Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID.

ESCUELA DE INGENIERÍAS INDUSTRIALES.

GRADO EN INGENIERÍA EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA.

Medición de Distancia y Velocidad empleando un Sensor de Ultrasonidos.

Autor:

González Antón, Javier

Tutor:

Andrés Rodríguez Trelles, Francisco José De
Departamento de Tecnología Electrónica.

Valladolid, Junio de 2015.

ÍNDICE

ÍNDICE DE FIGURAS

Palabras clave

Ultrasonidos, FPGA (*Field Programable Gate Array****)***, ***VHDL (****Very high speed integrate circuits Hardware Description Language****), PWM (****Pulse Width Modulation****) SOC (****Sistemas disgitales en un único chip****)***

1. Objetivos del trabajo de fin de grado

En el presente trabajo de fin de grado, se pretende diseñar un sistema para la medición de distancias y velocidades empleando un sensor de ultrasonidos, y un sistema electrónico reconfigurable tipo FPGA.

Como elemento de implantación, emplearemos una FPGA del fabricante Lattice, concretamente la Mach X02 1200 ZE. Aunque actualmente existen en el mercado un gran número de alternativas de implantación, como posteriormente mostraremos, nos hemos marcado como objetivo comprobar la viabilidad de los sistemas FPGA.

Como sensor de ultrasonidos, emplearemos uno del fabricante MaxBotix, modelo LV-MaxSonar EZ3.

Se diseñara toda la lógica para lograr una solución funcional que permita obtener la distancia a la que un obstáculo, o la velocidad con la que se acerca, o aleja el mismo.

Para ello, el sistema informará de la distancia en centímetros hasta el obstáculo, mediante tres elementos de visualización, dos displays de siete segmentos, que mostrarán las unidades y decenas de centímetros, a través de números enteros comprendidos entre 0 y 9 cada display, y un array de 8 leds, que mostrarán las centenas de centímetro, a razón de un led, por cada 100 cm.

Además, se persigue cubrir los siguientes objetivos parciales:

- Profundizar en el conocimiento de los sistemas electrónicos reconfigurables del tipo FPGA.

- Profundizar en el conocimiento de los sensores de ultrasonidos y sus alternativas, analizando cuándo debe de aplicarse cada uno de ellos.

- Emplear técnicas de diseño modernas basadas tanto en lenguajes de descripción de hardware, como VHDL, como en el empleo de esquemáticos.

- Profundizar en el conocimiento de las herramientas de síntesis de circuitos, analizando cómo influyen dichas herramientas en el consumo de recursos de la FPGA, y las herramientas de simulación.

2. Introducción

2.1. Introducción a los ultrasonidos

El oído humano es capaz de captar ondas sonoras cuya frecuencia de oscilación está comprendida entre 20 Hz y 20 kHz. Así, el espectro acústico audible, puede dividirse en tres grandes grupos de tonos:

- Tonos graves: Son todos aquellos cuya frecuencia de oscilación está comprendida entre los 16 Hz y los 256 Hz.
- Tonos medios: Sus frecuencias están comprendidas entre 256 Hz y los 2 kHz.
- Tonos agudos: Con frecuencias comprendidas entre 2 kHz y 16 kHz.

Todas aquellas perturbaciones sonoras que se encuentran fuera del rango de frecuencias anteriores, se denominan ultrasonidos, si su frecuencia de oscilación es superior a los 20kHz o infrasonidos, si su frecuencia de oscilación es inferior a 20 Hz.

El estudio y control de la tecnología de los ultrasonidos tiene actualmente una gran relevancia en nuestras vidas:

- Guiado y sondeo: El empleo de los ultrasonidos a modo de radar, es una técnica usada desde hace miles de años, por animales como los murciélagos, (figura 2) que emplean para guiarse sonidos de frecuencias superiores a 100 kHz, y más recientemente, por el ser humano, por ejemplo acústica submarina para el guiado de barcos o submarinos, (figura 3) navegación autónoma de robots, sistemas de ayuda para estacionar vehículos (figura 1), etc.

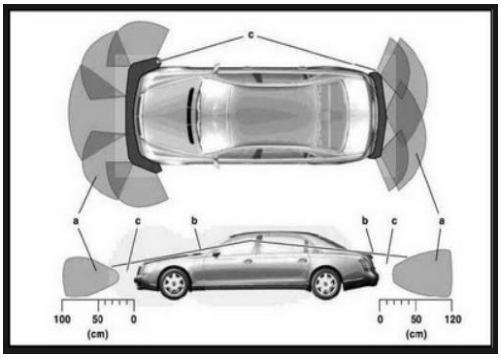


Figura 1: Parktronic

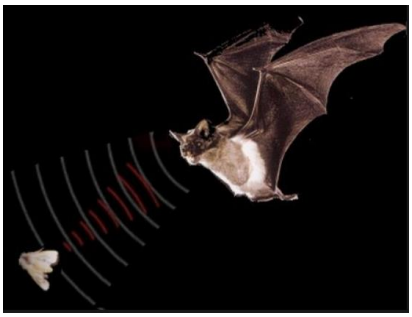


Figura 2: Muerciélago

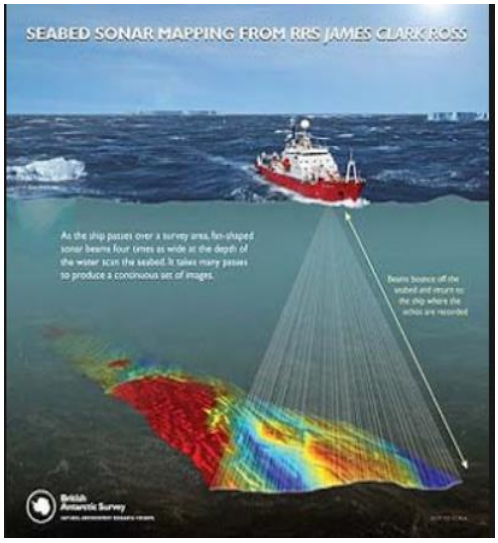


Figura 3: Sónar

- Medicina y biología: Los ultrasonidos pueden ser empleados en este campo tanto para curar ciertas dolencias, como la formación de cálculos renales, como para diagnosticar ciertas enfermedades, como es el caso de las ecografías (figura 4).

  También se están empezando a emplear los ultrasonidos para el tratamiento de obesidades, ya que al tener huesos, músculos y grasa, diferentes impedancias acústicas, nos permitirán calcula el porcentaje de cada tejido que forma nuestro cuerpo.

Figura 4: Ecografía

- Tratamiento de productos alimenticios: Desde hace unos años, se han desarrollado técnicas de tratamiento de productos alimenticios, que poco a poco van desbancando a las técnicas tradicionales como la refrigeración, ahumado, presurización, salazón, etc.

Figura 5. Conservación de alimentos

La aplicación de los ultrasonidos a este campo, recibe el nombre de "procesado mínimo de los alimentos", y su idea básica es la de destruir selectivamente los microorganismos que causan el deterioro de los alimentos, pero sin cambiar su presencia (figura 5), como ocurría con las técnicas tradicionales.

- Purificación del agua: Actualmente se encuentra en desarrollo nuevas técnicas para la purificación indirecta del agua mediante el empleo de ultrasonidos.
  Estas técnicas, se basan concretamente en generar el fenómeno de cavitación en el agua, con objetivo de que se formen burbujas en el seno de la misma (figura 6).

  Cuando estas burbujas impactan sobre los filtros de purificación, o bien sobre las partículas sólidas de impurezas en el agua, las dividen, y de esta forma, se consigue limpiar el fluido.

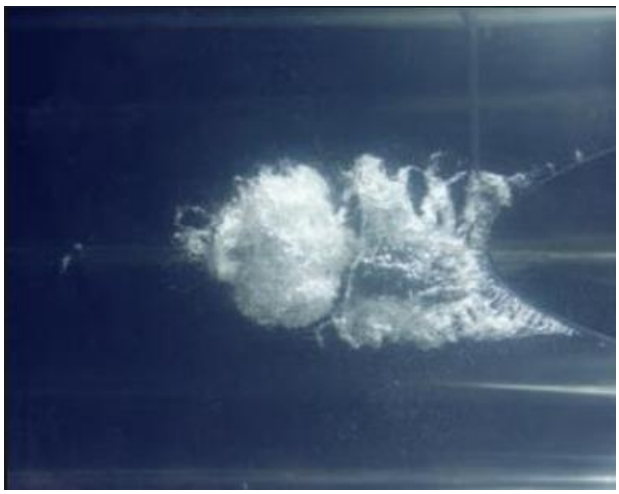


Figura 6: Burbjas por cavitación

El fenómeno de cavitación se da cuando el número de Reynolds, (Re), aumenta por encima de 2300. Éste parámetro, depende de diversos factores como:

$$Re = \frac{\rho\, V\, L}{\mu}$$

Como podemos ver en la ecuación anterior, ρ y μ, densidad y viscosidad dinámica del agua, son constantes para un cierto fluido, a una cierta temperatura. L, también se puede considerar constante, al ser la longitud de la tubería por la que discurre el fluido.

De esta forma, el parámetro variable, será V, la velocidad del fluido.

- Aplicaciones físicas: En la industria física, la aplicación de los ultrasonidos, se basa en el conocimiento de los materiales, y la medida de sus propiedades elásticas, condiciones de propagación de las ondas sonoras, cantidad de energía que un material puede propagar, estudio de resonancias, etc.
  Es precisamente el estudio de la frecuencia de resonancia, una de las aplicaciones más importantes en este campo. Por dicho efecto, por ejemplo se derrumbó el puente de Tacoma Narrows (Figura 7).

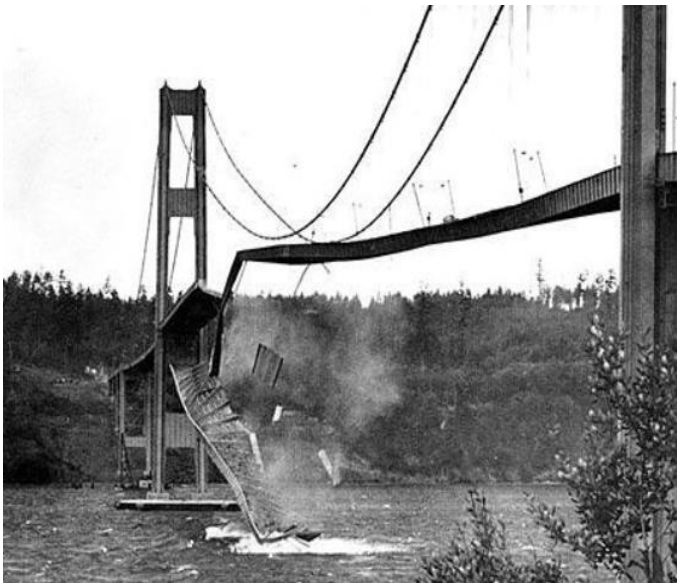


Figura 7: Puente Tacoma Narrows en resonancia

- Aplicaciones químicas: La aplicación fundamental de los ultrasonidos en el campo de la química, es la de servir como catalizador, activando en un momento determinado, ciertos reactivos que cambien el curso de una reacción química.

- Aplicaciones técnicas: La aplicación de los ultrasonidos a la industria moderna es enorme. Estos se emplean en aplicaciones tan variopintas, como desde la detección de defectos en piezas metálicas, hasta la apertura automática de puertas.

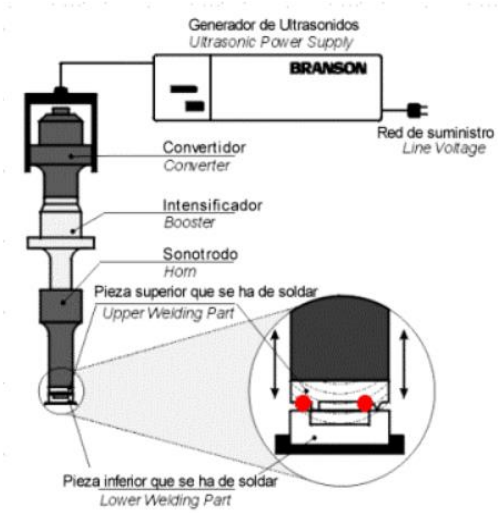


Figura 8: Soladura por ultrasonidos

Sin embargo, una de las aplicaciones más importantes, es la "Soldadura de plásticos por ultrasonidos", figura 8.

La soldadura por ultrasonidos, permite unir dos piezas plásticas en una sola, sin necesidad de aportar un fundente, y sin necesidad de precalentar a temperaturas elevadas.

Este tipo de soldadura, es mucho más rápida, económica, y además no genera contaminantes.

Su funcionamiento, se basa en convertir los ultrasonidos en energía calorífica que funda y suelde los plásticos.

2.2. Precauciones en la medición por ultrasonidos

A pesar de que su funcionamiento parece muy sencillo, existen factores inherentes tanto a los ultrasonidos como al mundo real, que influyen de una forma determinante en las medidas realizadas. Por tanto, es necesario un conocimiento de las diversas fuentes de incertidumbre que afectan a las medidas para poder tratarlas de forma adecuada, minimizando su efecto en el conocimiento del entorno que se desea adquirir. Entre los diversos factores que alteran las lecturas que se realizan con los sensores de ultrasonido cabe destacar:

- El campo de actuación del pulso que se emite desde un transductor de ultrasonido tiene forma cónica (figura 9). El eco que se recibe como respuesta a la reflexión del sonido indica la presencia del objeto más cercano que se encuentra dentro del cono acústico y no especifica en ningún momento la localización angular del mismo. Aunque la máxima probabilidad es que el objeto detectado esté sobre el eje central del cono acústico, la probabilidad de que el eco se haya producido por un objeto presente en la periferia del eje central no es en absoluto despreciable y ha de ser tenida en cuenta y tratada convenientemente.

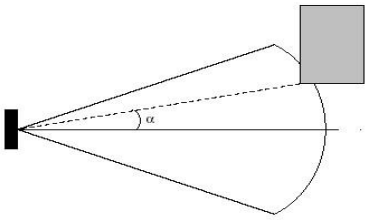


Figura 9: Perfil del campo de ultrasonidos

- La cantidad de energía acústica reflejada por el obstáculo depende en gran medida de la estructura de su superficie (figura 10). Cuando las irregularidades de la superficie reflectora, son comparables a la longitud de onda de la onda de ultrasonido incidente, se producirá una reflexión difusa de los ultrasonidos. En esta ocasión, la cantidad de energía incidente en el receptor, será muy baja, y la detección del obstáculo, se complica.

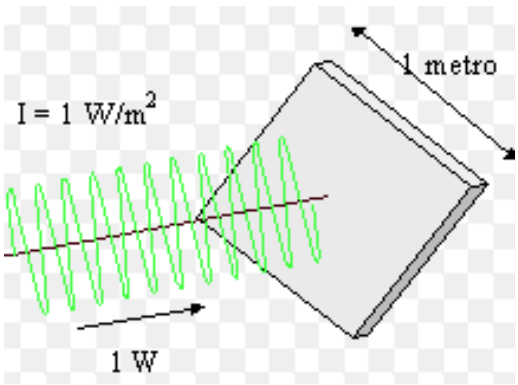


Figura 10: Absorcion energética de un cuerpo

- En los sensores de ultrasonido de bajo coste se utiliza el mismo transductor como emisor y receptor. Tras la emisión del ultrasonido se espera un determinado tiempo a que las vibraciones en el sensor desaparezcan y esté preparado para recibir el eco producido por el obstáculo. Esto implica que existe una distancia mínima distancia (proporcional al tiempo de relajación del transductor) a partir de la cual el sensor mide con precisión. Por lo general, todos los objetos que se encuentren por debajo de esta distancia, d, serán interpretados por el sistema como que están a una distancia igual a la distancia mínima. Véase la figura 11.

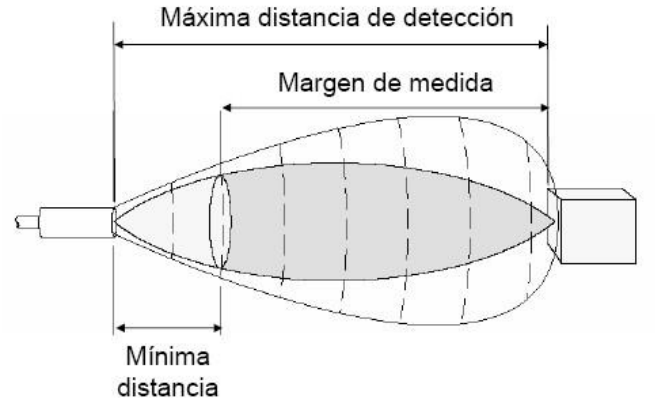


Figura 11: Gradiente térmico

- Los factores ambientales tienen una gran repercusión sobre las medidas: Las ondas de ultrasonido se mueven por un medio material que es el aire. La densidad del aire depende de la temperatura, influyendo este factor sobre la velocidad de propagación de la onda según la expresión:

$$V_{PROPAGACION}(T) = V_{PROPAGCACION}(273.15\ K)\sqrt{1 + \frac{T[K]}{273.15}}$$

Ademas, la direccion de propagacion de los ultrasonidos se ve afectada por los gradientes térmicos (figura 12).

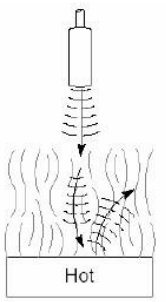


Figura 12: Diagrama de distamcias

- Un factor de error muy común es el conocido como falsos ecos. Estos falsos ecos se pueden producir por razones diferentes: Puede darse el caso en que la onda emitida por el transductor se refleje varias veces en diversas superficies antes de que vuelva a incidir en el transductor (si es que incide). Este fenómeno, conocido como reflexiones múltiples, implica que la lectura del sensor evidencia la presencia de un obstáculo a una distancia proporcional al tiempo transcurrido en el viaje de la onda; es decir, una distancia mucho mayor que a la que está en realidad el obstáculo más cercano, que pudo producir la primera reflexión de la onda. Otra fuente más común de falsos ecos, conocida como crosstalk (figuras 13 y 14), se produce cuando se emplea un cinturón de ultrasonidos donde una serie de sensores están trabajando al mismo tiempo. En este caso puede ocurrir (y ocurre con una frecuencia relativamente alta) que un sensor emita un pulso y sea recibido por otro sensor que estuviese esperando el eco del pulso que él había enviado con anterioridad (o viceversa).

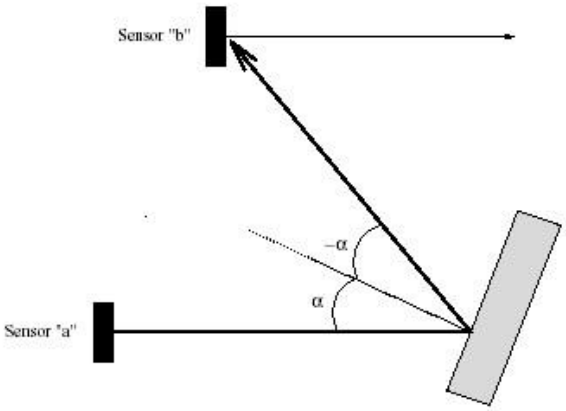


Figura 14: Efecto corsstalk (1)

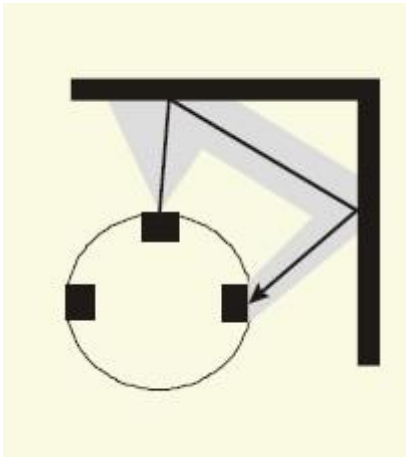


Figura 13: Efecto crosstalk (2)

- Las ondas de ultrasonido obedecen a las leyes de reflexión de las ondas, por lo que una onda de ultrasonido tiene el mismo ángulo de incidencia y reflexión respecto a la normal a la superficie. Esto implica que si la orientación relativa de la superficie reflectora con respecto al eje del sensor de ultrasonido es mayor que un cierto umbral, el sensor nunca reciba el pulso de sonido que emitió. Este efecto se muestra en la figura 15.

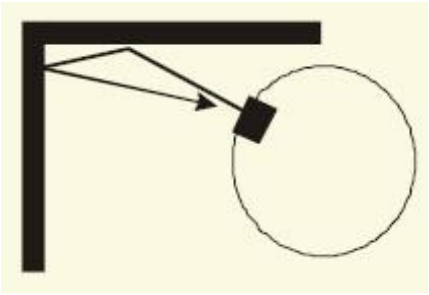


Figura 15: Reflexión de ondas

Además de los problemas ya señalados con más detalle anteriormente a continuación se muestran de una manera más esquemática otras situaciones que pueden ser problemáticas a la hora de diseñar un sistema de detección de distancias, para un robot, basado en ultrasonidos (figura 16):

- La posición real del objeto es desconocida. (Cualquier posición del cono a distancia d).
- Reflejos especulares: La dirección del reflejo depende del ángulo de Incidencia
- Cuanto menor sea el ángulo, mayor es la probabilidad de perderse y producir falsas medidas de gran longitud
- Las superficies pulidas agravan el problema (las rugosas producen reflejos que llegan antes)
- Ejemplo: un robot que se acerca a una pared con muy poco ángulo puede "no verla"
- Si un gran número de robots emplean ultrasonidos, puede existir un problema de falsa detección.

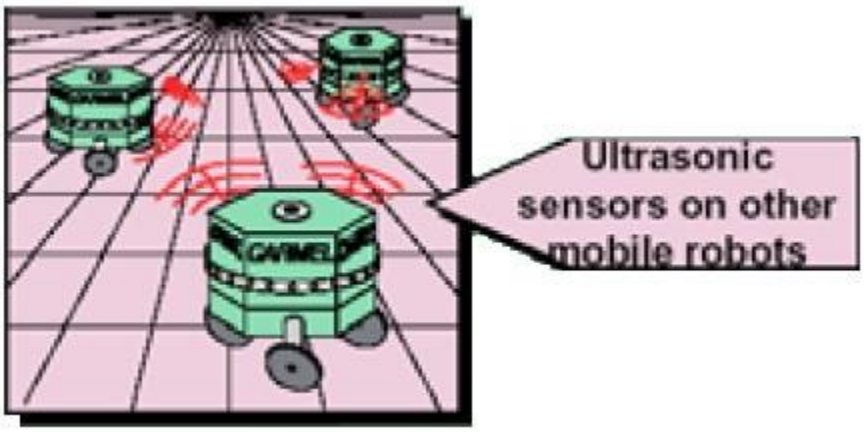


Figura 16: Robots guiados por ultrasonidos

2.3: Alternativas a los ultrasonidos

Actualmente, la medición de distancias sin contacto, no sólo se puede realizar empleando ultrasonidos. Las alternativas más recomendables al uso de los sensores de ultrasonidos, son los sensores de infrarrojos, y los sensores láser.

2.3.1. Sensores de infrarrojos

Un sensor de distancia por infrarrojos, (figura 17) utiliza un haz de ondas electromagnéticas, cuya longitud de onda, corresponde al espectro del infrarrojo, para medir las distancias.

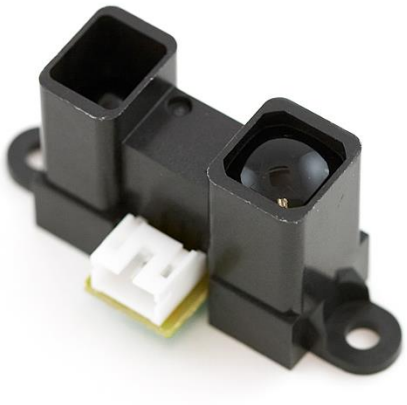


Figura 17: Sensor de distancia por infrarrojos

El principio de funcionamiento, es similar al de los ultrasonidos, sólo que en lugar de emitir un haz de ultrasonidos, y esperar su rebote, se emite un haz de infrarrojos y se espera su rebote.

A su vez, los sensores de infrarrojos se pueden dividir en los activos, formados por un LED infrarrojo como emisor y un fototransistor como receptor, y los pasivos, formados exclusivamente por un fototransistor, y destinados a recibir las radiaciones emitidas por otros dispositivos.

Las características, más destacables de dichos sensores son:

VENTAJAS:

Son muy precisos para la medición de pequeñas distancias. Normalmente se suelen emplear para la medición de distancias máximas de unas pocas decenas de centímetros.

Son sensores relativamente baratos.

INCONVENIENTES:

Su alcance es muy limitado. Por poner un ejemplo, en el caso de los sensores activos, el alcance puede ser de entre 8 a 15 cm.

Sólo detecta la dirección al obstáculo normal al sensor. Éste tipo de sensores se caracterizan por detectar únicamente el obstáculo que se encuentra delante de ellos, es decir, son direccionales. Esto, sin embargo, dependiendo de la aplicación no tiene por qué ser una desventaja.

2.3.2. Sensores láser

Los sensores láser (figura 18) se componen de un emisor y un receptor. El láser calcula distancias enviando un pulso de radiación láser no visible. Cuando este pulso encuentra un objeto en su camino se refleja. El reflejo vuelve al dispositivo, que lo detecta empleando un foto receptor láser. La diferencia de tiempo entre el haz emitido y el haz reflejado permite calcular la distancia a la que se encuentra el objeto.

Figura 18. Sensor de distancia por Láser

El láser tiene el mismo principio de funcionamiento del sensor de ultrasonidos, o del sensor infrarrojo. La gran diferencia del láser respecto a estos sistemas es el tipo de radiación empleada.

En éste sensor, se utiliza una radiación a una longitud de onda determinada y un sistema de lentes que dirigen esa radiación de forma precisa (formando en definitiva un rayo láser).

VENTAJAS:

El láser es mucho más preciso que el sónar, o que el infrarrojo. A todos los efectos, mientras que los ultrasonidos y el infrarrojo acumulan un arco de incertidumbre que crece al propagarse, el láser traza una línea delgada y prácticamente no se dispersa con la distancia. Por ello, ofrece una localización del obstáculo muchísimo más exacta.

El láser tiene mucho más alcance que el infrarrojo, debido a que ésta radicación no tiende a dispersarse. Puede recorrer mayores distancias, incluso de hasta 20 km. Los ultrasonidos, por otro lado, pueden tener alcances parecidos al láser, aunque la diferencia de precisión muy elevada.

DESVENTAJAS:

El láser también es mucho más complicado de fabricar, lo que lo hace mucho más caro.

También tiene un mayor consumo energético.

2.4. Alternativas de implantación

Actualmente, podemos barajar distintos dispositivos para la creación de nuestro diseño.

xPLD: En este campo, podemos encontrarnos con dos variantes fundamentales, los SPLD (Simple Programmable Logic Device) y los CPLD (Complex Programmable Logic Device).

Ambos dispositivos, están construidos en base a dos planos lógicos, un plano de sumas (OR) y un plano de productos (AND). La estructura más común es la de SUMA de PRODUCTOS. A continuación, en la figura 19, mostramos la estructura de una PAL22V10. Ésta, es la estructura de una SPLD implementada como suma de productos.

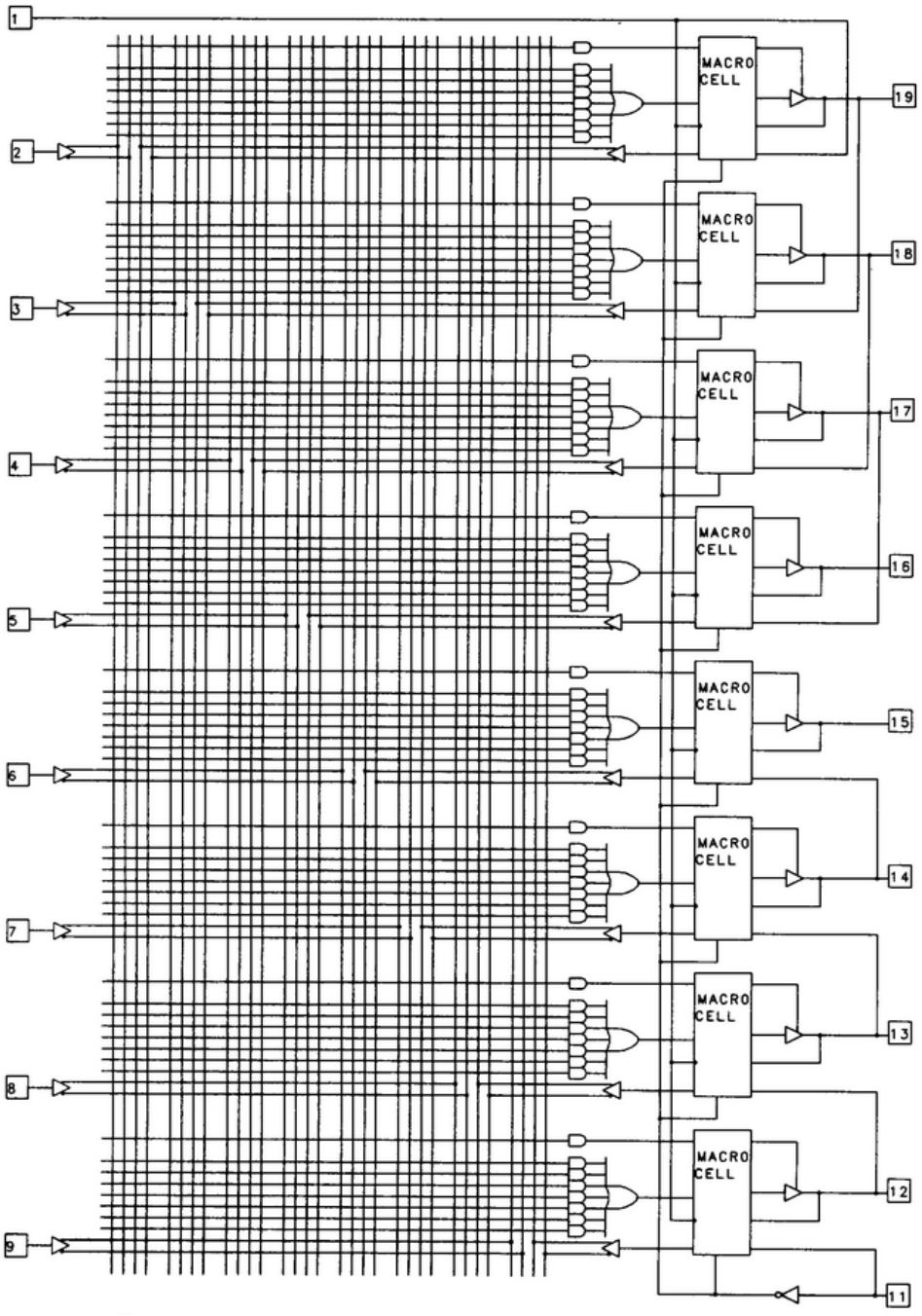


Figura 19: Diagrama lógico de una PAL22V10

Donde el bloque de MACROCELL, representa la macro-celda asociada a cada salida, la cual nos permite, realizar realimentaciones, seleccionar si un pin actúa como entrada o salida, elegir entre salidas combinacionales o registradas, etc.

La diferencia entre hablar de un dispositivo simple, o de uno complejo, es la capacidad de lógica que pueden incorporar. Los dispositivos tipo CPLD, incorporan una capacidad de lógica muy superior a los SPLD.

A su vez, estos dispositivos se pueden dividir en dos clases, los dispositivos con estructura tipo PAL, y los dispositivos con estructura tipo PLA, de acuerdo, a la flexibilidad de sus planos lógicos:

Un dispositivo tipo PAL (figura 20), está constituido por un plano AND programable, y un plano OR fijo. Esta configuración le hace ser menos flexible, pero más rápido, ya que incorpora menos transistores como interruptores

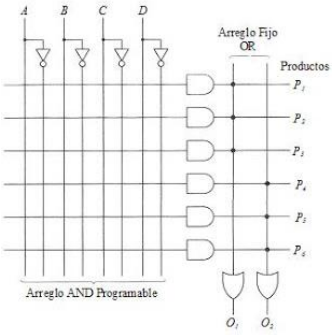


Figura 20: Estructura tipo PAL

Un dispositivo tipo PLA (figura 21), está constituido por ambos planos programables. Esto les hace ser más lentos, pero mucho más flexibles

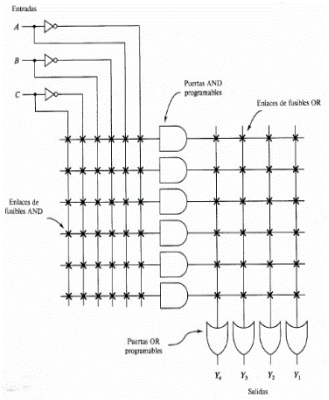


Figura 21: Estructura tipo PLA

Como podemos ver en ambos esquemas, en un dispositivo xPLD, ya sea de tipo PAL, o PLA, la lógica se hace por el cierre de conexiones.

Por esta razón, se suele decir que estos dispositivos implementan lógica en base a fusibles. Inicialmente las conexiones se hacían fundiendo todos aquellos fusibles cuya conexión no se deseaba (figura 22).

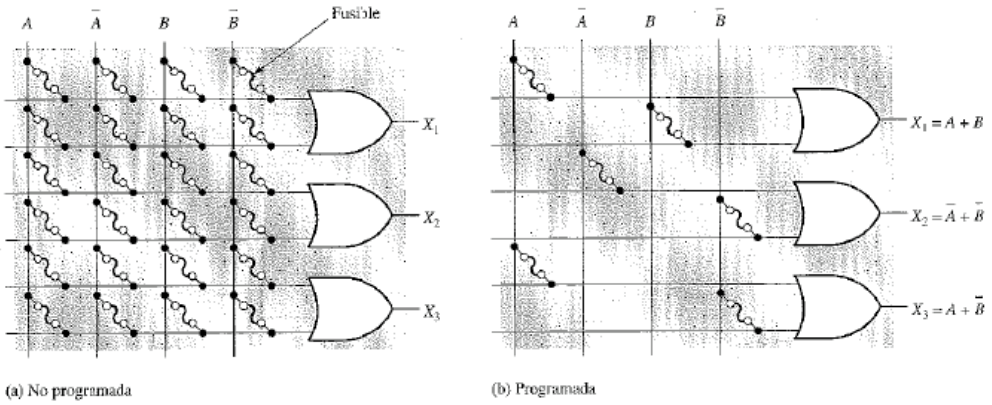


Figura 22: Programacion de PLD's por fusibles

Actualmente, los fusibles han sido sustituidos por transistores MOS de puerta aislada, (figura 23) lo cual da más fiabilidad al dispositivo, y le permite ser reconfigurable.

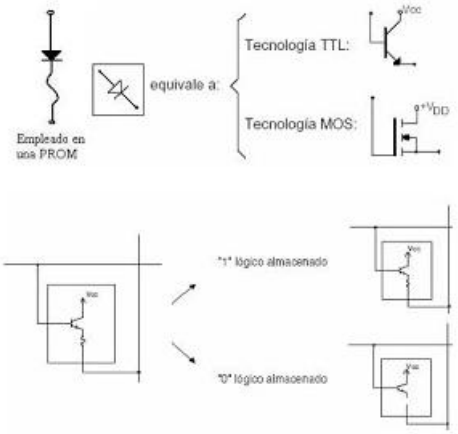


Figura 23: Programacion de PLD's por transistores MOS

FPGA: En términos generales de uso, una FPGA (Field Programmable Gate Array), es equivalente en todos los aspectos, a una CPLD. La diferencia entre ambos dispositivos es la forma de implementar la lógica.

Mientras como hemos visto, en una CPLD, la lógica se implementa en base a la activación/desactivación de conexiones, en uno o dos planos lógicos, en una FPGA, la lógica se implementa en base a memorias. Ello, no cambia la aplicación para la que se pueden emplear dichos dispositivos, pero si su filosofía de diseño.

Las FPGA, son dispositivos de grano fino, con mucha capacidad de memoria, pero poca de lógica, mientras que las CPLD, son dispositivos de grano grueso, con poca capacidad de memoria, pero mucha de lógica.

Ello hace, que una FPGA, sea más atractiva para diseños segmentados o de tipo PIPELINE.

Podemos decir a grandes rasgos, que una FPGA, está compuesta por multitud de pequeños elementos de memoria interconectables (figura 24). Cada uno de dichos elementos de memoria, podrá implementar lógica. Con una memoria de X líneas de direccionamiento, podemos implementar cualquier función lógica de X variables.

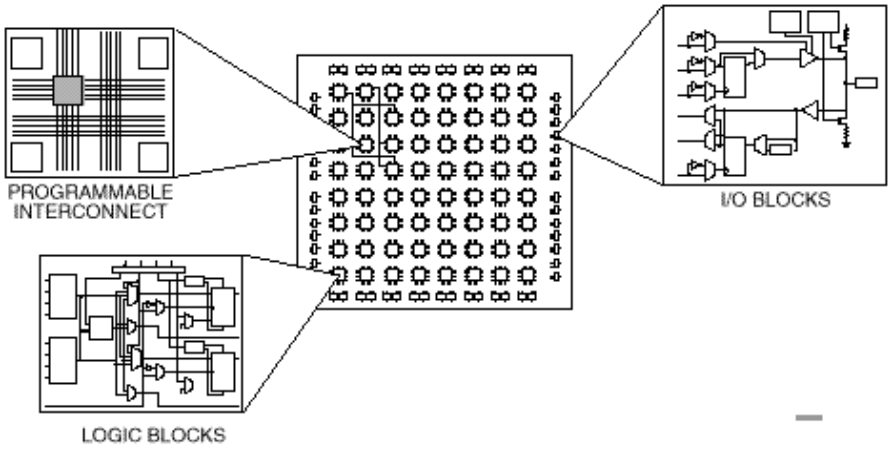


Figura 24: Diagrama de un dispositivo FPGA

Microcontroladores: Podemos definir un microcontrolador, como un circuito integrado de alta escala de integración, que incorpora en un único chip de Silicio o Germanio, todos los elementos que configuran un controlador (figura 25).

Los microcontroladores, se han convertido actualmente en la mejor solución para el control de procesos, ya que minimizan el tamaño, y el coste, manteniendo, o incluso mejorando las prestaciones de los controladores clásicos.

Un microcontrolador, dispone normalmente de los siguientes elementos:

- Procesador o CPU, formada por la unidad de control UC, unidad aritmético lógica ALU, y registros.
- Memoria RAM
- Memoria de programación, de tipo NO volátil, como EPROM, EEPROM, ROM…
- Líneas de entrada u salidas para su comunicación con el exterior
- Módulos para el control de periféricos, como conversores A/D, o D/A, temporizadores, contadores, moduladores de anchura de pulso, PLL, etc.

Los microcontroladores, son dispositivos de arquitectura cerrada, ya que no admiten modificaciones de tipo hardware. A continuación se muestra un grafico que representa la aquitectura de un microcontrolador

Los lenguajes de programación más comunes de los microcontroladores, son C y Basic, para un nivel medio de abstracción, y ensamblador, para un nivel bajo de abstracción. Mediante el lenguaje ensamblador, se puede programar usando directamente el juego de instrucciones

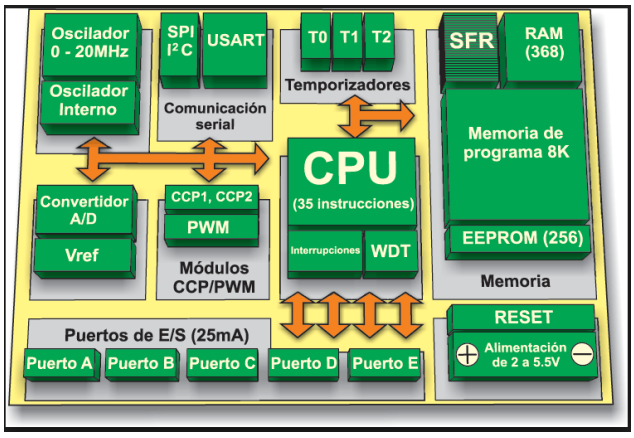


Figura 25: Diagrama de un microcontrolador

Los microcontroladores, suelen incorporar dos tipos distintos de juegos de instrucciones, juego RISC o reducido de instrucciones, y juego CISC o complejo de instrucciones.

Actualmente, la aplicación de los microcontroladores es inmensa en prácticamente todas las aplicaciones de control de procesos. Uno de los mayores fabricantes de microcontroladores en la actualidad es Microchip, con su familia PIC, como el PIC 16F876x (figura 26). Microcontroladores de este tipo, se pueden observar en aplicaciones tan dispares como son desde el control de las revoluciones del motor de una lavadora, hasta el control del programa de estabilidad de un vehículo.

Figura 26: microcontrolador comercial PIC16F876A

Microprocesadores: Los microprocesadores, aparecen y se popularizan a partir de año 1970. Se denomina microprocesador a un circuito integrado LSI o de baja escala de integración, que contiene todos los elementos que constituyen la llamada unidad central de proceso o CPU (figura 27).

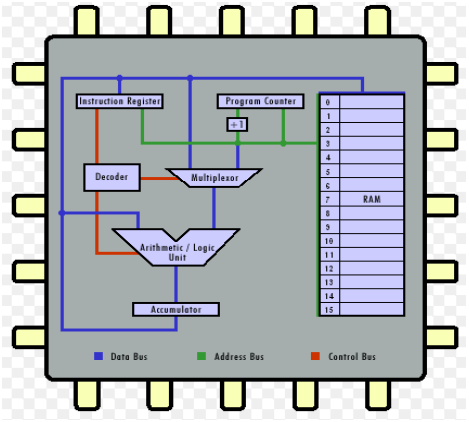


Figura 27: Diagrama de un microprocesador

A diferencia de un microcontrolador, quien por si sólo es perfectamente funcional, un microprocesador, no es funcional por sí solo. El microprocesador precisa de memoria de trabajo o RAM, memoria no volátil de almacenamiento del programa y los módulos de entradas y salidas para comunicarse.

Los microprocesadores, permiten proporcionar una base común para la mayor parte de aplicaciones de control, que el usuario, puede personalizar. Esta es una arquitectura de control abierta, en contraposición a la arquitectura de los microcontroladores.

Tanto los microprocesadores, como los microcontroladores, se pueden clasificar en función del número de bits de las instrucciones con las que trabajan. Podemos encontrarnos microcontroladores y microprocesadores, desde los 4 hasta los 64 bits (4, 8, 16, 32, 64).

Hoy en día, los más comunes son microcontroladores de 4 u 8 bits, empleados en aplicaciones sencillas de control, como electrodomésticos, etc, y microprocesadores de 16 o 32 bits, para aplicaciones complejas de control, como el ABS de un vehículo, etc. Al primer grupo, pertenece el soporte Arduino, en todas sus variantes, salvo en Arduino DUE, que pertenece al segundo.

ASIC's: Éstos son circuitos integrados hechos a medida para una determinada aplicación. Actualmente, podemos encontrarnos con circuitos ASIC que oscilan desde SSI o de pequeña escala de integración, hasta ULSI, o de ultra gran escala de integración.

Un circuito ASIC, es un circuito completamente personalizado por el diseñador. Éste, ha descrito el comportamiento del mismo, y lo ha diseñado al completo.

Para ello, ha tenido que realizar las máscaras de fabricación, como las de difusiones, metalización, atacado, como las mostradas en la figura 28 (el número de máscaras y el tipo de las mismas depende del ASIC concreto) ha tenido que determinar el tipo y cantidad de dopantes, la pureza del silicio, el tipo y material del encapsulado, etc. En definitiva, es una técnica para fabricar un circuito integrado a medida, donde se puedan modificar y controlar todos y cada uno de los aspectos del mismo.

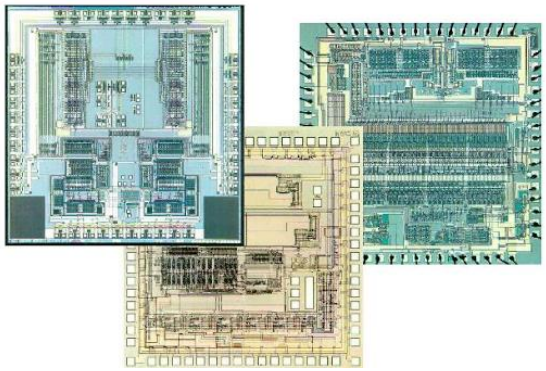


Figura 28: Máscaras de fabricación de circuitos ASIC's

Ésta técnica sólo tiene sentido para la fabricación de grandes series de circuitos, y no para el diseño de prototipos, ya que los costes económicos, e intelectuales, no lo permiten así.

Otros sistemas embebidos: Existen otras alternativas, que normalmente se basan en una combinación de microprocesadores y FPGA's, como es el caso de sistemas embebidos como el MyRIO fabricado por National Intruments®.

Éste sistema se programa bajo LabVIEW (figura 29). Éste se trata de un lenguaje de programación de alto nivel de abstracción, en el que se emplean diagramas de bloques.

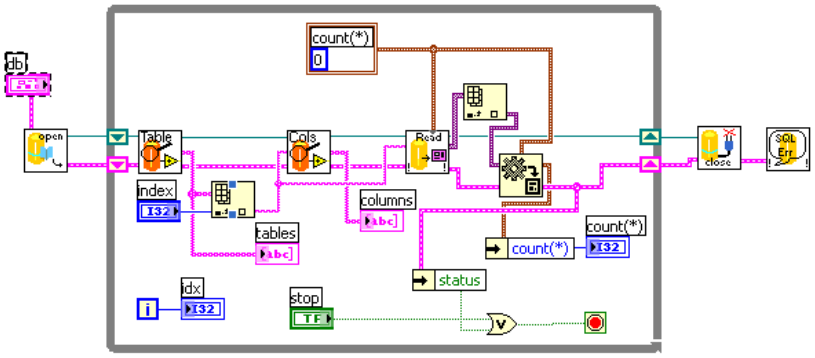


Figura 29: Esquema diseñado bajo LabVIEW

La gran ventaja de LabVIEW, es precisamente su alto nivel de abstracción, que permite programar funciones y comportamientos muy complejos con apenas unos cuantos bloques.

Estos sistemas permiten realizar prácticamente cualquier diseño, proporcionando mucha libertad, y potencia. Además, como es el caso del MyRIO, (figura 30) incorporan protocolos de comunicación listos para usar, como la conexión vía WiFi con elementos remotos, vía USB, etc.

El problema básico de los mismos, es su elevado precio en la actualidad.

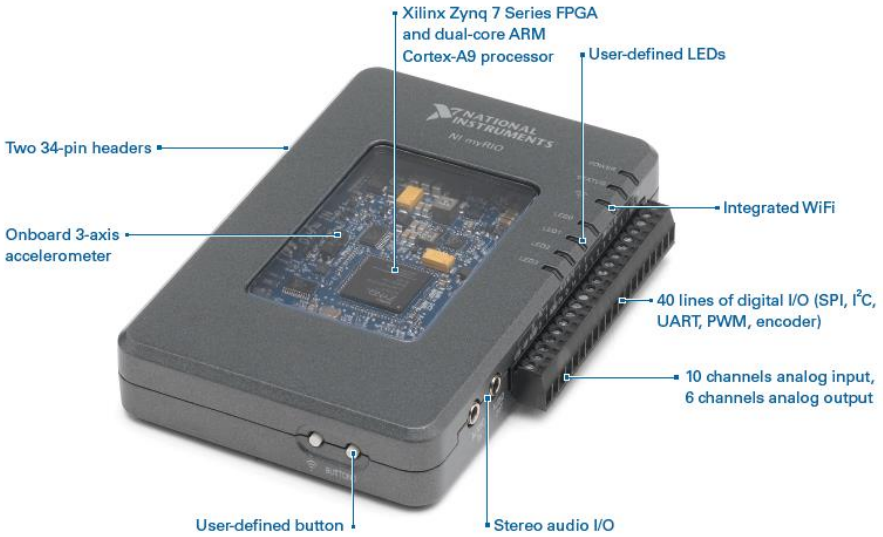


Figura 30: Dispositivo comercial MyRIO de National Instruments

En nuestro caso cualquiera de las alternativas anteriores de implantación, sería válida. No obstante, nos decantaremos por el uso de una FPGA, ya que una de las metas u objetivos de este TFG, es comprobar la viabilidad de la implantación del sistema en una FPGA.

Dentro de todos los fabricantes y modelos de FPGA, nos decantaremos por una MACHX02 1200 o por una MACHX02 7000, ya que son los dos dispositivos de los que se dispone en el departamento de tecnología electrónica.

Elegiremos la MACHX02 1200, ya que la capacidad de lógica que posee es suficiente para la implantación de nuestro sistema. Emplear una 7000, sería infravalorar gravemente sus recursos.

2.5. Dispositivos FPGA

Una FPGA (Field Programable Gate Array), es un dispositivo electrónico reconfigurable capaz de implementar la lógica, mediante memorias, y no mediante "fusibles" o conexiones reprogramables como en el caso de los dispositivos xPLD. Recodemos, que una memoria de X líneas de direcciones, puede realizar cualquier función lógica de X variables.

2.5.1. Comparativa FPGA vs CPLD

Son dispositivos lógicos reconfigurables de similares características en cuanto a su potencia de lógica se refiere son las FPGA's y las CPLD's (Dispositivos lógicos programables complejos)

Podemos distinguir dos grandes diferencias en torno a estos dos:

1. En relación a su naturaleza: Una FPGA, es un dispositivo reconfigurable de grano fino, es decir, es un dispositivo que posee más capacidad de almacenamiento que de lógica, contrariamente a los dispositivos xPLD, los cuales eran de grano grueso, o con más capacidad de lógica que de almacenamiento.
   Es decir, en una FPGA, la lógica se implementa en base a memorias, mientras que en un CPLD, la lógica se implementa en base a puertas lógicas de tipo AND, u OR.

2. En relación a su velocidad: Ambos dispositivos pueden llegar a alcanzar velocidades similares, y muy elevadas.

3. Volatilidad: Tradicionalmente los dispositivos tipo CPLD, han sido NO VOLÁTILES, mientras que los dispositivos tipo FPGA han sido VOLÁTILES.
   Sin embargo, hace ya años, que esta "norma" ha quedado relegada, ya que se han diseñado desde hace algunos años FPGA's no volátiles.

4. Seguridad: La seguridad, es un aspecto intrínseco a la volatilidad. Un dispositivo reconfigurable NO VOLÁTIL, será seguro, mientras que uno VOLÁTIL será no seguro.
   Tradicionalmente, se ha considerado a las FPGA, como dispositivos no seguros. Ello se debe a que en sus inicios, los dispositivos tipo FPGA, eran volátiles por lo que en cada arranque es necesario, pasar el fichero de configuración de una memoria ROM externa al dispositivo. En esta transferencia, dicho fichero, puede ser copiado.

   Por su parte, los dispositivos tipo CPLD, siempre se han considerado seguros, ya que al ser no volátiles, prescinden de dicha ROM.

   Todo esto actualmente ha cambiado, ya que se han conseguido desde hace años FPGA's no volátiles.

2.5.2. Familia MACHX02

Para el desarrollo de nuestro proyecto, vamos a decantarnos por utilizar una FPGA del fabricante Lattice Semiconductor® (http://www.latticesemi.com).

De la gama de dispositivos de dicho fabricante, vamos a trabajar con la FPGA MACHX02 – 1200 ZE con empaquetado de montaje superficial de 144 pines cuadrado, de 20 mm de lado.

MACHX02	**1200**	**ZE**
Familia	1200 LUT 4	Extra bajo consumo

Algunas de las características más importantes de nuestra MACHX02 - 1200 ZE son las siguientes:

1. Arquitectura lógica flexible. La MACHX02, se comercializa en 6 variantes distintas, con entre 256 y 6864 LUT4 y de 18 a 334 entradas/salidas.

2. Gran diversidad de empaquetados, desde los empaquetados de 2.5 mm X 2.5 mm de 25 pines, hasta los empaquetados de 20 mm X 20 mm de 144 pines.

3. Incorpora modo de ahorro de energía. Además, se fabrica en la variante ZE, o de consumo de energía ultra bajo. La variante ZE, consigue un consumo de energía prácticamente 0 en modo standby.

4. Permite implementar tanto lógica registrada, como lógica combinacional.

5. Intrínsecamente segura, puesto que es NO VOLÁTIL, y no precisa de elementos adicionales para su arranque, como memorias ROM de apoyo.

6. Diversidad de memorias integradas. La MACHX02 integra 138 kbits de memoria distribuidos del siguiente modo:

    - 64 kb de memoria Flash. Memoria no volátil, de propósito general, que permite al usuario, poder almacenar información propia.

    - 64 kb de memoria RAM dedicada: La memoria RAM dedicada, sólo puede ser usada como almacenamiento volátil de datos. Dicho almacenamiento puede ser empleado, cuando se implementa por ejemplo un microprocesador en la FPGA. Lattice, dispone de arquitecturas libres de microprocesadores, como por ejemplo el MICO8.

    - 10 kb de memoria RAM distribuida: La memoria RAM distribuida, puede ser empleada, tanto para memoria RAM, como en el caso de la memoria RAM dedicada, como para la implementación de lógica.

En ambos casos, la memoria RAM, puede ser usada, por ejemplo para realizar un histórico de los datos tomados, almacenar eventos que han ocurrido en el sistema, etc.

7. Permite actualizar su configuración en funcionamiento. La MACHX02, ha sido diseñada para que pueda cambiarse su configuración, sin necesidad de detener su funcionamiento. Para ello, dispone de un espacio de memoria flash especial, llamado ON CHIP CONFIGURATION FLASH MEMORY.

   La misión de esta es almacenar los nuevos bits de configuración, y volcarlos en la FPGA en el momento del nuevo arranque. Ello permite que mientras que la FPGA esté en funcionamiento pueda ser actualizada, y en el momento del reinicio, pueda cargarse la nueva configuración.

8. Alimentación: La MACHX02 permite funcionar con diferentes niveles de tensión. El núcleo de la FPGA, puede funcionar a tensión reducida de 1.2 V, mientras que su periferia, que interacciona con dispositivos externos, puede funcionar a tensión de 3.3 V.
   Ésta, es una opción que favorece la eficiencia energética, pero no está implementada en todas las FPGA's.

9. Gran variedad de tensiones de entrada y salidas soportadas:

   - ENTRADAS SOPORTADAS: Véase la figura 31, proporcionada por Lattice.

	VCCIO (Typ.)				
Input Standard	**3.3 V**	**2.5 V**	**1.8 V**	**1.5**	**1.2 V**
Single-Ended Interfaces					
LVTTL	✓	✓[2]	✓[2]	✓[2]	
LVCMOS33	✓	✓[2]	✓[2]	✓[2]	
LVCMOS25	✓[2]	✓	✓[2]	✓[2]	
LVCMOS18	✓[2]	✓[2]	✓	✓[2]	
LVCMOS15	✓[2]	✓[2]	✓[2]	✓	✓[2]
LVCMOS12	✓[2]	✓[2]	✓[2]	✓[2]	✓
PCI[1]	✓				
SSTL18 (Class I, Class II)	✓	✓	✓		
SSTL25 (Class I, Class II)	✓	✓			
HSTL18 (Class I, Class II)	✓	✓	✓		
Differential Interfaces					
LVDS	✓	✓			
BLVDS, MVDS, LVPECL, RSDS	✓	✓			
MIPI[3]	✓	✓			
Differential SSTL18 Class I, II	✓	✓	✓		
Differential SSTL25 Class I, II	✓	✓			
Differential HSTL18 Class I, II	✓	✓	✓		

1. Bottom banks of MachXO2-640U, MachXO2-1200/U and higher density devices only.
2. Reduced functionality. Refer to TN1202, MachXO2 sysIO Usage Guide for more detail.
3. These interfaces can be emulated with external resistors in all devices.

Figura 31: Niveles lógicos de las entradas en una MACHX02 1200 ZE

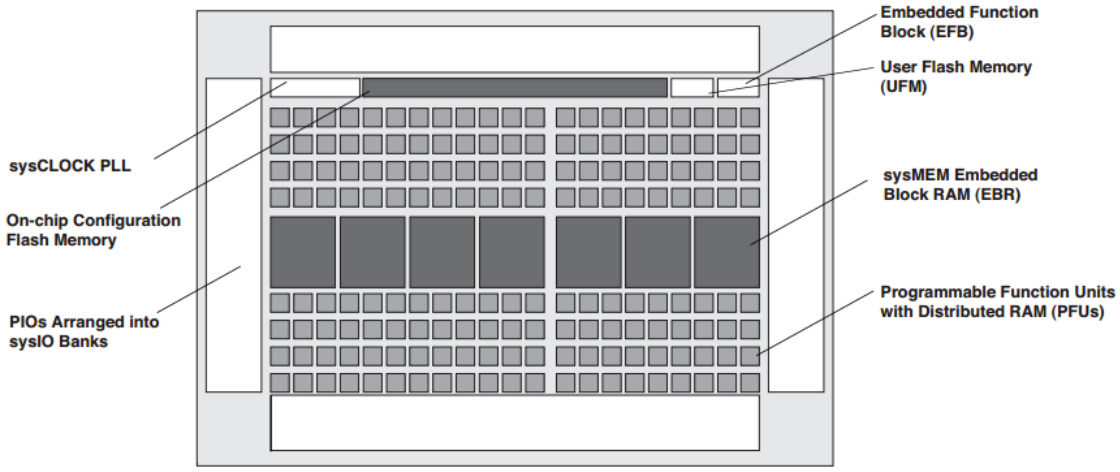
- SALIDAS SOPORTADAS: Véase la figura 32, proporcionada por Lattice.

Output Standard	V_{CCIO} (Typ.)
Single-Ended Interfaces	
LVTTL	3.3
LVCMOS33	3.3
LVCMOS25	2.5
LVCMOS18	1.8
LVCMOS15	1.5
LVCMOS12	1.2
LVCMOS33, Open Drain	—
LVCMOS25, Open Drain	—
LVCMOS18, Open Drain	—
LVCMOS15, Open Drain	—
LVCMOS12, Open Drain	—
PCI33	3.3
SSTL25 (Class I)	2.5
SSTL18 (Class I)	1.8
HSTL18(Class I)	1.8
Differential Interfaces	
LVDS[1,2]	2.5, 3.3
BLVDS, MLVDS, RSDS [2]	2.5
LVPECL[2]	3.3
MIPI[2]	2.5
Differential SSTL18	1.8
Differential SSTL25	2.5
Differential HSTL18	1.8

1. MachXO2-640U, MachXO2-1200/U and larger devices have dedicated LVDS buffers.
2. These interfaces can be emulated with external resistors in all devices.

Figura 32: Niveles lógicos de las salidas en una MACHX02 1200 ZE

La estructura lógica de nuestra MACHX02 – 1200 ZE, queda reflejada en la figura 33:



Note: MachXO2-256, and MachXO2-640/U are similar to MachXO2-1200. MachXO2-256 has a lower LUT count and no PLL or EBR blocks. MachXO2-640 has no PLL, a lower LUT count and two EBR blocks. MachXO2-640U has a lower LUT count, one PLL and seven EBR blocks.

Figura 33: Esquema de una MACHX02 1200 ZE

Un PFU (figura 34), es la unidad mínima en la que se implementa la lógica. Según el fabricante, un PFU está formado por 4 unidades llamadas SLICE's.

El fabricante Lattice, implementa la lógica mediante agrupaciones de LUT4, llamadas SLICE. A cada uno de estos SLICE's les llegaran las señales del rutado general, y proporcionaran sus resultados, también al rutado general.

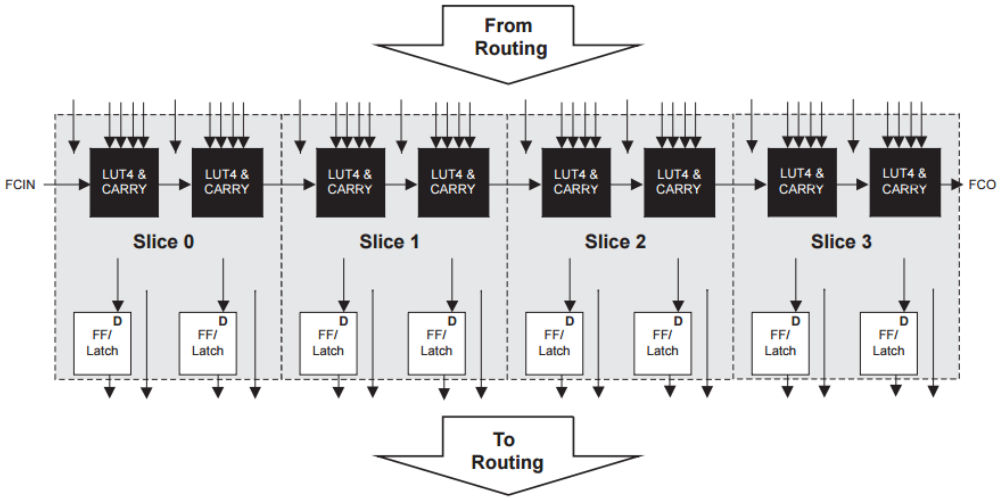


Figura 34: Diagrama lógico de un PFU

Además, cada uno de estos SLICE's, permite realizar lógica tanto registrada, a través de registros tipo D, o bien combinacional.

Estos SLICE's, pueden trabajar en distintos modos:

Logic >> En este modo de funcionamiento, cada LUT de cada SLICE, se configura como una tabla de búsqueda. Así, una función lógica puede ser generada por la programación de una tabla de en la que para determinadas posiciones se obtenga un '0' o un '1' lógicos. Las posiciones de la memoria serán seleccionadas por el correspondiente juego de variables de direccionamiento (fundamento de lógica con memoria).

Ripple >> Modo aritmético. El modo aritmético, está optimizado para poder realizar las operaciones más comunes, como sumas, restas, contajes, comparaciones, de manera rápida y eficiente.

Estas optimizaciones, se traducen en hechos, como una entrada de acarreo (FCIN) y una salida de acarreo (FCO).

RAM >> En el caso de no usar la memoria RAM distribuida o PFU para implementar la lógica de nuestra FPGA, esta puede ser añadida a la memoria RAM dedicada o EBR. Dentro de un PFU, solo pueden ser usados como memoria los SLICE 0, 1, y 2, pero nunca a el SLICE 3.

En modo RAM, se emplea el SLICE0, como memoria de 16 x 4, el SLICE 1 como memoria de 16 x 1 y el SLICE2, para proporcionar las señales de direccionamiento y control.

ROM >> Se pueden emplear todos los SLICE's de un PFU como memoria no volátil.

Tanto en el caso de los modos RAM, y ROM, se pueden configurar las características de los bloques de memoria con la aplicación IP-Express, que más adelante usaremos (figura 35).

Slice	PFU Block	
	Resources	Modes
Slice 0	2 LUT4s and 2 Registers	Logic, Ripple, RAM, ROM
Slice 1	2 LUT4s and 2 Registers	Logic, Ripple, RAM, ROM
Slice 2	2 LUT4s and 2 Registers	Logic, Ripple, RAM, ROM
Slice 3	2 LUT4s and 2 Registers	Logic, Ripple, ROM

Figura 35: Modos de funcionamiento de cada SLICE's de un PFU

Finalmente, cada SLICE, está compuesto por dos LUT4.

Cada LUT4, es la unidad mínima de lógica en la FPGA. Una LUT4, es una memoria de 4 líneas de direccionamiento, que por lo tanto, permite realizar cualquier función lógica de hasta 4 variables (o menos).

De esta forma, un SLICE, podrá realizar:

Hasta 2 funciones de 4 variables (Lógica utilizada al 100%).

Hasta 1 función de 5 variables.

A partir de aquí, introducir una nueva variable en nuestra función lógica, significara duplicar la capacidad de la memoria, por lo que:

1 función de 6 variables, requerirá el uso de 2 SLICE's.

1 función de 7 variables., requerirá el uso de 4 SLICE's o de un PFU completo.

La disposición del conexionado de cada SLICE, se puede ver en la figura 36.

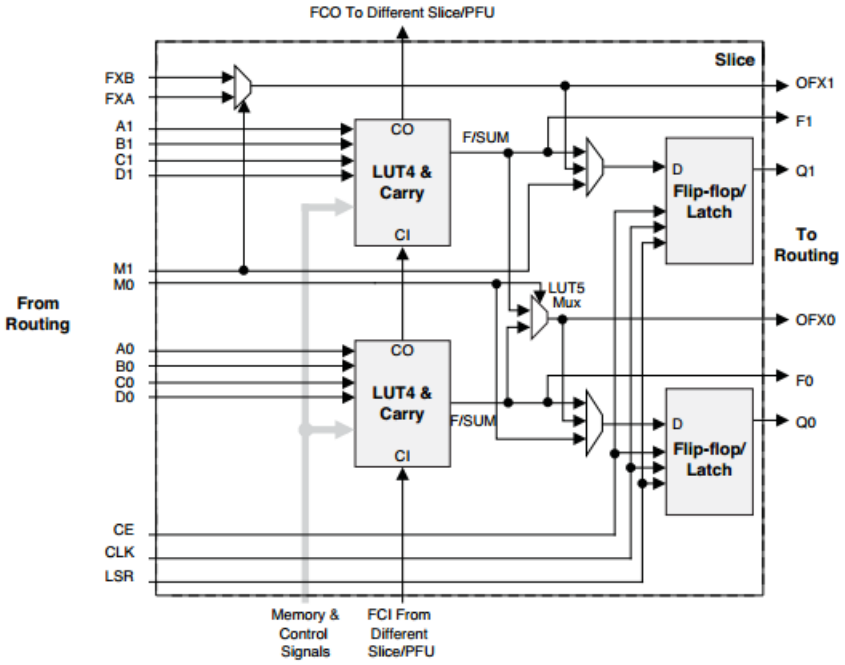


Figura 36: Diagrama lógico de un SLICE's

Por otra parte, la familia MACHX02 incorpora una serie de bloques denominados EBR. La EBR, es una memoria RAM dedicada. Esta memoria, no puede ser usada para la implementación de lógica, pero si para el almacenamiento de datos de usuario.

Esta memoria, puede ser usada como RAM, o como ROM.

La EBR, está formada por primitivas de memoria. Una primitiva de memoria es un bloque de memoria de 8kbits. Uniendo de manera adecuada dichas primitivas de memoria mediante la aplicación IP-Express, podemos lograr diferentes configuraciones:

Puerto Simple (figura 37) **>>** Memoria RAM, con un solo puerto, que le permite en cada ciclo de reloj, realizar sólo una operación de lectura, o bien una operación de escritura, pero nunca ambas a la vez.

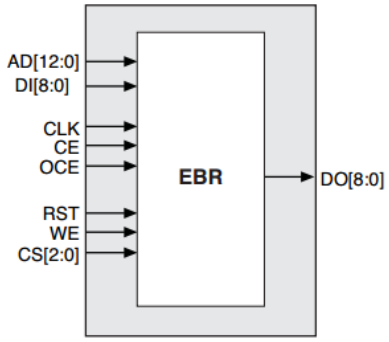


Figura 37: Memoria de puerto simple

Puerto doble verdadero (figura 38) **>>** Memoria RAM, con dos puertos independientes, que le permiten en un mismo ciclo de reloj realizar dos operaciones:

Lectura y escritura.

Escritura y lectura.

Lectura y lectura.

Escritura y escritura.

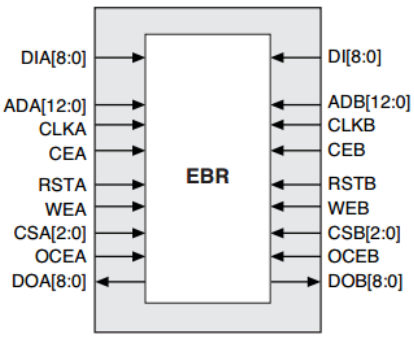


Figura 38: Memoria de verdadero doble puerto

Pseudo doble puerto (figura 39) **>>** Memoria RAM de dos puertos, que permite realizar en un mismo ciclo de reloj dos operaciones, siempre y cuando no coincidan simultáneamente dos lecturas o dos escrituras.

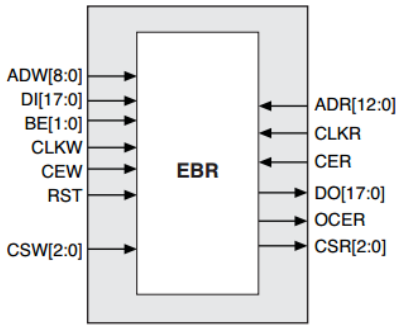


Figura 39: Memoria de pseudo doble puerto

FIFO (figura 40) **>>** Memoria de desplazamiento del tipo First In First Out, es decir, primer dato en entrar, primer dato en salir.

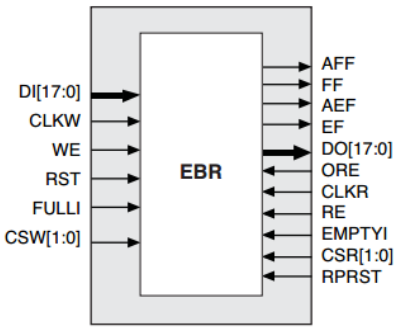


Figura 40: Memoria FIFO

En la figura 41, se reflejan todos los modos de funcionamiento de la EBR, como el número de palabras x ancho de palabra realizable en cada modo.

Memory Mode	Configurations
Single Port	8,192 x 1 4,096 x 2 2,048 x 4 1,024 x 9
True Dual Port	8,192 x 1 4,096 x 2 2,048 x 4 1,024 x 9
Pseudo Dual Port	8,192 x 1 4,096 x 2 2,048 x 4 1,024 x 9 512 x 18
FIFO	8,192 x 1 4,096 x 2 2,048 x 4 1,024 x 9 512 x 18

Figura 41: Capacidades máximas de memoria

Las señales que interactúan con dichos bloques se muestran en la figura 42.

Port Name	Description	Active State
CLK	Clock	Rising Clock Edge
CE	Clock Enable	Active High
OCE[1]	Output Clock Enable	Active High
RST	Reset	Active High
BE[1]	Byte Enable	Active High
WE	Write Enable	Active High
AD	Address Bus	—
DI	Data In	—
DO	Data Out	—
CS	Chip Select	Active High
AFF	FIFO RAM Almost Full Flag	—
FF	FIFO RAM Full Flag	—
AEF	FIFO RAM Almost Empty Flag	—
EF	FIFO RAM Empty Flag	—
RPRST	FIFO RAM Read Pointer Reset	—

1. Optional signals.
2. For dual port EBR primitives a trailing 'A' or 'B' in the signal name specifies the EBR port A or port B respectively.
3. For FIFO RAM mode primitive, a trailing 'R' or 'W' in the signal name specifies the FIFO read port or write port respectively.
4. For FIFO RAM mode primitive FULLI has the same function as CSW(2) and EMPTYI has the same function as CSR(2).
5. In FIFO mode, CLKW is the write port clock, CSW is the write port chip select, CLKR is the read port clock, CSR is the read port chip select, ORE is the output read enable.

Figura 42: Señales de las memorias

Para finalizar, la EBR, puede trabajar en tres modos diferentes.

Modo NORMAL: Los datos sólo aparecen en la salida en el ciclo de lectura, pero nunca en el ciclo de escritura.

Modo Write Through: Los datos aparecen en el ciclo de escritura automáticamente en la salida del puerto donde han sido escritos.

Modo Read before Write: Lectura antes de escritura. En este modo de funcionamiento, los datos almacenados en una dirección, aparecen automáticamente en la salida del puerto donde estamos escribiendo.

Para finalizar con los bloques de almacenamiento, la familia MACHX02, incorpora un bloque denominado UFM o User Flash Memory. Esta memoria no volátil, puede ser usada para una gran variedad de propósitos, como:

- Almacenamiento de una porción, o totalidad del fichero de configuración de la FPGA.
- Almacenamiento de datos.
- Memoria flash de usuario.

Este bloque es accesible mediante el Wishbone Bus.

La familia MACHX02 incorpora además, bloques EFB, o Embedded Function Block. Estos bloques, incorporan todas las funciones “prefabricadas” o precargadas de fábrica:

- PLL (Sintetizador de frecuencias).
- I^2C (Sistema de comunicación entre circuitos, mediante 2 hilos).
- SPI (Sistema de comunicación entre circuitos, mediante 3 hilos).
- Timer.
- Counter.
- Oscilador interno de 2.08 MHz

Todos estos periféricos, son accesibles, mediante el Wishbone Bus.

2.6. Material empleado

Para el desarrollo del presente trabajo de fin de grado, emplearemos los siguientes elementos:

2.6.1. Dispositivo FPGA

Vamos a emplear, la placa Lattice MachX02 1200ZE Breakout Board REV B, serie P/N LCMX02 – 1200ZE – B – EVN, (figuras 43 y 44), a la que se le han añadido empleando cable plano de 40 pines, una tarjeta con dos displays de 7 segmentos, y un conector de 6 pines para el sensor de ultrasonidos, y otra tarjeta con una botonera de 8 interruptores NA y un interruptor DIP de 8 líneas independientes.

Figura 43: Placa de prototipo. Vision anterior

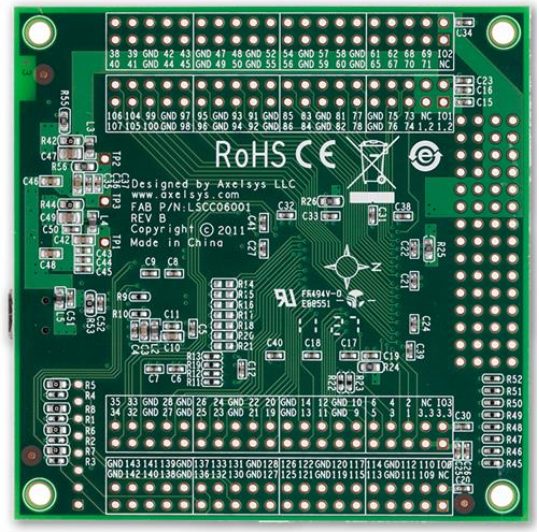


Figura 44: Placa de prototipo. Vista posterior

2.6.2. Sensor LV - MaxSonar – EZ3

Para la realización de este sistema, vamos a utilizar el sensor de ultrasonidos LV-MaxSonar®-EZ3 (figura 45), fabricado por MaxBotix®.

Figura 45: Sensor de ultrasonidos. Vista anterior

Se trata de un módulo que integra en un empaquetado de reducidas dimensiones (PCB de 19.9 X 22.1 mm) un emisor de ultrasonidos, un sensor de ultrasonidos, y finalmente un microcontrolador PIC16F676 que será utilizado como acondicionador de la señal.

Entre sus características más destacables, cabe señalar:

1. Rango de alimentación soportada: 2.5 a 5.5 V.

2. Consumo típico: 2mA.

3. Rango de detección de objetos: 6 a 254 pulgadas, o lo que es lo mismo 15.24 a 645 cm.

4. Frecuencia máxima de funcionamiento: 20 Hz. Esta frecuencia, nos limita la velocidad a la que podemos tomar medidas, ya que sólo se permite tomar una medida cada 50 ms.

5. Modos de funcionamiento: La información del sensor se puede presentar de tres formas distintas gracias al microcontrolador que la acondiciona:

    - Usando la salida analógica (AN): El valor de la distancia al obstáculo, se muestra a través de un valor de tensión variable. Esta tensión se calcula como $\frac{Vcc}{512}$ V/pulgada.

    - Usando la salida digital serie (TX) El valor de la distancia al obstáculo, se envía a través del interface RS232.
      Para ello, en primer lugar, se manda la cadena de bits correspondientes al código ASCII de la letra “R”. A continuación manda tres dígitos comprendidos entre 0 y 255, correspondientes a la distancia hasta el obstáculo, y finalmente, manda el valor 13 del código ASCII, correspondiente al retorno del carro.

    - Usando la salida moduladora de anchura de pulso (PW): El valor de la distancia al obstáculo, se muestra a través de un valor digital, de periodo constante pero tiempo a 1 lógico, variable. Esta técnica es la técnica conocida como modulación de anchura de pulso, o PWM.
      De esta manera, el tiempo que permanece a 1 lógico, se calcula mediante el factor 147 µs por pulgada.

6. Lóbulo de emisión de ultrasonidos variable: En función de la distancia al objeto, y de la tensión de alimentación, la forma del lóbulo de emisión de ultrasonidos varia. A continuación, exponemos en la figura 46, dicho efecto.

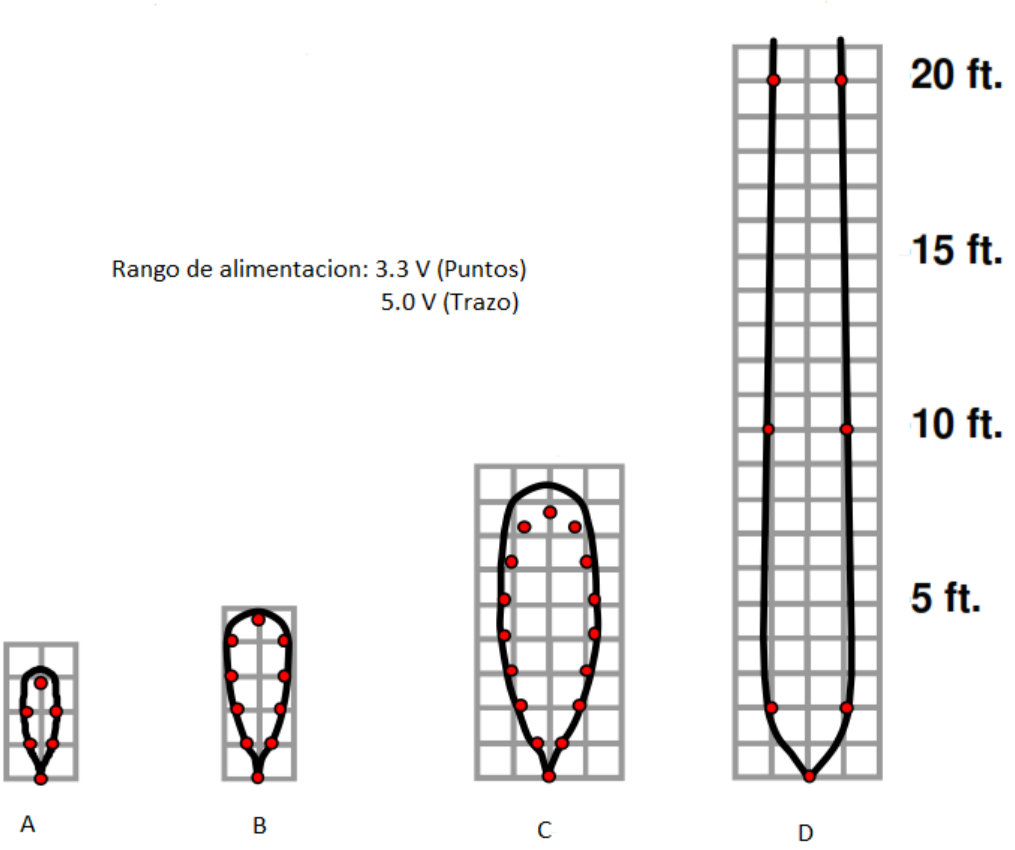


Figura 46: Forma del lóbulo de emisión

Como se puede apreciar las configuraciones A y B, son más adecuadas para la detección de pequeños objetos, mientras que las configuraciones C y D, al ser más direccionales, con más adecuadas para aplicaciones en las que los obstáculos sean de mayores dimensiones.

La información completa del sensor se encuentra en el Anexo Datasheet del sensor de ultrasonidos.

3. Desarrollo del TFG

Disponemos de un sensor que puede trabajar en distintos modos de funcionamiento. El modo que a nosotros nos interesa es el de modulación de anchura de pulso (PWM).

En este modo de funcionamiento, el sensor genera una onda en cuadratura, en la que sobre un periodo constante de 49 ms (Ver anexo **LV-MaxSonar®-EZ3™ Timing Description** del Datasheet del sensor), se modifica el tiempo que la señal permanece a '1' lógico, de acuerdo a la relación 147 $^{\mu s}/_{inch}$.

Así pues, en el caso de que el obstáculo se encuentre pegado al sensor se generará una onda de periodo 49 ms y tiempo a '1' lógico 0 µs, mientras que en el caso de que el obstáculo se encuentre en el fondo de escala (254 pulgadas), se generará una onda de periodo 49 ms y tiempo a '1' lógico 37.338 ms.

Dicha salida PW la utilizaremos para habilitar un contador. Dicho contador, tendrá como misión contar el número de pulsos que le llegan desde un oscilador, mientras la salida PW lo habilite.

Ajustando correctamente la frecuencia de oscilación anterior, podremos conseguir que cada pulso que cuente el contador, represente 1 cm. Así, sabiendo el número de pulsos contados, sabremos el número de cm que distan entre el sensor y el obstáculo.

Si tomamos dos medidas de distancia consecutivas, realizamos la resta de ambas en valor absoluto, y dividimos entre el tiempo transcurrido entre las dos medidas, podremos saber la velocidad.

Además, como internamente trabajaremos en formato BCD, emplearemos decodificadores BCD a 7 segmentos, y decodificadores a medida, para que el usuario pueda visualizar la distancia mediante enteros.

3.1. Implementación

3.1.1. Generador de eventos

Vamos a diseñar un generador de eventos, que permita establecer un esquema cronológico del sistema.

El generador de eventos, es la pieza clave de nuestro diseño, ya que permite establecer el orden y duración de las tareas que se desarrollan, dentro del ciclo de operación.

En mi caso, voy a diseñar un generador de eventos con las siguientes características:

1. Resolución: La resolución, es la unidad mínima de tiempo. En cualquier generador de eventos, ésta viene determinada por el reloj básico con el que trabajemos.

   Como reloj básico, vamos a usar el oscilador interno de la FPGA, cuya frecuencia de oscilación es de 2.08 MHz. De esta forma, el periodo de cada pulso de reloj será:

$$\frac{1}{2.08\ MHz} = \frac{1}{2080000} = 0.487\ \mu s$$

2. Ciclo: La duración del ciclo, es el tiempo que transcurre, hasta que el esquema cronológico de operaciones se repite de nuevo.
   La duración del ciclo, puede ser elegida por el usuario, dentro de ciertos límites, claro está. En concreto, el ciclo mínimo, lo proporcionará el dispositivo más lento, en mi caso el sensor de ultrasonidos, que como ya dijimos, sólo puede trabajar a 20 Hz cómo máximo.

3. Señales generadas: Cada 50 ms, se han de repetir el siguiente conjunto de operaciones:

   1. **Aplicar un Reset al decodificador tiempo - distancia (RST)**

      Instante de inicio del pulso: 0 µs. (Pulso 0 de reloj básico).

      Duración del pulso: 3.84 µs. (8 pulsos del reloj básico).

      Instante de fin del pulso: 3.84 µs (Pulso 7 de reloj básico).

      Función: Eliminar el valor de la distancia que se hubiera decodificado a partir de la señal PWM en el ciclo de operación anterior.

2. **Iniciar la conversión distancia – tiempo (RX)**

   Instante de inicio del pulso: 4.32 µs. (Pulso 8 de reloj básico).

   Tiempo estimado 23.07 µs. (48 pulsos de reloj básico).

   Instante de fin del pulso: 27.40 µs (Pulso 55 de reloj básico).

   Función: Esta señal, permite iniciar la lectura de distancias. Según el Datasheet del sensor, para iniciar la conversión Distancia – Tiempo, es necesario, aplicar un '1' fuerte al pin RX del sensor durante al menos 20 µs.

3. **Lapso para realizar la decodificación de distancias (LDD):**

   Instante de inicio del pulso: 27.88 µs. (Pulso 56 de reloj básico).

   Tiempo estimado 37341.34 µs como máximo. (77670 pulsos de reloj básico).

   Instante de fin del pulso: 37369.23 (Pulso 77725 de reloj básico).

   Función: Proporcionamos el tiempo necesario para realizar la decodificación tiempo de la onda PWM – distancia expresada en BCD.

   Lógicamente, el tiempo que proporcionaremos, será el del caso más desfavorable, es decir 37.338 ms como mínimo, correspondientes al caso de la máxima distancia medida, 254 pulgadas.

4. **Refresco del registro de datos 2 (RRD2):**

   Instante de inicio del pulso: 37369.71 µs. (Pulso 77726 de reloj básico).

   Duración del pulso: 3.84 µs. (8 pulsos del reloj básico).

   Instante de fin del pulso: 37373.55 µs (Pulso 77733 de reloj básico).

   Función: Actualizar el valor del registro de datos que actúa a modo de esclavo. Nótese, que antes de haber actualizado el registro de datos que actúa a modo de maestro, debemos de actualizar el esclavo, para evitar así que el maestro modifique el valor previo, antes de que el esclavo tenga garantías de haberlo almacenado.

5. **Refresco del registro de datos 1 (RRD1):**

   Instante de inicio del pulso: 37374.03 µs. (Pulso 77734 de reloj básico).

   Duración del pulso: 3.84 µs. (8 pulsos del reloj básico).

   Instante de fin del pulso: 37377.88 µs (Pulso 77741 de reloj básico).

   Función: Actualizar el valor del registro de datos que actúa a modo de maestro.

6. **Lapso para el cálculo de la velocidad (LCV):**

   Instante de inicio del pulso: 37378.36 µs. (Pulso 77742 de reloj básico).

   Duración del pulso: 7.69 µs. (16 pulsos del reloj básico).

   Instante de fin del pulso: 37386.05 µs (Pulso 77757 de reloj básico).

   Función: Proporcionamos el tiempo necesario, para que puedan realizarse los cálculos de la velocidad, a partir de dos distancias, tomadas en el registro maestro y el esclavo, y a partir del lapso entre ambas medidas.

A continuación, en la tabla 1, elaborada usando EXCEL, podemos observar la información de los tiempos anteriores. Para cada señal, podemos ver el instante de inicio, y de finalización, el pulso de inicio y de finalización, el número de pulsos, o de unidades de tiempo que ocupan, y finalmente, la suma global de tiempos y pulsos.

Frecuecnia de reloj básico	2,08	MHz						
Periodo de reloj básico	0,48	us						
Señales	**INSTANTE INICIO**		**TIEMPO REAL**		**TIEMPO MINIMO**		**INSTANTE FIN**	
	Pulso	**us**	**Pulso**	**us**	**Pulso**	**us**	**Pulso**	**us**
Reset al decodificador Tiempo-Distancia	0,00	0,00	8,00	3,85			7,00	3,85
Conversion Distancia-Tiempo	8,00	4,33	48,00	23,08		20,00	55,00	27,40
Lapso para Decodificacion de distancias	56,00	27,88	77670,00	37341,35		37338,00	77725,00	37369,23
Refresco del R.Desplazamiento Esclavo	77726,00	37369,71	8,00	3,85			77733,00	37373,56
Refresco del R.Desplazamiento Maestro	77734,00	37374,04	8,00	3,85			77741,00	37377,88
Lapso Calculo de velocidad	77742,00	37378,37	16,00	7,69			77757,00	37386,06
							Pulsos de reloj	Tiempo en ms
						TOTAL	77757,00	37,39
						DISPONIBLE	"-"	50,00
						SOBRA		12,61

Tabla 1: Organización cronológica del generador

Como podemos ver, en un ciclo de duración máxima de 50 ms, estamos consumiendo un tiempo de 37.38 ms, es decir:

$$\frac{37.38}{50} \cdot 100 = 75\%$$

Estamos consumiendo el 75% del ciclo en realizar operaciones. Aunque estamos consumiendo un porcentaje alto del ciclo en operar, el circuito no será crítico en tiempos, pues aún nos sobraría tiempo para hacer más operaciones.

Por esta razón, tiempos que he considerado más críticos, como el inicio de la conversión distancia – tiempo o el lapso para la decodificación de la distancia, los he sobredimensionado.

Vamos a dimensionar ahora, el contador que se encargará de generar la base de tiempos. Como ya dijimos, el reloj básico, es el oscilador interno de la FPGA, cuya frecuencia de oscilación era de 2.08 MHz. El número de estados de nuestro contador será entonces de:

$$2.08\ \text{Mhz} = 2080000\ \text{Hz}$$

$$\frac{2080000\text{Hz}}{104000} = 20\ \text{Hz}$$

Como podemos ver, si empleamos un contador de 104000 (Ciento cuatro mil) estados diferentes, conseguiremos un tiempo de desbordamiento del contador, y por tanto un tiempo de ciclo de 50 ms.

Vamos a emplear por tanto un contador de 17 bits con contaje de 0 a 104000 (El valor final de cuenta está limitado, no llegamos a los 2^{17}estados posibles).

La duración de cada estado del contador está determinada por el reloj básico que lo alimente, así, si alimentamos con un contador de 2.08 MHz, cada estado tendrá una duración de:

$$\frac{1}{2080000} = 0.487\ \mu s$$

Describiremos el contador anterior, mediante un bloque generado en el asistente IP-Express.

El primer paso, será crear un nuevo proyecto, llamado en este caso ContadorEventos. Seguiremos los siguientes pasos:

1. Abrimos la aplicación Diamond haciendo doble clic en el icono de acceso directo (Figura 47).

Figura 47: Icono Diamond 3.3

2. Se nos abrirá a continuación una ventana de bienvenida, como la que aparece en la figura 48.

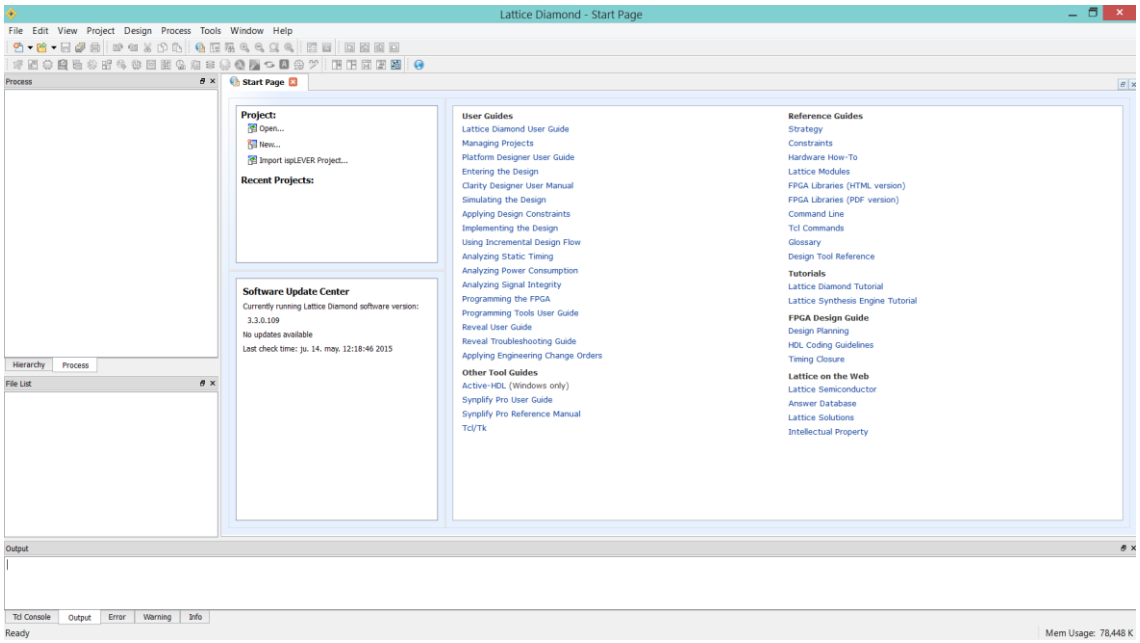


Figura 48: Ventana de bienvenida de Diamond

Para crear un nuevo proyecto, haremos clic en File – New – Project (figura 49).

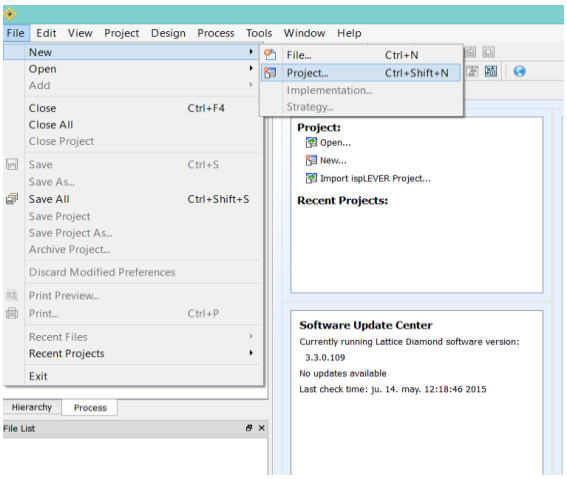


Figura 49: Creación de nuevo proyecto (1)

3. Se nos abrirá a continuación un asistente, figura 51, en el que se nos solicitan las características del proyecto:

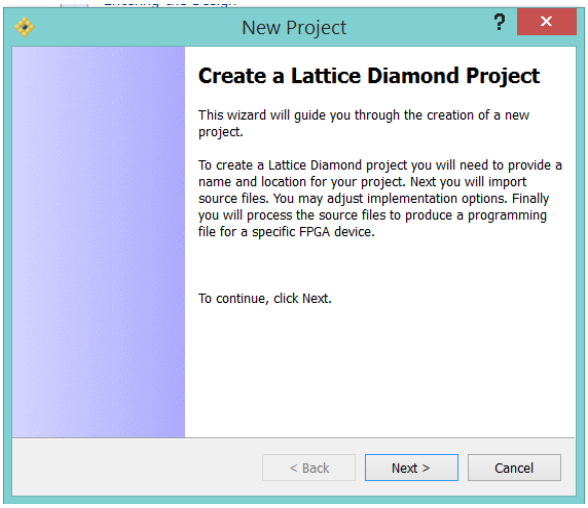


Figura 50: Creación de nuevo proyecto (2)

4. Si hacemos clic en "Next", se abrirá la ventana en la que seleccionaremos Nombre del proyecto, y ubicación (figura 51). Como nombre, emplearemos el que se desee, pero como ubicación mantendremos la misma ubicación para todos los proyectos que posteriormente creemos, para de esa forma, mantener el path, y poder usar los bloques generados en un proyecto, en otro distinto:

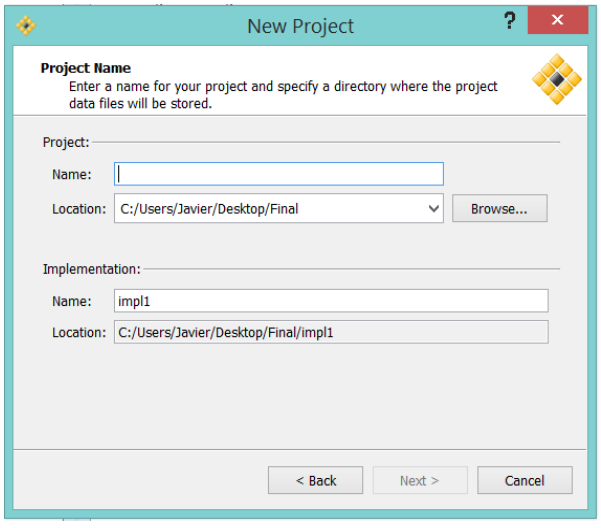


Figura 51: Creación de nuevo proyecto (3)

5. Una vez completados los campos anteriores, haciendo clic en “Next” se abrirá una ventana, donde se nos da la opción de incorporar al proyecto ficheros fuente (.vhd o .sch en nuestro caso), ya existentes, figura 52:

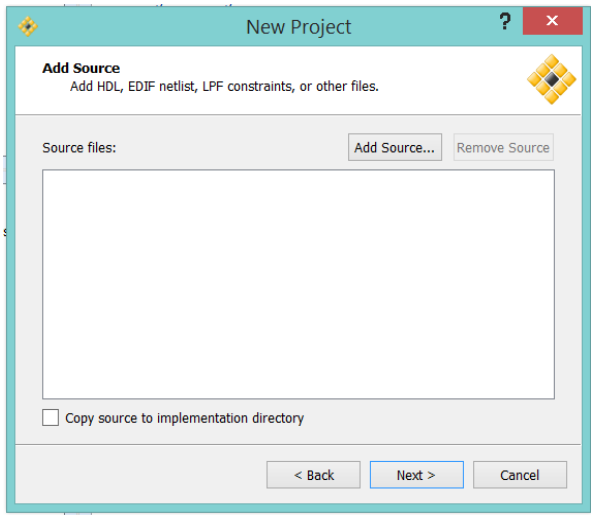


Figura 52: Creación de nuevo proyecto (4)

6. El último paso del asistente será elegir el dispositivo concreto al que va destinado nuestro diseño. Las opciones elegidas, son las que se muestran en la figura 53:

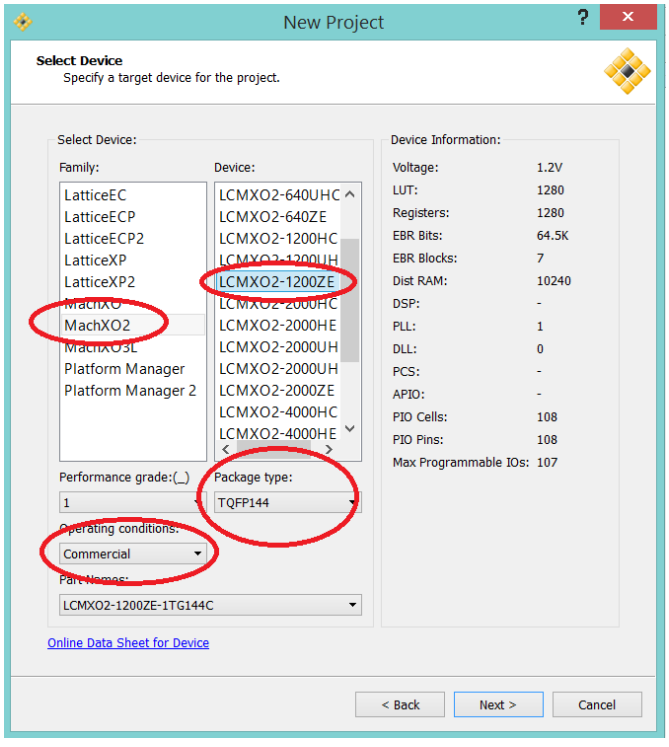


Figura 53: Creación de nuevo proyecto (5)

Una vez finalizado el asistente, se abrirá la ventana principal de trabajo (figura 54):

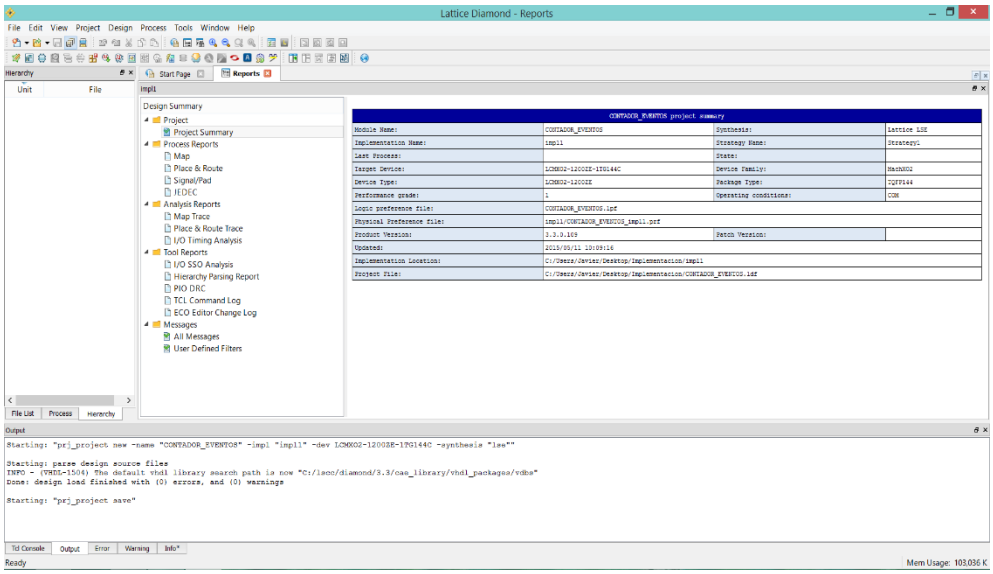


Figura 54: Ventana principal del proyecto

En este primer diseño, vamos a trabajar con la herramienta IP-Express, para la generación automática de hardmacros. Para ello, hacemos clic en el símbolo  de la barra de herramientas.

Se nos abrirá un asistente, en el que deberemos seleccionar el tipo de módulo (Counter), y las opciones de diseño que se muestran en la figura 55:

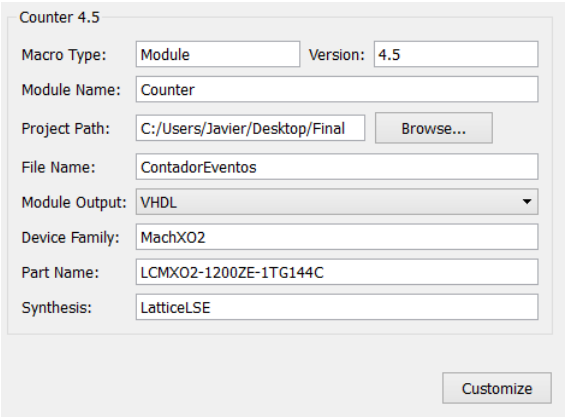


Figura 55: Opciones IPExpress (1)

Una vez completados todos los campos anteriores, haremos clic en CUSTOMIZE. Se cerrará la ventana anterior, y se abrirá automáticamente una nueva ventana, donde deberemos especificar:

1. Ancho del bus de datos.
2. Valor mínimo de cuenta.
3. Valor máximo de cuenta.

Recordemos que todos estos parámetros, ya fueron definidos y dimensionados cuando realizamos los cálculos del contador.

Las opciones de diseño elegidas, se muestran en la figura 56:

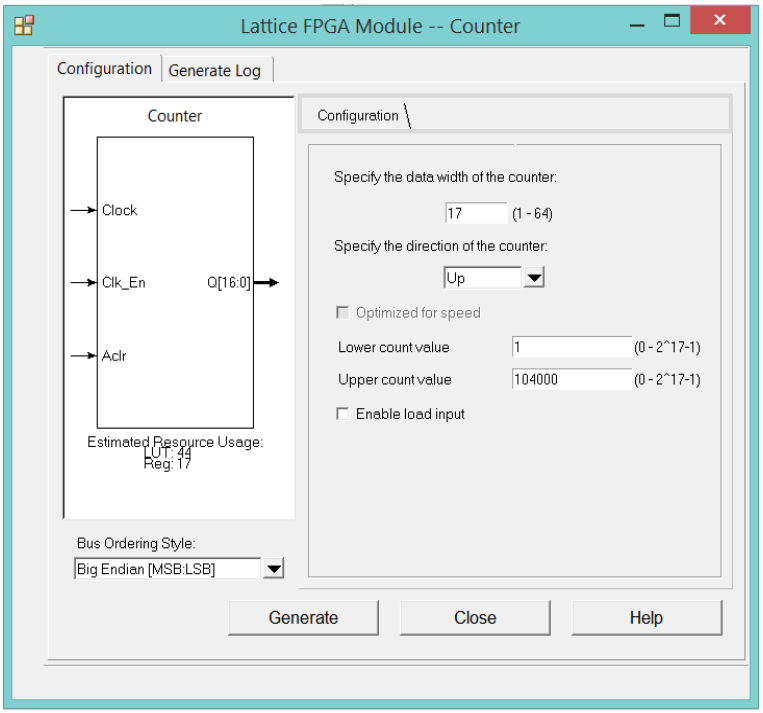


Figura 56: Opciones IPExpress (2)

El objetivo final que perseguimos en este primer proyecto, es crear una unidad, que podamos usar en el proyecto final, o general, que englobará a todos los que creemos.

Para ello, es necesario, que asignemos a cada nuevo elemento creado, un símbolo que posteriormente nos permita usarlo.

Cuando trabajamos con la herramienta IPExpress, si seleccionamos la opción “Import IPX to Diamond Project”, el propio asistente, creará dicho símbolo.

El símbolo de nuestro nuevo contador creado de forma automática se muestra en la figura 57:

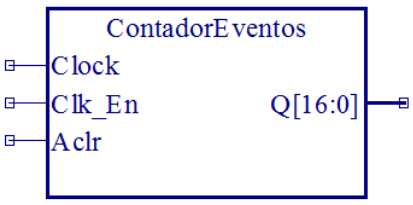


Figura 57: Símbolo Contador de Eventos

Una vez creado el contador, debemos crear el bloque que permita hallar las funciones lógicas de activación de cada señal.

Siguiendo los pasos anteriores, vamos a crear un nuevo proyecto, llamado LogicaEventos. Puesto que en esta ocasión, no vamos a trabajar con IPExpress, puesto que debemos de crear nuestro propio bloque, será necesario que hagamos clic en File – New File (figura 58):

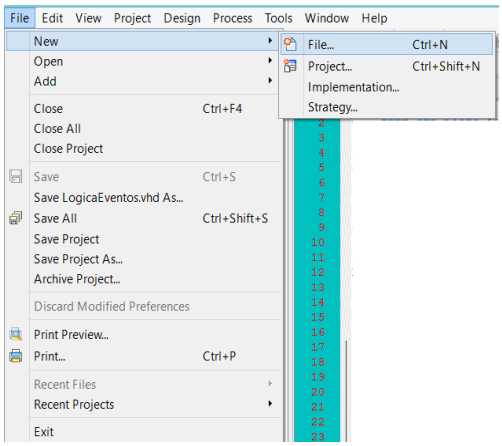


Figura 58: Creación de un nuevo fichero

Se nos abrirá como consecuencia, un menú, como el mostrado en la figura 59, donde se nos solicita el tipo de fichero, y su nombre. En nuestro caso, el fichero será de tipo VHDL, y se llamará LogicaEventos.

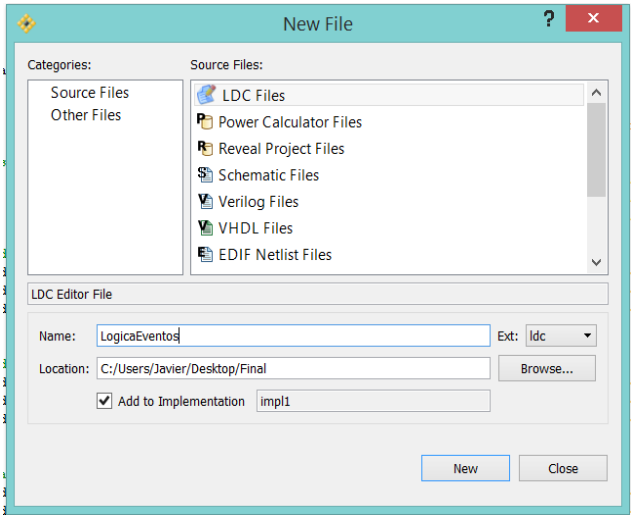


Figura 59 Creación de un nuevo fichero (2)

En nuestro nuevo fichero, escribiremos el siguiente código VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.alL;          --Contiene los tipos de datos STD_LOGIC

use IEEE.STD_LOGIC_ARITH.all;         --Obligatoria para hacer conversiones
use IEEE.STD_LOGIC_UNSIGNED.all;      --Idem
use IEEE.NUMERIC_STD.all;             --Idem


entity LogicaEventos is
        port(Q: in STD_LOGIC_VECTOR (16 downto 0);
             CUENTA: inout INTEGER;
             RST:out STD_LOGIC := '0';          --Inicializamos las
             RX: out STD_LOGIC := '0';          --Idem
             RRD2: out STD_LOGIC:= '0';         --Idem
             RRD1: out STD_LOGIC:= '0';         --Idem
             LCV: out STD_LOGIC:= '0'           --Idem
            );
end LogicaEventos;




architecture ArqLogicaEventos of LogicaEventos is
begin

        --Convertimos el vector Q a entero
        CUENTA<=conv_integer(Q);

        --Obtenemos las señales deseadas
        with CUENTA select
                RST <=  '1' when 0 to 7,
                                '0' when others;
        with CUENTA select
                RX <=   '1' when 8 to 55,
                                '0' when others;

        with CUENTA select
                RRD2 <= '1' when 77726 to 77733,
                                '0' when others;

        with CUENTA select
                RRD1 <= '1' when 77734 to 77741,
                                '0' when others;

        with CUENTA select
                LCV <=  '1' when 77742 to 77757,
                                '0' when others;


end ArqLogicaEventos;
```

Como podemos observar en el código anterior, hemos trabajado con CUENTA, una señal de tipo inout. Ésta señal de tipo inout, representa una señal interna, o variable interna.

Cuando salvemos el diseño, se generara la correspondiente jerarquía (figura 60):

Figura 60: Símbolo Lógica de Eventos

En esta ocasión, si deseamos crear un símbolo para la lógica anterior, deberemos hacerlo manualmente. Para ello, haremos clic derecho sobre la jerarquía anterior y seleccionaremos la opcion “Generate Schematic Symbol”.

El símbolo obtenido tras este proceso se muestra en la figura 61:

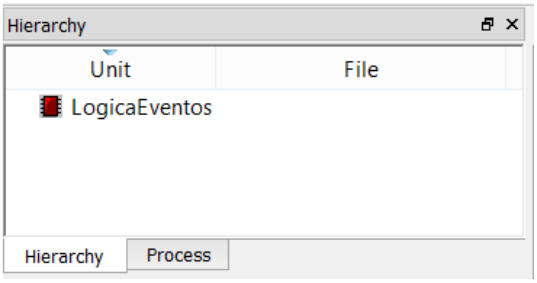


Figura 61: Jerarquía de Logica de Eventos

El generador de eventos, quedará finalizado cuando conectemos los dos bloques anteriores en cascada, y los alimentemos con las señales adecuadas, como aparece en la figura 62:

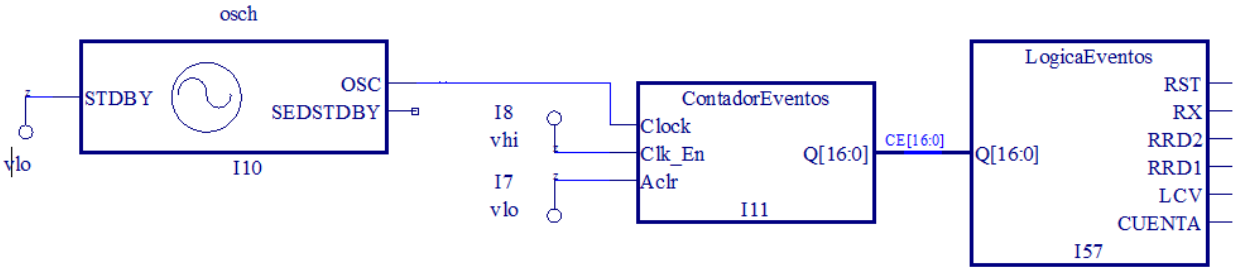


Figura 62: Esquema general del Generador de Eventos

Como vemos, es necesario alimentar adicionalmente al contador con:

- Clock: Entrada de eventos al contador. Esta es la entrada en la que debemos de aplicar la señal cuyos pulsos queremos contar, en nuestro caso, el reloj grueso, o reloj interno de la FPGA, representado por el bloque “osch”.
- Clk_En: Habilitación de la entrada de reloj. Para que el circuito pueda contar el número de pulsos procedentes de reloj, es necesario que se encuentre a ‘1’ lógico. Internamente la pondremos a TRUE con el símbolo “vhi”.
- Aclr: Reset asíncrono del contador. Permite reiniciar la cuenta a 0 en cualquier momento. Nosotros no la usaremos, ya que deseamos una cuenta cíclica, en la que cuando el contador llegue a su valor tope (Top Count) se reinicie.
  Dicha entrada la pondremos a FALSE con el símbolo “vlo”.

Como vemos, un detalle muy importante a tener en cuenta, que si se pasa por alto, nos traerá muchos problemas, es poner la entrada del oscilador STDBY a ‘0’ lógico ya que el oscilador interno, se habilita a nivel bajo.

Para finalizar, vamos a comprobar el correcto funcionamiento de nuestro generador de eventos. Para ello, vamos a simular su comportamiento en un periodo de reloj completo.

Abrimos el simulador haciendo clic sobre el botón  de la barra de menú. Se nos abrirá entonces la aplicación de simulación denominada Active-HDL.

De forma automática, nos aparecerá la ventana de bienvenida (figura 63):

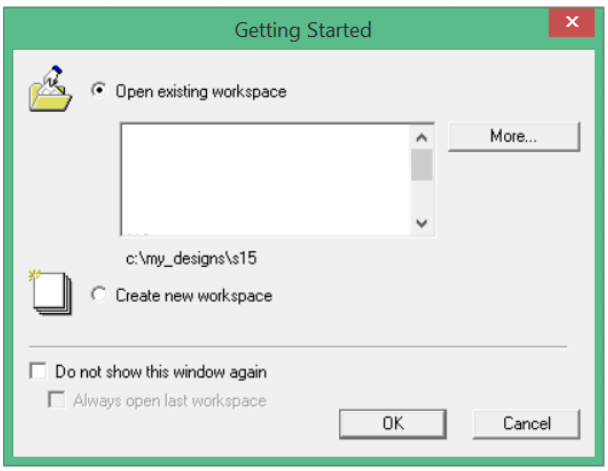


Figura 63: Creación de un nuevo proyecto de simulación (1)

Como no tenemos un espacio de trabajo previo, ya que aún no hemos realizado ninguna simulación, vamos a crear un nuevo espacio de trabajo haciendo clic en Create new workspace.

Se nos abrirá la ventana que aparece en la figura 64, donde elegiremos el nombre del espacio de trabajo, y su ubicación:

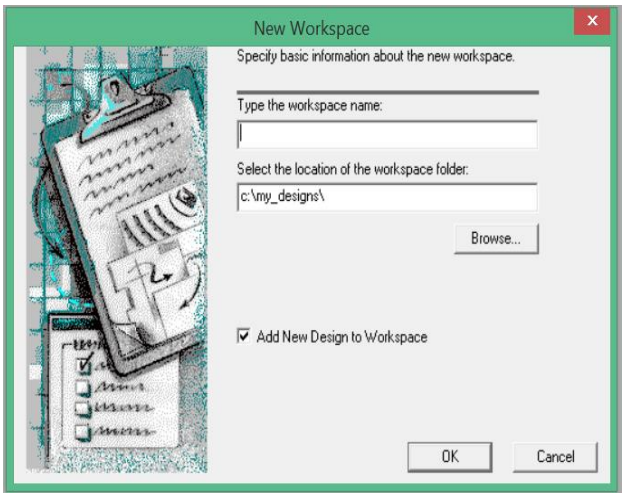


Figura 64: Creación de un nuevo proyecto de simulación (2)

Si damos OK, se nos abrirá la ventana que aparece en la figura 65, donde especificaremos, la fuente de nuestro diseño. El asistente, nos da la opción de elegir entre varias fuentes, de las cuales, a nosotros nos interesa Add existing Resource files, ya que previamente habíamos escrito y compilado en Diamond nuestro diseño:

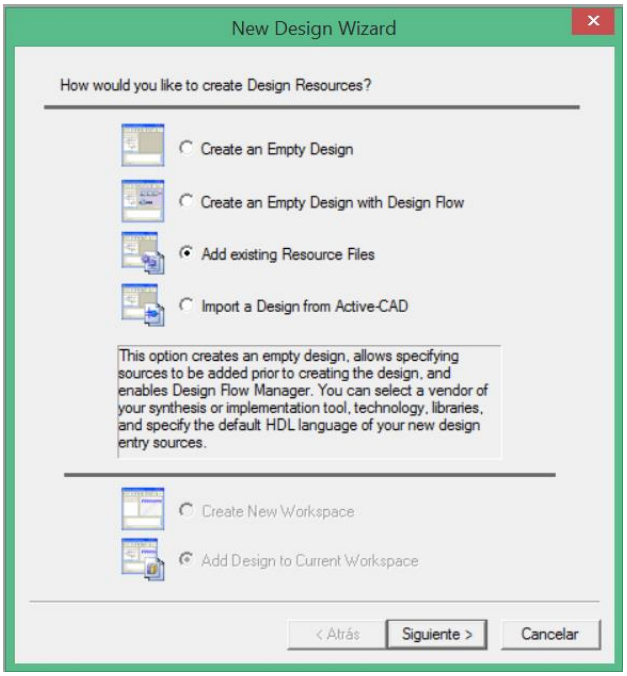


Figura 65: Creación de un nuevo proyecto de simulación (3)

Si hacemos clic en siguiente, se abrirá una nueva ventana, la ventana que aparece en la figura 66, donde se nos muestran los ficheros añadidos a nuestro espacio de trabajo. Lógicamente, al principio debe estar vacío, ya que no hemos añadido ningún fichero.

Para añadir algún fichero, hacemos clic en el botón Add Files… y seleccionamos el fichero LogicaEventos.vhd.

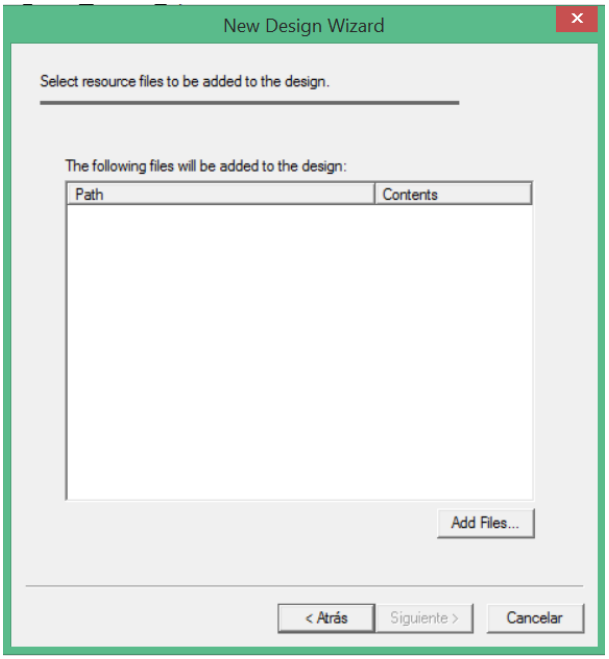


Figura 66: : Creación de un nuevo proyecto de simulación (4)

Haciendo clic en siguiente, se nos abrirá, la ventana que aparece en la figura 67, donde seleccionaremos las opciones mostradas:

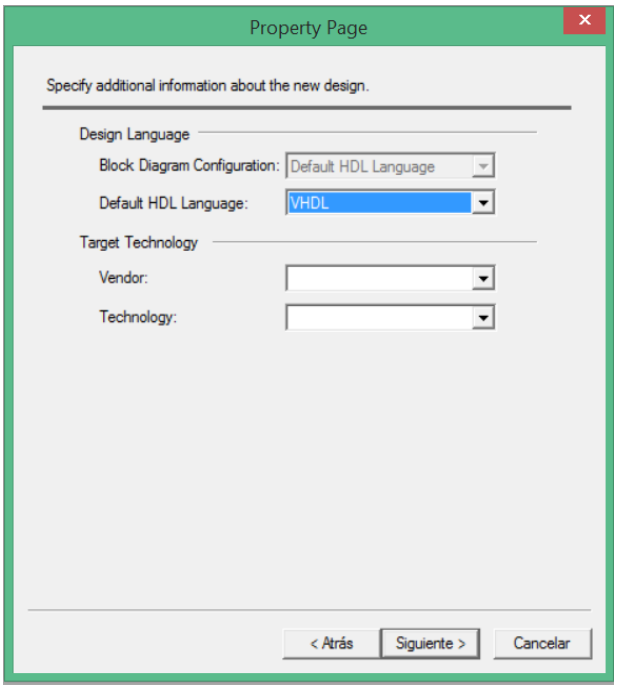


Figura 67 : Creación de un nuevo proyecto de simulación (5)

De nuevo haremos clic en siguiente, apareciendo la ventana que aparece en la figura 68.

En ella, deberemos especificar el nombre y la ubicación donde Active-HDL guardará el diseño añadido anteriormente. La ubicación podemos dejar la que viene por defecto, y el nombre, podemos repetir el que escribimos inicialmente.

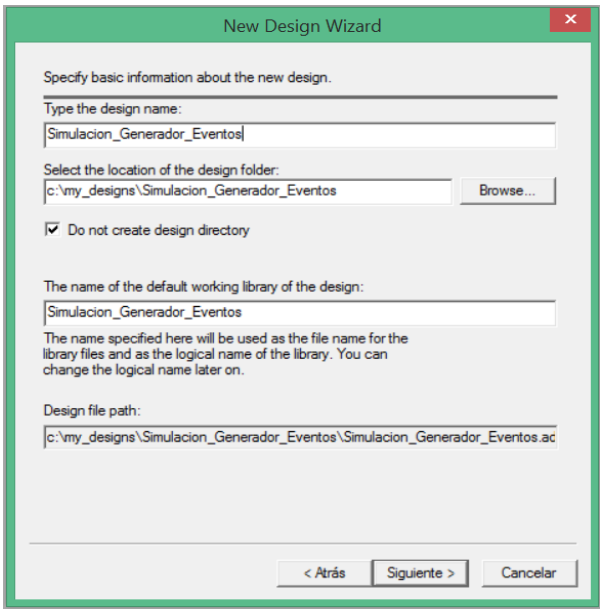


Figura 68 : Creación de un nuevo proyecto de simulación (6)

Una vez hayamos dado a Siguiente, aparecerá la ventana que aparece en la figura 69. En esta última ventana, sólo debemos tener precaución en haber seleccionado la casilla de compilación.

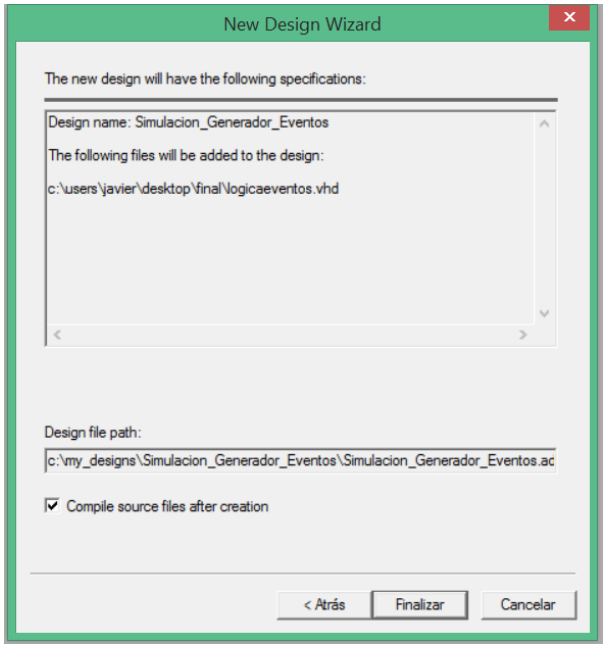


Figura 69: Creación de un nuevo proyecto de simulación (7)

Haciendo clic en Finalizar, habremos terminado.

Para comenzar a simular el bloque, deberemos hacer clic en iniciar simulación (figura 70).

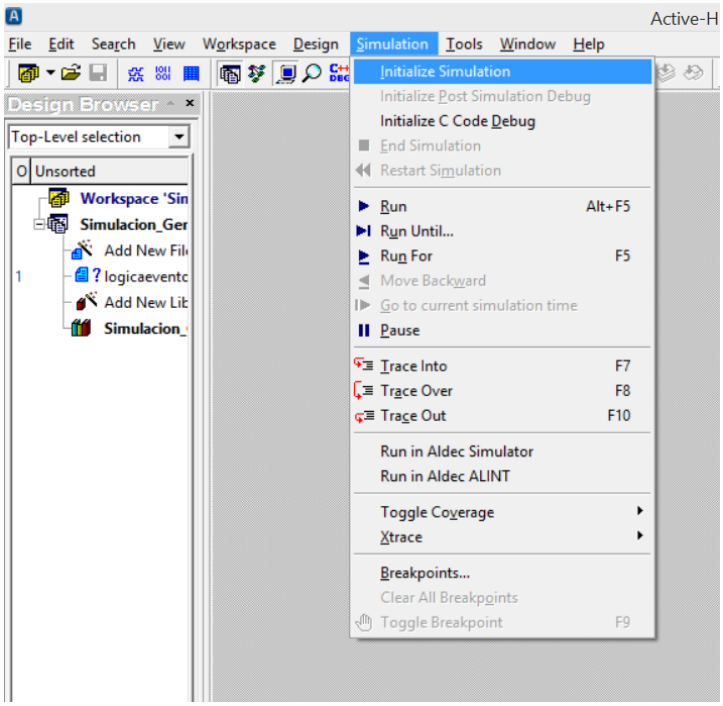


Figura 70: Inicialización de una simulación.

Posteriormente, haremos clic en el icono  para añadir una nueva grafica donde representar las formas de onda.

El resultado se puede ver en la figura 71.

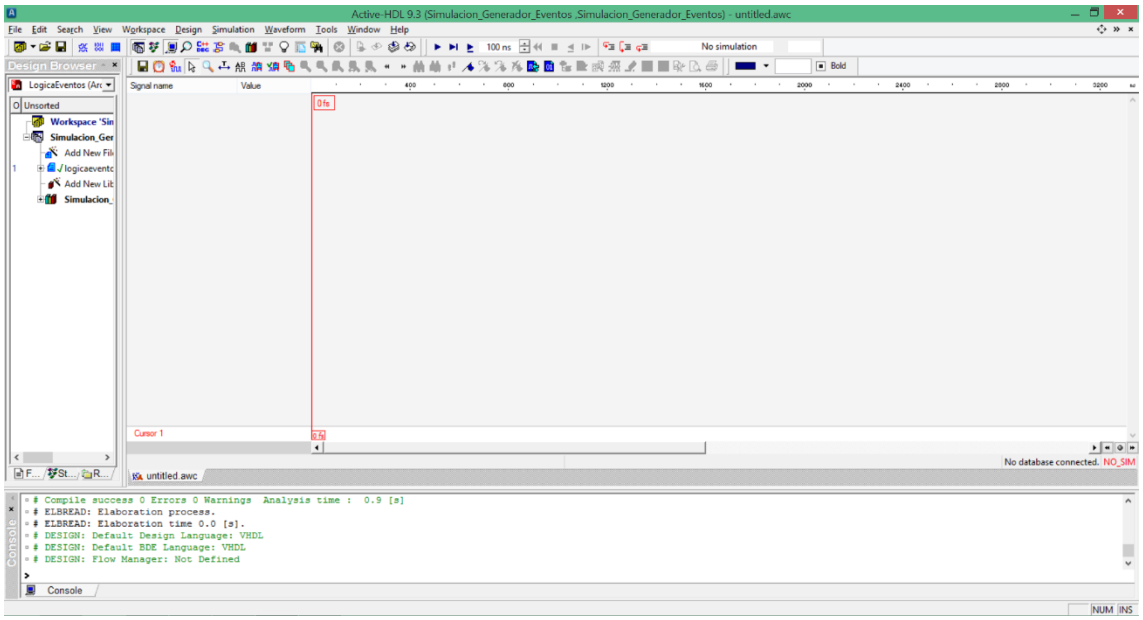


Figura 71: Ventana de simulación

Ahora, vamos a añadir las señales de nuestro bloque. Para ello, en el cuadro de la izquierda, (figura 72) seleccionamos la pestaña Structure. Ésta pestaña, nos permitirá ver las señales de entrada, salida, o internas de nuestro bloque.

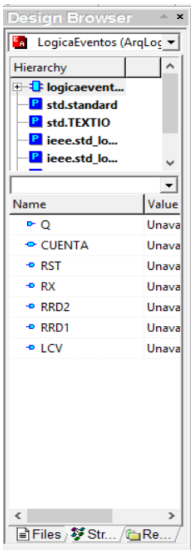


Figura 72: Lista de señales

Para añadir las señales, no tenemos más que seleccionarlas, ya hacer clic con el botón derecho y seleccionar Add to waveform

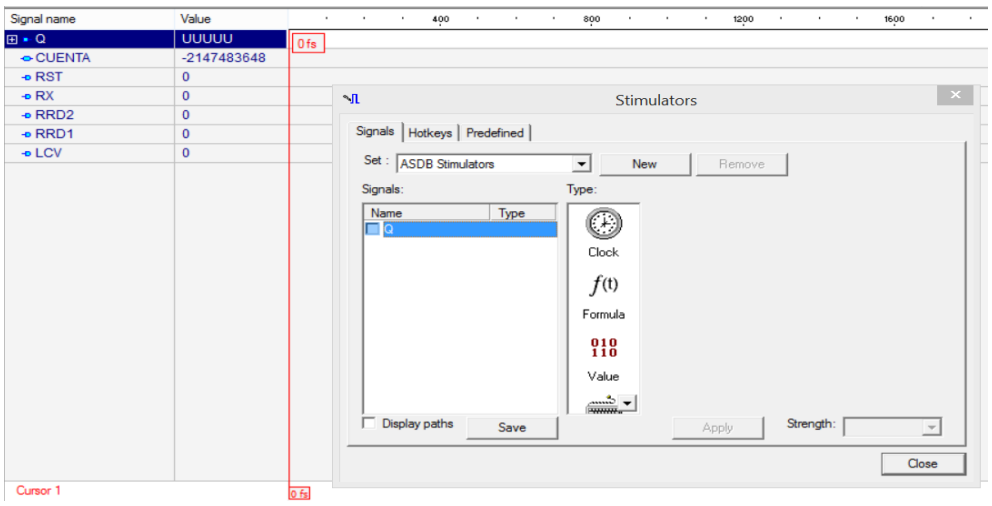


Figura 73: Ventana de estimulación de entradas

Una vez añadidas las señales, haremos clic derecho en Q, nuestra entrada, y seleccionaremos la opción Stimulators. Se nos abrirá un menú de estímulos (figura 73), con el que poder determinar qué entrada aplicamos a dicha señal.

Para nuestra señal Q, seleccionaremos un contador, creciente, con incrementos de unidad en unidad, y un tiempo de cada estado de 1 ns (figura 74).

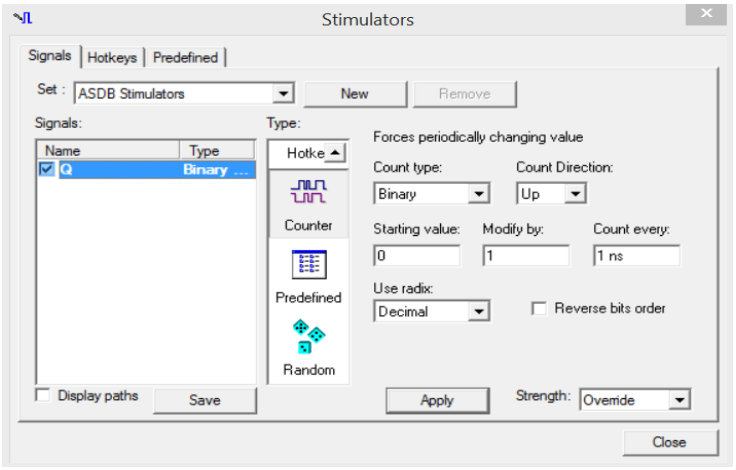


Figura 74: Estimulación con contador

El tiempo de simulación será de 80 µs (De esta forma simulamos 80000 estados)

A continuación se muestra en las figuras 75 y 76, la evolución de las distintas señales:

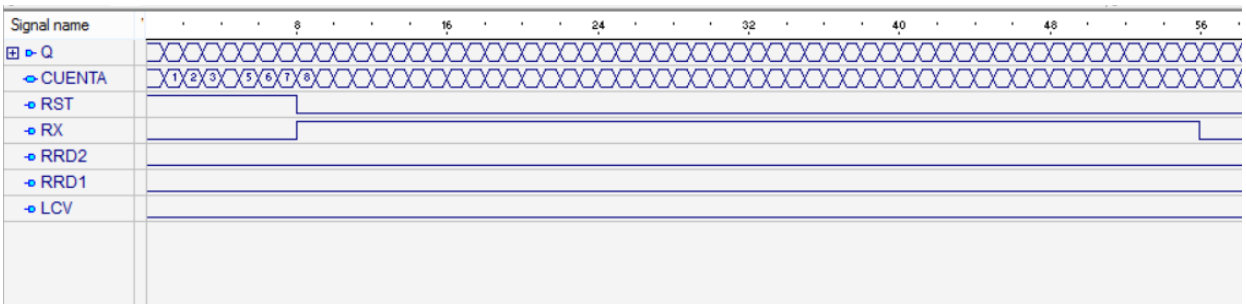


Figura 75: Comportamiento de señales (1)

Como se puede apreciar, la señal RST se activa entre las combinaciones 0 y 7, y RX, entre las combinaciones 8 y 56. De este modo, vemos que el comportamiento de las mismas, es el esperado

RRD2

RRD1

Y

LCV

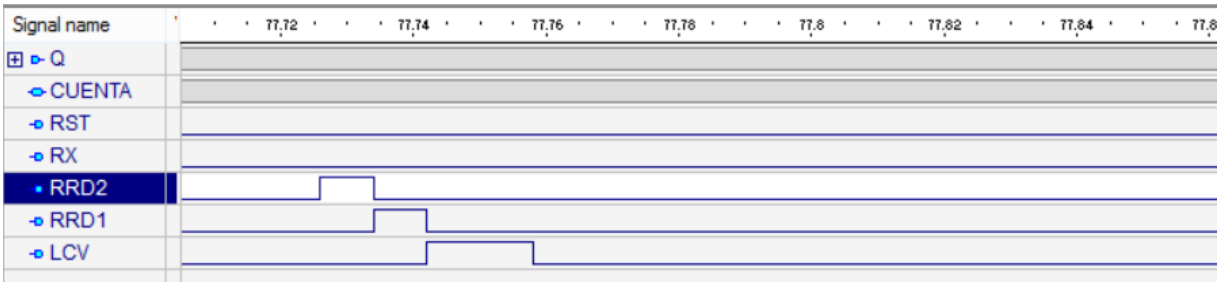


Figura 76: Comportamiento de señales (2)

Se observa las mismas conclusiones que en las señales anteriores.

3.1.2. Reloj fino

Vamos a emplear un reloj, en el que ajustaremos su frecuencia de oscilación, de modo que 1 pulso equivalga a 1 cm.

La misión de este reloj, será permitirnos medir la distancia proporcionada por el sensor, ya que recordemos que el sensor transformaba la distancia medida a tiempo mediante una onda modulada de tipo PWM.

Del DataSheet del emisor/receptor de ultrasonidos, tenemos que:

> La duración del pulso generado por el emisor/receptor es de 147µs/inch.
>
> Por lo que la duración del pulso en unidades del Sistema métrico decimal es de $147\mu s/2.54cm$.
>
> O lo que es lo mismo: $\frac{147}{2.54}\ {}^{\mu s}/_{cm} = 57.874015\ {}^{\mu s}/_{cm}$.
>
> De esta manera, la frecuencia de oscilación será: $\frac{1}{57.874015 \cdot 10^{-6}} = 17.278\ kHz$.

En este caso, puesto que el reloj fino desempeña una función vital, necesitaremos que sea preciso, exacto, y repetible. Queda por lo tanto descartado usar como fuente del reloj fino el oscilador de la FPGA, ya que es algo inexacto. Según el Datasheet de nuestra FPGA, la precisión del oscilador es de 5.5%

En su lugar usaremos el oscilador externo, que se ha incorporado a la placa de pruebas.

De la documentación adjunta con la placa de pruebas (Ver anexo Módulo de prácticas de sistemas electrónicos reconfigurables, sección Reloj externo) hemos obtenido que la frecuencia de oscilación del reloj externo es de 50 MHz. Por lo tanto, para ajustar la frecuencia a 17.278 kHz, deberemos usar el siguiente sistema de escalamiento:

$$\frac{50000\ kHz}{X} = 17.278\ kHz.$$

$$\frac{50000\ kHz}{17.278\ kHz} = X;\ \ X = 2893.853455.$$

De esta manera, necesitaremos un contador de 2894 (Dos mil ochocientos noventa y cuatro) estados distintos.

El cálculo del número de bits es inmediato sabiendo que necesitamos, por lo menos 2894 estados diferentes:

$2^{11} = 2048\ ;\ 2^{12} = 4096.$

Todo lo anterior nos muestra que necesitamos un contador de 12 bits con cuenta de 1 a 2894.

Dicho contador será implementado en el proyecto llamado EscaladorFino.

Procederemos del mismo modo que en el caso anterior, empleando la herramienta IP-Express, para personalizar un contador.

En esta ocasión, sin embargo, las opciones elegidas serán las mostradas en las figuras 77 y 78.

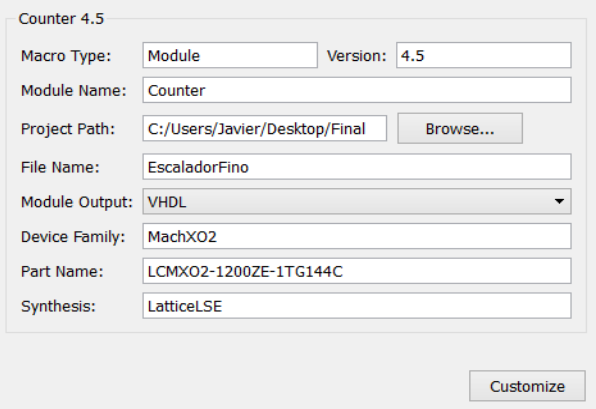


Figura 77: Opciones IPExpress (3)

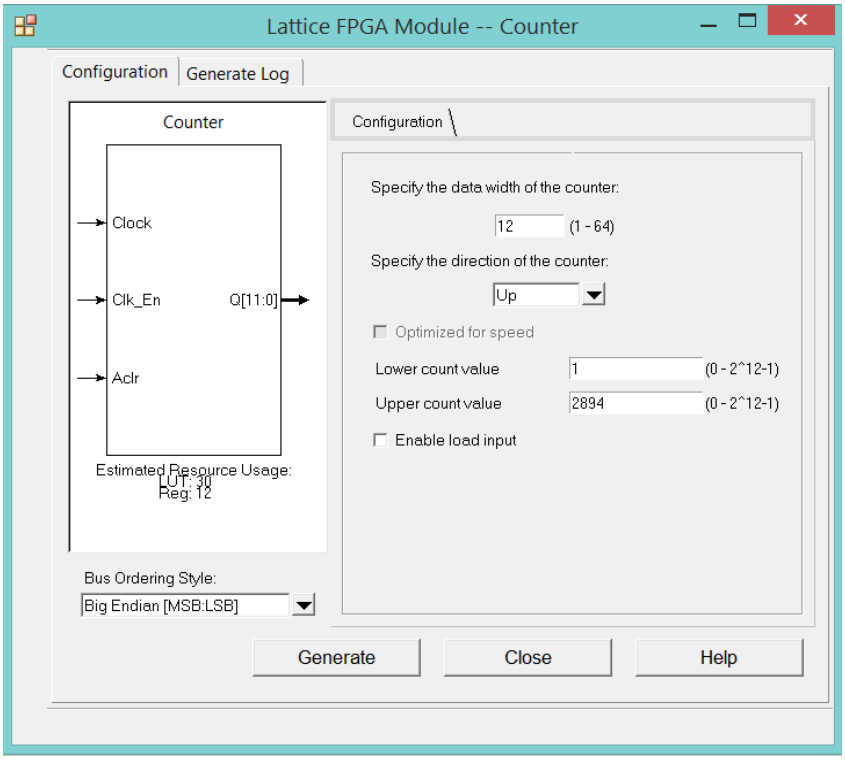


Figura 78: Opciones IPExpress (4)

Tras hacer clic en “Generate”, habremos terminado.

En esta ocasión, el símbolo obtenido se muestra en la figura 79:

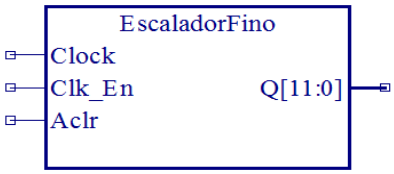


Figura 79: Símbolo Escalador Fino

Finalmente, procederemos a interconectar el bloque EscaladorFino con el resto de elementos del esquema (figura 80):

Como vemos, es preciso conectar:

- Clock: La entrada de pulsos del reloj, se conectara al reloj fino, que como ya especificamos oscilaba a una frecuencia de 50 kHz.
- Clk_En: Ya hemos explicado con el bloque ContadorEventos, la función de dicha entrada. Sólo volveré a resaltar, que debemos conectarla a ‘1’ lógico.
- Aclr: Idem. La conectaremos en este caso a ‘0’ lógico.

Finalmente, la salida que tomaremos será el bit de mayor peso de la cuenta, el bit Q[11], que según nuestros cálculos, oscilará a una frecuencia de 1 Hz aproximadamente.

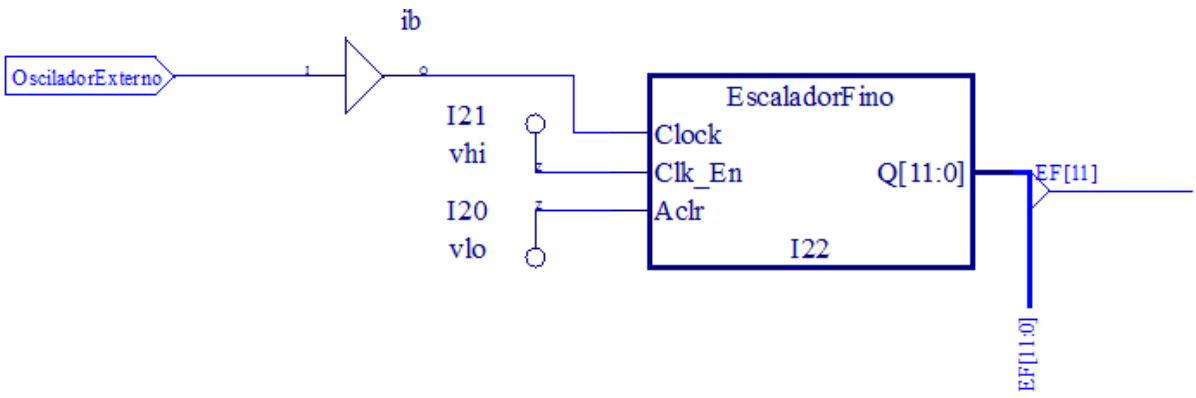


Figura 80: Esquema general Reloj Fino

3.1.3. Decodificador de distancias

Anteriormente, hemos dimensionado un escalador, que permite transformar la frecuencia de oscilación del reloj fino, a una frecuencia tal que la duración de cada pulso, coincide con la anchura de pulso que emite el sensor por el pin PW cuando mide una distancia de 1 cm, es decir, con el escalador anterior, hemos conseguido que cada pulso tenga una duración de $57.874015\ \mu s$.

El sensor de ultrasonidos, nos informa de la distancia medida a través de una onda en cuadratura que genera por su pin PW. En este pin, el tiempo que la onda permanece a '1' lógico es proporcional a la distancia, según la constante $147\mu s/2.54cm$.

Vamos a emplear un circuito decodificador TIEMPO (Proporcionado por el sensor) – DISTANCIA en BCD (Objetivo perseguido).

Para ello, vamos a emplear la señal PW del sensor como habilitación para un contador segmentado en cascada, que se encargar de contar los pulsos procedentes del escalador fino, que le llegan mientras la señal PW lo habilita.

El sensor de ultrasonidos es capaz de medir de 0 a 645 cm, y por su parte, hemos escogido la frecuencia del reloj fino para que 1 pulso, equivalga a 1 cm, con lo cual podremos contar de 0 a 645 pulsos o centímetros.

Así pues este sistema estará formado por un contador de módulo 646, construido a partir tres contadores BCD en cascada. Para construir este contador, emplearemos dos contadores módulo 10, que permitan contar de 0 a 9, y que servirán para posteriormente excitar los displays de 7 segmentos, y un tercer contador módulo 7, que permita contar de 0 a 6 que servirá para excitar los leds.

- Lógica de habilitación de contadores en cascada:
  La misión de esta lógica es permitir que los contadores en cascada sean habilitados justamente en el momento oportuno.
  El contador que proporciona el número menos significativo, será habilitado por el propio sensor de ultrasonidos.

$$ENABLE_0 = PWM$$

  El contador que proporciona el número intermedio será habilitado por el anterior, cuando este alcance la combinación.

$$ENABLE_1 = A_0\overline{B_0C_0}D_0\ (C0MBINACION\ 9)$$

  Finalmente, el contador que proporciona el número más significativo, será habilitado cuando los dos anteriores lleguen a 9.

$$ENABLE_2 = A_1\overline{B_1C_1}D_1 \cdot A_0\overline{B_0C_0}D_0\ \ (COBINACION\ 9\ 9)$$

Nótese que el haber elegido implantar un contador de pulsos, a partir de la interconexión en cascada de tres contadores, en lugar de utilizar exclusivamente un solo contador, no es casual, ya que de esta forma, nos aseguramos que cada uno de los tres dígitos que forman el valor de la distancia, se obtienen en formato BCD, y por lo tanto nos será posteriormente más fácil su tratamiento.

Para implementar el diseño anterior, creamos un nuevo proyecto llamado ContadoresModulo en el que utilizando IP-Express, describiremos dos tipos de contadores módulo, uno módulo 10 (10 estados diferentes) para contar pulsos de 0 a 9, y uno módulo 7, que será utilizado para contar pulsos de 0 a 6.

Los contadores módulo 10, tendrán una anchura del bus de datos de 4 bits, mientras que el contador módulo 7, la tendrá sólo de 3 bits.

A continuación se muestran las opciones elegidas para los diferentes contadores, en las figuras 81, 82, 84 y 85:

CONTADORES MODULO 10 (Cuenta de 0 a 9).

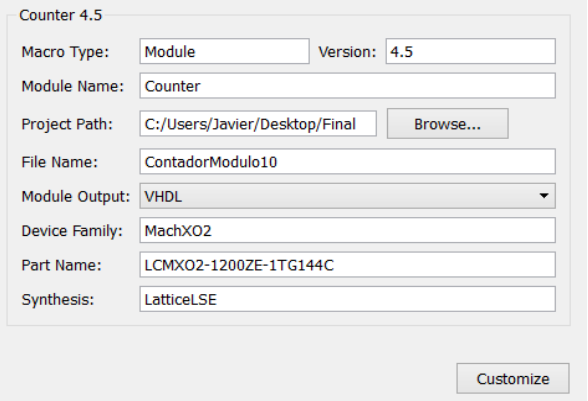


Figura 81: Opciones IPExpress (5)

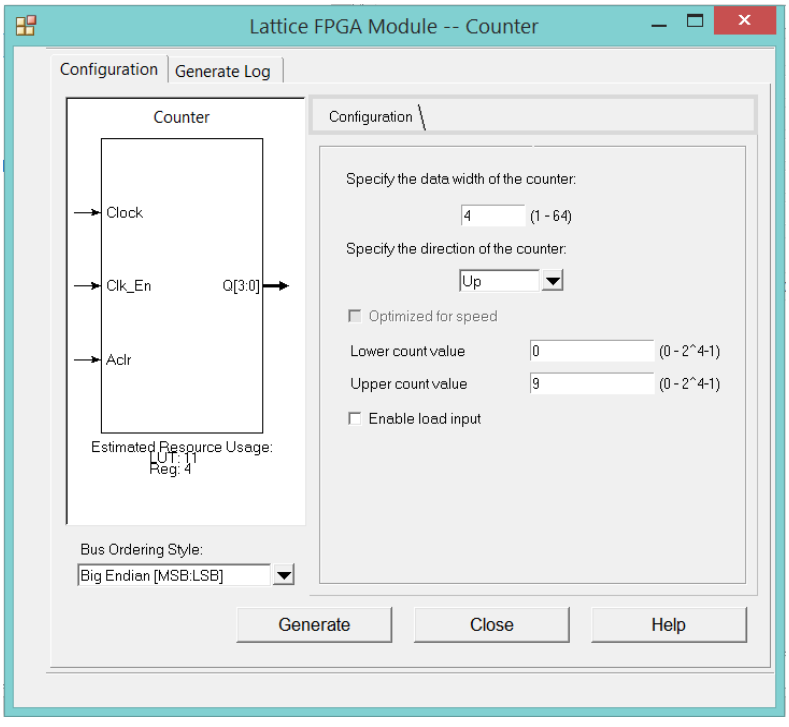


Figura 82 : Opciones IPExpress (6)

El resultado generado por la herramienta IPExpress se muestra en la figura 83:

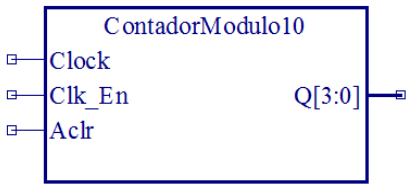


Figura 83: Símbolo Contador Módulo 10

CONTADOR MODULO 7 (Cuenta de 0 a 6).

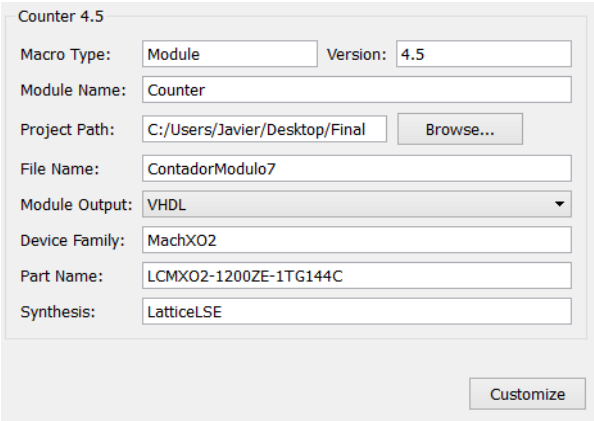


Figura 84: Opciones IPExpress (7)

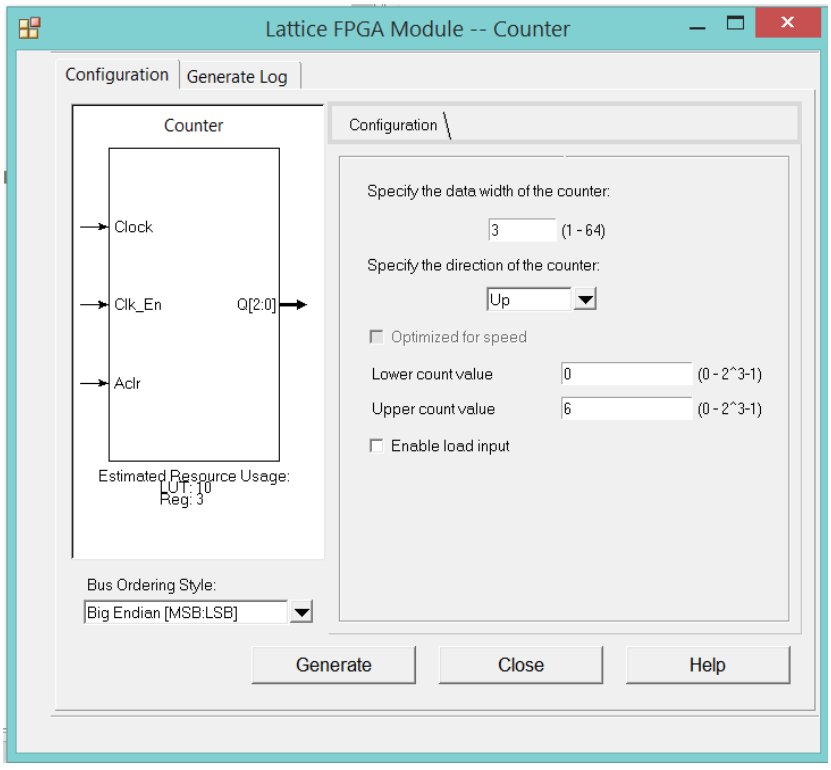


Figura 85: Opciones IPExpress (8)

El resultado generado por la herramienta IPExpress se muestra en la figura 86:

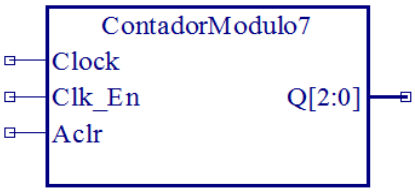


Figura 86: Símbolo Contador Módulo 7

De nada nos sirve haber creado los contadores módulo anteriores, si ahora no diseñamos la lógica necesaria para su interconexión en cascada.

Vamos a diseñar dicha lógica empleando código en VHDL. Para ello, creamos un nuevo proyecto llamado HabilitacionContadores.

Abrimos un nuevo fichero de tipo VHDL en el que describimos la siguiente lógica:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;


entity HABILITACION_CONTADORES is
        port(CUENTA_0, CUENTA_1: in STD_LOGIC_VECTOR (3 downto 0);
             ENABLE_2, ENABLE_1: out STD_LOGIC
            );
end HABILITACION_CONTADORES;

architecture ARQ_HABILITACION_CONTADORES of HABILITACION_CONTADORES is
begin

        ENABLE_2 <= '1' when CUENTA_0 = "1001" and CUENTA_1 = "1001" else '0';
        ENABLE_1 <= '1' when CUENTA_0 = "1001" else '0';

end ARQ_HABILITACION_CONTADORES;
```

Cuando salvemos el diseño, si no ha habido errores de sintaxis, se generara el bloque de la jerarquía (figura 87).

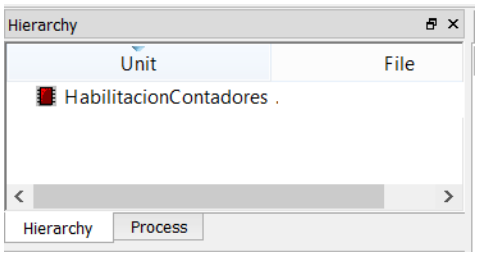


Figura 87: Jerarquía Habilitación de Contadores

De nuevo, si generamos el símbolo correspondiente, el resultado será (figura 88):

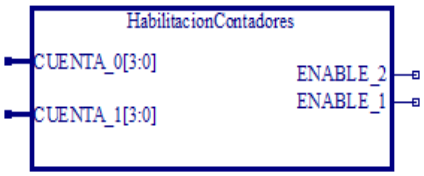


Figura 88: Símbolo Habilitación de Contadores

Finalmente, vamos a proceder a interconectar entre sí, todos los elementos que forman el bloque decodificador de distancias, en este caso, dos contadores módulo 10, un contador módulo 7, y un bloque de habilitación de los mismos.

Véase la figura 89.

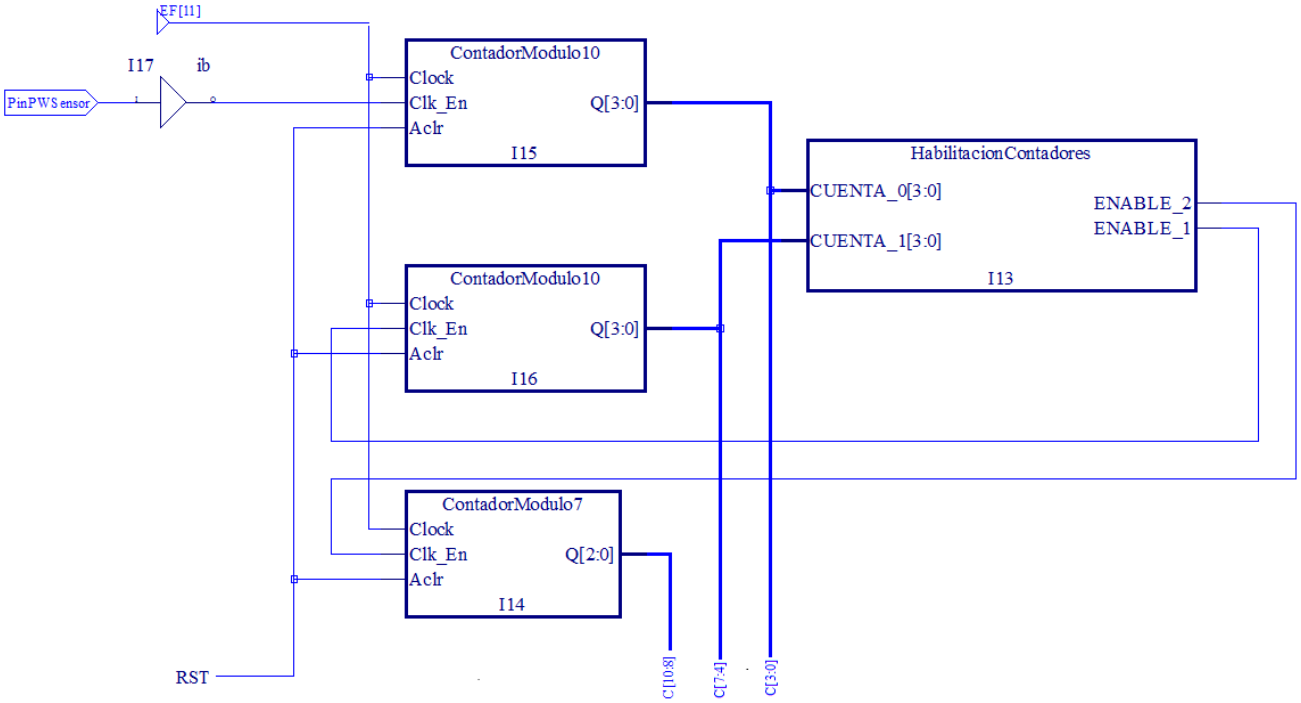


Figura 89: Esquema general Decodificación de Distancias

3.1.4. Registros de datos

Vamos a utilizar elementos de almacenamiento tipo D, con un ancho de palabra de 11 bits.

Estos elementos de almacenamiento, dispondrán de una entrada de reloj, activada por el generador de eventos según corresponda, una entrada de datos con un ancho de palabra de 11 bits, y una salida con un ancho de palabra de 11 bits también.

Crearemos un nuevo proyecto llamado Memoria, en el que procediendo de igual modo que antes, añadiremos un nuevo fichero de tipo VHDL en el que describiremos el código correspondiente al elemento de almacenamiento deseado:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

--Tipo de Registro   >> D
--Activo por         >> flanco ascendente
--Ancho de palabra >> 11 bits (4 x Display 0 + 4 x Display 1 + 3 x Leds on board)
entity Memoria is
        port(D: in STD_LOGIC_VECTOR (10 downto 0);
             CLK: in STD_LOGIC;
             Q: out STD_LOGIC_VECTOR (10 downto 0)
             );
end Memoria;

architecture ArqMemoria of Memoria is

begin
        process (CLK)  --Lista de sensibilidad
                begin
                        if (CLK' event and CLK='1') then
                                Q <= D;
                        end if;
        end process;

end ArqMemoria;
```

Como podemos ver en el código, se trata de un circuito secuencial, donde la lista de sensibilidad de nuestro process, solo está afectada por CLK, lo que quiere decir que únicamente entraremos a ejecutar las instrucciones contenidas en el process si se cambia el valor de la señal CLK.

Además, con la sentencia CLK'event and CLK='1', estamos especificando el flanco ascendente del reloj CLK.

CLK'event >> Flanco asociado a CLK.

CLK='1' >> Flanco de tipo ascendente.

Cuando salvemos el diseño, si no ha habido errores de sintaxis, se generara la jerarquía correspondiente, como la que aparece en la figura 90:

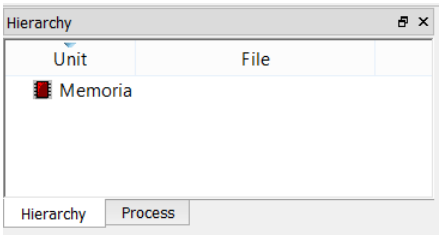


Figura 90: Jerarquía Registros de Datos

Una vez más será necesario, que hagamos clic derecho sobre la jerarquía y seleccionemos "Generate Schematic Symbol.

El bloque generado, se muestra en la figura 91:

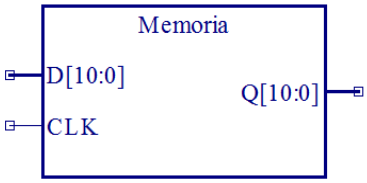


Figura 91: Símbolo Registro de Datos

Los elementos de almacenamiento anteriores serán usados en una única configuración similar a la del maestro-esclavo. Crearemos un registro de desplazamiento con capacidad para almacenar 2 datos con un ancho de palabra de 11 bits cada dato.

El objetivo de esta configuración es disponer de dos datos, que se van actualizando cada 50 ms, con los que podamos realizar el cálculo de velocidades, pues como ya vimos anteriormente, si tenemos dos datos de distancias (dato del maestro y dato del esclavo) y sabemos el lapso de tiempo que ha pasado entre la captura de datos (50 ms), podremos calcular la velocidad.

El dato que captura el registro maestro, será el dato procedente del circuito de conversión tiempo – distancia, mientras que el dato que captura el registro esclavo, es el procedente del maestro.

Los elementos de almacenamiento van a ser actualizados con sus propias señales de refresco, donde como ya vimos en el generador de eventos, la señal de refresco de registro esclavo, va adelantada respecto a la del maestro, para que de esta forma, el esclavo, pueda tomar el dato del maestro antes de que este lo actualice.

La configuración usada se muestra en la figura 92:

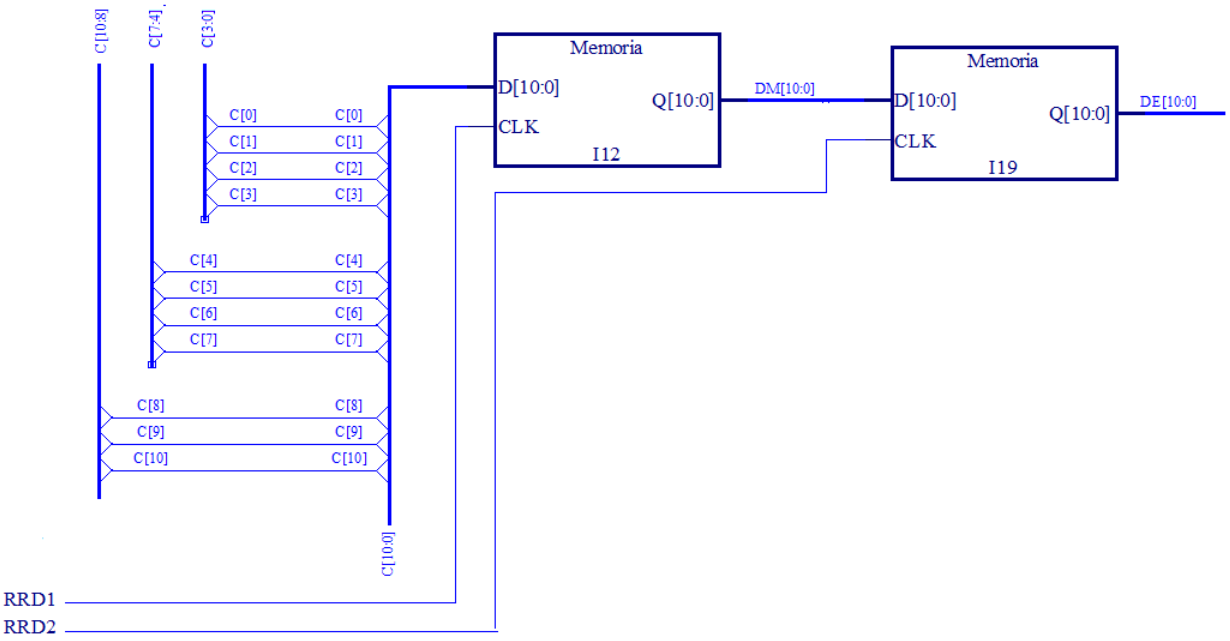


Figura 92: Esquema general Registros de Datos

3.1.5. Cálculo de velocidad

Vamos a diseñar un circuito encargado de obtener la velocidad a partir de dos medidas de distancia.

Sabemos que nuestro sistema va a captar dos valores de distancia con un lapso entre medidas de $^{1}/_{20\ Hz} = 50\ ms$.

La velocidad se puede calcular fácilmente con la ecuación $V = {}^{s}/_{t}$

Como valor de espacio, podemos tomar la diferencia entre dos distancias medidas de forma consecutiva, medidas almacenadas en los elementos de memoria Maestro – Esclavo. Como valor de tiempo, tomaremos los 50 ms anteriormente mencionados, ya que es el lapso entre dos medidas consecutivas.

Si deseamos obtener la velocidad en unidades del S.I, $^{[m]}/_{[s]}$, deberemos usar un factor de conversión al que denominaremos K.

$$\frac{[cm]}{0.05\ s} = K\frac{[m]}{s}$$

$$K = \frac{1}{1000} \cdot 0.05\ ; K = \frac{1}{20000}$$

Así, pues, vemos que deberemos multiplicar la velocidad obtenida como resta de dos medidas consecutivas de distancia por K, para obtener el resultado final.

Implementaremos el diseño del circuito de velocidad mediante un código VHDL, que escribiremos en el proyecto Velocidad.

```
library IEEE;
use IEEE.STD_LOGIC_1164.alL;          --Contiene los tipos de datos STD_LOGIC

use IEEE.STD_LOGIC_ARITH.all;         --Obligatoria para hacer conversiones
use IEEE.STD_LOGIC_UNSIGNED.all;      --Idem
use IEEE.NUMERIC_STD.all;             --Idem

entity Velocidad is
        port(LCV: in STD_LOGIC;
             DATOA: in STD_LOGIC_VECTOR (10 downto 0);
             DATOB: in STD_LOGIC_VECTOR (10 downto 0);
             VALORA, VALORB: inout INTEGER;
             VELOCIDAD: out STD_LOGIC_VECTOR (10 downto 0)
             );
end Velocidad;

architecture ArqVelocidad of Velocidad is
begin

        process (LCV)
        begin

                --Transformamor el DATOA a integer
                VALORA<=conv_integer(DATOA);
                --Transformamor el DATOA a integer
                VALORB<=conv_integer(DATOB);

                if (VALORA>VALORB) then
                     VELOCIDAD<=conv_std_logic_vector(((VALORA-VALORB)/20000),11);
                     --Conversion a BIT_VECTOR[0:10]
                end if;

                if (VALORA<VALORB) then
                     VELOCIDAD<=conv_std_logic_vector(((VALORB-VALORA)/20000),11);
                     --Conversion a BIT_VECTOR[0:10]
                end if;

                if (VALORA=VALORB) then
                    VELOCIDAD<="00000000000";
                end if;

        end process;

end ArqVelocidad;
```

En el código anterior, hemos trabajado con dos variables internas, VALORA y VALORB, que representan los datos DATOA y DATOB, en formato integer.

VHDL, dispone de un amplio conjunto de operaciones aritméticas y relacionales:

OPERADORES ARITMÉTICOS	
SUMA	A+B
RESTA	A-B
PRODUCTO	A*B
DIVISIÓN	A/B
EXPONENCIACIÓN	A**B
MODULO	A mod B
RESTO DE LA DIVISIÓN ENTERA	A rem B
VALOR ABSOLUTO	abs A

OPERADORES RELACIONALES	
IGUAL	A=B
DIFERENTE	A/=B
MAYOR QUE	A>B
MENOR QUE	A<B
MAYOR O IGUAL QUE	A>=B
MENOR O IGUAL QUE	A<=B

NOTA: En color verde, se muestran los operadores empleados en el código anterior.

Sin embargo, todas estas operaciones están solo disponibles para datos de tipo INTEGER. Por este motivo, me he visto obligado a emplear librerías y sentencias de conversión.

Para finalizar, como podemos observar, la lista de sensibilidad solo está afectada por la señal CLK, lo cual implica que el circuito de cálculo de velocidades, deberá funcionar de forma síncrona con la señal de reloj con que se le alimente, procedente del generador de eventos, como ya dijimos.

Cuando salvemos el diseño, si no ha habido errores de sintaxis, se generará la jerarquía correspondiente (figura 93).

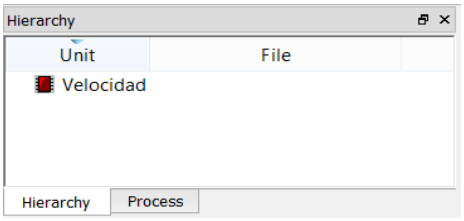


Figura 93: Jerarquía Cálculo de Velocidad

De nuevo, será necesario, que hagamos clic derecho sobre la jerarquía y seleccionemos "Generate Schematic Symbol.

El bloque generado, será finalmente (figura 94):

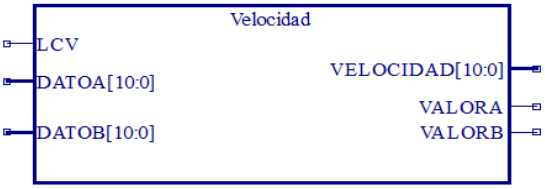


Figura 94: Símbolo Cálculo de Velocidad

Finalmente, si interconectamos el bloque Velocidad anterior con todos los elementos con los que interactuará, obtendremos el esquema de la figura 95:

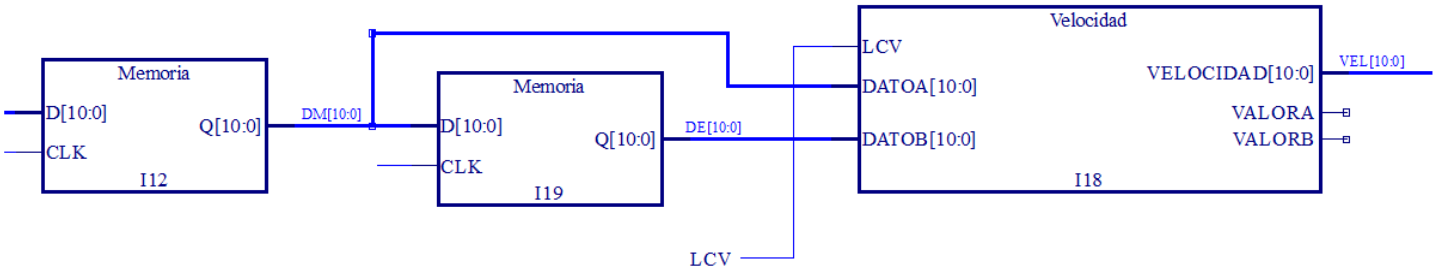


Figura 95: Esquema general Cálculo de velocidad

3.1.6. Multiplexor

Usaremos un circuito multiplexor, cuya función sea la de permitir que el usuario, decida qué información quiere visualizar, bien la información de la distancia, caso de SELECCIÓN = '0' o bien la información de velocidades, caso de SELECCIÓN = '1'.

Implementaremos el diseño del circuito multiplexor empleando código VHDL. Dicho código, lo escribiremos en un nuevo fichero contenido en el proyecto Multiplexor.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Mutiplexor is
        port(SELECCION: in STD_LOGIC;
             DISTANCIA: in STD_LOGIC_VECTOR (10 downto 0);
             VELOCIDAD: in STD_LOGIC_VECTOR (10 downto 0);
             SALIDA: out STD_LOGIC_VECTOR (10 downto 0)
             );
end Multiplexor;

architecture ArqMultiplexor of Multiplexor is
begin

            SALIDA <= DISTANCIA when SELECCION = '0' else VELOCIDAD;

end ArqMultiplexor;
```

Haciendo clic en el botón de salvar, automáticamente, se generara la arquitectura, a no ser que haya habido errores de compilación, como se muestra en la figura 96:

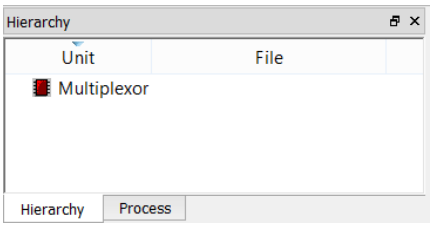


Figura 96: Jerarquía Multiplexor

Si generamos el símbolo correspondiente, obtendremos el resultado de la figura 97:

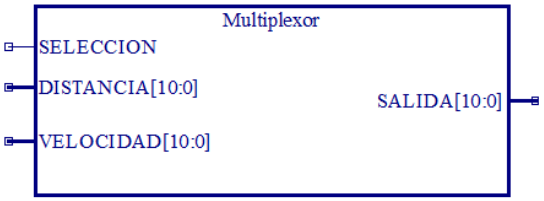


Figura 97: Símbolo Multiplexor

Finalmente, si interconectamos el bloque MULTIPLEXOR anterior con todos los elementos con los que interactuará, obtendremos el resultado de la figura 98:

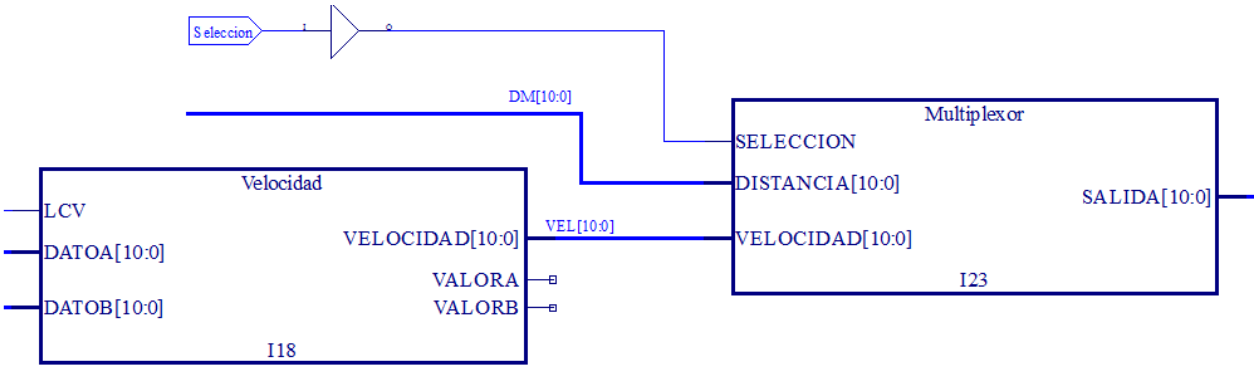


Figura 98: Esquema general Multiplexor

3.1.7. Generador de la señal de visualización VIS

Vamos a construir un último generador de señal, que nos proporcione la señal de visualización, a la que llamaremos VIS.

La misión de esta señal, es permitir actualizar los displays y leds, con los que el usuario obtiene la información de la distancia/velocidad, de segundo en segundo.

Para crear esta señal, procederemos a dimensionar un contador, que alimentado por el oscilador interno de la FPGA, nos permita escalar la frecuencia de oscilación, de los 2.08 MHz, a 1 Hz, y proceder a su implementación con un bloque IPExpress.

$$2.08\ \text{Mhz} = 2080000\ \text{Hz}.$$

$$\frac{2080000\text{Hz}}{2080000} = 1\ \text{Hz}.$$

De esta manera, necesitaremos un contador de 2080000 (Dos millones ochenta mil) estados distintos.

El cálculo del número de bits es inmediato sabiendo que necesitamos, por lo menos 2080000 estados diferentes:

$2^{20} = 1048576\ ;\ 2^{21} = 2097152.$

Todo lo anterior nos muestra que necesitamos un contador de 21 bits con cuenta de 1 a 2080000.

Dicho contador será implementado mediante IP Express, (figuras 99 y 100) en el proyecto llamado SennalVis.

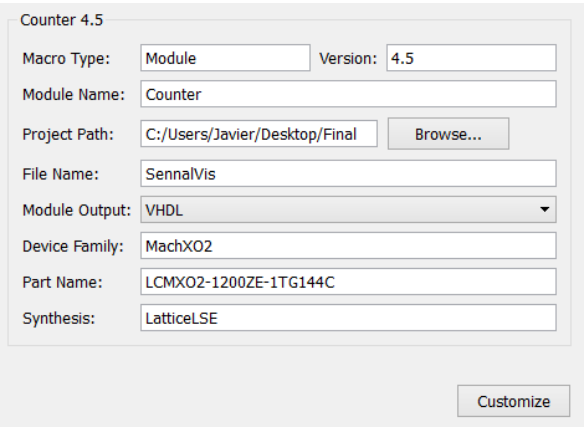


Figura 99: Opciones IPExpress (9)

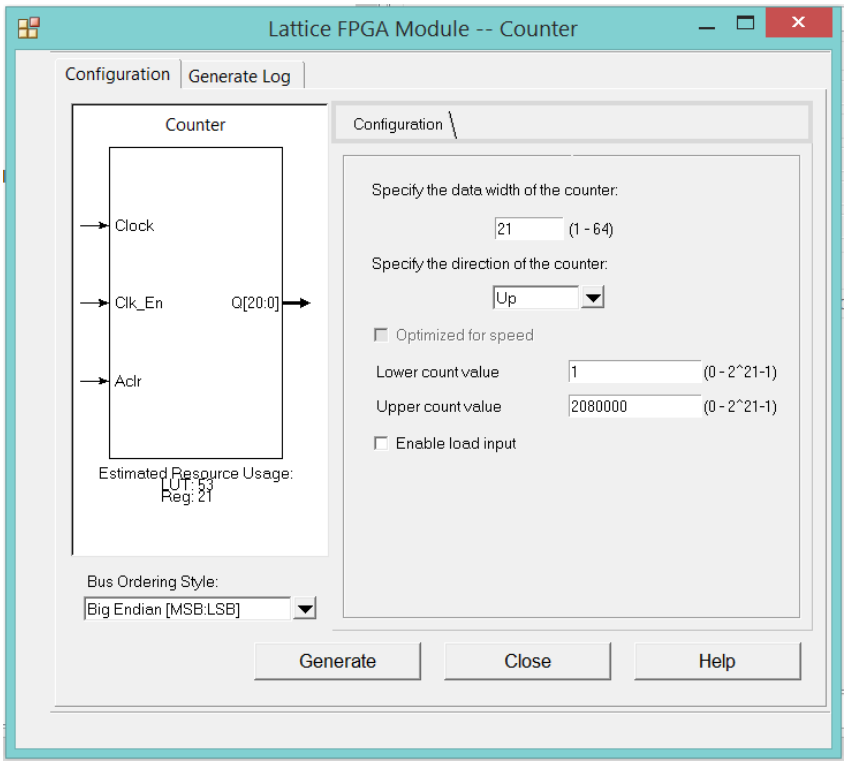


Figura 100: Opciones IPExpress (10)

Tras hacer clic en “Generate”, habremos terminado.

El símbolo de nuestro nuevo contador se muestra en la figura 101:

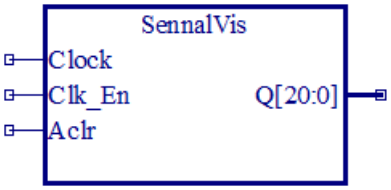


Figura 101: Símbolo Señal Visualización

Si alimentamos Clock con la salida del bloque “osch”, y tomamos el bit de mayor peso de la salida Q, conseguiremos escalar el reloj de 2.08 MHz, a aproximadamente 1 Hz, obteniendo la señal VIS deseada.

3.1.8. Decodificador BCD a 7 segmentos

Como ya hemos expuesto, internamente, representamos los números en codificación BCD.

Para poder mostrar al usuario la distancia/velocidad, será necesario emplear bloques decodificadores BCD a 7 segmentos.

Usaremos dos decodificadores BCD a 7 segmentos, uno de ellos, que permita mostrar el número de menor peso, (unidades de cm) y el otro que permita mostrar el número de mayor peso (decenas de cm).

Vamos a configurar estos decodificadores para que se actualicen de segundo en segundo. Cada segundo, se producirá la decodificación correspondiente, y se mantendrá constante hasta el siguiente ciclo.

En primer lugar, vamos a describir el comportamiento de dichos decodificadores usando código VHDL. Para ello, creamos un nuevo proyecto llamado Bcd7Seg.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Bcd7Seg is
	port(BCD: in STD_LOGIC_VECTOR (3 downto 0);
		VIS: in STD_LOGIC;
		 SEG: out STD_LOGIC_VECTOR (6 downto 0)
		);
end Bcd7Seg;

architecture ArqBcd7Seg of Bcd7Seg is
--Entrada: Código BCD de 4 bits
--Salida: Código de 7 bits que excitan un diplay 7 segmentos
	constant CERO: STD_LOGIC_VECTOR (6 downto 0) := "1111110";
	constant UNO: STD_LOGIC_VECTOR (6 downto 0) := "0110000";
	constant DOS: STD_LOGIC_VECTOR (6 downto 0) := "1101101";
	constant TRES: STD_LOGIC_VECTOR (6 downto 0) := "1111001";
	constant CUATRO: STD_LOGIC_VECTOR (6 downto 0) := "0110011";
	constant CINCO: STD_LOGIC_VECTOR (6 downto 0) := "1011011";
	constant SEIS: STD_LOGIC_VECTOR (6 downto 0) := "1011111";
	constant SIETE: STD_LOGIC_VECTOR (6 downto 0) := "1110000";
	constant OCHO: STD_LOGIC_VECTOR (6 downto 0) := "1111111";
	constant NUEVE: STD_LOGIC_VECTOR (6 downto 0) := "1111011";
	constant OTRO: STD_LOGIC_VECTOR (6 downto 0) := "0000000";
```

```
begin
        process(VIS)
        begin
                if (VIS'event and VIS='1') then
                        case BCD is
                                when "0000" => SEG <= CERO;
                                when "0001" => SEG <= UNO;
                                when "0010" => SEG <= DOS;
                                when "0011" => SEG <= TRES;
                                when "0100" => SEG <= CUATRO;
                                when "0101" => SEG <= CINCO;
                                when "0110" => SEG <= SEIS;
                                when "0111" => SEG <= SIETE;
                                when "1000" => SEG <= OCHO;
                                when "1001" => SEG <= NUEVE;
                                when others => SEG <= OTRO;
                        end case;
                end if;
        end process;
end ArqBcd7Seg;
```

Como podemos ver en el código, el BCD a 7 Segmentos, se actualiza únicamente en los flancos ascendentes de la señal de VIS.

Con la sentencia VIS'event and VIS='1', estamos especificando el flanco ascendente de la señal VIS.

VIS'event >> Flanco asociado a señal VIS.

VIS='1' >> Flanco de tipo ascendente.

Haciendo clic en el botón de salvar, automáticamente, se generara la arquitectura del decodificador, a no ser que haya habido errores de compilación (figura 102):

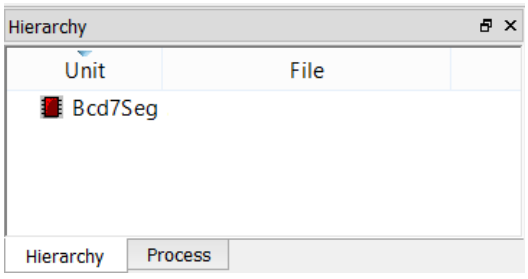


Figura 102: Jerarquía Bcd a 7 Segmentos

Finalmente, si generamos un símbolo, mediante DESIGN – GENERATE SYMBOL, obtendremos el resultado de la figura 103:

Figura 103: Símbolo Bcd a 7 Segmentos

El esquema de interconexionado de los decodificadores Bcd a 7 Segmentos con el resto del circuito se muestra en la figura 104:

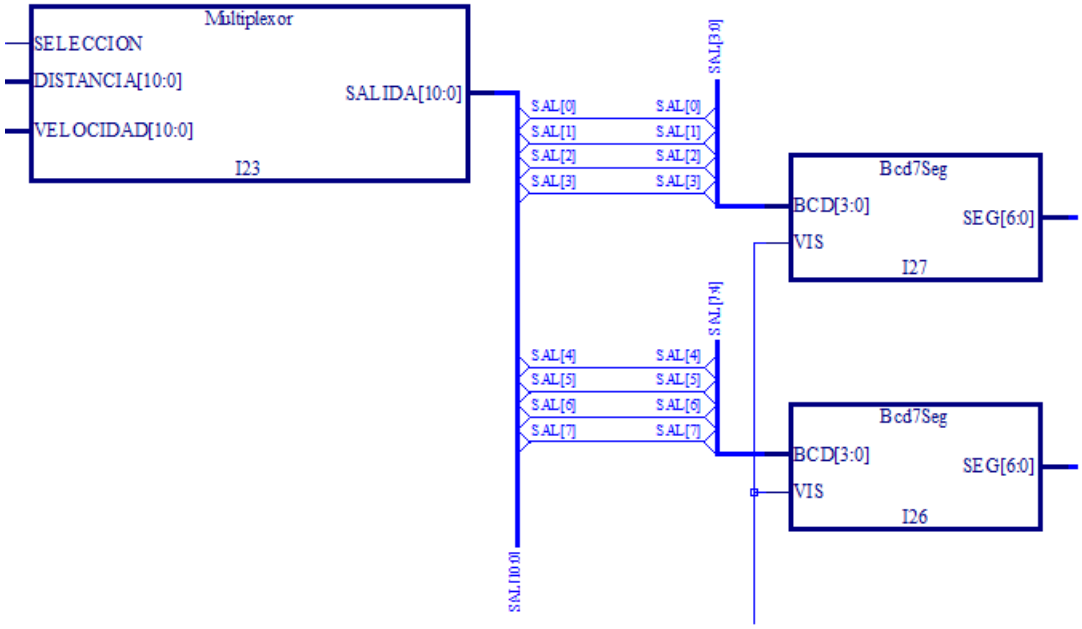


Figura 104: Esquema general BCD a 7 Segmentos

3.1.9. Leds on board

Los leds, son los dispositivos encargados de mostrar al usuario en valor final de las centenas de la distancia/velocidad medida.

Recordemos que, puesto que sólo disponemos de dos displays de siete segmentos, y deseamos mostrar números de hasta tres cifras decimales, es necesario, introducir algún elemento adicional que nos permita mostrara las centenas.

La forma de mostrar al usuario las medidas, puede ser la siguiente:

DISTANCIA/VELOCIDAD	LED'S EN ON (Centenas)	NUMERACIÓN EN DISPLAYS (Unidades y Decenas)
0 a 100 cm O m/s	Ninguno	De 00 a 99
100 a 200 O m/s	D1	
200 a 300 O m/s	D1+D2	
300 a 400 O m/s	D1+D2+D3	
400 a 500 O m/s	D1+D2+D3+D4	
500 a 600 O m/s	D1+D2+D3+D4+D5	
600 a 700 O m/s	D1+D2+D3+D4+D5+D6	

Tabla 2: Visualización de distancias

El sistema encargado de mostrar las centenas, es decir, de manejar los LED'S, podrá ser fácilmente implementado en VHDL. Para ello, creamos un nuevo proyecto llamado SelectorLeds.

En este nuevo proyecto, escribimos el siguiente código:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity SelectorLeds is
	port(BCD: in STD_LOGIC_VECTOR (2 downto 0);
		VIS: in STD_LOGIC;
		LED: out STD_LOGIC_VECTOR (5 downto 0)
		);
end SelectorLeds;

architecture ArqSelectorLeds of SelectorLeds is
--Entrada: Codigo BCD correspondiente al contador Q2 integrado en CONTADOR_PPAL
--Salida: Vector de 6 bits que excita leds MSB,LSB L5 a L0

	constant NADA: STD_LOGIC_VECTOR (5 downto 0) := "111111";
	constant LED_1: STD_LOGIC_VECTOR (5 downto 0) := "111110";
	constant LED_12: STD_LOGIC_VECTOR (5 downto 0) := "111100";
	constant LED_123: STD_LOGIC_VECTOR (5 downto 0) := "111000";
	constant LED_1234: STD_LOGIC_VECTOR (5 downto 0) := "110000";
	constant LED_12345: STD_LOGIC_VECTOR (5 downto 0) := "100000";
	constant LED_123456: STD_LOGIC_VECTOR (5 downto 0) := "000000";


	begin

		process (VIS)
		begin

			if (VIS'event and VIS='1') then
				case BCD is
					when "001" => LED <= LED_1;
					when "010" => LED <= LED_12;
					when "011" => LED <= LED_123;
					when "100" => LED <= LED_1234;
					when "101" => LED <= LED_12345;
					when "110" => LED <= LED_123456;
					when others => LED <= NADA;
				end case;
			end if;
		end process;

end ArqSelectorLeds;
```

Como se puede ver en el código VHDL anterior, estamos encendiendo los LEDS, con un ‘0’ lógico, en lugar de con un ‘1’ lógico, como parece más obvio.

Ello se debe a que físicamente, dichos leds se encuentran en una configuración de ánodo común.

Como en los casos anteriores, vemos que se trata de un circuito secuencial, activado por la señal VIS.

Cuando salvemos el diseño, se generara la correspondiente jerarquía (figura 105):

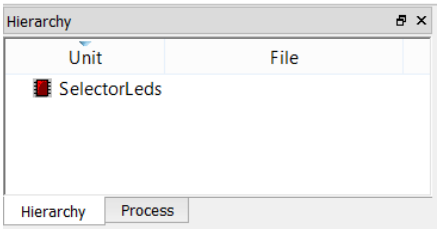


Figura 105: Jerarquía Selector de Leds

Generando el símbolo para el esquema anterior, obtendremos el resultado de la figura 106:

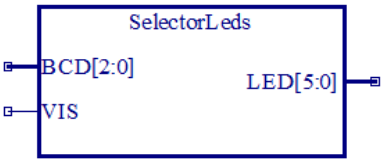


Figura 106: Símbolo Selector de Leds

El esquema de interconexionado con el resto del circuito se muestra en la figura 107:

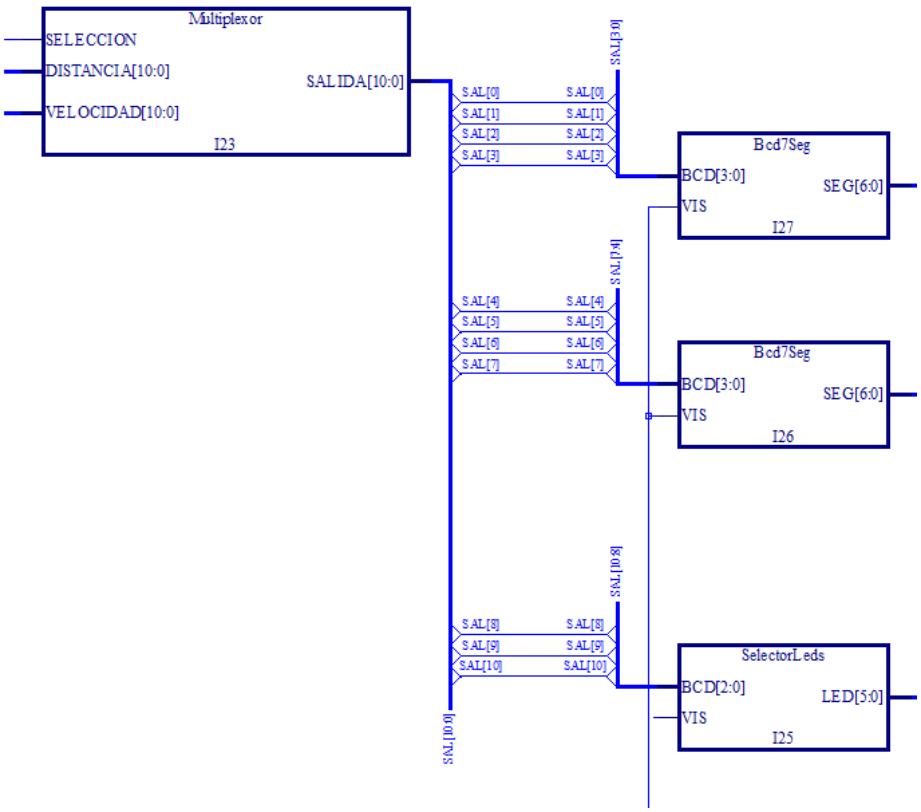


Figura 107: Esquema general Selector de Leds

3.1.10. Habilitación del oscilador externo

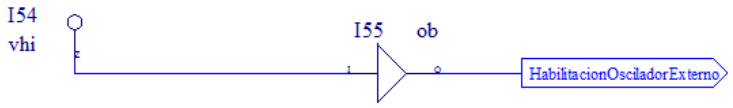


Figura 108: Esquema general Habilitación del Oscilador Externo

3.1.11. Esquema general

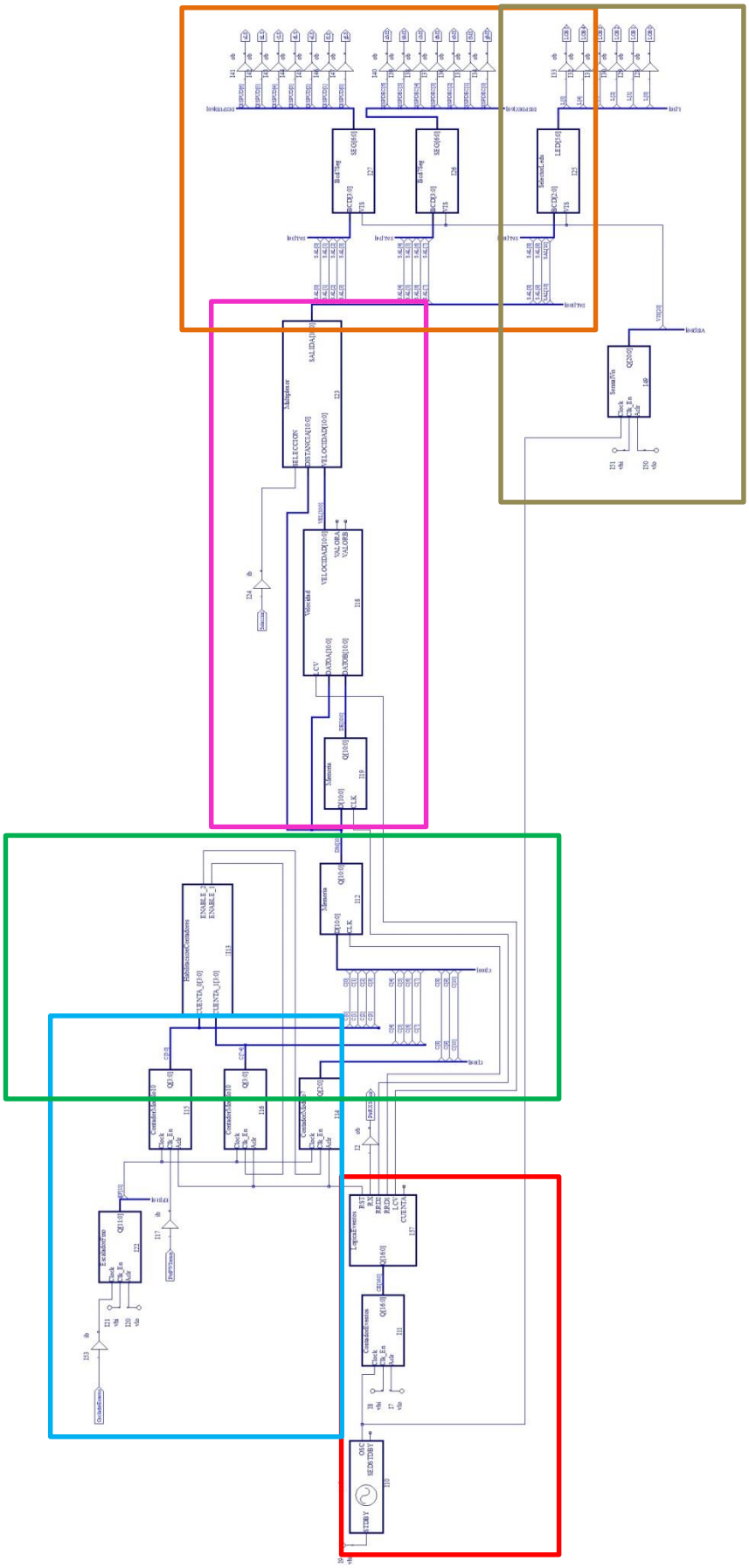


Figura 109: Esquema general

Si ampliamos el esquema anterior por cuadrantes, podemos ver:

CUADRANTE 1 (rojo)

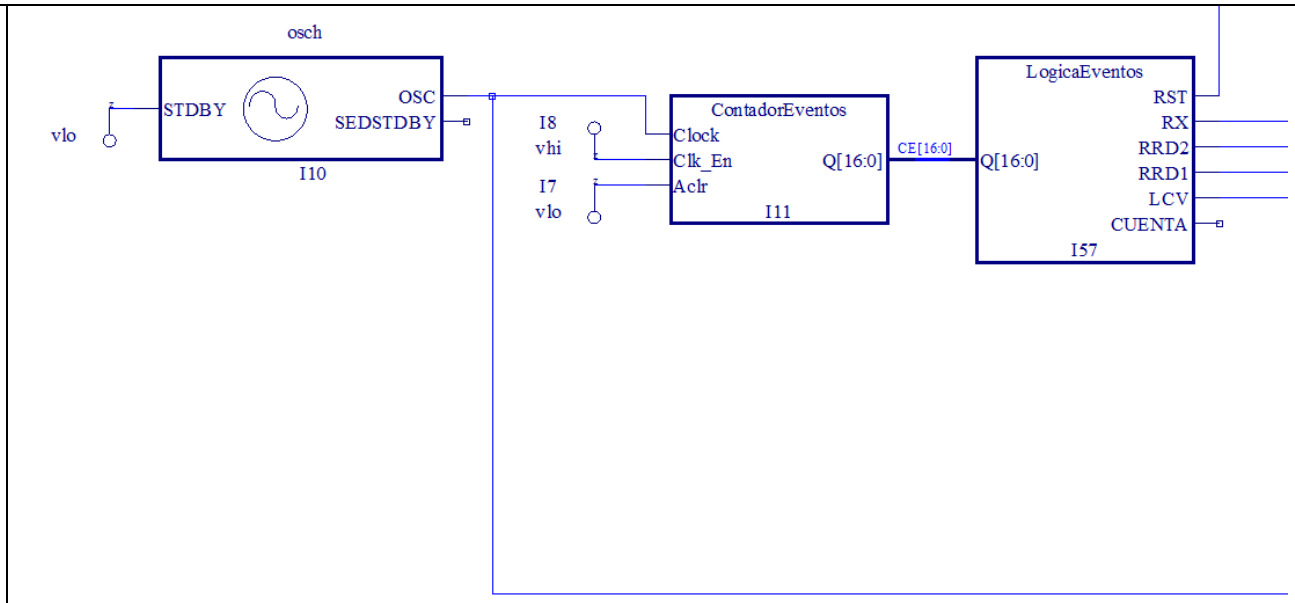


Figura 110: Detalle del módulo de generación de eventos

CUADRANTE 2 (azul)

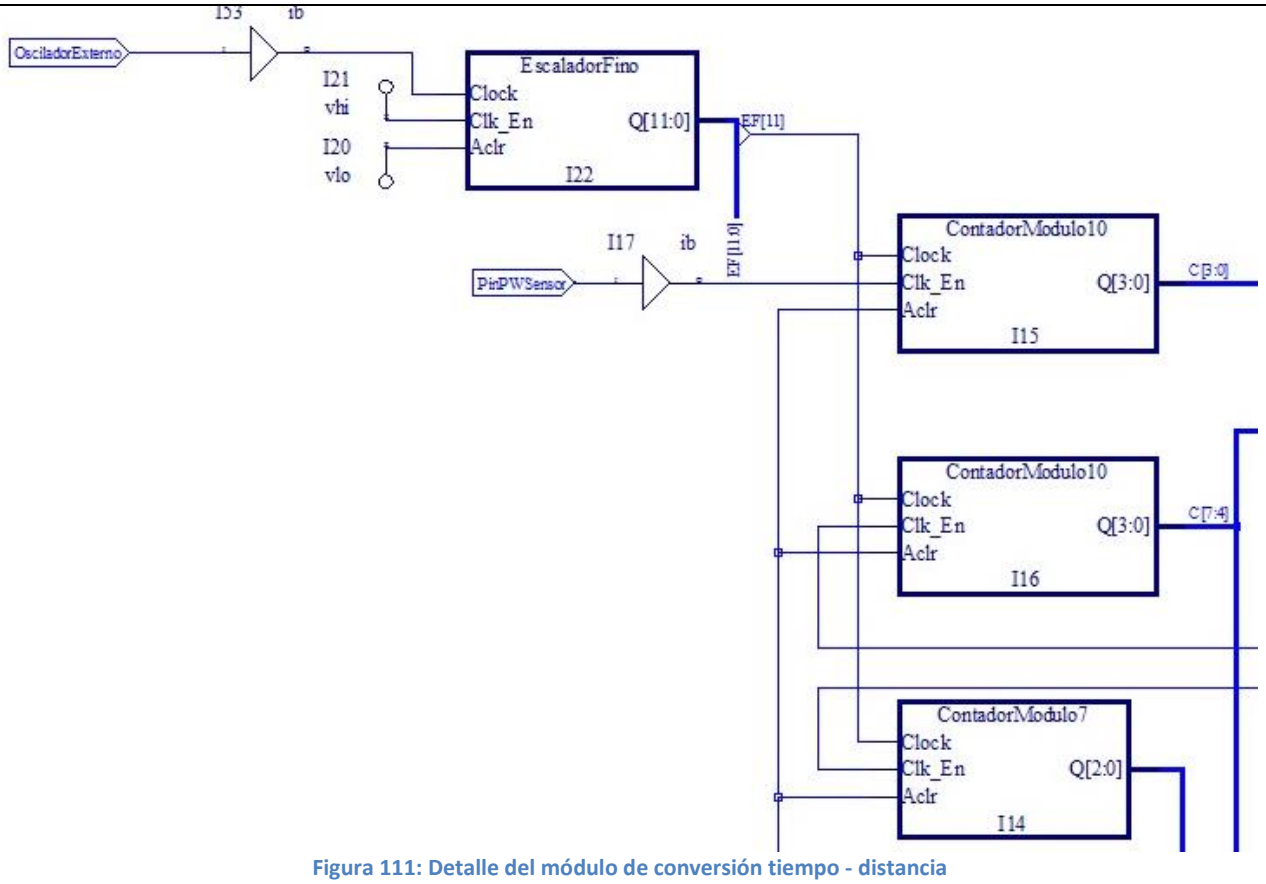


Figura 111: Detalle del módulo de conversión tiempo - distancia

CUADRANTE 3 (verde)

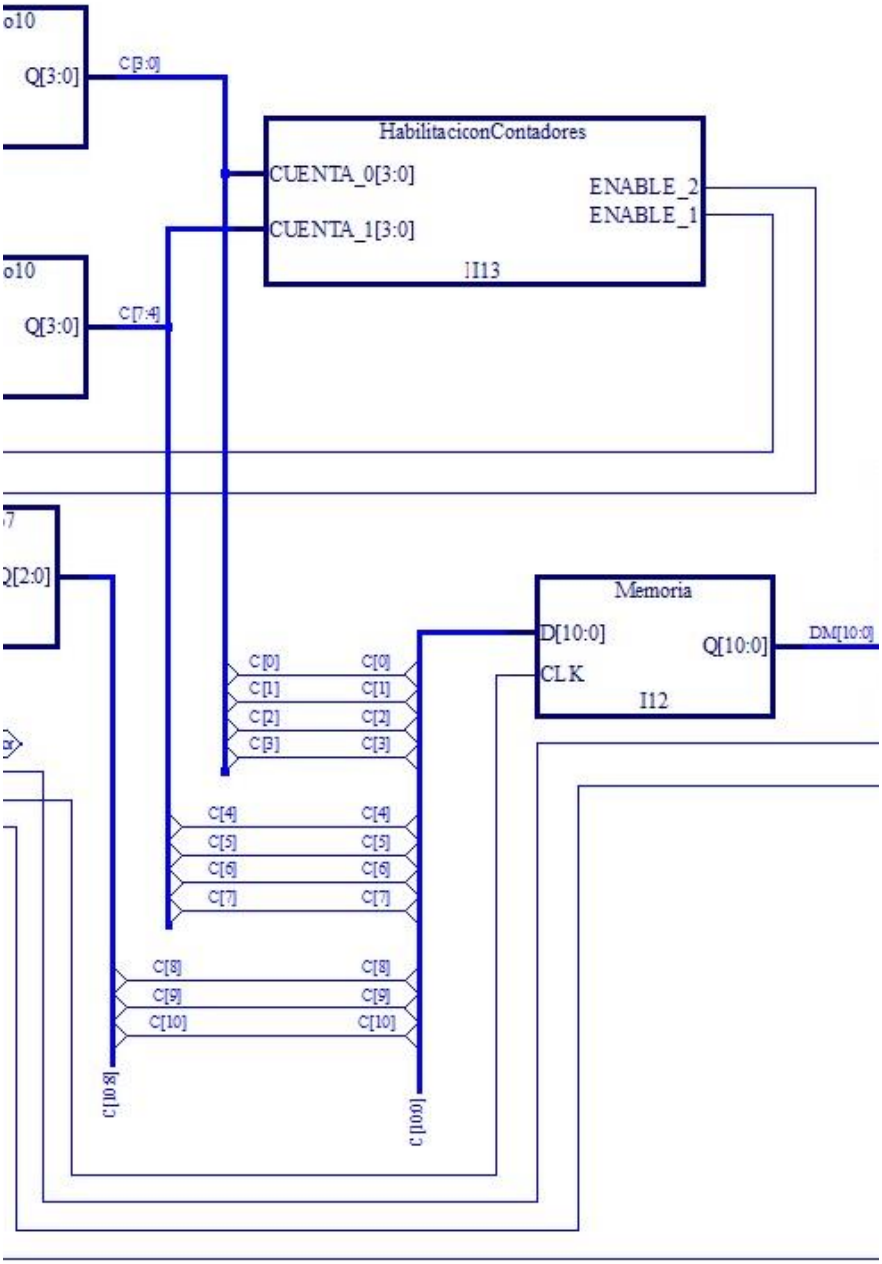


Figura 112: Detalle de la habilitación de contadores y el registro maestro

CUADRANTE 4 (rosa)

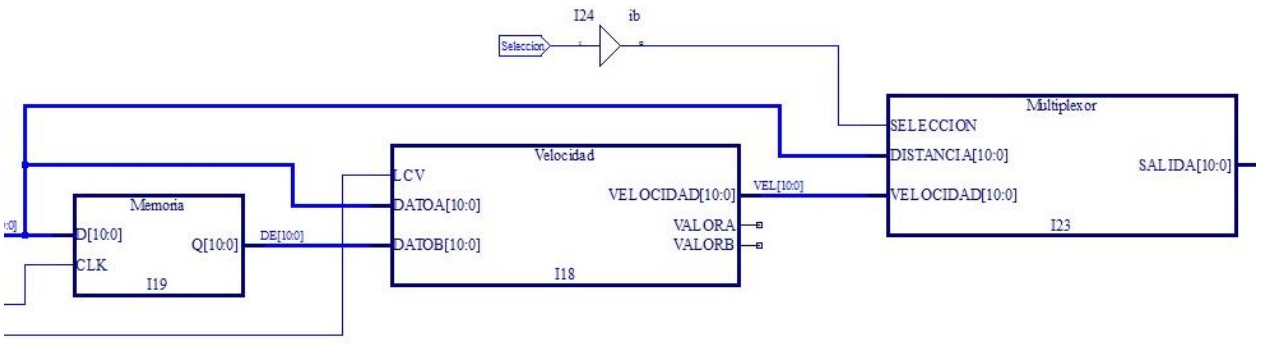


Figura 113: Detalle del registro esclavo, el cálculo de velocidad y la selección de datos

CUADRANTE 5 (naranja)

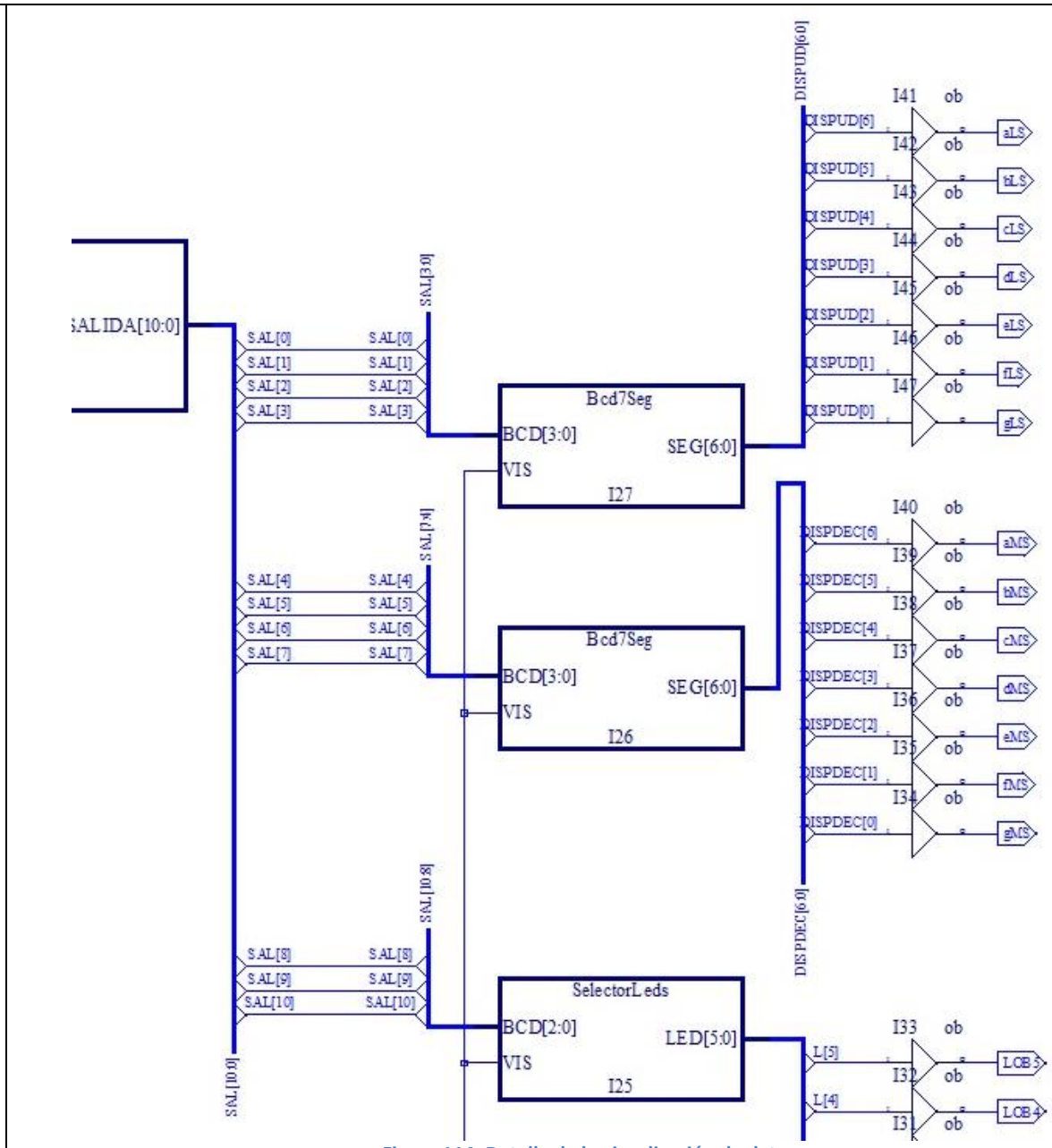


Figura 114: Detalle de la visualización de datos

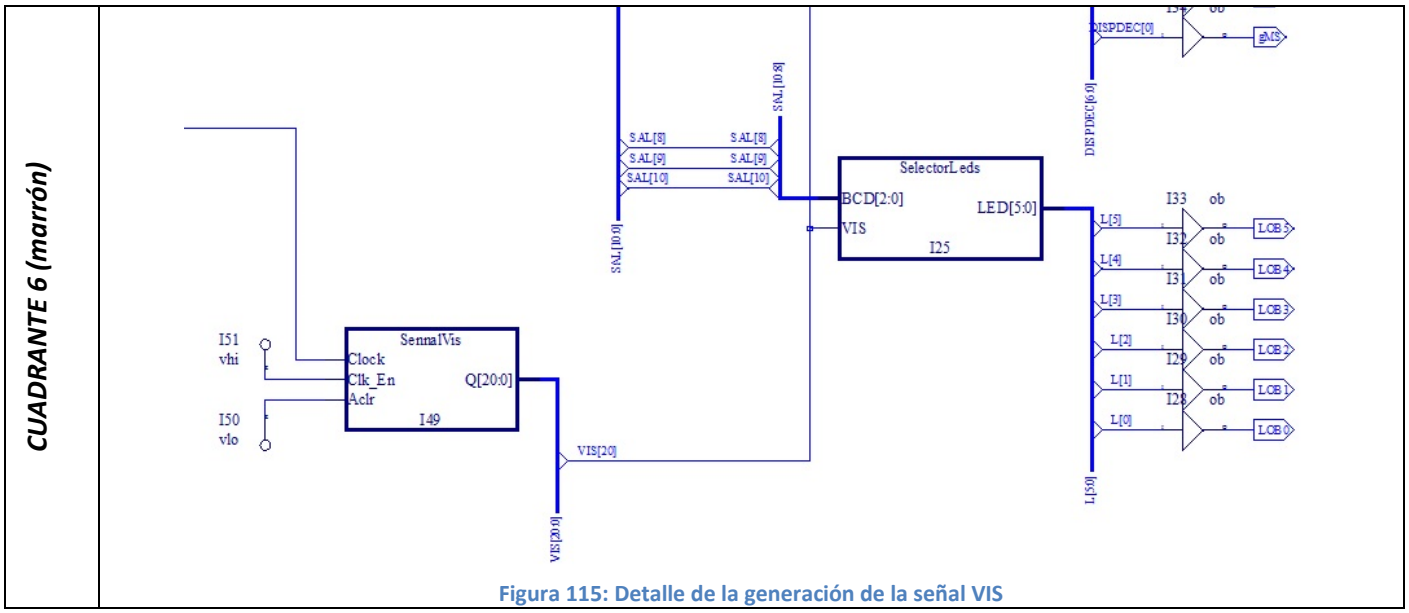


Figura 115: Detalle de la generación de la señal VIS

3.2. Herramientas de síntesis

Diamond dispone de dos herramientas de síntesis.

Estas herramientas, son las que nos permiten acoplar un diseño en un dispositivo lógico reconfigurable concreto. Éstas, son además, las herramientas que permiten, determinar la cantidad de lógica que es necesaria para un determinado diseño.

Vamos a ver, cómo pueden cambiar los porcentajes de utilización de lógica, dependiendo de la herramienta de síntesis empleada.

Podemos cambiar la herramienta de síntesis haciendo clic en Project, Active Implementation, y finalmente Select Synthesis Tool…, como se muestra en la figura 116.

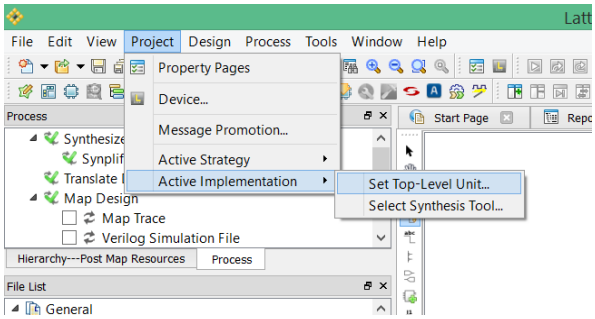


Figura 116: Cambio de herramienta de síntesis (1)

Se nos abrirá una ventana como la mostrada en la figura 117:

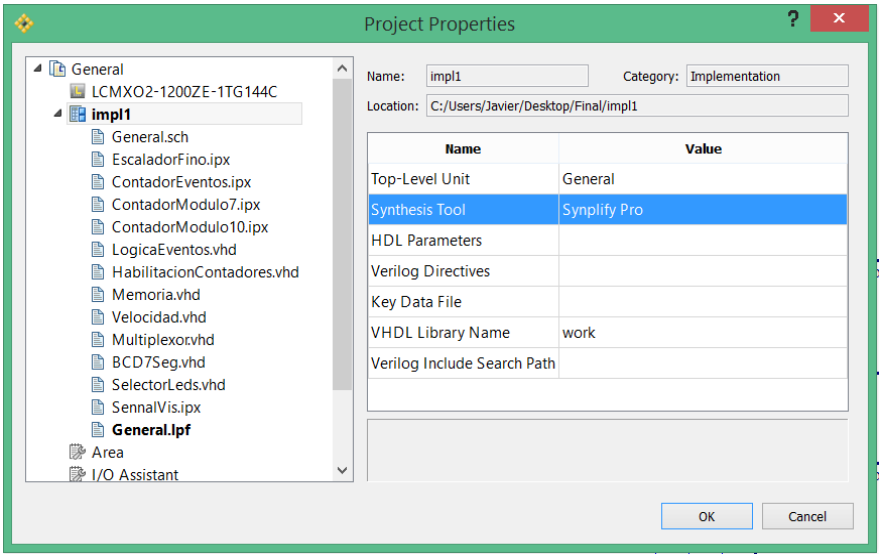


Figura 117: Cambio de herramienta de síntesis (2)

En dicha ventana, se nos permite elegir entre dos herramientas de síntesis:

Synplify Pro: Herramienta específica de síntesis. Con ella, se obtienen mejores resultados, y mayor grado de optimización

Lattice LSE: Herramienta genérica de síntesis. Con ella, se obtienen resultados peores, y menos optimizados.

Para comenzar la síntesis de un diseño, es preciso, seguir los siguientes pasos:

1. Añadir a nuestro proyecto General, los ficheros donde se describen los bloques usados. Para ello, debemos distinguir dos casos:
   a. El bloque fue generado con IPExpress. Añadiremos el bloque de extensión ".ipx" correspondiente.
   b. El bloque fue descrito mediante código escrito en VHDL. Añadiremos el bloque de extensión ".vhd" correspondiente

   Para añadir los ficheros, debemos hacer clic en File, Add, Existing File (figura 118).

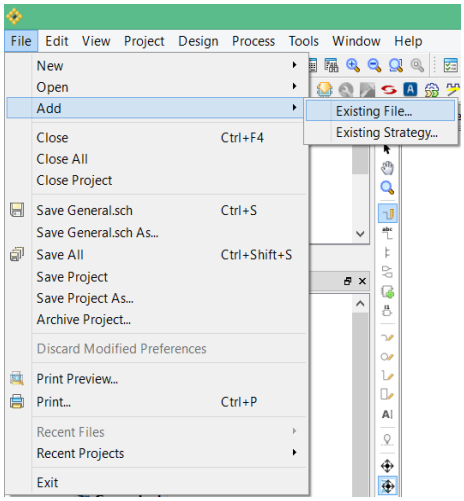


Figura 118: Adición de ficheros existentes (1)

Una vez añadidos todos los ficheros, deberemos obtener un resultado como el que se muestra en la figura 119:

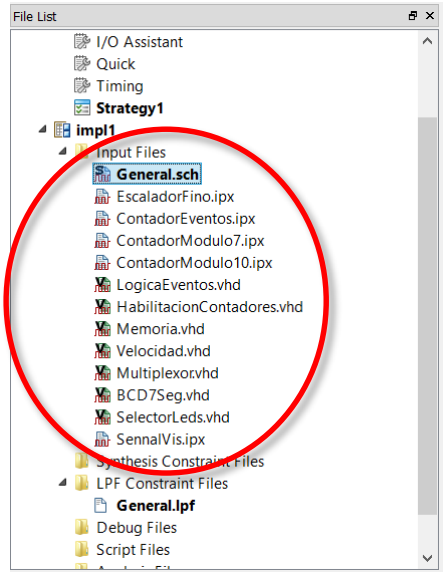


Figura 119 : Adición de ficheros existentes (2)

2. Establecemos nuestro esquema general como Top Level de la jerarquía. Para ello, en la ventana de jerarquía, hacemos clic izquierdo sobre General, y seleccionamos Set as top level unit, como se muestra en la figura 120:

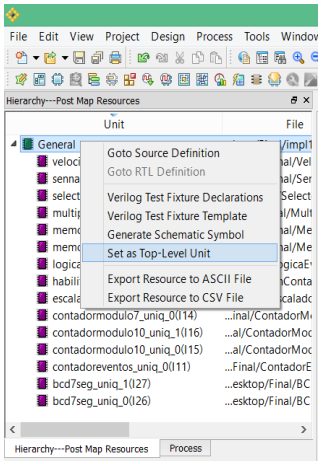


Figura 120: Selección del TOP LEVEL de un diseño

Ahora vamos a realizar la síntesis. Para ello, generaremos el fichero de configuración del dispositivo, (JEDEC), haciendo doble clic sobre JEDEC, como mostramos en la figura 121:

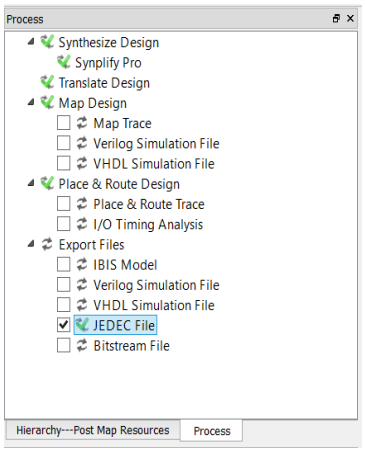


Figura 121: Generacion del JEDEC

En la ventana de consola, veremos si se ha producido algún error en la generación del JEDEC, o si se ha generado correctamente.

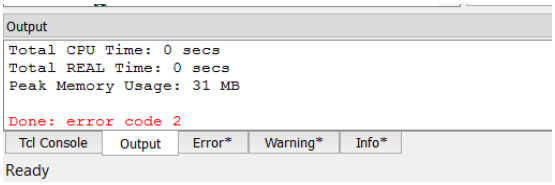


Figura 122: Informe de consola (1)

En primer lugar, realizaremos la síntesis del diseño con Lattice LSE: Con ésta primera herramienta de síntesis, no podemos generar el JEDEC. Si visualizamos la consola, figura 122, podemos ver como se ha generado un error denominado Error Code 2.

Analizaremos el report de MAP. Para ello, vamos a la pestaña Reports, figura 123, y hacemos doble clic en el informe del mapeado, que es último que se ha generado.

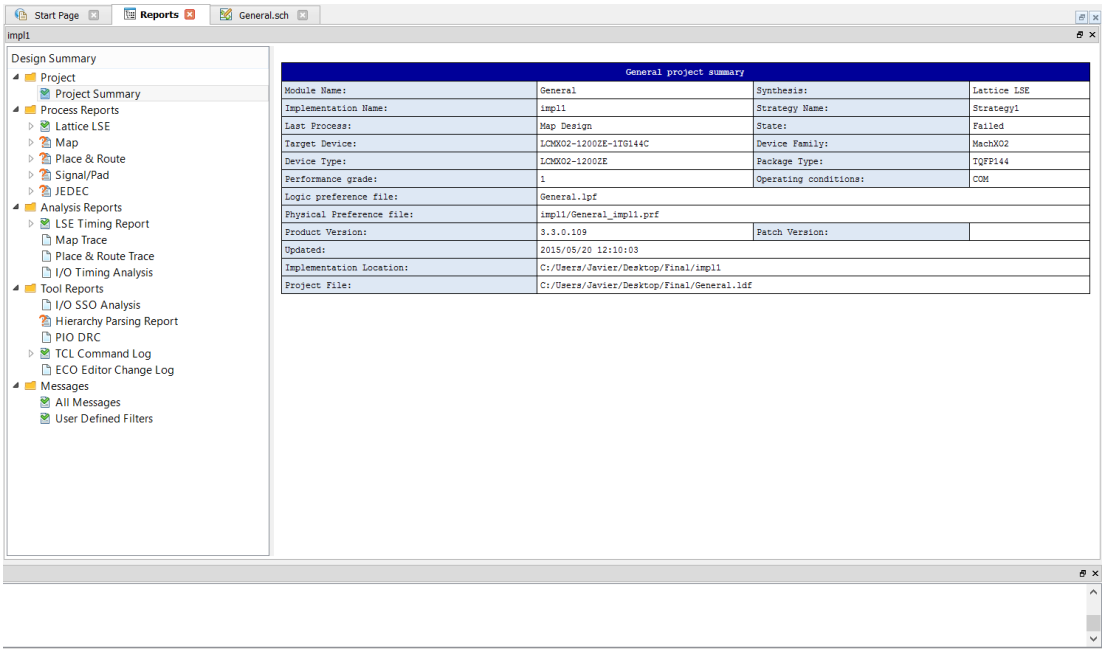


Figura 124: Pantalla de informes

```
Design Summary
   Number of registers:    114 out of  1604 (7%)
      PFU registers:           114 out of  1280 (9%)
      PIO registers:             0 out of   324 (0%)
   Number of SLICEs:       961 out of   640 (150%)
      SLICEs as Logic/ROM:    961 out of   640 (150%)
      SLICEs as RAM:            0 out of   480 (0%)
      SLICEs as Carry:        442 out of   640 (69%)
   Number of LUT4s:        1921 out of  1280 (150%)
      Number of logic LUTs:     1037
      Number of distributed RAM:   0 (0 LUT4s)
      Number of ripple logic:    442 (884 LUT4s)
      Number of shift registers:   0
   Number of PIO sites used: 24 + 4(JTAG) out of 108 (26%)
   Number of block RAMs:  0 out of 7 (0%)
   Number of GSRs:  1 out of 1 (100%)
   EFB used :       No
   JTAG used :      No
   Readback used :  No
   Oscillator used :  Yes
   Startup used :   No
   POR :            On
   Bandgap :        On
   Number of Power Controller:  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCINRD):  0 out of 4 (0%)
   Number of Dynamic Bank Controller (BCLVDSO):  0 out of 1 (0%)
```

Figura 123: Informe de mapeado con Lattice LSE

En este informe, figura 124, nos llama la atención el dato de número de LUT usados (ROJO)

Como podemos ver, usando la herramienta de síntesis Lattice SE, necesitaríamos un total de 1921LUT4, para encajar el diseño. Sin embargo, sólo disponemos de 1280, con lo cual, es efectivamente implantar el diseño en una MACHX02 1200 ZE. Deberíamos ir a una FPGA con más capacidad de lógica.

Veamos ahora que pasa si cambiamos la herramienta de síntesis, y usamos Synplify Pro. Ahora como vemos en la consola, la síntesis ha concluido con éxito, figura 125.

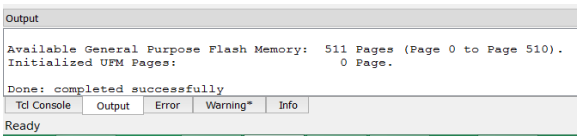


Figura 125: Informe de consola (2)

Analizando el Report de la figura 126, vemos que estamos usando un 51% de la capacidad de lógica, por lo que ahora sí podemos encajar nuestro diseño.

```
Design Summary
   Number of registers:    113 out of  1604 (7%)
      PFU registers:           94 out of  1280 (7%)
      PIO registers:           19 out of   324 (6%)
   Number of SLICEs:       139 out of   640 (22%)
      SLICEs as Logic/ROM:    139 out of   640 (22%)
      SLICEs as RAM:            0 out of   480 (0%)
      SLICEs as Carry:         62 out of   640 (10%)
   Number of LUT4s:        276 out of  1280 (22%)
      Number of logic LUTs:      152
      Number of distributed RAM:   0 (0 LUT4s)
      Number of ripple logic:     62 (124 LUT4s)
      Number of 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 GSRs:  1 out of 1 (100%)
   EFB used :       No
   JTAG used :      No
   Readback used :  No
   Oscillator used :  Yes
   Startup used :   No
   POR :            On
   Bandgap :        On
   Number of Power Controller:  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCINRD):  0 out of 4 (0%)
   Number of Dynamic Bank Controller (BCLVDSO):  0 out of 1 (0%)
```

Figura 126: Informe de mapeado con Synplify Pro

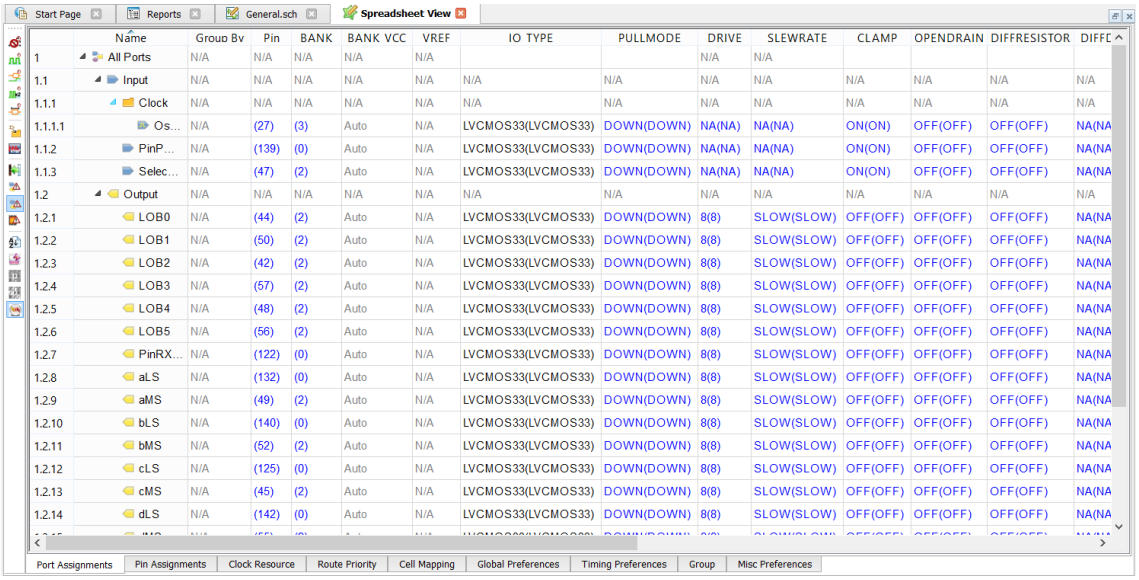
Con los resultados anteriores, hemos demostrado, que la herramienta de síntesis influye y mucho, en la optimización de un diseño, incluso hasta el punto de poder o no encajarlo en una FPGA.

3.3. Personalización de pines y niveles lógicos

Una vez que hemos generado el fichero JEDEC, es hora de seleccionar los pines concretos a los que queremos distribuir nuestras señales, tanto de entrada como de salida, así como la tecnología concreta de trabajo.

Para ello, vamos a ir al menú Spreadsheet View, pulsando sobre 

Se nos abrirá entonces el siguiente menú (figura 127):



	Name	Group By	Pin	BANK	BANK VCC	VREF	IO TYPE	PULLMODE	DRIVE	SLEWRATE	CLAMP	OPENDRAIN	DIFFRESISTOR	DIFFE
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	Os...	N/A	(27)	(3)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA
1.1.2	PinP...	N/A	(139)	(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	NA(NA)	NA(NA)	ON(ON)	OFF(OFF)	OFF(OFF)	NA(NA
1.1.3	Selec...	N/A	(47)	(2)	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	LOB0	N/A	(44)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.2	LOB1	N/A	(50)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.3	LOB2	N/A	(42)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.4	LOB3	N/A	(57)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.5	LOB4	N/A	(48)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.6	LOB5	N/A	(56)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.7	PinRX...	N/A	(122)	(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.8	aLS	N/A	(132)	(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.9	aMS	N/A	(49)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.10	bLS	N/A	(140)	(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.11	bMS	N/A	(52)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.12	cLS	N/A	(125)	(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.13	cMS	N/A	(45)	(2)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA
1.2.14	dLS	N/A	(142)	(0)	Auto	N/A	LVCMOS33(LVCMOS33)	DOWN(DOWN)	8(8)	SLOW(SLOW)	OFF(OFF)	OFF(OFF)	OFF(OFF)	NA(NA

Figura 127: Spreadsheet View (1)

Al generar el JEDEC, y realizar la síntesis de nuestro proyecto, de forma automática Diamond, asigna las señales de entrada y salida de nuestro diseño, a unos determinados pines, y a cada una de dichas señales, les asigna una tecnología.

Estas asignaciones por defecto, pueden ser adecuadas para nuestro diseño, o no serlo. En éste último caso, podemos modificarlas nosotros manualmente.

Trabajaremos en la pestaña Port Assignments, donde estableceremos las siguientes opciones:

SEÑAL	PIN	TECNOLOGIA
INPUT		
OsciladorExterno	27	LVCMOS33 (Mos de 3.3V)
PinPWSensor	1	LVCMOS33 (Mos de 3.3V)
Selección	86	LVCMOS33 (Mos de 3.3V)
		LVCMOS33 (Mos de 3.3V)
OUTPUT		
HabilitacionOsciladorExterno	32	LVCMOS33 (Mos de 3.3V)
LOB0	97	LVCMOS33 (Mos de 3.3V)
LOB1	98	LVCMOS33 (Mos de 3.3V)
LOB2	99	LVCMOS33 (Mos de 3.3V)
LOB3	100	LVCMOS33 (Mos de 3.3V)
LOB4	104	LVCMOS33 (Mos de 3.3V)
LOB5	105	LVCMOS33 (Mos de 3.3V)
PinRXSensor	3	LVCMOS33 (Mos de 3.3V)
aLS	21	LVCMOS33 (Mos de 3.3V)
aMS	23	LVCMOS33 (Mos de 3.3V)
bLS	22	LVCMOS33 (Mos de 3.3V)
bMS	25	LVCMOS33 (Mos de 3.3V)
cLS	11	LVCMOS33 (Mos de 3.3V)
cMS	33	LVCMOS33 (Mos de 3.3V)
dLS	10	LVCMOS33 (Mos de 3.3V)
dMS	34	LVCMOS33 (Mos de 3.3V)
eLS	13	LVCMOS33 (Mos de 3.3V)
eMS	35	LVCMOS33 (Mos de 3.3V)
fLS	12	LVCMOS33 (Mos de 3.3V)
fMS	26	LVCMOS33 (Mos de 3.3V)
gLS	14	LVCMOS33 (Mos de 3.3V)
gMS	24	LVCMOS33 (Mos de 3.3V)

Tabla 3: Distribución de señales, pines y niveles lógicos deseados

Cuando modifiquemos las propiedades de los pines en el menú Spreadsheet View, nos aparecerá la propiedad nueva, y la propiedad antigua entre paréntesis, como se muestra en la figura 128:

	Name	Group By	Pin	BANK	BANK VCC	VREF	IO TYPE
1	All Ports	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
1.1.1	Clock	N/A	N/A	N/A	N/A	N/A	N/A
1.1.1.1	Oscilado...	N/A	27(27)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)
1.1.2	PinPWSensor	N/A	1(14)	3(3)	Auto	N/A	LVCMOS33(LVCMOS33)
1.1.3	Seleccion	N/A	86(42)	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2	Output	N/A	N/A	N/A	N/A	N/A	N/A
1.2.1	Habilitacion...	N/A	32(1...	3(0)	Auto	N/A	LVCMOS33(LVCMOS25)
1.2.2	LOB0	N/A	97(44)	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.3	LOB1	N/A	98(49)	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.4	LOB2	N/A	99(54)	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.5	LOB3	N/A	100(...	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.6	LOB4	N/A	104(...	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.7	LOB5	N/A	105(...	1(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.8	PinRXSensor	N/A	3(65)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.9	aLS	N/A	21(59)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.10	aMS	N/A	23(56)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.11	bLS	N/A	22(45)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.12	bMS	N/A	25(67)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.13	cLS	N/A	11(48)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.14	cMS	N/A	33(62)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.15	dLS	N/A	10(55)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.16	dMS	N/A	34(58)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.17	eLS	N/A	13(84)	3(1)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.18	eMS	N/A	35(61)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.19	fLS	N/A	12(57)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.20	fMS	N/A	26(43)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.21	gLS	N/A	14(47)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)
1.2.22	gMS	N/A	24(60)	3(2)	Auto	N/A	LVCMOS33(LVCMOS33)

Figura 128: Spreadsheet View (2)

Una vez modificada la tabla anterior, guardamos los cambios con  , y regeneramos el JEDEC, como ya sabemos.

El resultado final, será nuestro fichero JEDEC de configuración, listo para ser grabado en la FPGA.

3.4. Grabado del diseño en la FPGA

Una vez sintetizado el diseño, y generado el fichero de configuración de la FPGA, fichero JEDEC, nos disponemos a grabar el diseño en nuestra FPGA.

Para ello, hacemos clic en el menú  de la barra de herramientas. Tras esto, se nos abrirá la pestaña de programación, llamada, Programmer (figura 129):

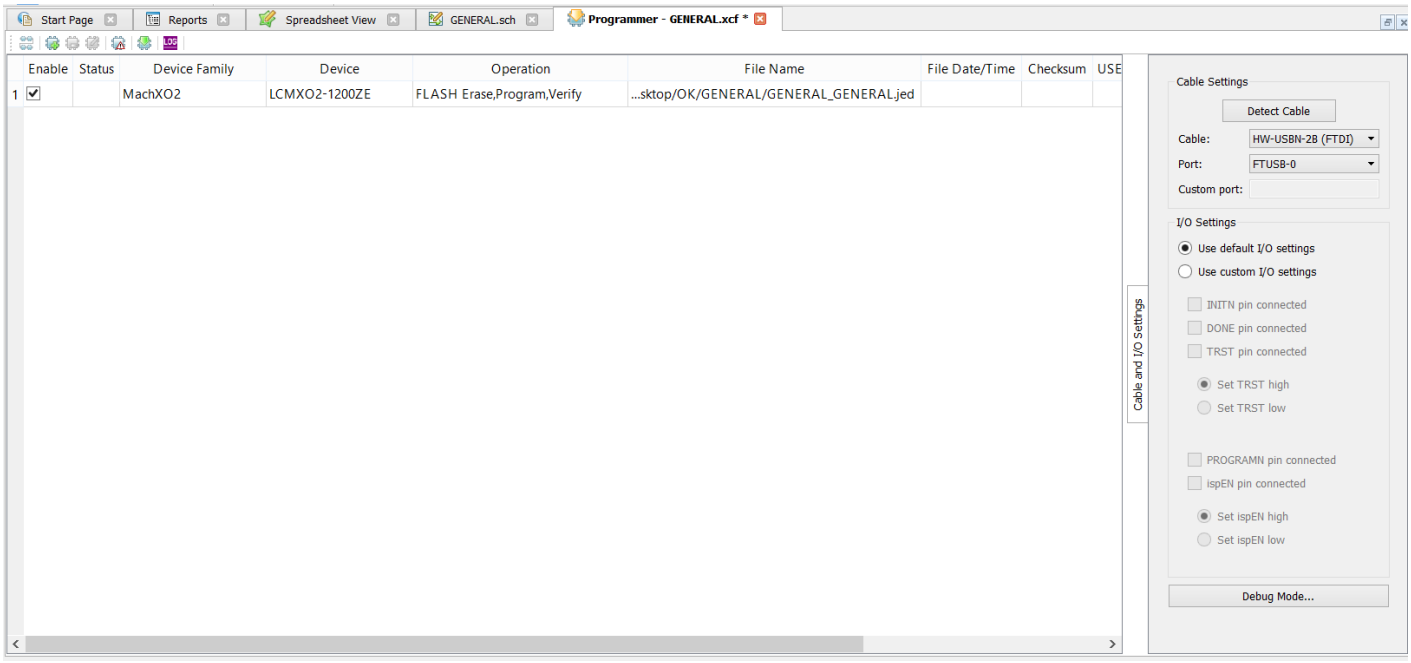


Figura 129: Programación de la FGPA (1)

Para poder grabar el JEDEC en la FPGA, hemos antes de configurar algunos parámetros de la FPGA (figura 130):

Device Family: Hacemos clic en Device family y seleccionamos nuestra familia, en este caso una MACH X02.

Device: Hacemos clic en device y seleccionamos el dispositivo concreto de la familia con la que estamos trabajando, en este caso, será una LCMX02-1200ZE.

Operation: Por defecto, se conceden los permisos de Borrado, Programación y Verificación de la memoria FLASH. Si no se permitieran estos permisos, no habría más que hacer doble clic sobre los permisos y cambiarlos por los que a continuación se muestran:

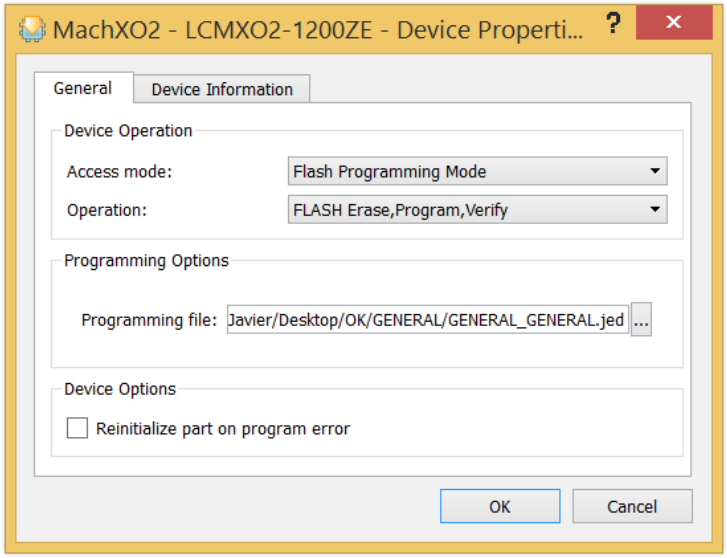


Figura 130: Programación de la FGPA (2)

File Name: En este apartado debemos de seleccionar la ubicación de nuestro fichero JEDEC. Por defecto el fichero JEDEC se encuentra ubicado en la carpeta que lleva por nombre el nombre del proyecto con el que estamos trabajando.

Tras configurar los anteriores parámetros hemos de detectar el cable de programación.
Para ello, accedemos al menú que se encuentra a la izquierda de la pantalla Programmer.
En este menú, podemos detectar automáticamente el cable haciendo clic sobre el botón Detect Cable (figura 131).

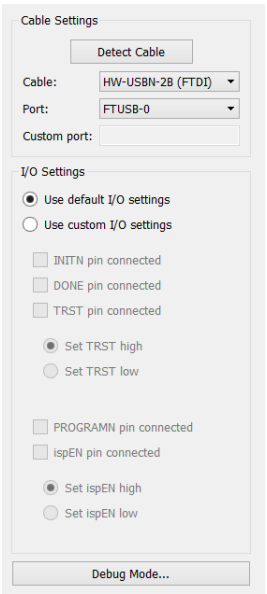


Figura 131: Detección del cable

Con todo lo anterior configurado, es hora de programar nuestra FPGA.
Para ello, hacemos clic en el botón 

3.5. Comprobación del correcto funcionamiento

Tras compilar y grabar el diseño, hemos hecho varias mediciones de distancias y velocidades.

3.5.1. Mediciones de distancia

Medición de distancia mínima:

Recordemos que la distancia mínima que puede medir el sensor son 6 pulgadas, es decir, 6 · 2.54 = 15.24 cm
El sistema deberá mostrar:

- Displays: 15
- Leds: Ninguno

El sensor, no es capaz de mostrar distancias inferiores a 15,24 cm. Como podemos ver en la figura 132, para una distancia de 10 cm, el sistema muestra 16 cm.

En la figura 132, se puede observar cómo el sistema funciona correctamente. El que muestre 15 o 16, depende de cómo aproxime el sistema.

Figura 132. Medición de la distancia mínima

Distancia intermedia

En este caso, el obstáculo se encontraba a 241 cm, como podemos ver en la cinta métrica de la figura 133.

El sistema, muestra la medida 241 cm.

El resultado se puede ver en las figuras 133, 41 cm y 134, 200 cm.

En la figura 134, podemos observar cómo se han encendido los leds D0 y D1, para mostrar los 200 cm

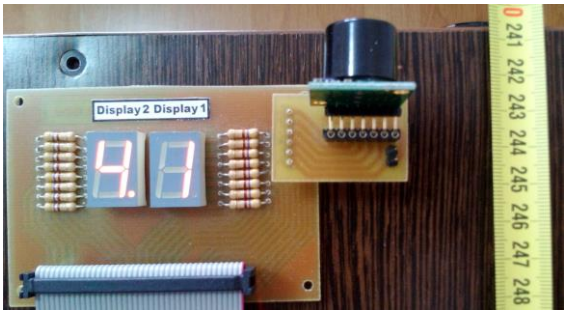


Figura 133. Medición de distancia intermedia.

Figura 134. Medición de distancia intermedia.

3.5.2. Mediciones de velocidad

Para la medición de las velocidades, hemos podido comprobar, que el sensor empleado, no es el más adecuado, debido a que su frecuencia de operación es muy baja.

Para obtener precisión en la medición de la velocidad, deberíamos haber empleado un sensor que trabajara a más frecuencia de la que trabaja el presente sensor.

En el primer caso, hemos podido comprobar el resultado de medir la velocidad en un cuerpo no móvil (una pared)
Lógicamente, como podemos ver en la figura 135, el resultado mostrado es 0 m/s.
Como podemos ver, los displays muestran 00, y los leds están apagados

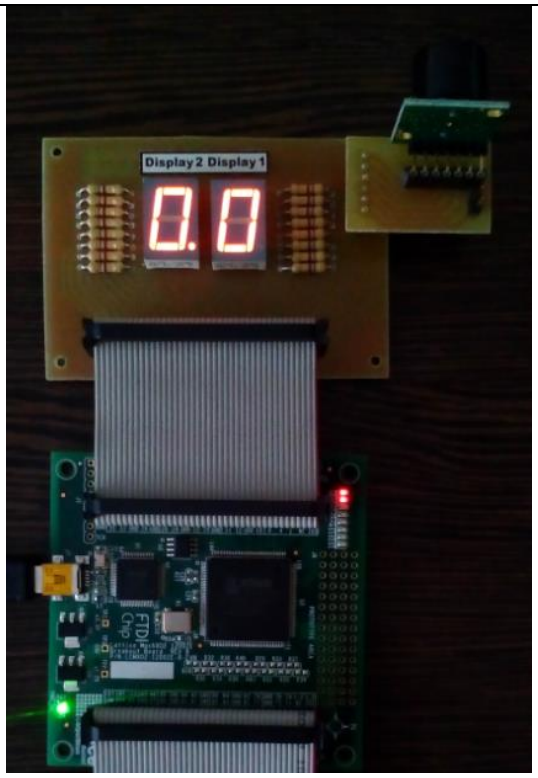


Figura 135. Velocidad de un objeto no móvil

En el segundo caso, hemos medido la velocidad de un objeto móvil.
El resultado obtenido se muestra en la figura 136.
El objeto en este caso, se mueve a una velocidad de 1 m/s, ya que los leds están apagados

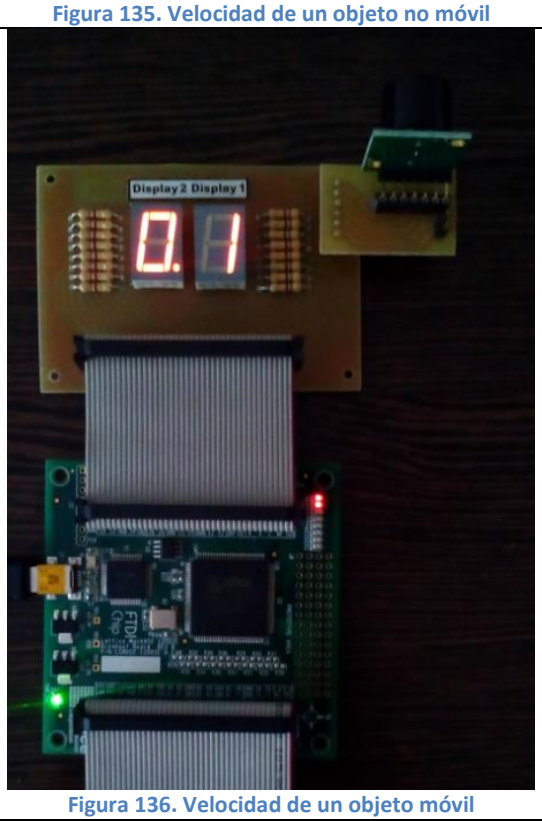


Figura 136. Velocidad de un objeto móvil

4. Recursos empleados

Tras compilar el diseño, y generar el JEDEC, o fichero binario de configuración, se nos generan una serie de informes o reports. A continuación analizaremos las partes más interesantes de algunos de ellos.

4.1. Map Report

En este informe, se nos presentan los resultados generados al encajar el diseño una vez pasado por las herramientas de síntesis en la FPGA.

En el apartado Design Information, se nos presenta la información general del diseño, como el fabricante de la FPGA donde hemos encajado el diseño o el modelo de FPGA, entre otros:

```
Design Information

Command line:   map -a MachXO2 -p LCMXO2-1200ZE -t TQFP144 -s 1 -oc
Commercial
     General_impl1.ngd -o General_impl1_map.ncd -pr General_impl1.prf -mp
     General_impl1.mrp -lpf
     C:/Users/Javier/Desktop/Esquemas/Final/impl1/General_impl1_synplify.lpf
     -lpf C:/Users/Javier/Desktop/Esquemas/Final/General.lpf -c 0 -gui
Target Vendor:  LATTICE
Target Device:  LCMXO2-1200ZETQFP144
Target Performance:   1
Mapper:  xo2c00,  version:  Diamond Version 3.3.0.109
Mapped on:  05/25/15  17:14:13
```

Informe 1: Map Report. Apartado Design Information

En el apartado Design Summary, podemos observar la ocupación que el diseño ha tenido en la FPGA. Leyendo el informe, podemos observar, que nuestro diseño ha consumido el 22% de la capacidad de lógica de la FPGA, es decir 276 de 1280 LUTS4.

Además, podemos observar el número de LUT4 que se han consumido de cada tipo:

152 de lógica.

124 en modo aritmético.

```
Design Summary
   Number of registers:    114 out of  1604 (7%)
      PFU registers:           94 out of  1280 (7%)
      PIO registers:           20 out of   324 (6%)
   Number of SLICEs:       139 out of   640 (22%)
      SLICEs as Logic/ROM:    139 out of   640 (22%)
      SLICEs as RAM:            0 out of   480 (0%)
      SLICEs as Carry:         62 out of   640 (10%)
   Number of LUT4s:        276 out of  1280 (22%)
      Number of logic LUTs:      152
      Number of distributed RAM:   0 (0 LUT4s)
      Number of ripple logic:     62 (124 LUT4s)
      Number of 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 GSRs:  1 out of 1 (100%)
   EFB used :       No
   JTAG used :      No
   Readback used :  No
   Oscillator used :  Yes
   Startup used :   No
   POR :            On
   Bandgap :        On
   Number of Power Controller:  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCINRD):  0 out of 4 (0%)
   Number of Dynamic Bank Controller (BCLVDSO):  0 out of 1 (0%)
   Number of DCCA:  0 out of 8 (0%)
   Number of DCMA:  0 out of 2 (0%)
   Number of PLLs:  0 out of 1 (0%)
   Number of DQSDLLs:  0 out of 2 (0%)
   Number of CLKDIVC:  0 out of 4 (0%)
   Number of ECLKSYNCA:  0 out of 4 (0%)
   Number of ECLKBRIDGECS:  0 out of 2 (0%)

   Number of warnings:  1
   Number of errors:    0
```

Informe 2: Map Report. Apartado Design Summary

En el siguiente apartado, IO Attributes, podemos observar, el nombre de todas las entradas y salidas de nuestro diseño, junto con el tipo de lógica usada.

Como vemos, las entradas y salidas son del tipo LVCMOS33, o MOS de 3.3 V. Además, como vemos estamos empleando tanto entradas/salidas registradas, como combinacionales.

IO (PIO) Attributes

IO Name	Direction	Levelmode IO_TYPE	IO Register
PinRXSensor	OUTPUT	LVCMOS33	
PinPWSensor	INPUT	LVCMOS33	
HabilitacionOscil…	OUTPUT	LVCMOS33	
LOB0	OUTPUT	LVCMOS33	OUT
LOB1	OUTPUT	LVCMOS33	OUT
LOB2	OUTPUT	LVCMOS33	OUT
LOB3	OUTPUT	LVCMOS33	OUT
LOB4	OUTPUT	LVCMOS33	OUT
LOB5	OUTPUT	LVCMOS33	OUT
gMS	OUTPUT	LVCMOS33	OUT
fMS	OUTPUT	LVCMOS33	OUT
eMS	OUTPUT	LVCMOS33	OUT
dMS	OUTPUT	LVCMOS33	OUT
cMS	OUTPUT	LVCMOS33	OUT
bMS	OUTPUT	LVCMOS33	OUT
aMS	OUTPUT	LVCMOS33	OUT
aLS	OUTPUT	LVCMOS33	OUT
bLS	OUTPUT	LVCMOS33	OUT
cLS	OUTPUT	LVCMOS33	OUT
dLS	OUTPUT	LVCMOS33	OUT
eLS	OUTPUT	LVCMOS33	OUT
fLS	OUTPUT	LVCMOS33	OUT
gLS	OUTPUT	LVCMOS33	OUT
OsciladorExterno	INPUT	LVCMOS33	
Seleccion	INPUT	LVCMOS33	

Informe 3: IO Atributtes

4.2 Signal/Pad

Este informe, nos permite observar las especificaciones de los PAD de la FPGA.

Podemos observar en el mismo, el tipo de FPGA, con su empaquetado:

```
PART TYPE:        LCMXO2-1200ZE
Performance Grade:      1
PACKAGE:          TQFP144
Package Status:                     Final          Version 1.39

Mon May 25 17:14:25 2015
```

Informe 4: Signal Pad. Apartado de resumen

El pinout, o la distribución de las señales en los pines. En esta ocasión la descripción de las señales es mucho más exhaustiva que en el informe anterior:

```
Pinout by Port Name:
+------------------------------+----------+--------------+-------+-----------
+-----------+----------------------------------------+
| Port Name                    | Pin/Bank | Buffer Type  | Site  | PG Enable |
BC Enable | Properties                             |
+------------------------------+----------+--------------+-------+-----------
+-----------+----------------------------------------+
| HabilitacionOsciladorExterno | 32/3     | LVCMOS33_OUT | PL10A |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| LOB0                         | 97/1     | LVCMOS33_OUT | PR4B  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| LOB1                         | 98/1     | LVCMOS33_OUT | PR4A  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| LOB2                         | 99/1     | LVCMOS33_OUT | PR3B  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| LOB3                         | 100/1    | LVCMOS33_OUT | PR3A  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| LOB4                         | 104/1    | LVCMOS33_OUT | PR2D  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| LOB5                         | 105/1    | LVCMOS33_OUT | PR2C  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| OsciladorExterno             | 27/3     | LVCMOS33_IN  | PL9A  |           |
| PULL:DOWN CLAMP:ON HYSTERESIS:SMALL  |
| PinPWSensor                  | 1/3      | LVCMOS33_IN  | PL2A  |           |
| PULL:DOWN CLAMP:ON HYSTERESIS:SMALL  |
| PinRXSensor                  | 3/3      | LVCMOS33_OUT | PL2C  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| Seleccion                    | 86/1     | LVCMOS33_IN  | PR8A  |           |
| PULL:DOWN CLAMP:ON HYSTERESIS:SMALL  |
| aLS                          | 21/3     | LVCMOS33_OUT | PL5C  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| aMS                          | 23/3     | LVCMOS33_OUT | PL8A  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| bLS                          | 22/3     | LVCMOS33_OUT | PL5D  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| bMS                          | 25/3     | LVCMOS33_OUT | PL8C  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| cLS                          | 11/3     | LVCMOS33_OUT | PL4A  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| cMS                          | 33/3     | LVCMOS33_OUT | PL10B |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| dLS                          | 10/3     | LVCMOS33_OUT | PL3D  |           |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
```

```
| dMS                            | 34/3     | LVCMOS33_OUT | PL10C |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| eLS                            | 13/3     | LVCMOS33_OUT | PL4C  |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| eMS                            | 35/3     | LVCMOS33_OUT | PL10D |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| fLS                            | 12/3     | LVCMOS33_OUT | PL4B  |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| fMS                            | 26/3     | LVCMOS33_OUT | PL8D  |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| gLS                            | 14/3     | LVCMOS33_OUT | PL4D  |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
| gMS                            | 24/3     | LVCMOS33_OUT | PL8B  |            |
| DRIVE:8mA PULL:DOWN SLEW:SLOW        |
+--------------------------------+----------+--------------+-------+------------+
```

Informe 5: Signal Pad. Apartado Pinout

Los niveles lógicos usados en cada banco. Nuestra FPGA, está dividida en 4 bancos de entradas y salidas, de los cuales, nosotros, sólo empleamos 2.

```
Vccio by Bank:
+------+-------+
| Bank | Vccio |
+------+-------+
| 1    | 3.3V  |
| 3    | 3.3V  |
+------+-------+
```

Informe 6: Signal Pad. Apartado VccIO

5. Conclusiones y líneas futuras de desarrollo:

Al término del presente trabajo de fin de grado, podemos afirmar que se han alcanzado y logrado los objetivos que nos marcamos al inicio del mismo.

Como ya especificamos, y hemos mostrado, se ha logrado el objetivo principal de diseñar un sistema funcional de medición de distancias y velocidades, empleando un sistema electrónico reconfigurable tipo FPGA, junto con un sensor de ultrasonidos.

Además, se han logrado los objetivos parciales marcados.

- Se han evaluado las diferentes alternativas de implantación lógica disponibles en el mercado, justificando por qué elegimos un sistema tipo FPGA.

- Se han evaluado los diferentes sensores de medición de distancias sin contacto disponibles en el mercado, justificando por qué elegimos los ultrasonidos.

- Se ha utilizado herramientas de diseño modernas, basadas en una combinación de lenguajes de descripción de hardware (VHDL), y el empleo de esquemáticos.

- Se ha utilizado diferentes herramientas de síntesis, y analizado cómo afectan a la cantidad de recursos que se consumen al encajar un diseño en el sistema electrónico reconfigurable concreto.

- Se ha utilizado herramientas de simulación de circuitos.

Como líneas futuras para el desarrollo y crecimiento del presente TFG, propongo profundizar en los siguientes puntos:

1. Profundizar en algoritmos que permitan filtrar los falsos ecos que recibe el sensor.
   En las pruebas que he realizado con el mismo, se aprecia como en lugares pequeños, y sobrecargados de obstáculos, se generan medidas erróneas de distancia, lo cual viene motivado por la existencia de falsos ecos.

2. Emplear el Wishbone Bus, para utilizar las mediciones de distancia con otros objetivos, como por ejemplo sintetizar relojes de diferente frecuencia en función de la distancia mediante el PLL, con el objetivo de emplearlo en aplicaciones como parktronic, alarmas, etc.

3. Analizar el efecto de emplear un emisor/receptor de ultrasonidos separado, en lugar del conjunto que tenemos en la actualidad.

4. Analizar el efecto de emplear un emisor/receptor que pueda trabajar a mayor frecuencia. Actualmente, nuestro sensor permite realizar mediciones a una frecuencia máxima de 20 Hz. Esta frecuencia, es suficiente para la medición de distancias, pero claramente insuficiente para la medición de velocidades.

6. Bibliografía

- ✓ González Antón, J. *Apuntes de Sistemas Electrónicos Reconfigurables.* (Curso académico 2013 – 2014). Profesor encargado de la docencia: D. Francisco José De Andrés Rodríguez Trelles. 4º de Grado en electrónica industrial y automática. Universidad de Valladolid.

- ✓ Lattice Semiconductor Corp. (2014). *MachX02 Family DataSheet V 2.6.* Portland, Oregon.

- ✓ MaxBotix Inc. (2014). LV-MaxSonar-EZ3 DataSheet. Shawkia Drive, Brainerd.

- ✓ Pallás Areny R., Valdés Pérez F. E. (2007). *Microcontroladores. Fundamentos y aplicaciones con PIC*. Primera Edición. Barcelona, España: Ed. Marcombo.

- ✓ Angulo Usategui, J. M., Romero Yesa, S., Angulo Martínez I. (2006). *Microcontroladores PIC. Diseño práctico de aplicaciones*. Segunda Edición. Madrid, Espala: Ed. McGraw Hill.

- ✓ Alfonso Pérez S., Soto Campos E., Fernández Gómez S. (2002). *Diseño de sistemas digitales con VHDL.* Madrid: Ed.Thomson.

- ✓ Chan, K. C. (1997). *Digital design and modeling with VHDL and Synthesis.* Ed. Wiley-IEEE Computer Society Press.

- ✓ Vega Fidalgo Luis Miguel, Zorita Téllez David. *Universidad de Valladolid. E.T.S. de Ingenieros de Telecomunicación. Proyecto de ingeniería de las ondas I*. <http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_03_04/infra_y_ultra/aplicaciones_ultrasonidos.htm>. [Consulta: 13 de junio de 2015].

- ✓ Xilinx. *Article#233098.2i XST: "ERROR:HDLParsers:808; C:/.../des.vhd; Line 599. TO_INTEGER can not have such operands in this context."*. <http://www.xilinx.com/support/answers/23309.html>. [Consulta: 13 de junio de 2015].

- ✓ Lewis Jim. *VHDL Math tricks of the trade.* <http://www.synthworks.com/papers/vhdl_math_tricks_mapld_2003.pdf>. [Consulta: 13 de junio de 2015].

- ✓ Pérez De Diego, Diego. *Seminario acerca de sensores de distancia por ultrasonidos.* <http://www.alcabot.com/alcabot/seminario2006/Trabajos/DiegoPerezDeDiego.pdf>. [Consulta a 13 de junio de 2015.].

Anexos

A1. Redimensionamiento de la hoja del editor de esquemas

Diamond, permite trabajar con hojas normalizadas de distintos tamaños, a la hora de realizar un esquema.

Para redimensionar la hoja, hemos de estar trabajando en un fichero *.sch, y hacer clic en Edit, Sheet…

Una vez realizadas dichas operaciones se abrirá un cuadro de diálogo como el mostrado en la figura 132:

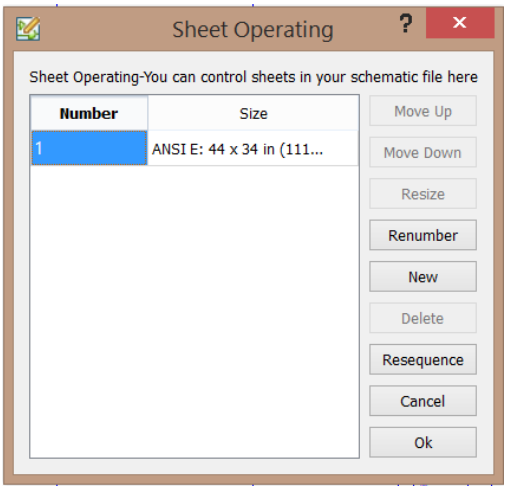


Figura 137: Redimensionamiento del espacio de trabajo (1)

En este cuadro, podemos añadir nuevas hojas a nuestro proyecto, renumerarlas, borrarlas… y lo que nos interesa, redimensionarlas. Para redimensionarlas, haremos clic sobre el cuadro de dimensiones de la hoja deseada, y después, sobre el botón RESIZE. Se nos abrirá la siguiente ventana (figura 133):

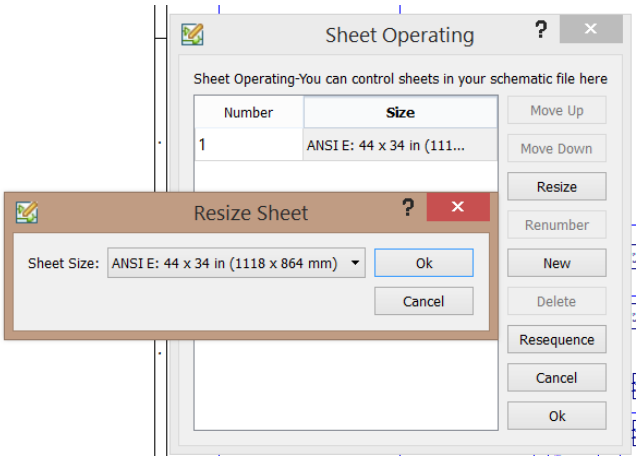


Figura 138: Redimensionamiento del espacio de trabajo (2)

En ella, podemos elegir entre diferentes formatos de papel estandarizados, desde ANSI A (el más pequeño), hasta ANSI E, (el mayor).

A2. Distribución de pines en un display 7 segmentos

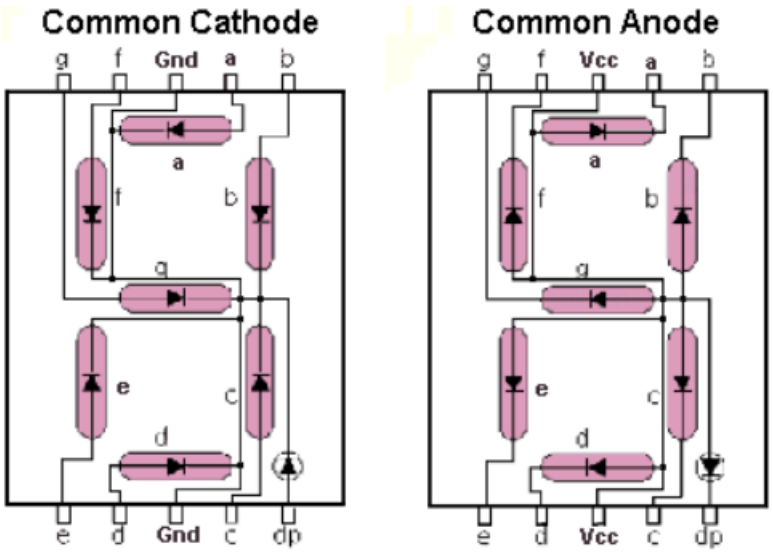
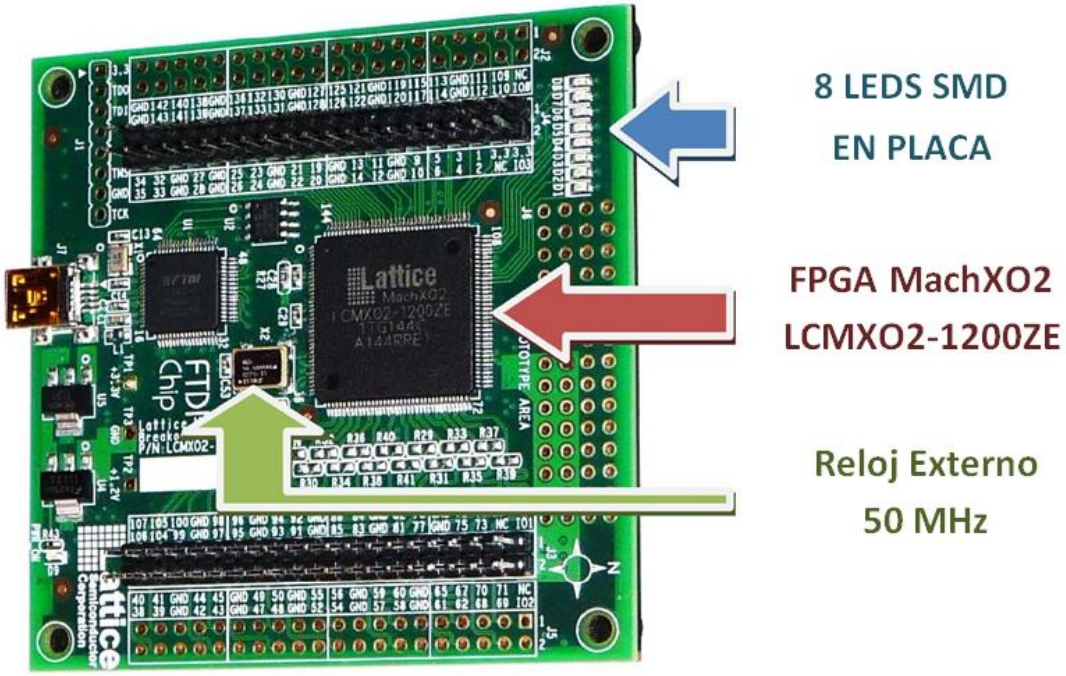


Figura 139: Displays de 7 segmentos en ánodo y cátodo común

A3. Módulo de prácticas de sistemas electrónicos reconfigurables

A3.1. Distribución de pines: PLACA MachXO2 BREAKOUT BOARD



Distribución de pines: Reloj externo:

CLK_OUT => PIN 27.

CLK_ENABLE => PIN 32.

Distribución de pines: Leds On Board

D1 => PIN 97.

D2 => PIN 98.

D3 => PIN 99.

D4 => PIN 100.

D5 => PIN 104.

D6 => PIN 105.

D7 => PIN 106.

D8 => PIN 108.

A3.2. Distribución de pines: PLACA PCB SWITCH & PULSADORES 2

Distribución de pines: Switches.

SW1 => PIN 86.

SW2 => PIN 85.

SW3 => PIN 84.

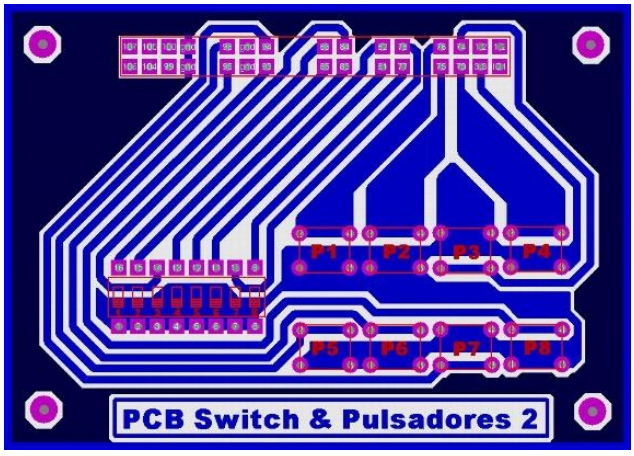
SW4 => PIN 83.

SW5 => PIN 82.

SW6 => PIN 81.

SW7 => PIN 78.

SW8 => PIN 77.



Distribución de pines: Pulsadores.

PULSADOR 1 => PIN 76.

PULSADOR 2 => PIN 75.

PULSADOR 3 => PIN 74.

PULSADOR 4 => PIN 73.

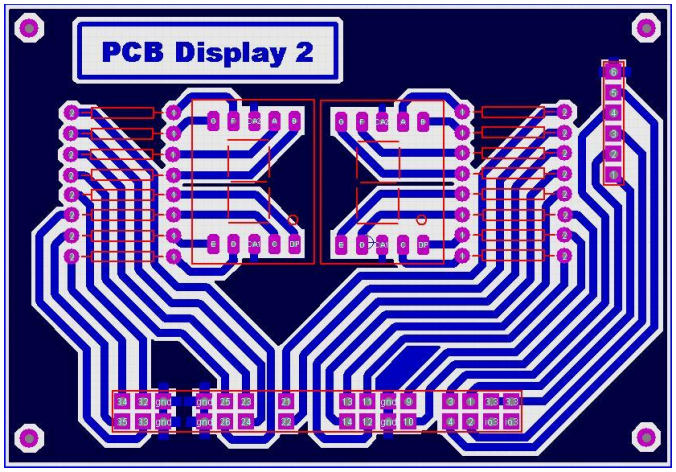
PULSADOR 5 => PIN 96.

PULSADOR 6 => PIN 95.

PULSADOR 7 => PIN 94.

PULSADOR 8 => PIN 93.

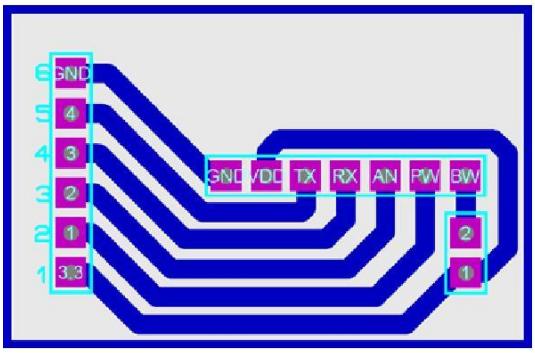
A3.3. Distribución de pines: PLACA PCB DISPLAY 2

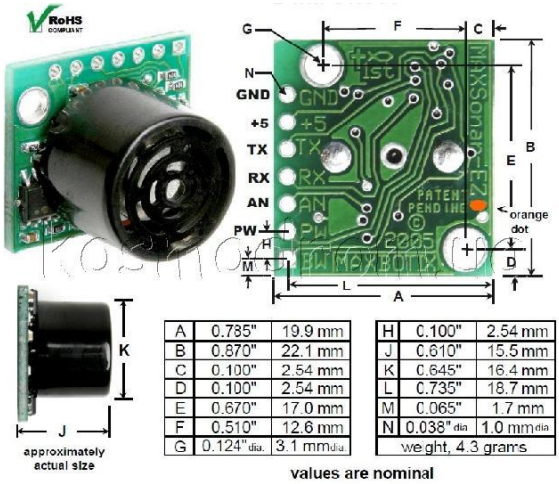


Segmento	PINES	
	Display 2 (Izquierdo)	**Display 1 (Derecho)**
A	23	21
B	25	22
C	33	11
D	34	10
E	35	13
F	26	12
G	24	14
DP	32	9

Tabla 5: Distribución de segmentos en los pines de la FPGA

A3.4. Distribución de pines: SENSOR LV-MAXSONAR EZ3





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		

RX => PIN 3 DE LA FPGA.

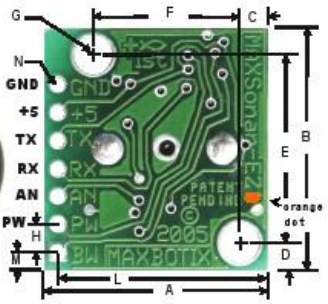
PW => PIN 1 DE LA FPGA.

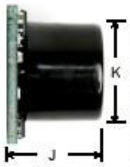
A4. Datasheet del sensor de ultrasonidos

MB1030

LV-MaxSonar®-EZ3™ High Performance Sonar Range Finder

With 2.5V - 5.5V power, the LV-MaxSonar®-EZ3™ provides very short to long-range detection and ranging, in an incredibly small package. The LV-MaxSonar®-EZ3™ 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 range as 6-inches. The interface output formats included are pulse width output, analog voltage output, and serial digital output.





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

Features

- Continuously variable gain for beam control and side lobe suppression
- Object detection includes 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
- All interfaces are active simultaneously
- Serial, 0 to Vcc, 9600Baud, 8N1
- 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)

Benefits

- Very low cost sonar ranger
- Reliable and stable range data
- Sensor dead zone virtually gone
- Lowest power ranger
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery based systems
- Can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the three sensor outputs

Beam Characteristics

Many applications require a narrower beam or lower sensitivity than the LV-MaxSonar®-EZ1™. MaxBotix® Inc., is offering, the EZ2™, EZ3™, & EZ4™ with progressively narrower beam angles allowing the sensor to match the application. Sample results for the LV-MaxSonar®-EZ3™ measured beam patterns are shown below on a 12-inch grid. The detection pattern is shown for;

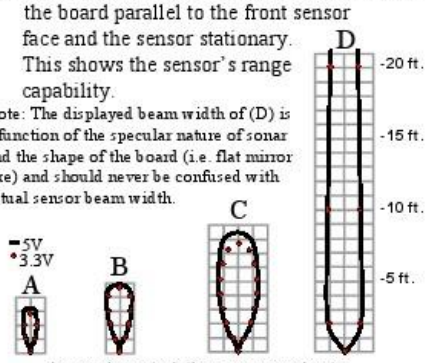
(A) 0.25-inch diameter dowel, note the narrow beam for close small objects,

(B) 1-inch diameter dowel, note the long narrow detection pattern,

(C) 3.25-inch diameter rod, note the long controlled detection pattern,

(D) 11-inch wide board moved left to right with the board parallel to the front sensor face and the sensor stationary. This shows the sensor's range capability.

Note: The displayed beam width of (D) is a function of the specular nature of sonar and the shape of the board (i.e. flat mirror like) and should never be confused with actual sensor beam width.



beam characteristics are approximate

MaxBotix® Inc.

MaxBotix, MaxSonar, EZ2, EZ3 & EZ4 are trademarks of MaxBotix Inc.

LV-EZ3™ • Patent 7,679,996 • Copyright 2005 - 2012

Page 1

Email: info@maxbotix.com

Web: www.maxbotix.com

PD10007e

LV-MaxSonar®-EZ3™ Pin Out

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

+5V – Vcc – Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V.

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).

RX – This pin is internally pulled high. The EZ3™ will continually measure range and output if RX data is left unconnected or held high. If held low the EZ3™ will stop ranging. Bring high for 20uS or more to command a range reading.

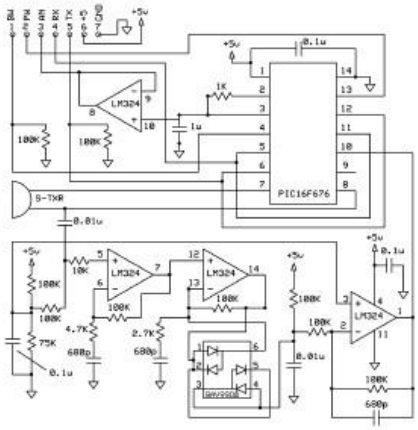
AN – Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.

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

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.

LV-MaxSonar®-EZ3™ Circuit

The LV-MaxSonar®-EZ3™ sensor functions using active components consisting of an LM324, a diode array, a PIC16F676, together with a variety of passive components.



LV-MaxSonar®-EZ3™ Timing Description

250mS after power-up, the LV-MaxSonar®-EZ3™ 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®-EZ3™ 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®-EZ3™ sends thirteen 42KHz waves, 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. The LV-MaxSonar®-EZ3™ timing is factory calibrated to one percent at five volts, and in use is better than two percent. In addition, operation at 3.3V typically causes the objects range, to be reported, one to two percent further than actual.

LV-MaxSonar®-EZ3™ General Power-Up Instruction

Each time after the LV-MaxSonar®-EZ3™ 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 it 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 then ignore objects at that distance.

The LV-MaxSonar®-EZ3™ 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®-EZ3™, cycle power, then command a read cycle.