estarán con valor 1 mientras los valores de cuenta estén entre 1 y 2 ó 1 y 32.

De tal manera, que conjuntamente conseguimos estas 2 señales. El código escrito en VHDL para este bloque, es el siguiente:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity control_disp is
6 PORT (
7 clk : IN STD_LOGIC;
8 reset : IN STD_LOGIC;
9 sal2 : Out STD_LOGIC;
10 sal32 : Out STD_LOGIC
11 );
12 end control_disp;
13
14 architecture control_arch of control_disp is
15 signal cnt : unsigned(20 downto 0);
16
17 begin
18
19 -- Proceso del contador, de 0 a 1 000 000.
20 contador: process (reset, clk) begin
21 if (reset = '1') then
22 cnt <= (others => '0');
23 elsif rising_edge(clk) then
24 if (cnt = 1000000) then --En decimal
25 cnt <= (others => '0');
26 else
27 cnt <= cnt + 1;
28 end if;
29 end if;
30 end process;
31
32 sal2 <= '1' when ((cnt > 0) AND (cnt < 3 )) else '0';
33 sal32 <= '1' when ((cnt > 0) AND (cnt < 33)) else '0';
34
```

```
35 end control_arch;
```

Cuya arquitectura puede definirse también por :

```
14 architecture control_arch of control_disp is
15 -- Contador de 0 a 1279.
16 signal cnt : unsigned(20 downto 0):="00000000000000000000";
17
18 begin
19
20 -- Proceso del contador, de 0 a 1 000 000.
21 contador: process (reset, clk) begin
22 if (reset = '1') then
23 cnt <="00000000000000000000";
24 elsif (clk'event and clk='1') then
25 if (cnt = "11111111111111111111") then --En decimal
26 cnt <= "00000000000000000000";
27 else
28 cnt <= cnt + 1;
29 end if;
30 end if;
31 end process;
32
33 sal2 <= '1' when ((cnt > "00000000000000000000") AND (cnt <
"00000000000000000011" )) else '0';
34 sal32 <= '1' when ((cnt > "00000000000000000000") AND (cnt <
"00000000000000100001")) else '0';
35
36 end control_arch;
```

Una vez generado el bloque e incluido en el diseño, siguiendo el procedimiento explicado en la Etapa 1, se realizan las conexiones oportunas en el esquema del proyecto:

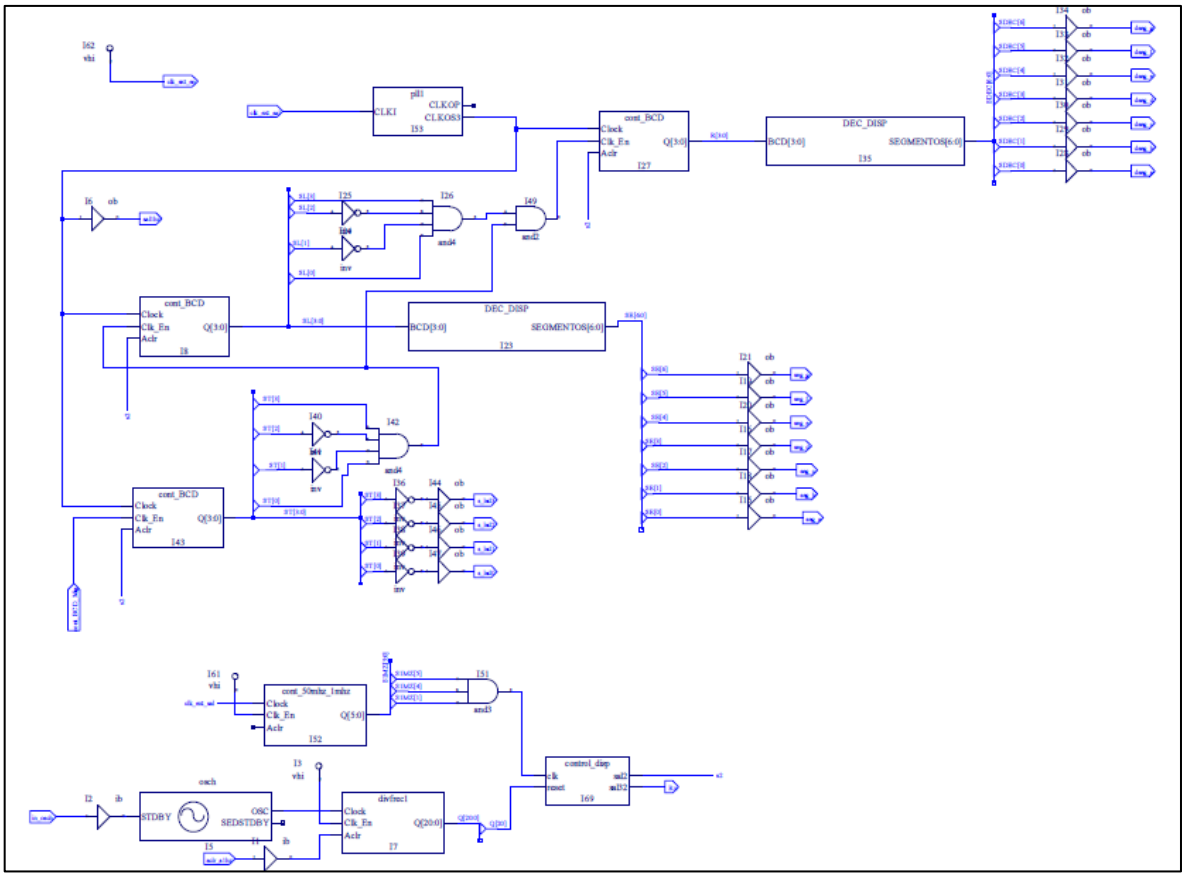


Figura 39 Esquema final del diseño 1

3.1.3.8 Etapa 8

Finalmente, se ejecuta la herramienta de síntesis (Synplify Pro), se eligen y escriben las patillas de la FPGA a utilizar para las entradas y salidas, y se genera el archivo Jedec.

Se configura la FPGA MACHXO2 1200ZE y se comprueba el funcionamiento del dispositivo.

El resultado es satisfactorio, se puede ver como el dispositivo muestra en centímetros, los valores de las distancias medidas por el sensor ultrasonidos, apuntando con él a distintos cuerpos y analizando los resultados.

A continuación, se incluyen fotografías del dispositivo funcionado:

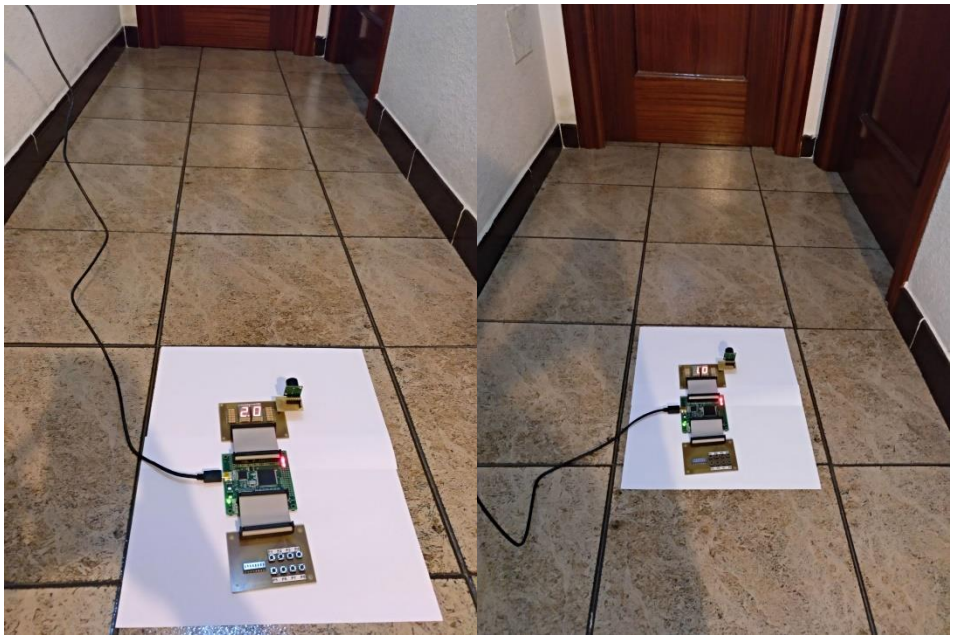


Figura 40 Funcionamiento del diseño 1

Se puede observar que las mediciones son bastante acertadas y que los resultados son satisfactorios.

Nota: *Para 3 azulejos la medida es de aproximadamente un metro y para 6 azulejos la medida es de aproximadamente 2 metros.*

Hay que recordar que tiene una distancia focal de 15 centímetros, en la que por delante de ella el dispositivo da error.

3.1.4 Simulación bloque “control_disp”.

Se realiza la simulación del elemento “control_disp”, con el objetivo de verificar el correcto funcionamiento del mismo, ya que es un bloque que no se puede apreciar a “simple vista” su comportamiento.

Las simulaciones se realizan sobre el proyecto en el que se creó el elemento, no en los proyectos globales.

Se abre el proyecto de Lattice Diamond “control_disp.ldf”.

El software Lattice Diamond posee una herramienta para simular código VHDL, estas simulaciones no reflejan el comportamiento real en el sistema físico, pero si el comportamiento teórico, donde se podrá observar si el código es correcto o no.

Para acceder al simulador, Tools → Simulator Wizard.

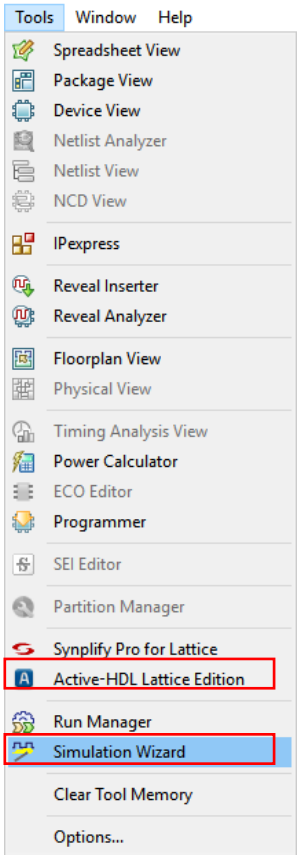


Figura 41 Simulación en Lattice Diamond

Una vez en el simulador, se añaden las señales de la arquitectura del código VHDL que se quieran mostrar y se configuran las entradas.

Como entradas al código de simulación hay que definir las variables “clk” y “reset”, para ello se usa la ventana de configuración: Stimulators.

Se realiza la siguiente configuración:

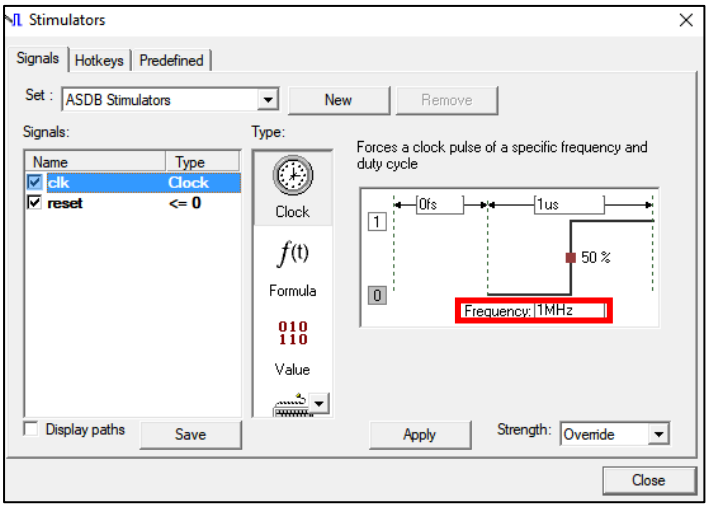


Figura 42 Estímulos para señales del simulador

En ella se elige la frecuencia del reloj que es la misma que alimentará el bloque en el esquema del diseño.

Cuando se quiera cambiar de valor un estímulo hay que volver a entrar en esta ventana, y modificarlo, como es el caso del “reset”, cambiarlo a ’1’ y ejecutar el simulador más tiempo.

En el estado inicial se puede observar que el valor “cnt” de la señal cuenta está inicializado a cero, como se ve en la figura 43.

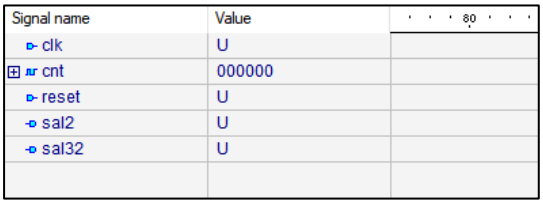


Figura 43 Estado inicia simulaciónl

Se seleccionan las unidades del simulador como milisegundos.

Se ejecuta el simulador 100 ms y se obtiene el siguiente resultado:

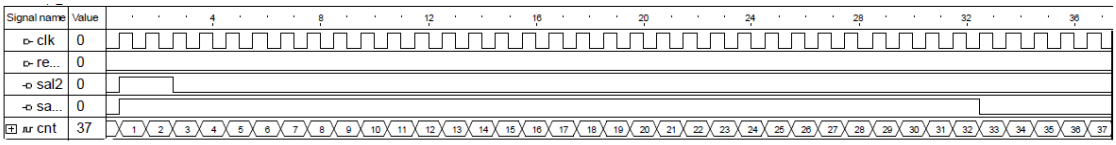


Figura 44 Cronograma: Primera simulación

Se puede analizar en el cronograma, que el comportamiento es el adecuado. Cuando el reset está a cero, se producen los pulsos deseados para “sal2” y “sal32”, se puede ver en la Figura 44 que en cuanto a tiempos y al valor del contador concuerda.

Al hacer un reset asíncrono (habilitar señal “reset”), el contador deja de contar y pasa a valer 0 hasta que deja de estar habilitado el reset.

Cuando se deshabilita el “reset”, vuelve a empezar a contar desde el principio.

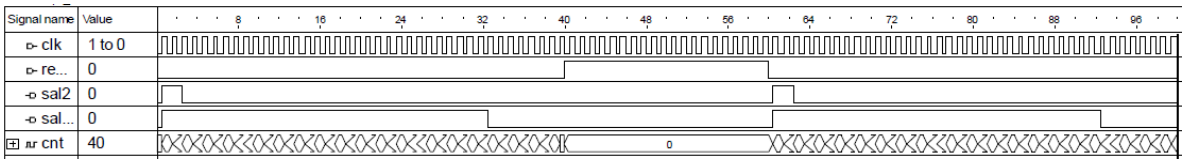


Figura 45 Cronograma: Segunda Simulación

Se concluye que el código VHDL del bloque “control_disp” funciona adecuadamente.

3.2 Diseño 2: Implantación de un Microcontrolador.

En este diseño se tratará de explicar cómo implementar un microcontrolador en la FPGA y vincularlo a un símbolo, para luego usarlo en los esquemas junto a más lógica, como se realizó, por ejemplo, en el diseño anterior.

Se mostrará cómo realizar diseños de microcontroladores, ya que estos tendrán los elementos que elijamos, como GPIOs o UART, memoria del tamaño que queramos (siempre dentro de lo que pueda caber en el dispositivo FPGA que se utilice).

Haciendo el Microcontrolador a medida, se pueden aprovechar recursos y espacio de la FPGA, para que el restante, pueda ser aprovechado por la lógica adicional que queramos hacer funcionar en conjunto con el microcontrolador.

Además, se explicará cómo escribir el código que queremos ejecutar en el micro y cargar el programa en lenguaje C al microcontrolador.

El método que a continuación se muestra para implementar el microcontrolador no aparece en ninguna referencia documental ni en sitios web

3.2.1 Objetivo del diseño.

Se va a realizar un diseño que contendrá un esquema ".sch" dentro de un proyecto de Lattice Diamond, con el símbolo del microprocesador (sin oscilador interno) y la lógica necesaria para unir este, con los pines de entrada y salida del Microcontrolador a los del dispositivo FPGA.

El funcionamiento del microcontrolador será, leer un bus de 8 pines de entrada y proporcionar en la salida de este, lo mismo que se ha leído, en el mismo orden.

Como el objetivo es la implantación de un microcontrolador, no del desarrollo de software (los recursos empleados, dependen de la cantidad de memoria asignada, no de su contenido), este diseño se implantará con un software extremadamente sencillo, cuya función es únicamente leer un bus de entrada y copiar esa información en el bus de salida.

Para ello se conecta el bus de pines de entrada a 8 pulsadores (que se encuentran en la placa de expansión de la breakout board) de tal manera que cuando se presione cualquiera de ellos se enviará un '1 'al microcontrolador.

Se conecta el bus de salida a 8 LEDs con sus respectivas resistencias, embebidas en la breakout board.

Cuando se presione un botón se deberá apagar el equivalente LED de la breakout board (estos se apagan y no se encienden, porque funcionan en lógica negativa).

Además, durante los 2 primeros segundos está programado en el código que se enciendan cuatro LEDs y cuatro permanezcan apagados.

3.2.2 Material utilizado.

El material utilizado en este diseño es el siguiente:

Software:

Lattice Diamond 3.8.

LatticeMico System.

Hardware:

FPGA MACH2OX-1200ZE junto con su breakout board.

Placa de extensión con botones y SWICHs.

3.2.3 Desarrollo.

3.2.3.1 Creación de los archivos necesarios.

Para comenzar, se crea una carpeta en un directorio del PC arbitrario, es recomendable que sea en una extensión cercana a la raíz del disco duro, ya que Lattice Diamond no funciona con ubicaciones y nombres de archivos muy largos.

Los nombres de la carpeta y de los archivos deberán ser cortos también.

Yo aconsejo ubicar los proyectos en la raíz del disco duro, así no tendremos problemas.

En mi caso se crea el siguiente directorio:

C:\TFG\ex2

En esta carpeta se irán añadiendo todos los proyectos y carpetas que genera Lattice Diamond.

Es muy importante que TODO lo que se vaya a utilizar se cree en esta carpeta, para que el software organice de forma coherente las subcarpetas que crea automáticamente.

Se abre el programa Lattice Diamond:

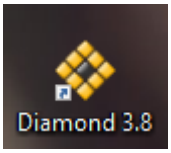


Figura 46 Icono Lattice Diamond

Se nos mostrará la página principal del mismo.

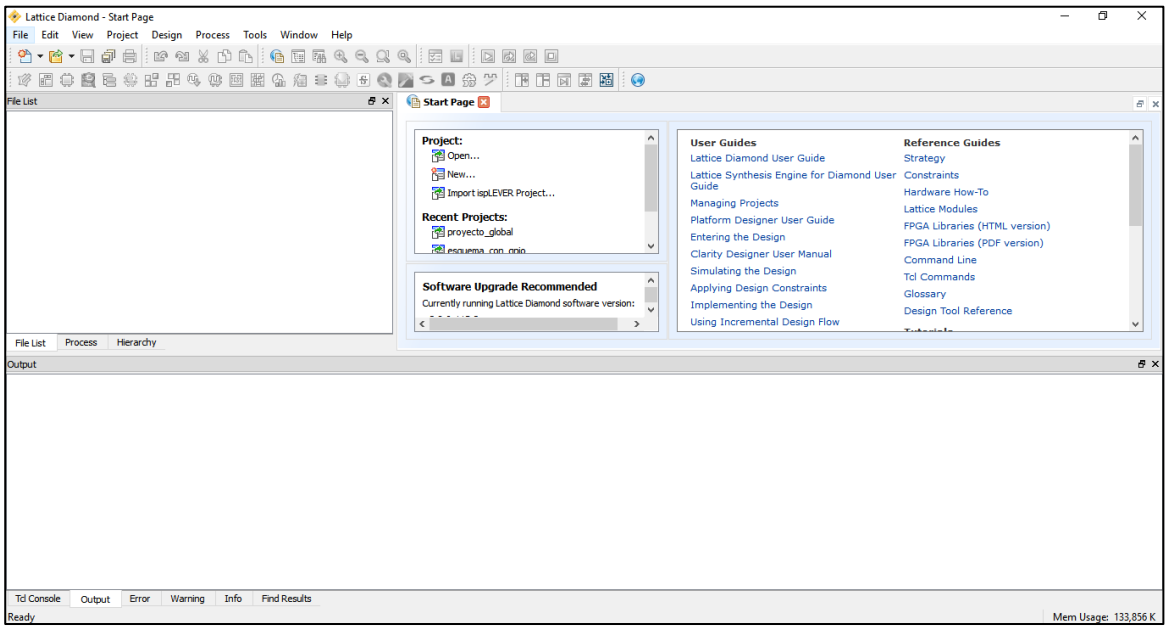


Figura 47 Lattice Diamond, ventana principal

Se crea un proyecto nuevo que guardaré en la carpeta creada anteriormente:

File > New > Project, seguidamente **Next.** En la ventana emergente de New Project.

A continuación, se completa la información de la siguiente ventana, ubicaremos todo en la carpeta que se haya creado.

Es recomendable que el nombre del proyecto y el de la carpeta de implementación sean el mismo.
Ya que esto generará dos elementos, un archivo ".lpf" (es el archivo proyecto)

y una carpeta que contendrá todos los archivos asociados al proyecto, cuando tengamos varios proyectos distintos en la misma carpeta, que los archivos tengan el mismo nombre facilita el trabajo.

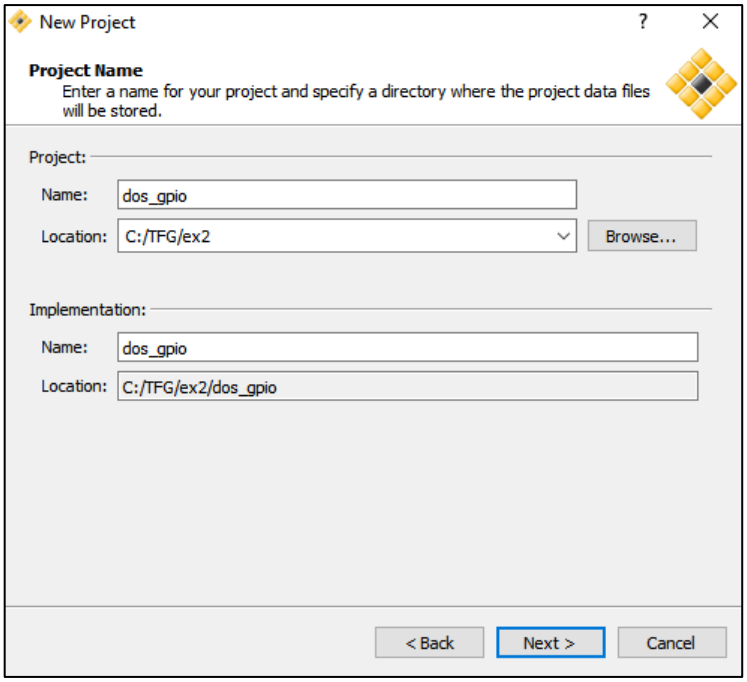


Figura 48 Nuevo proyecto Lattice Diamond

Cuando este todo hacemos clic en **Next**.

Hacer **Click en Next en la ventana de Add Source files**, ya que esta tarea lo haremos más adelante.

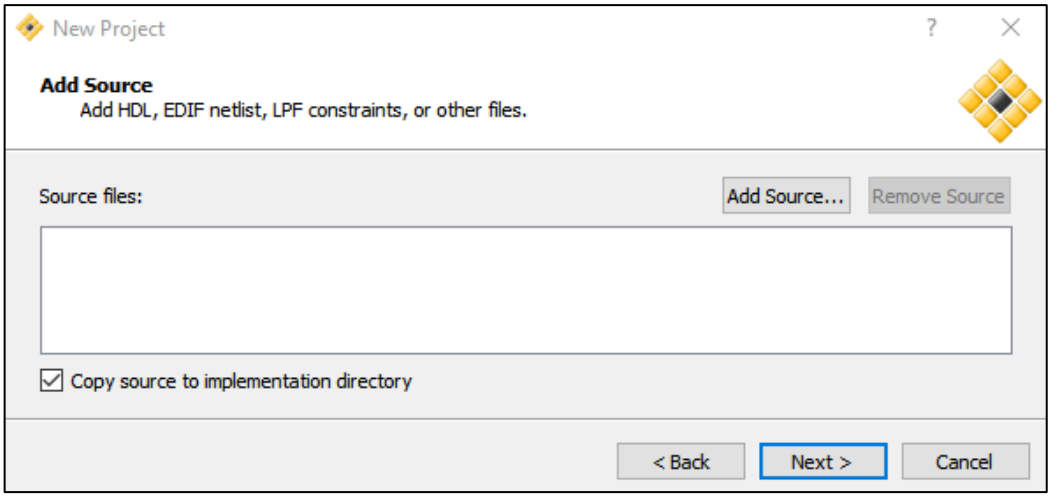


Figura 49 Add Source Lattice Diamond

En la siguiente ventana tenemos que **elegir el dispositivo que vayamos a usar**, es conveniente hacerlo bien, pero si nos equivocamos o queremos cambiar de dispositivo se podrá cambiar en cualquier momento.

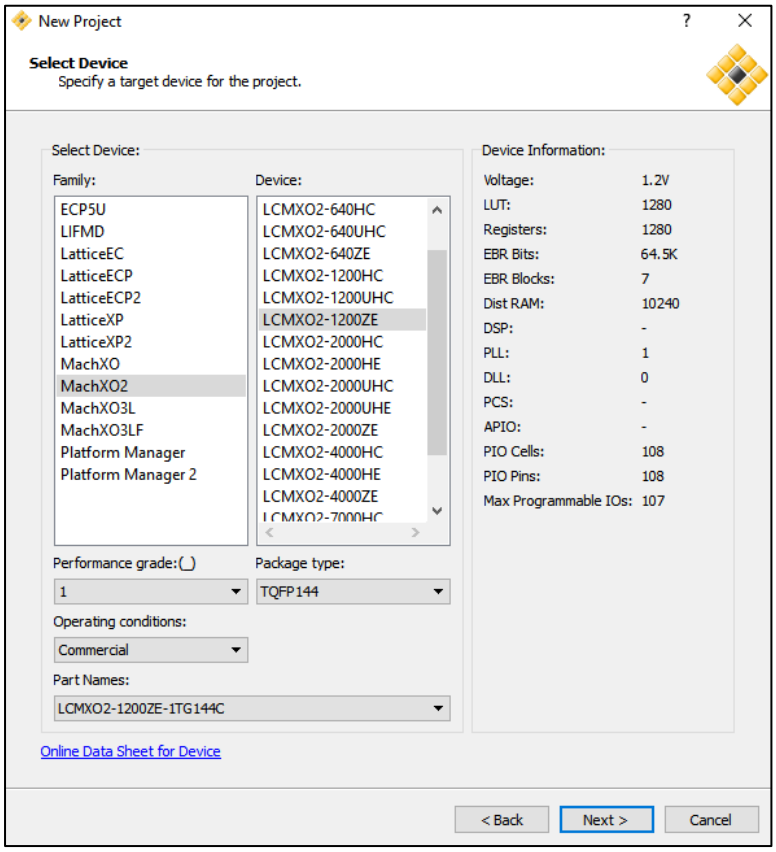


Figura 50 Selección de FPGA

Cuando lo tengamos hacemos **Click en Next.**

En la siguiente ventana se elige la herramienta de síntesis, **se selecciona la de Synplify Pro,** que será la que emplee el programa Lattice Mico.

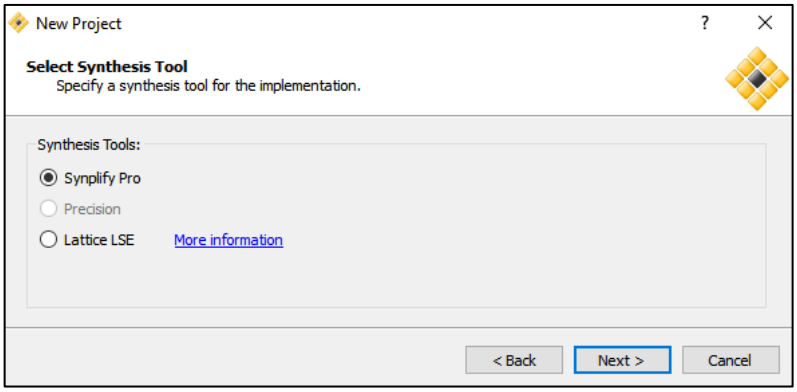


Figura 51 Selección de herramienta de síntesis

Click en Next.

Una vez esta todo tenemos la siguiente ventana con la información del proyecto:

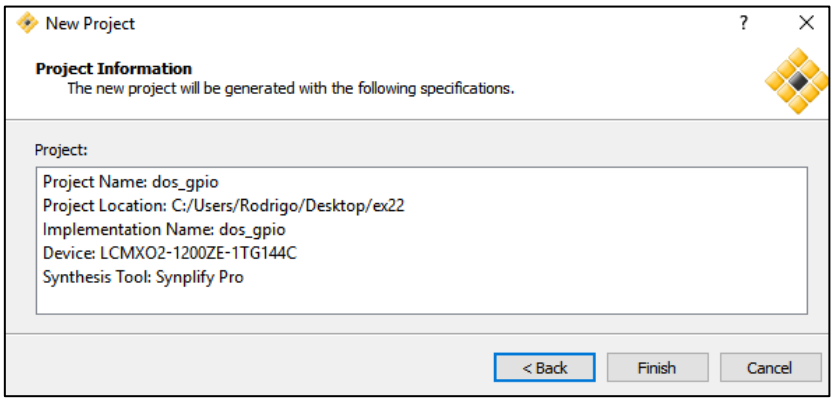


Figura 52 Finish creación de proyecto

Click en Finish.

Con esto tenemos el proyecto creado en Lattice Diamond, que será necesario para generar sobre él, el microcontrolador con el software de LatticeMico8 y posteriormente crear el símbolo desde este proyecto.

Una vez creado el proyecto nos aparecerán las siguientes ventanas:

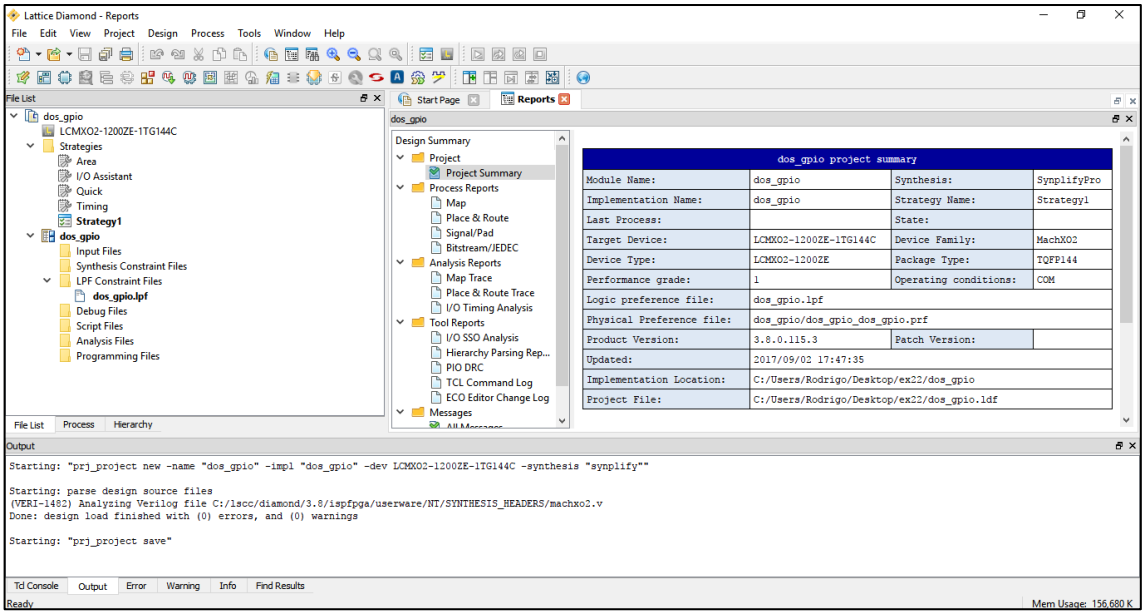


Figura 53 Proyecto creado Lattce Diamond

Cerramos el proyecto en Lattice Diamond clicando en la X Roja y si aparece alguna ventana emergente con opción de guardar, le damos a guardar.

A continuación, entramos en el **software de Lattice Mico8**

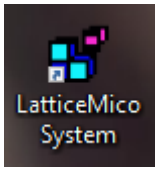


Figura 54 Icono Lattice MicoSystem

Aparecerá una ventana emergente para elegir el Workspace, hacemos clic en Ok en la que viene por defecto o la cambiamos a otra arbitrariamente, pero que no sea la del proyecto ya que se generan archivos que puede llevarnos a confusión.

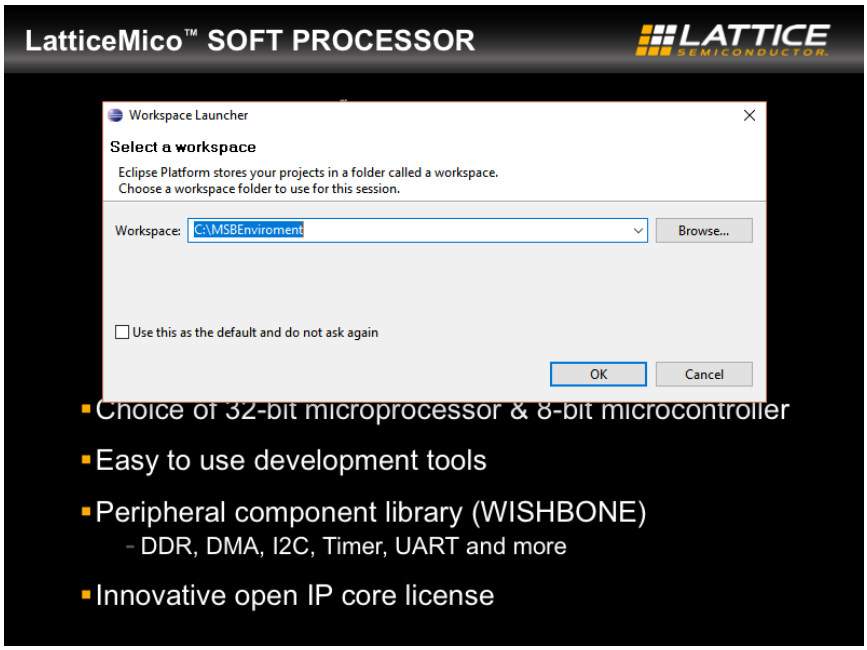


Figura 55 Workspace Lattice MicoSystem

3.2.3.2 Diseño del Hardware del microcontrolador

Una vez hemos entrado en el entorno (se trata de un entorno eclipse), hay que ir a la Perspectiva MSB cuyo botón podemos ver en la siguiente imagen.

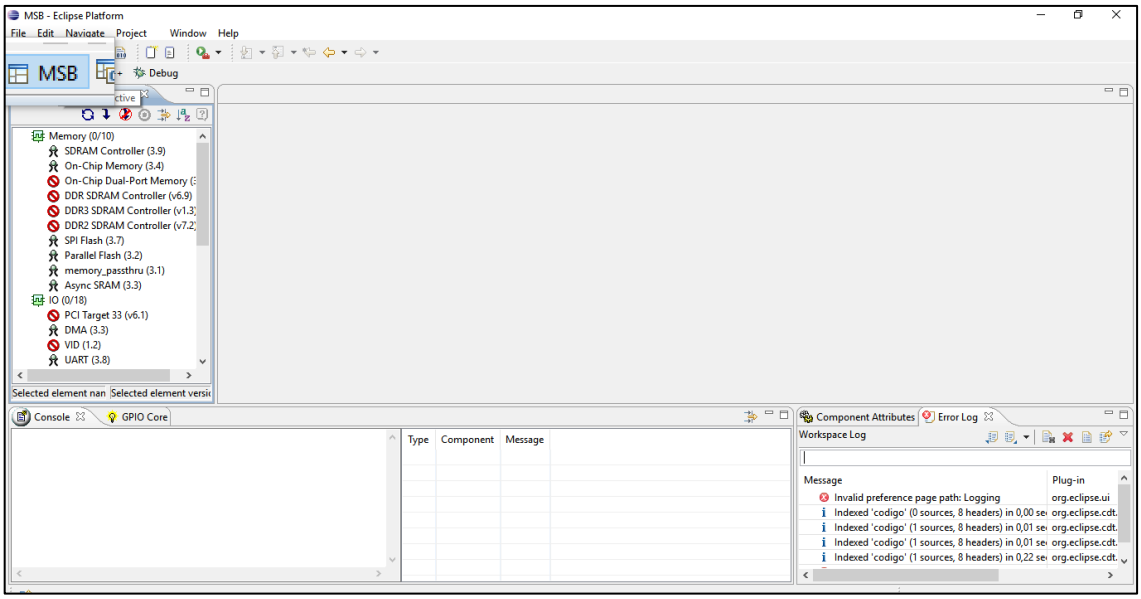


Figura 56 Perspectiva MSB Lattice MicoSystem

En esta perspectiva podremos añadir, modificar y eliminar los componentes "hardware" del microcontrolador, así como el propio microcontrolador. Si nos encontramos en otra perspectiva los diálogos desplegables File, Edit, Progect… son diferentes, será necesario asegurarse de en cual estamos antes de realizar ninguna operación.

Ahora hay que crear una nueva plataforma (se trata del hardware virtual a introducir), para ello **File > New Platform.**

Nos emergerá una ventana como la de la siguiente imagen:

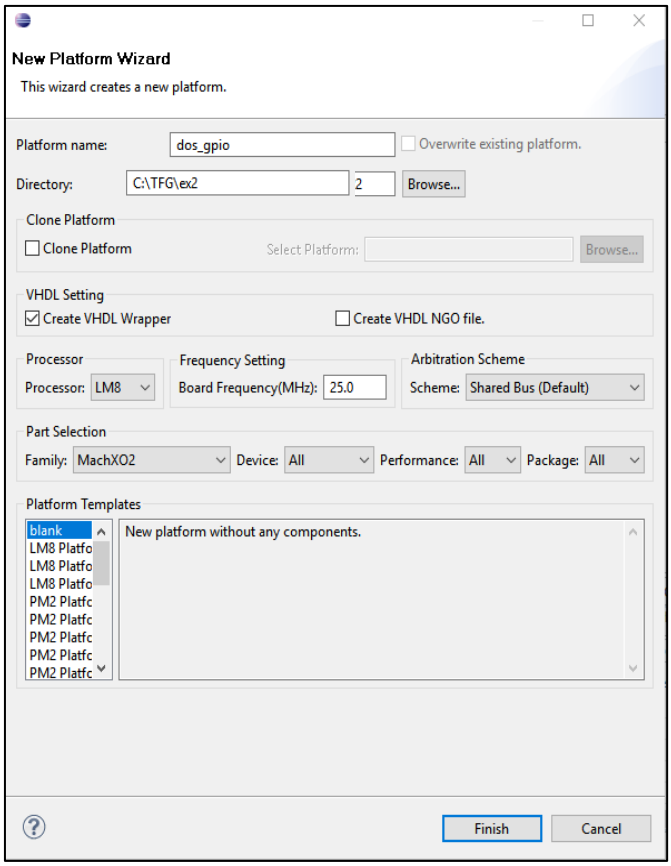


Figura 57 Nueva Plataforma

Deberemos **rellenar los campos tal y como aparecen.**

Es muy importante que la carpeta donde guardemos sea la misma de antes y tener la precaución de no guardar dentro de la carpeta que se ha generado con el proyecto del Lattice Diamond, es decir, si guardamos **dentro de** ***C:\TFG\ex2\dos_gpio*** **no se organizan las carpetas de forma coherente.**

También es muy importante **marcar la casilla de Create VHDL Wrapper**
Ya que nosotros vamos a manejar todo en lenguaje VHDL, pero este entorno (programa Lattice MicoSystem) funciona en lenguaje verilog, para luego combinar todo en los esquemas tendrá que estar en VHDL.

A parte de este comentario, podremos configurar el microprocesador como queramos, dentro de las posibilidades disponibles. En mi caso será un procesador de 8 bits, que trabajará en una frecuencia de 25 MHz, con una estructura de bus compartido y una memoria PROM de 2048 palabras.

Hacemos clic en Finish.

Vemos que nos aparece un entorno y nuevos botones en la barra de herramientas:

Se trata del entorno MSB ya descrito anteriormente, donde se puede crear el hardware del microcontrolador. Aquí se podrán añadir los distintos elementos que conformen su arquitectura y configurar las posiciones de memoria del procesador y de los periféricos. También se puede editar la jerarquía de las interrupciones.

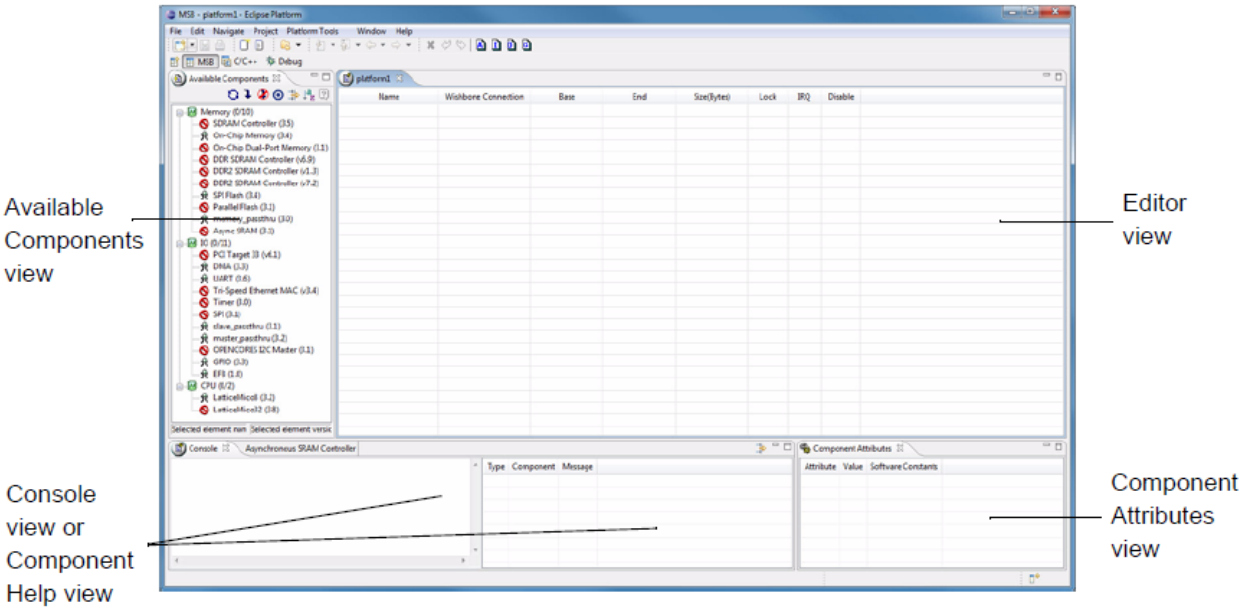


Figura 58 Ventanas Lattice MicoSystem

Ahora es el momento de añadir los componentes a nuestro Microcontrolador.

Lo primero es añadir un Microprocesador LM8, para ello lo tomaremos de la ventana Available Components view.

Haciendo Doble Clic lo añadimos a la ventana de Editor view, donde aparecerán aquellos componentes que vamos a utilizar.

Es importante indicar, que, si en cualquier componente hacemos únicamente un Clic, nos aparecerá toda la información sobre el componente a introducir, y nos dará toda la información acerca del mismo.

En la siguiente imagen podremos ver un ejemplo:

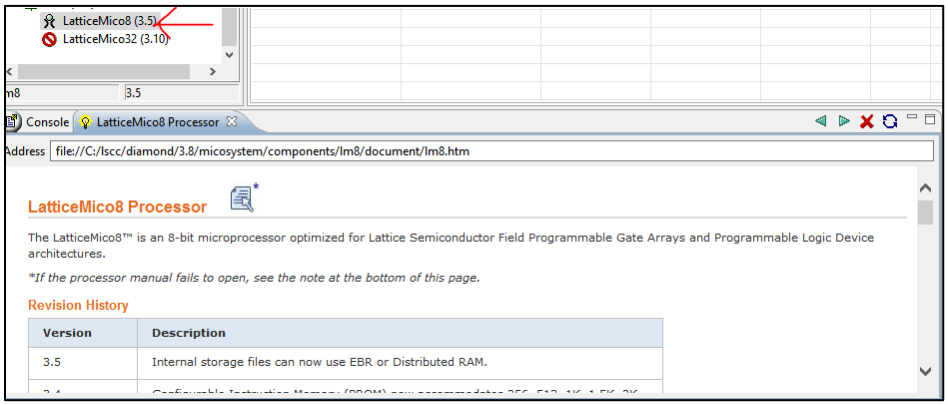


Figura 59 LatticeMico8

En esta ventana podremos hacer “Scrol” y ver la información relevante acerca del microcontrolador que voy a añadir.

Si se hace clic en el símbolo (junto al título del componente):

Se abre un documento PDF con toda la información del mismo, esta será muy importante ya que para crear el código C, de la aplicación que queramos introducir en el microcontrolador, se podrá usar funciones ya implementadas en “librerias” y en estos PDF se encuentra dicha información.

Cada componente tiene su propio PDF en el que encontraremos las funciones necesarias para trabajar con ellos.

Añado el microprocesador:

Doble Clic sobre el nombre del mismo, este aparte de tener el microprocesador ya contiene la memoria interna (que se usará para los registros, instrucciones y para almacenar el programa que tiene que ejecutar tras compilarlo).

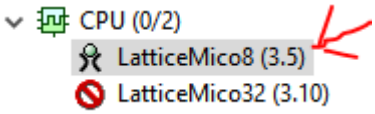


Figura 60 Microprocesador LM8

Como al crear el proyecto seleccione LM8 solo podré elegir el procesador de 8 bits.

La información referente al juego de instrucciones del micrprocesador se encuentra en el anexo 3. Tabla de instrucciones de LatticeMico8.

La información referente a la arquitectura del microcontrolador se encuentra en el anexo 4. Arquitectura de LatticeMico8.

Se despliega un menú en el cual puedo elegir las características del microprocesador:

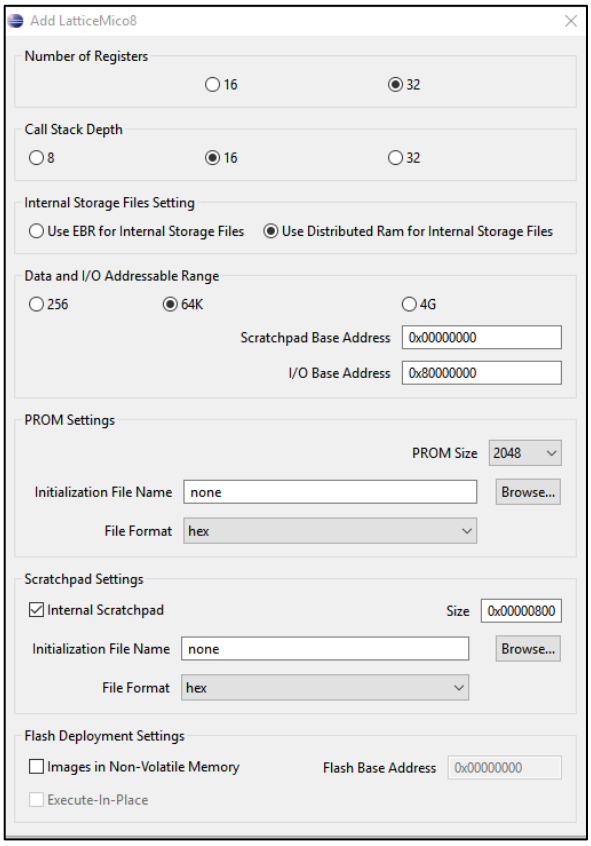


Figura 61 Configuración del LM8

Elijo las características del procesador y de la memoria, de momento no modifico las direcciones de memoria donde se almacenarán el código u otros elementos, ya que primero debo diseñar el conjunto y luego modificarlo para que funcione todo.

Pulso la **tecla Enter** cuando ya está todo decidido, me aparecerá en la pantalla Editor View el símbolo del procesador y su conexión al bus compartido (wishbone bus).

Name	Wishbone Connection	Base	End	Size(Bytes)	Lock	IRQ	Disable
LM8							☐
Data port	1						
Scratchpad		0x00000000	*0x000007FF*	*0x00000800*	☑		

Figura 62 LM8 en la ventana Editor View

La información referente al bus de comunicaciones se encuentra en el anexo 5. WISHBONE bus Interface.

A continuación, añado el resto de componentes que vaya a utilizar

En mi caso solo voy a añadir dos elementos adicionales, dos GPIO, unas para entrada y otras para salida.

Para ello hago **doble Clic en el nombre de GPIO y realizo la configuración de las mismas**.

Esta se encuentra dentro del Subgrupo I/O (entradas y salidas).

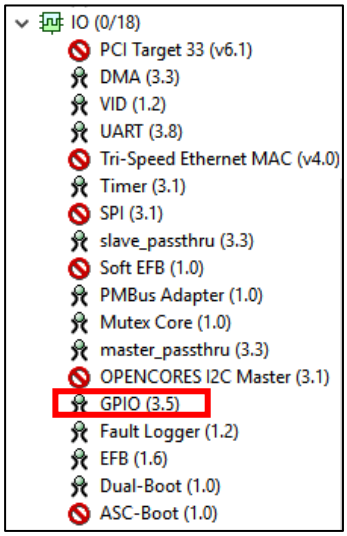


Figura 63 Inserción de GPIOs

En cada uno de los dos casos se desplegará una ventana que deberé **rellenar** de la siguiente manera:

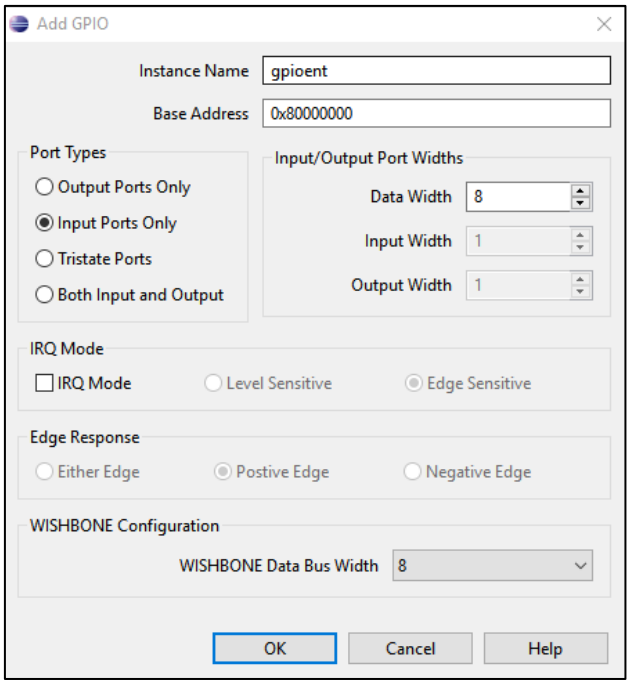


Figura 64 Configuración de GPIOs

En mi caso será de 8 bits, y por lo tanto, la configuro como en la figura 64, cabe destacar que los datos que vienen predefinidos de direcciones de memoria una vez estamos añadiendo elementos conviene no modificarlos.

En el caso de las GPIO de salida será exactamente igual, pero con el nombre "gpiosal" y seleccionando Output Ports Only en el tipo de puerto.

Al final de configurar cada uno de ellos, hacer clic en OK.

Una vez añadidos los componentes, tendremos el siguiente esquema en la ventana Editor View:

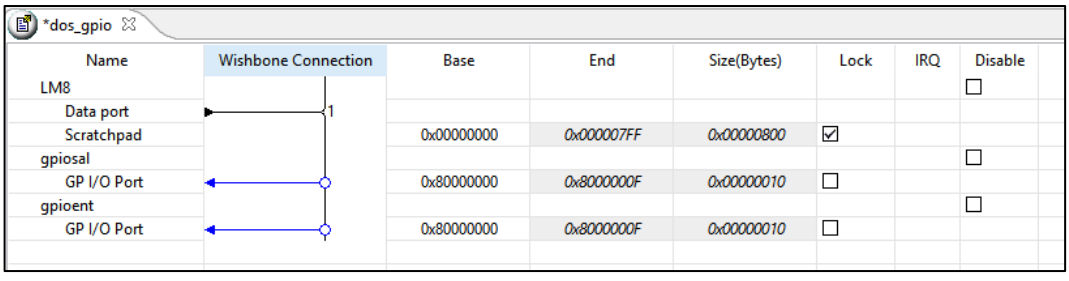


Figura 65 Elementos No conectados

Es hora de establecer las conexiones:

Debido a que se puede tener varios buses y buses de distintos tipos (se puede ver fácilmente, si al crear el proyecto cambiamos las opciones del bus y el procesador, el dibujo cambia) hay que hacer las conexiones manualmente.

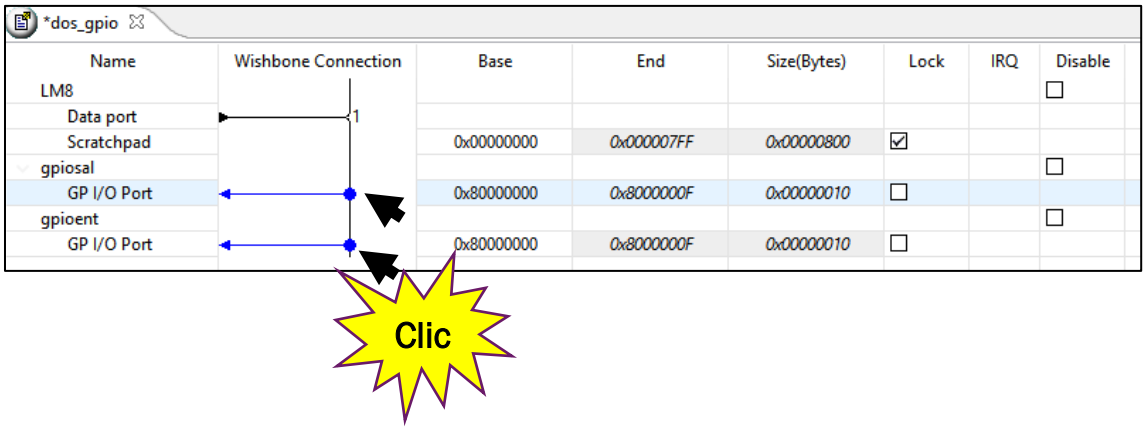


Figura 66 Conexión de elementos

Se deberá hacer clic en los dos puntos de tal manera que queden rellenos cuando estén así, significa que hay conexión entre el elemento y el bus.

Una vez tenemos todo el "hardware" del microcontrolador diseñado "compilamos", es decir, realizaremos las siguientes operaciones:

Hacemos clic en los siguientes botones de forma sucesiva y en el orden indicado (después explicare que hace cada uno), al realizar esta operación hay que asegurarse de que no aparece ningún error, si aparece, tratarlo antes de continuar.

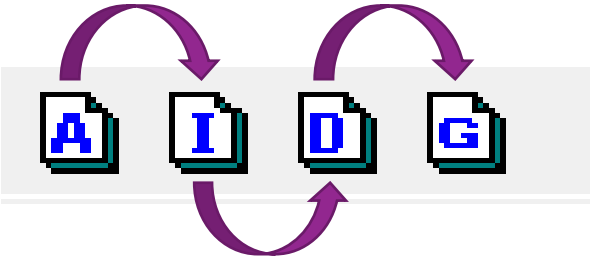


Figura 67 Proceso de creación del Hardware del microcontrolador

También se puede acceder a estas opciones desde el desplegable Platform Tools.

Se encarga de generar las direcciones de memoria, si no se modifican, el generador establece las más comvenientes, optimizando recursos.Si en la ventana Editor View se modifican, cambiando el valor y seleccionando la casilla LOCK, el generador lo tendrá en cuenta y, si es posible, establece esas direcciones donde se ha elegido. Estas se modifican en la columna "Base".

Se encarga de generar la jerarquia de interrupciones de los elementos, este generador utiliza la información de la columna “IRQ” de la ventana Editor View.

Desing Rule Check, verifica que los componentes, las direcciones de memoria, y las interrupciones se han diseñado correctamente y pueden ser implementadas.

Genera los archivos importantes almacenados en la carpeta...\soc\... con el diseño final del microcontrolador. En estos archivos, se aloja tanto el hardware como el software (se verá más adelante como integrarlo).

3.2.3.3 Integración del Software al microcontrolador

Una vez realizado todo esto, pasamos a crear la aplicación para el microcontrolador. Esta estará escrita en C o C++, se puede elegir, yo opto por código C, ya que estoy más familiarizado con él.

Hay que **cambiar a la perspectiva de C/C++**, en donde como ya se mencionó anteriormente muchas opciones y herramientas cambian.

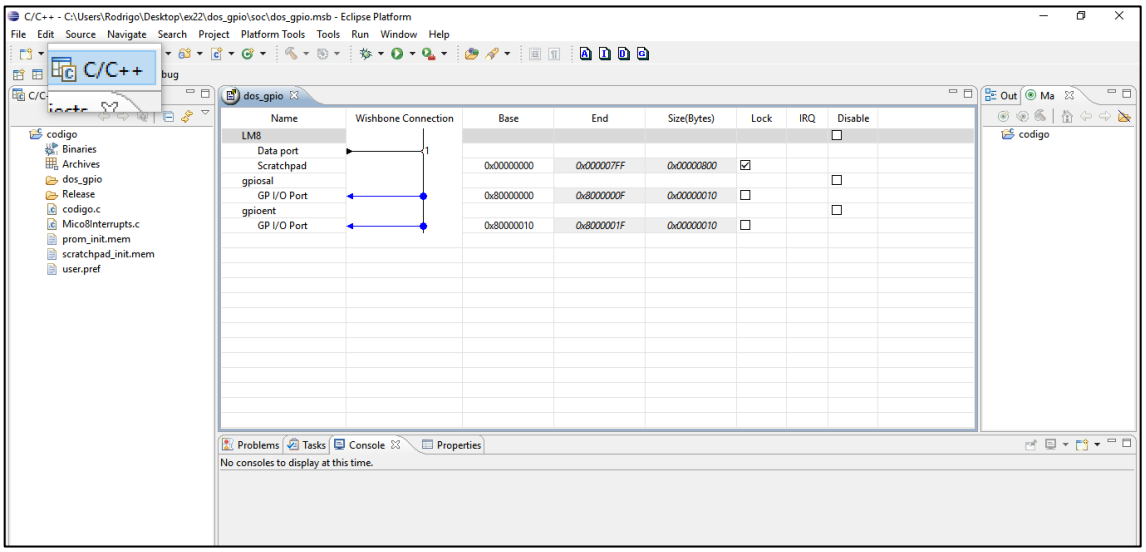


Figura 68 “Botón” para ir a Perspectiva C/C++

Lo que hay que hacer es crear un nuevo C/C++ SPE Project.

Para ello selecciono:

File > New > Mico Managed Make C Project.

Emergerá la siguiente ventana y en ella tengo que **completar los campos como se muestra en la figura 69.**

Es importante que la localización sea la del proyecto y seleccionar el archivo MSB que creamos en los pasos anteriores.

Aparecen varias plataformas predefinidas, pero se va a crear una específica, con los periféricos necesarios para nuestra aplicación. De esta otra forma tendría que haber configurado el MSB con otros componentes.

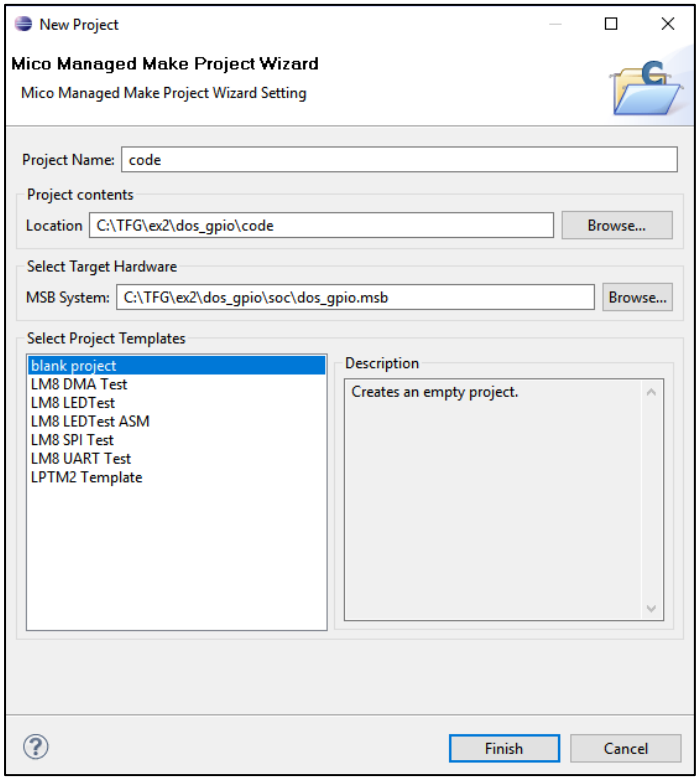


Figura 69 Creación Proyecto C

Cuando ya está el proyecto generado hay que añadir el código. Para ello será imprescindible añadir un archivo ".c" al proyecto.

En **File > Ne Source File ó:**

Figura 70 Nuevo Archivo C.

Hay que añadir la extensión del archivo ".c", si no se guardará como una hoja de texto (tal y como aparece en la imagen).

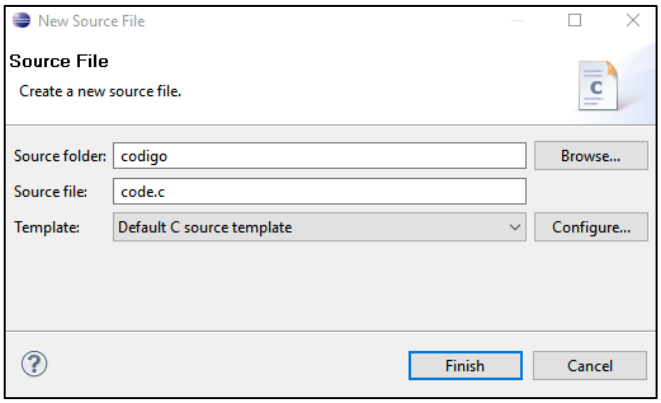


Figura 71 Añadir archivo C

Se hace clic en Finish.

Hacemos clic sobre el archivo C de la ventana C projects

Se añade el siguiente código en la ventana de edición de la Perspectiva actual:

```
#include "MicoUtils.h"
#include "MicoGPIO.h"
#include "DDStructs.h"

int main(void)
{
    unsigned char pos_interr;
    unsigned char valor_leds = 0xAA;

    MicoGPIOCtx_t *pos_gpio_ent = &gpio_gpioent;
    if(pos_gpio_ent == 0){
        return(0);
    }
        MicoGPIOCtx_t *pos_gpio_sal = &gpio_gpiosal;
        if(pos_gpio_sal == 0){
            return(0);
        }

        /* Escribir 0xaa en los leds y esperar 2 segundos */
        MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_sal->base, valor_leds);
        MicoSleepMilliSecs(2000);

    /* scroll the LEDs, every 100 msecs forever */
    for(;;)
    {
        MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_ent->base, pos_interr);
        valor_leds =pos_interr;
        MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_sal->base, valor_leds);
    }
    return(0);
```

Las tareas que va a desempeñar el microcontrolador para este código van a ser:

- Encender los LEDs impares de la placa Breakout board durante dos segundos, tras cada reset.
- A continuación, muestrear constantemente el estado de los interruptores, a través del puerto de entrada.
- Después, escribir en el puerto de salida la información muestreada, de forma que se reflejará continuamente en todos los LEDs el estado de los interruptores.

Para ello, se declara las GPIOs con las funciones que aparecen como: MicoGPIOCtx_t y hay que respetar los nombres utilizados en las variables y punteros de memoria asociados, ya que unen el código con los elementos llamados así en la perspectiva MSB. Son “gpioent” y “gpiosal”.

Se escribe 0XAA, haciendo uso de la variable “valor_leds” (que por defecto deberá ser de tipo “char”) durante dos segundos (enciende números impares) en el puerto de salida que pasado a binario es:

1010 1010 (en el bus de 8 bits)

En la función FOR del código, se puede observar, que lo que se hace, es copiar la información que hay en el puerto de entrada a una variable. Después, se escribe en la salida el valor de dicha variable. Como se puede observar se utilizan variables de tipo “char” que el software manipula en hexadecimal.

Este comportamiento descrito dentro de la función FOR se ejecuta cíclicamente.

Las librerías que se han añadido con los #Include son propias del elemento GPIO y vienen explicadas en el documento PDF que se obtenía en la ventana Help View en el botón 

Una vez escrito el código, se debe compilar. Para ello se debe continuar en la misma perspectiva (C/C++ perspective)

Se salva, (Hay que tener en cuenta que, si no se salva, el proyecto se compilará, pero utilizará aquello que estuviese de un proyecto anterior o nada, dando error).

Se compila: Project > Build Project.

Veremos que se crean nuevos archivos internos, como el makefile, los archivos de cabecera…

Se dispone, ya, del contenido de las memorias de programa y de los datos del microcontrolador que habrá que introducir en él.

Si todo ha funcionado correctamente, ahora hay que cargar el programa en el diseño del microcontrolador.

Hacemos clic en:

Tools > Software Deployment.

Aparecerá la venta que se muestra en el siguiente paso, Figura 72.

Seleccionar **Mico8 Memory Deployment**, y hacer clic en **New launch configuration**

Que es este botón → 

Configuramos como aparece en la figura 72

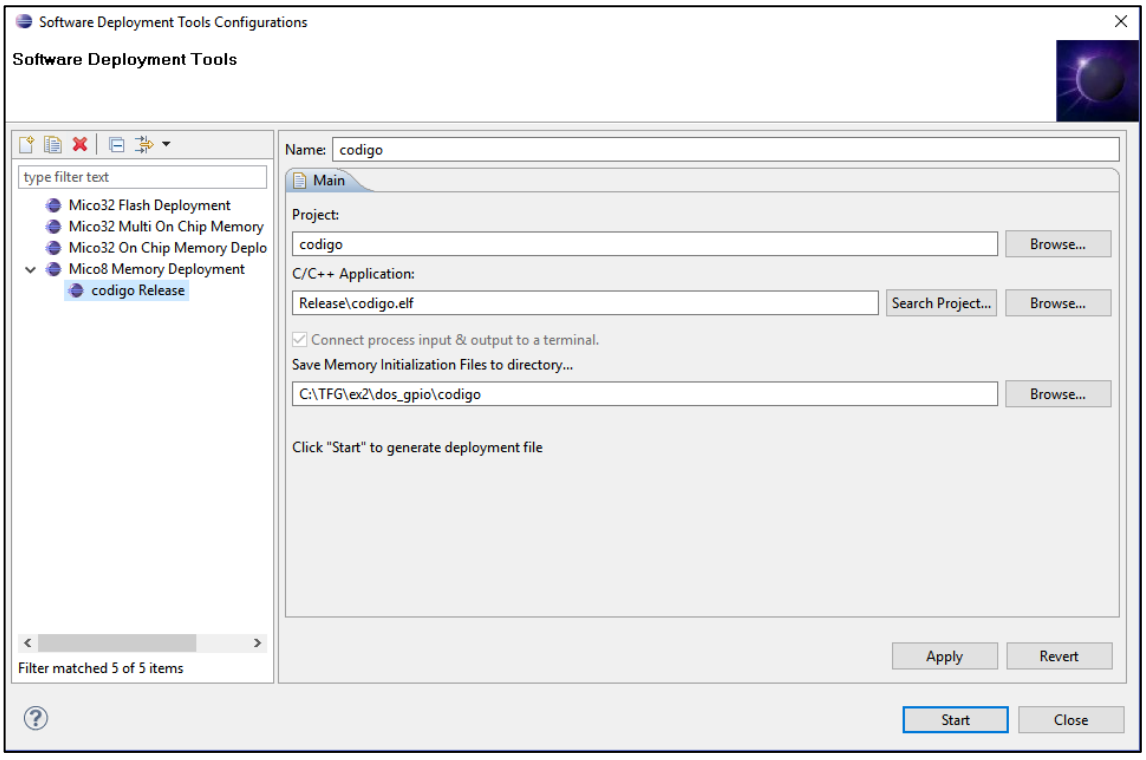


Figura 72 Ventana Software Deployment

Hacemos **clic en Apply y** seguidamente **Start**
Con este paso, se convierte el archivo con extensión ".elf" (proyecto en C, con el programa compilado) en archivos ".mem", que son necesarios para introducirlo en el diseño MSB, como se verá a continuación.

Ahora habrá que cargar el código en el microcontrolador, y con esto estaría todo el microcontrolador terminado, faltaría, añadirlo al esquema.

Vuelvo a la perspectiva MSB → MSB
Doble clic en el nombre del microprocesador o CPU LM8 para abrir su ventana de propiedades.

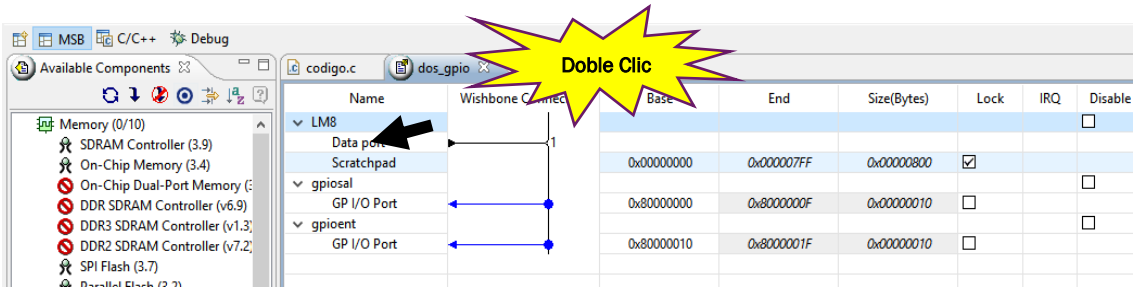


Figura 73 Introducción de programa

Abrimos la ventana del microprocesador LM8

La configuramos de la siguiente forma, como se muestra en la figura 74, (leer los archivos que he generado con el código que tiene que ejecutar para llevarlo a la memoria del diseño).

Estos archivos estarán en la carpeta del proyecto SPE C, que yo he llamado código y que estará dentro de la carpeta del proyecto "general" del microcontrolador.

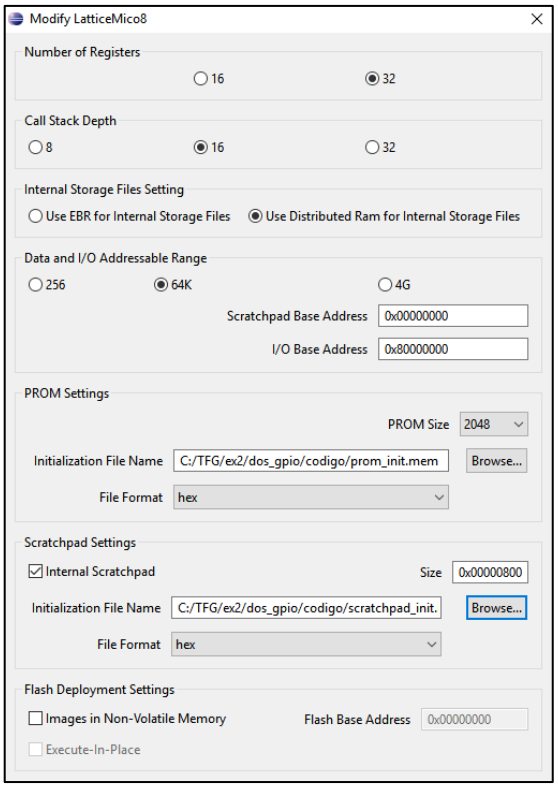


Figura 74 Insertar archivos ".mem" en el microcontrolador

Como se puede ver lo que estoy haciendo es incluir esos archivos “.mem” en las direcciones de memoria indicadas.

Cuando está todo listo **pulsar la tecla ENTER** para salir de la ventana guardando los cambios.

Hacer clic en  **ó Platform Tools > Run Generator**

Esto creará definitivamente el microcontrolador, con todos los elementos conectados y el código de la aplicación cargada en la memoria del LM8.

Salimos de LatticeMico8 salvando todo. Ya tenemos el microcontrolador implementado.

Los siguientes pasos no están previstos por Lattice, se expondrá en el apartado 3.2.4 Alternativas para incorporar el microcontrolador al proyecto.

3.2.3.4 Generar el símbolo del Microcontrolador para el esquema.

El método desarrollado a continuación permitirá trabajar con lógica de la FPGA externa al microcontrolador cómodamente. El utilizado por el fabricante Lattice no es tan evidente, e implica altos conocimientos de la herramienta y VHDL.

Abrir el programa Lattice Diamond y abrir el proyecto que se había creado al principio, para ello hacemos clic en:

File > Open > Project…

Y buscamos el archivo con extensión ".lpf" dentro de la carpeta principal donde íbamos a realizar el proyecto.

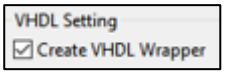
En mi caso → C:\TFG\ex2

Una vez está abierto el proyecto, hay que añadir el archivo VHDL que se ha generado del Lattice MicoSystem al proyecto. Este se encontrará dentro de la carpeta \soc dentro de la carpeta de la plataforma.

File > Add > Existing File…

En mi caso la plataforma se llama "dos_gpio" y en su interior está la carpeta \soc

C:\TFG\ex2\dos_gpio\soc

En este directorio encontraremos un archivo que terminara "_vhd.vhd" y con el nombre de la plataforma, este se ha generado porque marcamos la opción → 

cuando creamos la plataforma en el LatticeMico8.

Luego cargamos el archivo:

C:\TFG\ex2\dos_gpio\soc\ dos_gpio_vhd.vhd

Efectuamos la carga del archivo guardando. Guardar → 

Vamos a la ventana Hierarchy

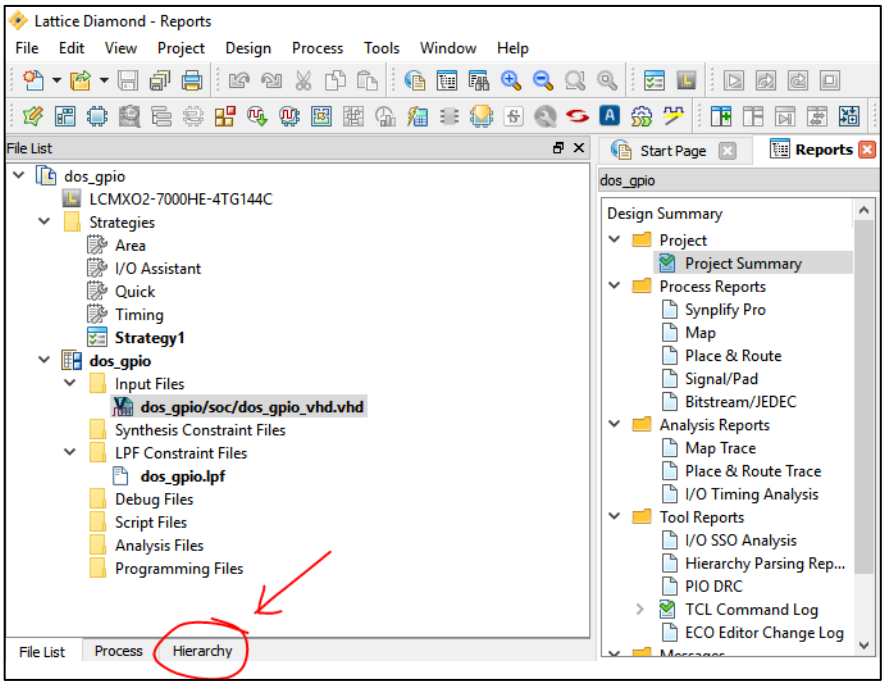


Figura 75 Ventana Jerarquía en Lattice Diamond

Hacemos lo mismo que en los otros proyectos para generar un símbolo, una vez estamos en la ventana Hierarchy hacemos clic con el botón derecho, desplegándose la ventana que se muestra a continuación:

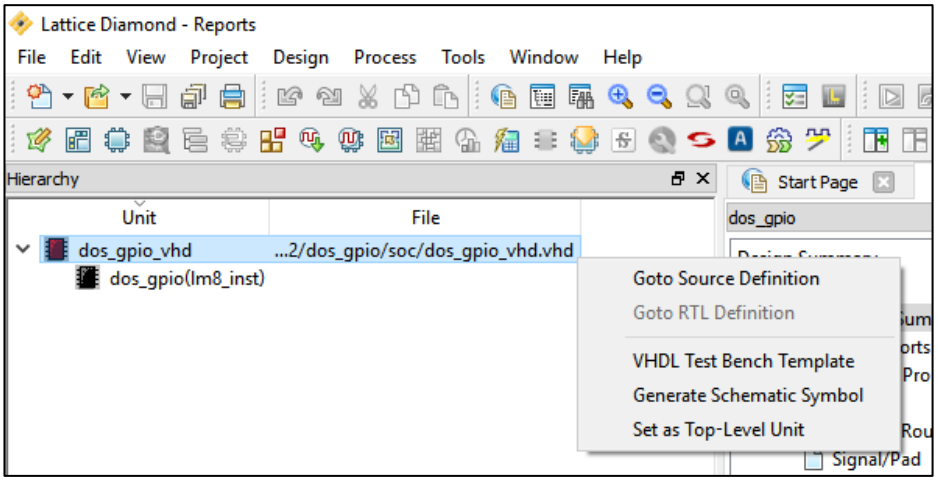


Figura 76 Establecer Jerarquia y crear Simbolo

Hacemos clic en Set as Top-Level Unit

Y volvemos a hacer clic con el botón derecho y seleccionamos Generate Schematic Symbol

Efectuamos todas las modificaciones guardando. Guardar →  .

Y cerramos el proyecto.

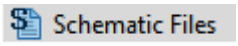
3.2.3.5 Introducir el símbolo del Microcontrolador en un proyecto con esquema.

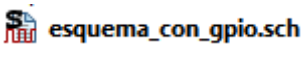
Hay que **crear un proyecto** nuevo en el cual dibujaremos el esquema con el microcontrolador.

En mi caso se llamará → "esquema_con_gpio"

Igual que en otros diseños (diseño1, por ejemplo), **creo un archivo Schematic**

File > New > File...

Selecciono Schematic Files 

En mi caso lo he llamado: 

Al finalizar antes de compilar, haré top leve a este archivo, igual que en cualquier otro proyecto **Set as Top-Level Unit (al archivo esquema)**

Y será en este archivo de tipo esquema, donde añada el simbolo del microcontrolador, que, como cualquier otro simbolo, este aparerá con el nombre de la plataforma.

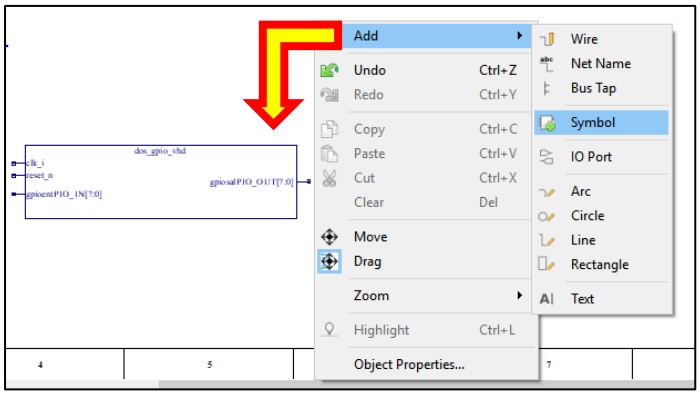


Figura 77 Insertar Símbolo del microcontrolador

En este proyecto se trabajará igual que en los otros proyectos, como el diseño1, pero habrá unas pequeñas diferencias que voy a explicar a continuación:

Lo primero, para que el símbolo del microcontrolador funcione, deberé añadir dos archivos a la File List:

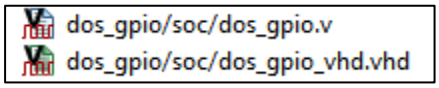


Figura 78 Archivos que añadir

Lo realizaré con el procedimiento **File > Add > Existing File...**

Estos archivos están ambos en el directorio:

C:\TFG\ex2\dos_gpio\soc

Como vemos, se trata del archivo utilizado anteriormente en el proyecto, para generar el símbolo. El otro es de tipo Verilog, que es el leguaje que usa Lattice Diamond por defecto. Este lo usará para las Path (ficheros que dan contenido al símbolo).

Segundo

Tools > Options.

Se abrirá la siguiente ventana. Hacer clic en **General,** situado en la izquierda.

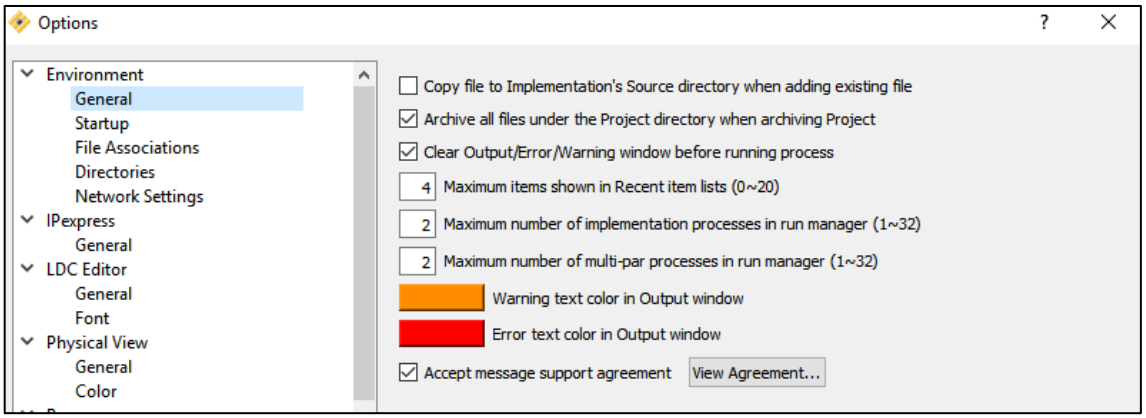


Figura 79 Unchek en Options

Si la opción Copy file to Implementation's Source directory whenadding existing file está seleccionada, hay que **deseleccionarla.** Tiene que estar igual que en la figura 79

Tercero

Como el proyecto tiene componentes en VHDL y Verilog, ya que LatticeMicoSystem trabaja en este lenguaje, hay que realizar los siguientes pasos:

Project > Property Pages

Se abre la siguiente ventana (figura 80), donde hay que completar el recuadro señalado en rojo, con la dirección donde se ubica el archivo verilog.

Se escribirá la dirección de la carpeta \soc, de la plataforma del microcontrolador disñado, en mi caso:

C:\TFG\ex2\dos_gpio\soc

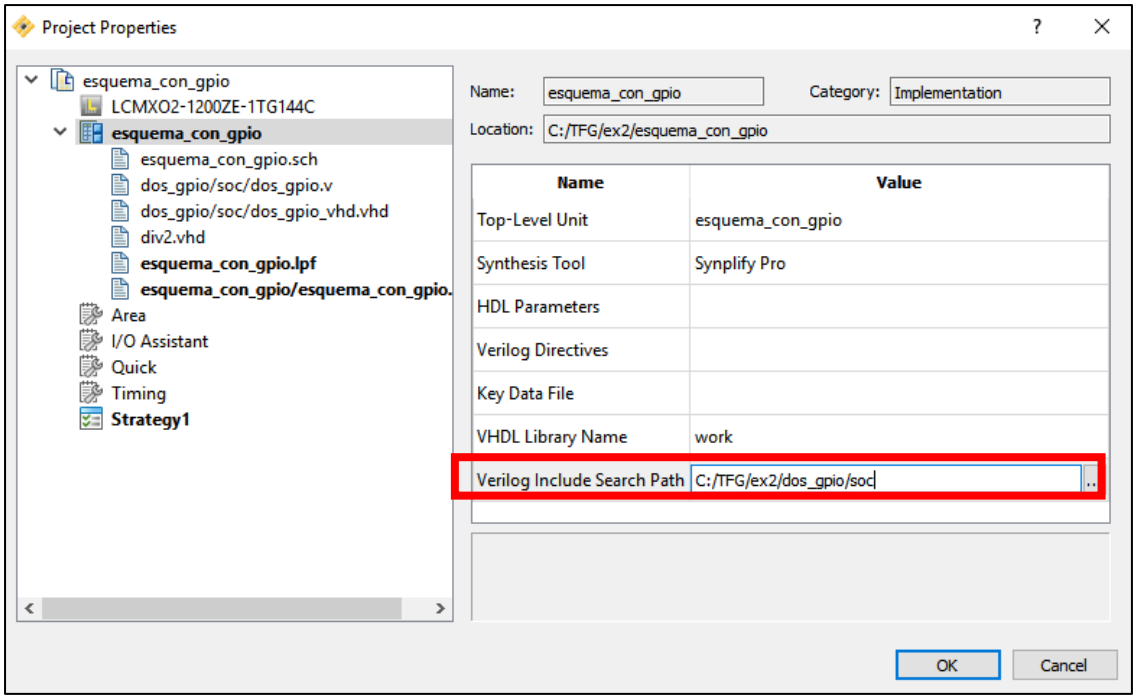


Figura 80 Permitir archivos Verilog

De esta forma, se ha establecido un procedimiento general, para generar un símbolo de un microcontrolador implementado con la herramienta LatticeMicoSystem, y, posteriormente incorporarlo a un esquema, en el que se le podrá conectar con otros elementos, en diseños más complejos.

3.2.3.6 Esquema del proyecto y su funcionamiento.

A continuación, seguiré explicando el diseño que se ha realizado, como funciona y que contiene a parte del microcontrolador.

En el esquema el núcleo central es el microcontrolador, al cual le entra un bus de 8 bits correspondientes a las GPIO de entrada, y sale de él un bus con otros 8 bits correspondientes a las salidas.

Las salidas se conectan a output buffers para llevarlas a los pines de la FPGA, lo mismo sucede con las entradas, estas están unidas a input buffers y vinculadas a patillas de la FPGA.

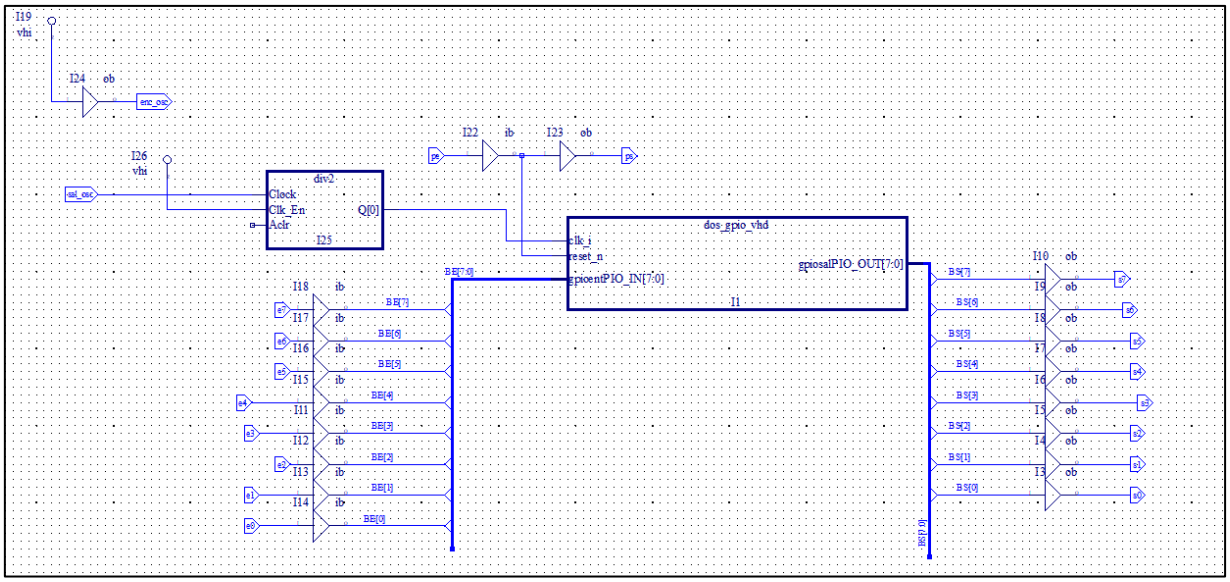


Figura 81 Esquema con Microcontrolador.

Además, también se puede ver que el reset está conectado a otro pin para poder reiniciar en cualquier momento, de tal forma que cuando este se active, se volverá a ejecutar desde el principio todo el código de la aplicación cargada. En el caso de este diseño volverán a encenderse los LEDs impares durante 2 segundos.

Otro dato importante de mi diseño es que el microcontrolador no posee oscilador interno, y, por lo tanto, se puede observar que el símbolo tiene una patilla llamada “clk_i”, que deberemos alimentar con un tren de pulsos a una frecuencia de 25 MHz, como se especificó al diseñar el microcontrolador.

Para generar la frecuencia de 25 MHz se divide por 2 la frecuencia de 50MHz, disponible en el oscilador que se ha añadido a la placa breakout board.

Div2 es un contador de 1 solo bit que cuenta del ‘0’ al ‘1’, dividiendo la frecuencia en 2. El bloque estará siempre activo, y como señal de reloj será la del oscilador externo.

La parte del esquema que se muestra en la figura 82, sirve para utilizar el oscilador externo. A este se le añade un “Vhi” que lo habilita permanentemente.

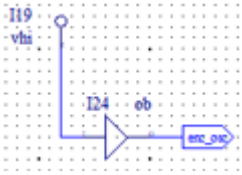


Figura 82 Habilitación de reloj externo

3.2.4 Alternativa para incorporar el microcontrolador al proyecto.

Lo habitual y dispuesto por el fabricante Lattice para implementar el microcontrolador, es generar un archivo VHDL o verilog, que sea el nivel superior de la jerarquía del diseño, instanciar todos los componentes del diseño, y establecer todas las interconexiones, descritas con código dentro del archivo top Level (VHDL o verilog).

Es decir, hacer una descripción estructural, VHDL o verilog, del sistema completo. Se trataría de un proceso laborioso, y si, como es nuestro caso, es un sistema complejo, las posibilidades de cometer errores son muy grandes.

En este proyecto se plantea un sistema alternativo, no contemplado por el fabricante, en el que se genera un símbolo del microcontrolador, y se conecta al resto de los elementos en un esquema.

De esta forma al salvar el esquema, se genera, de forma automática, un fichero verilog, que será el nivel superior de la jerarquía. Así se consigue la creación de diseños de una forma más sencilla, más rápida, evitando la posibilidad de errores y de manera visual.

3.3 Diseño 3: Integración del Microcontrolador en un diseño con funcionamiento dinámico.

En este diseño, se busca unificar los dos diseños anteriores (diseño 1 y diseño 2) en uno nuevo, en el que el microcontrolador y los bloques que conformaban el medidor de distancia compartan información.

Esto demostrará que es posible realizar diseños complejos con bloques funcionales creados en VHDL e IPExpress y combinarlo con lo diseñado en LatticeMico.

En este caso el microcontrolador desempeñará una función trivial, que es copiar los datos a la entrada y mostrarlos a la salida, justo antes del decodificador de uno de los visualizadores de 7 segmentos.

Aunque se trata de una tarea “sencilla” conviene no introducir muchos cambios en un diseño, puesto que así es más sencillo diagnosticar errores.

3.3.1 Objetivo del diseño.

Unir los dos proyectos anteriores y comprobar que funciona correctamente.

Se comparará que los dos dígitos de los visualizadores se muestran a la vez, esto indica que el microcontrolador no introduce retrasos.
Uno de los visualizadores de 7 segmentos estará unido a un decodificador que va directamente al contador. El otro tendrá entre medias el microcontrolador.

Aprender cómo gestionar los archivos generados con Lattice Mico8, ya que mientras que en los proyectos de Lattice Diamond simplemente hay que copiar y pegar, en Lattice MicoSystem hay que modificar directorios y repetir ciertos pasos, o incluso crear el microcontrolador desde cero.

3.3.2 Material utilizado.

Se ha utilizado los siguientes materiales para el diseño:

Software:

Lattice Diamond 3.8.

LatticeMico System.

Hardware:

FPGA MACH2OX-1200ZE junto con su breakout board.

Placa de extensión con botones y SWICHs.

Placa de extensión con dos displays.

Sensor de Ultrasonidos LV-MaxSonarEZ3.

3.3.3 Desarrollo.

Se parte del diseño 1, el cual no tiene incorporado ningún elemento de Lattice MicoSystem.

Se copia todo lo que contenía la carpeta del diseño 1 en una nueva que creo con el nombre de “exp3”.

C:\TFG\exp1 → C:\TFG\exp3

Se abre el proyecto global del esquema en la carpeta “\exp3” y se comprueba que todos los archivos están vinculados al nuevo proyecto, y su directorio ha sido modificado apareciendo “exp3” como el destino donde están localizados ahora.

Se genera de nuevo un microcontrolador siguiendo los pasos del diseño 2, ya explicados, creando un nuevo proyecto el Lattice Diamond y LatticeMico. Lo construyo exactamente igual que en el diseño 2, con dos GPIO y un microcontrolador de 8 bits LM8.

En este diseño se programará el mismo código en C que en el diseño 2, pues se busca unir los dos proyectos sin introducir muchos cambios (como ya se mencionó), después, se verificará que el funcionamiento es correcto y en diseños posteriores se alterará el software.

Para ello una vez realizados todos los pasos anteriores, abrimos el proyecto global.

Traemos al proyecto el símbolo, como en el diseño 2

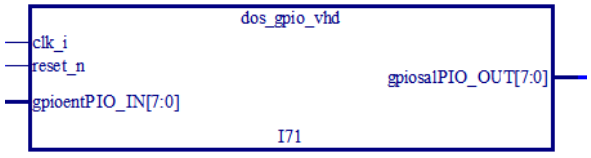


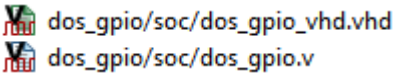
Figura 83 Símbolo microcontrolador

Y se añaden los archivos de la carpeta ..\soc\ que estarán en:

C:\TFG\exp3\dos_gpio

Con File → Add → Exting File…

Y se añaden los archivos:



Igual que se hizo en el diseño 2, pero, ahora en el nuevo diseño.

Una vez hecho esto, se modifica el esquema y se verifica el comportamiento del diseño.

Ha sido necesario añadir un bloque VHDL llamado “comber_bus” que se utiliza para que el bus que tiene que entrar al microcontrolador, sea de 7 bits y no de 6.

Al ser de 6 no se puede conectar directamente, en el código VHD lo que se hace es añadir un bit al bus (el bus en código VHDL es un vector).

A continuación, muestro el código empleado:

```
1 library IEEE;
2 use Ieee.STD_LOGIC_1164.all;
3
4 entity conver_bus is
5 port
6 (
7 ent: in std_logic_vector(6 downto 0);
8 sal: out std_logic_vector(7 downto 0)
9 );
10 end conver_bus;
11
12 architecture conver_arch of conver_bus is
13 begin
14 sal <= '0' & ent ;
15 end conver_arch;
```

Después se ha realizado la conexión que se muestra a continuación en el esquema:

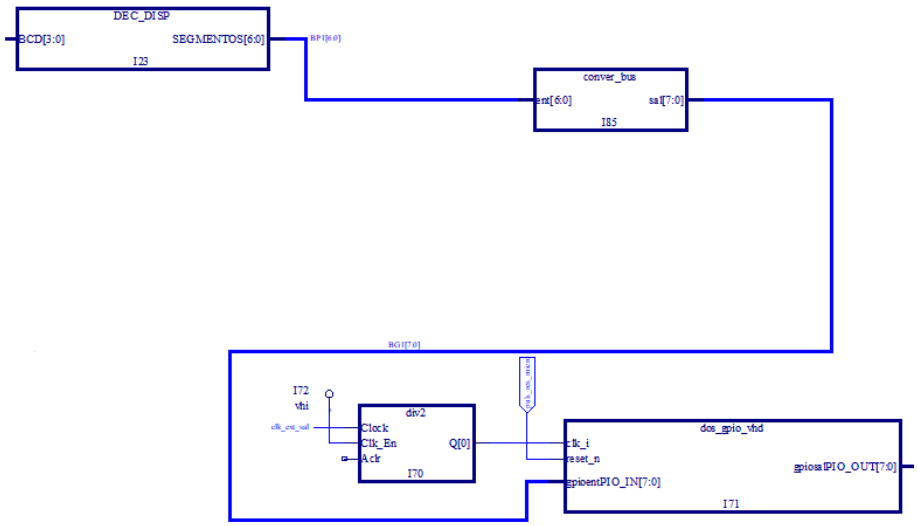


Figura 84 Esquema con unión del Microcontrolador, a través de "comber_bus"

Como se aprecia en la figura 84, se puede observar que el microcontrolador hace un "bypass" de la salida del decodificador del segundo digito al correspondiente visualizador de 7 segmentos.

Además, durante los dos primeros segundos, se puede observar que se encenderán los LEDs del segundo digito de la siguiente manera: 1010 1010, ya que es lo que estaba programado en la aplicación C del diseño 2, y ahora esa misma aplicación está en este microcontrolador.

A continuación, se puede observar el símbolo del microcontrolador correctamente conectado:

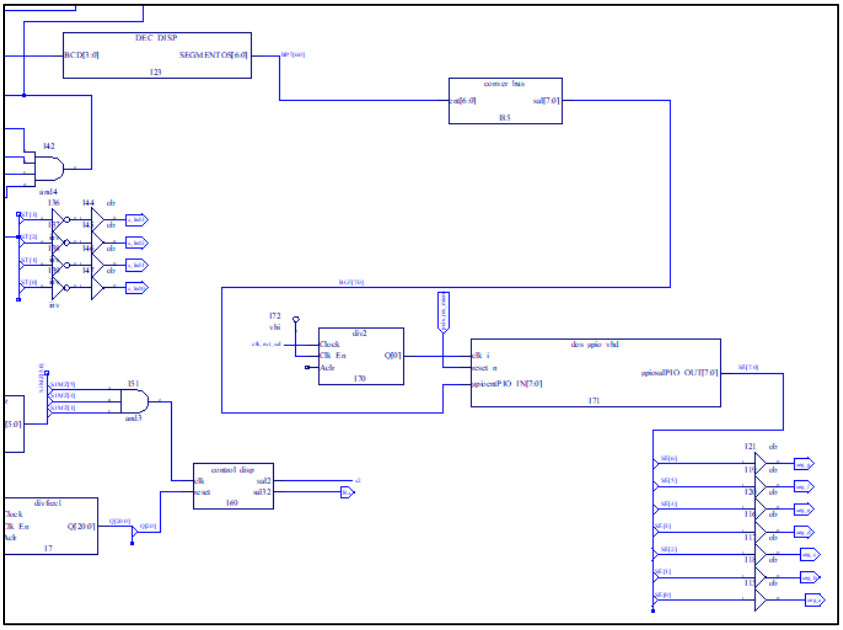


Figura 85 Esquema de conexiones para el microcontrolador, en el diseño 3

Una vez realizados estos pasos, se configura la FPGA y se observa el comportamiento.

El esquema del diseño final, en el cual se incluyen todas las conexiones de los bloques, se puede observar en la siguiente imagen:

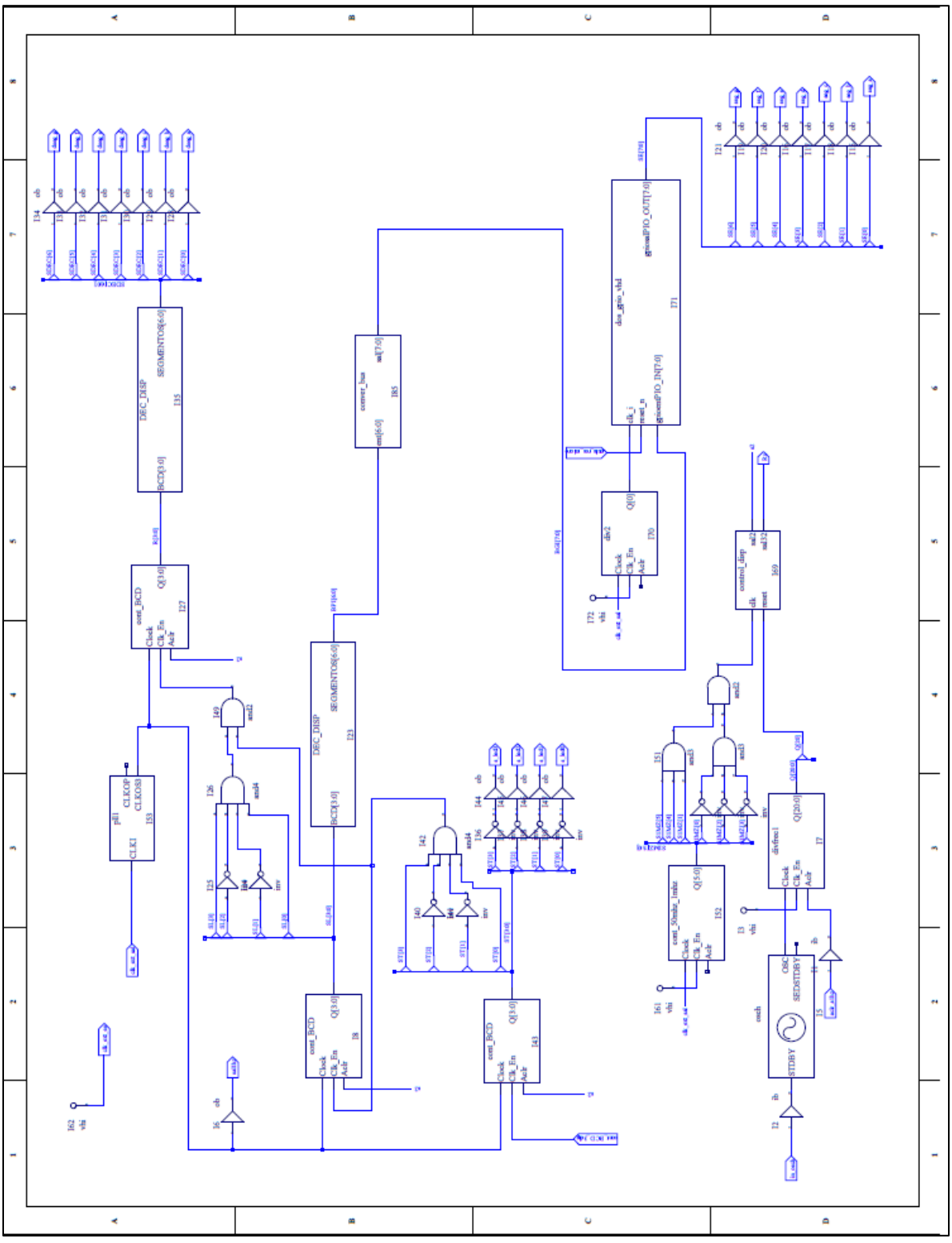


Figura 86 Esquema Diseño 3

3.3.4 Mejora del diseño, integración de registros.

Se realiza una mejora a este diseño y al diseño 1, en el cual se añaden una serie de registros o biestables tipo D.

Los objetivos de esta mejora son varios:

Evitar el parpadeo de los visualizadores de 7 segmentos mientras están contando, ya que resulta “vistosamente” menos profesional.

Al añadir los registros a los diseños con microcontrolador, se mantendrán las entradas de datos constantes, porque la salida de los registros mantiene el valor de su entrada hasta que se dé un flanco de subida en su entrada de reloj.

De esta forma se mantendrá el valor durante un tiempo igual al periodo de la señal de reloj. En los diseños se alimentarán estos registros con una frecuencia de 1 Hz.

En el diseño 1 anterior, durante los primeros milisegundos de cada refresco, se podía observar un parpadeo, que se reflejaba en los visualizadores de salida.

En este diseño, los parpadeos los recibía el microcontrolador a la entrada, no es recomendable que cambien los valores mientras está realizando cálculos con los datos de entrada.

Estos registros se añaden como símbolos. La herramienta de esquema de Diamond Tool posee estos elementos en su carpeta local.

No todos los registros pueden ser implantados en todos los dispositivos. Los que tienen nombre “fd1s3ax” sí, funcionan en los dispositivos MACHXO2.
Se puede ver a continuación como son, sus entradas y salidas:

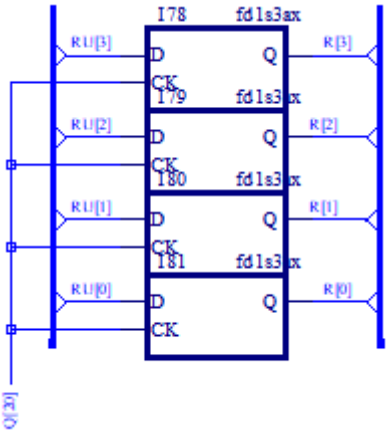


Figura 87 Registros fd1s3ax

Se alimentará con el reloj de 1 Hz (señal Q[20]) a cada registro, de tal manera que cada segundo cargará un nuevo valor procedente del bus del contador del sensor.

Se ejecuta la herramienta de síntesis (Synplify Pro), se seleccionan adecuadamente las patillas de la FPGA a utilizar, y se genera el archivo Jedec.

Se configura la FPGA MACHXO2 1200ZE y se comprueba el funcionamiento del dispositivo.

3.4 Diseño 4: Alteración del software del Microcontrolador.

En este diseño se manipulará el software del microcontrolador, haciendo uso de Lattice MicoSystem. Concretamente en la C/C++ perspective.

En este caso, el microcontrolador ya no va a desempeñar una función tan trivial, como en los diseños anteriores, si no que se le incorporará un programa o código secuencial complejo, que debe ejecutar.

Al igual que antes, no conviene introducir muchos cambios en un diseño, puesto que así, es más sencillo diagnosticar errores.

Además, este diseño ha servido para comprender muy bien cómo se organizan los archivos en Lattice Mico, elaborar un método para cargar software a un microcontrolador implementado en la FPGA y modificarlo con facilidad.

Como sucedía en los otros diseños, hubiese sido más sencillo si la documentación de Lattice MicoSystem fuese completa, ya que la más adecuada es el tutorial de Lattice Mico y las hojas de características de los elementos a utilizar, en la cual no se contemplan estos problemas.

3.4.1 Objetivo del diseño

Comprender el funcionamiento de Lattice MicoSystem con el software C que emplea y las funciones asociadas.

Identificar que funciones de C se pueden utilizar y cuáles no. El microprocesador alojado es de 8 Bits, no tiene multiplicador, no tiene divisor, no utiliza la memoria Flash…

Trabajar en sistema decimal, y entender cómo se comportan las funciones de las GPIO.

Aprender como cargar el programa en C, con facilidad en Lattice MicoSystem, entendiendo que pasos hay que repetir de la creación del microcontrolador para cargar el programa.
Como se verá a continuación, la gran ventaja es que ahora ya no hay que empezar desde cero a crear el Microcontrolador.

Modificar los archivos internos del procesador y llevarlos al esquema con facilidad.

3.4.2 Material utilizado.

El material empleado para este diseño es el siguiente:

Software:

Lattice Diamond 3.8.

LatticeMico System.

Hardware:

FPGA MACH2OX-1200ZE junto con su breakout board.

Placa de extensión con botones y SWICHs.

Placa de extensión con dos displays.

3.4.3 Desarrollo.

Se parte del diseño 2, cuyo proyecto contiene el esquema que incorpora el símbolo del Mico8 llamado “dos_gpio”.
Se copian todos los elementos que contenía la carpeta del diseño 2 en una nueva carpeta llamada “exp4”.

C:\TFG\exp2 → C:\TFG\exp4

También se puede comenzar desde cero y crear un nuevo esquema como el del diseño 2.

En este se le introduce una pequeña mejora, puesto que voy a trabajar con los LEDs de la Breakout board (para observar y analizar resultados), se sitúan unos inversores justo antes de la salida, debido a que los LEDs de la placa trabajan en lógica negativa, cuando se quiera ver un ‘0’ se verá un ‘1’ y viceversa.

Esto simplificará la tarea de interpretar los resultados.

El esquema queda como el siguiente:

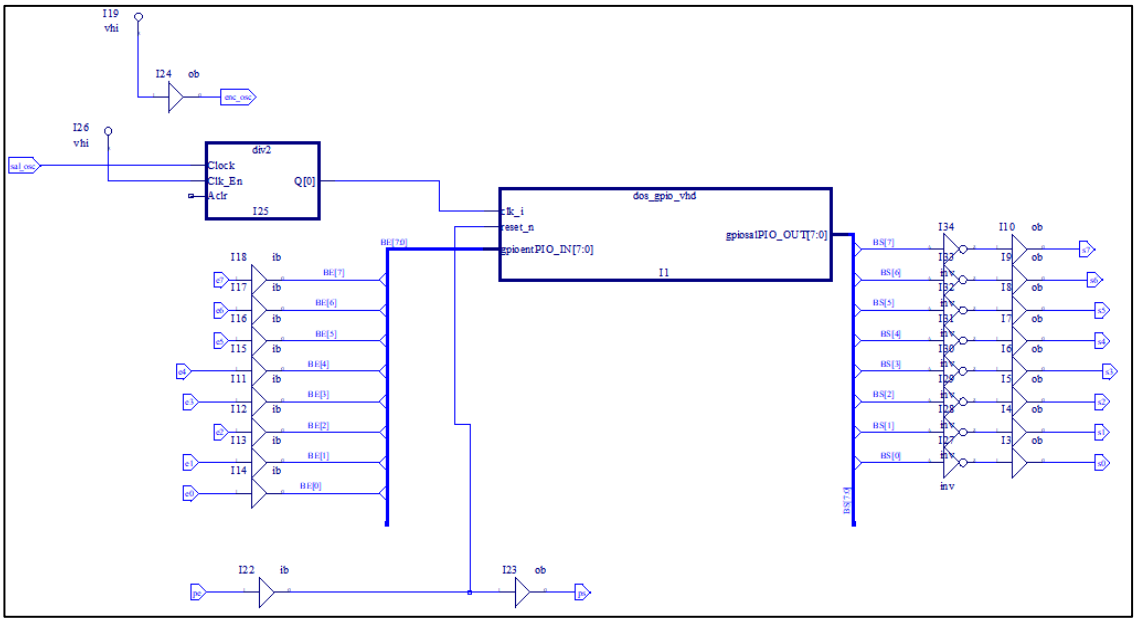


Figura 88 Esquema Microcontrolador con Inversores

Una vez se ha realizado este cambio, se genera el archivo Jedec, y se configura la FPGA, observando el comportamiento. El software cargado en el microprocesador será el del diseño 2, ya que no se ha modificado.

Se accede a Lattice Mico 8 y se abre el MSB alojado en la carpeta:

File > Open Platform…

C:\TFG\exp4\dos_gpio\soc\dos_gpio.msb

Que es la referente al diseño 4.

Se accede a C/C++ perspective

Y se crea un nuevo proyecto del tipo Mico Managed Make C Project.

Lo creo de tipo blank (vacío) y se añade un archivo C, como ya se explicó en el diseño 2.

Nota:

Conjuntamente a este diseño se ha creado,3.4.4 Localización de archivos en Lattice MicoSystem y su manipulación*, una pequeña guía de como modificar el software.*

Una vez realizados estos pasos comienzo a modificar el software tal y como se explica en: *3.4.4* Localización de archivos en Lattice MicoSystem y su manipulación.

El programa final generado (tras haber compilado y experimentado varias veces con el código) es el siguiente:

cod3.c

```
#include "MicoUtils.h"
#include "MicoGPIO.h"
#include "DDStructs.h"
int main(void)
{
unsigned char valor_leds = 0xAA;//170 en hexa
/*unsigned char *test = 0;
unsigned int *x = test;*/
unsigned int ocho=8;
unsigned int quince=15;
unsigned int suma;
suma = valor_leds + quince + ocho; //170 +15 +8
unsigned int numbus;
unsigned int i;

/* Fetch GPIO instance named "gpioent" */
MicoGPIOCtx_t *pos_gpio_ent = &gpio_gpioent;
if(pos_gpio_ent == 0){
return(0);
}
/* Fetch GPIO instance named "gpiosal" */
MicoGPIOCtx_t *pos_gpio_sal = &gpio_gpiosal;
if(pos_gpio_sal == 0){
return(0);
}
/* Escribir 0xaa en los leds y esperar 2 segundos */
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_sal->base, suma);
MicoSleepMilliSecs(20000);
while(1)
{
MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_ent->base,numbus);
if (numbus!=0){
/* scroll the LEDs, every 100 msecs forever */
for(i=1;i <= numbus;i++)
{
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_sal->base, i);
MicoSleepMilliSecs(5000);
if (i==numbus)
{
numbus = 0;
i=0;
}
}
MicoSleepMilliSecs(500);
}
}
return(0);
}
```

El código C introducido funciona correctamente. Es el resultado de probar varias instrucciones que explicare a continuación.

En “principio” se pueden utilizar todas las instrucciones y funciones del lenguaje C.

A mayores, se comprueban las propias de los componentes introducidos en el microcontrolador, como:

Propias del Procesador:

Esperar 20000 milisegundos.

```
MicoSleepMilliSecs(20000);
```

Las que son propias de las GPIO

Escribe en la GPIO

```
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_sal->base, suma);
```

Leer de GPIO

```
MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_ent->base,numbus);
```

El tipo de variable Base hace referencia a la dirección de memoria en la que está alojada la variable dentro de la FPGA, en este caso elemento GPIO.

Se deberán respetar los siguientes nombres de las declaraciones de las GPIO, ya que se han nombrado así en la plataforma MSB.

```
MicoGPIOCtx_t *pos_gpio_ent = &gpio_gpioent;
MicoGPIOCtx_t *pos_gpio_sal = &gpio_gpiosal;
```

A parte de estas instrucciones y las respectivas declaraciones de las GPIO, el resto son funciones e instrucciones de lenguaje C, como por ejemplo las que se han utilizado: FOR, WHILE, IF... Se comprueba que funcionan correctamente.

Se ha probado el uso de las variables en decimal (dentro del código), ya que en toda la documentación existente se utiliza el hexadecimal o variables de tipo “chars”(caractér), que hace más complejo el uso del controlador. Se incorporan pues variables de tipo “int”(entero).

Se puede usar los números en decimal para operaciones internas o incluso con las GPIO, si se introducen los números en decimal en el bus GPIO aparecerán en Binario y viceversa

De tal forma que cuando se envia 197 a la salida de la GPIO, en la salida del símbolo (y por lo tanto de las patillas) tendré 1100 1010, y lo mismo sucede con el resto de números.

Cuando se lee la variable “numbus”, la guardo en un número entero y opero con el mismo, aunque en realidad el microcontrolador la haya leído en binario (ya que la entrada está conectada a pulsadores).

Se ha comprobado experimentalmente este comportamiento.

También cabe destacar que, si el reloj utilizado es adecuado, los tiempos de funcionamiento son muy exactos.

En el diseño actual se ha intentado introducir código C que requiere elementos “sofisticados”, para un microprocesador de 8 bits, como pueden ser la multiplicación o la división.

El resultado obtenido:

El programa permite guardar el código y no hay mensaje de error, pero a la hora de compilar aparecen errores, que impiden cargar el software en la Memoria PROM del microprocesador.

Se llega a la conclusión de que un procesador de 8 bits, como el que se está usando (si se revisa la documentación del LM8, o el juego de instrucciones) no puede desempeñar estas tareas de forma directa.

Se podría crear un algoritmo con sumas, restas y desplazamientos que desempeñen estas funciones.

Por ejemplo, la función de multiplicación sería:

```
int multiplicacion (int factor_1 , int factor_2)
{
int resultado_mult = 0;
int i=0;
for( i ; i <= factor_2; i++ )
{
resultado_mult=resultado_mult + factor_1;
}
return(resultado_mult);
}
```

Hay que tener en cuenta que dicho valor no puede exceder de más de 255, se explicará más adelante el motivo.

Tampoco será posible introducir librerías de C, ya que gran parte de ellas usan instrucciones que el microcontrolador no posee.

Lattice MicoSystem también permite implementar microcontroladores de 32 bits a medida, que tienen un mayor juego de instrucciones, e incluyen multiplicador, divisor, memoria cache, depurador de código (Debug)...Sin embargo su implementación es bastante compleja y consume muchos recursos (se analizará en el apartado 4.4 Comparativa de recursos entre LM8 y LM32).

Esto solventaría el problema de la “facilidad” de manejar el software en C, permitiendo utilizar librerías más complejas y desarrollar programas con cálculos costosos que un controlador de 8 bits no puede desempeñar fácilmente. Sin embargo, el proyecto no se centrará en esta parte, ya que la implementación de un microcontrolador de 32 bits no es válida para muchas

FPGAs (por tamaño del diseño, periféricos y bloques específicos, que debe contener la FPGA). Además, las FPGAs adecuadas para estos controladores tienen precios más elevados.

Por este motivo en el siguiente diseño se utilizará (como hasta el momento) un microcontrolador de 8 bits (con el LM8).

3.4.4 Localización de archivos en Lattice MicoSystem y su manipulación.

3.4.4.1 Introducción sobre la organización de Lattice Diamond y MicoSystem.

Lo primero, diferenciar Proyecto de Plataforma.

Lattice Diamond trabaja con Proyectos, Lattice Mico trabaja con Plataformas. No deben confundirse los términos.

Cuando queremos crear una Plataforma, es necesario crear un proyecto el Lattice Diamond con el mismo nombre de la plataforma a crear.

La plataforma, se guardará en la carpeta de implementación del Proyecto de Lattice Diamond.

Ya que cuando se crea un proyecto de Lattice Diamond, se genera:

-Archivo con extensión ".ldf" y una carpeta de implementación con el nombre que nosotros elegiremos, para mayor simplicidad se utiliza el mismo nombre del proyecto.

Ejemplo:

...\carpeta_de_proyectos

Donde estarán:

- El archivo Proyecto:

  ...\carpeta_de_protectos\proyecto1.ldf

- Carpeta de implementación (con el mismo nombre del proyecto):

  ...\carpeta_de_proyectos\proyecto1\

Cuando se crea una plataforma en Lattice MicoSystem, lo llamamos con el nombre del Proyecto y lo guardamos en la carpeta donde está dicho proyecto.

Hay que tener la precaución de no guardarlo en la carpeta de implementación.

Nombre: proyecto1

Se guarda en …\carpeta_de_proyectos

Aunque seleccionemos esta carpeta, los archivos asociados a Lattice mico se guardan automáticamente dentro de la carpeta de implementación, en una carpeta llamada \soc

…\carpeta_de_proyectos\proyecto1\soc\

En ella se guardarán los archivos". v", ".vhd" y ".msb" (que son los correspondientes al "hardware" del microcontrolador).

Una vez entendido esto, falta explicar donde se guarda el Código C o la aplicación a cargar en el microcontrolador.

3.4.4.2 Introducir Software en el microcontrolador:

Cuando estamos en la perspectiva C/C++ perspective hay que crear otro Proyecto, pero este es de otro tipo:

Se trata de un Mico Managed C progect.

Cuando se genera este, se crea una carpeta con el nombre del proyecto, dentro se guardarán los archivos del mismo. Esta carpeta debe crearse dentro de la carpeta de implementación.

Si el proyecto Mico Managed C progect se llama "código", por ejemplo, se creará la ruta:

…\carpeta_de_proyectos\proyecto1\codigo\

Y en esta carpeta, se generarán los archivos propios de este apartado.

Por último, quedan los archivos de Mico8 para introducirlos en el microcontrolador.

Estos deberemos generarlos cada vez que se modifique el código C de la aplicación. Son los que se utilizan para el Scrachpad y para la memoria PROM del Microcontrolador.

Los guardaremos siempre dentro de la carpeta del proyecto Mico Managed C progect.

…\carpeta_de_proyectos\proyecto1\codigo\

3.4.4.3 ¿Cómo mover o Copiar archivos con Lattice Mico8?

Una vez localizados todos los archivos, dentro de las carpetas, podremos mover los archivos siempre que se respete la arquitectura anteriormente explicada. Y a parte de los archivos mencionados, que son los que voy a utilizar, hay muchos otros que deben permanecer ordenados.

Para mover el proyecto y plataforma conjuntamente, copio todos los archivos que se encuentren dentro de la carpeta raíz donde he creado todo. En los ejemplos anteriores equivale a:

…\carpeta_de_protectos\

Y lo pego todo en una nueva carpeta:

…\nueva_carpeta_de_proyectos\

Al hacer esto, si abro los proyectos en Lattice Diamond y colocamos el cursor encima, vemos que las ubicaciones de los archivos se han actualizado.

Sin embargo, en el programa de Lattice MicoSystem, los archivos ".mem" siguen vinculados al proyecto anterior.

Es necesario crear un nuevo Mico Managed C progect con un nuevo archivo C el cual esté todo referenciado a la nueva carpeta, quedando el proyecto nuevo, ubicado en:

…\nueva_carpeta_de_proyectos\proyecto1\codigo\

Para actualizar el símbolo del esquema global del Microcontrolador (que funcione con el último programa cargado) y que este funcione con el código localizado dentro de su directorio, se realizarán los siguientes pasos:

3.4.4.4 Modificar la aplicación del Microprocesador

Siempre que modifico el código C (software del Microcontrolador) o creo un proyecto nuevo tengo que realizar las siguientes operaciones:

**Punto de referencia 1*

Voy a Lattice MicoSystem (doy por hecho que el diseño MSB está Creado y Operativo).

Entro en la Perspectiva C/C++ perspective.

Si no hay proyecto creo uno nuevo de tipo blank y se añade el archivo ".c" con el programa, se guarda en el directorio:

…\nueva_carpeta_de_proyectos\proyecto1\codigo\

Una vez hayamos creado el proyecto o modifiquemos uno anteriormente creado se realizarán los siguientes pasos.

Guardar (para que se apliquen todos los cambios, si no da error al compilar).

Project > Build Project.

Cuando acabe de compilar arreglamos los errores y warnings del código.
Y compilamos hasta que no haya errores si es necesario.

Cuando compile, hacemos clic en Tools > Software Deployment…

Se abre la ventana:

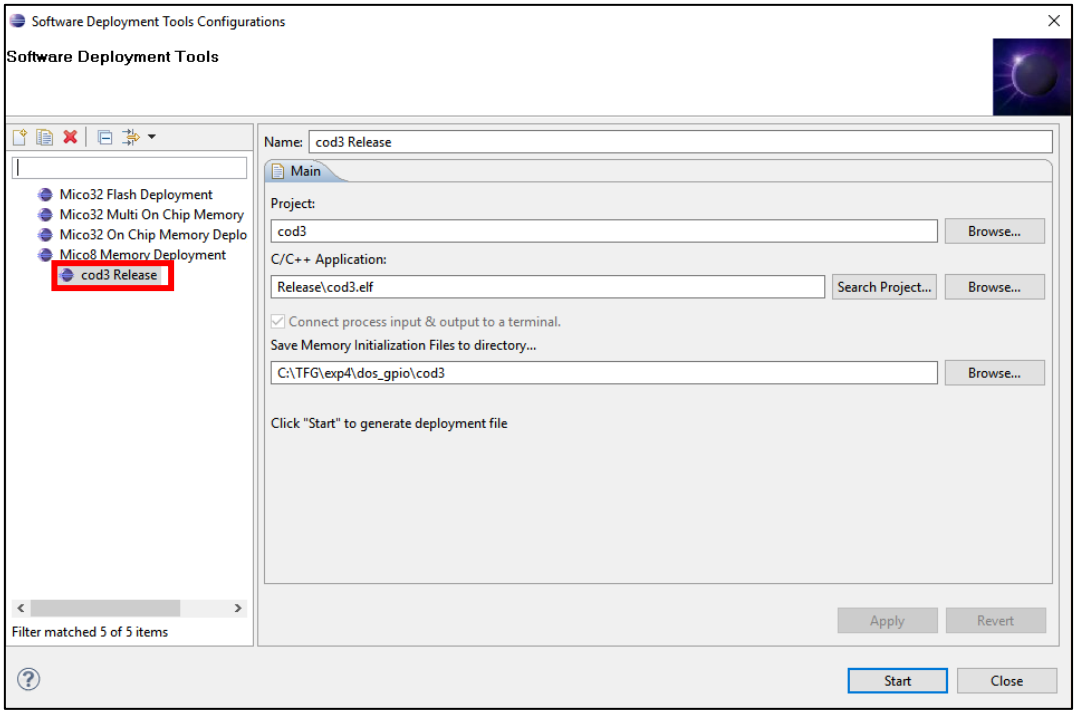


Figura 89 Eliminar proyectos Deployment anteriores.

En el caso de que bajo Mico8 Memory Deployment exista algún archivo como el marcado en rojo (existirá si el proyecto no es nuevo), deberemos eliminarlo

Figura 90 Crear Nueva Memory Deplyment

Para ello clic derecho y Delete.

Creamos uno nuevo con clic derecho, New, y lo referenciamos a la carpeta y el proyecto adecuado (esto es importante para que se salven los cambios).

La carpeta adecuada será:

…\nueva_carpeta_de_proyectos\proyecto1\codigo\

Una vez acabado hay que hacer clic en Aply y Start en ese orden de esta misma ventana.

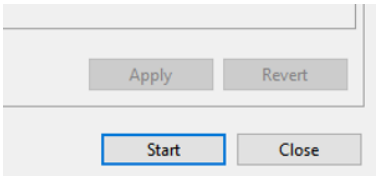


Figura 91 Botones de Aply y Start de La ventana anterior

Esperamos unos segundos mientras aparecen cambios en la consola del programa.

Volvemos a la perspectiva MSB, y ahora si podemos cargar el programa actualizado en el microprocesador LM8. Se repiten los pasos de creación del microcontrolador tras haber realizado la última operación.

Consiste en colocar los nuevos archivos “.mem” generados, en el LM8, que se encuentran dentro de la carpeta:

…\nueva_carpeta_de_proyectos\proyecto1\codigo\

Para ello se hace doble clic en el LM8. Y se configura como acabo de explicar.

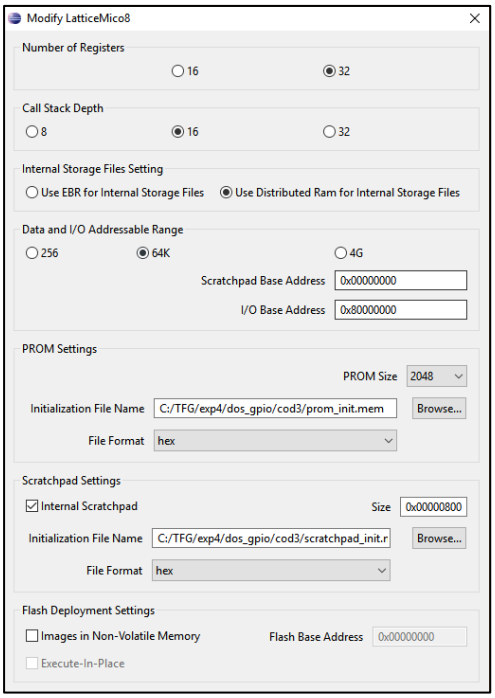


Figura 92 Reconfigurar Microprocesador LM8

Tras esto hay que hacer clic en Run Generator y ya se salvarían todos los cambios automáticamente.

Platform Tools > Run Generator.

Cada vez que modifique el código C del Mico Managed C progect deberé realizar los pasos desde*Punto de referencia 1, hasta aquí.

3.5 Diseño 5: Diseño final con modos de Funcionamiento.

En este diseño se tratará de integrar un software "más complejo" en el microcontrolador a implementar.

Se creará un nuevo microcontrolador en el cual se le modifican las entradas y salidas, demostrando que es posible diseñar el controlador como el usuario decida, siempre dentro de unas limitaciones.

Se explicará cómo implementar un diseño que contenga GPIOs con un bus de datos superior a los 8 bits permitidos.

Se introducirá un software con varios modos de funcionamiento, y se utilizará una función.

El diseño seguirá el siguiente comportamiento:

El sensor de ultrasonidos será la fuente de información del mundo físico, esta información entra en la FPGA en forma de señal digital (también existe una señal de control unida a los SWICH que van directamente al microcontrolador), después existe una lógica descrita en VHDL y mediante esquemas dentro de la FPGA, que se encarga de adaptar la información para el microcontrolador.

Este manipula los datos según un algoritmo establecido en el código, con los distintos modos de funcionamiento. Tras el procesamiento de la información, se envía a la lógica de la FPGA descrita en VHDL y esquemas, donde se adapta para ser visualizada.

Se puede sintetizar con la siguiente imagen:

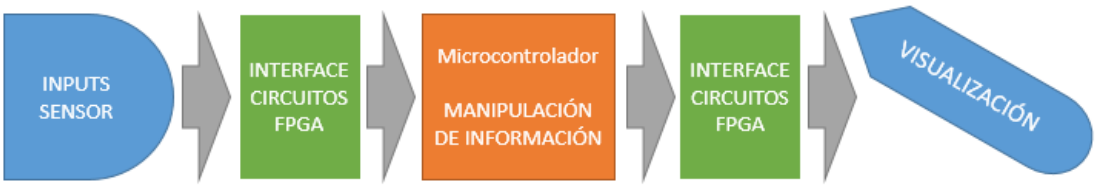


Figura 93 Síntesis del funcionamiento del diseño 5

3.5.1 Objetivo del diseño.

Crear un diseño que contenga 12 entradas y doce salidas, de tal manera que, la información del contador en cascada (de 0 a 999), tras pasar unos

registros (para que las señales que entran en el microcontrolador se mantengan estables 1 segundo) entre en el microcontrolador en binario, y este se encargará de tratar los datos y pilotar las salidas, adecuadamente.

Esto supondrá tener un controlador cuyo "diseño físico o hardware" es creado a medida, y el software a integrar es manipulado hasta el nivel que se puede llegar con este microcontrolador de 8 bits.

3.5.2 Material Utilizado.

Material utilizado para el diseño:

Software:

Lattice Diamond 3.8.

LatticeMico System.

Hardware:

FPGA MACH2OX-1200ZE junto con su breakout board.

Placa de extensión con botones y SWICHs.

Placa de extensión con dos displays.

Sensor de Ultrasonidos LV-MaxSonarEZ3.

3.5.3 Desarrollo.

Se genera un nuevo proyecto con un nuevo microcontrolador siguiendo los pasos del diseño 3.

3.5.3.1 Creacion del Hardware del microcontrolador.

A este diseño se le introducirá un microcontrolador con 3 GPIOs de entrada y tres GPIOs de salida, cada una de ellas con 4 bits.

Nota Importante:

Para poder introducir los 12 bits de entrada y salida, es necesario hacerlo de esta manera (varios buses de I/O de no más de 8 bits), y no introduciendo dos GPIOs de 12, ya que, aunque Lattice MicoSystem y Lattice Diamond no reporten error, los 4 bits más significativos en buses de 12 no funcionan.

Al configurar la FPGA (con buses de 12 bits) cuando la información pasa por el procesador LM8 las variables son "recortadas" hasta los 8 bits, siendo:

Bin: 1111 1111

Dec: 255

Hex: 0x FF

Los valores máximos que se pueden introducir en el procesador LM8 de forma directa.

Luego:

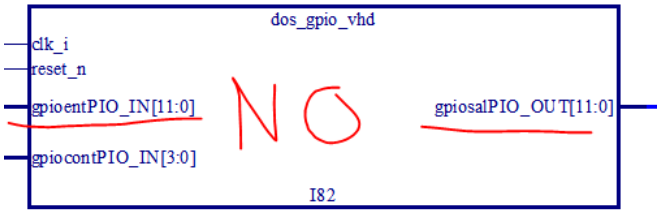


Figura 94 Microprocesador NO válido

En este caso sería imprescindible usar el procesador LM32 que es más complejo y no se contempla en el proyecto.

En la perspectiva MSB se introducen 3 GPIO de entrada y otras 3 de salida, como en la siguiente imagen:

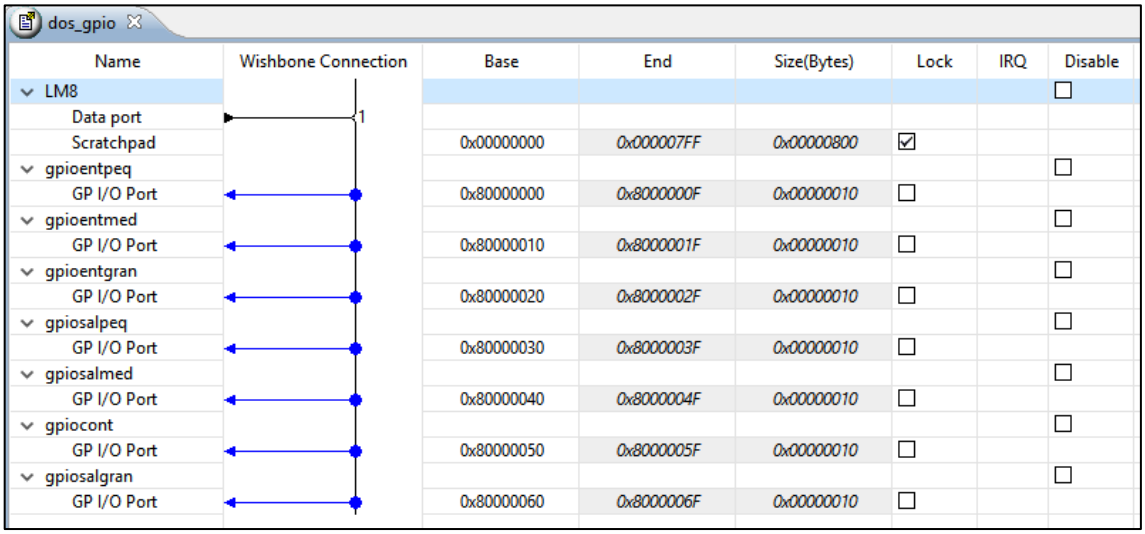


Figura 95 Nueva configuración con 12 PIOs

Se ha mantenido el nombre “dos_gpio” con el objetivo de mantener el orden de las carpetas y archivos como ya se explicó en los diseños anteriores.

De tal manera que para generar el símbolo en el esquema se seguirán los pasos del diseño 3.

El código C a introducir, se ha basado en el del diseño 3 y después se ha modificado.

Al generar el símbolo, queda de la siguiente manera:

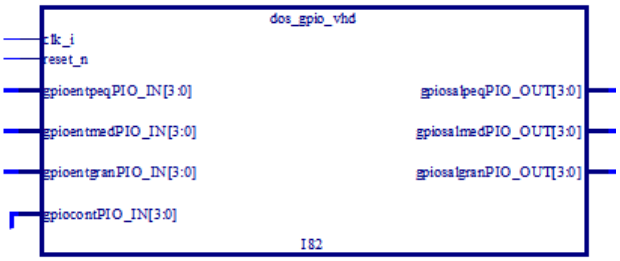


Figura 96 Símbolo Microcontrolador con 12 PIOs

Cada una de las PIO de entrada se conectará a los distintos dígitos procedentes de los registros que vienen de la salida de los contadores, se han llamado “entGran”, “entMed” y “entPeq” representan los nombres asociados a las centenas, decenas y unidades respectivamente.

El otro bus de PIOs de entrada, es la señal de control que se une a dos SWITCH. Este bus de entrada me permitirá elegir el modo de funcionamiento del microcontrolador.

Los modos de funcionamiento se han codificado, con el objetivo de aprovechar los Switch

- 00 → Funcionamiento normal (muestra mediadas en tiempo real).
- 01 → Intervalos, muestra en los visualizadores si se está en un intervalo o no.
- 10 → Máximo, muestra el valor máximo medido desde que se activó el modo.
- 11 → Mínimo, muestra el valor mínimo medido desde que se activó el modo.

Esta codificación se aplica a los SW7 y SW8 de la placa de expansión de entradas.

Además, en este caso, se unirá el puerto de reset con un SWITCH exterior, para poder resetear el microcontrolador en cualquier momento.

Las salidas del microcontrolador se conectarán a los distintos decodificadores que se dirigen a los displays de salida, se han llamado “salGran”, “salMed” y “salPeq” representan los nombres asociados a las centenas, decenas y unidades respectivamente.

Además, se añaden las siguientes modificaciones al esquema:

3.5.3.2 Modificación Bloque PLL

Se modifica el bloque PLL, por PLL2, el objetivo es aumentar su precisión, y evitar el contador que reducía la frecuencia de 50MHz a 1 MHz, de esta forma se reducen los errores.

La configuración de PLL2 es la siguiente:

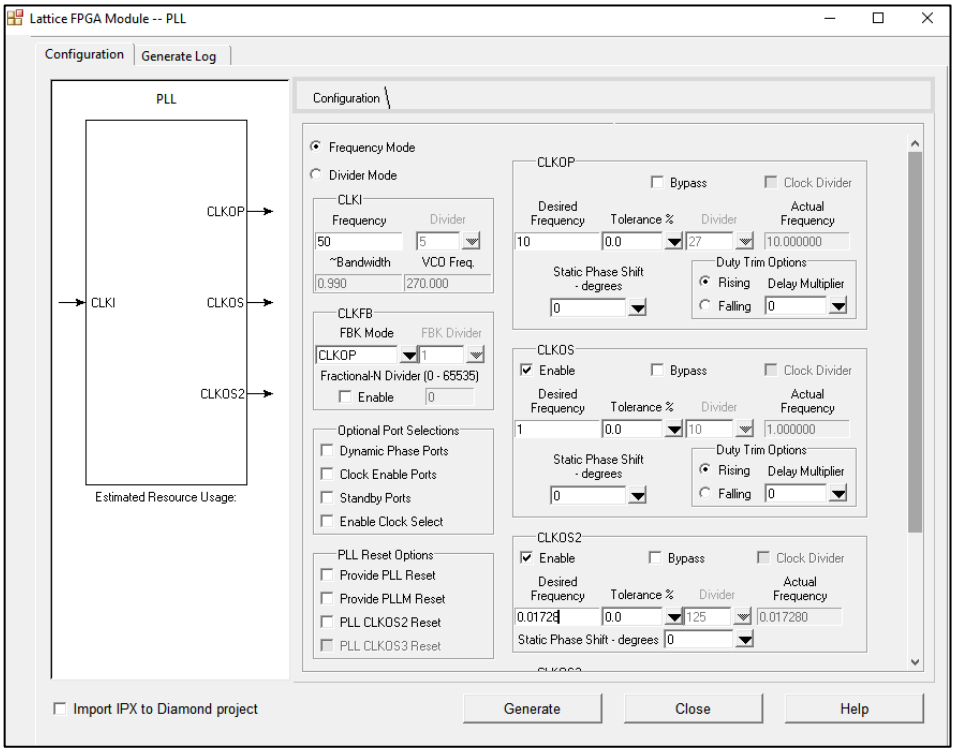


Figura 97 Configuración PLL2

Tras presionar el "botón" calcular (igual que se realizó en el diseño 1) se observan los valores de la frecuencia actual, que es la que genera el bloque PLL, esta frecuencia, es mucho más exacta y además me permite obtener como salida dos frecuencias necesarias para el diseño.

La señal de 17.28KHz se sigue uniendo al reloj de los contadores de la medición.

Se conecta directamente la señal de 1MHz a la entrada de reloj del bloque que genera los pulsos de 2µs y 32µs (se podrán observar todos los cambios en el esquema final).

3.5.3.3 Modificación Codificadores

Se modifica el código VHDL de los decodificadores, con el objetivo de poder mostrar por pantalla las palabras "si" y "no" para el modo 1 de

funcionamiento que indicará si se encuentran dentro de los intervalos, y se incluye la letra “E” para casos de error.

Solo se modifica la parte correspondiente a la arquitectura, quedando:

.....

```
8 architecture DEC_DISP_arch of DEC_DISP is
9 BEGIN
10
11 WITH BCD SELECT
12
13 SEGMENTOS <=
14 "0111111" when "0000", --0 o O
15 "0000110" when "0001", --1
16 "1011011" when "0010", --2
17 "1001111" when "0011", --3
18 "1100110" when "0100", --4
19 "1101101" when "0101", --5 o S
20 "1111101" when "0110", --6
21 "0000111" when "0111", --7
22 "1111111" when "1000", --8
23 "1101111" when "1001", --9
24 "0110000" when "1010", --10 saca I
25 "0110111" when "1011", --11 saca N
26 "1111001" when others; --E de error
27 end DEC_DISP_arch;
```

.....

A continuación, se simula el comportamiento del convertidor BCD a 7 segmentos y se muestra el cronograma. En esta simulación se puede apreciar que el funcionamiento es adecuado.
Para realizarlo se ha utilizado las herramientas de Simulator Wizard y Active HDL simulator.

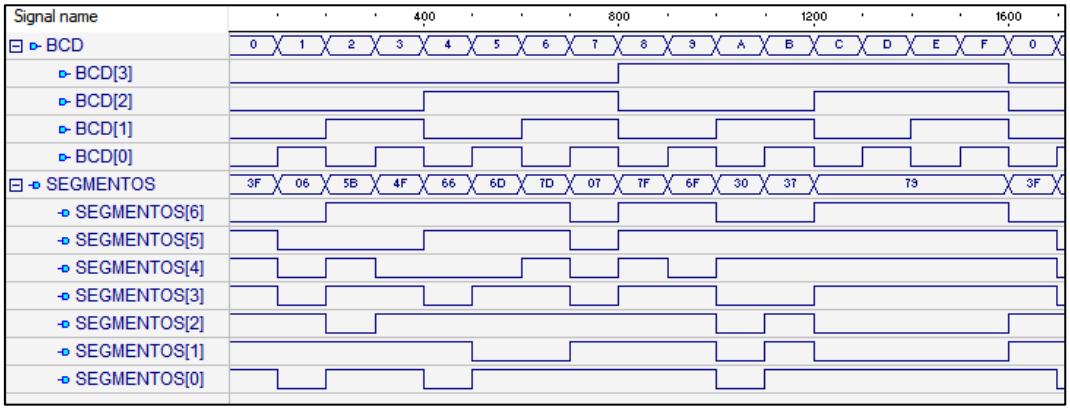


Figura 98 Cronogram: Decodificador BCD a 7 segmentos

3.5.3.4 Modificación del Software

Por último, se introducirá software al diseño.
Se incluye una función propia del lenguaje C, que se emplea como solución a

la falta de multiplicación del µC, para unir la información de los buses de entrada.

La información que se utiliza en este código se basa en los dos buses grandes (gran y med) tanto de entradas como de salidas, despreciándose la información de las unidades, ya que, por motivos de material, no se dispone de un tercer visualizador. Además, el rango de valores de una variable tipo "unsigned int" en un bus de 8 bits, es de 0 a 255, que implicaría un software más sofisticado para trabajar con 3 cifras.

El código empleado para este diseño (utilizando las metodologías de los diseños anteriores) es el siguiente:

```
#include "MicoUtils.h"
#include "MicoGPIO.h"
#include "DDStructs.h"
int calc_punto (int ,int,int);
int main(void)
{
/*-----------------Declaraciones variables----------------------------*/
unsigned char valor_leds = 0xAA;//170 en hexa
/*unsigned char *test = 0;
unsigned int *x = test;*/
unsigned char pos_interr;
unsigned int modo;
unsigned int gran = 0;
unsigned int med = 0;
unsigned int punto = 0;
unsigned int gran2 = 0;
unsigned int med2 = 0;
unsigned int puntomem = 0;
unsigned int puntomem2 = 99;
/*--------------- Declaraciones de GPIOs ------------------*/
/* Fetch GPIO instance named "gpioentpeq" */
MicoGPIOCtx_t *pos_gpio_entpeq = &gpio_gpioentpeq;
if(pos_gpio_entpeq == 0){
return(0);
}
/* Fetch GPIO instance named "gpioentmed" */
MicoGPIOCtx_t *pos_gpio_entmed = &gpio_gpioentmed;
if(pos_gpio_entmed == 0){
return(0);
}
/* Fetch GPIO instance named "gpioentgran" */
MicoGPIOCtx_t *pos_gpio_entgran = &gpio_gpioentgran;
if(pos_gpio_entgran == 0){
return(0);
}
      /* Fetch GPIO instance named "gpiocont" */
      MicoGPIOCtx_t *pos_gpio_cont = &gpio_gpiocont;
      if(pos_gpio_cont == 0){
      return(0);
      }
      /* Fetch GPIO instance named "gpiosalpeq" */
      MicoGPIOCtx_t *pos_gpio_salpeq = &gpio_gpiosalpeq;
      if(pos_gpio_salpeq == 0){
      return(0);
      }
      /* Fetch GPIO instance named "gpiosalmed" */
      MicoGPIOCtx_t *pos_gpio_salmed = &gpio_gpiosalmed;
      if(pos_gpio_salmed == 0){
      return(0);
      }
      /* Fetch GPIO instance named "gpiosalgran" */
      MicoGPIOCtx_t *pos_gpio_salgran = &gpio_gpiosalgran;

      if(pos_gpio_salgran == 0){
```

```
	return(0);
	}
	/*-----------------Fin declaraciones------------------------------*/
	while(1)
	{
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_cont->base, modo);
	MicoSleepMilliSecs(20);
	//Modo 1
	while(modo==0)
	{
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entgran->base, pos_interr);
	valor_leds =pos_interr;
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salgran->base, valor_leds);
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entmed->base, pos_interr);
	valor_leds =pos_interr;
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salmed->base, valor_leds);
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entpeq->base, pos_interr);
	valor_leds =pos_interr;
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salpeq->base, valor_leds);
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_cont->base, modo);
	MicoSleepMilliSecs(100);
	}
	//Modo 2
	while(modo==1)
	{
	//dirá si esta entre uno y tres metros
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entgran->base, gran);
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entmed->base, med);
	punto=calc_punto ( gran , med, punto);
	if((punto > 10)&(punto < 30))
	{
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salgran->base, 5);
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salmed->base, 10);
	}
	else
	{
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salgran->base, 11);
	MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salmed->base, 0);
	}
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_cont->base, modo);
	}
	gran=0;
	med=0;

	//Modo 3
	while(modo==2)
	{
	//saca el valor maximo medido
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entgran->base, gran);
	MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entmed->base, med);

	punto=1;
	punto=calc_punto ( gran , med, punto);
	if(punto > puntomem)
	{
gran2 =gran;
med2=med;
MicoSleepMilliSecs(5000);
puntomem = punto;
}
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salgran->base, gran2);
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salmed->base, med2);
/* else
{
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salgran->base, 13);//Error
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salmed->base, 13);//Error
}
*/
MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_cont->base, modo);
MicoSleepMilliSecs(100);
}
gran2=0;
med2=0;
// Modo 4
```

```
while(modo==3)/*valor minimo*/
{
MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entgran->base, gran);
MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_entmed->base, med);
punto=99;
punto=calc_punto ( gran , med, punto);
if(punto < puntomem2)
{
gran2 =gran;
med2=med;
MicoSleepMilliSecs(5000);
puntomem2 = punto;
}
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salgran->base, gran2);
MICO_GPIO_WRITE_DATA_BYTE0 (pos_gpio_salmed->base, med2);

MICO_GPIO_READ_DATA_BYTE0 (pos_gpio_cont->base, modo);
MicoSleepMilliSecs(100);
}
gran2=0;
med2=0;
}
}
/*-------funciones---------*/
int calc_punto (int gran , int med ,int punto)
{
int dec=0;
dec = gran+gran+gran+gran+gran+gran+gran+gran+gran+gran;
punto = dec + med ;
return(punto);
}
```

El esquema incluyendo todas las mejoras mencionadas, tanto en este diseño como en los anteriores se puede observar a continuación:

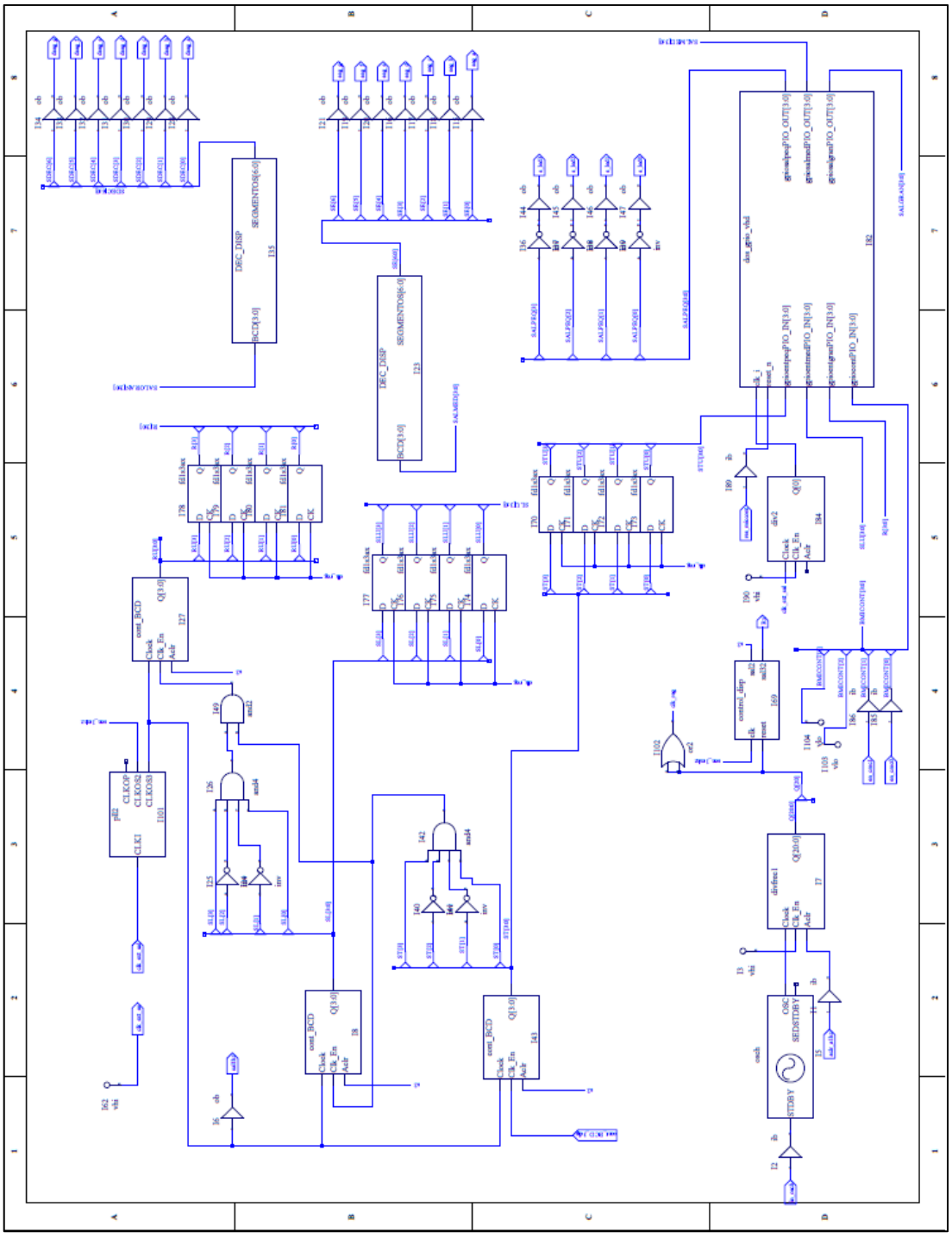


Figura 99 Esquema Diseño 5

Una vez añadido, se realizan todos los pasos hasta configurar la FPGA y observar los resultados.

Se puede comprobar que todos los modos funcionan satisfactoriamente, y que el comportamiento es el esperado.

4 Análisis de recursos.

Se dispone a analizar el consumo de recursos de los distintos diseños creados en este proyecto, en los dos dispositivos FPGA de la familia MACHXO2 disponibles en el departamento de Tecnología Electrónica de la Escuela de Ingenierías Industriales.

En estos dos dispositivos se implementarán los diseños y se configurarán.

Se realizará un tercer análisis en la FPGA en el dispositivo FPGA MACHXO2-640U, la cual no está disponible en el departamento, pero si puedo crear el fichero de configuración del dispositivo donde analizar los recursos utilizados.

Estos dispositivos poseen tamaños distintos, y contienen distintos bloques funcionales específicos embebidos.

Se pueden observar los recursos disponibles de los dispositivos en la figura 100, obtenida del Datasheet de la familia de MACHXO2:

Table 1-1. MachXO2™ Family Selection Guide

		XO2-256	XO2-640	XO2-640U[1]	XO2-1200	XO2-1200U[1]	XO2-2000	XO2-2000U[1]	XO2-4000	XO2-7000
LUTs		256	640	640	1280	1280	2112	2112	4320	6864
Distributed RAM (kbits)		2	5	5	10	10	16	16	34	54
EBR SRAM (kbits)		0	18	64	64	74	74	92	92	240
Number of EBR SRAM Blocks (9 kbits/block)		0	2	7	7	8	8	10	10	26
UFM (kbits)		0	24	64	64	80	80	96	96	256
Device Options:	HC[2]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HE[3]						Yes	Yes	Yes	Yes
	ZE[4]	Yes	Yes		Yes		Yes		Yes	Yes
Number of PLLs		0	0	1	1	1	1	2	2	2
Hardened Functions:	I2C	2	2	2	2	2	2	2	2	2
	SPI	1	1	1	1	1	1	1	1	1
	Timer/Counter	1	1	1	1	1	1	1	1	1

Figura 100 Características de las FPGAs MachXO2

En la imagen anterior se han remarcado los principales elementos a tener en cuenta en el análisis para los distintos dispositivos.

El objetivo del análisis es, determinar si la implementación de un microcontrolador de 8 bits, con el procesador LM8, conlleva el consumo de gran parte del dispositivo.

Esto implicaría que, al utilizar la lógica como microcontrolador, no podré destinarla a otras funciones, desaprovechando el dispositivo.

Se mantendrá el empaquetado de 144 pines llamado TFP144 en todos los diseños para no modificar las entradas y salidas al dispositivo.

Se analizarán solo 3 diseños, debido a que, al resto, se pueden extrapolar la información analizada.

Ademas, al final, se realiza una comparativa de recursos entre los microcontroladores de 8 y 32 bits.

Nota: *los datos reflejados en los gráficos estarán expresados en tanto por ciento (%).*

4.1 Análisis de recursos de Diseño 1.

En este diseño no existía microcontrolador implementado en el dispositivo, de tal manera que se hace un análisis de los recursos consumidos solamente por lógica de uso general.

Este diseño contiene el bloque embebido PLL, lo que quiere decir que solo se puede implementar en FPGAs que contengan al menos 1 de estos.

Se utiliza el diseño que contiene los registros incluidos en la mejora (se hará lo mismo en resto de diseños con el fin de ser lo más objetivo posible).

La información que se muestra a continuación, esta seleccionada dentro del resumen de diseño, con la finalidad de mostrar solo los elementos de interés para el análisis.

Información obtenida:

Design Summary MACHXO2-640U Para el diseño 1

```
Number of registers:     72 out of   964 (7%)
      PFU registers:           72 out of   640 (11%)
      PIO registers:            0 out of   324 (0%)
   Number of SLICEs:        88 out of   320 (28%)
      SLICEs as Logic/ROM:     88 out of   320 (28%)
      SLICEs as RAM:            0 out of   240 (0%)
      SLICEs as Carry:         36 out of   320 (11%)
   Number of LUT4s:        176 out of   640 (28%)
      Number used as logic LUTs:        104
      Number used as distributed RAM:     0
      Number used as ripple logic:       72
      Number used as shift registers:     0
   Number of PIO sites used: 25 + 4(JTAG) out of 108 (27%)
   Number of block RAMs:  0 out of 7 (0%)
   Number of PLLs:  1 out of 1 (100%)
```

Design Summary MACHXO2-1200 Para el diseño 1

```
Number of registers:     72 out of  1604 (4%)
      PFU registers:           72 out of  1280 (6%)
      PIO registers:            0 out of   324 (0%)
   Number of SLICEs:        88 out of   640 (14%)
      SLICEs as Logic/ROM:     88 out of   640 (14%)
      SLICEs as RAM:            0 out of   480 (0%)
      SLICEs as Carry:         36 out of   640 (6%)
   Number of LUT4s:        176 out of  1280 (14%)
      Number used as logic LUTs:        104
      Number used as distributed RAM:     0
      Number used as ripple logic:       72
      Number used as shift registers:     0
   Number of PIO sites used: 25 + 4(JTAG) out of 108 (27%)
   Number of block RAMs:  0 out of 7 (0%)
   Number of PLLs:  1 out of 1 (100%)
```

Design Summary MACHXO2-7000 Para el diseño 1

```
Number of registers:     72 out of  7209 (1%)
      PFU registers:           72 out of  6864 (1%)
      PIO registers:            0 out of   345 (0%)
```

```
Number of SLICEs:        88 out of  3432 (3%)
   SLICEs as Logic/ROM:     88 out of  3432 (3%)
   SLICEs as RAM:            0 out of  2574 (0%)
   SLICEs as Carry:         36 out of  3432 (1%)
Number of LUT4s:        176 out of  6864 (3%)
   Number used as logic LUTs:        104
   Number used as distributed RAM:     0
   Number used as ripple logic:       72
   Number used as shift registers:     0
Number of PIO sites used: 25 + 4(JTAG) out of 115 (25%)
Number of block RAMs:  0 out of 26 (0%)
Number of PLLs:  1 out of 2 (50%)
```

De los datos obtenidos, se crea el siguiente gráfico, donde se analiza la información:

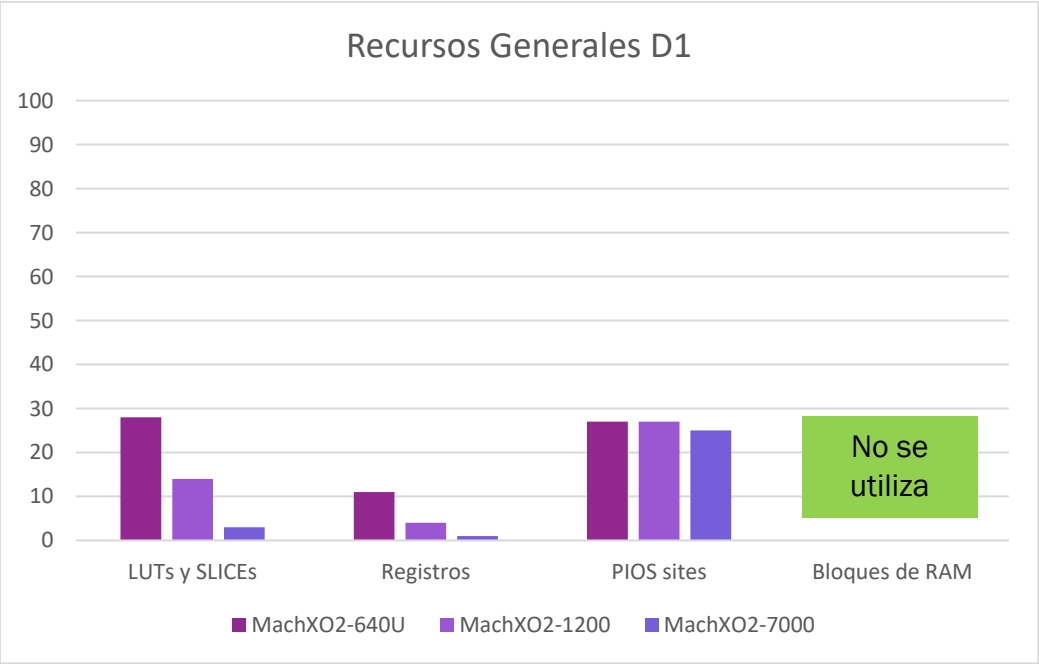


Gráfico 1 Analisis en porcentaje del Diseño 1

Se puede analizar que los recursos consumidos son muy bajos en las tres FPGAs.

El número de PIOS se mantendrá constante, debido a que son las que se utilizan para entradas y salidas.
Se utilizarían muchas más si se emplease más LUTS (el diseño fuese más grande), porque el dispositivo utiliza las PIOS cuando necesita unir Slices y no existe otro camino (también depende del grado de optimización).

La memoria RAM no se utiliza, ya que solo se emplean bloques lógicos y registros (que están dentro de los SLICEs).

Puede apreciarse que las FPGAs grandes, como la MACHXO2-7000, estaría desaprovechada para una aplicación como esta.

4.2 Análisis de recursos de Diseño 4.

En este diseño solo se incluye el microcontrolador, y por lo tanto como mínimo, se requerirán los siguientes elementos para crear un microcontrolador (LM8).

Sera necesario:

- Number of SLICEs: 205 Slices.
  - Number of LUT4s: 410 LUT4s
- Number of block RAMs: 6 bloques.
- Number of registers: 121 registros.

Además, como se puede apreciar en los reportes, no es necesario el bloque PLL, permitiendo a otras familias de FPGAs más simples implantar el microcontrolador.

La información que se muestra a continuación, esta seleccionada dentro del resumen de diseño, con el objetivo de mostrar solo los elementos de interés para el análisis.

Información obtenida:

Design Summary MACHXO2-256 Para el diseño 4

```
   Number of registers:    121 out of   424 (29%)
      PFU registers:          113 out of   256 (44%)
      PIO registers:            8 out of   168 (5%)
   Number of SLICEs:       205 out of   128 (160%)
      SLICEs as Logic/ROM:    169 out of   128 (132%)
      SLICEs as RAM:           36 out of    96 (38%)
      SLICEs as Carry:         17 out of   128 (13%)
   Number of LUT4s:        410 out of   256 (160%)
      Number used as logic LUTs:        304
      Number used as distributed RAM:    72
      Number used as ripple logic:       34
      Number used as shift registers:     0
   Number of PIO sites used: 20 + 4(JTAG) out of 56 (43%)
   Number of block RAMs:  6 out of 0
```

Design Summary MACHXO2-640U Para el diseño 4

```
Number of registers:    121 out of   964 (13%)
      PFU registers:          113 out of   640 (18%)
      PIO registers:            8 out of   324 (2%)
   Number of SLICEs:       205 out of   320 (64%)
      SLICEs as Logic/ROM:    169 out of   320 (53%)
      SLICEs as RAM:           36 out of   240 (15%)
      SLICEs as Carry:         17 out of   320 (5%)
   Number of LUT4s:        410 out of   640 (64%)
      Number used as logic LUTs:        304
      Number used as distributed RAM:    72
      Number used as ripple logic:       34
      Number used as shift registers:     0
   Number of PIO sites used: 20 + 4(JTAG) out of 108 (22%)
   Number of block RAMs:  6 out of 7 (86%)
   Number of PLLs:  0 out of 1 (0%)
```

Design Summary MACHXO2-1200 Para el diseño 4

```
Number of registers:    121 out of  1604 (8%)
      PFU registers:          113 out of  1280 (9%)
      PIO registers:            8 out of   324 (2%)
   Number of SLICEs:       205 out of   640 (32%)
      SLICEs as Logic/ROM:    169 out of   640 (26%)
      SLICEs as RAM:           36 out of   480 (8%)
      SLICEs as Carry:         17 out of   640 (3%)
   Number of LUT4s:        410 out of  1280 (32%)
      Number used as logic LUTs:        304
      Number used as distributed RAM:    72
      Number used as ripple logic:       34
      Number used as shift registers:     0
   Number of PIO sites used: 20 + 4(JTAG) out of 108 (22%)
   Number of block RAMs:  6 out of 7 (86%)
   Number of PLLs:  0 out of 1 (0%)
```

Design Summary MACHXO2-7000 Para el diseño 4

```
Number of registers:    121 out of  7209 (2%)
      PFU registers:          113 out of  6864 (2%)
      PIO registers:            8 out of   345 (2%)
   Number of SLICEs:       205 out of  3432 (6%)
      SLICEs as Logic/ROM:    169 out of  3432 (5%)
      SLICEs as RAM:           36 out of  2574 (1%)
      SLICEs as Carry:         17 out of  3432 (0%)
   Number of LUT4s:        410 out of  6864 (6%)
      Number used as logic LUTs:        304
      Number used as distributed RAM:    72
      Number used as ripple logic:       34
      Number used as shift registers:     0
   Number of PIO sites used: 20 + 4(JTAG) out of 115 (21%)
   Number of block RAMs:  6 out of 26 (23%)
   Number of PLLs:  0 out of 2 (0%)
```

De los datos obtenidos, se crea el siguiente gráfico, donde se analiza la información:

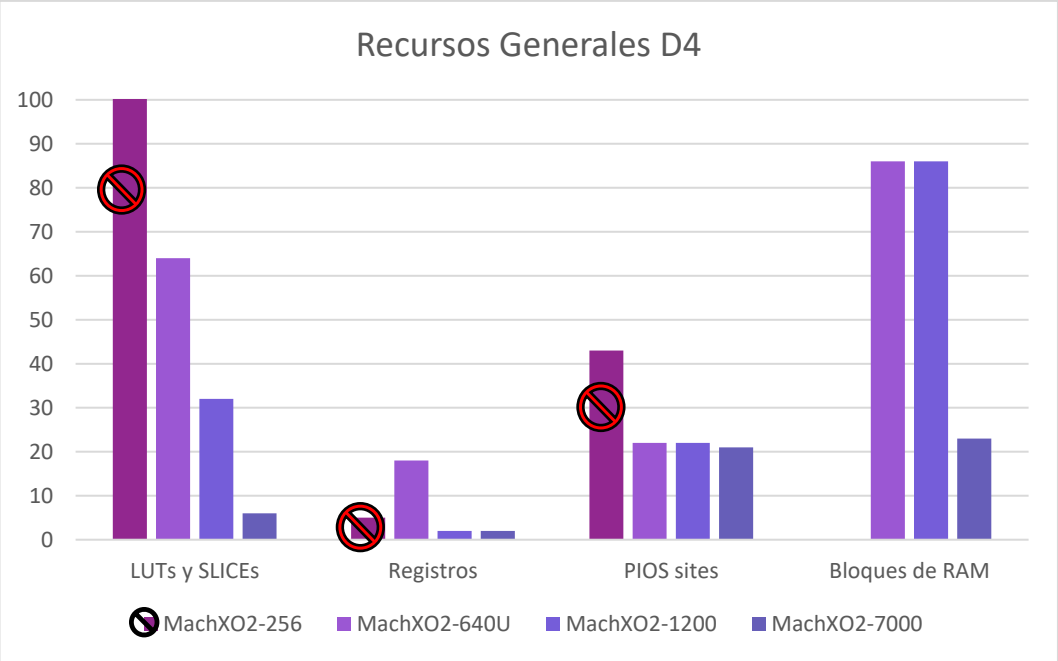


Gráfico 2 Análisis en porcentaje del Diseño 4

Se puede observar que la MachXO2-256, la FPGA más pequeña de la familia MachXO2, pese a no necesitar el bloque PLL, no es apta para implementar el microcontrolador.

Si se analiza la información, el problema es por falta de SLICEs para lógica, quiere decir, que aun reduciendo la memoria PROM asignada al microprocesador (en la configuración del LM8 en el MSB de Lattice MicoSystem), desciende la memoria RAM utilizada del dispositivo, pero no la lógica,y por lo tanto, es imposible implementar el microcontrolador en una FPGA tan pequeña.

En el resto de FPGAs se puede observar que la implantación es viable, resta espacio suficiente para otros elementos y que, además, ahora, se utilizan los bloques de memoria RAM del dispositivo.

4.3 Análisis de recursos de Diseño 5.

La información que se muestra a continuación, esta seleccionada dentro del resumen de diseño, con el objetivo de mostrar solo los elementos de interés para el análisis.

Información obtenida:

Design Summary MACHXO2-640U Para el diseño 5

```
Number of registers:    198 out of   964 (21%)
      PFU registers:          196 out of   640 (31%)
      PIO registers:            2 out of   324 (1%)
   Number of SLICEs:       295 out of   320 (92%)
      SLICEs as Logic/ROM:    259 out of   320 (81%)
      SLICEs as RAM:           36 out of   240 (15%)
      SLICEs as Carry:         49 out of   320 (15%)
   Number of LUT4s:        581 out of   640 (91%)
      Number used as logic LUTs:        411
      Number used as distributed RAM:    72
      Number used as ripple logic:       98
      Number used as shift registers:     0
   Number of PIO sites used: 28 + 4(JTAG) out of 108 (30%)
   Number of block RAMs:  6 out of 7 (86%)
   Number of PLLs:  1 out of 1 (100%)
```

Design Summary MACHXO2-1200 Para el diseño 5

```
   Number of registers:    198 out of  1604 (12%)
      PFU registers:          196 out of  1280 (15%)
      PIO registers:            2 out of   324 (1%)
   Number of SLICEs:       295 out of   640 (46%)
      SLICEs as Logic/ROM:    259 out of   640 (40%)
      SLICEs as RAM:          36 out of   480 (8%)
      SLICEs as Carry:         49 out of   640 (8%)
   Number of LUT4s:        581 out of  1280 (45%)
      Number used as logic LUTs:        411
      Number used as distributed RAM:    72
      Number used as ripple logic:       98
      Number used as shift registers:     0
   Number of PIO sites used: 28 + 4(JTAG) out of 108 (30%)
   Number of block RAMs:  6 out of 7 (86%)
   Number of PLLs:  1 out of 1 (100%)
```

Design Summary MACHXO2-7000 Para el diseño 5

```
   Number of registers:    198 out of  7209 (3%)
      PFU registers:          196 out of  6864 (3%)
      PIO registers:            2 out of   345 (1%)
   Number of SLICEs:       295 out of  3432 (9%)
      SLICEs as Logic/ROM:    259 out of  3432 (8%)
      SLICEs as RAM:           36 out of  2574 (1%)
      SLICEs as Carry:         49 out of  3432 (1%)
   Number of LUT4s:        581 out of  6864 (8%)
      Number used as logic LUTs:        411
      Number used as distributed RAM:    72
      Number used as ripple logic:       98
      Number used as shift registers:     0
   Number of PIO sites used: 28 + 4(JTAG) out of 115 (28%)
   Number of block RAMs:  6 out of 26 (23%)
   Number of PLLs:  1 out of 2 (50%)
```

De los datos obtenidos, se crea el siguiente gráfico, donde se analiza la información:

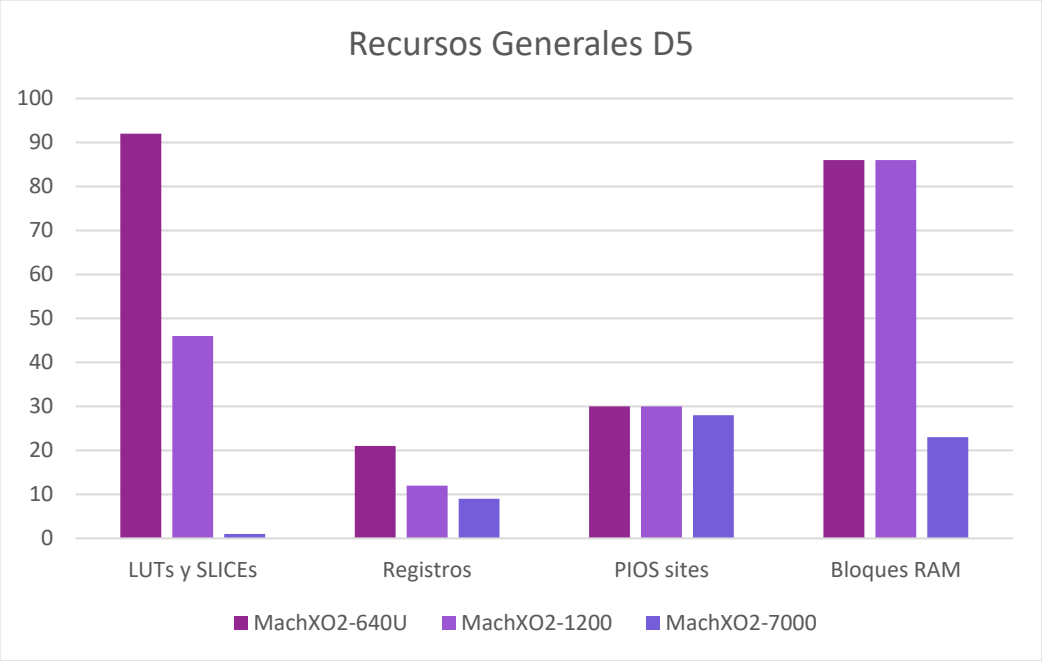


Gráfico 3 Análisis en porcentaje del Diseño 5

De estos datos se pueden analizar varios aspectos, para el microcontrolador implementado,
El diseño es viable en todos los dispositivos utilizados.

En los más pequeños, se utilizan gran parte de los recursos, luego para diseños más complejos se necesitarán FPGAs como la MACHXO2-1200 o superiores.

Se necesitan al menos 6 bloques de memoria RAM, solo para el microprocesador, la lógica adicional no repercute a la memoria RAM, (salvo que se añadan memorias al diseño) solo a las Slices consumidas.

Se observa un ligero aumento en las PIOs, esto es debido a que algunas patillas quedan bloqueadas para ser utilizadas como lógica.

Se puede concluir que los recursos utilizados por el microcontrolador y la lógica asociada al mismo para su integración con otros elementos no es muy elevada.

4.4 Comparativa de recursos entre LM8 y LM32.

En este apartado, se compararán los recursos empleados por los dos microcontroladores que puede implementar el software de Lattice MicoSystem.

El microcontrolador, con el LM32 (microprocesador de 32 bits) no se ha conseguido hacer funcionar en este proyecto. Sin embargo, se ha creado un símbolo con el hardware del mismo.

Puesto que este proyecto se centra en el µC con el LM8, no se explicará más sobre el microcontrolador de 32 bits.

La información de los recursos empleados del microcontrolador de 32, no depende del software introducido (solo del hardware, que si se ha podido implementar).
El microprocesador LM32, incluye elementos que el LM8 no poseía (que son los mínimos que se le puede añadir al LM32, permitiendo usar la mayoría de funciones y librerías del lenguaje C), como, por ejemplo, Multiplicador, Barrel Sifter, Debuger, sets de cache….

También se incluyen memorias, de tipo FLASH y ASRAM completamente necesarias para el funcionamiento.

En cuanto a los periféricos se añaden dos buses de GPIOs de 8 bits cada uno.

Esta implantación se compara con la del diseño 4, en la FPGA MACHXO2-7000, que es en la única FPGA de la familia MACHXO2 donde se puede implementar por capacidad de RAM y Slices.

Información obtenida:

Design Summary del microcntrolador de 32 bits en MACHXO2-7000

```
Number of registers:   1500 out of  7209 (21%)
   PFU registers:         1492 out of  6864 (22%)
   PIO registers:            8 out of   345 (2%)
Number of SLICEs:      1426 out of  3432 (42%)
   SLICEs as Logic/ROM:   1330 out of  3432 (39%)
   SLICEs as RAM:           96 out of  2574 (4%)
   SLICEs as Carry:        113 out of  3432 (3%)
Number of LUT4s:        2829 out of  6864 (41%)
   Number used as logic LUTs:        2411
   Number used as distributed RAM:   192
   Number used as ripple logic:      226
   Number used as shift registers:     0
Number of PIO sites used: 44 + 4(JTAG) out of 115 (42%)
Number of block RAMs:  16 out of 26 (62%)
Number of PLLs:  0 out of 2 (0%)
```

De los datos obtenidos del reporte del microcontrolador de 32 bits y del reporte del diseño 4 con la MACHXO2-7000, se crea el siguiente gráfico, de donde se analizará la información:

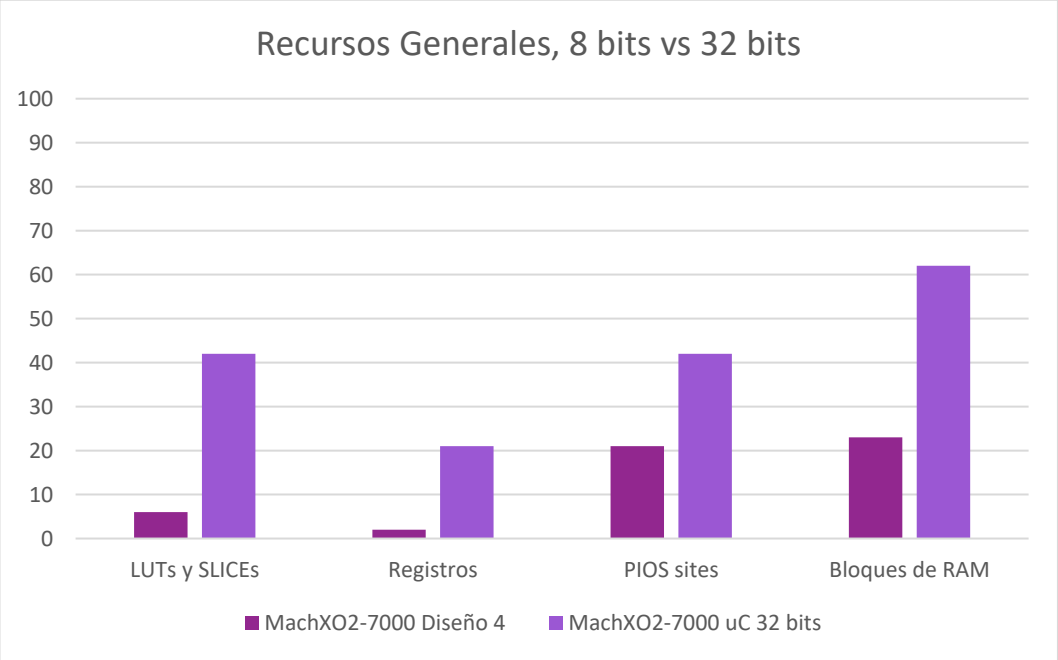


Gráfico 4 Comparativa entre microprocesadores de 8 y 32 bits

Se puede observar que el consumo de recursos del microcontrolador de 32 bits respecto al de 8 bits es muy superior, conlleva emplear gran parte de la FPGA (entre el 40% y el 50%). En cualquier otro dispositivo de la familia resulta imposible su implantación.
Hay un gasto elevado de RAM, Slices, y, se puede observar, que gran parte de las PIOs quedan bloqueadas (aprox. 20%).
Si se aumenta los recursos a utilizar en la perspectiva MSB de Lattice MicoSystem del LM32 (con el fin de mejorar el microcontrolador) puede llegar a no entrar en la FPGA MACHXO2-7000.

5 Ventajas, inconvenientes y aplicaciones.

5.1 Ventajas.

- No será necesario diseñar una PCB que una todos los componentes electrónicos digitales.
  Aunque, si es necesario la PCB para conectar con la alimentación, sensores, elementos de potencia u elementos analógicos.
  Por lo tanto, se simplifican las uniones de elementos y se reduce el tamaño de PCB eliminando pistas.
- Como los elementos están realizados mediante software y no hay pistas (externas) que unan componentes electrónicos, se reduce el consumo y los efectos de ruidos, mejorando la eficiencia y eficacia del producto.
- La implementación de este microcontrolador y la programación en el mismo es relativamente sencilla. Los microcontroladores son muy útiles para tareas en la que haya que realizar acciones de forma secuencial y realizando a mayores ciertos cálculos.
- Permite realizar funciones Multitarea más reales en un solo chip. Ya que la FPGA puede trabajar de forma combinacional y secuencial. Se puede emplear esta ventaja, algo que los Microcontroladores normales no pueden hacer, ya que las acciones más combinacionales que se pueden emplear, es mediante el uso de interrupciones.
  En mi caso usaré el Micro para que trabaje secuencialmente y otra lógica que trabaje de forma paralela combinacionalmente.
- Puedo incluir varios microcontroladores trabajando conjuntamente.
- Permite cargar programas, descargarlos y editarlos de una forma sencilla y rápida.
- Permite realizar simulaciones.
- Puedo editar mi diseño digital siempre que quiera. En un circuito impreso tendría que repetir la placa.

5.2 Inconvenientes.

- Requiere conocer y saber programar en leguaje C (para el programa interno del Microcontrolador).
  Requiere saber realizar los diseños electrónicos (aunque es cierto, que para un microcontrolador normal tenemos el mismo problema).
  Requiere saber programar en VHDL, ya que bloques se construirán describiéndolos en este lenguaje.
- Al ser una tecnología menos utilizada, la ayuda es limitada ya que no existen comunidades grandes que desarrollen software para FPGAs.
- La FPGA tiene un precio más elevado que un microcontrolador junto con circuitería y elementos digitales que irían dentro de la FPGA.
- Es más lento que Microcontroladores de precio elevado y diseñados para que sean rápidos. Aunque, se diseñase la lógica de un procesador de ordenador, este sería más lento que uno comprado igual con el mismo fin.
- Las comunicaciones (sobre todo serie) deberían ser implementadas, ya que, como comunicaciones base, el microcontrolador solo cuenta con comunicación I2C y SPI.

5.3 Aplicaciones.

Cualquier circuito de aplicación específica puede ser implementado en una FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan las FPGA incluyen a los DSP, radio definida por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, etc. Cabe notar que su uso en otras tareas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

Los microcontroladores pueden ser utilizados (y de hecho hoy en día se utilizan) para un sinfín de aparatos electrónicos. Desde controladores de sensores (industriales y de aplicación comercial), controladores de electrodomésticos, elementos de teléfonos móviles, módems, plotters, fotocopiadoras, impresoras 3D decodificadores de T.V., alarmas, sistemas de navegación, etc.

El uso conjunto de ambos en un mismo dispositivo permite realizar, todas las aplicaciones anteriormente citadas, de forma conjunta y separada, y otras muchas más que al usuario se le ocurra.

6 Conclusiones y líneas futuras de desarrollo.

- Se ha realizado el desarrollo, simulación e implementación de un sistema digital complejo, en un único dispositivo tipo FPGA, de la familia MACHXO2, de Lattice. El sistema incluye un microcontrolador de 8 bits.
- El sistema realiza la evaluación de la distancia, a la que se encuentra un obstáculo, para lo que se emplea un sensor de ultrasonidos.
- La generación de la información, de la distancia al obstáculo, se realiza en la lógica de propósito general del dispositivo, quedando libre el microcontrolador para el postprocesamiento de la información.
- Se ha hecho un análisis de los recursos empleados, dentro de la FPGA, comparando la implantación de dos tipos de procesadores, comprobándose que un microcontrolador de 8 bits “encaja” bien en un dispositivo de la familia MACHXO2, mientras que, si se quiere implementar un microcontrolador de 32 bits, se debe recurrir a otras familias que dispongan de más recursos.
- El diseño se ha realizado de forma estructurada y jerárquica, empleando técnicas de diseño modernas. Para realizar el diseño se han empleado esquemas, descripciones en VHDL y verilog, y bloques generados con la herramienta IPexpress
- La programación del microcontrolador se realiza en lenguaje C.
- Se ha generado un símbolo para el microcontrolador, lo que ha permitido interconectarle con el resto de los elementos en un esquema. Este procedimiento, original, evita el tener que generar una descripción estructural del sistema completo.
- Se ha generado la implantación del sistema completo, en un prototipo, comprobando su completa funcionalidad.

  Como líneas futuras de desarrollo, se proponen:
  - Aumentar las funcionalidades del sistema, mediante el software incorporado en el microcontrolador.
  - Empleo de lenguaje ensamblador, como complemento a la programación en lenguaje C
  - Utilización de microcontroladores más potentes (32 bits) lo que requeriría emplear, familias de FPGA con más recursos.

7 Bibliografía.

[1] LatticeMico8 Tutorial 3.9

[2] MachXO2BreakoutBoardEvaluationKitUsersGuide

[3] "Lattice Diamond User Guide" (Guía de usuario del programa Lattice Diamond)

[4] K. C. Chang, "Digital Design and Modeling with VHDL and synthesis".

[5] S. A. Pérez, E. Soto, S. Fernández, "Diseño de sistemas digitales VHDL".

[6] "MachXO2 Family Handbook" (Manual de usuario de la familia de dispositivos MachXO2)

[7] https://elrincondelc.com/ Visitado por ultima vez 20/09/2017

[8] https://www.academia.edu/15170541/Comparaci%C3%B3n_entre_Microcontroladores_y_FPGA Visitado por ultima vez 15/09/2017

[9] https://hackaday.io/project/26175-hdl-training-board-by-fpga-for-real-beginner Visitado por ultima vez 28/09/2017

[10] http://eprints.ucm.es/26200/1/intro_VHDL.pdf Visitado por ultima vez 28/09/2017

[11] Sensor ultrasonidos de Ángel Rodríguez Sánche (AFRS) (se puede encontrar en: https://es.slideshare.net/)

[12] https://learn.mikroe.com/ebooks/microcontroladorespicc/chapter/introduccion-al-mundo-de-los-microcontroladores/ Visitado por ultima vez 28/09/2017

Anexos.

1. Anexo 1: Datasheet LV-MaxSonar-EZ

LV-MaxSonar®-EZ™ Series

High Performance Sonar Range Finder

MB1000, MB1010, MB1020, MB1030, MB1040[2]

With 2.5V - 5.5V power the LV-MaxSonar-EZ provides very short to long-range detection and ranging in a very small package. The LV-MaxSonar-EZ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches typically range as 6-inches[1]. The interface output formats included are pulse width output, analog voltage output, and RS232 serial output. Factory calibration and testing is completed with a flat object. ***[1]See Close Range Operation***

Features

- Continuously variable gain for control and side lobe suppression
- Object detection to zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- Interfaces are active simultaneously
- Serial, 0 to Vcc, 9600 Baud, 81N
- Analog, (Vcc/512) / inch
- Pulse width, (147uS/inch)
- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)
- Actual operating temperature range from –40°C to +65°C, Recommended operating temperature range from 0°C to +60°C [1]

Benefits

- Very low cost ultrasonic rangefinder
- Reliable and stable range data
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery-based systems
- Fast measurement cycles
- Sensor reports the range reading directly and frees up user processor
- Choose one of three sensor outputs
- Triggered externally or internally

Applications and Uses

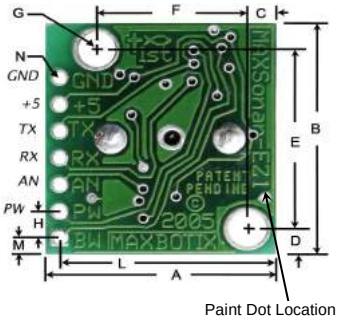
- UAV blimps, micro planes and some helicopters
- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Wide beam sensitivity

Notes:

[1]Please reference page 4 for minimum operating voltage verses temperature information.

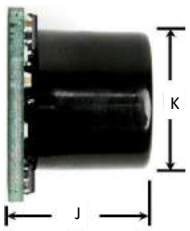
[2] Please reference page 12 for part number key.

LV-MaxSonar-EZ Mechanical Dimensions



A	0.785"	19.9 mm	H	0.100"	2.54 mm
B	0.870"	22.1 mm	J	0.610"	15.5 mm
C	0.100"	2.54 mm	K	0.645"	16.4 mm
D	0.100"	2.54 mm	L	0.735"	18.7 mm
E	0.670"	17.0 mm	M	0.065"	1.7 mm
F	0.510"	12.6 mm	N	0.038" dia.	1.0 mm dia.
G	0.124" dia.	3.1 mm dia.	weight, 4.3 grams		

Part Number	MB1000	MB1010	MB1020	MB1030	MB1040
Paint Dot Color	Black	Brown	Red	Orange	Yellow



Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 6 inches. Although most users find MaxSonar sensors to work reliably from 0 to 6 inches for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

About Ultrasonic Sensors

Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

Pin Out Description

Pin 1-BW-*Leave open or hold low for serial output on the TX output. When BW pin is held high the TX output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2-PW- This pin outputs a pulse width representation of range. The distance can be calculated using the scale factor of 147uS per inch.

Pin 3-AN- Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.

Pin 4-RX– This pin is internally pulled high. The LV-MaxSonar-EZ will continually measure range and output if RX data is left unconnected or held high. If held low the sensor will stop ranging. Bring high for 20uS or more to command a range reading.

Pin 5-TX- When the *BW is open or held low, the TX output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in inches up to a maximum of 255, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When BW pin is held high the TX output sends a single pulse, suitable for low noise chaining. (no serial data)

Pin 6-+5V- Vcc – Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V. Please reference page 4 for minimum operating voltage verses temperature information.

Pin 7-GND- Return for the DC power supply. GND (& Vcc) must be ripple and noise free for best operation.

Range "0" Location

Range Zero

The range is measured from the front of the transducer.

The LV-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.

In general, the LV-MaxSonar-EZ will report the range to the leading edge of the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Sensor Minimum Distance

The sensor minimum reported distance is 6-inches (15.2 cm). However, the LV-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 6-inches will typically range as 6-inches.

Sensor Operation from 6-inches to 20-inches

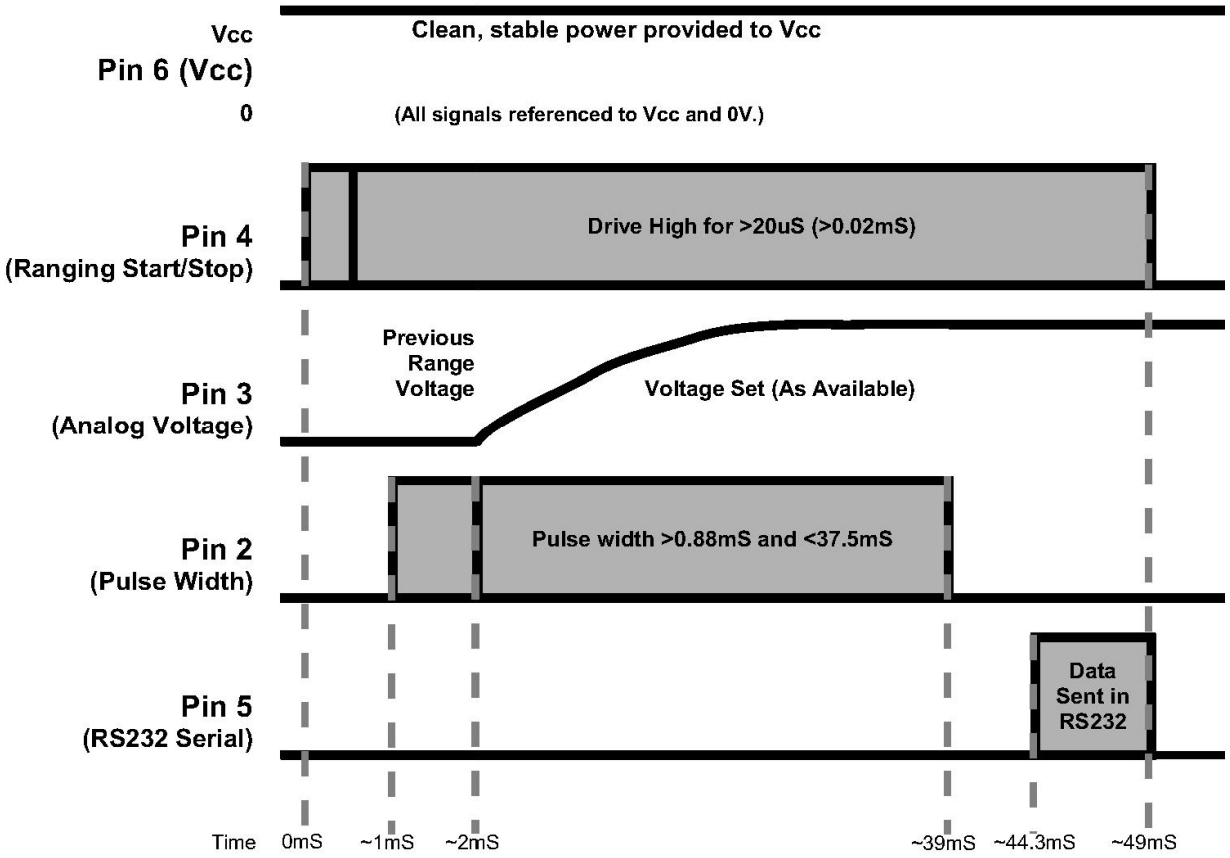
Because of acoustic phase effects in the near field, objects between 6-inches and 20-inches may experience acoustic phase

cancellation of the returning waveform resulting in inaccuracies of up to 2-inches. These effects become less prevalent as the target distance increases, and has not been observed past 20-inches.

General Power-Up Instruction

Each time the LV-MaxSonar-EZ is powered up, it will calibrate during its first read cycle. The sensor uses this stored information to range a close object. It is important that objects not be close to the sensor during this calibration cycle. The best sensitivity is obtained when the detection area is clear for fourteen inches, but good results are common when clear for at least seven inches. If an object is too close during the calibration cycle, the sensor may ignore objects at that distance.

The LV-MaxSonar-EZ does not use the calibration data to temperature compensate for range, but instead to compensate for the sensor ringdown pattern. If the temperature, humidity, or applied voltage changes during operation, the sensor may require recalibration to reacquire the ringdown pattern. Unless recalibrated, if the temperature increases, the sensor is more likely to have false close readings. If the temperature decreases, the sensor is more likely to have reduced up close sensitivity. To recalibrate the LV-MaxSonar-EZ, cycle power, then command a read cycle.



Timing Diagram

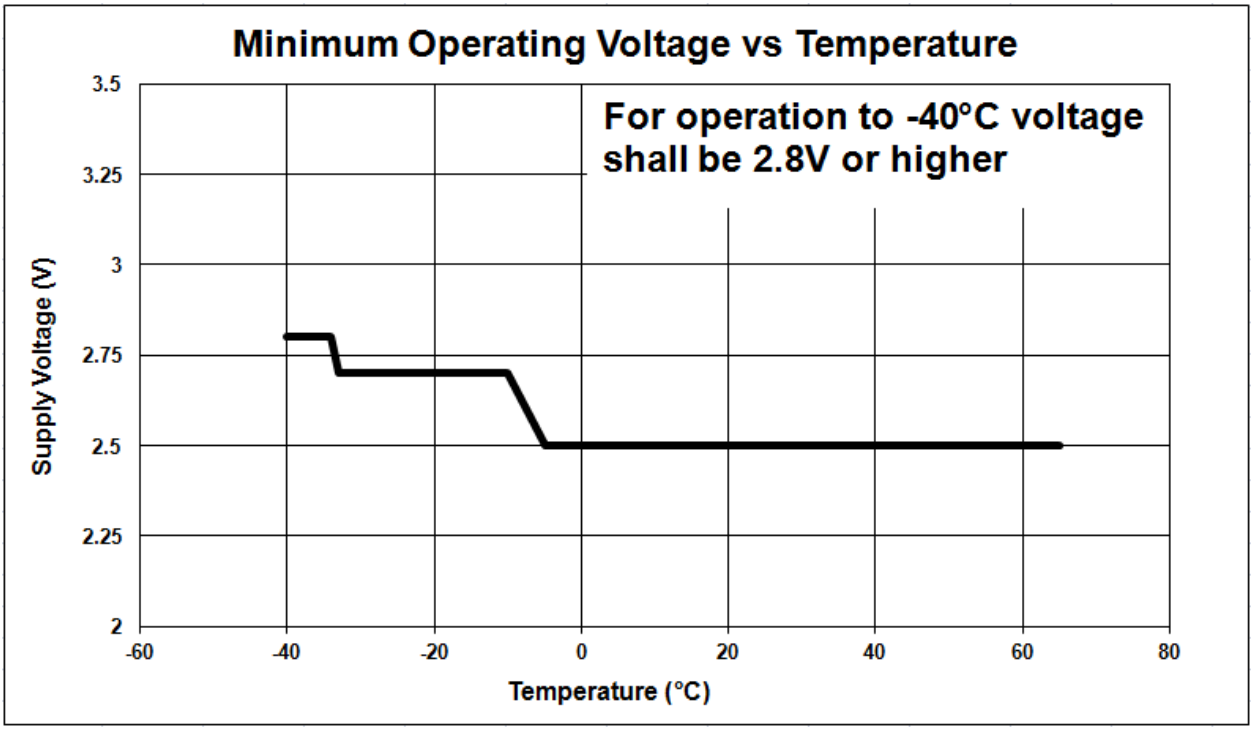
Timing Description

250mS after power-up, the LV-MaxSonar-EZ is ready to accept the RX command. If the RX pin is left open or held high, the sensor will first run a calibration cycle (49mS), and then it will take a range reading (49mS). After the power up delay, the first reading will take an additional ~100mS. Subsequent readings will take 49mS. The LV-MaxSonar-EZ checks the RX pin at the end of every cycle. Range data can be acquired once every 49mS.

Each 49mS period starts by the RX being high or open, after which the LV-MaxSonar-EZ sends the transmit burst, after which the pulse width pin (PW) is set high. When a target is detected the PW pin is pulled low. The PW pin is high for up to 37.5mS if no target is detected. The remainder of the 49mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level. When a long distance is measured immediately after a short distance reading, the analog voltage may not reach the exact level within one read cycle. During the last 4.7mS, the serial data is sent.

Voltage vs Temperature

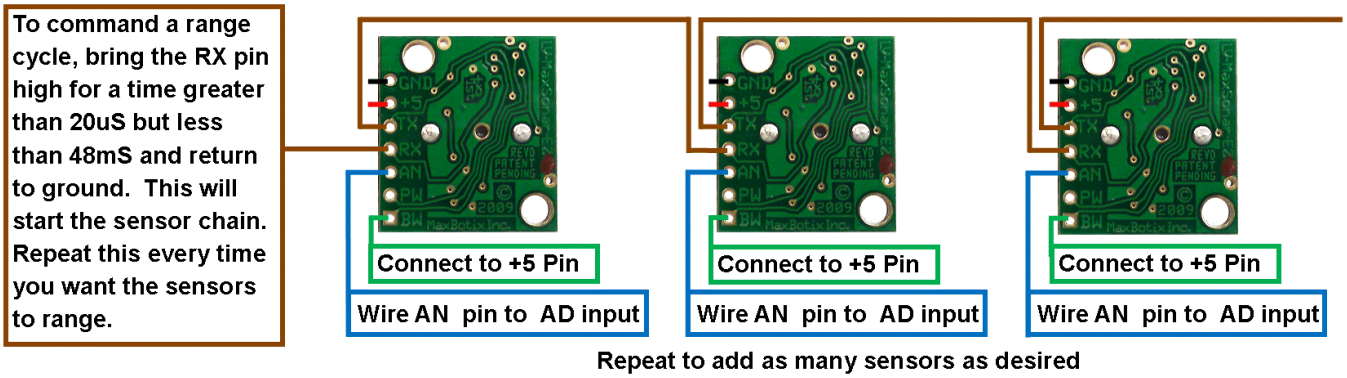
The graph below shows minimum operating voltage of the sensor verses temperature.



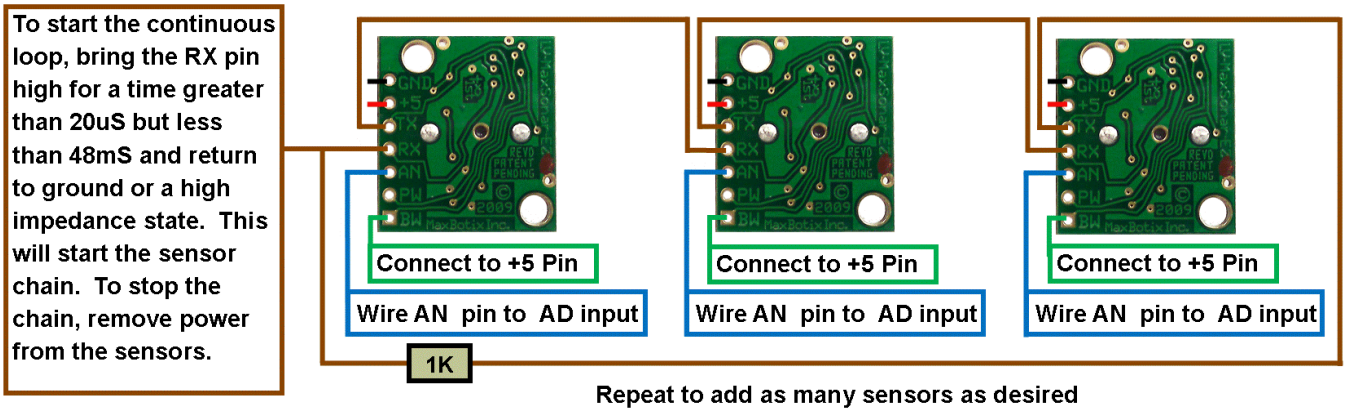
Using Multiple Sensors in a single system

When using multiple ultrasonic sensors in a single system, there can be interference (cross-talk) from the other sensors. MaxBotix Inc., has engineered and supplied a solution to this problem for the LV-MaxSonar-EZ sensors. The solution is referred to as chaining. We have 3 methods of chaining that work well to avoid the issue of cross-talk.

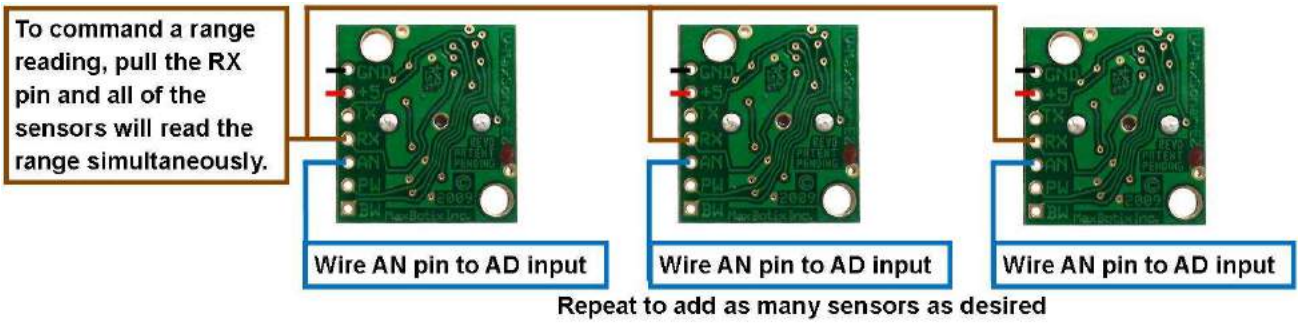
The first method is AN Output Commanded Loop. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, the array stops until the first sensor is triggered to range again. Below is a diagram on how to set this up.



The next method is AN Output Constantly Looping. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, it will trigger the first sensor in the array to range again and will continue this loop indefinitely. Below is a diagram on how to set this up.



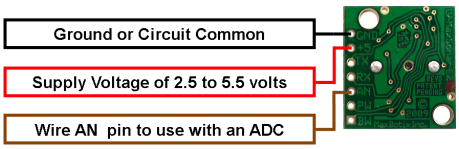
The final method is AN Output Simultaneous Operation. This method does not work in all applications and is sensitive to how the other sensors in the array are positioned in comparison to each other. Testing is recommend to verify this method will work for your application. All the sensors RX pins are conned together and triggered at the same time causing all the sensor to take a range reading at the same time. Once the range reading is complete, the sensors stop ranging until triggered next time. Below is a diagram on how to set this up.



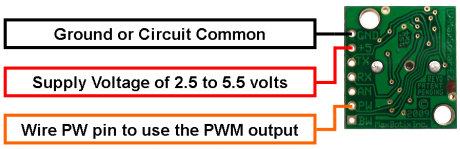
Independent Sensor Operation

The LV-MaxSonar-EZ sensors have the capability to operate independently when the user desires. When using the LV-MaxSonar-EZ sensors in single or independent sensor operation, it is easiest to allow the sensor to free-run. Free-run is the default mode of operation for all of the MaxBotix Inc., sensors. The LV-MaxSonar-EZ sensors have three separate outputs that update the range data simultaneously: Analog Voltage, Pulse Width, and RS232 Serial. Below are diagrams on how to connect the sensor for each of the three outputs when operating in a single or independent sensor operating environment.

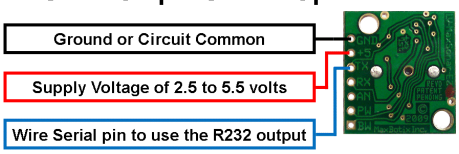
Analog Output Sensor Operation



Pulse Width Output Sensor Operation



Serial Output Sensor Operation



Selecting an LV-MaxSonar-EZ

Different applications require different sensors. The LV-MaxSonar-EZ product line offers varied sensitivity to allow you to select the best sensor to meet your needs.

The LV-MaxSonar-EZ Sensors At a Glance

People Detection **Wide Beam** **High Sensitivity**		**Best Balance**		**Large Targets** **Narrow Beam** **Noise Tolerance**
MB1000	**MB1010**	**MB1020**	**MB1030**	**MB1040**

The diagram above shows how each product balances sensitivity and noise tolerance. This does not effect the maximum range, pin outputs, or other operations of the sensor. To view how each sensor will function to different sized targets reference the LV-MaxSonar-EZ Beam Patterns.

Background Information Regarding our Beam Patterns

Each LV-MaxSonar-EZ sensor has a calibrated beam pattern. Each sensor is matched to provide the approximate detection pattern shown in this datasheet. This allows end users to select the part number that matches their given sensing application. Each part number has a consistent field of detection so additional units of the same part number will have similar beam patterns. The beam plots are provided to help identify an estimated detection zone for an application based on the acoustic properties of a target versus the plotted beam patterns.

People Sensing: For users that desire to detect people, the detection area to the 1-inch diameter dowel, in general, represents the area that the sensor will reliably detect people.

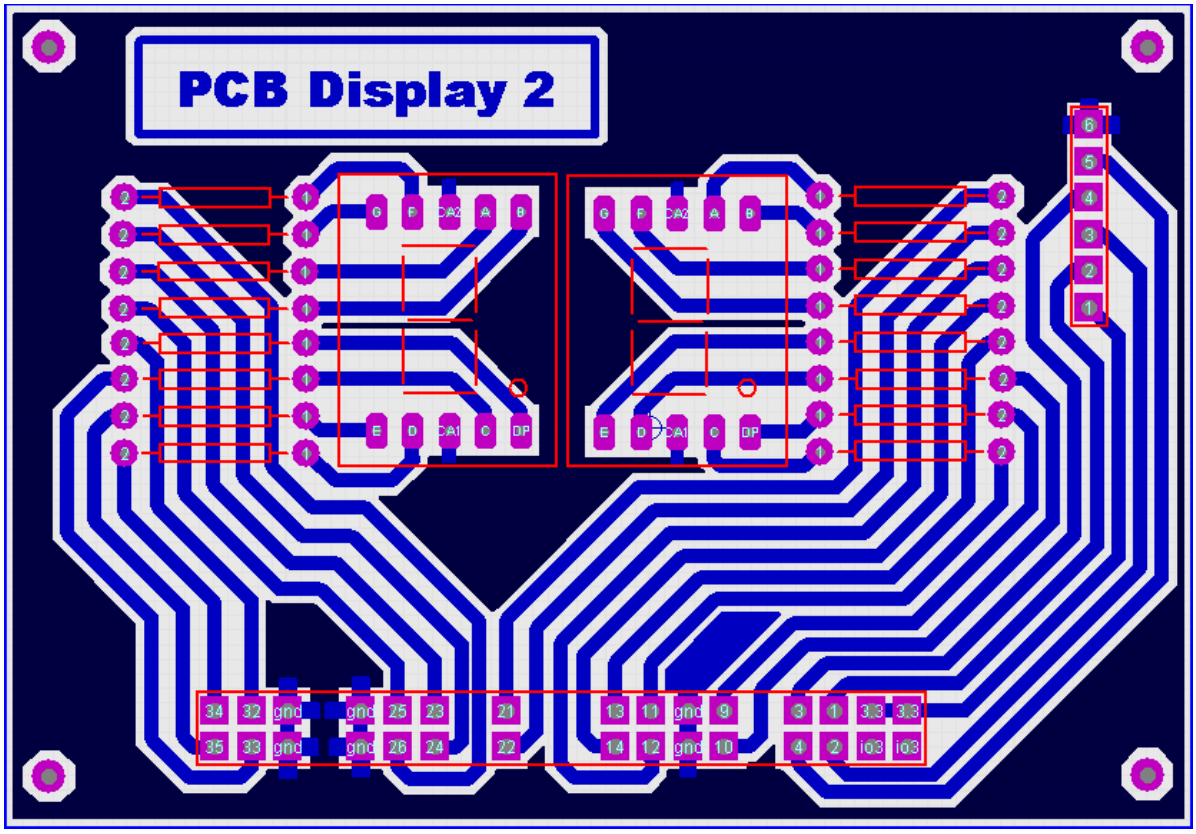
Each beam pattern is a 2D representation of the detection area of the sensor. The beam pattern is actually shaped like a 3D cone (having the same detection pattern both vertically and horizontally). Detection patterns for dowels are used to show the beam pattern of each sensor. Dowels are long cylindered targets of a given diameter. The dowels provide consistent target detection characteristics for a given size target which allows easy comparison of one MaxSonar sensor to another MaxSonar sensor.

For each part number, the four patterns (A, B, C, and D) represent the detection zone for a given target size. Each beam pattern shown is determined by the sensor's part number and target size.

The actual beam angle changes over the full range. Use the beam pattern for a specific target at any given distance to calculate the beam angle for that target at the specific distance. Generally, smaller targets are detected over a narrower beam angle and a shorter distance. Larger targets are detected over a wider beam angle and a longer range.

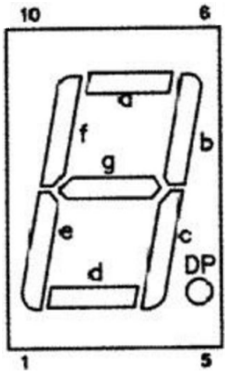
2. Anexo 2: Pines de entradas y salidas de las placas de expansión.

PLACA PCB DISPLAY 2



PINOUT DE LA FPGA ASIGNADO EN LA PLACA PCB DISPLAY 2:

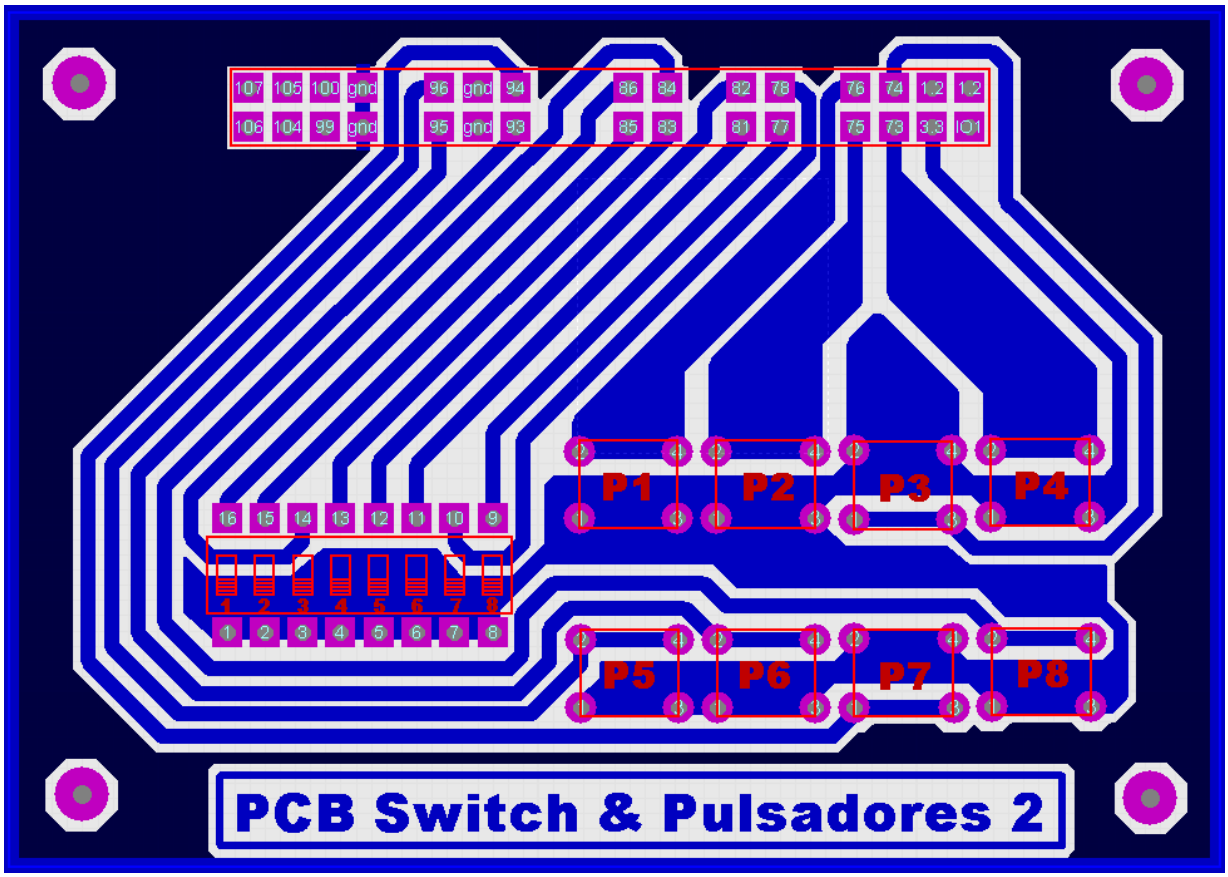
DISPLAY 2 (A LA IZQDA)				*DISPLAY1 (A LA DRCHA)*		
A2	PIN7	23		A1	PIN7	21
B2	PIN6	25		B1	PIN6	22
C2	PIN4	33		C1	PIN4	11
D2	PIN2	34		D1	PIN2	10
E2	PIN1	35		E1	PIN1	13
F2	PIN9	26		F1	PIN9	12
G2	PIN10	24		G1	PIN10	14
DP2	PIN5	32		DP1	PIN5	9



PINES AUXILIARES PARA OTRA PLACA DE AMPLIACIÓN (DE ABAJO A ARRIBA)

PIN1	VCC (3.3v)	PIN4	3
PIN2	1	PIN5	4
PIN3	2	PIN6	GND

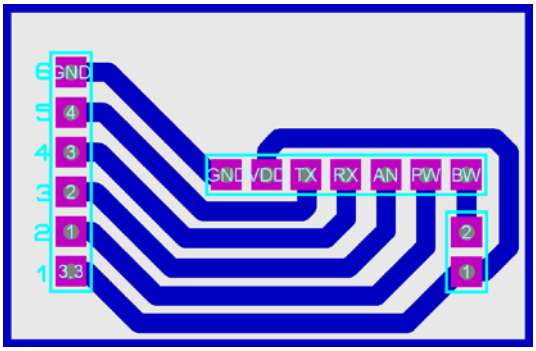
PLACA PCB SWITCH & PULSADORES 2

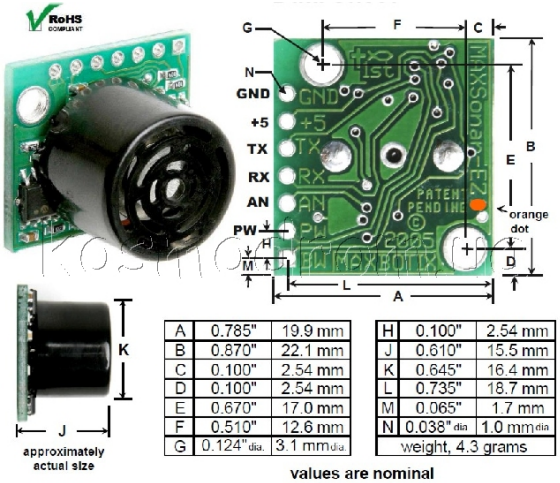


PINES MICROSWITCH		PINES PULSADORES	
SW1	86	PULS1	76
SW2	85	PULS2	75
SW3	84	PULS3	74
SW4	83	PULS4	73
SW5	82	PULS5	96
SW6	81	PULS6	95
SW7	78	PULS7	94
SW8	77	PULS8	93

NOTA: Al subir a 'ON' un microswitch o presionar un pulsador el estado de la entrada correspondiente en la FPGA se pone a nivel alto (3.3 V).

PLACA PCB MÓDULO ULTRASONIDOS





A	0.785"	19.9 mm
B	0.870"	22.1 mm
C	0.100"	2.54 mm
D	0.100"	2.54 mm
E	0.670"	17.0 mm
F	0.510"	12.6 mm
G	0.124" dia.	3.1 mm dia.

H	0.100"	2.54 mm
J	0.610"	15.5 mm
K	0.645"	16.4 mm
L	0.735"	18.7 mm
M	0.065"	1.7 mm
N	0.038" dia	1.0 mm dia
weight, 4.3 grams		

values are nominal

El LV-MaxSonar-EZ3 es un sensor ultrasónico que tiene integrado emisor y receptor en un solo módulo. Tiene un rango de detección de 0 cm a 6.45 m. Desde 15.24 cm hasta los 6.45 m tiene una resolución de una pulgada (2.54 cm). Dispone de tres formatos de salida:

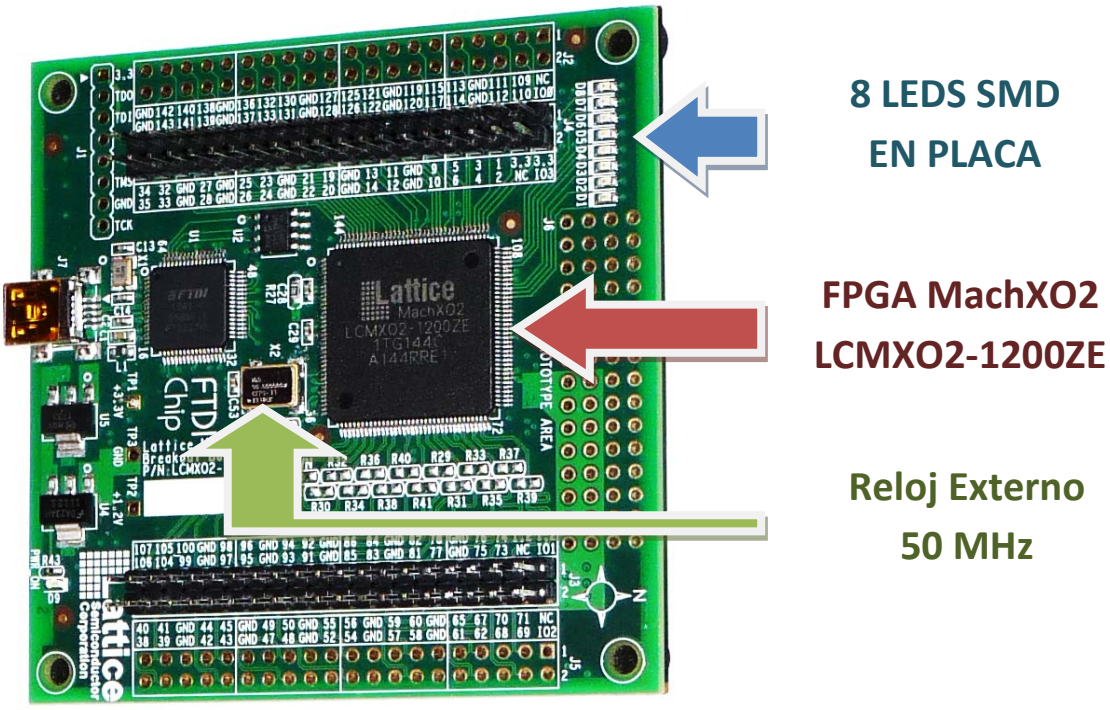
- **Ancho de pulso a través de la patilla PW (Pin 1 de la FPGA). La distancia se determina aplicando el factor de escala de 147uS por pulgada.**
- **Salida de tensión analógica AN (Pin 2 de la FPGA). La distancia se determina en nuestro caso para una Vcc de 3.3 voltios aplicando el factor de escala de 6.4mV/pulgada (El factor de escala es *Vcc/512* V/pulgada).**
- **Salida digital serie TX (Pin 4 de la FPGA). Para habilitar esta salida hay que poner a nivel bajo o sin conexión la patilla BW. Esta salida emplea comunicación RS232. Inicia la trama mandando el código ASCII 'R', después manda tres dígitos en ASCII indicando la distancia en pulgadas hasta un valor máximo de 255 y finalmente termina la trama mandando un retorno de carro (valor ASCII 13).**

El Pin 3 de la FPGA va al pin 'RX' del módulo. La entrada RX a nivel alto durante un mínimo de 20 uS realiza una lectura. En caso de permanecer a nivel alto realizará lecturas continuadas y a nivel bajo detiene la medición.

La placa dispone de un Jumper que define el valor del pin 'BW'. La entrada BW como se ha comentado anteriormente solo sirve para habilitar (BW='0') o deshabilitar (BW='1') la salida digital serie.
Si JMP está cerrado BW='1' y si se retira el jumper entonces BW='0'.

Nota: Esta placa solo se puede conectar mediante el conector de ampliación de la placa PCB Display.

PLACA MachXO2 BREAKOUT BOARD



- RELOJ EXTERNO

Se ha agregado a la placa un reloj externo de 50 MHz. El reloj dispone de una patilla de habilitación CLK_EN que va al PIN 32 de la FPGA. A nivel alto el Cristal externo esta habilitado y a nivel bajo deshabilitado. La salida del reloj CLK_OUT va al PIN 27 de la FPGA.

CLK_OUT PIN 27 **CLK EN PIN 32**

- LEDS ON BOARD

D1 -> PIN 97 **D5 -> PIN 104**

D2 -> PIN 98 **D6 -> PIN 105**

D3 -> PIN 99 **D7 -> PIN 106**

D4 -> PIN 100 **D8 -> PIN 107**

3. Anexo 3: Tabla de Instrucciones LatticeMico8.

Instruction Set

This chapter includes descriptions of all the instruction opcodes of the LatticeMico8 microcontroller.

Instruction Formats

All LatticeMico8 instructions are 18 bits wide. They are in three basic formats, as shown in Figure 5, Figure 6, and Figure 7.

Figure 5: Register-Register Format

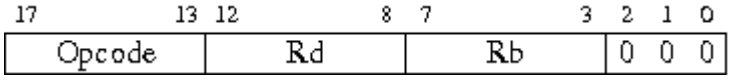


Figure 6: Register-Immediate Format

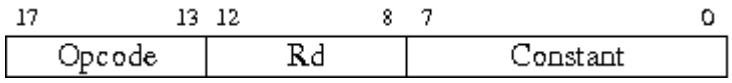
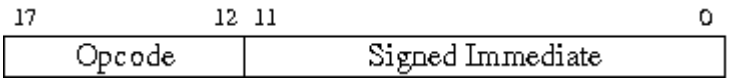


Figure 7: Immediate Format



Instruction Set Lookup Table

Table 7: Instruction Set Reference Card

Operation	Action	Flags
ADD Rd, Rb	Rd = Rd + Rb	Carry, Zero
ADDC Rd, Rb	Rd = Rd + Rb + Carry	Carry, Zero
ADDI Rd, C	Rd = Rd + C	Carry, Zero
ADDIC Rd, C	Rd = Rd + C + Carry	Carry, Zero
AND Rd, Rb	Rd = Rd & Rb	Zero
ANDI Rd, C	Rd = Rd & C	Zero
B Label	PC = PC + Label	
BC Label	If Carry = 1, PC = PC + Label	
BNC Label	If Carry = 0, PC = PC + Label	
BNZ Label	If Zero = 0, PC = PC + Label	
BZ Label	If Zero = 1, PC = PC + Label	
CALL Label	Stack = PC + 1, PC = PC + Label	
CALLC Label	If Carry = 1, Stack = PC + 1, PC = PC + Label	
CALLNC Label	If Carry = 0, Stack = PC + 1, PC = PC + Label	
CALLNZ Label	If Zero = 0, Stack = PC + 1, PC = PC + Label	
CALLZ Label	If Zero = 1, Stack = PC + 1, PC = PC + Label	
CLRC	Carry = 0	Carry
CLRI	IE = 0	
CLRZ	Zero = 0	Zero
CMP Rd, Rb	Rd – Rb	Carry, Zero
CMPI Rd, C	Rd – C	Carry, Zero
EXPORT Rd, Port#	Peripheral (Port #) = Rd	
EXPORTI Rd, Rb	Peripheral (Page Pointer, Rb) = Rd	
IMPORT Rd, Port#	Rd = Peripheral (Port #)	
IMPORTI Rd, Rb	Rd = Peripheral (Page Pointer, Rb)	
IRET	PC, Carry, Zero = Stack	Carry, Zero
LSP RD, SS	Rd = Scratchpad (SS)	
LSPI Rd, Rb	Rd = Scratchpad (Page Pointer, Rb)	
MOV Rd, Rb	Rd = Rb	
MOVI Rd, C	Rd = Const	

Table 7: Instruction Set Reference Card (Continued)

Operation	Action	Flags
NOP	PC = PC + 1	
OR Rd, Rb	Rd = Rd | Rb	Zero
ORI Rd, C	Rd = Rd | C	Zero
RCSR Rd, CRb	Rd = CSR (Rb)	
RET	PC = Stack	
ROL Rd, Rb	Rd = {(Rb<<1), Rb[0]}	Zero
ROLC Rd, Rb	Rd = {(Rb<<1), Carry}, Carry = Rb[7]	Carry, Zero
ROR Rd, Rb	Rd = {Rb[0], (Rb>>1)}	Zero
RORC Rd, Rb	Rd = {Carry, (Rb>>1)}, Carry = Rb[0]	Carry, Zero
SETC	Carry = 1	Carry
SETI	IE = 0	
SETZ	Zero = 1	Zero
SSP Rd, SS	Scratchpad (SS) = Rd	
SSPI Rd, Rb	Scratchpad (Page Pointer, Rb) = Rd	
SUB Rd, Rb	Rd = Rd – Rb	Carry, Zero
SUBC Rd, Rb	Rd = Rd – Rb – Carry	Carry, Zero
SUBI Rd, C	Rd = Rd – C	Carry, Zero
SUBIC Rd, C	Rd = Rd – C – Carry	Carry, Zero
TEST Rd, Rb	Rd & Rb	Zero
TESTI Rd, C	Rd & C	Zero
XOR Rd, Rb	Rd = Rd ^ Rb	Zero
XORI Rd, C	Rd = Rd ^ C	Zero
WCSR CRd, Rb	CSR (Rd) = Rb	Zero

4. Anexo 4: Arquitectura LatticeMico8.

Architecture

This chapter describes the LatticeMico8 register and memory architecture and explains the interrupt architecture and call stack.

Register Architecture

This section describes the general-purpose and control and status registers of the LatticeMico8 architecture.

General-Purpose Registers

The LatticeMico8 microcontroller can be configured to have either 16 or 32 general-purpose registers. Each register is 8 bits wide. The registers are implemented using a dual-port distributed memory. The LatticeMico8 opcode set permits the microcontroller to access 32 registers. When LatticeMico8 is configured with 16 registers, any opcode reference to R16 to R31 maps to R0 to R15 respectively.

General-purpose registers R13, R14, and R15 can also be used by the LatticeMico8 microcontroller as page-pointer registers, depending on the current memory mode. Page pointers (PP) are used when the scratchpad and peripheral memory spaces are larger than 256 bytes (see "Memory Modes" on page 9). The memory address is formed by concatenating the values in registers R13, R14, and R15 with an 8-bit value derived from the LatticeMico8 memory instruction. Table 1 on page 4 highlights the three LatticeMico8 memory modes and corresponding designation of registers R13, R14, and R15.

- In the large memory mode, registers R13, R14, and R15 indicate which of the 16M pages is currently active. R13 provides the least-significant byte of page address and R15 provides most-significant byte.

- In the medium memory mode, register R13 indicates which of the 256 pages is currently active.

Table 1: Designation of LatticeMico8 Registers Based on LatticeMico8 Memory Mode

	LatticeMico8 Memory Mode		
Register Number	**Small**	**Medium**	**Large**
0 through 12	general-purpose	general-purpose	general-purpose
13	general-purpose	PP	PP (LSB)
14	general-purpose	general-purpose	PP
15	general-purpose	general-purpose	PP (MSB)
16 through 31	general-purpose	general-purpose	general-purpose

Control and Status Registers

Table 2 shows all the names of the control and status registers (CSR), the read and write access, and the index used when the register is accessed. All signal levels are active high.

Table 2: Control and Status Registers

Name	Access	Index	Description
IP	R/W	0	Interrupt Pending
IM	R/W	1	Interrupt Mask
IE	R/W	2	Global Interrupt Enable/Disable

IP – Interrupt Pending The IP CSR contains a pending bit for each of the 8 external interrupts. A pending bit is set when the corresponding interrupt request line is asserted low. Bit 0 corresponds to interrupt 0. Bits in the IP CSR can be cleared by writing a 1 with the *wcsr* instruction. Writing a 0 has no effect. After reset, the value of the IP CSR is 0.

IM – Interrupt Mask The IM CSR contains an enable bit for each of the 8 external interrupts. Bit 0 corresponds to interrupt 0. In order for an interrupt to be raised, both an enable bit in this register and the IE flag in the IE CSR must be set to 1. After reset, the value of the IM CSR is 0.

IE – Global Interrupt Enable The IE CSR contains a single-bit (bit position 0) flag, IE, which determines whether interrupts are enabled. This flag has priority over the IM CSR. After reset, the value of the IE CSR is 0.

Memory Architecture

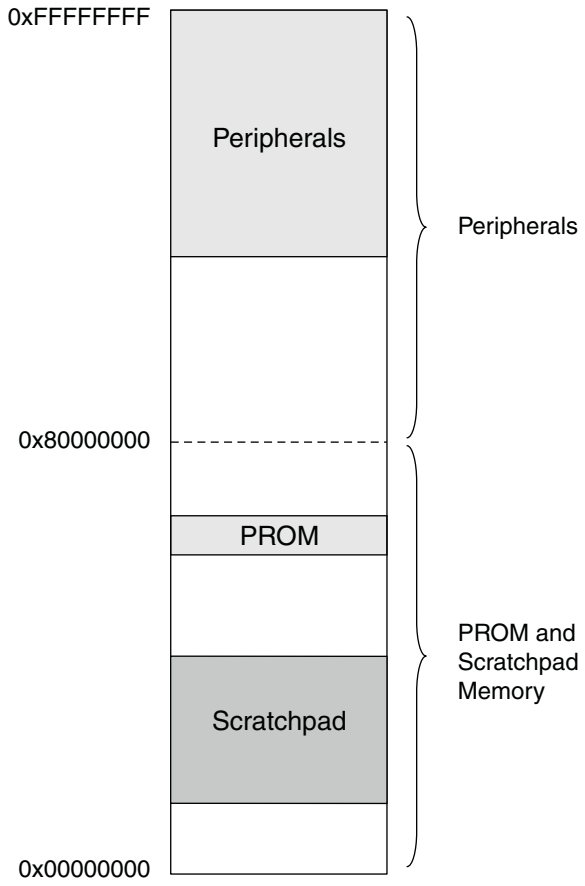
This section describes the memory architecture of the LatticeMico8 microcontroller.

Memory Regions

The LatticeMico8 microcontroller recognizes three independent memory regions. Each memory region has its own independent input/output interface and its own instruction set support. These three memory regions are called the PROM, the Scratchpad, and the Peripheral memory regions respectively. The size and location of each of these memory regions is configurable as long as all these three memory regions are located entirely within the 4GB address space. These memory regions can also be configured to overlap within LatticeMico System Builder. Figure 2 shows the three memory regions and the address space to which they are confined by LatticeMico System Builder.

See “Acessing LatticeMico8 Memory Regions” on page 42 for details on how to access each of the three memory regions from a software programmer's perspective.

Figure 2: Memory Organization



PROM Space

The PROM memory region contains the program code that will be executed by the LatticeMico8 microcontroller core and is accessible via its instruction fetch engine. The size of the PROM memory region can be configured to accommodate 256, 512, 1024, 2048, or 4096 instruction opcodes. By default the memory region is located within the LatticeMico8 microcontroller. The memory regions can also be configured to be external to the LatticeMico8 microcontroller.

When the PROM memory region is internal to the microcontroller, it is connected to the LatticeMico8 instruction fetch engine via a dedicated high-speed bus that fetches one instruction opcode per clock cycle. There is no instruction set support to write to internal PROM. When the PROM memory region is external to the microcontroller, it is accessed by the master WISHBONE interface within the LatticeMico8 instruction fetch engine. This WISHBONE interface has a 8-bit data bus and it takes three 8-bit WISHBONE accesses to fetch one LatticeMico8 instruction opcode. The instruction fetch latency is now dictated by the system WISHBONE latency and the latency of the PROM memory. The minimum instruction fetch latency is 12 clock cycles. Table 3 shows the WISHBONE interface signals. For more information about the WISHBONE System-On-Chip (SoC) Interconnection Architecture for Portable IP Cores, as it is formally known, refer to the OPENCORES.ORG Web site at www.opencores.org/projects.cgi/web/wishbone.

Table 3: PROM WISHBONE Interface Signals

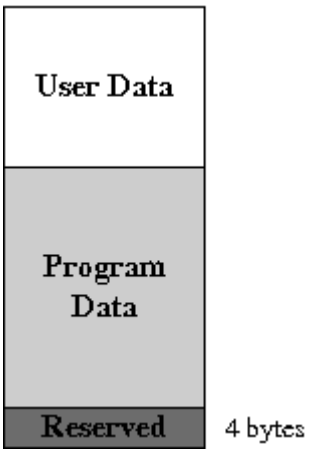
Name	Width	Direction	Description
I_CYC_O	1	Output	A new LatticeMico8 instruction fetch request is initiated by asserting this signal. This signal remains asserted until I_ACK_I is asserted, which indicates the completion of the request.
I_STB_O	1	Output	A new LatticeMico8 instruction fetch request is initiated by asserting this signal. This signal may be valid only for the first cycle.
I_CTI_O	2	Output	Always has a value 2'b00
I_BTE_O	3	Output	Always has a value 3'b000
I_ADR_O	32	Output	The address output array I_ADR_O( ) is used to pass a binary address.
I_WE_O	1	Output	Always has a value 1'b0
I_SEL_O	4	Output	Always has a value 4'b1111
I_DAT_O	8	Output	Unused
I_LOCK_O	1	Output	Unused (signal exists, but it is not implemented)
I_ACK_I	1	Input	When asserted, the signal indicates the normal termination of a bus cycle and that an instruction is available on I_DAT_I bus.
I_ERR_I	1	Input	Unused (signal exists, but it is not implemented)
I_RTY_I	1	Input	Unused (signal exists, but it is not implemented)
I_DAT_I	8	Input	One byte of the LatticeMico8 18-bit instruction opcode is available on this bus when I_ACK_I is asserted. It takes three WISHBONE transactions to complete one LatticeMico8 instruction fetch.

The advantage of configuring the PROM memory region as external to the LatticeMico8 microcontroller is that the PROM memory region can now be configured to overlap with other LatticeMico8 memory regions within Lattice Mico System Builder and, therefore, be directly written to by LatticeMico8 opcodes. This configuration also offers the ability to store and execute LatticeMico8 instructions from non-volatile memory such as Flash. As shown in Figure 2 on page 5, the external PROM memory region can be placed at any location within a 4GB address range. When the LatticeMico8 microcontroller is instantiated using Lattice Mico System Builder, it will restrict the placement of external PROM between 0x00000000 and 0x80000000.

Scratchpad Space

LatticeMico8 provides an independent memory space that is designed to be used for program read/write and read-only data as well as other user-defined data. The size of this scratchpad memory can be configured from 32 bytes to 4G bytes, in power-of-two increments. Figure 3 shows the structure of this scratchpad space and how data is located within this space. The scratchpad memory space can be placed at any location within a 4GB address range. The first 4 bytes are reserved for LatticeMico8 interrupt handling. Program data is situated above this reserved space. The designer can configure the size of scratchpad memory that is used for program data. User-defined data is optional and is always located after program data.

Figure 3: Scratchpad Space Structure



The scratchpad memory can be configured to be entirely internal to the LatticeMico8 microcontroller, entirely external to LatticeMico8 microcontroller, or a combination of both.

- The internal scratchpad is implemented using single-port EBRs and is hooked up to the LatticeMico8 core through a dedicated bus. Reads or writes to the internal scratchpad take a single clock cycle.
- The external scratchpad is accessed through the Peripheral WISHONE interface of the LatticeMico8 microcontroller (see “Interrupt Architecture” on page 10). Each read or write will take a minimum of 2 clock cycles.

Peripheral (Input/Output) Space

LatticeMico8 provides an independent memory space that is designed to be used for peripherals and other memory-mapped hardware. The size of this peripheral memory space can be configured from 0 bytes to 4G bytes in power-of-two increments. While the peripheral memory space can be placed at any location within a 4GB address range, Lattice Mico System Builder restricts the peripheral memory space to the addresses between 0x80000000 and 0xFFFFFFFF.

This memory space is always external to the LatticeMico8 microcontroller and is primarily used to enable LatticeMico8 to communicate with memory-mapped hardware and peripherals. The LatticeMico8 microcontroller can communicate with any hardware or peripheral within the peripheral memory space, through the peripheral WISHBONE interface within LatticeMico8 core, using LatticeMico8 instruction opcodes. This WISHBONE interface has 8-bit input and output data busses and a 32-bit address bus. Table 4 shows the Peripheral WISHBONE interface signals.

Table 4: Peripheral WISHBONE Interface Signals

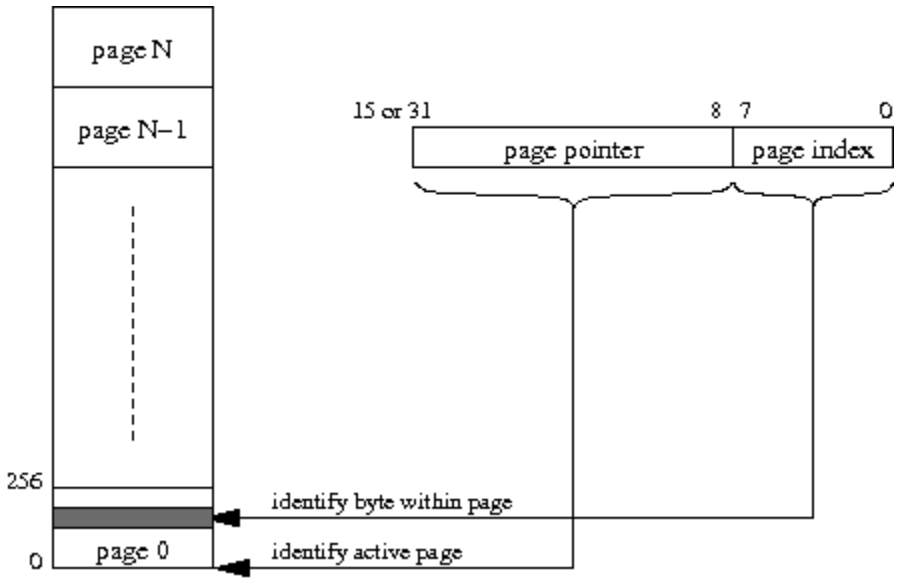
Name	Width	Direction	Description
D_CYC_O	1	Output	A new LatticeMico8 data request is initiated by asserting this signal. This signal remains asserted until D_ACK_I is asserted, which indicates completion of the request.
D_STB_O	1	Output	A new LatticeMico8 data request is initiated by asserting this signal. This signal may be valid only for first cycle.
D_CTI_O	2	Output	This bus will always have a value 2'b00
D_BTE_O	3	Output	This bus will always have a value 3'b000
D_ADR_O	32	Output	The address output array D_ADR_O( ) is used to pass a binary address. D_ADR_O( ) actually has a full 32 bits.
D_WE_O	1	Output	This signal indicates whether a new data request is a read (0) or a write (1). This signal must hold its value as long as D_CYC_O is asserted.
D_SEL_O	1	Output	Always has a value 1'b1
D_DAT_O	8	Output	Has valid data when D_WE_O is 1'b1.
D_LOCK_O	1	Output	Unused (signal exists, but it is not implemented)
D_ACK_I	1	Input	When asserted, the signal indicates the normal termination of a bus cycle.
D_ERR_I	1	Input	Unused (signal exists, but it is not implemented)
D_RTY_I	1	Input	Unused (signal exists, but it is not implemented)
D_DAT_I	8	Input	Data is available on this bus when D_ACK_I and D_WEO are asserted.

Memory Modes

The LatticeMico8 microcontroller can be configured for different sizes for the scratchpad and peripheral memory regions. The size of scratchpad and peripheral memory regions can be as small as 32 bytes and as large as 4G bytes. A 32-byte memory region requires only 5 address bits, while a 4GB memory region requires 32 address bits.

The LatticeMico8 instruction set can directly access only 256 memory locations, since all general-purpose registers are 8 bits wide. (See “Instruction Set” on page 13.) To access memory regions that are larger than 256 bytes, LatticeMico8 relies on a concept called “paging,” in which the memory is logically divided into 256-byte pages. The memory address is composed of two parts, as shown in Figure 4: the page index and the page pointer. The page index is 8 bits wide and addresses a byte in the currently active page, while the page pointer provides the address of the currently active page.

Figure 4: Memory Modes



The page pointers are essentially general-purpose registers that have been retargeted to provide a memory address. (See “Memory Regions” on page 5.) Table 5 shows the memory modes of the LatticeMico8 microcontroller, the size of addressable memory space in each mode, and the general-purpose registers used as page pointers.

Table 5: LatticeMico8 Memory Modes

Memory Mode	Maximum Memory Size	Address Bits	Page Pointer Registers
Small	256 bytes	8	N/A
Medium	16K bytes	16	R13
Large	4G bytes	32	R13, R14, R15

Interrupt Architecture

The LatticeMico8 microcontroller supports up to 8 maskable, active-low, level-sensitive interrupts. Each interrupt line has a corresponding mask bit in the IM CSR. The mask enable is active high. A global interrupt-enable flag is implemented in the IE CSR. The software can query the status of the interrupts and acknowledge them through the IP CSR. If more interrupt sources or more sophisticated interrupt detection methods are required, external interrupt controllers can be cascaded onto the microcontroller's interrupt pins to provide the needed functionality.

When an interrupt is received, the address of the next instruction is pushed into the call stack (see "Call Stack" on page 10), and the microcontroller continues execution from the interrupt vector (address 0). The flags (carry and zero) are pushed onto the call stack along with the return address. An *iret* instruction will pop the call stack and transfer control to the address on top of the stack. The flags (carry and zero) are also popped from the call stack.

See "Interrupt Convention" on page 41 for details on the programming model for interrupts.

Note

The LatticeMico8 microcontroller does not support nested interrupts. Locations 0 through 3 in the scratchpad are reserved for interrupt handling and should not used for any other purpose.

Call Stack

The LatticeMico8 microcontroller implements a hardware call stack to handle procedure calls (*call* instruction) and procedure/interrupt return (*ret* and *iret* instructions). The depth of this call stack determines the number of nested procedure calls that can be handled by the LatticeMico8 microcontroller, and designers can choose the depth to be 8, 16, or 32. When a *call* instruction is executed, the address of the next instruction is pushed on to the call stack. A *ret* or *iret* instruction will pop the stack and continue execution from the location at the top of the stack.

Note

There is no mechanism in hardware to detect whether the number of nested procedure calls has exceeded the depth of the call stack. It is up to the software developer to ensure that the call stack does not overflow.

5. Anexo 5: WISHBONE bus Interface.

.

WISBONE Bus Interface

The WISHBONE Bus in the MachXO2 is compliant with the WISHBONE standard from OpenCores. It provides connectivity between FPGA user logic and the EFB functional blocks. The user can implement a WISHBONE Master interface to interact with the EFB WISHBONE slave interface or a LatticeMico8™ soft processor core can be used to interact with the EFB WISHBONE.

The block diagram in Figure 17-2 shows the supported WISHBONE bus signals between the FPGA core and the EFB. Table 17-2 provides a detailed definition of the supported signals.

Figure 17-2. WISHBONE Bus Interface Between the FPGA Core and the EFB Module

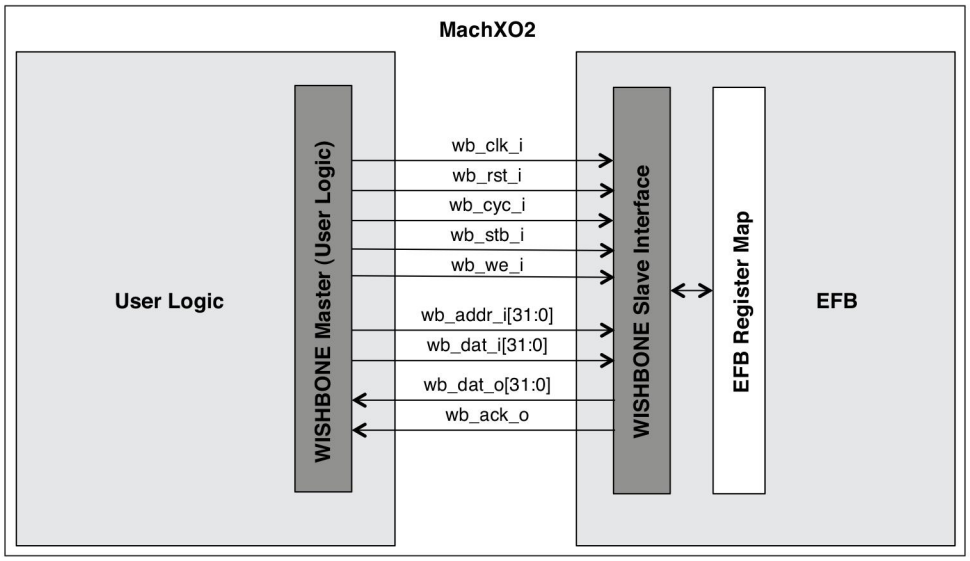


Table 17-2. WISHBONE Slave Interface Signals of the EFB Module

Signal Name	I/O	Width	Description
wb_clk_i	Input	1	Positive edge clock used by WISHBONE Interface registers and hardened functions within the EFB module. Supports clock speeds up to 133 MHz.
wb_rst_i	Input	1	Active-high, synchronous reset signal that will only reset the WISHBONE interface logic. This signal will not affect the contents of any registers. It will only affect ongoing bus transactions. Wait 1us after de-assertion before starting any subsequent WISHBONE transactions.
wb_cyc_i	Input	1	Active-high signal, asserted by the WISHBONE master, indicates a valid bus cycle is present on the bus.
wb_stb_i	Input	1	Active-high strobe, input signal, indicating the WISHBONE slave is the target for the current transaction on the bus. The EFB module asserts an acknowledgment in response to the assertion of the strobe.
wb_we_i	Input	1	Level sensitive Write/Read control signal. Low indicates a Read operation, and High indicates a Write operation.
wb_adr_i	Input	8	8-bit wide address used to select a specific register from the register map of the EFB module.
wb_dat_i	Input	8	8-bit input data path used to write a byte of data to a specific register in the register map of the EFB module.
wb_dat_o	Output	8	8-bit output data path used to read a byte of data from a specific register in the register map of the EFB module.
wb_ack_o	Output	1	Active-high, transfer acknowledge signal asserted by the EFB module, indicating the requested transfer is acknowledged.

To interface to the EFB you must create a WISHBONE Master controller in the User Logic. In a multiple-Master configuration, the WISHBONE Master outputs are multiplexed in a user-defined arbiter. A LatticeMico8 soft processor can also be utilized along with the Mico System Builder (MSB) platform which can implement multi-Master bus configurations. If two Masters request the bus in the same cycle, only the outputs of the arbitration winner reach the Slave interface.

The EFB WISHBONE bus supports the “Classic” version of the WISHBONE standard. Given that the WISHBONE bus is an open source standard, not all features of the standard are implemented or required:

- Tags are not supported in the WISHBONE Slave interface of the EFB module. Given that the EFB is a hardened block, these signals cannot be added by the user.
- The Slave WISHBONE bus interface of the EFB module does not require the byte select signals (sel_i or sel_o), since the data bus is only a single byte wide.
- The EFB WISHBONE slave interface does not support the optional error and retry access termination signals. If the slave receives an access to an invalid address, it will simply respond by asserting wb_ack_o signal. It is the responsibility of the user to stay within the valid address range.

WISHBONE Write Cycle

Figure 17-3 shows the waveform of a Write cycle from the perspective of the EFB WISHBONE Slave interface. During a single Write cycle, only one byte of data is written to the EFB block from the WISHBONE Master. A Write operation requires a minimum three clock cycles.

On clock Edge 0, the Master updates the address, data and asserts control signals. During this cycle:

- The Master updates the address on the wb_adr_i[7:0] address lines
- Updates the data that will be written to the EFB block, wb_dat_i[7:0] data lines
- Asserts the write enable wb_we_i signal, indicating a write cycle
- Asserts the wb_cyc_i to indicate the start of the cycle
- Asserts the wb_stb_i, selecting a specific slave module

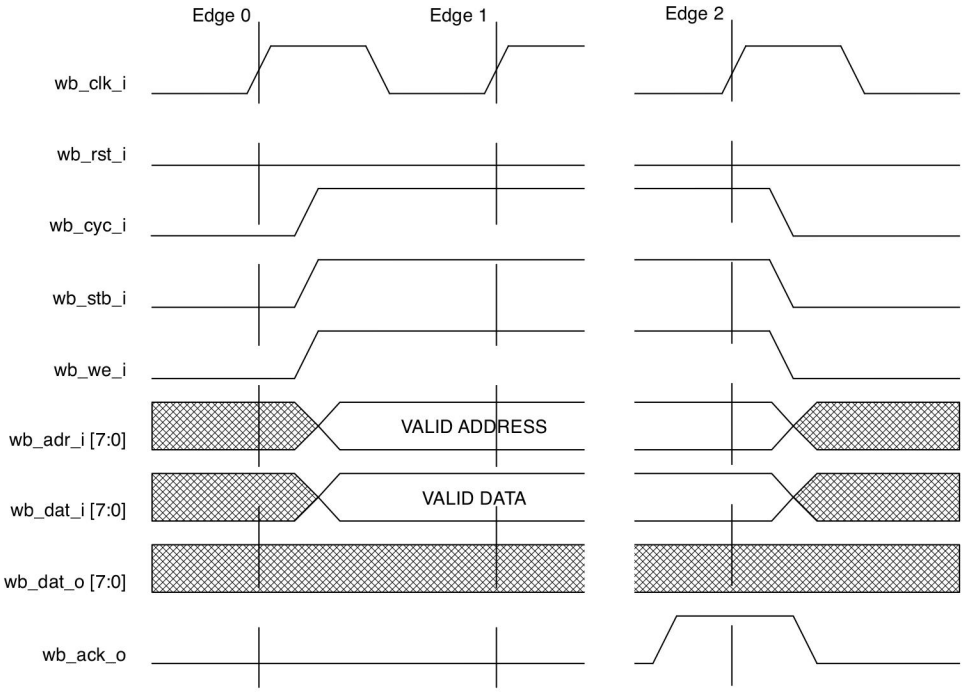
On clock Edge 1, the EFB WISHBONE Slave decodes the input signals presented by the master. During this cycle:

- The Slave decodes the address presented on the wb_adr_i[7:0] address lines
- The Slave prepares to latch the data presented on the wb_dat_i[7:0] data lines
- The Master waits for an active-high level on the wb_ack_o line and prepares to terminate the cycle on the next clock edge, if an active-high level is detected on the wb_ack_o line
- The EFB may insert wait states before asserting wb_ack_o, thereby allowing it to throttle the cycle speed. Any number of wait states may be added
- The Slave asserts wb_ack_o signal

The following occurs on clock Edge 2:

- The Slave latches the data presented on the wb_dat_i[7:0] data lines
- The Master de-asserts the strobe signal, wb_stb_i, the cycle signal, wb_cyc_i, and the write enable signal, wb_we_i
- The Slave de-asserts the acknowledge signal, wb_ack_o, in response to the Master de-assertion of the strobe signal

Figure 17-3. WISHBONE Bus Write Operation



WISHBONE Read Cycle

Figure 17-4 shows the waveform of a Read cycle from the perspective of the EFB WISHBONE Slave interface. During a single Read cycle, only one byte of data is read from the EFB block by the WISHBONE master. A Read operation requires a minimum three clock cycles.

On clock Edge 0, the Master updates the address, data and asserts control signals. The following occurs during this cycle:

- The Master updates the address on the wb_adr_i[7:0] address lines
- De-asserts the write enable wb_we_i signal, indicating a Read cycle
- Asserts the wb_cyc_i to indicate the start of the cycle
- Asserts the wb_stb_i, selecting a specific Slave module

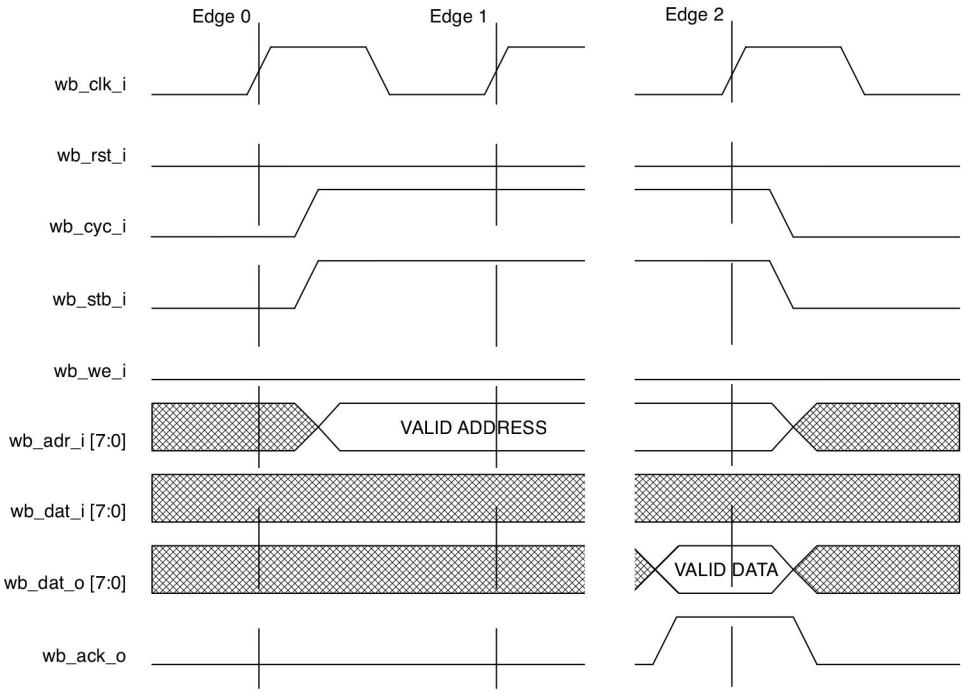
On clock Edge 1, the EFB WISHBONE slave decodes the input signals presented by the master. The following occurs during this cycle:

- The Slave decodes the address presented on the wb_adr_i[7:0] address lines
- The Master prepares to latch the data presented on wb_dat_o[7:0] data lines from the EFB WISHBONE slave on the following clock edge
- The Master waits for an active-high level on the wb_ack_o line and prepares to terminate the cycle on the next clock edge, if an active-high level is detected on the wb_ack_o line
- The EFB may insert wait states before asserting wb_ack_o, thereby allowing it to throttle the cycle speed. Any number of wait states may be added.
- The Slave presents valid data on the wb_dat_o[7:0] data lines
- The Slave asserts wb_ack_o signal in response to the strobe, wb_stb_i signal

The following occurs on clock Edge 2:

- The Master latches the data presented on the wb_dat_o[7:0] data lines
- The Master de-asserts the strobe signal, wb_stb_i, and the cycle signal, wb_cyc_i
- The Slave de-asserts the acknowledge signal, wb_ack_o, in response to the master de-assertion of the strobe signal

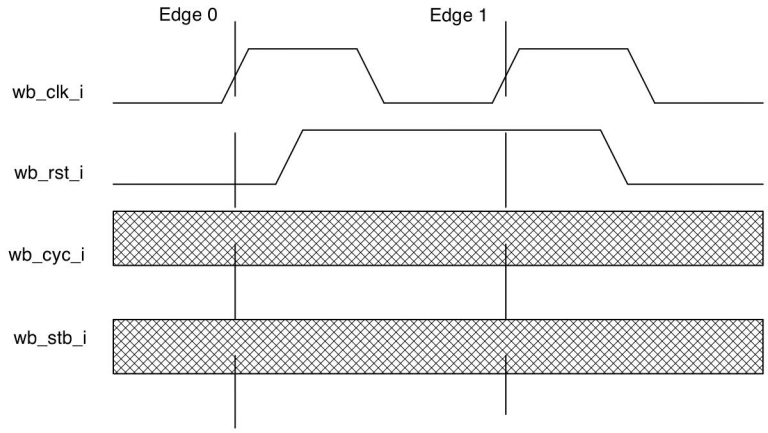
Figure 17-4. WISHBONE Bus Read Operation



WISHBONE Reset Cycle

Figure 17-5 shows the waveform of the synchronous wb_rst_i signal. Asserting the reset signal will only reset the WISHBONE interface logic. This signal will not affect the contents of any registers in the EFB register map. It will only affect ongoing bus transactions.

Figure 17-5. EFB WISHBONE Interface Reset



The wb_rst_i signal can be asserted for any length of time.