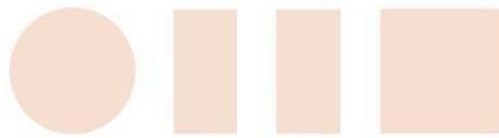
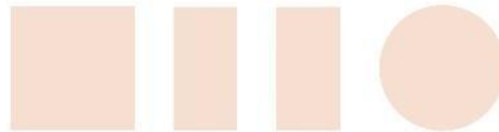
Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES



Trabajo Fin de Grado

ESTUDIO E IMPLEMENTACIÓN DE LA HERRAMIENTA CHIPSCOPE DE XILINX EN LA DEPURACIÓN DE FPGAs



ESCUELA POLITECNICA SUPERIOR

Autor: Anabel Almodóvar Hernández

Tutor/es: Ignacio Fernández Lorenzo

2018

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

Trabajo Fin de Grado

ESTUDIO E IMPLEMENTACIÓN DE LA HERRAMIENTA CHIPSCOPE DE XILINX EN LA DEPURACIÓN DE FPGAS.

Autor: Anabel Almodóvar Hernández
Tutor/es: Ignacio Fernández Lorenzo

TRIBUNAL:

Presidente: Mª DEL CARMEN PÉREZ RUBIO.

Vocal 1º: IGNACIO FERNÁNDEZ LORENZO.

Vocal 2º: IGNACIO BRAVO MUÑOZ.

FECHA:

“Dime y lo olvido, enséñame y lo recuerdo, involúcrame y lo aprendo”

Benjamín Franklin

Agradecimientos

Gracias a mi tutor Ignacio Fernández y a mi tutor Erasmus Martyn Rafcliffe, quienes me han dado su apoyo y ayuda a la hora de realizar este proyecto. Durante este proceso he podido descubrir esta parte de la electrónica dedicada al diseño que tanto me ha fascinado y cautivado. Me gustaría agradecer también a mi familia quienes han estado a mi lado en todo momento y a las grandes personas que he encontrado en la universidad y que ahora forman parte de mi vida.

Resumen

En el presente Trabajo Fin de Grado se va a estudiar la herramienta ChipScope Pro que proporciona el entorno ISE Design de Xilinx. Esta herramienta proporciona la posibilidad de depurar diseños implementados en FPGAs de forma directa y sin emplear herramientas adicionales. ChipScope Pro contiene tres módulos principales para la depuración: ICON, ILA y VIO; los cuales permiten al usuario visualizar señales internas que intervienen en el diseño y tienen la capacidad de simular entradas y salidas de los distintos módulos pertenecientes al diseño. A través de distintos ejemplos se irán analizando cada uno de los posibles usos que puede ofrecer ChipScope Pro. Al final del documento se hará una valoración de la herramienta y la utilidad de esta en referencia al diseño y depuración de sistemas a través de FPGAs.

Palabras clave

ChipScope, Xilinx, FPGAs, ILA, VIO

Abstract

In the following document we will study the ChipScope Pro tool that provides the Xilinx ISE Design environment. This tool provides the possibility to debug designs implemented in FPGAs directly and without using additional tools. ChipScope Pro contains three main modules for debugging: ICON, ILA and VIO; which allow the user to visualize internal signals that intervene in the design and have the ability to simulate inputs and outputs of the different modules belonging to the design. Through different examples, each of the possible uses that ChipScope Pro can offer will be analyzed. At the end of the document there will be an evaluation of the tool and the utility of this in reference to the design and debugging of systems through FPGAs.

Keywords

ChipScope, Xilinx, FPGAs, ILA, VIO

Resumen extendido:

El uso de chips FPGA en la industria va en aumento ya que incorpora lo mejor de los ASICs y de los sistemas basados en procesadores. Al ser chips de silicio reprogramable permiten el diseño personalizado sin los gastos que conlleva toda la electrónica correspondiente.

Las FPGAs permiten una simulación de las señales empleadas en el diseño gracias al uso de módulos testbench. Sin embargo, solo es posible observar las señales de entrada y salida en tiempo real, lo que dificulta depurar el diseño en busca de errores. Gracias a la herramienta ChipScope Pro que incorpora Xilinx es posible visualizar las señales internas, que de otra manera solo sería posible empleando métodos más complejos y costosos.

La herramienta ChipScope Pro genera una serie de módulos que, agregados al diseño, nos permite crear una conexión con el PC de modo que el usuario pueda monitorizar y depurar las señales internas del diseño en tiempo real, lo que facilita al usuario la tarea de encontrar los posibles fallos en este. Estos módulos se pueden incorporar, sintetizar e implementar como cualquier otro código, lo que es una gran ventaja para el diseñador.

El siguiente documento contendrá el estudio de la herramienta ChipScope Pro de Xilinx y la demostración de su funcionamiento a través de la implementación de varios diseños en un dispositivo FPGA.

Los módulos a analizar serán principalmente los módulos ILA y VIO. Estos módulos se implementan gracias a la lógica interna de la FPGA, llegando así a las señales internas del diseño. El responsable de realizar la conexión entre el analizador lógico integrado en la FPGA y el PC es el núcleo ICON. Para ello hace uso de las cadenas Boundary Scan del puerto JTAG del dispositivo, que actúa como interfaz de comunicación entre ambos lados.

El núcleo ILA implementa un analizador lógico adaptable que permite la captura de las señales internas que intervienen en el diseño. Incorpora una serie de puertos trigger de entrada y salida que, conectados a las señales del diseño, se encargan de indicar el inicio de captura de las muestras, las cuales se visualizarán posteriormente en el PC.

El puerto trigger de salida puede ser usado como una señal de disparo externa ligada a un pin de salida para probar equipos externos. También puede ser usado en la lógica interna como una interrupción, una señal de disparo o para la conexión de varios núcleos ILA en cascada.

La información capturada por el núcleo ILA es almacenada on-chip, por lo que consume recursos de la placa. Este núcleo incluye la opción 'storage qualification condition' que permite optimizar el diseño y evitar así un excesivo uso de recursos en el proceso de depuración.

A través del núcleo VIO se puede llevar a cabo el acceso a los puertos de salida y entrada del diseño que con ILA no es posible. Es capaz de simular puertos de entrada y salida como LEDs o switches virtuales de modo que sea posible interactuar con las señales internas del diseño durante la depuración de este. Incluye además detector de actividades en los puertos síncronos de entrada capaces de chequear si ha ocurrido una transición entre el tiempo

transcurrido entre muestras. VIO también ofrece la posibilidad de definir un tren de pulsos de 16 ciclos de reloj en sus puertos de salida síncronos.

Tras el desarrollo de varios ejemplos que demuestran la funcionalidad de ChipScope Pro se ha llegado a la conclusión de que es una herramienta sencilla y útil, con numerosas ventajas a la hora de diseñar un sistema. La importancia de la depuración de diseños hace que ChipScope sea una herramienta fundamental en el entorno ISE Design Suite de Xilinx.

Tabla de contenidos.

Resumen extendido 9

Tabla de contenidos. 11

Tabla de figuras. 15

Acrónimos 19

CAPITULO 1: 21

Introducción a las FPGAs 21

1.1. Introducción: 21

1.2. Arquitectura de las FPGAs de Xilinx: 22

1.3. Ventajas y desventajas: 24

1.4. Comprobación de un diseño: 24

1.5. ChipScope Pro tool: 26

CAPITULO 2: 29

ChipScope Pro Tool 29

2.1. ICON 30

2.2. ILA 33

Ejemplo 34

2.3. VIO 47

Ejemplo 48

2.4. ATC2 55

CAPITULO 3: 57

Core Inserter 57

CAPITULO 4: 65

Sistema a depurar – Semáforo 65

CAPITULO 5: 71

Sistema a depurar – Reproductor de partituras 71

5.1. Epp_controller 73

5.1.1. Diseño 75

5.2. Decoder_epp 76

5.2.1. Diseño 77

5.3. Gen_frec_code 78

5.3.1. Diseño 79

5.4. Códec_controler 80

5.4.1. Diseño 81
CAPITULO 6: 87
Implementación y depuración. 87
6.1. Controlador del puerto EPP 87
6.2. Decoder_epp 91
6.3. Gen_frec_code 93
6.4. Codec_controller 98
Rst_and_sync_generation 100
Output_frame 102
Capitulo 7: 105
Evaluación de las funciones de ChipScope 105
Medida de tiempos: 105
Triggering: 105
Optimización: 106
Output trigger: 106
Bus plot: 107
Tren de pulsos: 107
Detector de actividades: 107
Capitulo 8: 109
Conclusiones 109
Capitulo 9: 111
Pliego de condiciones 111
Capitulo 10: 115
Presupuesto 115
ANEXOS 119
ANEXO I: Códec de audio LM4550 121
BIT_CLK 121
Reset: 121
SYNC 122
Trama 122
Registros 124
ANEXO II: Código 129
- Ejemplo ILA: Contador 129
- Ejemplo VIO: ALU 131

- Ejemplo Core Inserter: Registro 133
- Semáforo 134
  Main file: 134
  Prescaler 136
  Contador 137
  Edge_detector 139
- Epp_controller 140
- DECODER_EPP 143
  Main 143
  decoder_epp 144
- FIFO 146
- Codec_controller 148
- Rst_and_sync_generation 151
- Output_frame 153

Bibliografía: 159

Tabla de figuras.

Figura 1: estructura interna de logicas programables 22
Figura 2: arquitectura básica de una FPGA 23
Figura 3: arquitectura interna de una FPGA de Xilinx 23
Figura 4: Diagrama de bloques de de un sistema ATE 25
Figura 5: especificaciones de los módulo ILA y VIO 28
Figura 7: módulo ICON con BSCAN interno vs BSCAN externo 30
Figura 8: opciones en la generación de módulos compatibles con la versión ISE utilizada 31
Figura 9: generación del módulo ICON 32
Figura 10: fichero ICON.vho 32
Figura 11: conexión del módulo ILA 33
Figura 12: contador 16 bits a diseñar 34
Figura 13: señales conectadas al módulo ILA 35
Figura 14: configuración de los puertos del módulo ILA 36
Figura 15: configuración del trigger del módulo ILA 36
Figura 16: configuración de las opciones de match del trigger 37
Figura 17: fichero .vho generado en la creacion del módulo ILA 37
Figura 18: conexionado de las señales a observar con el núcleo ILA 38
Figura 20: icono utilizado para añadir el proyecto a programar en la placa 39
Figura 22: pantalla inicial de la herramienta ChipScope Pro 39
Figura 23: configuración del dispositivo a analizar 40
Figura 24: señales del analizador lógico 40
Figura 25: adaptación de las señales a visualizar al proyecto con el que se está trabajando 41
Figura 26: configuración del trigger 41
Figura 27: configuración de la condición del trigger a aplicar 42
Figura 28: configuración de la opción ‘storage condition’ a aplicar 43
Figura 29: opciones a la hora de ejecutar el trigger 44
Figura 30: configuración del modo Startup del trigger 45
Figura 31: pantalla resultante tras la configuración de los parámetros del analizador lógico ... 45
Figura 32: resultado del analizador lógico 46
Figura 33: funcionamiento del diseño tras aplicar un zoom 46
Figura 34: ventana Bus Plot del analizador lógico 46
Figura 35: ventana Listing del analizador lógico 47
Figura 36: conexión del módulo VIO 48
Figura 37: diagrama de bloques del diseño a implementar 48
Figura 38: diagrama de bloques resultante al insertar ChipScope Pro 49
Figura 39: generación del módulo ICON para dos núcleos 49
Figura 40: configuración del puerto de datos del núcleo ILA 50
Figura 41: configuración de los puertos trigger del núcleo ILA 50
Figura 42: configuración del núcleo VIO 51
Figura 43: archivo .vho del núcleo VIO generado por la herramienta IPCore 51
Figura 44: código de instanciación de ChipScope 52
Figura 45: conexión de las señales que van a intervenir en la depuración con ChipScope 52

Figura 46: ventana del analizador lógico de ILA 53
Figura 47: ventana consola VIO sin configurar 53
Figura 48: opciones de configuración de las señales del módulo VIO 54
Figura 49: opciones ofrecidas por el núcleo VIO 54
Figura 50: resultado de la depuración del módulo ALU 55
Figura 51: diagrama de bloques del núcleo ATC2 55
Figura 52: comunicación con el núcleo ICON. 56
Figura 53: selección de Core Inserter para el uso de la herramienta ChipScope 58
Figura 54: fichero de Core Inserter de la herramienta ChipScope 58
Figura 55: diagrama de bloques del registro 59
Figura 56: configuración del núcleo ICON 59
Figura 57: configuración de los triggers del núcleo ILA 60
Figura 58: configuración del bus de datos del núcleo ILA 60
Figura 59: configuración ILA 61
Figura 60: selección de las señales a conectar en el núcleo ILA 62
Figura 61: configuración del puerto trigger 62
Figura 62: resultado del módulo en el analizador del módulo ILA 63
Figura 64: a) máquina del sistema a diseñar. b) Secuencia a seguir por el semáforo. 66
Figura 65: simulación del diseño 66
Figura 66: señales internas del diseño capturadas con el analizador lógico de ChipScope 67
Figura 67: sistema a depurar con ChipScope 67
Figura 68: resultado del test del módulo prescaler 68
Figura 69: nuevo código a analizar 68
Figura 70: módulo prescaler con rst=0 68
Figura 71: sistema tras el cambio en la señal reset 69
Figura 72: contador tras el reset. 69
Figura 73: Contador en las pulsaciones consecutivas 69
Figura 74: cambio del codigo en el módulo contador 70
Figura 75: resultado deseado del contador 70
Figura 76: funcionamiento final del sistema 70
Figura 77: direcciones y datos a utilizar por la aplicación 72
Figura 78: estructura del diseño 72
Figura 79: señales que intervienen en el protocolo EPP 73
Figura 80: entidad epp_controller 74
Figura 81: ciclo de escritura 74
Figura 82: entidad creada para la depuración 75
Figura 83: detección de flancos en una señal registrada 76
Figura 84: entidad decoder_epp 77
Figura 85: cronograma de la entidad decoder_epp 78
Figura 86: entidad gen_frec_code 78
Figura 87: entidad fifo 79
Figura 88: entidad codec_controler 80
Figura 89: interfaz del códec con la FPGA 81
Figura 90: nueva entidad códec_controller 82
Figura 91: diagrama de tiempos de la señal de reset 83

Figura 92: cronograma de generación de la señal SYNC 83
Figura 94: state chart de la trama 84
Figura 95: entidad output_frame 85
Figura 96: diagrama de bloques usado en la depuración 88
Figura 97: conexionado de los núcleos de ChipScope en el diseño 88
Figura 98: Configuración de la señal de disparo 89
Figura 99: escritura de datos 89
Figura 100: lectura de datos 89
Figura 101: diagrama de bloques del diseño modificado 90
Figura 102: Ciclo de escritura 90
Figura 105: cronograma de tiempos reales del ciclo de lectura 91
Figura 106: diagrama de bloques de la entidad decoder_epp tras insertar ChipScope 92
Figura 107: conexion de las señales de ChipScope 92
Figura 108: configuracion trigger 92
Figura 109: comportamineto del módulo decoder_epp 93
Figura 110: diagrama de bloques del diseño insertando la herramienta ChipScope 94
Figura 111: configuración del núcleo VIO 94
Figura 112: configuración del núcleo ILA 95
Figura 113: conexión de las señales de ChipScope con el diseño 95
Figura 114: consola del módulo VIO 96
Figura 115: consola VIO configurando din como tren de pulsos 96
Figura 116: configuración de los valores del tren de pulsos 96
Figura 117: configuración puerto de entrada del núcleo VIO 97
Figura 118: funcionamiento de los puertos de salida del núcleo VIO 97
Figura 119: configuración del trigger 97
Figura 120: lectura del dato escrito en la memoria FIFO 98
Figura 121: diagramas de bloque del diseño a depurar 98
Figura 122: configuración de los puertos trigger del núcleo ILA principal 99
Figura 123: consola VIO del módulo principal 99
Figura 124: depuracion entidad codec_controller 99
Figura 125: diagrama de bloques del conexionado de núcleos ILA 100
Figura 127: Rst_and_sync_generation 101
Figura 128: Codec_controller 101
Figura 129: diagrama de bloques del diseño a depurar 102
Figura 130: Output_frame 103
Figura 131: muestras de audio pertenecientes al SLOT3 103
Figura 132: muestras de audio en formato decimal 104
Figura 133: detector de actividad de la señal r_edge_b_clk 104
Figura 134: señal bit_clk del códec 121
Figura 135: señal de “cold reset” del códec 121
Figura 136: señal de “warm reset” del códec 122
Figura 137: señal SYNC del códec 122
Figura 138: trama a enviar para la configuración del códec de audio LM4550 123
Figura 139: diagrama de bloques del códec de audio LM4550 125
Figura 140: mapa de registros del códec de audio LM4550 125

Acrónimos

FPGA – Field Programmable Gate Array
PLD – Programmable Logic Device
PLA – Programmable Logic Array
PAL – Programmable Array Logic
GAL – Generic Array Logic
DSP – Digital Signal Processor
ASIC – Application-Specific Integrated Circuit
LUT – Look Up Table
MAC – Multiply-and-Accumulate
ATE – Automatic Test Equipment
ALU – Arithmetic Logic Unit
ICON – Integrated Logic Analyzer
ILA – Integrated Controller
VIO – Virtual Input/Output
ATC2 – Agilent Trace Core 2
IBA – Integrated Bus Analyzer

CAPITULO 1:

Introducción a las FPGAs

1.1. Introducción:

Las FPGAs surgen como evolución de los PLDs (dispositivos de lógica programable) los cuales fueron aquellos cuya funcionalidad era definida por el usuario una vez fabricado. Estos dispositivos eran capaces de implementar lógica combinacional, realizando cualquier función lógica mediante la suma de términos productos (AND-OR). Para conseguir una mejora de estos dispositivos se desarrollaron los arrays de lógica programable (PLA), más rápidos y capaces de ofrecer lógica secuencial, compuestos por una matriz de puertas AND y OR los cuales son configurables. Debido a la complejidad de esta arquitectura se optó por dejar una de las matrices programable mientras la otra permanece fija, de esta manera se evolucionó a los dispositivos PAL. Estos mantienen su matriz AND programable mientras la matriz OR permanece fija.

Los dispositivos PAL eran fabricados a partir de fusibles, encargados de las interconexiones en las matrices, que más tarde fueron remplazados por celdas CMOS eléctricamente borrables. Esto dio fruto a las estructuras GAL (Generic Array Logic) pudiendo así generar cualquier operación producto que se desee activando o desactivando las celdas. En la Figura 1 se muestra la diferencia entre estos dispositivos.

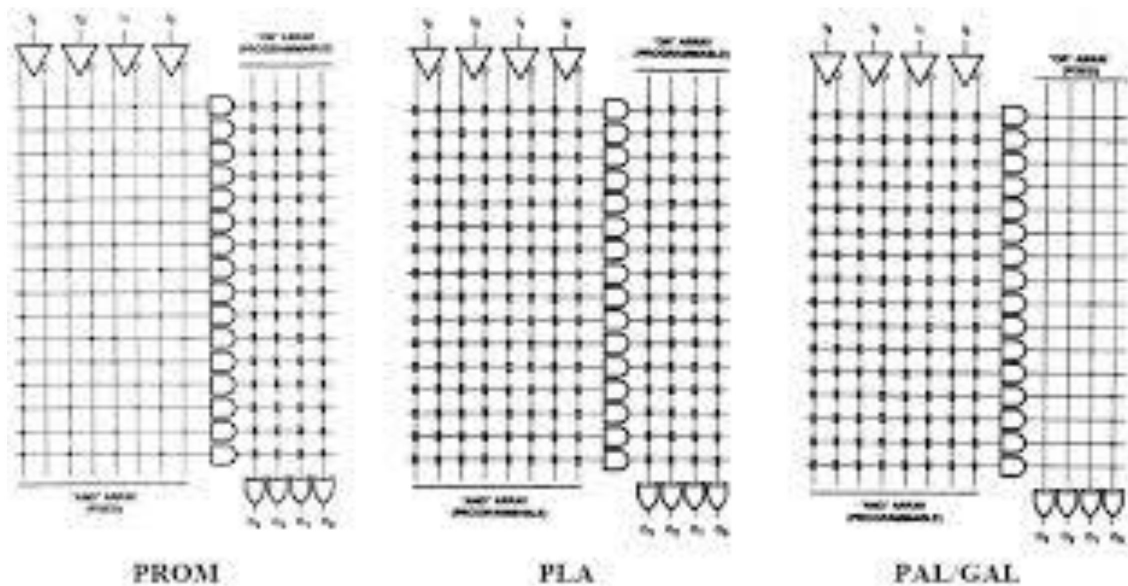


Figura 1: estructura interna de logicas programables

Para suplir la escasez de recursos de los dispositivos PLD se llegó al desarrollo de las FPGAs (Field Programmable Gate Array) las cuales son semiconductores reprogramables basados en matrices de bloque lógicos que pueden implementar una gran variedad de funciones lógicas, cuya funcionalidad e interconexión pueden ser configuradas por el usuario. La programación de las interconexiones permite un diseño personalizado según las necesidades del sistema, configurados después de su fabricación. Esto hace que cada tarea procesada sea asignada a un bloque lógico distinto, lo que permite añadir otros procesos sin que afecten a la aplicación.

A diferencia de los ASICs, que pueden implementar funciones de gran complejidad, el proceso de diseño con FPGAs es menos costoso y más rápido además de la ventaja de la realización de posibles cambios en el diseño de manera más sencilla. Por esta razón las FPGAs ofrecen la posibilidad de diseños complejos al igual que los ASICs pero con un diseño personalizado como los PLDs. Estas ventajas ofrecen la oportunidad a los diseñadores de obtener un entorno donde puedan realizar diseños hardware y software con las pruebas que ello conlleve sin la necesidad de incurrir a las costosas herramientas asociadas a los ASICs.

En la actualidad las FPGAs son usadas en la implementación de cualquier tipo de sistema, desde sistemas de radio a algoritmos de visión artificial, sistemas aeroespaciales y de defensa, emulación de hardware de computadora, etc. La mayor utilidad de estos dispositivos es la implementación de prototipos y diseños personalizados que implementen sistemas específicos en pequeñas cantidades.

1.2. Arquitectura de las FPGAs de Xilinx:

La estructura interna de las FPGAs de Xilinx está compuesta por celdas de bloques lógicos interconectados entre si y a bloques de I/O (Figura 2). Cada uno de estos bloques lógicos (CLB) consta de una parte combinacional, la cual permite implementar funciones lógicas, y de una parte secuencial, compuesta por biestables.

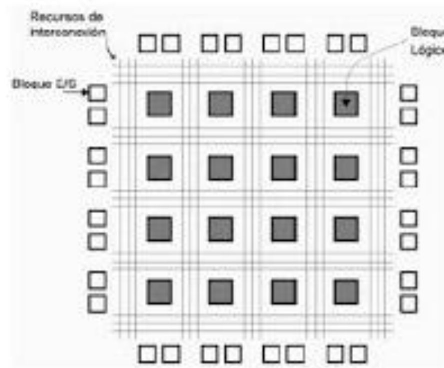
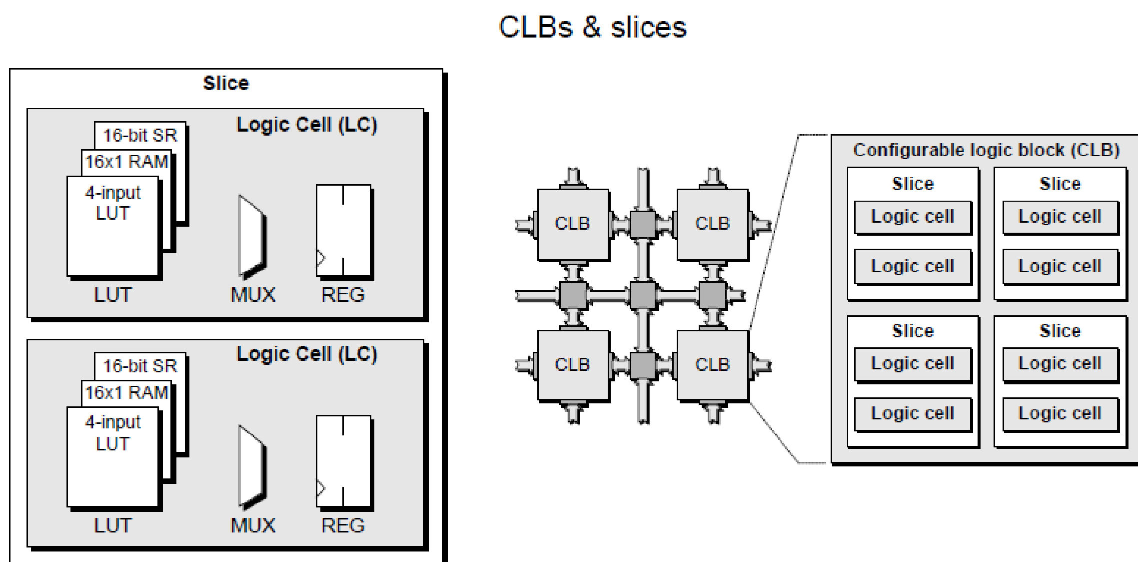


Figura 2: arquitectura básica de una FPGA

Los encargados de dotar la funcionalidad a las FPGAs son los Slices (Figura 3), agrupados para formar los bloques lógicos. Los Slice están compuestos por multiplexores, flip-flops y tablas de búsquedas (LUTs), encargados de implementar las funciones lógicas necesarias para el diseño. Una parte fundamental de estos Slices son las tablas de busqueda (LUTs), las cuales puede actuar como una RAM de 16x1 o como un registro de desplazamiento de 16 bits. Estas se hacen cargo de la parte combinacional de las FPGA cuyo concepto se asemeja al de una tabla de verdad donde las señales de entrada actúan como puntero de manera que cada posible combinación contenga el valor deseado. Las LUTs aportan un notable beneficio en diseños a altas frecuencias ya que el retardo producido por estas es constante, independientemente de la función lógica que se esté llevando a cabo.



The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)

Figura 3: arquitectura interna de una FPGA de Xilinx

Actualmente muchas aplicaciones requieren de bloques de memoria u otras funciones como multiplicadores o funciones MAC (multiply-and-accumulate) comunes en dispositivos DSPs, por esta razón muchas de las FPGAs los incluyen en sus estructuras de manera que puedan proporcionar al diseñador de todos los elementos necesarios para el sistema.

1.3. Ventajas y desventajas:

Las FPGAs son un término medio entre los PLDs y los ASICs por lo que aporta las mejores propiedades de cada uno de estos dispositivos.

Uno de los mayores beneficios de las FPGAs son la gran cantidad de bloques lógicos y terminales de entrada/salida que posee a diferencia de los PLDs, además de multitud de flip-flops, LUTs, funciones aritméticas y bloques de memoria RAM, lo que admite la implementación de sistemas más complejos y grandes.

El paralelismo a la hora de ejecutar algoritmos de procesamiento es otra de las grandes ventajas que proporcionan las FPGAs en comparación a las CPU en dispositivos DSP que trabajan de manera secuencial.

Son dispositivos reprogramables, lo que les da una mayor flexibilidad a la hora de realizar un diseño, al contrario que los ASICs las cuales solo pueden ser programados una vez. Esto reduce los costes y tiempo de desarrollo. Sin embargo, a diferencia de los ASICs, tienen un consumo mayor además de ser más lentas, aun así siguen siendo la mejor opción a la hora de realizar diseños personalizados o prototipos dado que permiten comprobar el comportamiento del sistema sin la necesidad de pasar por el proceso de fabricación que ello conlleva en los ASICs.

Para realizar el diseño del sistema se emplean los llamados 'lenguajes de descripción hardware'. Existen varios: VHDL, Verilog, Handle C, JBits; siendo el más común VHDL (Hardware Description Lenguaje), lenguaje que posibilita describir y modelar el sistema descomponiendo el diseño en subsistemas para después interconectarlos. Una vez programado el sistema el compilador se encarga de obtener un fichero binario bitstream, el cual contiene el diseño hardware a implementar por la FPGA, es decir, las interconexiones internas necesarias para llevar a cabo el diseño.

1.4. Comprobación de un diseño:

El desarrollo de un sistema se lleva a cabo a partir de distintas fases de modo que el proceso de diseño se realice de forma lo más sencilla y eficientemente posible con el objetivo de conseguir un diseño que funcione de forma correcta. Los pasos en el desarrollo de un sistema empiezan con la "creación" (análisis, planificación y ejecución), definiendo como realizar el producto de manera que cumpla las especificaciones requeridas por el usuario. A continuación, se comprueba el comportamiento del diseño a través de su simulación. Si se observa un comportamiento erróneo se analizará el fallo volviendo al diseño realizado hasta que se obtenga el comportamiento deseado.

Una vez se ha obtenido el diseño adecuado es necesario pasar a la síntesis de forma que el software diseñado sea comprensible por el dispositivo, traduciendo el lenguaje de

programación en elementos de lógica digital para su posterior implementación. Acto seguido, se realizará el 'place and route' de manera que se ubiquen y se interconecten los elementos del diseño, obteniendo los retardos de propagación de las señales.

La verificación de un diseño normalmente se realiza a través de la simulación RTL que junto con la simulación de tiempos se encarga de asegurar las posibles combinaciones de señales de entrada en las pruebas aplicadas. A partir de esta simulación se pasará a la verificación in-hardware donde es común usar vectores de test que verifiquen los requisitos.

El último paso del desarrollo de un sistema es la comprobación del producto, en esta fase se encuentran dos tipos de test: funcional (Functional Testing), en el que se comprueba que el diseño cumple con las especificaciones deseadas para poder llevarlo al mercado; y la comprobación en caso de fallo en el sistema. Los retardos de propagación, una mala implementación, efectos eléctricos, las condiciones del entorno, etc., son responsables de los posibles fallos en el sistema. Dependiendo de cual podría ser el causante del fallo y lo que se quiera comprobar se realizarán unas pruebas u otras. En esta última fase del proceso interviene como método de depuración ChipScope Pro, herramienta estudiada en este Trabajo Fin de Grado.

Para la realización de las pruebas que se llevan a cabo en la comprobación, una opción posible es el uso de equipos de test ATE (Automatic Test Equipment), equipos dedicados a la realización de pruebas del producto final (Figura 4). Estos equipos usan estímulos con los que pueden observar la respuesta del sistema de manera que se observe el comportamiento externo del diseño. Esto nos permite realizar un análisis funcional con el cual se podrá testear a velocidad normal. Sin embargo, este tipo de análisis es difícil e incluso hay situaciones que lo hacen imposible debido a que no es fácil ver señales internas. Por ello, otra forma de análisis que permite un ATE, llamada in-circuit testing, es a través de accesorios específicos los cuales pueden hacer las líneas de señal accesibles, no obstante esto reduce la velocidad de testeo a unos pocos megahercios.

Figura 4: Diagrama de bloques de de un sistema ATE

Al realizar la comprobación de un diseño implementado en una FPGA puede dar como desenlace que no se obtengan los resultados esperados debido a algún fallo en la implementación del código. Por ello es necesario realizar una depuración del sistema en busca de errores. La forma más común de comprobar un diseño en FPGA y en otros diversos dispositivos es el uso del puerto JTAG. No sólo se emplea como procedimiento de test sino también puede cumplir la función de comunicación entre dispositivos de manera que permitan la depuración del diseño. En el caso de la herramienta ChipScope Pro se empleará el puerto JTAG como interfaz de comunicación entre los módulos de la herramienta dentro del dispositivo FPGA y el analizador lógico que esta incorpora, permitiendo obtener en el PC una imagen detallada de lo que está ocurriendo en el dispositivo en tiempo real.

JTAG es el nombre usado para definir al estándar IEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. En este estándar se definen los circuitos que componen un puerto de acceso de prueba y la arquitectura Boundary-Scan, los cuales proporcionan una solución para probar y tener un mantenimiento de circuitos integrados embebidos. La lógica que se define permite probar las interconexiones entre los distintos chips una vez montado en un PCB además de poder observar la actividad del componente durante su funcionamiento.

JTAG es una norma abierta por lo que además de los comandos base es posible añadir comandos específicos para la aplicación que se esté empleando, adaptándose a las necesidades del diseño. En conclusión, JTAG define un puerto serie capaz de admitir una serie de comandos para realizar labores de test a nivel de tarjeta.

1.5. ChipScope Pro tool:

El sistema de desarrollo ISE de Xilinx ofrece la posibilidad de depurar los sistemas diseñados a través de la herramienta ChipScope Pro que integra un analizador lógico capaz de obtener las señales internas o nodos producidos tras la implementación del diseño. Esto se consigue de forma sencilla tras introducir en el diseño los módulos específicos a través del Core Generator o Core Inserter. Estos bloques son el ILA (Integrated Logic Analyzer) y el VIO (Virtual Input/Output). Una vez configurado el dispositivo será posible lanzar el analizador de ChipScope que hace uso de los módulos anteriormente insertados.

El módulo ILA usa un bloque RAM interno que permite almacenar las muestras de datos para su posterior volcado en un analizador lógico en el PC proporcionado por ChipScope. ILA contiene un máximo de 256 bits de datos (nodos), con un total de 16 triggers de entrada y un trigger externo de salida con la posibilidad de trabajar en un rango de 1 a 256 bits, además de una señal de reloj que marcará la frecuencia de muestreo. En caso que sea necesaria más capacidad de datos se podrían añadir varios módulos ILA aumentando los nodos de trabajo. El módulo ILA cuenta con múltiples triggers seleccionables y personalizables que permiten combinarse para ofrecer un único trigger con la condición a aplicar. Como ILA es un núcleo síncrono se puede utilizar cualquier reloj del diseño.

El módulo VIO permite añadir entradas y salidas virtuales al diseño, pudiendo ser tanto síncronas como asíncronas. A su vez, permite incluir LEDs u otros indicadores de estado en sus

salidas y swithes u otros controladores a sus entradas, además de contar con un tren de pulsos de 16-ciclos de reloj.

Xilinx Inc. escribió: "A medida que la complejidad del FPGA continúa aumentando, sigue la demanda de depuración y verificación más avanzadas. La herramienta ChipScope Pro proporciona el primer analizador de bus integrado en la industria para satisfacer estos requisitos. Con la herramienta ChipScope Pro, los usuarios pueden insertar núcleos de análisis de buses integrados completamente parametrizables (IBA- Integrated Bus Analyzer) directamente en un diseño y obtener acceso a los buses del sistema que son difíciles o imposibles de acceder mediante métodos tradicionales. El nuevo analizador ChipScope Pro permite al usuario ver los datos en una pantalla de nivel de ciclo de bus completa con marcas de tiempo. " [1]

Debido al hecho de que ChipScope se implementa dentro de la FPGA se pueden encontrar ciertas limitaciones. Esto es, la herramienta utiliza los recursos de la FPGAs como los bloques RAM y la lógica combinacional proporcionada por los Slices para usar los módulos, por lo que es necesario dejar espacio para la lógica que va autilizar la herramienta ChipScope. Por la misma razón, la memoria utilizada para el almacenamiento de las muestras capturadas por el analizador es limitada. La velocidad de muestreo no es tan alta como la de los analizadores externos, siendo el mejor de los casos aplicado por el módulo ILA una frecuencia de 412MHz, y la velocidad de reloj del sistema suele ser la que marca esta frecuencia. Por esta razón, no es posible detectar interferencias a través de ChipScope. La Figura 5 muestra las especificaciones generales de los núcleos ILA y VIO.

[1] Xilinx, Inc. (2002) *Xilinx ChipScope pro supports the industry's First integrated bus analyzer for programmable logic*. Available at: http://newit.gsu.by/resources/Journals%5Cdacafe%5C2002_03%5C21556.htm

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family [1]	Kintex-7[6], Virtex-7, Virtex-6[4], Virtex-5, Virtex-4, Spartan-6[5], Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP/XA				
Supported User Interfaces	Not applicable				
	Resources				**Frequency**
Configuration [3]	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**	**Max. Freq.**
Config1	156	270	0	1	313.239 MHz
Config2	391	698	0	4	243.858 MHz
Config3	4262	8400	0	228	412.788 MHz
Provided with Core					
Documentation	Product Specification User Guide				
Design Files	Netlist				
Example Design	Verilog /VHDL				
Test Bench	Not Provided				
Constraints File	Xilinx Constraints File				
Simulation Model	Not Provided				
Tested Design Tools [2]					
Design Entry Tools	CORE Generator tool, XPS				
Simulation	Not Provided				
Synthesis Tools	Not Provided.				
Support					
Provided by Xilinx, Inc.					

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family [1]	Kintex®-7, Virtex®-7, Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP/XA				
Supported User Interfaces	Not applicable.				
Provided with Core					
	Resources				**Frequency**
Configuration [2]	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**	**Max Freq**
Config1	60	87	0	0	399.808 MHz
Config2	131	291	0	0	399.808 MHz
Config3	227	543	0	0	399.808 MHz
Documentation	Product Specification User Guide				
Design Files	Netlist				
Example Design	Verilog /VHDL				
Test Bench	Not Provided				
Constraints File	Xilinx Constraints File				
Simulation Model	Not Provided				
Tested Design Tools					
Design Entry Tools	CORE Generator™ tool, XPS				
Simulation	Not Provided				
Synthesis Tools	Not Provided.				
Support					
Provided by Xilinx, Inc.					

Figura 5: especificaciones de los módulo ILA y VIO

De la misma manera, si el tiempo que se desea muestrear es mayor que el permitido por la señal de reloj interna, entonces es necesario introducir otra señal de reloj que nos permita obtener un periodo de muestreo adecuado. Esto es un problema ya que estamos añadiendo hardware adicional que ralentiza el diseño inicial, además de perder sincronismo.

CAPITULO 2:

ChipScope Pro Tool

En este capítulo se pretende analizar los diferentes núcleos de los cuales está compuesta la herramienta ChipScope Pro (ICON, ILA, VIO y ATC2). En particular, se hablará más en detalle de los núcleos ILA y VIO debido a que son los que intervienen principalmente en la depuración del diseño. Para ello se van a emplear distintos ejemplos que ilustren su funcionamiento general y sirvan a modo de manual para el usuario. El conexionado de los núcleos de ChipScope Pro en el diseño se muestra en la Figura 6.

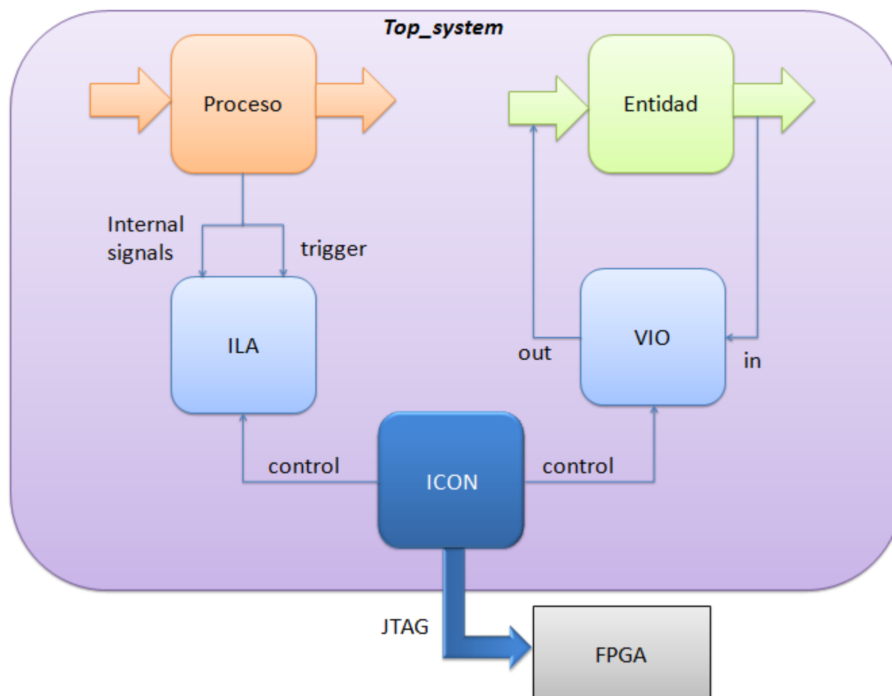


Figura 6: diagrama de bloques de los núcleos de ChipScope

Las señales internas que se deseen analizar se conectarán a ILA mientras las entradas a simular se conectarán a VIO. Ambos núcleos se comunicarán a través de las señales de control del núcleo ICON. Es importante destacar que no es posible visualizar los puertos de entrada y salida de los módulos a analizar a través de ILA Para ello se deberá usar señales auxiliares de modo que las entradas y salidas se conviertan en señales internas.

2.1. ICON

Para poder trabajar con los núcleos que proporciona la herramienta ChipScope es necesario realizar una comunicación entre el dispositivo FPGA y el PC. Dicha conexión se realiza a través del módulo ICON (Integrated Controller) que hace uso del interfaz JTAG Boundary Scan de la FPGA como protocolo de comunicación.

Utiliza las cadenas de exploración del usuario (USER scan chains) que proporciona el componente primitivo BSCAN de la FPGA. El núcleo ICON puede configurarse para que obtenga automáticamente el componente primitivo BSCAN o usar un BSCAN incluido en otra parte del diseño, la conexión de ambos casos se muestra en la Figura 7, donde se observa la necesidad de señales adicionales en el caso de usar un componente BSCAN externo. El bloque BSCAN permite una extensión de la interfaz JTAG a las cadenas internas de escaneo definidas por el usuario.

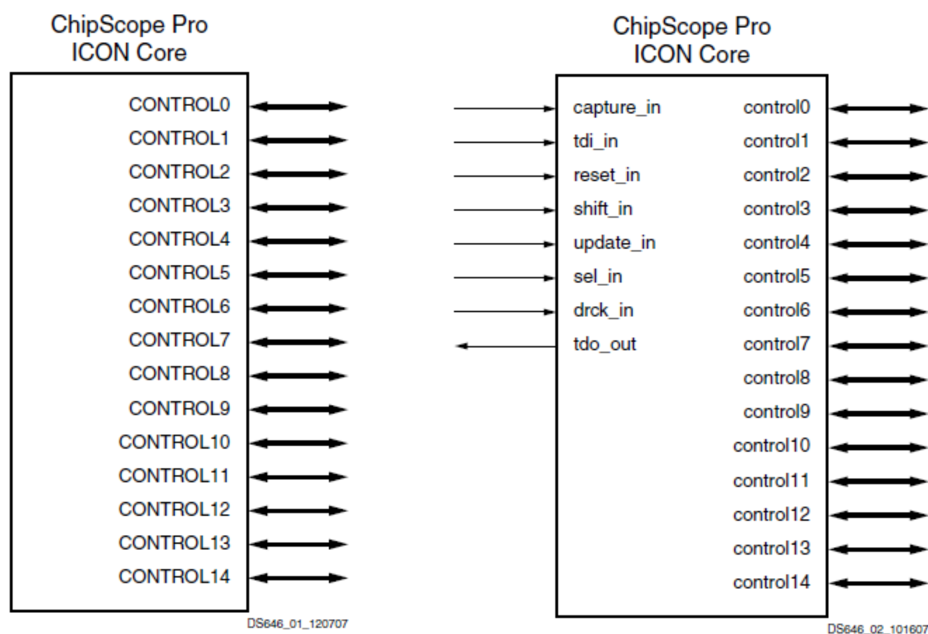


Figura 7: módulo ICON con BSCAN interno vs BSCAN externo

Este módulo tiene una fácil inserción en el diseño, pudiendo incorporarse a través de dos procesos distintos: Core Inserter o Core Generation. A lo largo del trabajo de fin de grado se ha decidido el uso del Core Generator para la introducción de los distintos núcleos. De este modo se evita que se estén mezclando conceptos y se pueda entender de una manera sencilla el objetivo principal de la herramienta y las posibilidades que ofrece ChipScope a la hora de depurar un diseño en FPGAs.

A través de la herramienta Core Generator se puede elegir cualquier módulo disponible en el entorno de desarrollo ISE de Xilinx. En este caso se desean insertar los módulos de depuración correspondientes a ChipScope, concretamente ICON. En la Figura 8 se muestran las opciones que permite Core Generator en la versión de ISE utilizada en la realización del trabajo de fin de grado. Si se elige este método para su inserción será necesario instanciar el componente en el código del diseño a posteriori.

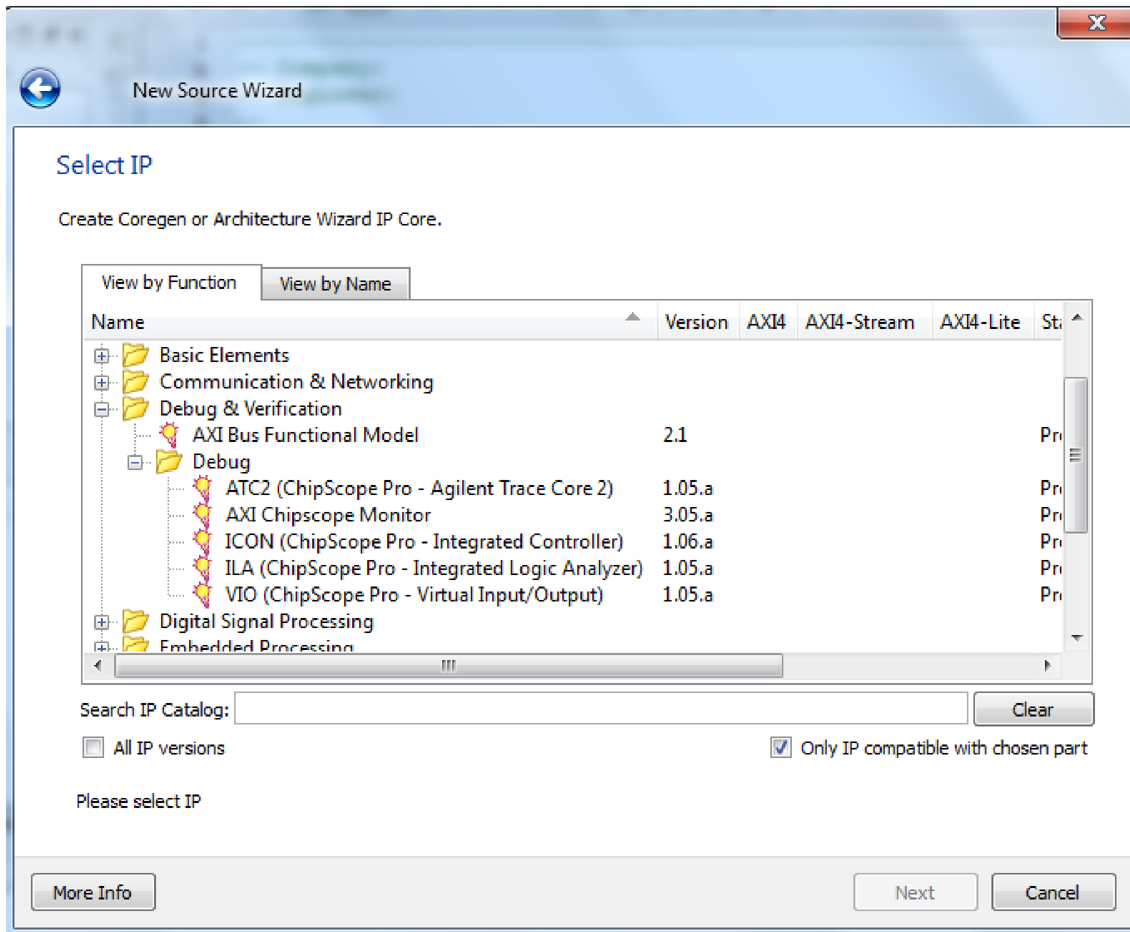


Figura 8: opciones en la generación de módulos compatibles con la versión ISE utilizada

Como primer paso se deben elegir los puertos que se desea que el núcleo ICON contenga en función de los núcleos que se vayan a insertar, estos pueden ser ILA, VIO, ATC2 o AXI. Cada núcleo debe estar conectado a un único puerto de control de ICON permitiendo hasta un máximo de 15 núcleos conectados. Sin embargo, solo es posible insertar un único núcleo ICON por diseño. En la Figura 9 se puede ver el cuadro correspondiente a la generación del núcleo ICON.

Figura 9: generación del módulo ICON

Tras generar el módulo el entorno de desarrollo genera un fichero *.vho* que proporciona el código necesario, junto con las instrucciones a seguir, que permiten la instanciación del componente en el código del diseño, este se puede ver en la Figura 10.

```
--------------------------------------------------------------------------------
-- Copyright (c) 2017 Xilinx, Inc.
-- All Rights Reserved
--------------------------------------------------------------------------------
--   ____  ____
--  /   /\/   /
-- /___/  \  /    Vendor     : Xilinx
-- \   \   \/     Version    : 14.2
--  \   \         Application: XILINX CORE Generator
--  /   /         Filename   : ICON.vho
-- /___/   /\     Timestamp  : Fri Mar 03 20:00:05 Hora estándar GMT 2017
-- \   \  /  \
--  \___\/\___\
--
-- Design Name: ISE Instantiation template
-- Component Identifier: xilinx.com:ip:chipscope_icon:1.06.a
--------------------------------------------------------------------------------
-- The following code must appear in the VHDL architecture header:

------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component ICON
  PORT (
    CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));

end component;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------
-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG

your_instance_name : ICON
  port map (
    CONTROL0 => CONTROL0);

-- INST_TAG_END ------ End INSTANTIATION Template ------------
```

Figura 10: fichero ICON.vho

2.2. ILA

El núcleo ILA (Integrated Logic Analizer) es un analizador lógico personalizable que permite capturar las señales internas del diseño. El módulo está compuesto por una serie de señales de disparo (triggers) de entrada que permiten capturar las muestras de datos al detectar determinados eventos. Además, incorpora un trigger de salida que puede ser usado como una señal de disparo externa, como lógica adicional o para incluir varios núcleos ILA interconectados. La Figura 11 muestra el conexionado del núcleo ILA.

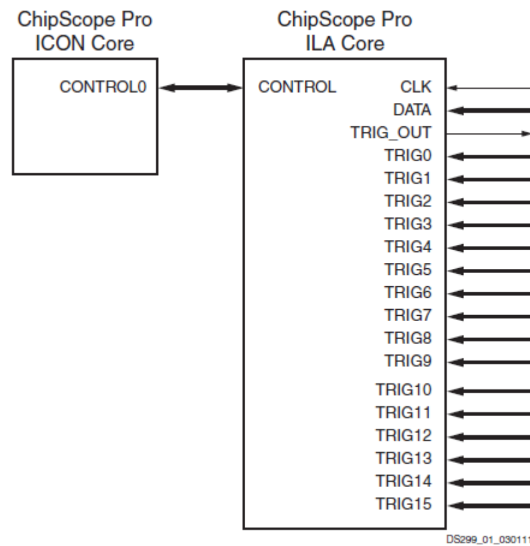


Figura 11: conexión del módulo ILA

La captura e información de los datos se almacenan usando los recursos proporcionados por un bloque RAM on-chip del dispositivo, esto quiere decir que consumirá recursos adicionales de la placa pudiendo ocasionar que no se tengan suficientes recursos si se trata de un diseño bastante complejo, por lo que se debe tener en cuenta la capacidad del dispositivo.

Una vez el diseño está cargado en el dispositivo FPGA se utilizará el software ChipScope Analizer para observar las señales capturadas. Antes de comenzar con la captura de datos es necesario configurar la señal de disparo, esto es, el momento en el que el analizador comienza a obtener las muestras de las señales a observar. Las posibilidades de configuración de la señal de disparo que ChipScope contiene provee al diseñador de una mayor facilidad de depuración del diseño.

El núcleo ILA contiene 16 puertos trigger de entrada con un tamaño máximo de 256 bits cada uno. Cada puerto trigger es capaz de conectarse hasta a un total de 16 unidades match las cuales son las encargadas de realizar las operaciones convenientes para obtener la señal de disparo que haya sido indicada por el usuario. Gracias a las unidades match es posible realizar múltiples comparaciones de las señales, operaciones AND u OR y obtener un secuenciador de trigger de varios niveles. También permiten la configuración de los puertos de manera que se pueda generar una señal de disparo gracias a la aparición de flancos, tanto de subida como de bajada.

Para diferentes tipos de señales y buses son necesarios distintos puertos de disparo (triggers) ya que si se utiliza un único puerto para todas las señales es posible que no se pueda observar las transiciones de bits individuales.

En caso de que se requiera una captura especial de muestras se puede habilitar la opción '*storage qualification condition*'. Esto permite evalúar los eventos de la unidad match del puerto trigger para decidir si se debe o no almacenar cada muestra de datos individualmente. Los datos capturados se corresponden a la condición indicada de antemano. Esto permite el ahorro de recursos de la memoria RAM on-chip al almacenar únicamente los datos que interesan para la depuración del diseño.

Otra posibilidad que ofrece el núcleo ILA es la configuración de un contador de eventos. Esto es, la captura de datos a partir de un determinado número de sucesos ocurridos en la señale de disparo.

El puerto trigger de salida puede actuar como un disparador para equipo de prueba externos uniendo la señal a un pin de salida, así como interrupción para lógica interna, trigger o para conectar varios módulos ILA en cascada. Sin embargo, hay que tener en cuenta al usar este puerto que la señal resultante tendrá una latencia de 10 ciclos de reloj.

Ejemplo

A continuación se mostrará un ejemplo sencillo que emplea el núcleo ILA para monitorizar las señales resultantes de un contador de 4 bits. El diseño cuenta con una señal de entrada ***U/D*** que indicará el sentido del contador, ascendente o descendente. Una señal de ***ce*** que habilite la cuenta, señal de fin de cuenta ***tc*** y señal de expansión ***ceo***, activa cuando se alcance el fin de cuenta y ***ce*** se encuentre activa. El módulo a diseñar es el mostrado en la Figura 12.

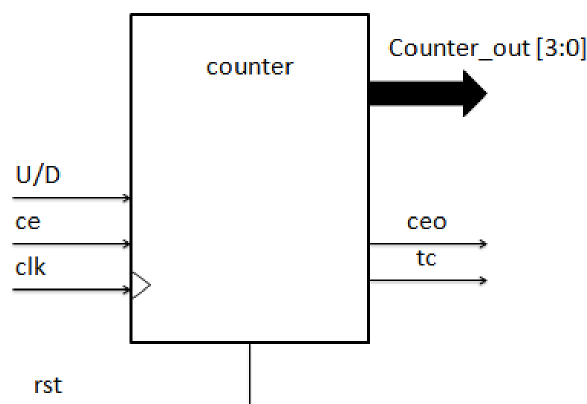


Figura 12: contador 16 bits a diseñar

Una vez generado el código en VHDL del diseño se insertará el módulo ILA. En este ejemplo se desea observar el comportamiento del sistema por lo que las señales a observar serán los datos de salida del contador, la señal de habilitación, el switch que controla la polaridad y las señales ***ceo*** y ***tc*** de salida, de este modo se puede comprobar el correcto funcionamiento de

todas las señales que intervienen en el diseño. Como evento de disparo utilizado para capturar las muestras de las señales se empleará la señal ***ce*** asociada a un switch de la placa Atlys. Al ser señales de entrada y salida se deben usar señales auxiliares de modo que se puedan conectar al núcleo ILA. En la Figura 13 se observan las señales que interactúan con el núcleo ILA.

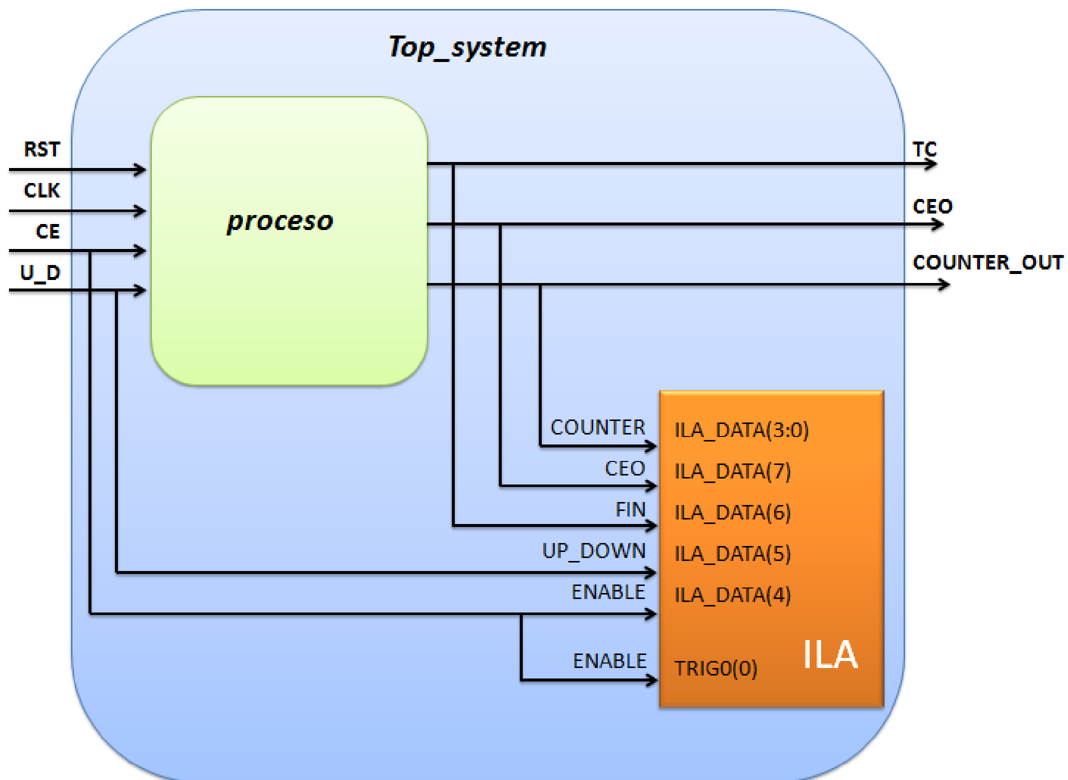


Figura 13: señales conectadas al módulo ILA

Como primer paso es necesario insertar el núcleo ICON que permita las comunicaciones entre placa y PC. A continuación, se insertará el núcleo ILA, con un único puerto trigger de entrada de 4 bits y el puerto de datos de un tamaño de 8 bits. Se ha configurado el analizador para que capture y muestre hasta un máximo de 16384 muestras de datos tal y como se muestra en la Figura 14 y Figura 15.

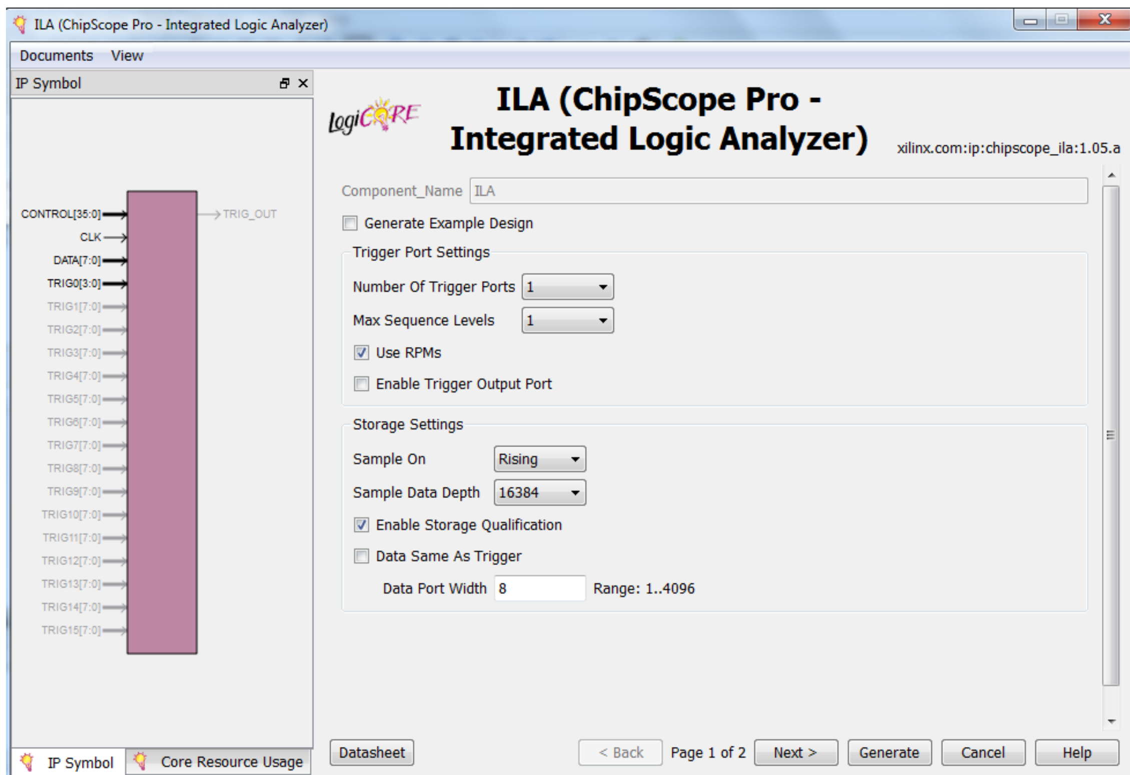


Figura 14: configuración de los puertos del módulo ILA

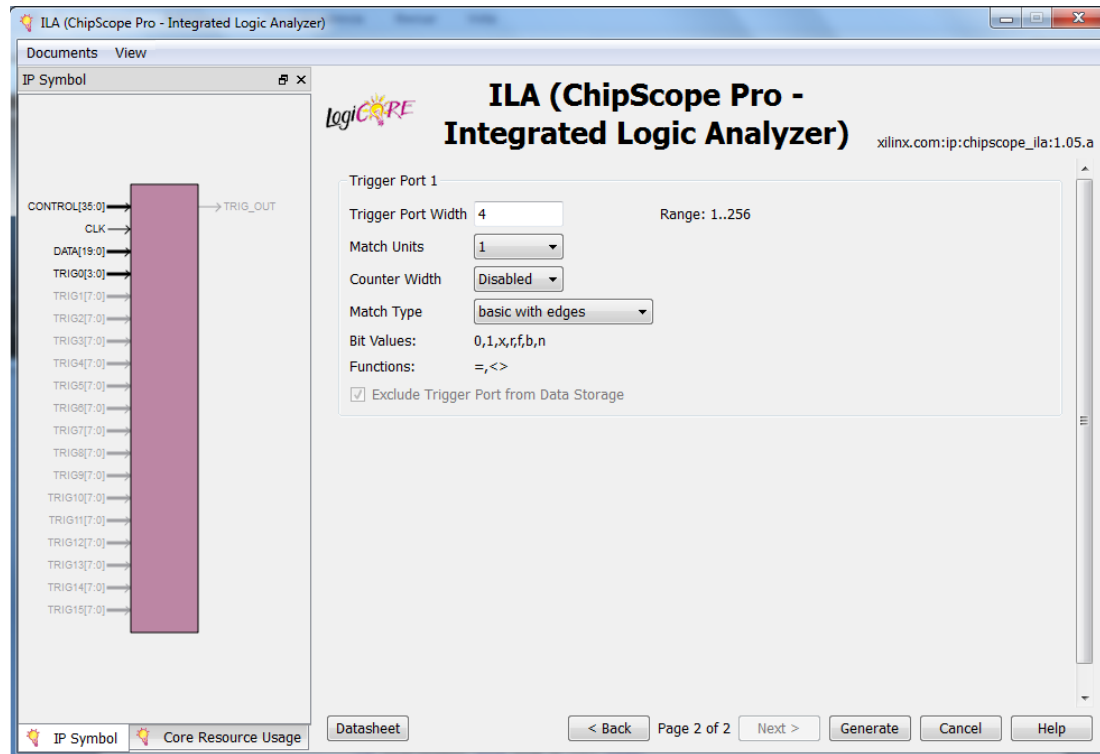


Figura 15: configuración del trigger del módulo ILA

La configuracion del trigger tiene la opcion de elegir distintos tipos de unidad match en función de la señal de disparo que se desea. Las opciones que icluye se muestran en la Figura 16.

Trigger Port 1

Trigger Port Width 8 Range: 1..256

Match Units 1

Counter Width Disabled

Match Type basic with edges

Bit Values: basic

Functions: basic with edges

Exclude Trigger extended

extended with edges

range

range with edges

Figura 16: configuración de las opciones de match del trigger

Una vez creado el módulo ILA a traves del Core Generator se genera el archivo *.vho* que incluye el código a incluir en el diseño para su instanciación (Figura 17).

```
--------------------------------------------------------------------------------
-- Copyright (c) 2017 Xilinx, Inc.
-- All Rights Reserved
--------------------------------------------------------------------------------
--   ____  ____
--  /   /\/   /
-- /___/  \  /    Vendor     : Xilinx
-- \   \   \/     Version    : 14.2
--  \   \         Application: XILINX CORE Generator
--  /   /         Filename   : ILA.vho
-- /___/   /\     Timestamp  : Mon Jun 26 21:02:22 Hora de verano romance 2017
-- \   \  /  \
--  \___\/\___\
--
-- Design Name: ISE Instantiation template
-- Component Identifier: xilinx.com:ip:chipscope_ila:1.05.a
--------------------------------------------------------------------------------
-- The following code must appear in the VHDL architecture header:

------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component ILA
  PORT (
    CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
    CLK : IN STD_LOGIC;
    DATA : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
    TRIG0 : IN STD_LOGIC_VECTOR(3 DOWNTO 0));

end component;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------
-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG

your_instance_name : ILA
  port map (
    CONTROL => CONTROL,
    CLK => CLK,
    DATA => DATA,
    TRIG0 => TRIG0);

-- INST_TAG_END ------ End INSTANTIATION Template ------------
```

Figura 17: fichero .vho generado en la creacion del módulo ILA

Una vez instanciado el componente en el código es necesario conectar las señales que se desean observar a través del módulo ILA, así como las señales de disparo. En este caso las señales conectadas al módulo ILA se reflejan en la Figura 18. El resto del código es proporcionado en el Anexo II (Ejemplo ILA: Contador).

```
ila_data(3 downto 0)<= std_logic_vector(counter);
ila_data(4)              <= enable;
ila_data(5)         <= up_down;
ila_data(6)         <= fin;
ila_data(7)         <= ceo_aux;
trig0(0)            <= enable;
trig0(3 downto 1)        <= (others => '0');
```

Figura 18: conexionado de las señales a observar con el núcleo ILA

Después de haber conectado los puertos del módulo, y tras programar el dispositivo, se puede ejecutar el analizador lógico de ChipScope desde la herramienta Analyze Design Using ChipScope de ISE. Una forma de programar el dispositivo es a través de la herramienta iMPACT, donde se hará uso de la plataforma de cable USB de Xilinx (Platform Cable USB II) que permite la depuración del diseño gracias a la utilización del puerto de comunicación JTAG.

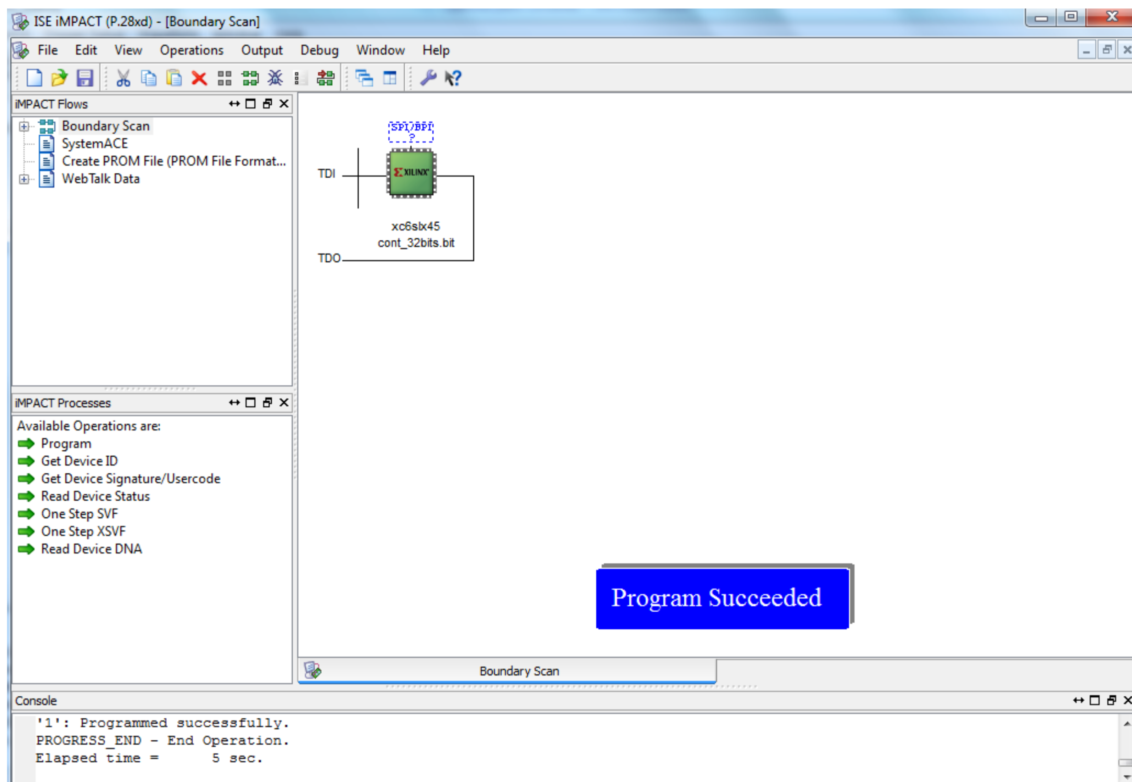


Figura 19: pantalla de configuración de iMPACT

En la pestaña Boundary Scan se añadirá el dispositivo a programar haciendo clic con el botón derecho y *'Add Xilinx Device'* o en su icono correspondiente mostrado en la Figura 20. Se añadirá el archivo *bitstream (.bit)* del proyecto que se quiera programar en la placa. Tras añadir el archivo se programará el dispositivo haciendo clic derecho sobre la imagen del dispositivo y en *'program'*. Como resultado se obtendrá la pantalla mostrada en la Figura 19.

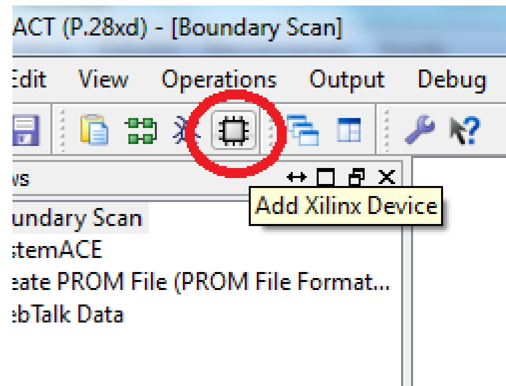


Figura 20: icono utilizado para añadir el proyecto a programar en la placa

Tras haber programado el dispositivo es posible depurar el proyecto que se encuentra en la placa Atlys. Para ello se debe ejecutar la herramienta ChipScope Pro Analyzer (Figura 21).

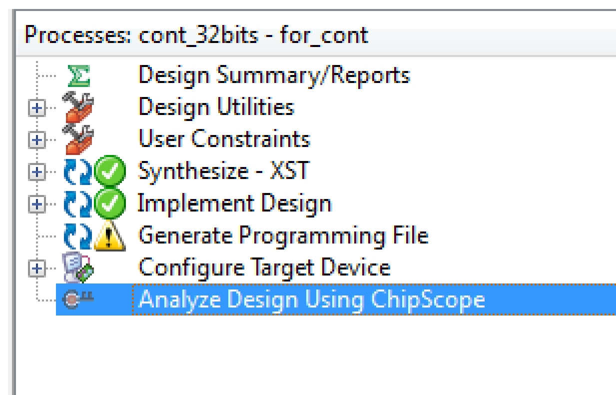


Figura 21: analizador lógico de la herramienta ChipScope Pro

En primer lugar, es necesario indicar el dispositivo con el que se va a trabajar. Para ello se debe hacer clic en el icono 'Open Cable/Search JTAG Chain' (Figura 22).

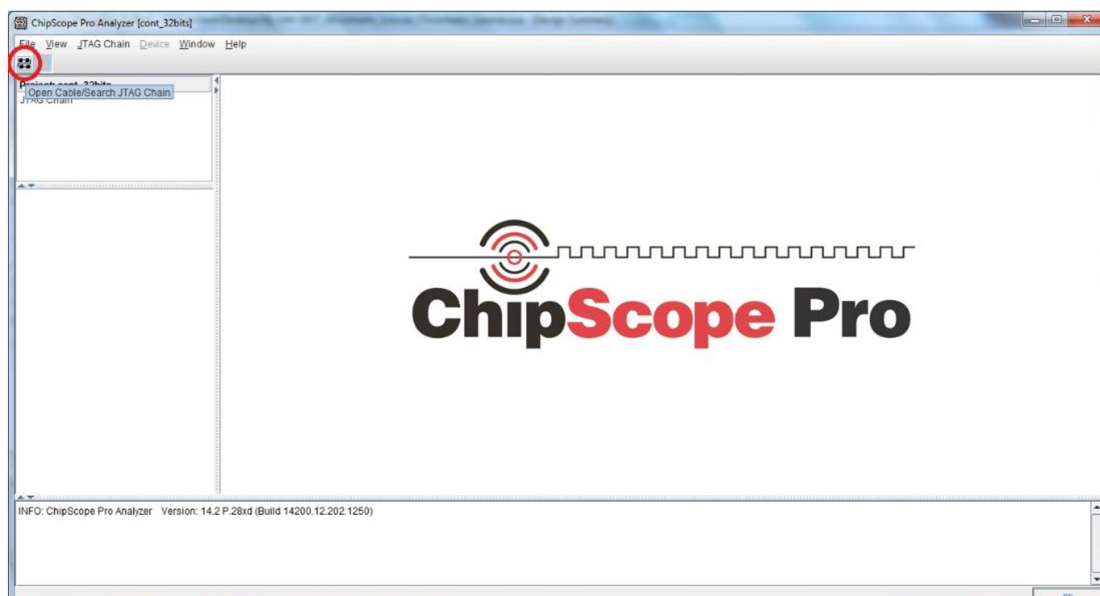


Figura 22: pantalla inicial de la herramienta ChipScope Pro

Se tendrá la opción de elegir entre los dispositivos conectados al PC que estén usando una cadena JTAG. Se debe elegir el dispositivo con el cual se esté trabajando, en este caso solo se posee una placa Atlys por lo que se elige dicha opción (Figura 23).

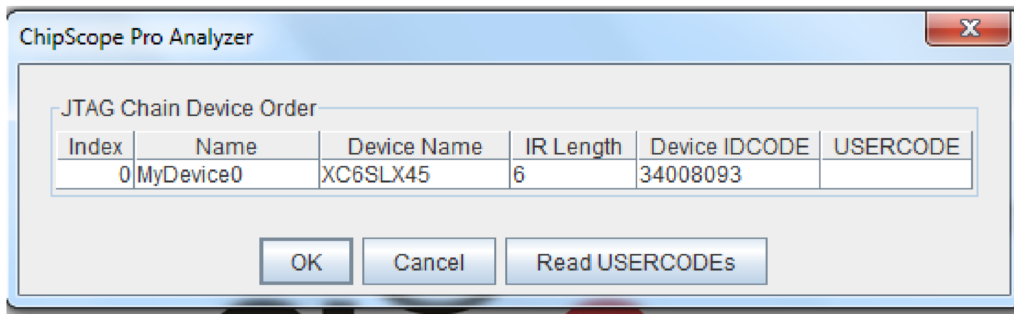


Figura 23: configuración del dispositivo a analizar

La pantalla obtenida nada más lanzar el analizador lógico muestra las señales y los triggers aun sin configurar. (Figura 24)

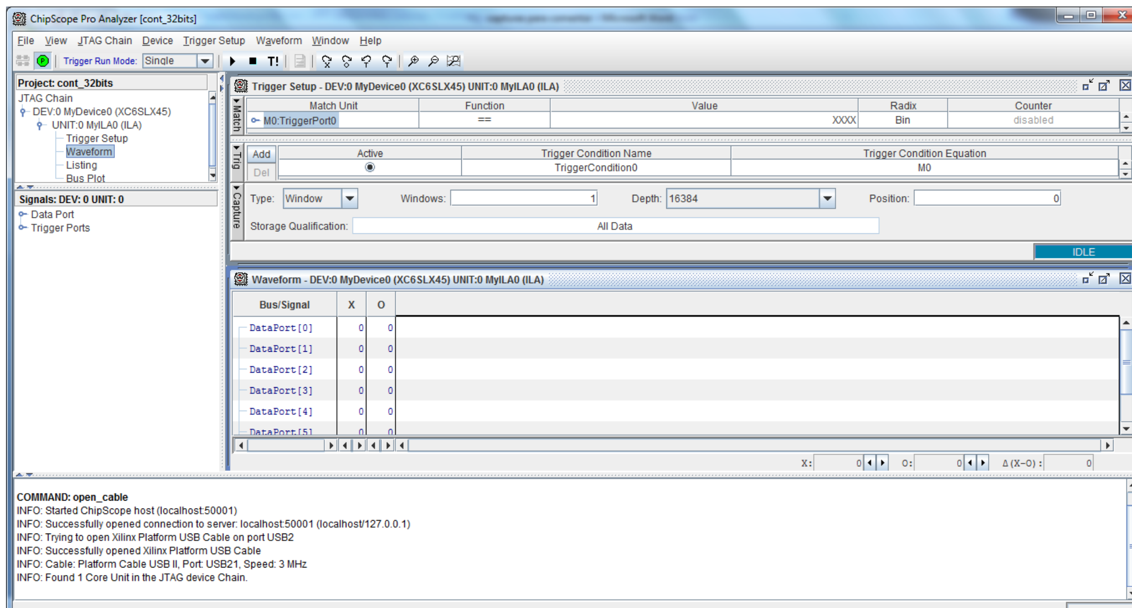


Figura 24: señales del analizador lógico

Para una mayor comodidad a la hora del análisis de las señales se cambiará el nombre de los bits del bus de datos a los de sus señales correspondientes. Aparte de eso es conveniente agrupar los bits de una misma señal para crear un bus de datos. (Figura 25)

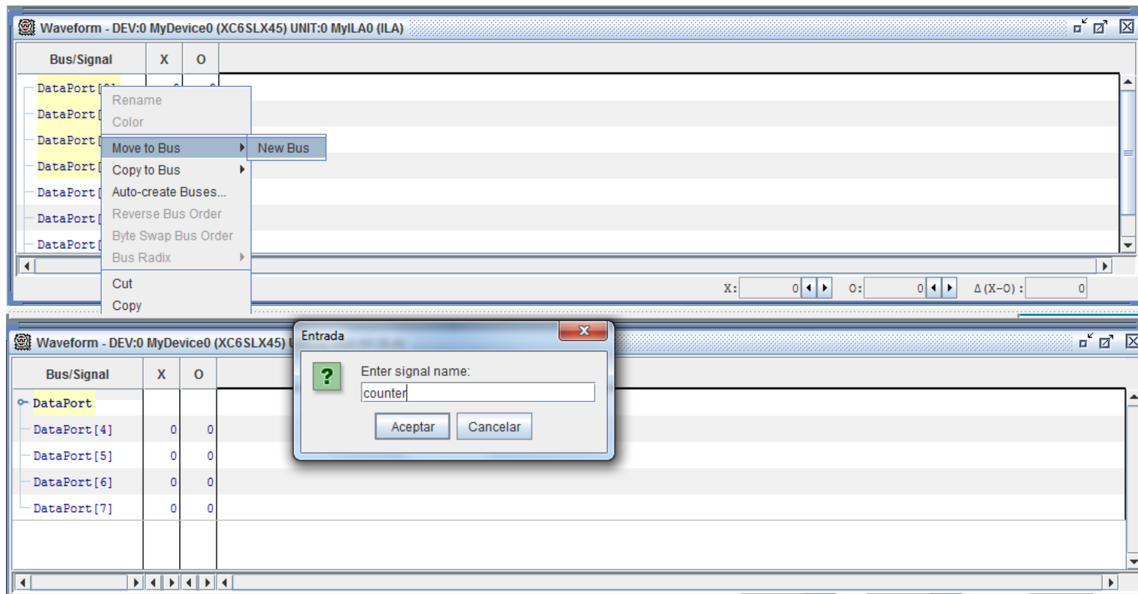


Figura 25: adaptación de las señales a visualizar al proyecto con el que se está trabajando

Una vez se han adaptado las pantallas de visualización al proyecto se debe configurar los triggers. La configuración del trigger se realiza principalmente en la pantalla *Trigger Setup* mostrada en la Figura 26. En esta pantalla se puede indicar el valor con el cual se iniciará la señal de disparo. Esta señal puede iniciarse a partir de un flanco de subida/bajada o un valor binario o hexadecimal. La señal de disparo se activará si el puerto cumple con la función indicada en la configuración. En este caso la señal de disparo se ha configurado de modo que la señal ***enable*** conectada en el puerto trigger cumpla cualquier condición (XXXX).

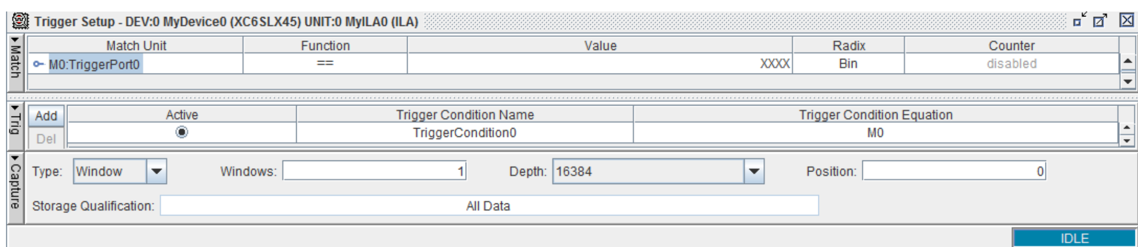


Figura 26: configuración del trigger

Existe la opción de combinar varios puertos trigger para que actúen como señal de disparo. Para ello se hace clic en la opción *Trigger Condition Equation*, apareciendo la pestaña mostrada en la Figura 27, donde se indicarán las condiciones a aplicar como señal de disparo.

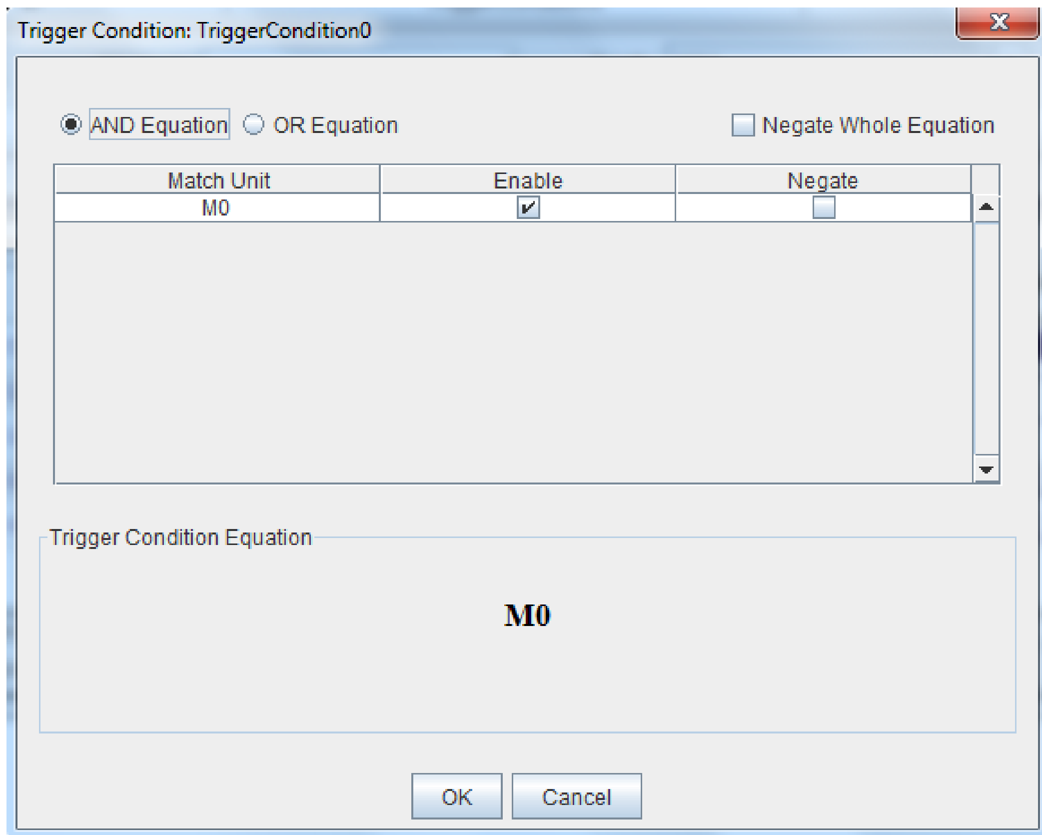


Figura 27: configuración de la condición del trigger a aplicar

Si se ha habilitado la opción 'Storage Qualification' al añadir el núcleo ILA al diseño, entonces es posible configurar la condición la cual permita el almacenamiento y visualización de las muestras capturadas. Esta condición se establece en la pantalla mostrada en la Figura 28 tras hacer clic en 'Storage Qualification: All Data'. Solo se visualizarán las muestras que cumplan con la condición impuesta. Las señales para la configuración de esta opción coinciden con las conectadas al puerto trigger.

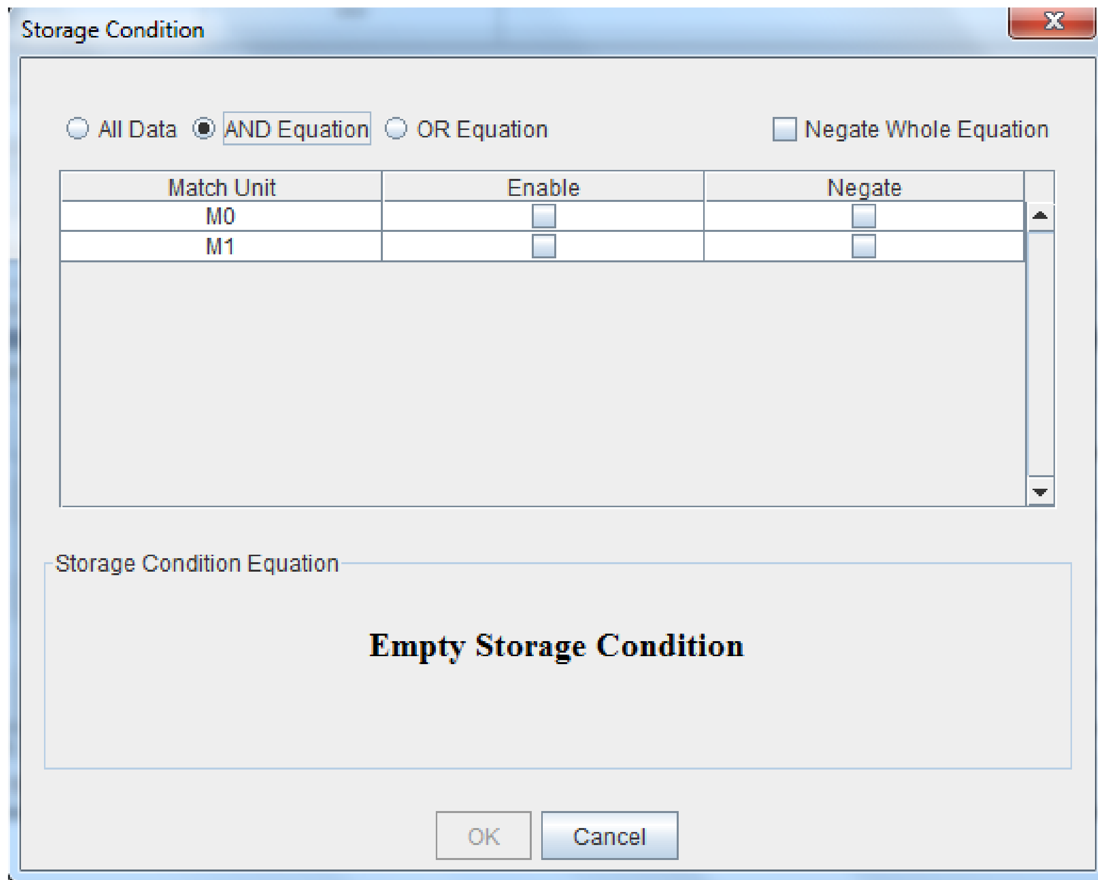
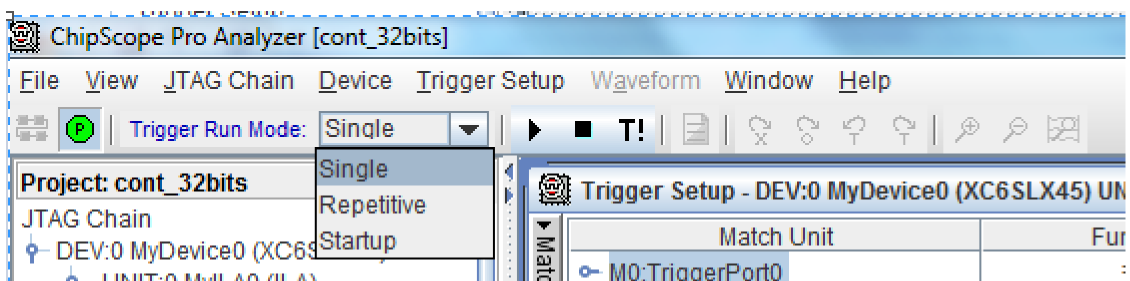


Figura 28: configuración de la opción 'storage condition' a aplicar

El Analizador lógico tiene tres modos de ejecución de la opción de disparo: Single, Repetitive y Startup (Figura 29). Lo que quiere decir que se puede ejecutar una única vez, que se ejecute de manera repetitiva o que se ejecuten las condiciones de disparo guardadas anteriormente por el usuario automáticamente. También existe la posibilidad de lanzar el analizador una única vez sin esperar a la condición de disparo impuesta al hacer clic en el icono ***T!***.



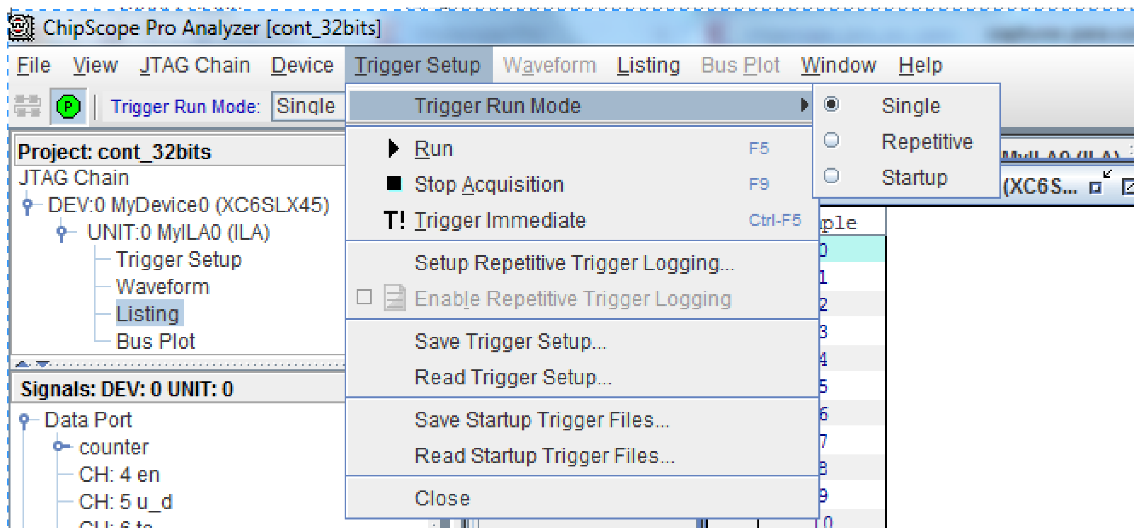


Figura 29: opciones a la hora de ejecutar el trigger

En modo 'single' el trigger se ejecuta una única vez, si se desea volver a capturar datos es necesario re-armar de forma manual el trigger haciendo clic en 'Run'. Mientras en la opción 'repetitive' este rearme se efectúa de forma automática. Existe la posibilidad de registrar los datos obtenidos en cada iteración en un archivo mediante la opción '*Setup Repetitive Trigger Logging*'.

El modo 'Startup' permite la activación de la señal de disparo sin la necesidad de rearmar el núcleo ILA a través de la herramienta Analyzer. Para ello necesita de un fichero CTJ, donde se especifiquen las condiciones de disparo, y un fichero UCF que contenga el diseño. Estos ficheros se crean a partir de la configuración de los triggers usada en modo 'single'. Solo es necesario usar la opción *'Save Startup Trigger Files'*. Una vez se han creado se debe re-implementar el diseño con dichos archivos incluidos (Figura 30). Tras la programación del dispositivo FPGA el núcleo ILA se armará, guardando así las muestras capturadas al darse la condición de disparo configurada. Esto se efectúa sin la necesidad de ejecutar la herramienta ChipScope Pro Analyzer.

Para ver los datos capturados se hará uso de la herramienta Analyzer en modo 'startup' mostrando así las señales obtenidas anteriormente. Para usar nuevamente esta opción se debe reconfigurar el dispositivo. Esta opción es beneficiosa cuando se desea capturar señales desde el mismo instante en el que se ha programado el dispositivo FPGA con el diseño. Sin embargo solo es soportada por las familias Virtex-6, Spartan-6 y los nuevos productos en la familia de la serie 7.

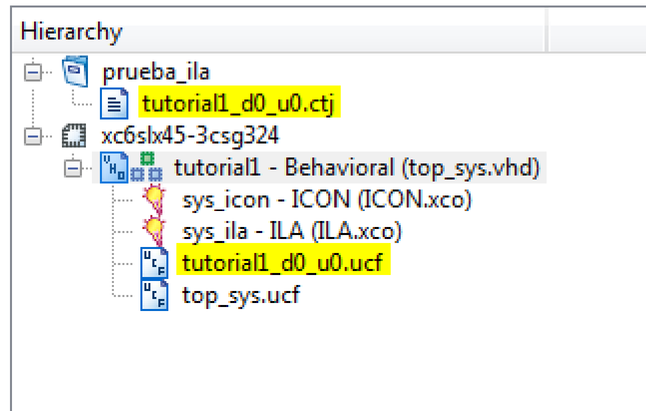


Figura 30: configuración del modo Startup del trigger

Una vez configurado el analizador lógico adaptado a las necesidades de este ejemplo la pantalla resultante será la mostrada en la Figura 31. Ahora es posible lanzar el analizador lógico que empezará a almacenar muestras una vez ocurrida la señal de disparo establecida. Estas muestras se visualizarán por pantalla cuando se haya llenado el buffer de datos.

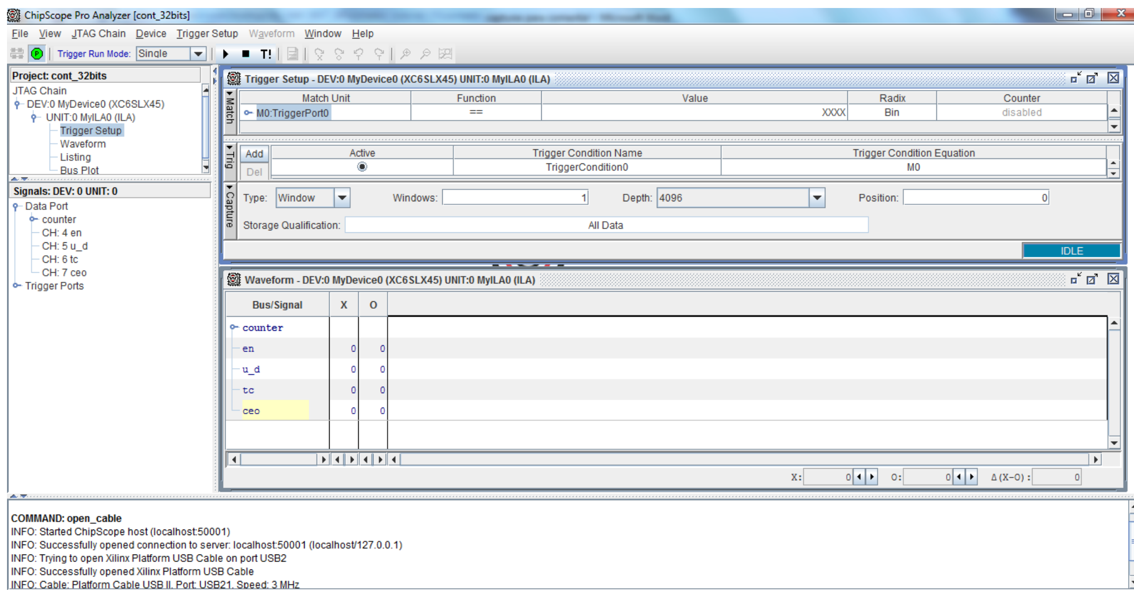


Figura 31: pantalla resultante tras la configuración de los parámetros del analizador lógico

Como se puede observar en la Figura 32, en el ejemplo realizado, tras iniciar la captura de datos y que se haya dado la condición impuesta por el trigger, se mostrará por pantalla la forma de las señales que se han indicado en el puerto data del núcleo ILA. A partir de los resultados se puede medir el tiempo deseado con los cursores o comprobar el funcionamiento del diseño.

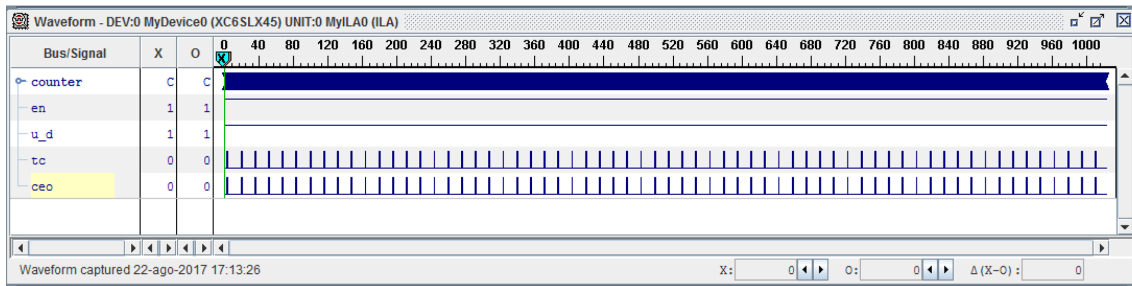


Figura 32: resultado del analizador lógico

Tras realizar un zoom se puede observar más claramente en la Figura 33 el correcto funcionamiento del diseño realizado.

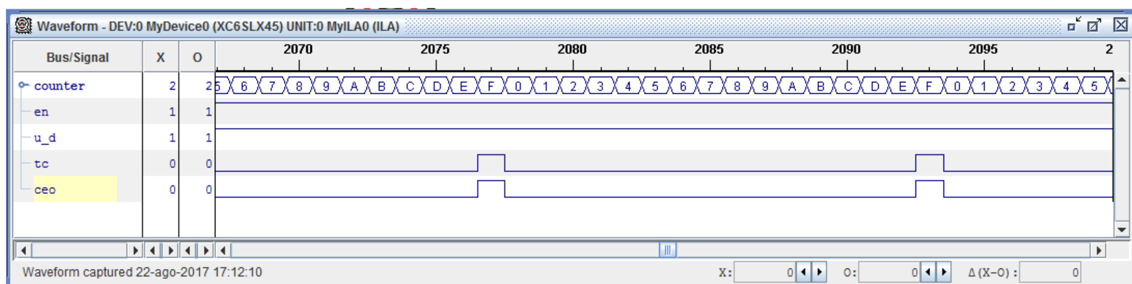


Figura 33: funcionamiento del diseño tras aplicar un zoom

El núcleo ILA ofrece la posibilidad de obtener la imagen de un bus de datos con respecto al tiempo, tal y como se muestra en la Figura 34. Esta opción permite observar la forma de onda del bus en lugar de ver únicamente su valor en la pantalla '*waveform*'.

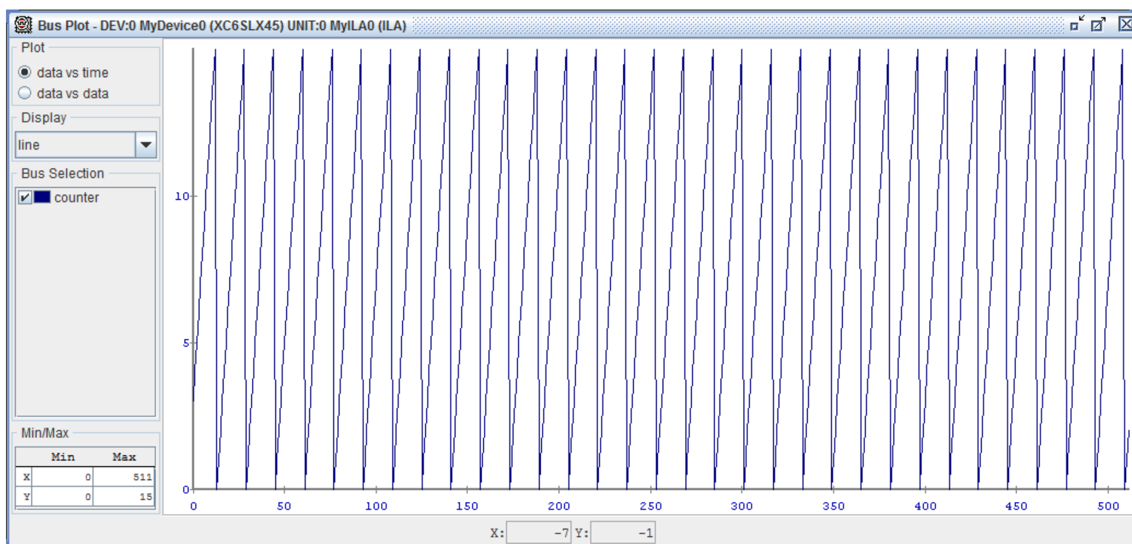


Figura 34: ventana Bus Plot del analizador lógico

Además, ofrece la opción de ver el valor de las muestras capturadas en forma de tabla. Esto es a través de la ventana '*Listing*' (Figura 35).

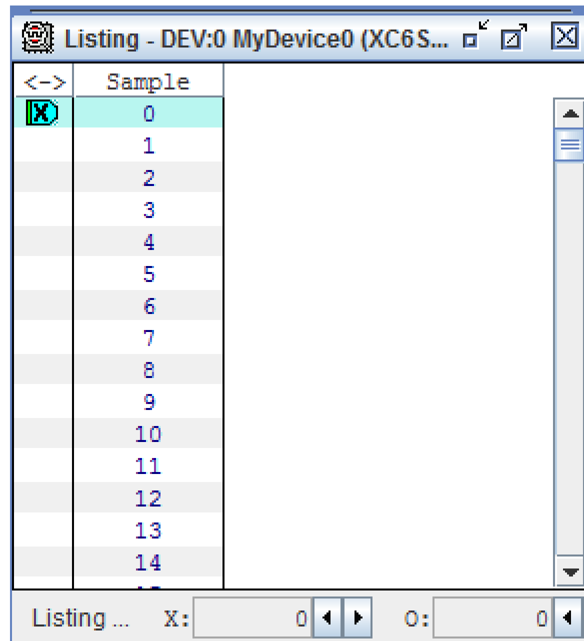


Figura 35: ventana Listing del analizador lógico

2.3. VIO

El núcleo VIO (Virtual Input/Output) permite tener acceso de una manera sencilla a entradas o salidas internas del diseño, simulando conexiones I/O de las entidades que componen el sistema. Al igual que el núcleo ILA, la interacción con este núcleo se realiza a través del uso del interfaz ICON conectado al JTAG del dispositivo (Figura 36). Al simular entradas/salidas del diseño se pueden recrear situaciones que permitan comprobar partes internas, asegurando que el módulo examinado funciona correctamente, descartando así fallos tras la depuración del diseño.

Para simular las entradas y salidas del módulo a depurar se dispone de LEDs y botones virtuales con los que poder interactuar con estas entidades además de detectores de actividad en los puertos de entrada síncronos que permiten comprobar si se han efectuado flancos de subida o de bajada entre el tiempo que transcurre entre muestras. El detector de flancos entre transiciones da la posibilidad de detectar glitches así como transiciones asíncronas en las señales de entrada síncronas.

En las salidas síncronas de este núcleo se puede obtener un tren de pulsos de 16 ciclos a velocidad de diseño.

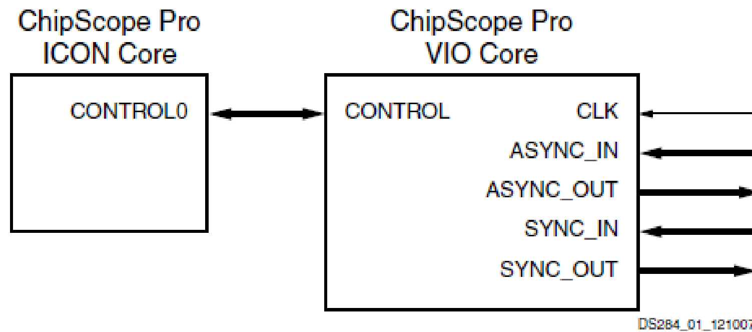


Figura 36: conexión del módulo VIO

Ejemplo

Para la demostración del núcleo VIO se va a implementar un ejemplo sencillo que sirva de guía para el uso de la herramienta ChipScope. El ejemplo a diseñar será una ALU. Este módulo debe ser capaz de realizar operaciones como sumas, restas, AND u OR sobre dos datos de 8 bits. En la Figura 37 se muestra el diagrama de bloques del diseño. Este contiene las señales ***op1***, ***op2*** y ***proceso*** de entrada y ***res*** de salida.

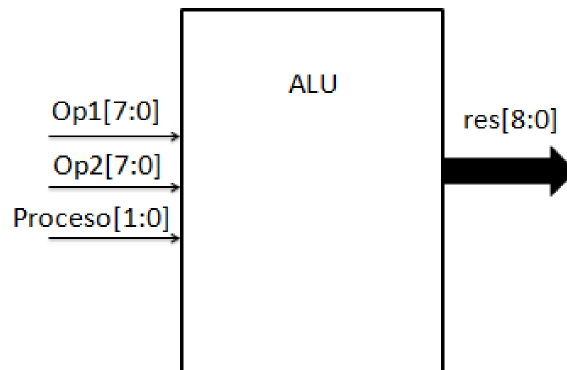


Figura 37: diagrama de bloques del diseño a implementar

Una vez se ha generado el código y se ha realizado la simulación del diseño es hora de programar la placa de manera que se incluya el uso de la herramienta ChipScope para su depuración. El primer paso para la incorporación de ChipScope en el diseño es la introducción de los núcleos a emplear (Figura 38).

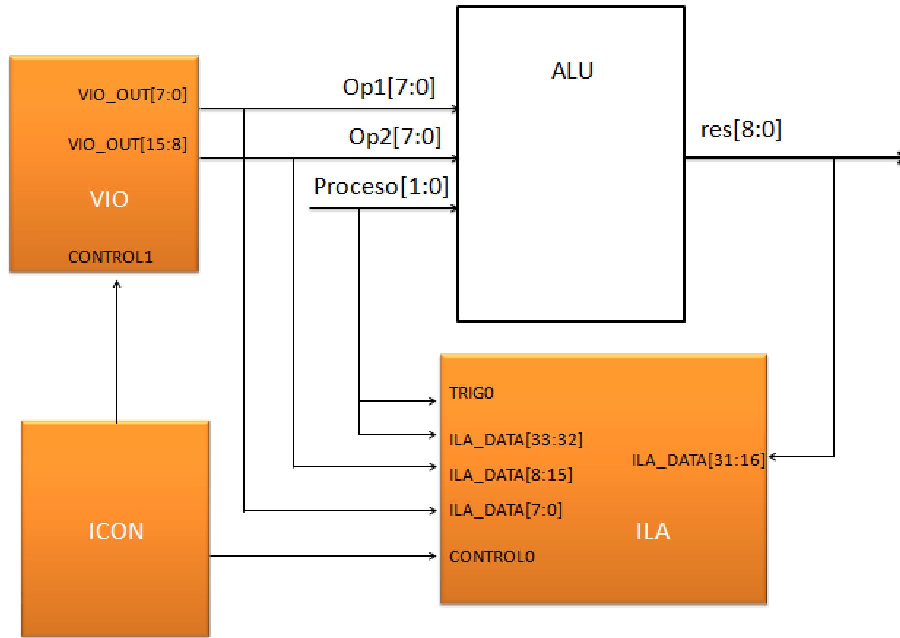


Figura 38: diagrama de bloques resultante al insertar ChipScope Pro

Es obligatorio insertar el núcleo ICON en todo diseño en el cual se quiera usar la herramienta ChipScope debido a que actúa como interfaz de conexión entre placa y PC. En este caso se tienen dos señales de control ya que se van a insertar dos núcleos para el análisis, los núcleos ILA y VIO. La pantalla de configuración del ICON se mostrará como en la Figura 39.

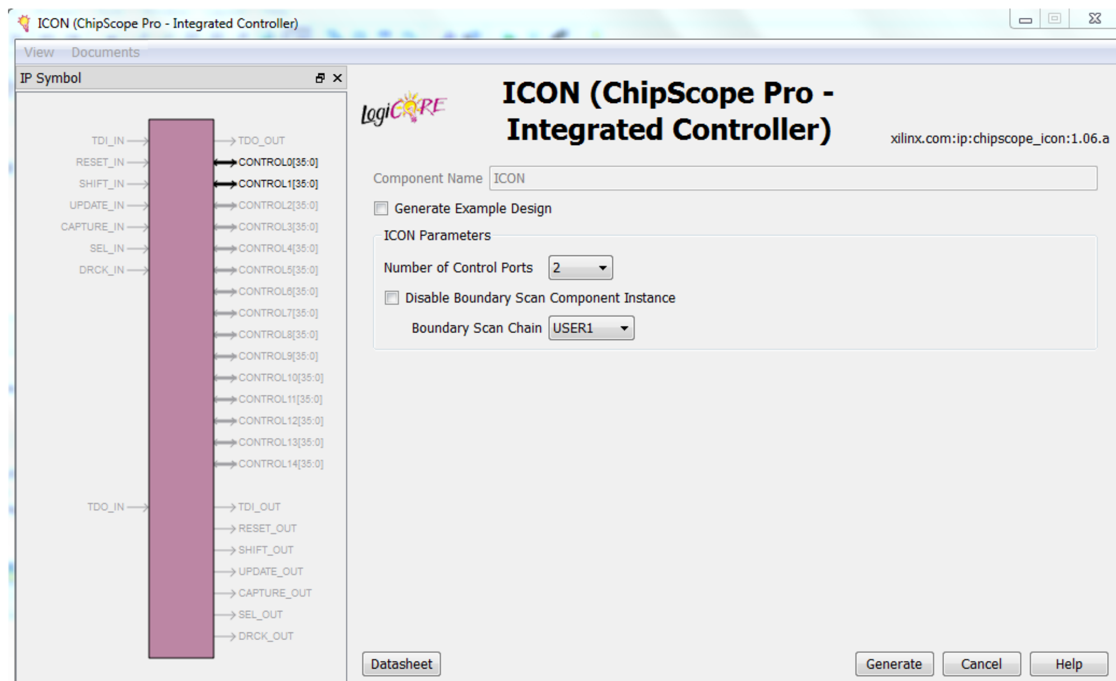


Figura 39: generación del módulo ICON para dos núcleos

Se introduce el núcleo ILA adaptado a las señales que se vayan a observar. Como se desean ver las señales ***op1, op2*** y ***proceso*** se tiene un bus de datos de 27 bits. En este ejemplo se va a

proceder a habilitar la opción 'storage qualification' para mostrar su uso. En la Figura 40 se observa la configuración de las señales que intervienen en el uso del núcleo ILA, mientras en la Figura 41 se indicarán los parámetros del trigger.

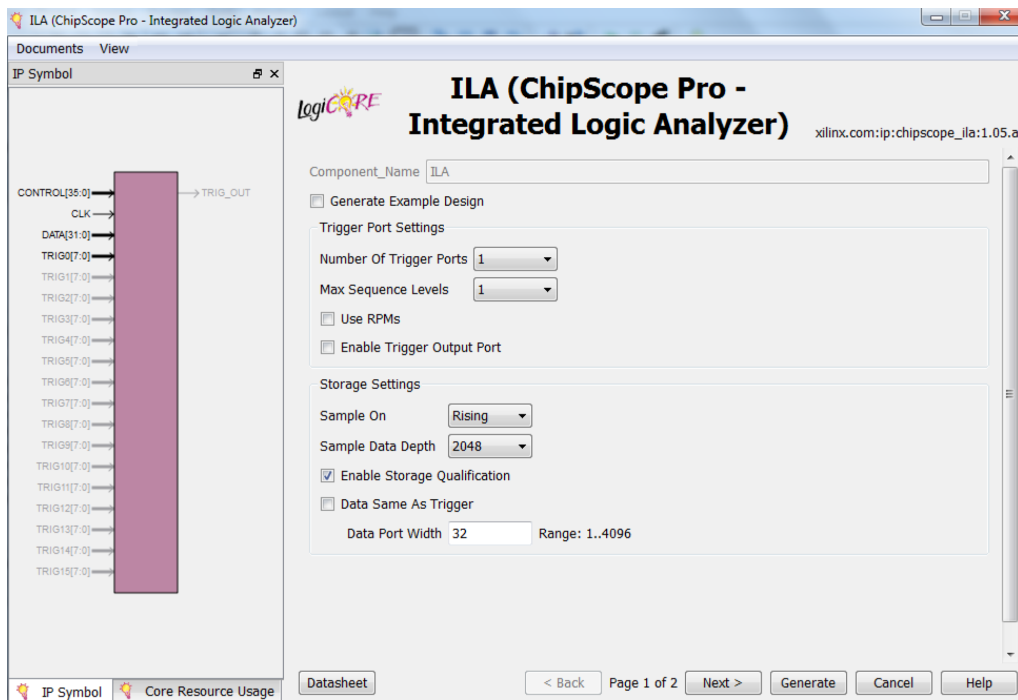


Figura 40: configuración del puerto de datos del núcleo ILA

Como trigger se ha optado por la señal de ***proceso*** de manera que el tamaño de trigger será de 2 bits.

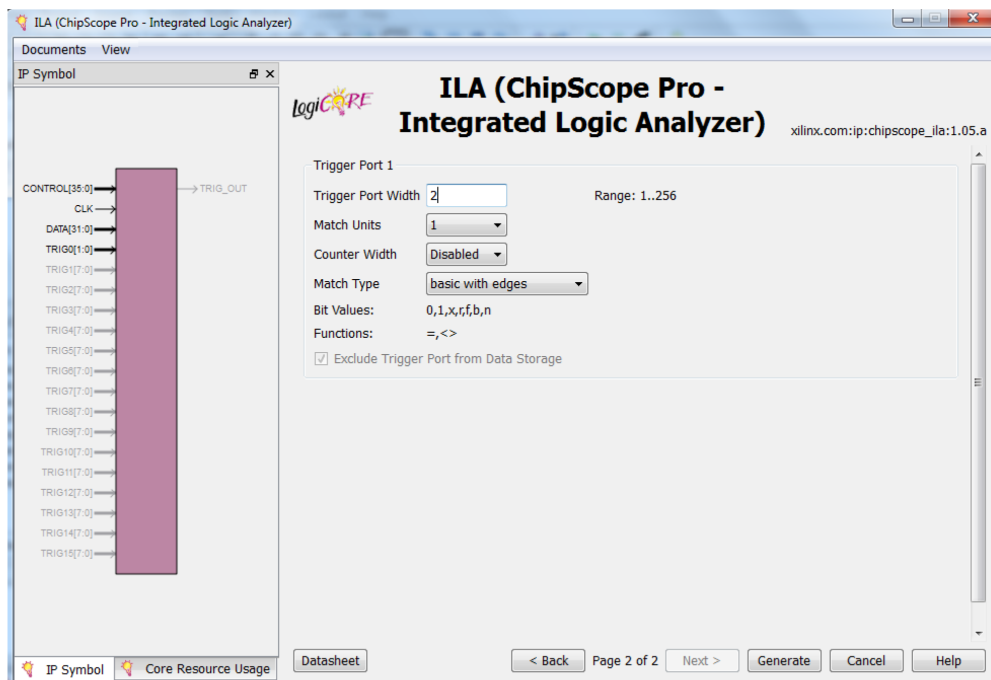


Figura 41: configuración de los puertos trigger del núcleo ILA

Por último, se inserta el núcleo VIO. Se desea comprobar el correcto funcionamiento de la ALU con diferentes entradas. Para ello se va a utilizar el núcleo VIO para simular dichas entradas. Al no existir señal de reloj se configurará el núcleo VIO con una salida asíncrona cuyas señales irán conectadas a las entradas del diseño (Figura 42).

Figura 42: configuración del núcleo VIO

Al igual que con los núcleos ICON e ILA, se genera un fichero *.vho* el cual contiene las instrucciones para la instanciación del componente VIO en el diseño (Figura 43).

```
-- Copyright (c) 2017 Xilinx, Inc.
-- All Rights Reserved
-------------------------------------------------------------------------------
--
--
--                    Vendor     : Xilinx
--                    Version    : 14.2
--                    Application: Xilinx CORE Generator
--                    Filename   : VIO.vho
--                    Timestamp  : Mon Aug 28 17:41:20 Hora de verano romance 2017
--
--
--
-- Design Name: ISE Instantiation template
-- Component Identifier: xilinx.com:ip:chipscope_vio:1.05.a
-------------------------------------------------------------------------------
-- The following code must appear in the VHDL architecture header:

------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component VIO
  PORT (
    CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
    ASYNC_OUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));

end component;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------
-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG

your_instance_name : VIO
  port map (
    CONTROL => CONTROL,
    ASYNC_OUT => ASYNC_OUT);

-- INST_TAG_END ------ End INSTANTIATION Template ------------
```

Figura 43: archivo .vho del núcleo VIO generado por la herramienta IPCore

Una vez instanciados los núcleos de ChipScope como se indican en las instrucciones dadas por el fichero *.vho* se deben conectar las señales que se vayan a utilizar con los núcleos. En la Figura 44 se muestra la instanciación de ChipScope, el código completo se muestra en el Anexo II (Ejemplo VIO: ALU).

```
ENTITY alu IS
   PORT(
    clk          : in  std_logic;
     -- op1 : IN std_logic_vector(7 DOWNTO 0);
     -- op2 : IN std_logic_vector(7 DOWNTO 0);
    proceso : IN std_logic_vector(1 DOWNTO 0);
    res : OUT std_logic_vector(8 DOWNTO 0));
   END alu;

ARCHITECTURE synth OF alu IS

  signal op1  :  std_logic_vector(7 DOWNTO 0);
  signal a,b  :  UNSIGNED(op1'range);
  signal c    :  UNSIGNED(res'range);
  signal proc  : std_logic_vector(1 DOWNTO 0);

        -- chipscope signals
signal control0:std_logic_vector(35 downto 0);
signal control1 :std_logic_vector(35 downto 0);
signal ila_data :std_logic_vector(26 downto 0);
signal trig0    :std_logic_vector(1 downto 0);
signal vio_out  :std_logic_vector(15 downto 0);
```

```
BEGIN
----------- chipscope------------------------
--------------------------------
 my_icon:entity work.ICON
   port map (
      CONTROL0 => control0,
      CONTROL1 => control1);

 my_ila:entity work.ILA
   port map (
      CONTROL => control0,
      CLK => clk,
      DATA => ila_data,
      TRIG0 => trig0);

 my_vio:entity work.VIO
   port map (
      CONTROL => control1,
      ASYNC_OUT => vio_out);
----------------------------------------------
------------------------------------
```

Figura 44: código de instanciación de ChipScope

En el caso del núcleo VIO se están simulando entradas y salidas de un módulo por lo que la conexión de estas señales se realiza de forma distinta que en el núcleo ILA. Estas se conectarán a los puertos de entrada que se quieran simular. Es posible programar el conexionado de dos maneras. Una de ellas es eliminando los puertos de entrada del módulo y conectando en su lugar las señales procedentes del núcleo VIO. En el ejemplo de la ALU que se está explicando se usa este método, por eso se puede ver en la Figura 44 como los puertos del módulo que se van a conectar al núcleo VIO están comentados. La otra opción es insertar el módulo a comprobar en un nuevo módulo 'top_system' el cual contenga a su vez los núcleos de ChipScope. El resultado final del conexionado de las señales del diseño con los núcleos de ChipScope se puede apreciar en la Figura 45.

```
a    <= UNSIGNED(vio_out(7 downto 0));
b    <= UNSIGNED(vio_out(15 downto 8));
res <= std_logic_vector(c);
proc <= proceso;

ila_data(7 downto 0)    <= std_logic_vector(a);
ila_data(15 downto 8)   <= std_logic_vector(b);
ila_data(24 downto 16)  <= std_logic_vector(c);
ila_data(26 downto 25)  <= proc;
trig0(1 downto 0)       <= proc;
```

Figura 45: conexión de las señales que van a intervenir en la depuración con ChipScope

Después de la conexión de las señales a monitorizar a los módulos de ChipScope se programa la placa Atlys al igual que en el ejemplo del núcleo ILA, haciendo uso de iMPACT. Después que la placa esté programada y lista para funcionar se lanza el analizador lógico de ChipScope.

Independientemente de los núcleos que se estén empleando en el diseño, el analizador lógico de ChipScope trabaja de la misma forma. Por lo tanto, la conexión de la placa y la primera

pantalla de trabajo será la misma que la mostrada en el ejemplo anterior. Por esta misma razón para este ejemplo se va a explicar únicamente las opciones que VIO tiene, obviando la configuración dada para el uso de ILA la cual se muestra en la Figura 46.

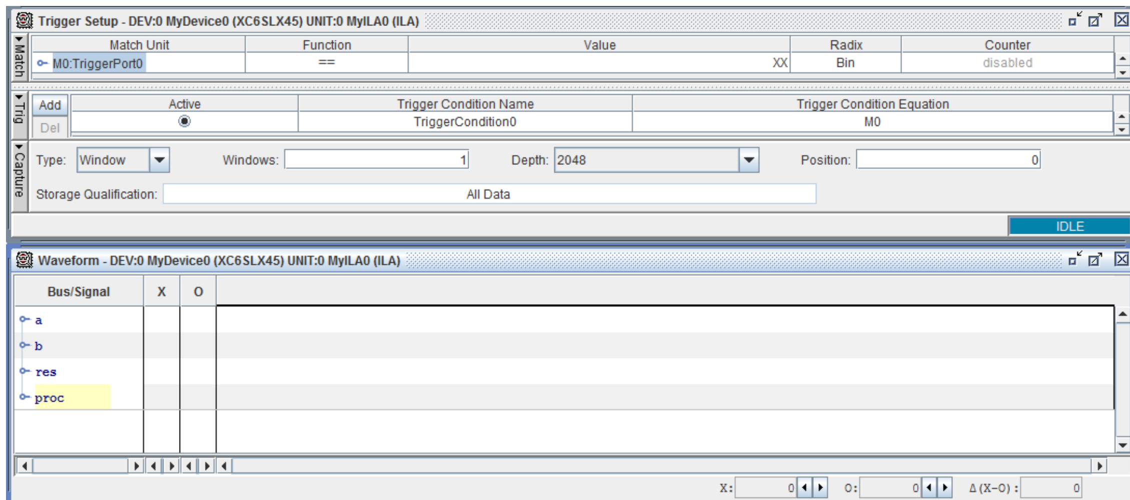


Figura 46: ventana del analizador lógico de ILA

Como primer paso se adaptará la ventana de la consola VIO a las señales que se estén empleando en el diseño a depurar ya que en un primer momento la consola VIO aparecerá sin configurar (Figura 47). Para ello se crearán los buses y se usarán los nombres correspondientes a las señales del diseño que intervienen haciendo clic en el botón derecho y seleccionando la opción adecuada.

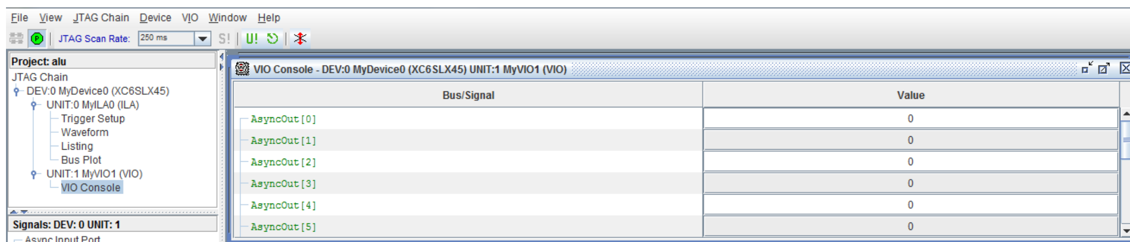


Figura 47: ventana consola VIO sin configurar

Las salidas del módulo VIO pueden configurarse de modo que realicen la función deseada. Se pueden configurar de forma que simulen un botón seleccionando la opción 'Push Button' o 'Toggle Button' dependiendo si se quiere un valor fijo o un pulso. La opción 'Text Field' permite la entrada de valores por teclado en el bus de datos. Las opciones a configurar dependen del tipo de señal que se esté simulando y se muestran en la Figura 48.

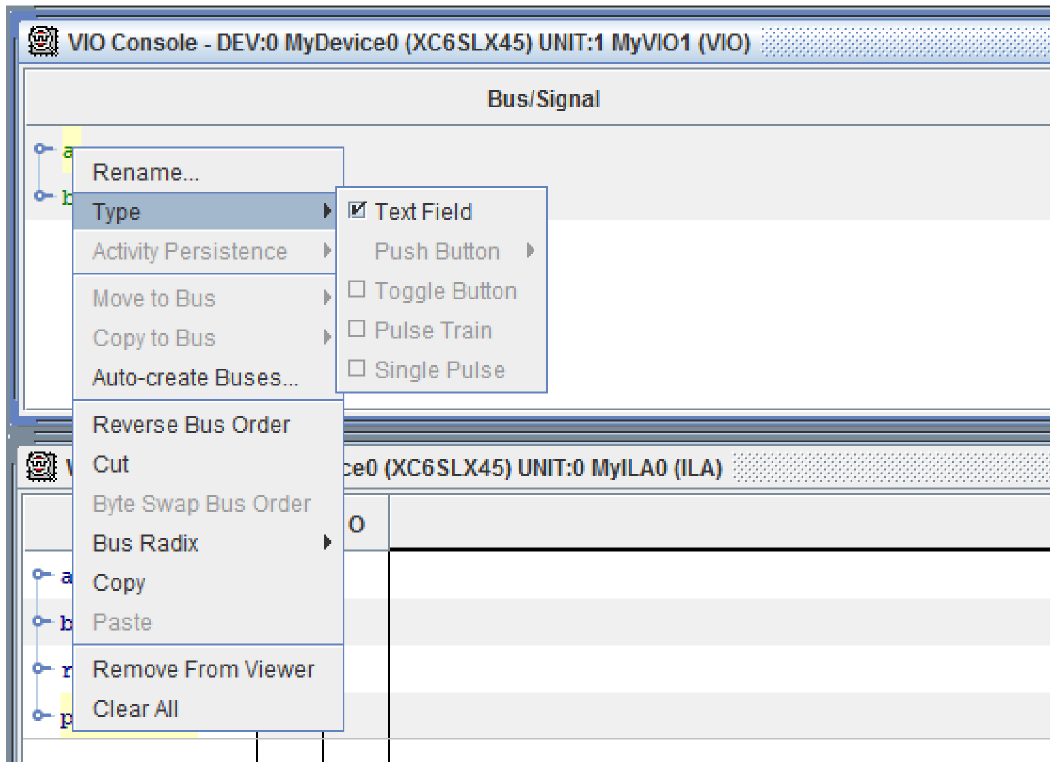


Figura 48: opciones de configuración de las señales del módulo VIO

El núcleo VIO permite la generación de un tren de 16 pulsos síncrono con la señal de reloj seleccionando la opción 'Pulse Train'. Es posible configurar el valor de cada pulso generado. Sin embargo, una vez finalizados los 16 pulsos si se desea volver a reproducir el tren es necesario volver a lanzarlo. Es posible generar un único pulso en la salida con la opción 'Single Pulse'.

El núcleo VIO ofrece la opción de actualizar los valores de las salidas, reestablecerlos a su valor original o limpiar todo tipo de actividad realizada anteriormente. Para la realización de estas tareas se debe ir a la pestaña de VIO tal y como se muestra en la Figura 49.

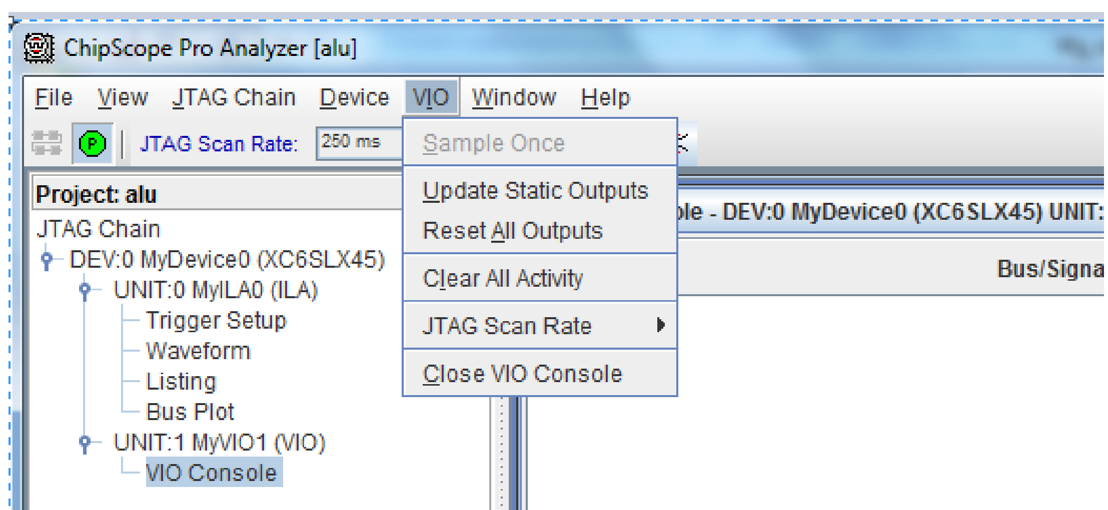


Figura 49: opciones ofrecidas por el núcleo VIO

Las entradas síncronas tienen la opción de capturar transiciones de la señal entre cada muestra tomada. Esto es útil para observar si existen glitches en dicha señal. También se pueden configurar como LEDs de indicación de actividad. En el ejemplo que se está realizando no se han utilizado entradas por lo que esta función se mostrará en otros ejemplos realizados a lo largo del trabajo de fin de grado.

Tras la configuración de las entradas y salidas, para analizar el comportamiento del resto de señales hay que lanzar el analizador lógico de ILA, donde se mostrará la respuesta del diseño a las entradas o salidas simuladas. En la Figura 50 se puede observar la herramienta funcionando de forma que se indique en las salidas del núcleo VIO el valor de las señales de entrada del módulo a depurar y el comportamiento de éste en el núcleo ILA como resultado.

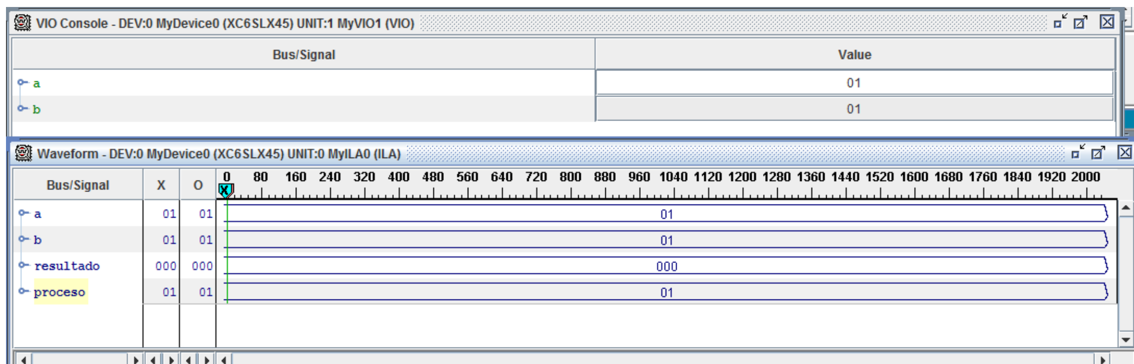


Figura 50: resultado de la depuración del módulo ALU

2.4. ATC2

Agilent Trace Core 2 (ATC2) es un núcleo personalizable que permite la depuración de capturas. Está diseñado para trabajar con las últimas generaciones de analizadores lógicos de Agilent. Este núcleo provee el acceso al analizador lógico externo a redes internas del diseño de la FPGA.

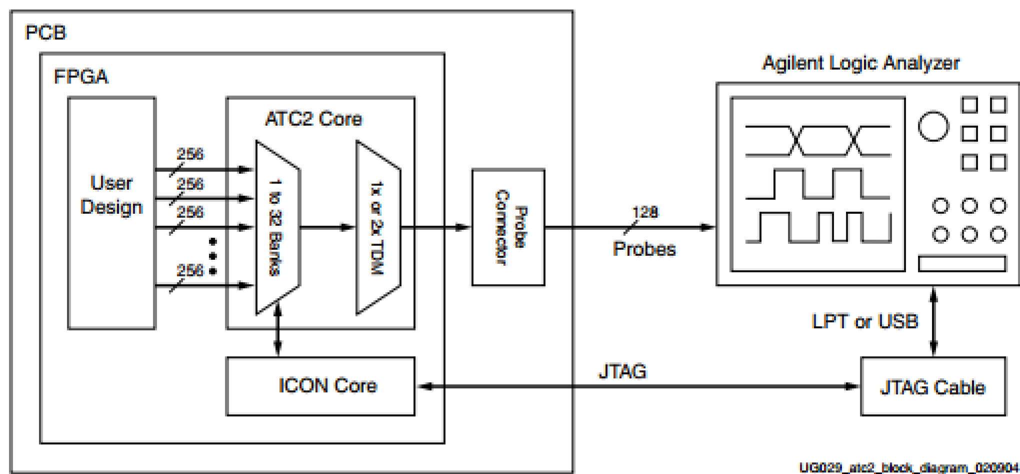


Figura 51: diagrama de bloques del núcleo ATC2

Al igual que el reto de los núcleos disponibles en ChipScope, estos se comunican a través del puerto JTAG usando el núcleo ICON (Figura 52)

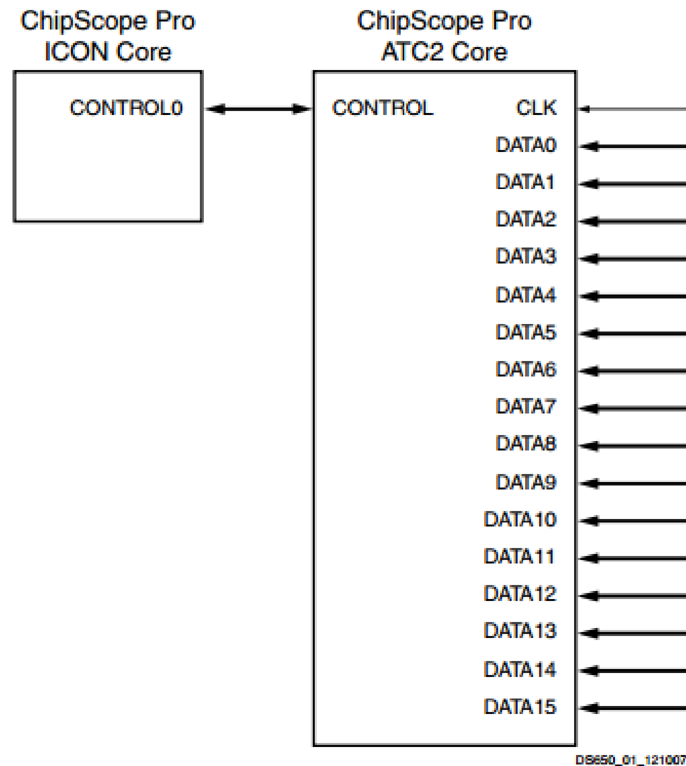


Figura 52: comunicación con el núcleo ICON.

Los datos permitidos por este núcleo consisten en hasta un total de 64 bancos de señales de entrada de tiempo real conectadas al diseño, hasta 128 pines de datos de salida conectados a la sonda del analizador lógico de Agilent. Soporta sincronización tanto síncrona como asíncrona en la captura. Soporta cualquier entrada o salida estándar valida y sondas pertenecientes a tecnología Agilent.

Tambien dispone de TDM opcional en cada pin de salida que permite doblar el tamaño de cada banco de señales de 128 a 256 bits.

Como no se dispone de analizadores lógicos de Agilent para la realización de este proyecto de fin de grado, no es posible analizar el uso real de esta herramienta disponible con ChipScope.

CAPITULO 3:

Core Inserter

Una manera más fácil y sencilla de insertar los módulos de ChipScope sin necesidad de instanciarlos después en el código es haciendo uso del Core Inserter. Esta es una herramienta post-síntesis que genera una netlist incluyendo los parámetros del diseño y de los componentes del ChipScope que se hayan insertado. Sin embargo no soporta los núcleos VIO e IBERT, debiendo incorporarlos a través de otro camino.

El Core Inserter hace uso del fichero *'definition and connection'* (.cdc) para introducir los módulos de ChipScope en el diseño, de manera que es necesario generar e introducir este fichero en el proyecto, el cual se podrá modificar en cualquier momento. Cuando el nivel superior del diseño se implemente los módulos serán insertados en la netlist de manera automática sin necesidad de establecer ninguna propiedad al proyecto.

Las conexiones de los núcleos a las señales que se deseen analizar se harán a través de esta herramienta, en un único fichero, sin necesidad de añadir cada núcleo de forma individual. De esta manera primero se insertará el núcleo ICON y a continuación los núcleos ILA que se deseen.

Para hacer uso de la herramienta Core Inserter se debe añadir un nuevo fichero en 'new source' seleccionando ChipScope Definition and Connection File (Figura 53).

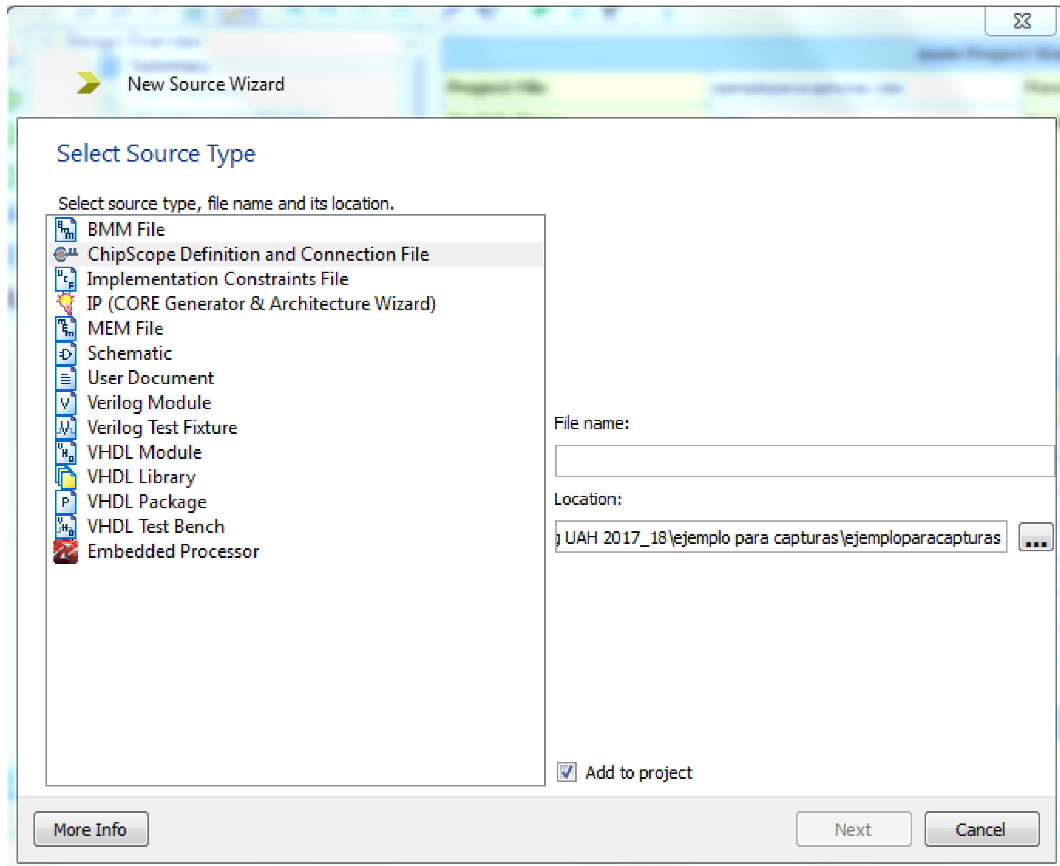


Figura 53: selección de Core Inserter para el uso de la herramienta ChipScope

Como resultado se añadirá un fichero al diseño tal y como se muestra en la Figura 54. Haciendo doble clic en dicho fichero se abrirá una pestaña en la cual configurar los núcleos de ChipScope.

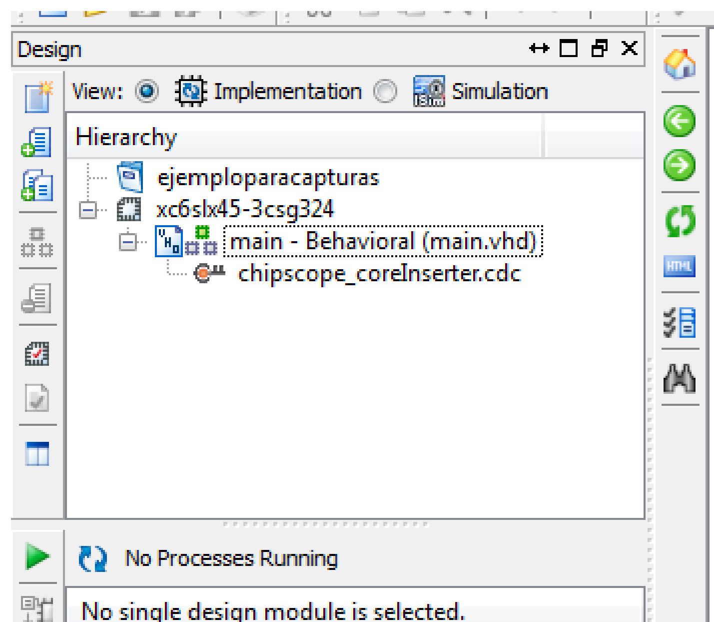


Figura 54: fichero de Core Inserter de la herramienta ChipScope

Para mostrar el uso de la herramienta Core Inserter se va a realizar un sencillo ejemplo de un registro de desplazamiento cuyo diagrama de bloque se muerta en la Figura 55. El proyecto consistirá en un registro de 5 bits el cual se irá desplazando a la derecha, completándose con el valor del puerto de entrada ***data_in***.

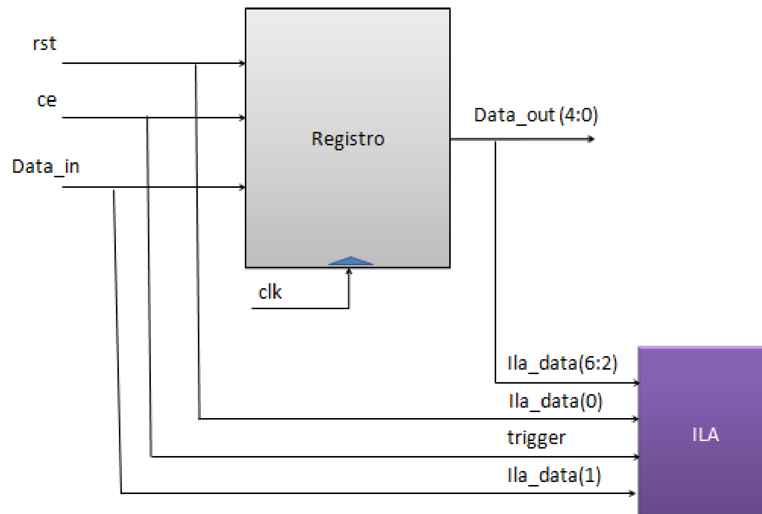


Figura 55: diagrama de bloques del registro

En primer lugar, se debe configurar el núcleo ICON en la pantalla mostrada en la Figura 56. El único parámetro a modificar es el uso del Boundary Scan. Las señales de control se añadirán automáticamente a medida que se vayan insertando núcleos ILA en el diseño.

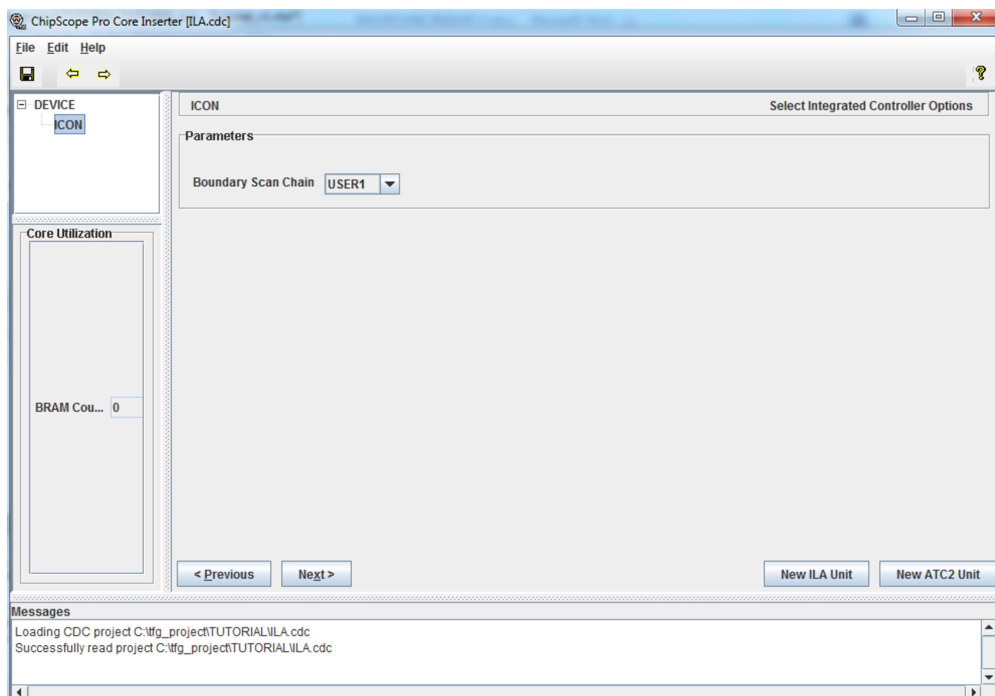


Figura 56: configuración del núcleo ICON

El siguiente paso es la configuración del núcleo ILA. Core Inserter hace uso de dos pantallas para la configuración del núcleo ILA. Una pantalla donde se configuran los triggers (Figura 57) y otra en la cual se elegirán las opciones del bus de datos (Figura 58).

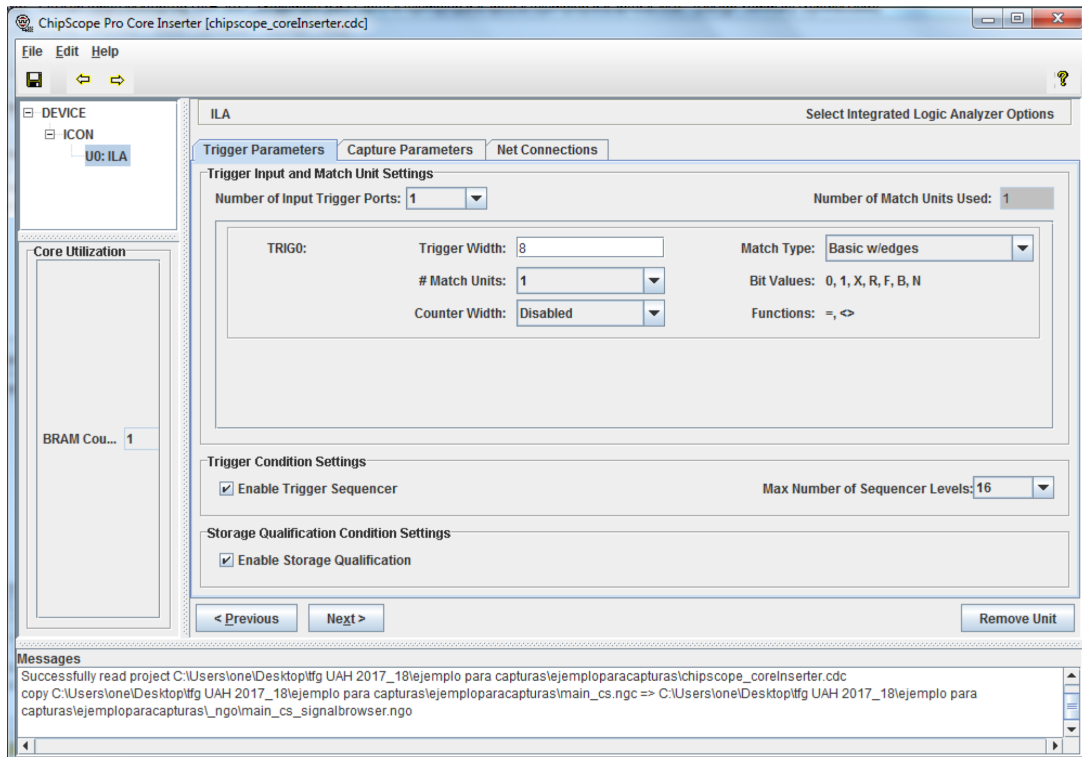


Figura 57: configuración de los triggers del núcleo ILA

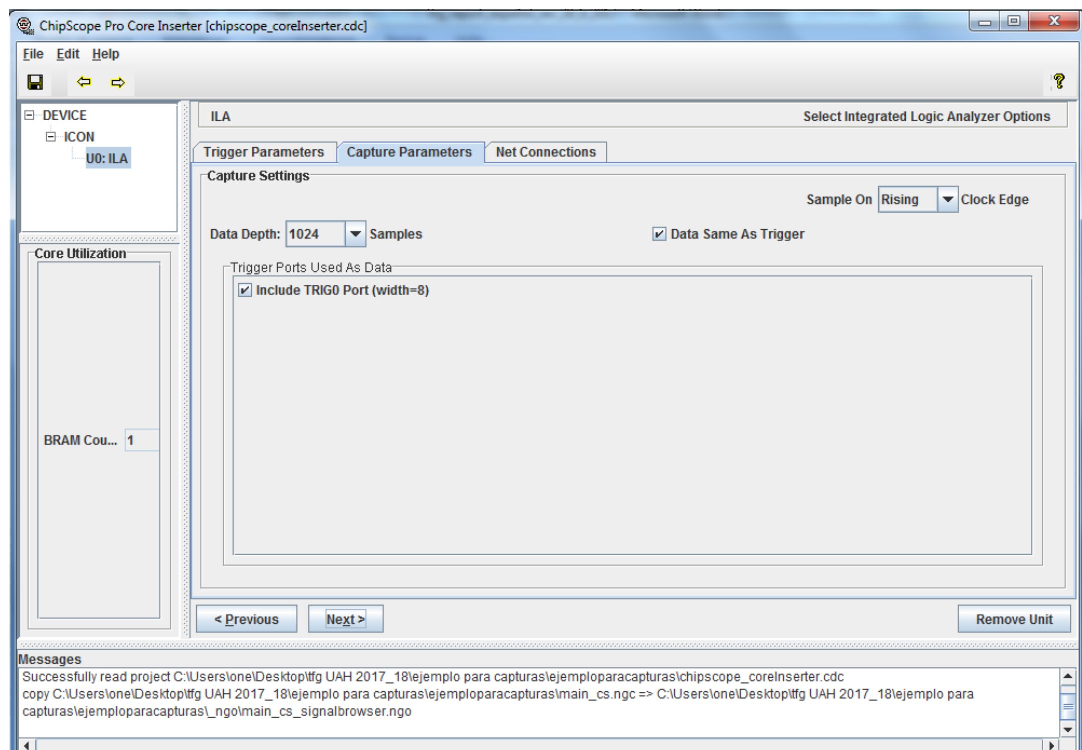


Figura 58: configuración del bus de datos del núcleo ILA

Como último paso se deben conectar las señales del diseño con los núcleos de la herramienta ChipScope insertados. Esto se realizará en la pantalla *Net Conections* mostrada en la Figura 59.

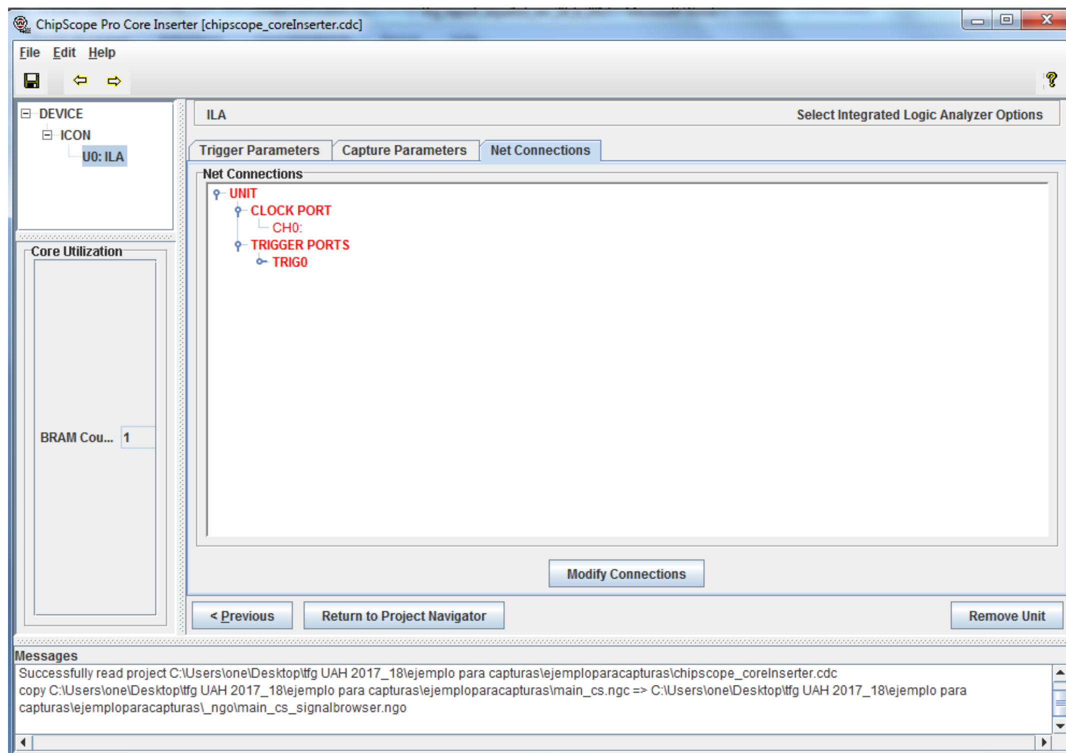


Figura 59: configuración ILA

Para realizar la conexión se debe hacer clic en 'modify connections' para añadir las señales internas del diseño que se conectarán al núcleo ILA. La pantalla resultante será la mostrada en la Figura 60 en la parte de la derecha se muestran las señales de los núcleos a conectar como son la señal de reloj, los bits del trigger y los bits del bus de datos. Estos se deben enlazar con las señales del diseño, las cuales aparecerán en la parte inferior de la pantalla. Se selecciona el bit de la señal del diseño en la parte inferior con el bit del bus de datos al cual se desea conectar y se hace clic en 'make connections'. Una vez se tienen todas las conexiones hechas se presiona 'ok'.

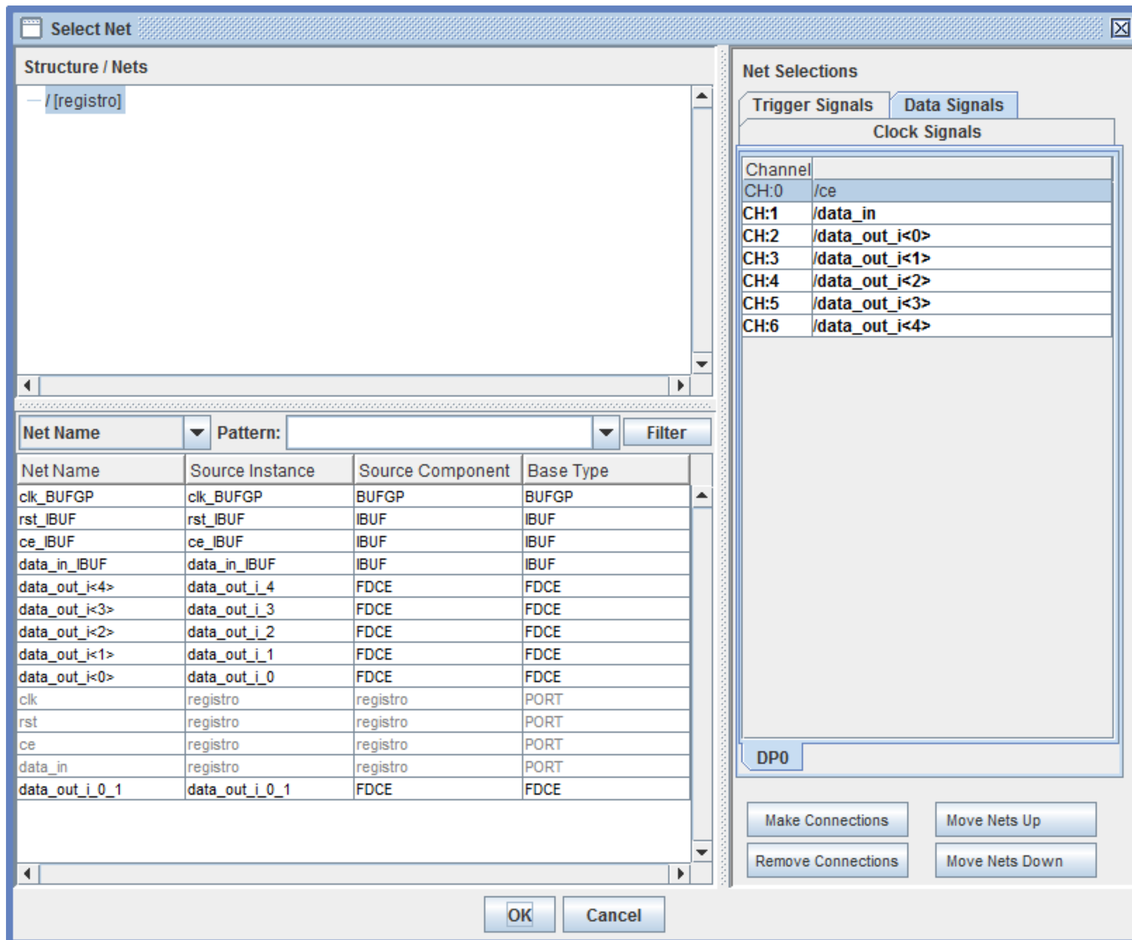


Figura 60: selección de las señales a conectar en el núcleo ILA

Una vez llegados a este punto, el procedimiento para la depuración del sistema es el mismo que al emplear Core Generation. Al cliquear en el analizador lógico de ChipScope aparecerá la pantalla de inicio en la cual se selecciona el dispositivo a programar y, posteriormente, la configuración de la herramienta. Para este ejemplo se ha empleado la configuración del puerto trigger mostrada en la Figura 61, donde la señal de disparo dará inicio al producirse un flanco de subida de la señal ***ce.***

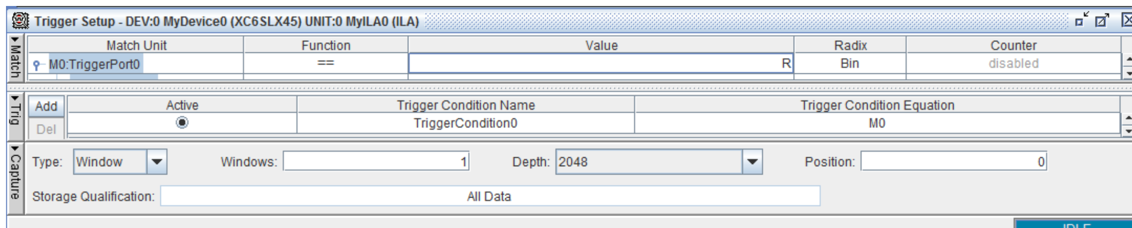


Figura 61: configuración del puerto trigger

Como resultado de la depuración se ha obtenido la Figura 62, que muestra el funcionamiento del registro de desplazamiento.

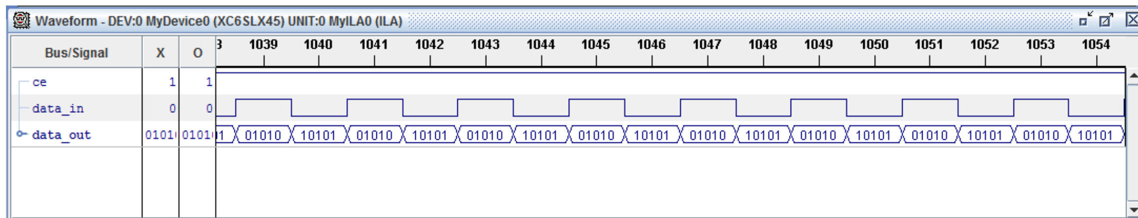


Figura 62: resultado del módulo en el analizador del módulo ILA

CAPITULO 4:

Sistema a depurar – Semáforo

Ahora que se tienen claros los conceptos principales de la herramienta es necesario ponerlos a prueba en un primer diseño sencillo: un semáforo. Este ejemplo ayudará a entender el uso de la herramienta ChipScope para la depuración de diseños. El ejemplo contendrá un error el cual se descubrirá y corregirá gracias al uso de ChipScope.

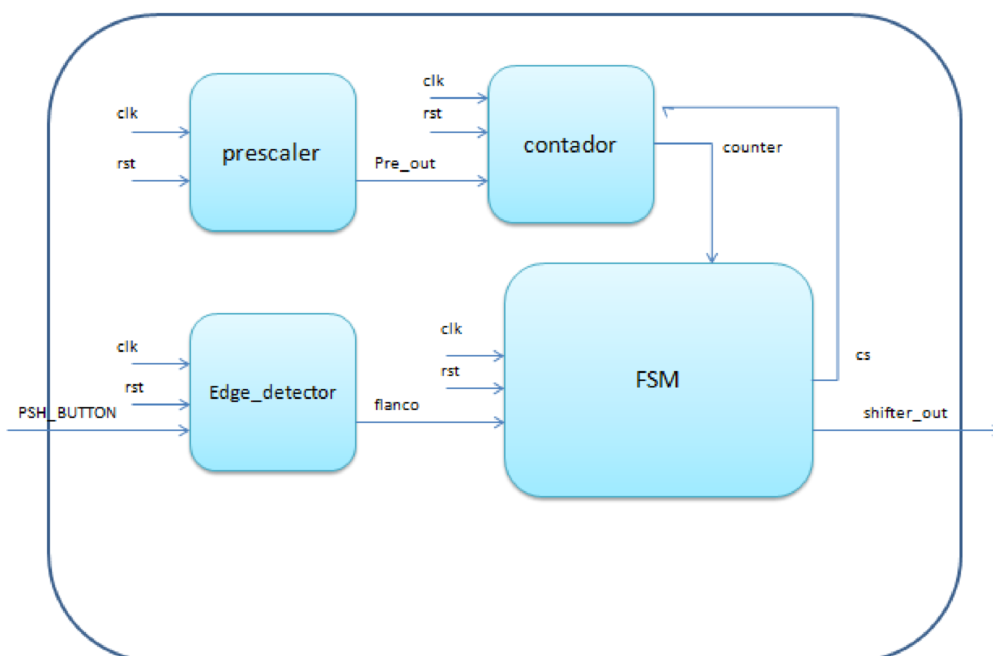


Figura 63: diagrama de boques del semáforo

En la Figura 63 se muestra el diagrama de bloques del diseño. El semáforo se implementará a partir de una pequeña maquina de estados, un contador y un codificador. El contador junto con el decodificador actuará como secuenciador ofreciendo las diferentes combinaciones del semáforo. Es sistema a diseñar será un semáforo típico situado en una rotonda, se partirá de un estado de espera donde el semáforo de automóviles esté en verde y el de peatones en rojo hasta que un pulsador sea presionado, momento en el que se cambiará a un estado activo en el cual se pondrá en marcha el secuenciador cambiando los semáforos haciendo que se permita el paso de los peatones. La máquina de estados se representa en la Figura 64 junto con la secuencia del semáforo.

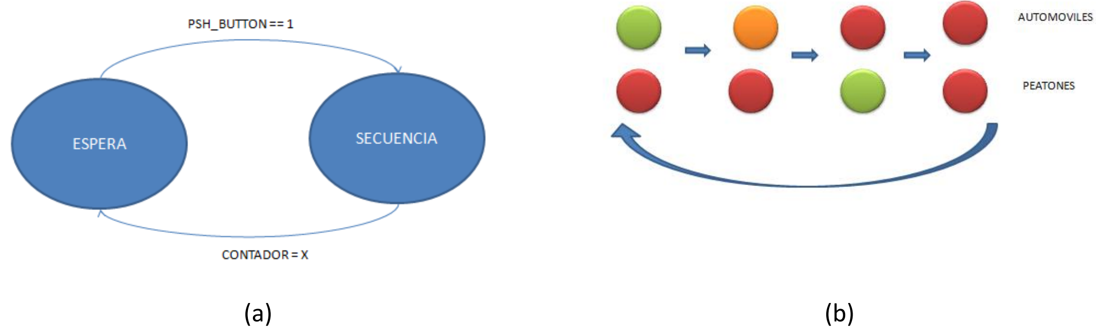


Figura 64: a) máquina del sistema a diseñar. b) Secuencia a seguir por el semáforo.

Se utilizarán los LEDs de la tarjeta Atlys para simular el sistema de manera que de los 7 LEDs que posee se utilicen tres para el semáforo de automóviles y dos para el semáforo de peatones de la siguiente manera: 0GRA/00GR (automóviles/peatones). Traducido a la FPGA los LEDs empleados para los coches son ***shifter_out(6)=***ROJO ***{P16}, shifter_out(5)=***AMBAR ***{D4}*** y ***shifter_out(4)=***VERDE ***{M13};*** mientras que para los peatones son utilizados ***shifter_out(1)=***ROJO ***{M14}*** y ***shifter_out(0)=***VERDE ***{U18}.***

Tras diseñar el sistema, cuyo código se muestra en el Anexo II (Semáforo), se obtiene la siguiente simulación del sistema mostrada en la Figura 65 que, como se puede ver, funciona de forma correcta. Tras programar la FPGA para reproducir el comportamiento del diseño se llega a un funcionamiento erroneo aunque se haya obtenido una simulación correcta. Esto es común en el diseño de sistemas a partir de lenguajes de programación debido a que la simulación ofrece el comportamiento basado en los conocimientos que el entorno de desarrollo posea sin embargo no es el funcionamiento real sino aproximado. En la implementación en la placa intervienen componentes reales como son el mapeo y enrutado en el dispositivo, el cual puede afectar a los tiempos en la propagación de las señales y ocasionar un mal funcionamiento.

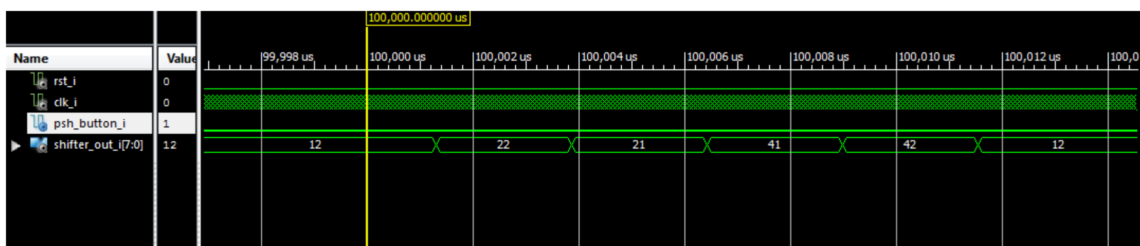


Figura 65: simulación del diseño

Para encontrar algún posible error en el diseño se va a emplear la herramienta ChipScope Pro, en concreto el núcleo ILA que crea un analizador lógico en el cual poder ver las señales internas que intervienen en el diseño. La inserción de los núcleos de ChipScope se encuentra en el Anexo II (Semáforo). Como primer paso se ha introducido ChipScope en el módulo principal y se ha obtenido el siguiente comportamiento.

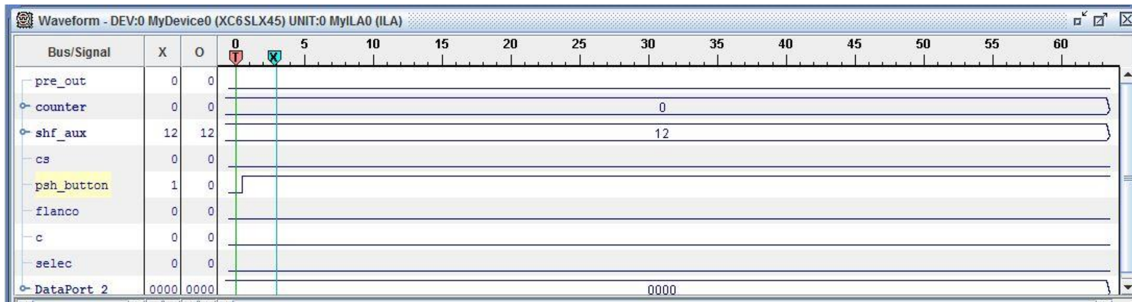


Figura 66: señales internas del diseño capturadas con el analizador lógico de ChipScope

Tal como se puede apreciar en la Figura 66 se comprueba cómo no se han generado las señales ***flanco*** y ***cs*** causadas por la máquina de estados, ni ***pre_out*** causada por el prescaler, señales responsables de habilitar el contador y poder realizar la secuencia del semáforo. Para poder identificar mejor el causante del problema se ha decidido por realizar un análisis de cada módulo por separado, por lo que se ha realizado una entidad separada para cada proceso que interviene en el diseño y se ha analizado a través de la herramienta ChipScope (Figura 67).

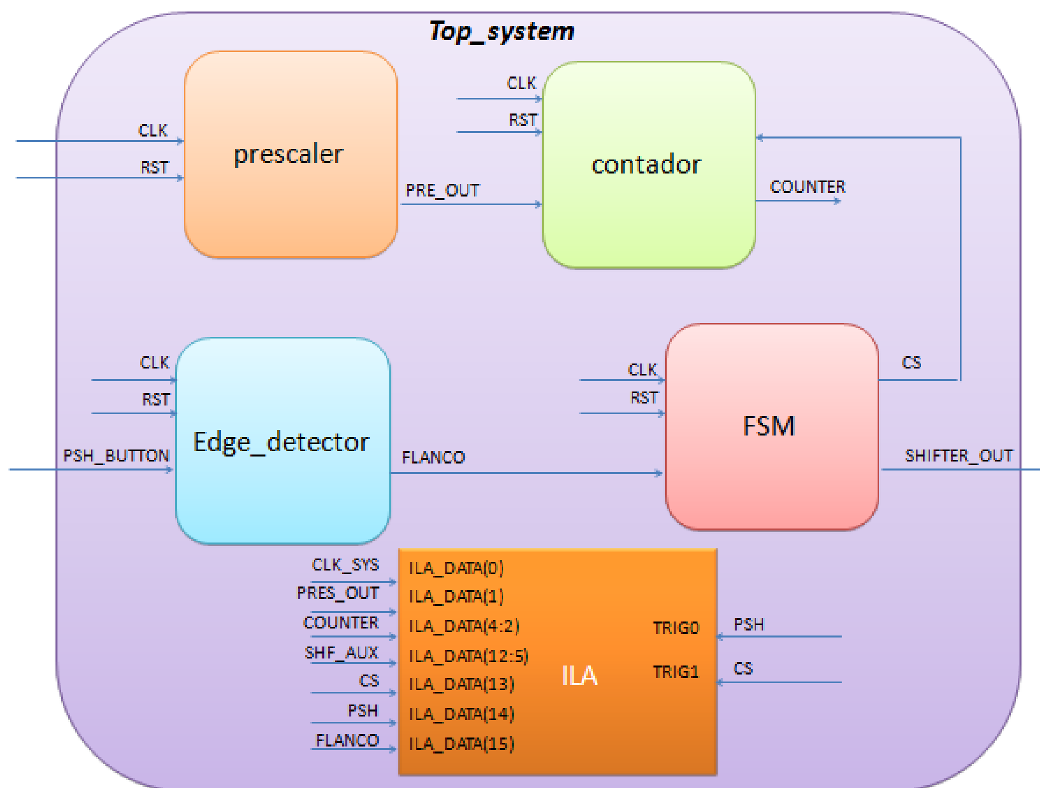


Figura 67: sistema a depurar con ChipScope

Primero se procederá a la comprobación del módulo prescaler, para ello se deben insertar los núcleos de ChipScope dentro de la entidad prescaler. En una primera prueba observamos como el sistema sigue sin funcionar y como se obtiene una señal de salida constante a nivel alto y el contador permanece a cero (Figura 68). Debido a que se obtienen muestras en el analizador lógico se puede descartar un fallo en la señal de reloj, esto es debido a que se trabaja con la misma señal de reloj tanto en el sistema como en el analizador.

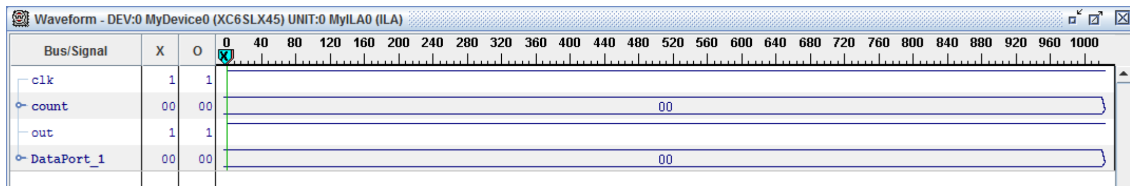


Figura 68: resultado del test del módulo prescaler

Por lo tanto solo se tiene la posibilidad de que haya un error con la señal de ***reset*** de modo que se ha decidido por hacer un cambio en el código tal y como se muestra en la Figura 69, cambiando la señal de ***reset*** para que actúe a nivel bajo. El resultado obtenido tras esta prueba se puede comprobar en la Figura 70, donde tras el cambio se ha obtenido el funcionamiento deseado. Hay que tener cuidado a la hora de asignar las señales de entrada de la placa en el fichero .ucf, como son los botones, ya que pueden causar este tipo de fallos si no se tiene en cuenta el nivel con el que actúan.

```
---------------------------- prescaler--------------------------
----------------------------------------------------------------
 process (clk,rst)
  begin   -- process
    if rst = '0' then
      counter_reg    <= 0;
    elsif clk'event and clk = '1' then
      if counter_reg = CLKDIV-1 then
        counter_reg <= 0;
      else
        counter_reg <= counter_reg+1;
      end if;
    end if;
  end process;

    pre_out   <= '1' when  counter_reg = CLKDIV-1 else '0';
```

Figura 69: nuevo código a analizar

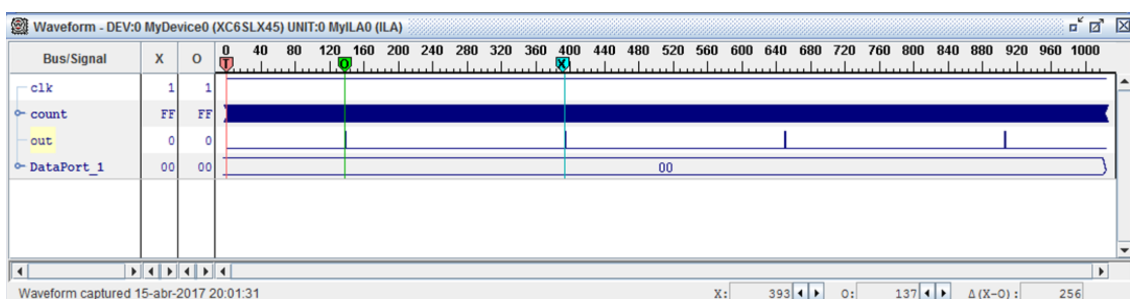


Figura 70: módulo prescaler con rst=0

Basado en estos resultados se ha cambiado la señal ***reset*** de todo el diseño y se han comprobado los resultados. Sin embargo, el diseño continúa teniendo un funcionamiento erróneo, observando como los LEDs permanecen sin cambios después de un flanco de subida en la señal ***PSH_BUTTON.***

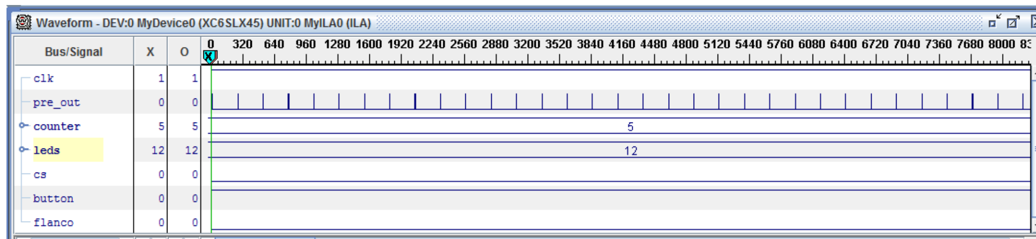


Figura 71: sistema tras el cambio en la señal reset

Ahora se comprueba el contador. Tras el reset, el sistema funciona correctamente (Figura 72), sin embargo el contador no regresa a 0 por lo que en la siguiente activación de la señal ***PSH_BUTTON*** la señal ***cs*** solo dura un periodo de reloj (Figura 73).

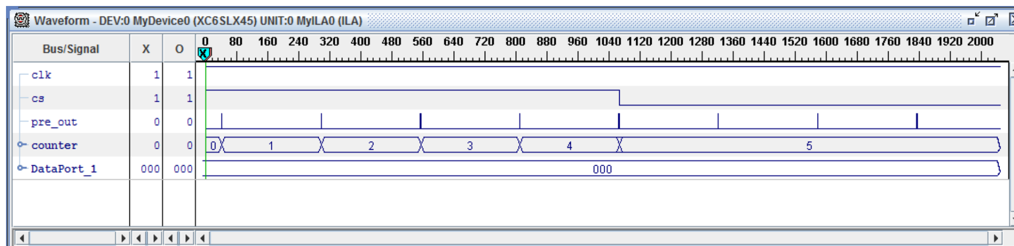


Figura 72: contador tras el reset.

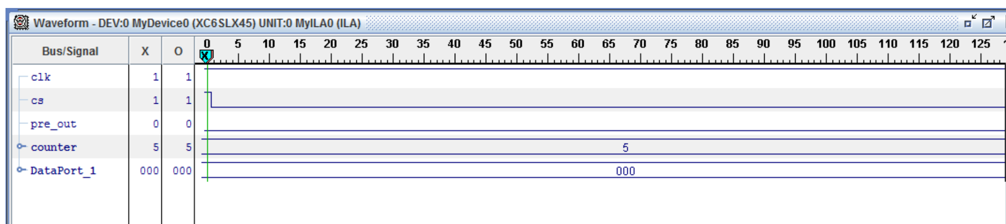


Figura 73: Contador en las pulsaciones consecutivas

Esto provoca un funcionamiento incorrecto del sistema que no se había mostrado en las simulaciones anteriormente realizadas. Tras analizar el problema del contador con detenimiento observando los resultados obtenidos, se ha modificado el código, el cual se muestra en la Figura 74, de manera que se consigue el resultado mostrado en la Figura 75.

```
process (clk, rst)

begin  -- process

    if rst = '0' then
     counter      <= (others => '0');
   elsif (clk'event and clk = '1') then
    if cs = '1' then  --si se ha apretado el boton
     if pre_out = '1' then
         counter <= counter+1;
----------------------------------------------------------------------
-- codigo modificado tras la depuracion del diseño con ChipScope
----------------------------------------------------------------------
--        if counter = "101" then
--           counter <= (others => '0');
--        end if;
----------------------------------------------------------------------
     end if;
    else
     counter <= (others => '0');
    end if;
   end if;
```

Figura 74: cambio del codigo en el módulo contador

Figura 75: resultado deseado del contador

Analizando el diseño completo comprobamos su correcto funcionamiento en la Figura 76. Sin embargo, el tiempo empleado en los cambios del semáforo es demasiado rápido para poder ser apreciado a simple vista, por esa razón se ha optado por un cambio de tiempo a 0.50s.

Figura 76: funcionamiento final del sistema

Ahora bien el resultado final es el correcto, pudiendo comprobar a través de los LEDs de la placa su funcionamiento, aunque haciendo este cambio en los tiempos no es posible observarlo a partir de la herramienta ChipScope sin tener que modificar el top_system de manera que se inserte un prescaler el cual nos dé un periodo de muestreo para el núcleo ILA mayor.

CAPITULO 5:

Sistema a depurar – Reproductor de partituras

Para la resolución del estudio de la herramienta ChipScope Pro de Xilinx se procederá a la realización de un sistema basado en la tarjeta Atlys de Digilent en una FPGA Spartan 6, de la cual se hará uso del códec de audio LM4550 que incorpora, de manera que el diseño permita realizar las pruebas necesarias para la comprobación del funcionamiento de dicha herramienta. El sistema de dividirá en los distintos módulos, creando asi diferentes proyectos que se adaptarán de forma que se logre una mejor visión de las funcionalidades que presenta la herramienta ChipScope.

El diseño consiste en el modelado en VHDL de un sistema digital que genere una señal de audio con el códec LM4550. El audio a reproducir se corresponde con una partitura musical, generada a través de la herramienta Matlab, donde cada nota es conseguida a partir de una señal sinusoidal con una determinada frecuencia y duración. Además el diseño debe permitir la selección del volumen de la señal de audio. Los datos necesarios para la generación de la señal de audio se transmitirán al dispositivo FPGA a través del puerto USB del ordenador haciendo uso del protocolo EPP (Enhanced Parallel Port).

La información que se introducirá al diseño seguirán los campos de dirección y de dato dados en la Figura 77.

Dirección	Dato	Función
0x11	0x11	Inicialización del códec LM4550 (RESTART)
0x22	0x22	Iniciar la carga de las notas (START)
0xAA	0xAA	Reproducción de las notas cargadas (PLAY)
0xF0	0x00-0x5A	Frecuencia de la nota
0xD0	0x00-0x07	Duración de la nota
0xB0	0x00-0x1F	Volumen de la señal de audio

Figura 77: direcciones y datos a utilizar por la aplicación

La frecuencia y la duración de las notas enviada a través de Matlab serán almacenadas en sendas FIFOs para su posterior lectura, donde será enviada al códec de audio de la FPGA. La tarjeta Atlys dispone del driver CY7C68013A que genera protocolo EPP, convirtiendo los datos recibidos por un puerto USB a un formato paralelo. El diseño a realizar hará uso de una señal de reloj de frecuencia 100MHz generada por un oscilador existente en la placa Atlys.

El sistema a diseñar se dividirá en varias entidades de modo que sea más sencillo su posterior depuración e inclusión de la herramienta ChipScope Pro. La Figura 78 contiene el diagrama de bloques del diseño completo dividido por módulos.

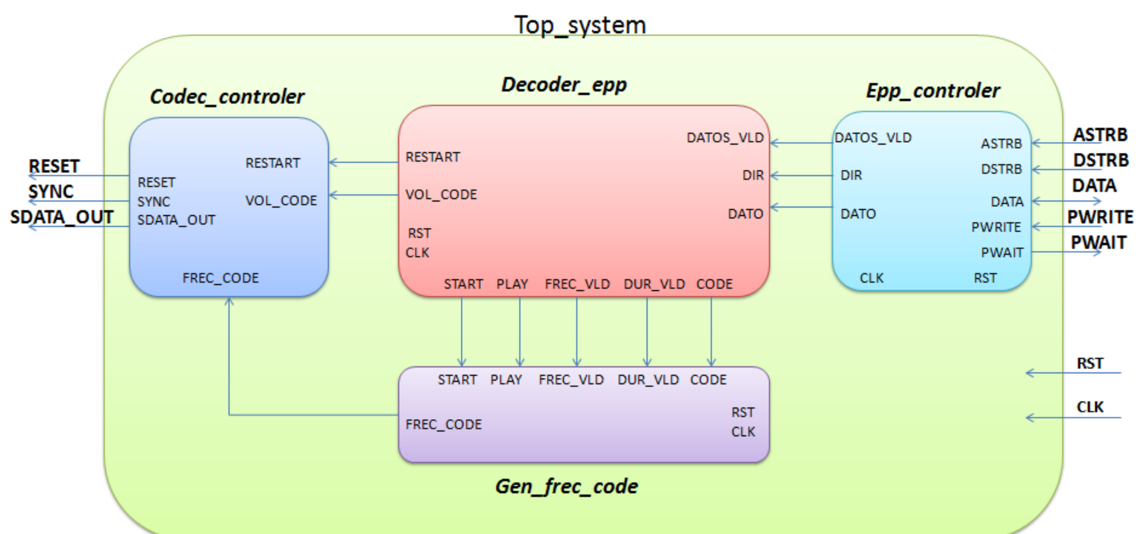


Figura 78: estructura del diseño

A través del módulo ***epp_controller*** se recibirán los datos enviados por el protocolo EPP y se obtendrán la dirección y el dato contenidos en la trama. La dirección y dato pasarán al módulo ***decoder_epp*** que decodificará la información contenida en dichas tramas, obteniendo así los valores de cada variable a configurar posteriormente. La información correspondiente a la frecuencia y duración de las notas pasará al módulo ***gen_freq_code***. Este módulo guarda en una memoria FIFO los valores de frecuencia y su corresponciente duración generando así un código. A continuación el módulo ***códec_controller*** recibirá el código de frecuencia y de volumen proporcionados por las entidades ***gen_frec_code*** y ***decoder_epp*** y se encargará de encapsular los datos del seno de audio acorde con la información recibida para luego configurar el códec de audio de la placa.

En los siguientes apartados se describirá cada módulo en detalle, realizando el diseño y la implementación de cada uno de ellos de manera individual.

5.1. Epp_controller

El protocolo EPP (Enhanced Parallel Port) fue desarrollado para proporcionar un puerto paralelo de alto rendimiento que fuera compatible con un puerto paralelo estándar. Dicho puerto usa las señales recogidas en la Figura 79 para realizar la comunicación periférico-host.

Señal	*Dirección*	*Función*
PWRITE	Out	activa a nivel bajo indica operación de escritura
ASTRB	Out	Activa a nivel bajo indica transferencia de dirección
DSTRB	Out	Activa a nivel bajo indica transferencia de dato
PWAIT	In	A nivel alto indica que es posible iniciar la transferencia (assert a strobe), mientras que el nivel bajo indica que es posible finalizar la transferencia (de-assert a strobe)
DATO	Bidireccional	Bus de datos de 8bits
RESET	Out	Activa a nivel bajo el reset del periférico
INTR	In	Interrupción del periférico. Permite generar una interrupción en el host

Figura 79: señales que intervienen en el protocolo EPP

Gracias al protocolo EPP se puede lograr velocidades de transferencia de datos de unas 500KBps hasta los 2MBps, adaptando esta velocidad al interfaz, adaptador de host o dispositivo periférico más lento. La "adaptación de velocidad" es posible gracias a la señal ***PWAIT*** la cual permite prolongar los ciclos de acceso.

La entidad ***epp_controller*** del sistema es la encargada de recibir los datos establecidos en el protocolo EPP y devolver los campos de dato y dirección por puertos diferentes, proporcionando al mismo tiempo una señal de validación que indique a nivel alto cuando la

dirección y el dato han sido transferidos. En la Figura 80 se oberva la entiad ***epp_controller*** del sistema.

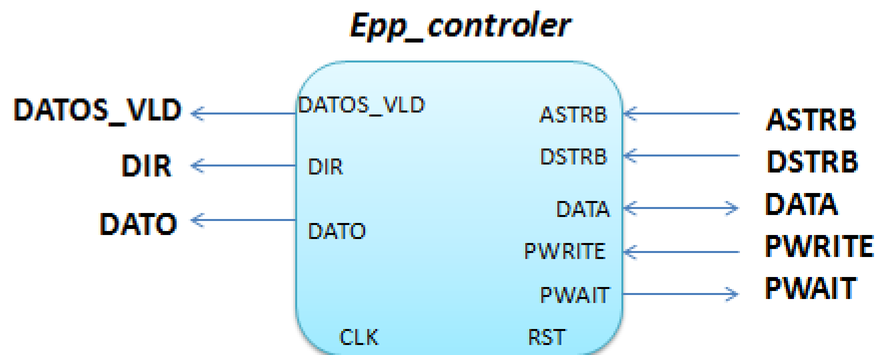


Figura 80: entidad epp_controller

En la Figura 81 se describe un ciclo de escritura del protocolo EPP en el cual se puede ver como primero se recibe la dirección cuando la señal ***ASTRB*** está activa y seguidamente se recibe el dato cuando la señal ***DSTRB*** está activa. De este modo es necesario fijarse en las señales ***ASTRB*** y ***DSTRB***, las cuales indican transferencia de dirección y de dato, para obtener la dirección y el dato recibidos. Estas señales son las causantes de generar la señal ***PWAIT***, señal que será diseñada en las siguientes etapas.

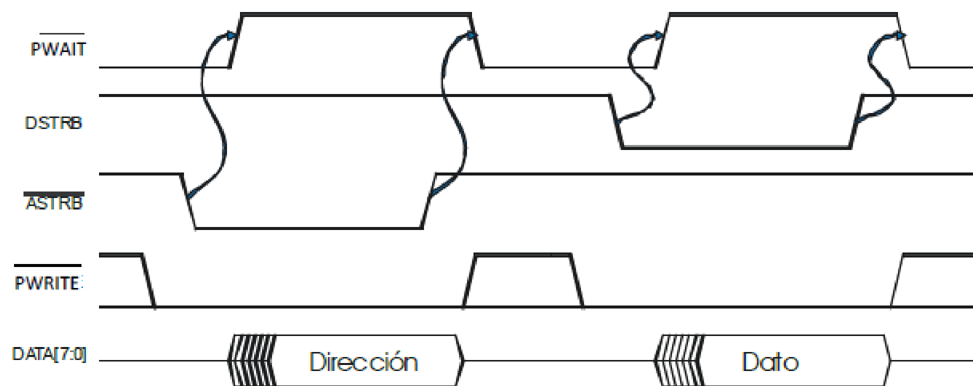


Figura 81: ciclo de escritura

La señal ***DIR_VLD*** será la encargada de validar la dirección recibida. Esta se activa a nivel alto durante un periodo de la señal de reloj cuando ha finalizado la transmisión de la dirección. Funcionamiento similar tiene la señal ***DATOS_VLD***, utilizada para validar el dato recibido.

A continuación se va a proceder a realizar la descripción de los puertos involucrados en la entidad ***epp_controller***:

- **ASTRB:** indica con un nivel bajo que se está transmitiendo la dirección.
- **DSTRB:** indica con un nivel bajo que se están transmitiendo los datos.

- **PWRITE:** indica con un nivel bajo que se está realizando la escritura de datos mientras que el nivel alto indica lectura de datos.
- **PWAIT:** permite prolongar los ciclos de acceso insertando estados de espera.
- **DATA:** bus de datos para transmitir tanto datos como direcciones.
- **DATOS_VLD:** indica con un pulso a nivel alto de duración un periodo de señal de reloj que se ha realizado la transferencia de datos, esto es que se ha realizado un ciclo de escritura primero escribiendo la dirección y a continuación escribiendo el dato.
- **DIR:** contiene la dirección transmitida.
- **DATO:** contiene el dato transmitido.

En el sistema solo se emplea el ciclo de escritura, de modo que para el análisis esta entidad se ha modificado para que sea posible visualizar los ciclos tanto de escritura como de lectura. El nuevo diseño empleado en la depuración se observa en la Figura 82.

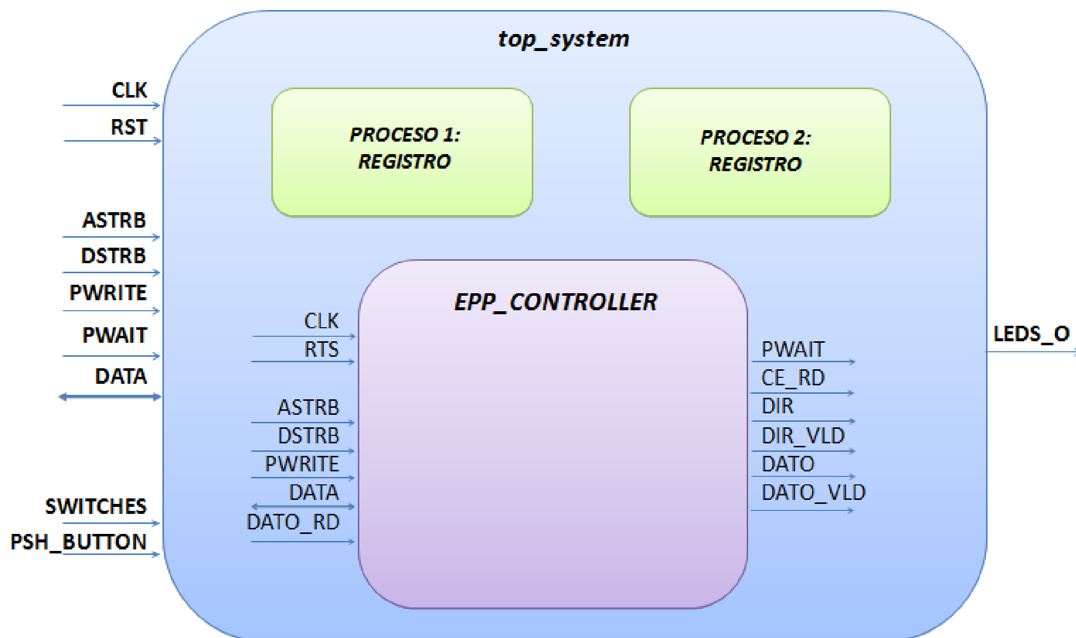


Figura 82: entidad creada para la depuración

El nuevo proyecto generado está compuesto por las señales del protocolo EPP además de una serie de switches y un botón. Cuando esté habitlitada la lectura en la dirección 0x32 se leerá en el campo ***DATO*** el valor perteneciente a los switches. Por otro lado, según la posición de uno de los botones de la placa se visualizará el dato o la dirección de la trama a través de los LEDs de la placa.

5.1.1. Diseño

Se desea obtener las señales ***PWAIT, DIR, DATO, DATOS_VLD*** y ***DIR_VLD*** obtenidas tras analizar el ciclo de escritura. Como se ha podido observa en la Figura 81 para generar dichas señales es necesario fijarse en las transiciones de las señales ***ASRTB*** y ***DSTRB*** de nivel alto a nivel bajo y viceversa. En el caso de la señal ***PWAIT*** se puede ver como se puede generar como combinación de las señales ***ASTRB*** y ***DSTRB***, de modo que permanece a nivel alto cuando estas

están activas a nivel bajo. Para ello se puede emplear una puerta lógica atendiendo a la tabla de verdad mostrada abajo.

DSTRB	ASTRB	PWAIT
0	0	**X**
0	1	**1**
1	0	**1**
1	1	**0**

Ahora bien, es necesario diferenciar los datos de las direcciones para ello se hará uso de las señales ***ASTRB*** y ***DSTRB*** de nuevo, las cuales indican cuando se está transmitiendo una dirección y cuando un dato. ***ASTRB*** a nivel bajo indica que se está transmitiendo una dirección por lo que cuando exista una transición de nivel bajo a alto esto nos indica que en el bus de datos el valor se corresponde a una dirección. De la misma manera cuando se encuentra una transición de nivel bajo a alto en la señal ***DSTRB*** indica que en el bus de datos el valor pertenece a los datos.

Para detectar estas transiciones de nivel se va a diseñar un detector de flancos de subida. En la Figura 83 se muestra el diagrama de la señal a analizar.

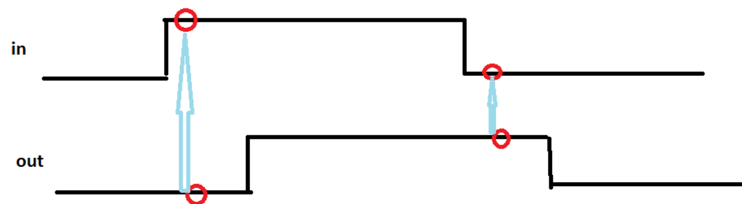


Figura 83: detección de flancos en una señal registrada

Una vez se ha detectado el flanco de subida en ***ASTRB*** y ***DSTRB*** se pueden generar las señales de validación de direcciones y datos, respectivamente.

5.2. Decoder_epp

La entidad ***decoder_epp*** actúa como decodificador de las direcciones y datos recibidos por la entidad ***epp_controller*** devolviendo las correspondientes señales de control siguiendo la tabla de configuración de la Figura 77. En la Figura 84 se muestra la entidad ***decoder_epp***.

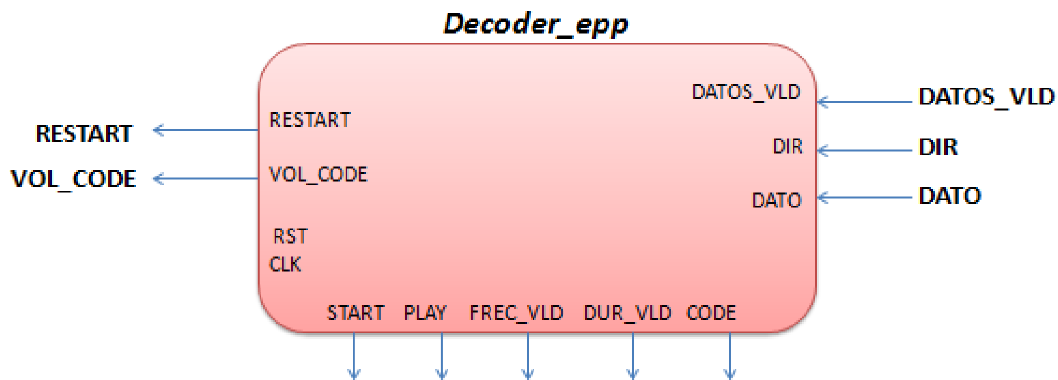


Figura 84: entidad decoder_epp

A continuación se realiza la descripción de los puertos de la entidad ***decoder_epp***:

- **RESTART:** indica con un pulso a nivel alto de duración un periodo de señal de reloj que se ha recibido la dirección 0x11 y dato 0x11.
- **START:** indica con un pulso a nivel alto de duración un periodo de señal de reloj que se ha recibido la dirección 0x22 y dato 0x22.
- **PLAY:** indica con un pulso a nivel alto de duración un periodo de señal de reloj que se ha recibido la dirección 0xAA y dato 0xAA.
- **FREC_VLD:** indica con un pulso a nivel alto de duración un periodo de señal de reloj que se ha recibido la dirección 0xF0. Representa que en el puerto CODE se encuentra un dato correspondiente a una frecuencia.
- **DUR_VLD:** indica con un pulso a nivel alto de duración un periodo de señal de reloj que se ha recibido la dirección 0xD0. Representa que en el puerto CODE se encuentra un dato correspondiente a la duración.
- **VOL_CODE:** esta señal contiene el dato correspondiente al valor del volumen a configurar. Este valor se corresponde al dato obtenido cuando se recibe la dirección 0xB0.
- **CODE:** señal que contiene el dato obtenido cuando se recibe la dirección 0x F0 o 0xD0.

5.2.1. Diseño

Para el desarrollo de la entidad ***decoder_epp*** hay que tener en cuenta que se trata de un decodificador, esto es, dependiendo de un código dado por la señal **DIR** y un código dado por la señal **DATO** se realizarán ciertas acciones. Esto se puede implementar a través del uso de la sentencia *if-else* de modo que se obtenga una salida distinta para cada combinación de códigos. El la Figura 85 se puede ver el cronograma a diseñar.

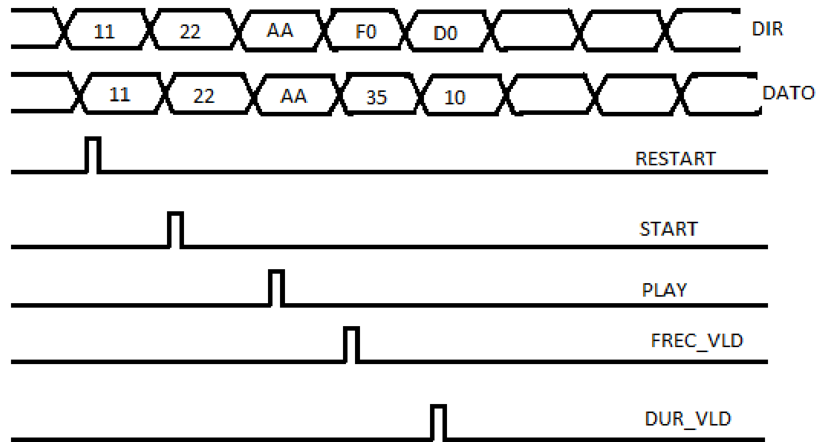


Figura 85: cronograma de la entidad decoder_epp

5.3. Gen_frec_code

La entidad ***gen_frec_code*** proporciona la frecuencia a usar para generar las notas musicales (Figura 86). El código de cada frecuencia se mantendrá en el puerto **frec_code** durante el tiempo correspondiente a la duración de la nota a reproducir. Para ello recibirá las señales de control y los datos proporcionados por la entidad ***decoder_epp***.

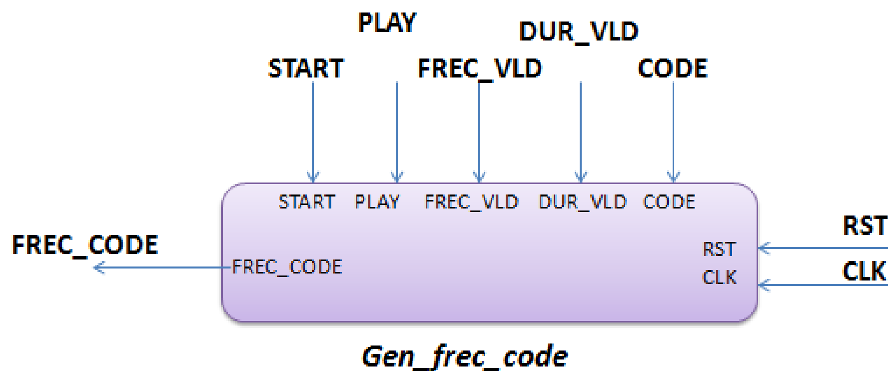


Figura 86: entidad gen_frec_code

Los datos recibidos se guardarán en dos FIFOs de tamaño 32x8 bits dependiendo de si el dato recibido se trata de frecuencias o duraciones de modo que se obtendrán dos FIFOs, una correspondiente para cada dato. La escritura de los datos en las FIFOs se realizará tras recibir el comando ***START***, por lo que primero se escriben los datos en ella y posteriormente se realiza su lectura. De esta manera nunca se producirá conflictos entre ambas operaciones de escritura y lectura.

A continuación se describirán de los puertos pertenecientes a la entidad ***gen_frec_code***:

- **START:** indica con un pulso a nivel alto el comienzo de escritura de las FIFOs.
- **PLAY:** indica con un pulso a nivel alto el comienzo de lectura de las FIFOs.
- **FREC_VLD:** indica que el código recibido pertenece a la frecuencia de las notas.

- **DUR_VLD:** indica que el código recibido pertenece a la duración de las notas.
- **CODE:** bus de datos cuyo valor corresponde a las frecuencias o duraciones de las notas, dependiendo de las señales ***frec_vld*** y ***dur_vld***.
- **FREC_CODE:** puerto de salida con el valor correspondiente al código de la frecuencia correspondiente a cada nota.

5.3.1. Diseño

En el diseño de la entidad ***gen_frec_code*** se hará uso de dos memorias FIFOs las cuales se harán cargo de guardar los valores de la frecuencia y la duración de cada nota para posteriormente ser leídas. En este documento únicamente se hará uso de una memoria FIFO para la inclusión de la herramienta ChipScope.

FIFOS

Para mantener los valores de las notas a reproducir, estos son la frecuencia y la duración, se empleará una memoria FIFO que se modelará en base a una plantilla establecida que se modificará para adaptarla a las necesidades del diseño (Figura 87).

Figura 87: entidad fifo

La entidad ***fifo*** contará con los siguientes puertos de entrada:

- **Rst_wr:** señal de entrada que indica el reset del puntero de escritura, posicionando la dirección de escritura de la memoria en la primera posición.
- **Rst_rd:** señal de entrada que indica el reset del puntero de lectura, posicionando la dirección de lectura de la memoria en la primera posición.
- **Wr_en:** señal de habilitación de escritura en la memoria FIFO.
- **Rd_en:** señal de habilitación de lectura en la memoria FIFO.
- **Din:** datos de entrada a escribir.
- **Dout:** datos de salida leídos.
- **Empty:** señal que indica a nivel alto que la memoria está vacía, esto es que se han leído todas las posiciones de memoria.
- **Full:** señal que indica a nivel alto que la memoria está llena, esto es que se han escrito todas las posiciones de memoria.

Se usarán las señales ***rd_addr*** y ***wr_addr*** para indicar la posición de memoria de lectura y escritura en la que se está leyendo o escribiendo, respectivamente. Tras realizar un reset estos punteros pasarán a la posición 0. En caso contrario, si está habilitado la escritura o lectura de datos, estos punteros incrementarán en uno por cada ciclo de la señal reloj, escribiendo o leyendo una posición cada vez, dependiendo de si se encuentra habilitada la escritura (***wr_en***=1) o la lectura (***rd_en***=1).

Una vez se hayan escrito todas las posiciones de memoria, indicadas por la señal ***wr_addr***, de la FIFO la señal ***full*** indicará con un nivel alto que no hay más espacio. En este momento se puede proceder a la reproducción de la señal de audio, esto es, leer la memoria FIFO. Tras la lectura de todas las posiciones de memoria, contempladas por a señal ***rd_addr***, el diseño indicará con un nivel alto en la señal ***empty*** que la memoria está vacía. Para el diseño de la memoria FIFO se usará una plantilla la cual se adaptará a las necesidades del diseño.

5.4. Códec_controler

Esta entidad será la encargada de proporcionar los datos necesarios para la configuración del códec de audio LM4550[2], que permitirá la generación de la señal de audio a reproducir. En el diseño del controlador del códec de audio se generará una trama la cual va a contener los valores de la señal sinusoidal transmitida anteriormente, además de los datos de volumen y frecuencia de la señal analógica reconstruida por los DACs que incorpora el circuito. En la Figura 89 se puede observar el interfaz del códec de audio LM4550 con la FPGA a utilizar, mientras que la Figura 88 muestra la entidad ***códec_controller*** que se diseñará para el control de dicho circuito integrado.

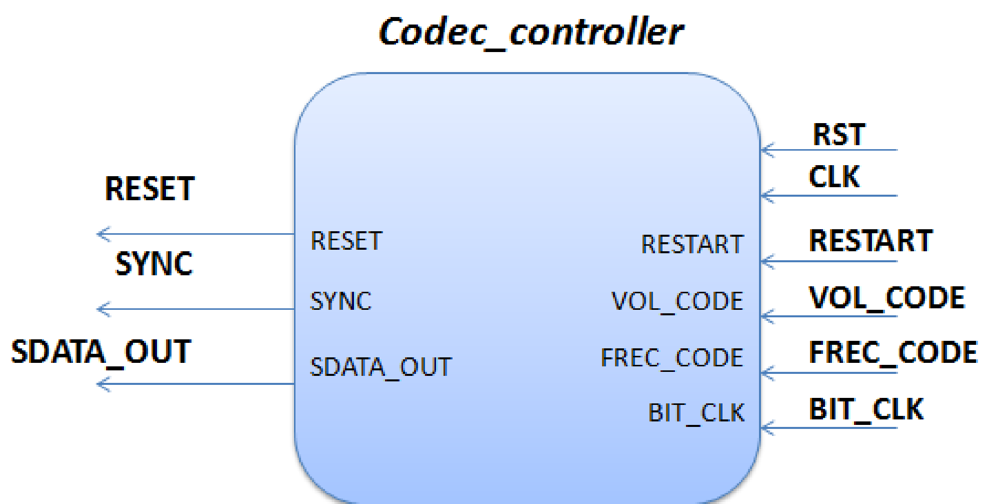


Figura 88: entidad codec_controler

[2] El anexo I proporciona la documentación del códec de audio LM4550.

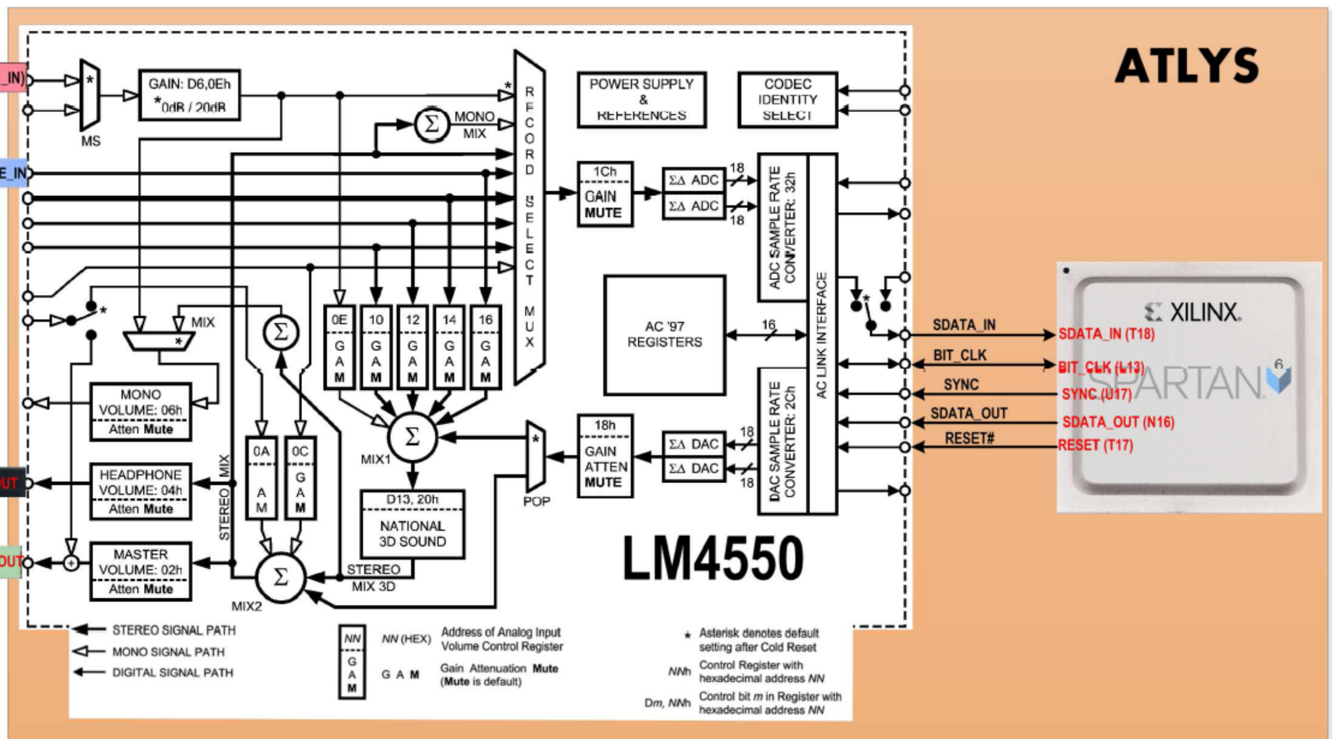


Figura 89: interfaz del códec con la FPGA

La descripción de los puertos de la entidad ***códec_controller*** se explican a continuación:

- **BIT_CLK:** señal de sincronismo de ciertos elementos secuenciales de la entidad
- **VOL_CODE:** valor del volumen de la señal de audio.
- **FREC_CODE:** valor de la frecuencia de la señal de audio.
- **RESTART:** señal de inicialización de la entidad.
- **RESET:** señal de inicialización del códec de audio LM4550.
- **SYNC:** señal de sincronismo generada para enviar la trama.
- **SDATA_OUT:** trama de salida a proporcionar a los DACs del códec LM4550.

5.4.1. Diseño

Para un mejor entendimiento y mejor depuración de código del diseño la entidad ***códec_controller*** se dividirá en varios módulos. Al igual que pasaba con la entidad ***epp_controller*** esta se ha modificado de forma que se adapte a las funciones de ChipScope que se desean mostrar. Por ello se tendrá un módulo encargado de la generación de la trama a enviar al códec de audio que constará de una máquina de estados, otro módulo encargado de la sincronización del envió de las tramas y otro módulo generará una señal de reloj específica para la sincronización del envió de las tramas. Dichos módulos se explicarán por separado a continuación, los cuales se muestran en la Figura 90.

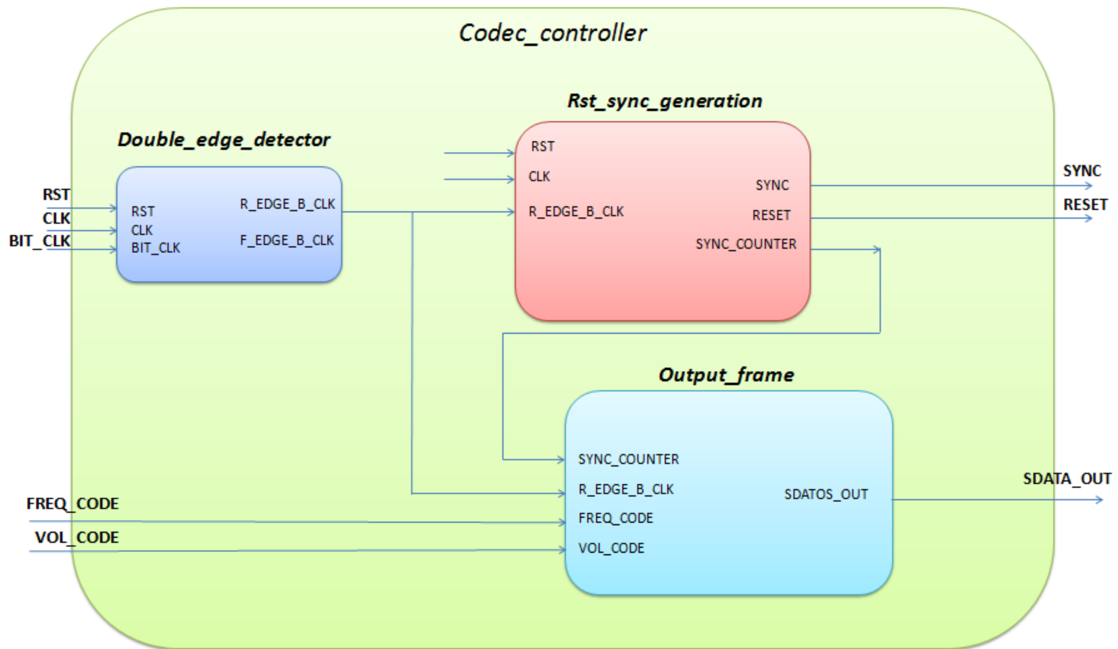


Figura 90: nueva entidad códec_controller

edge_detector.

La entidad ***edge_detector*** será la encargada de generar la señal ***bit_clk*** la cual se usará como señal de sincronismo en varios de los módulos del diseño encargados de la transmisión de la trama que controla el códec de audio LM4550.

El diseño funcionará con una señal de reloj de 100MHz, sin embargo el códec de audio LM4550 funciona con una señal de reloj de 12.288MHz lo que hace necesario generar una señal de sincronismo ***r_edge_b_clk*** a partir de esta señal de reloj. Para ello se diseñará un detector de flanco de subida de la señal ***bit_clk***. El diseño de un detector de flanco se ha explicado anteriormente en el diseño de la entidad ***epp_controler***.

Rst_and_sync_generation.[3]

Esta entidad es la encargada de generar el reset del códec de audio y la señal de sincronismo encargada de establecer el periodo de muestreo de los DACs del códec de audio.

El reset a diseñar será el "cold reset" el cual se genera tras obtener un nivel bajo en la señal ***RESET*** (pin 11) durante un periodo mayor de 1us, tal y como se puede apreciar en la Figura 91. Hay que tener en cuenta que la señal ***bit_clk*** no comienza a generarse hasta pasado 162.8ns después del reset, de manera que esta se diseña a partir de la señal de reloj.

[3] La información necesaria para poder generar tanto el reset como la señal de sincronismo se encuentra en el anexo I.

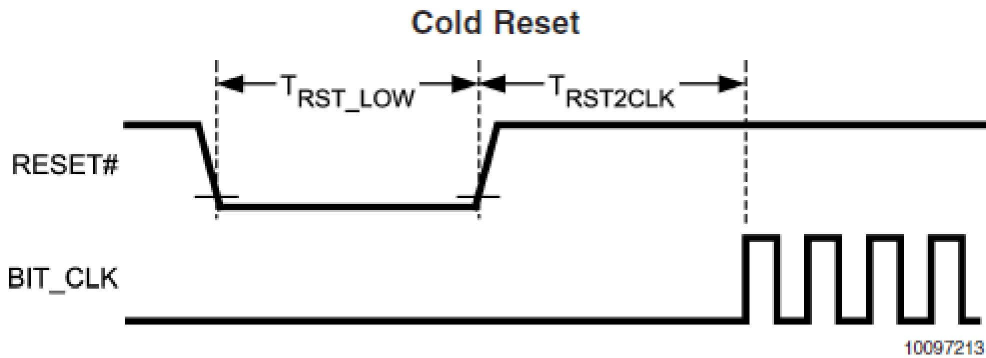


Figura 91: diagrama de tiempos de la señal de reset

La señal de reloj empleada en el diseño es de 100MHz, lo que proporciona un periodo de 10ns. El tiempo del reset a nivel bajo tiene que ser mayor a 1us de manera que se tiene que el tiempo de reset T_{RST_LOW} tiene que ser superior a 100 pulsos de reloj.

$$T_{RST_LOW} > \frac{1000ns}{10ns} = 100\ pulsos$$

Las tramas del códec de audio se alinean a partir de la transición de la señal ***SYNC***, la cual debe estar sincronizada con los flancos de subida de la señal ***bit_clk***. Cada transición de nivel bajo a alto indica una nueva trama de salida. La señal ***SYNC*** se genera con una frecuencia de 48 kHz, permaneciendo a nivel alto durante 1,3 µs, es decir 16 pulsos de la señal de reloj ***bit_clk***.

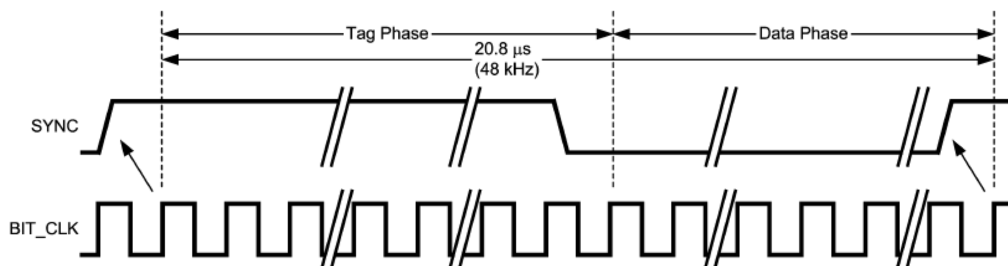


Figura 92: cronograma de generación de la señal SYNC

Para generar la señal ***SYNC*** se emplea la señal ***bit_clk***, sin embargo esto no es un buen diseño ya que se desea que sea un sistema totalmente síncrono por lo que al trabajar directamente con la señal ***bit_clk*** no se estaría teniendo en cuenta la señal de reloj del sistema. Debido a esto se generará una señal de habilitación (clock enable) que proporcione un pulso de duración un periodo de la señal de reloj cuando se produzca un flanco de subida en la señal ***bit_clk***. De este modo se tendrá una señal ***SYNC*** síncrona.

Para generar tanto la señal de reset como la señal de sincronismo de implementará un contador de pulsos de las señales de reloj que intervienen en el códec, ***clk*** para el ***reset*** y ***bit_clk*** para ***SYNC***. El diagrama del diseño se muestra en la Figura 93.

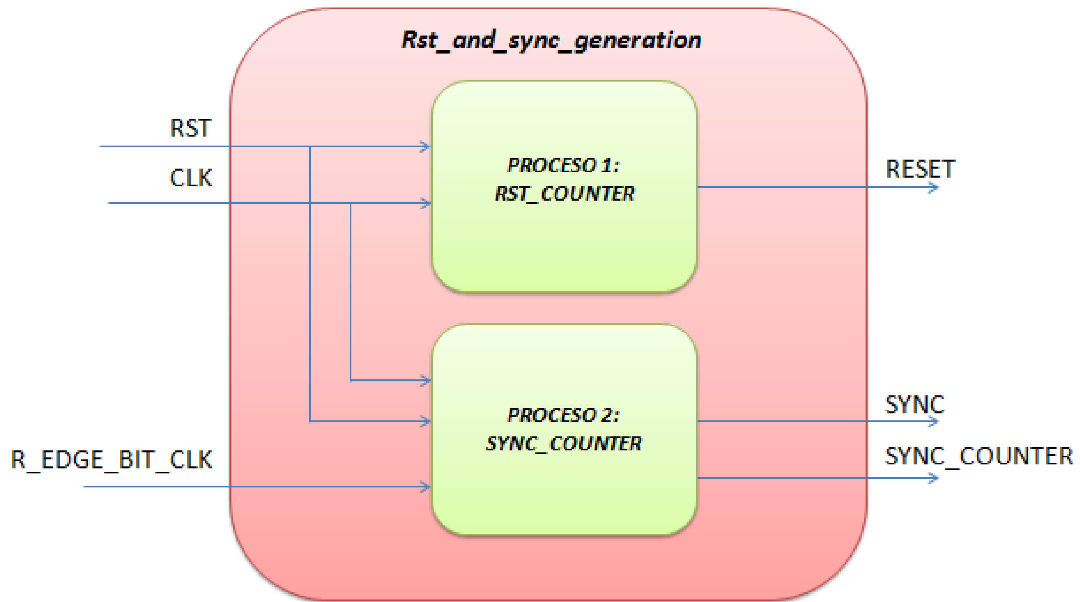


Figura 93: diagrama de bloques de la entidad rst_and_sync_generation

Output_frame.

El módulo ***output_frame*** genera las tramas responsables de la configuración del códec de audio. Las tramas se enviarán constantemente siguiendo la máquina de estados mostrada en la Figura 94.

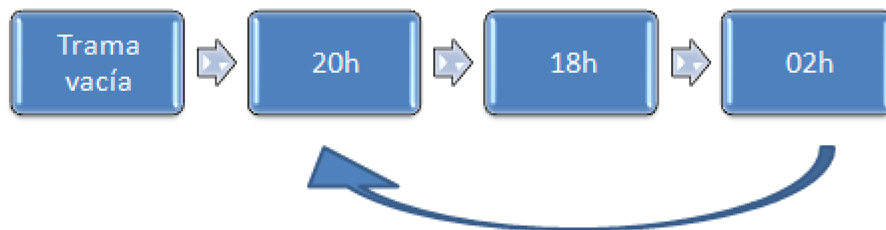


Figura 94: state chart de la trama

Las tramas enviarán la señal de audio a ser reproducida que será generada por una memoria la cual contenga las muestras correspondientes al seno que generará el audio. Desde la herramienta ChipScope será posible observar la forma de onda de los datos enviados en el SLOT 3.

Este módulo es también el responsable de decodificar la frecuencia a la cual el audio será reproducido. Esto se consigue a partir de un prescaler y un contador que barre la memoria que contiene las muestras. Se utilizará un registro para unir cada una de los slots que componen la trama además de mantener el sincronismo del sistema.

Una vez se tenga la trama completa se conseguirá un bus de datos de 256 bits. Debido a que la trama debe ser enviada por un puerto serie, es necesario serializarla. Esto se logra mediante

un multiplexor que envía un bit del bus de datos para cada ciclo de la señal de reloj. La Figura 95 muestra la entidad ***output_frame***.

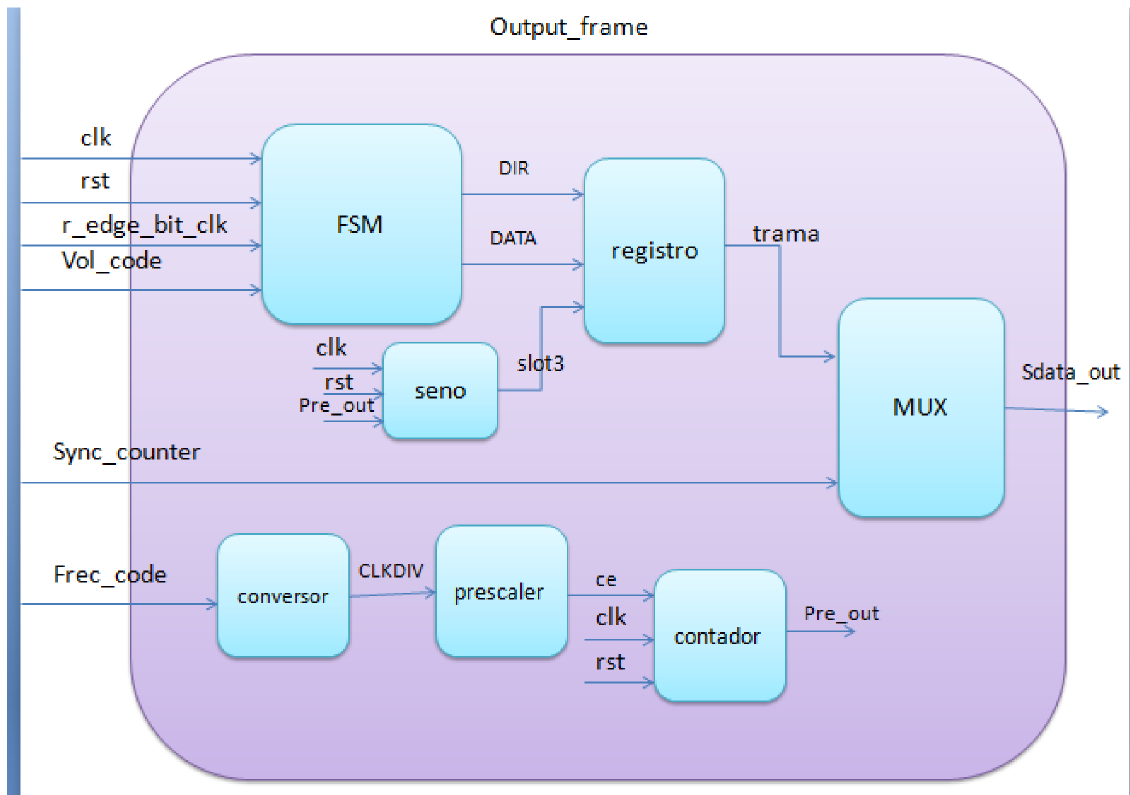


Figura 95: entidad output_frame

CAPITULO 6:

Implementación y depuración.

La última etapa del proceso es la implementación y depuración del sistema. En esta etapa se debe comprobar el correcto funcionamiento del diseño, así como la depuración de los posibles fallos que este contenga. Para ello se hará uso de la herramienta ChipScope integrada en el entorno de desarrollo ISE de Xilinx.

Debido a que es un trabajo de fin de grado centrado en el estudio y demostración de la herramienta ChipScope, se empleará solo parte del sistema antes diseñado, aplicando los módulos de ChipScope en varios de los bloques que componen el diseño del reproductor de partituras de música dependiendo de la función que se vaya a mostrar. Como se ha dicho anteriormente se generarán varios proyectos con los módulos a depurar y la inclusión de ChipScope.

6.1. Controlador del puerto EPP

Es posible que en ciertos diseños intervengan señales que sean la base para generar parte del diseño y cuyos tiempos sean críticos, y sin embargo que estos tiempos sean desconocidos. Este es el caso de las señales que intervienen en el protocolo EPP, de las cuales es conocido el cronograma pero no los tiempos.

Se usará la herramienta ChipScope para visualizar las señales internas correspondientes a ***ASTRB***, ***DSTRB*** y el bus de ***DATOS*** de modo que se pueda medir el tiempo de ejecución de los ciclos de lectura y escritura. Para ello se empleará el núcleo ILA el cual permite la observación de dichas señales.

Como primer paso se insertarán los núcleos ICON e ILA a través de la herramienta Core Generator. El núcleo ICON va a poseer una única señal de control, conectada al núcleo ILA. Se van a observar las señales ***ASTRB, DSTRB, PWRITE y DATA***, por lo que le bus de datos de núcleo ILA debe tener un tamaño de 12 bits. Como señal de disparo se hará uso de las señales ***ASTRB, DSTRB y PWRITE***, obteniendo un puerto trigger de 8 bits.

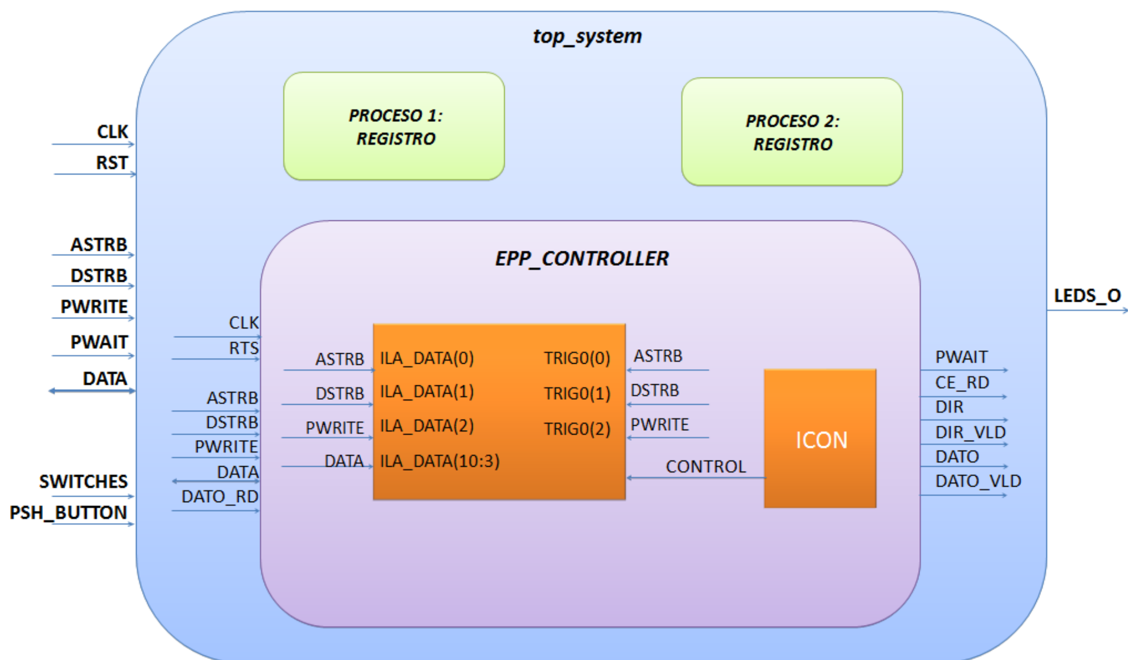


Figura 96: diagrama de bloques usado en la depuración

Como resultado al conexionado de las señales del diseño con el núcleo ILA se ha obtenido el código mostrado en la Figura 97, el reso del código se proporciona en el Anexo II (Epp_controller).

```
ila_data(0) <= ASTRB;
ila_data(1) <= DSTRB;
ila_data(2) <= PWRITE;
ila_data(10 downto 3) <= DATA;
ila_data(11) <= '0';
trig0(0) <= ASTRB;
trig0(1) <= DSTRB;
trig0(2) <= PWRITE;
trig0(7 downto 3) <= (others => '0');
```

Figura 97: conexionado de los núcleos de ChipScope en el diseño

Al lanzar el analizador lógico de ChipScope se debe configurar la señal de disparo para indicar el inicio de captura de muestras. Se quiere obtener la imagen de un ciclo de escritura y lectura por lo tanto se empleará el valor 010, esto es cuando se obtenga un nivel alto en la señal ***DSTRB*** mientras en las señales ***PWRITE*** y ***ASTRB*** estén en nivel bajo. Esta configuración indica transferencia de dirección siendo esta la primera etapa de los ciclos, tal como se ha visto anteriormente en la Figura 81. La configuración del trigger se puede ver en la Figura 98.

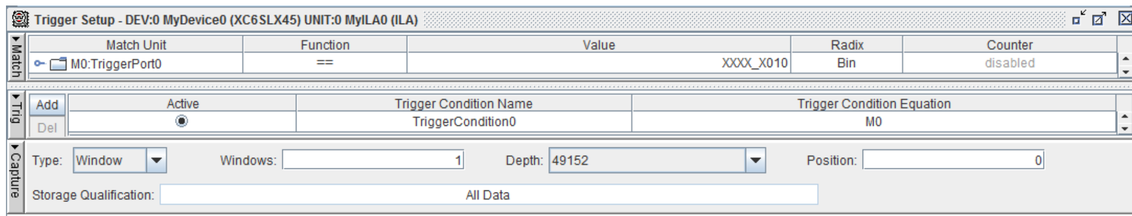


Figura 98: Configuración de la señal de disparo

En la primera simulación, en la que se utiliza la señal de reloj del sistema para obtener las muestras, se puede apreciar que el tiempo no es suficiente ya que la señal a observar dura más tiempo del mostrado, lo que hace que se muestre la señal incompleta (Figura 99 y Figura 100).

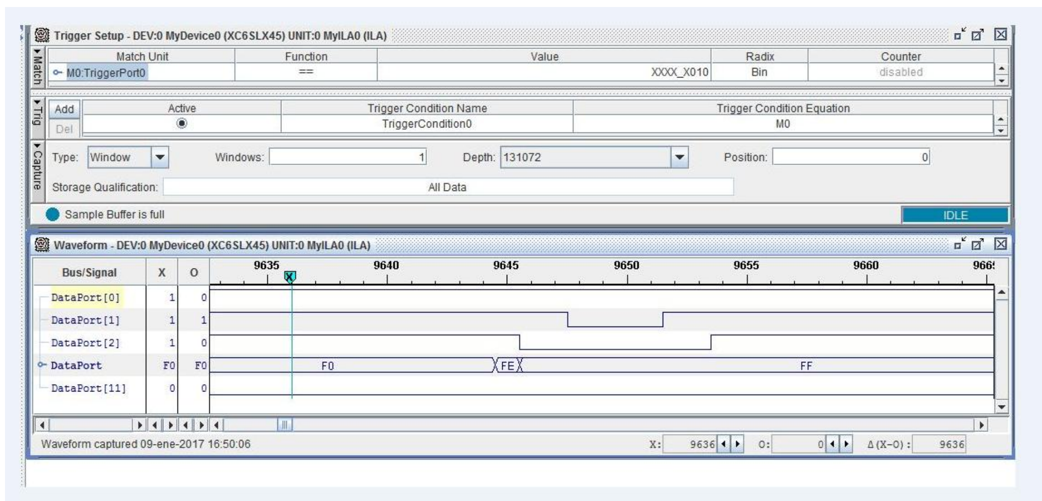


Figura 99: escritura de datos

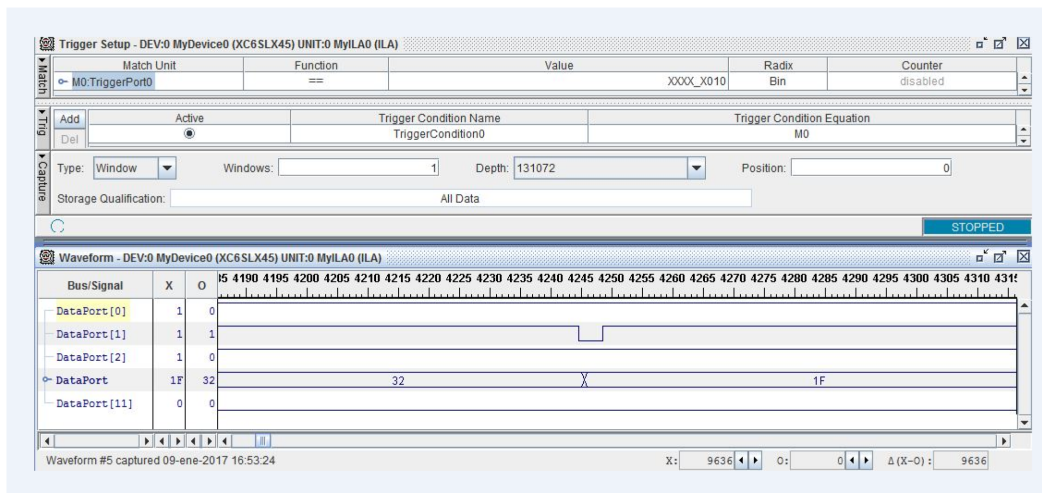


Figura 100: lectura de datos

Para solucionar este problema se deberá introducir un prescaler para generar la señal de reloj con la que capturar las muestras del analizador lógico, quedando ahora el diagrama de bloques resultante de la Figura 101. Se ha decidido por un periodo de muestreo de 50ns.

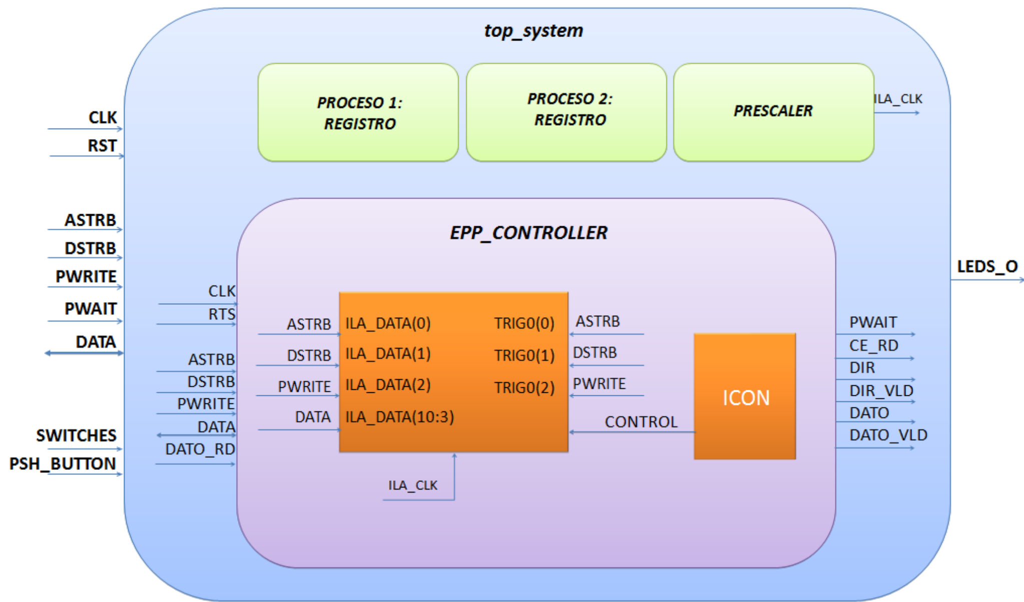
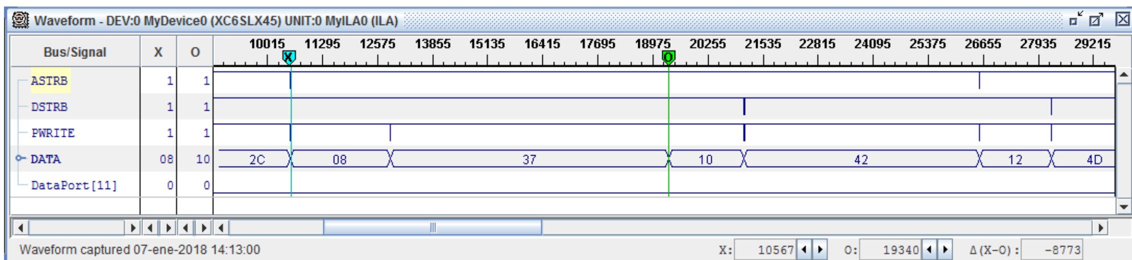


Figura 101: diagrama de bloques del diseño modificado

Los resultados ahora se muestran en las Figura 102 y la Figura 103 donde se puede ver la diferencia con las muestras anteriormente capturadas, pudiendo medir en este caso el tiempo en los ciclos de escritura y lectura del protocolo EPP.



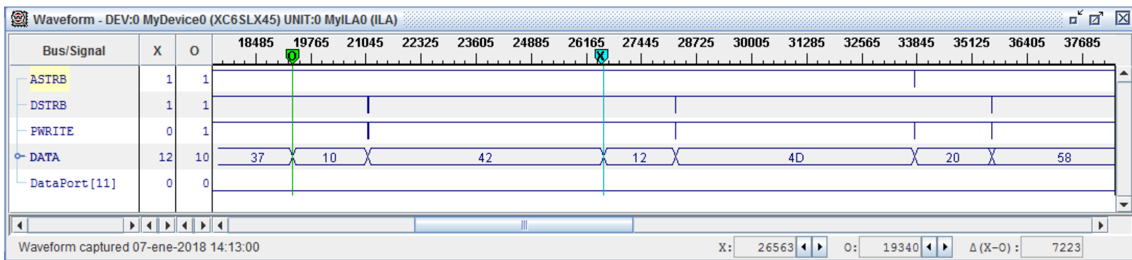


Figura 102: Ciclo de escritura

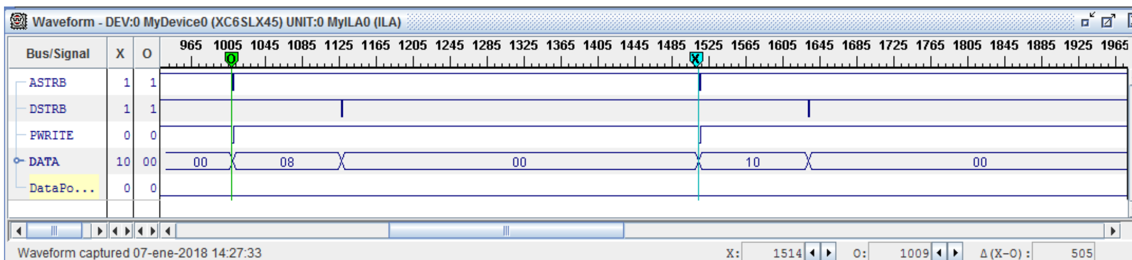


Figura 103: ciclo de lectura

Para el ciclo de escritura se han obtenido distintos resultados en la escritura de varias direcciones de entre 7223 y 8773 muestras por ciclo, por lo que se ha llegado a la conclusión de que un ciclo de lectura puede variar entre 7000*50ns= 350ms y 9000*50ns= 450ms. Como consecuencia se ha medido un ciclo de escritura y un ciclo de lectura del protocolo EPP obteniendo los tiempos mostrados en las imágenes de la Figura 104 y la Figura 105 para el ciclo de escritura y el ciclo de lectura, respectivamente.

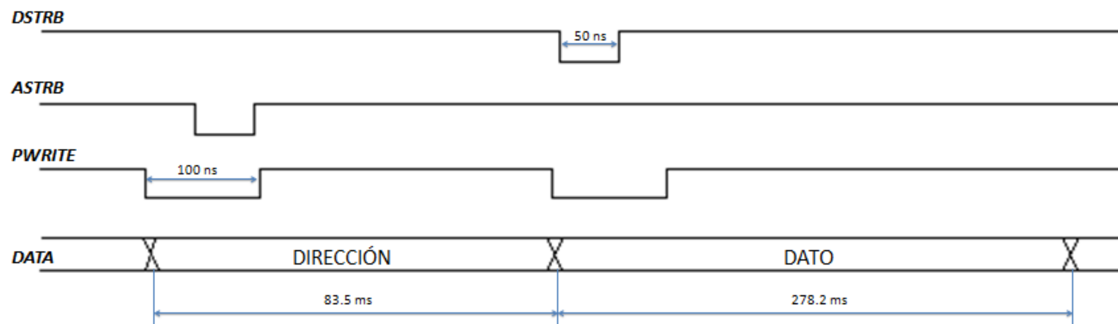


Figura 104: cronograma de tiempos reales del ciclo de escritura

En el caso del ciclo de lectura se tienen 505 muestras de duración, obteniendo así un ciclo de 505*50ns= 25.25ms.

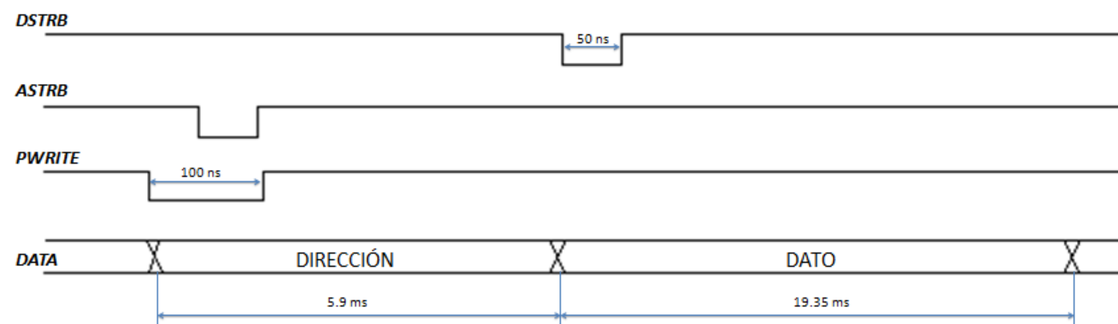


Figura 105: cronograma de tiempos reales del ciclo de lectura

6.2. Decoder_epp

El siguiente módulo a analizar dentro del diseño será la entidad ***decoder_epp***. Se va a utilizar los núcleos ILA y VIO del siguiente modo. El núcleo ILA será el encargado de visualizar las señales internas que intervienen en el diseño para comprobar su correcto funcionamiento. Como este módulo necesita de entrada de datos se empleará el núcleo VIO para ello. Con esto se va a mostrar un uso simple de la herramienta ChipScope, donde se simulará una entrada de datos para obtener una salida. El diagrama de bloques del diseño depurado con ChipScope se muestra en la Figura 106.

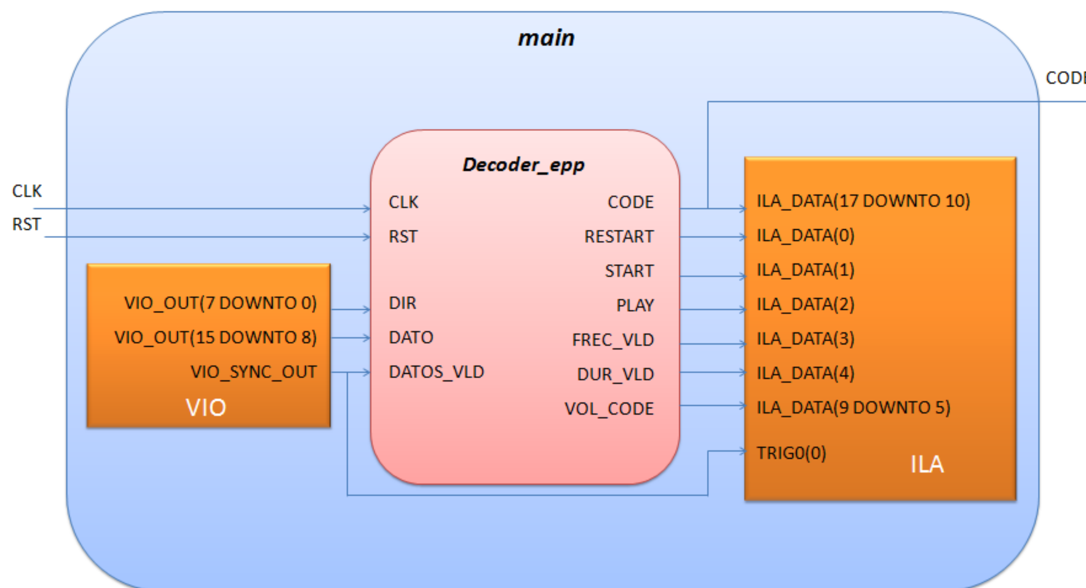


Figura 106: diagrama de bloques de la entidad decoder_epp tras insertar ChipScope

Al introducir el núcleo VIO, el módulo ***decoder_epp*** se introducirá dentro de un módulo ***main***. Esto también ayuda a visualizar los puertos de salida del módulo que, en este caso, son las únicas señales que intervienen en el diseño. Las señales conectadas a ChipScope se muestran en la Figura 107, mientras que en el Anexo II (DECODER_EPP) se encuentra la instanciación de los núcleos junto con el resto del código.

```
addr <= vio_out(7 downto 0); --vio
data <= vio_out(15 downto 8); --vio
valid_data<= vio_syn_out(0); --vio

  CODE <= code_out;

ila_data(0) <= rest;
ila_data(1) <= str;
ila_data(2) <= ply;
ila_data(3) <= valid_freq;
ila_data(4) <= valid_dur;
ila_data(9 DOWNTO 5)   <= volume;
ila_data(17 DOWNTO 10) <= code_out;
ila_data(20 DOWNTO 18) <= (OTHERS =>'0');
trig0(0) <= valid_data;
```

Figura 107: conexion de las señales de ChipScope

Las ventanas de ChipScope Analyzer se configurarán para adaptarse al diseño a depurar. En la consola de VIO se va a tener dos entradas de datos por teclado, ***DIR*** y ***DATO***, y una señal síncrona ***DATO_VLD*** configurada como un botón que generará un pulso a nivel alto. La señal de disparo coincidirá con la señal ***DATO_VLD*** como se puede observar en la Figura 108.

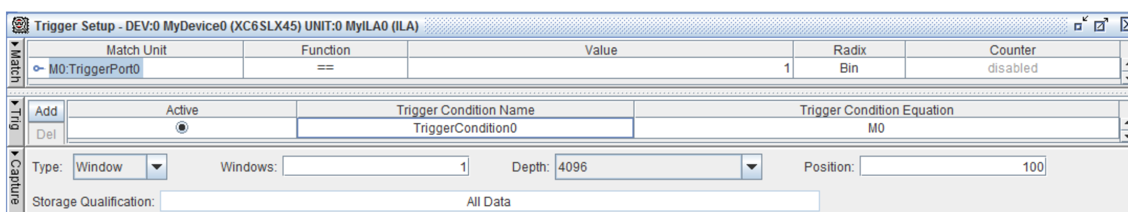


Figura 108: configuracion trigger

Una vez se tienen todas las pestañas configuradas según el diseño se puede lanzar el analizador del núcleo ILA. En este caso, como muestra del módulo, se ha introducido la dirección 0x”11” y el dato 0x”11” que dan como salida un nivel alto en la señal ***RESET***. Este comportamiento se observa en la Figura 109.

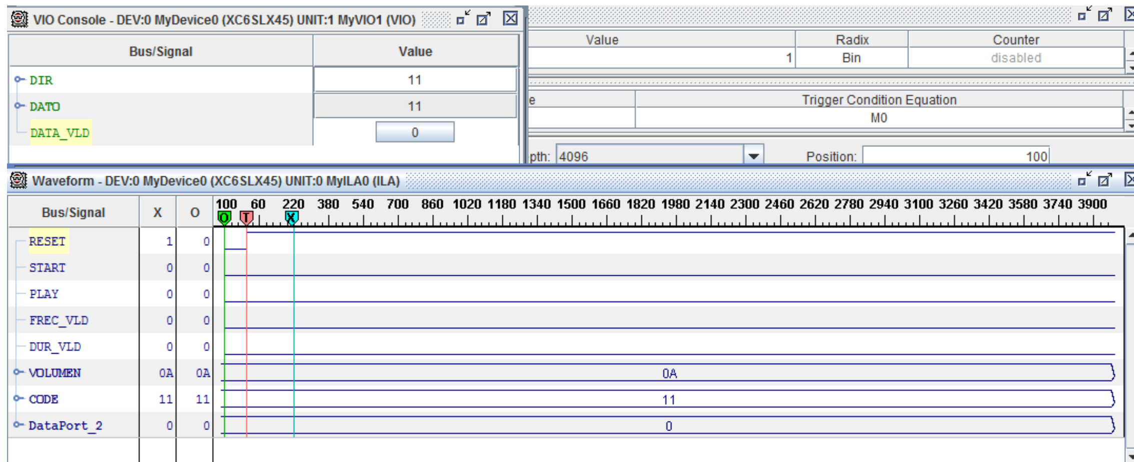


Figura 109: comportamineto del módulo decoder_epp

Se ha indicado que se representen las 100 muestras anteriores a que ocurra la señal de disparo, así es posible observar con claridad el flanco de subida de las señales a observar.

6.3. Gen_frec_code

Dentro de la entidad ***gen_frec_code*** se hará uso de la memoria FIFO para demostrar la funcionalidad del núcleo VIO. Se utilizará una memoria FIFO que será modelada en base a una plantilla establecida, adaptándola a las necesidades del diseño.

Gracias a la herramienta ChipScope Pro, se generarán los datos de entrada de la memoria FIFO para verificar su correcto funcionamiento. Para ello, se utiliza el núcleo VIO para que se pueda obtener una serie de botones que simulen las señales de reset de lectura y escritura y también las señales de habilitación.

De la misma manera, es posible simular un LED de modo que indique cuándo se alcanza un flanco ascendente en las señales ***empty*** y ***full***.

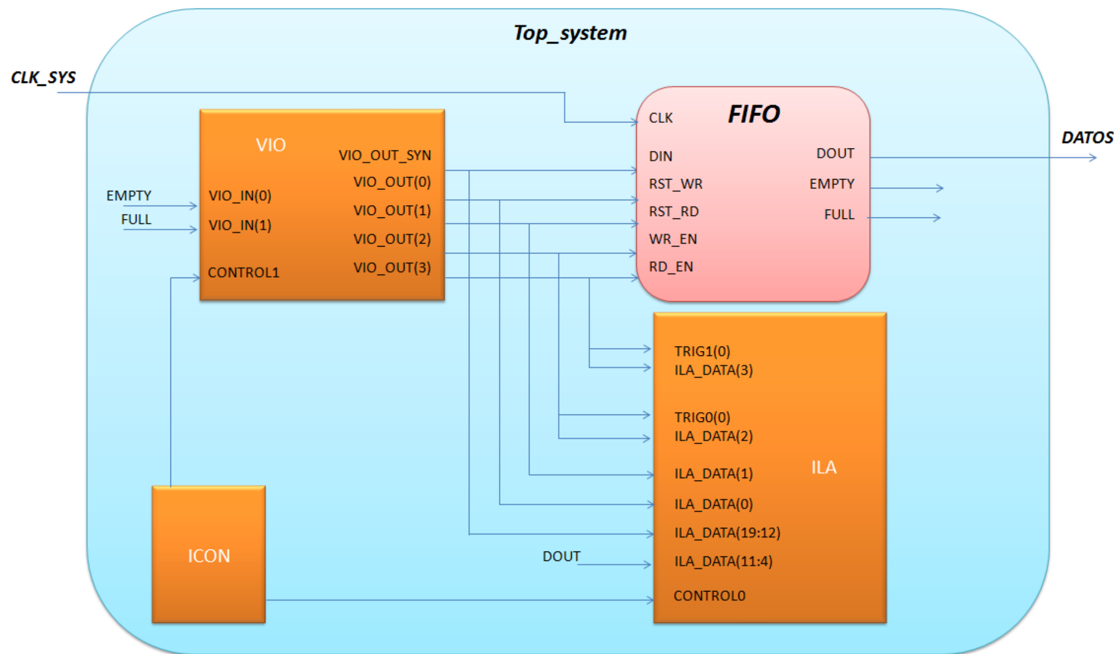


Figura 110: diagrama de bloques del diseño insertando la herramienta ChipScope

El núcleo VIO será configurado con una señal de entrada asíncrona de 2 bits y dos señales de salida, una síncrona de 8 bits y una asíncrona de 4 bits (Figura 111).

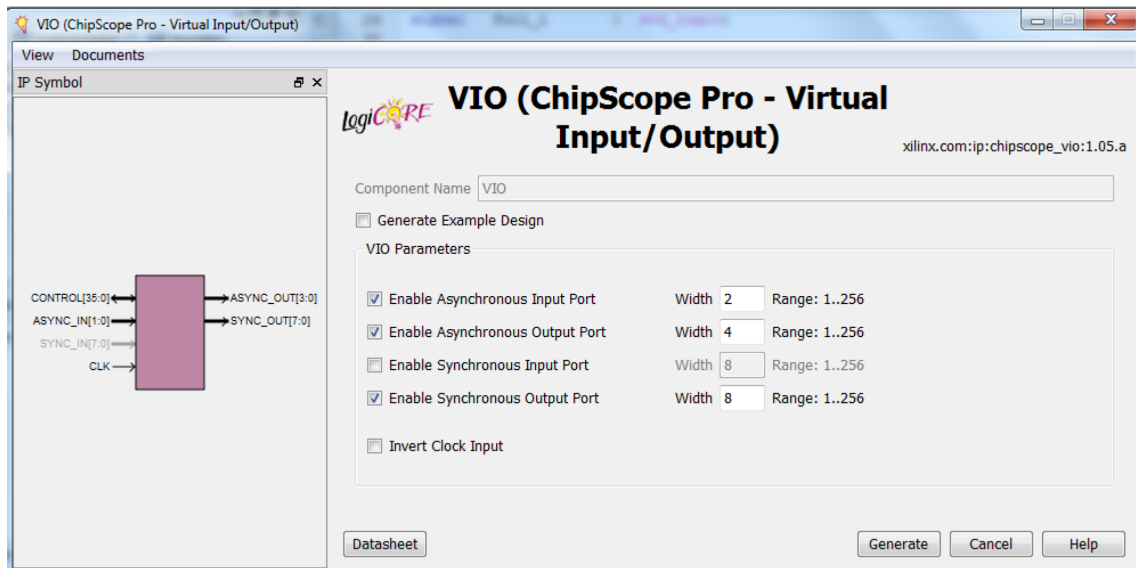


Figura 111: configuración del núcleo VIO

Se va a configurar el núcleo ILA de modo que se tengan dos señales de disparo, ***wr_en_i*** y ***rd_en_i***. Como resultado se van a programar dos puerto trigger de 4 bits cada uno. Por otro lado se ha configurado un bus de datos de 20 bits de tal forma que se puedan observar las señales necesarias para la depuración del diseño (Figura 112).

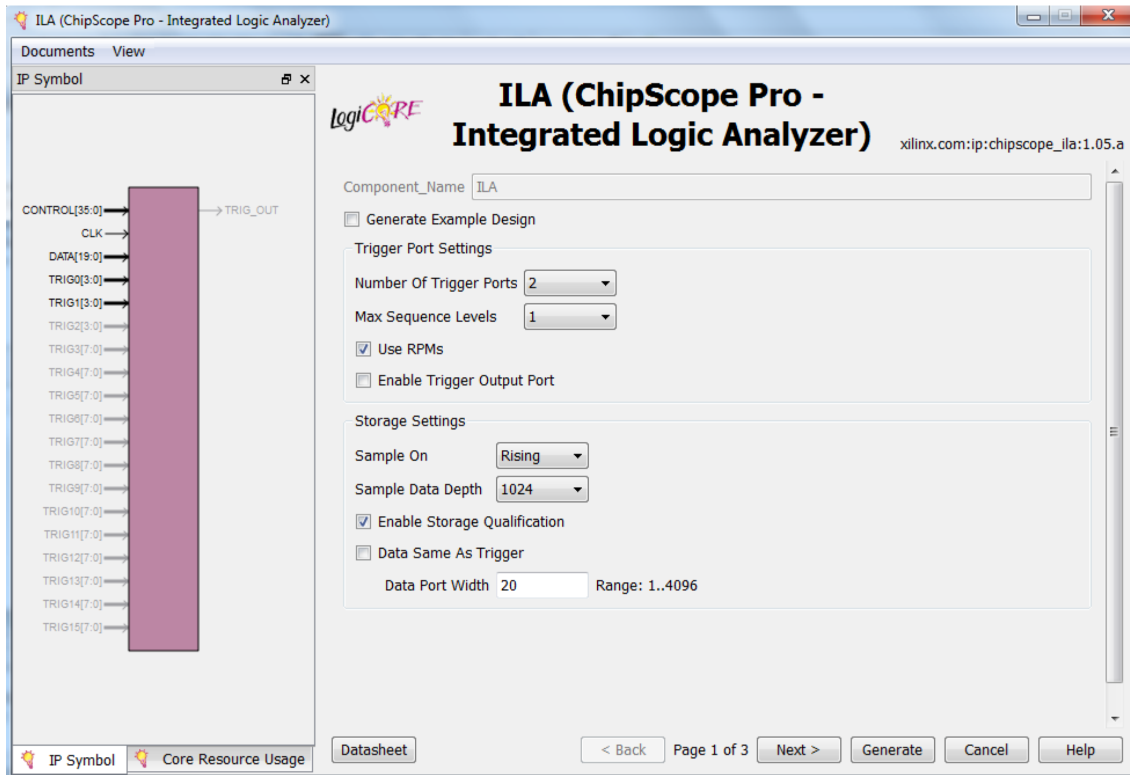


Figura 112: configuración del núcleo ILA

El conexionado de las señales a visuaizar a través de ChipScope se muestra en la Figura 113, el resto del código se mostrara en el Anexo II (FIFO).

```
ila_data(0) <= rst_wr_i;
ila_data(1) <= rst_rd_i;
ila_data(2) <= wr_en_i;
ila_data(3) <= rd_en_i;
ila_data(11 downto 4) <= dout_i (7 downto 0);
ila_data(19 downto 12) <= din_i(7 downto 0);

trig0(0) <= wr_en_i;
trig1(0) <= rd_en_i;
trig0(3 downto 1) <= (others => '0');
trig1(3 downto 1) <= (others => '0');

din_i     <=vio_out_syn(7 downto 0);
rst_wr_i  <=vio_out(0);
rst_rd_i  <=vio_out(1);
wr_en_i   <=vio_out(2);
rd_en_i   <=vio_out(3);
vio_in(0) <= empty_i;
vio_in(1) <= full_i;
```

Figura 113: conexión de las señales de ChipScope con el diseño

Tras programas la placa Atlys y ejecutar el analizador ChipScope se han adaptado las pantallas de la herramienta al diseño actual a depurar. En primer lugar se va a trabajar con la consola del núcleo VIO para generar las entradas del diseño.

Se muestran dos posibilidades para introducir datos en la memoria. El primero es el uso de una salida asincrónica configurada como entrada de texto, en la que los datos a escribir se introducirán manualmente. Esto se muestra en la Figura 114 con la señal ***din***.

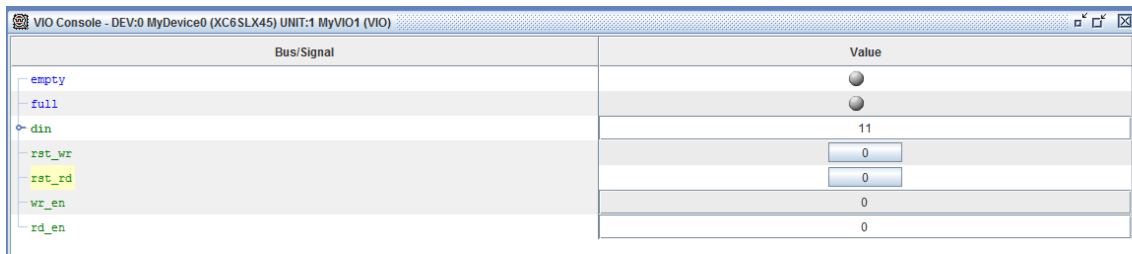


Figura 114: consola del módulo VIO

La otra opción es utilizar un tren de pulsos, en el que indicar los valores a escribir. Esto se puede ver en la señal ***din*** mostrada en la Figura 115. Este tren de pulsos permite editarse e introducir el valor de cada pulso individualmente. La Figura 116 muestra dicha pantalla de edición del tren de pulsos.

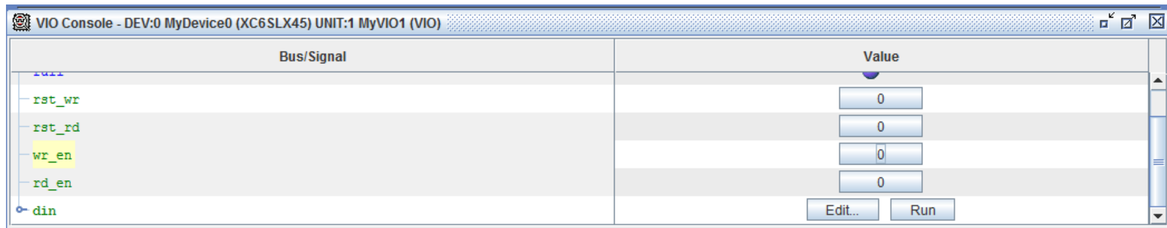


Figura 115: consola VIO configurando din como tren de pulsos

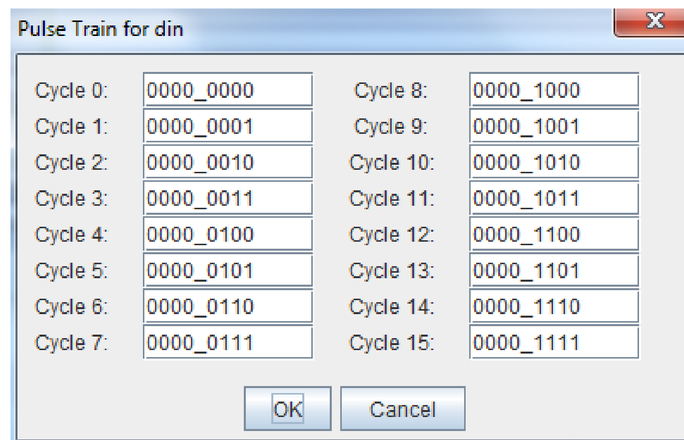


Figura 116: configuración de los valores del tren de pulsos

Los puertos de salida del núcleo VIO pueden configurarse también como botones, pudiendo ser switch o pulsadores (Figura 117). En este ejemplo se usa la opción 'Push Button' de manera que simule un comportamiento real del sistema obteniendo asi un pulso a nivel alto en la señal simulada.

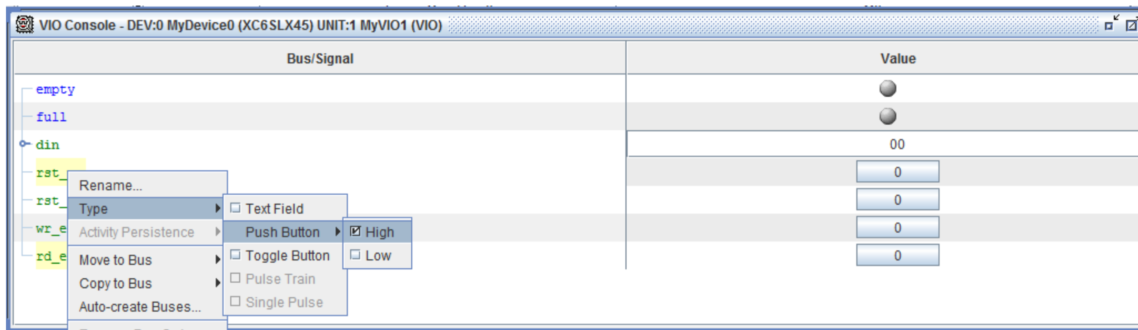


Figura 117: configuración puerto de entrada del núcleo VIO

El puerto de entrada asíncrono del núcleo VIO se ha empleado para observar el comportamiento de las señales ***empty*** y ***full***, las cuales se van a simular como LEDs. Desde que el núcleo ILA no permite observar señales pertenecientes a los puertos de entrada y salida, esta función es la idónea para suplir esta necesidad. En la Figura 118 se muestra como, al detectar un cambio en la señal, el LED simulado cambia de color además de indicar con una flecha si ha habido una transición.

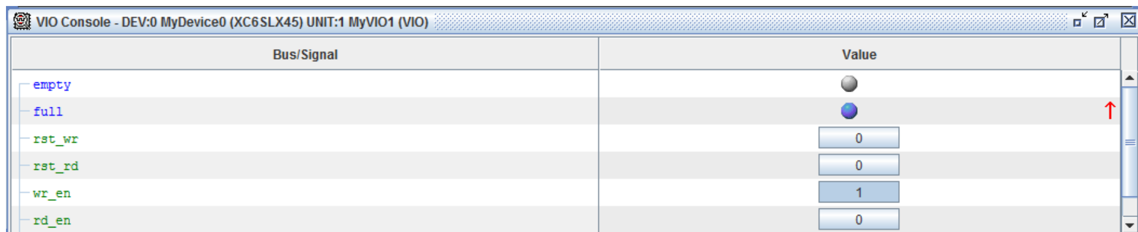


Figura 118: funcionamiento de los puertos de salida del núcleo VIO

Una vez configurada la consola VIO, se utilizará el analizador del núcleo ILA para observar el resultado de las señales que intervienen en el módulo FIFO. Como señal de disparo se ha empleado una combinación de dos puertos trigger, siendo disparada en el momento en que se obtenga un nivel alto en la señal ***wr_en_i*** o en la señal ***rd_en_i***. La configuración de la señal de disparo se muestra en la Figura 119.

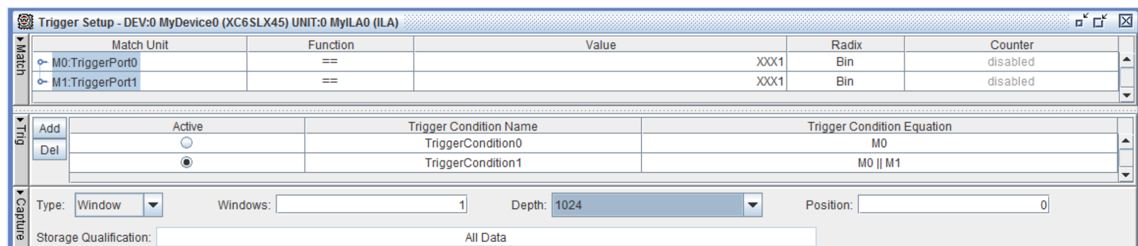


Figura 119: configuración del trigger

El resultado obtenido en la ventana del analizador lógico de ILA tras estas modificaciones se puede ver en la Figura 120.

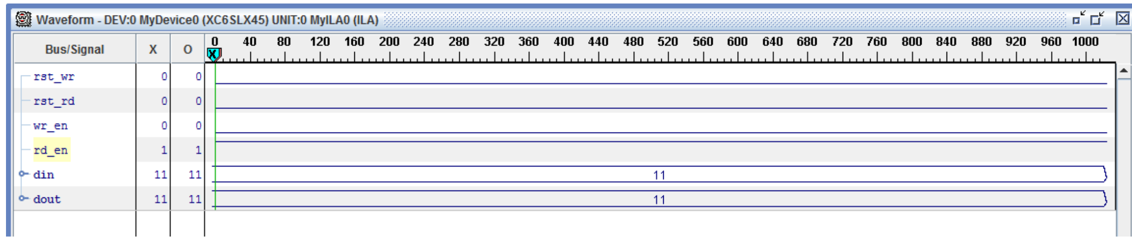


Figura 120: lectura del dato escrito en la memoria FIFO

6.4. Codec_controller

La entidad ***codec_controller*** demuestra varias funciones de los núcleos ILA y VIO como el uso de los trigger, el conexionado de varios núcleos ILA en cascada o el detector de actividad en puertos de entrada.

En el caso de la entidad ***codec_controller***, ChipScope se introducirá no sólo para depurar el diseño, sino también para mostrar ciertas funciones que tiene, aunque no sean necesarias en la depuración de este diseño en particular. En la Figura 121 se observa el diagrama de bloques de la entidad principal del diseño tras añadir la herramienta ChipScope.

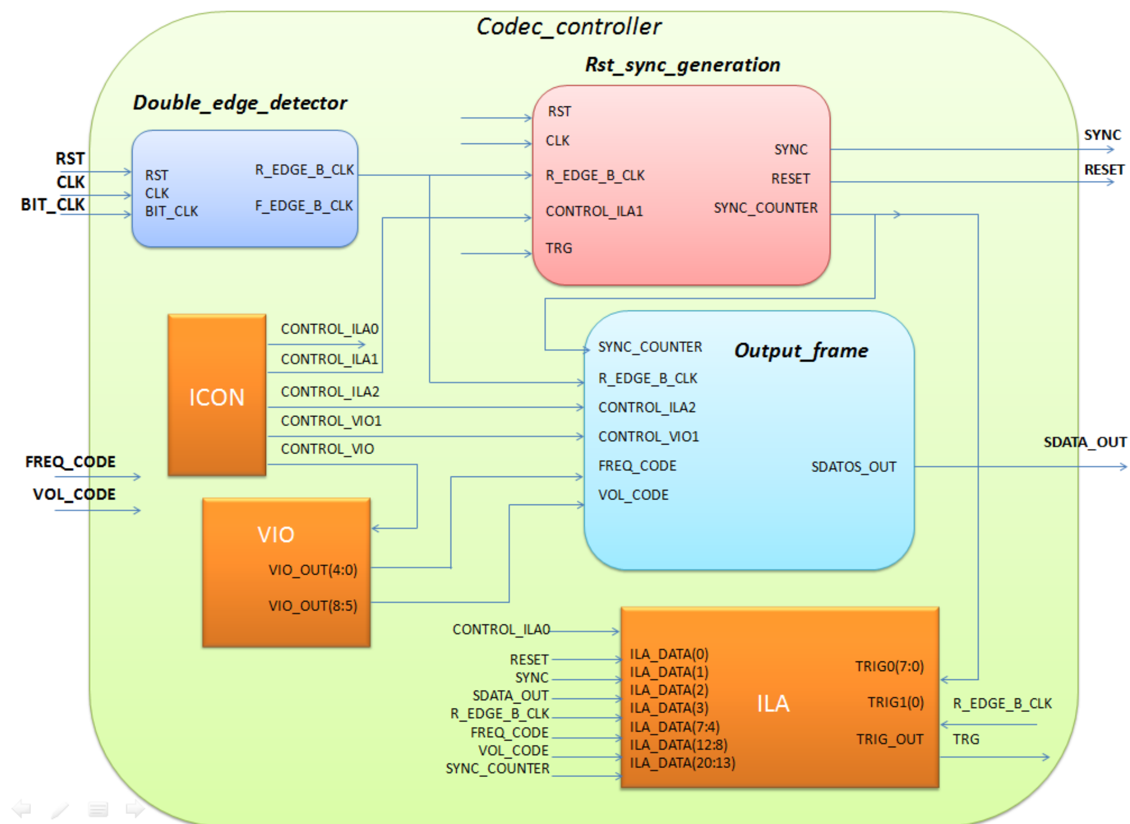


Figura 121: diagramas de bloque del diseño a depurar

Se introducirá el núcleo ICON en el módulo principal dando cobertura al resto de módulos que componen el diseño. Además se introducirá un núcleo ILA con el que poder ver el funcionamiento general del diseño y un núcleo VIO con el que poder simular los puertos de

entrada ***freq_code*** y ***vol_code***. El código empleado para esta entidad se encuentra en el Anexo II (Codec_controller).

Se ha utilizado un núcleo ILA en la entidad principal para que pueda visualizarse el comportamiento de todo el sistema. Las señales utilizadas como trigger son ***sync_count*** y ***r_edge_b_clk*** habilitando el trigger de salida de manera que se producirá un nivel alto en este puerto cuando se cumpla la condición de disparo, en este caso cuando ***sync_count*** esté a cero y haya un flanco ascendente en la señal ***r_edge_b_clk***. Entonces el analizador lógico comenzará a capturar y mostrar las señales pudiendo observar así la generación de la señal de sincronización. La configuración de la señal de disparo se muestra en la Figura 122.

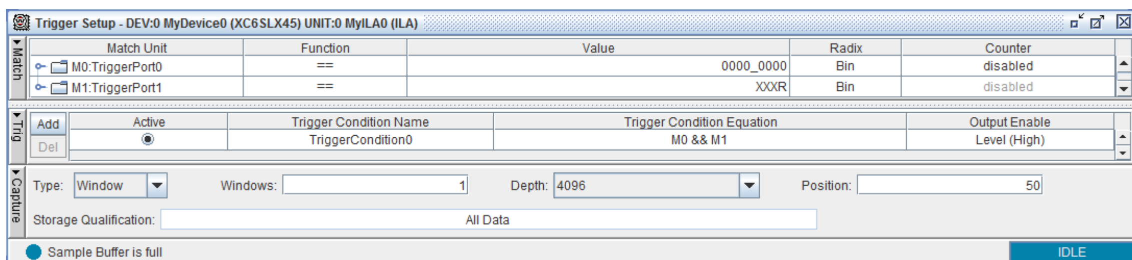


Figura 122: configuración de los puertos trigger del núcleo ILA principal

Gracias a la consola del núcleo VIO del módulo principal (Figura 123) es posible indicar los valores de frecuencia y volumen que recibiría la entidad ***códec_controller*** desde el resto del diseño.

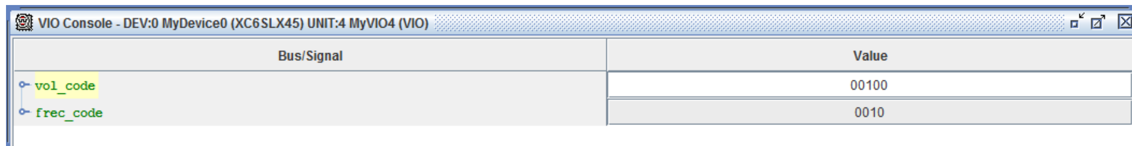


Figura 123: consola VIO del módulo principal

Como resultado del comportamiento general del diseño de ha obtenido la Figura 124.

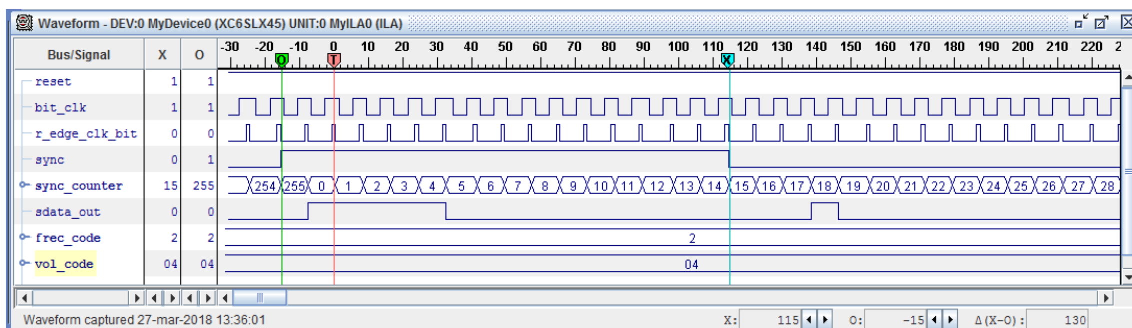


Figura 124: depuracion entidad codec_controller

Rst_and_sync_generation

Primero se comenzará con el análisis de la entidad ***rst_and_sync_generation***, cuyo diagrama de bloques se corresponde a la Figura 125. El código de esta entidad aparece en el Anexo II (Rst_and_sync_generation).

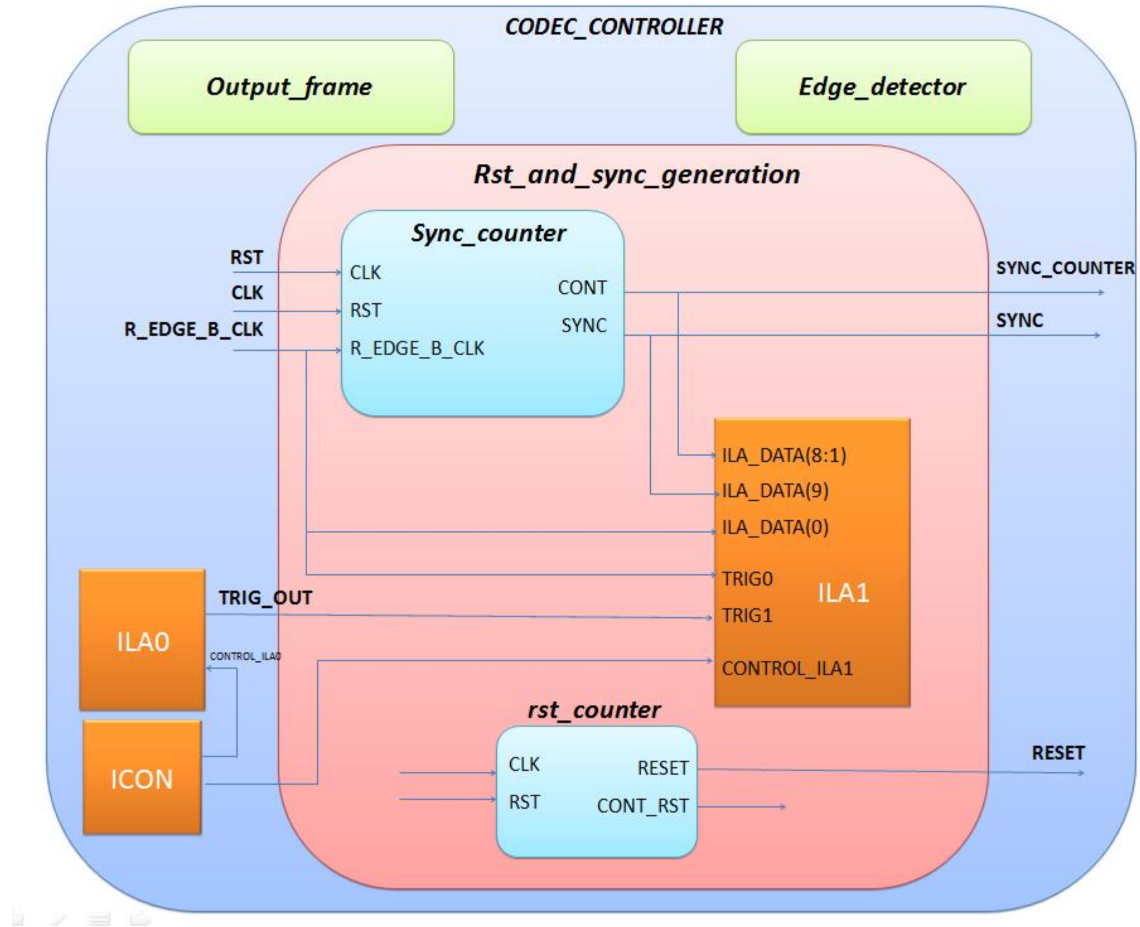


Figura 125: diagrama de bloques del conexionado de núcleos ILA

Puerto trigger de salida

En el caso de diseños complejos en los que intervienen varias entidades es posible que se desee comprobar el comportamiento de cada módulo en los mismos tiempos de ejecución. Al poseer varios núcleos ILA distintos cada uno tendrá puntos de captura distintos ya que se lanzará el analizador lógico en distintos momentos del tiempo. Para obtener sincronismo a la hora de obtener muestras de las señales conectadas en los distintos núcleos se emplea el trigger de salida como señal de disparo para la conexión en cascada de varios núcleos ILA. De esta manera se ejecutarán los analizadores lógicos de forma secuencial obteniendo una visión de todos los módulos que intervienen en el diseño a partir de una única condición y en el mismo periodo de tiempo.

Para este ejemplo se desea contemplar el comportamiento general del sistema junto con el funcionamiento del módulo ***rst_and_sync_generation***. Se ha conectado el puerto trigger de salida del núcleo ILA de la entidad principal con el puerto trigger de entrada del núcleo ILA de

la entidad ***rst_and_sync_generation*** tal como se ha visto en el diagrama de bloques. Se debe ejecutar primeramente el analizador ILA de los módulos secundarios y por último el núcleo ILA que contenga el trigger de salida para así poder ver el mismo instante de tiempo en ambos analizadores.

La configuración de la señal de disparo del núcleo ILA instanciado en la entidad ***rst_and_sync_generation*** se muestra en la Figura 126.

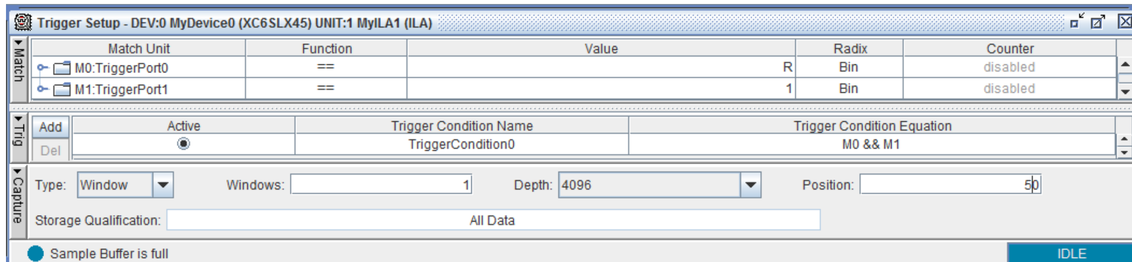


Figura 126: configuracion triggers

Como consecuencia al trigger de salida que actúa como señal de disparo del núcleo ILA secundario en la entidad ***rst_and_sync_generation***, se obtiene la imagen de la Figura 127.

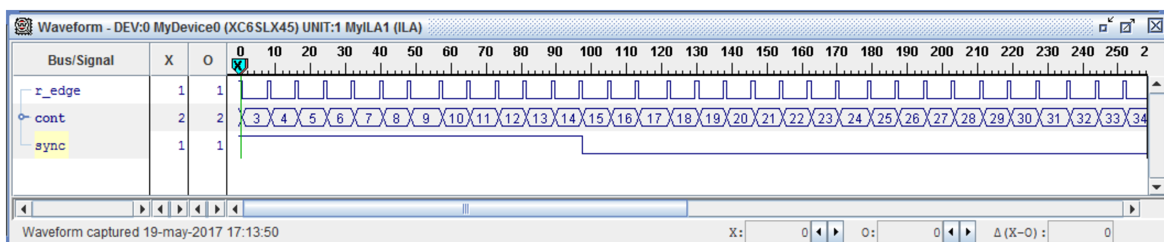


Figura 127: Rst_and_sync_generation

A continuación se comprobará como el resultado del sistema en conjunto que se muestra en la pantalla del núcleo ILA principal (Figura 128) coincide en el tiempo con el análisis del núcleo ILA secuenciado pero con un pequeño retardo debido a la latencia del puerto trigger de salida.

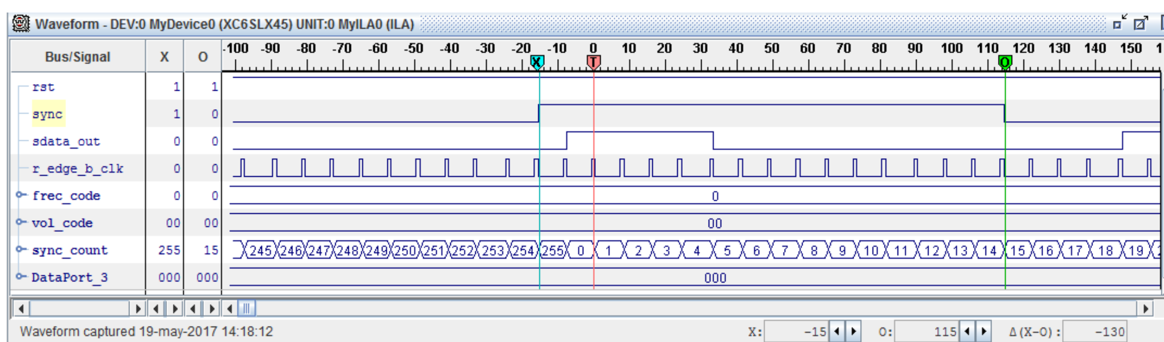


Figura 128: Codec_controller

Output_frame

El siguiente módulo a analizar es la entidad ***output_frame*** en la cual se incluirá un núcleo ILA con el que poder analizar las señales que lo componen y un núcleo VIO. La Figura 129 muestra el interconexionado de las señales pertenecientes al módulo ***output_frame*** con los núcleos de ChipScope cuyo código se muestra en el Anexo II (Output_frame).

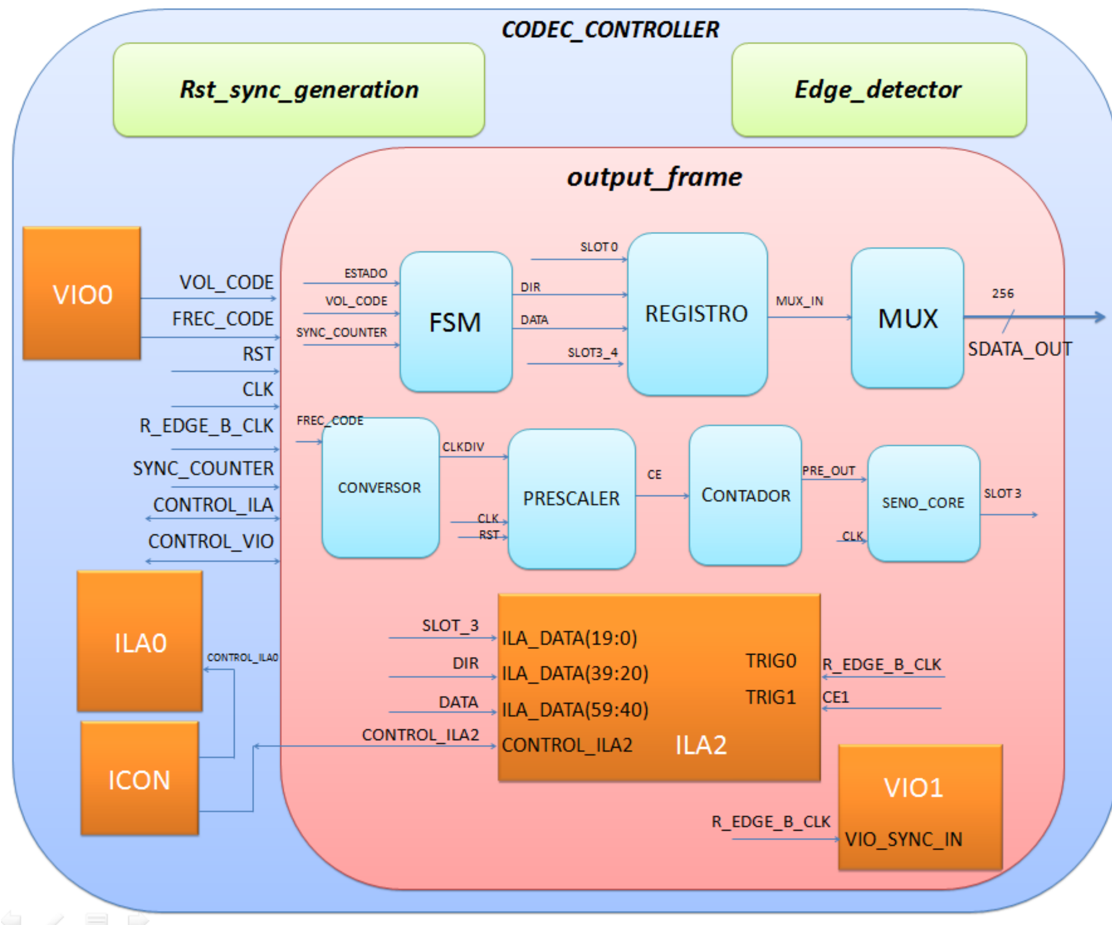


Figura 129: diagrama de bloques del diseño a depurar

Bus plot

Se utilizará el núcleo ILA en la entidad ***output_frame*** para mostrar la forma de onda del seno en relación con el tiempo. La Figura 130 muestra la ventana "*waveform*" del núcleo ILA, que es con la que se ha estado trabajando hasta ahora, donde la señal del bus de datos son dados como valores en hexadecimal; sin embargo, es difícil apreciar la forma de onda que se está produciendo. La ventana "*bus plot*" se utiliza para mostrar una imagen de *datos Vs tiempo* del bus de datos seleccionado.

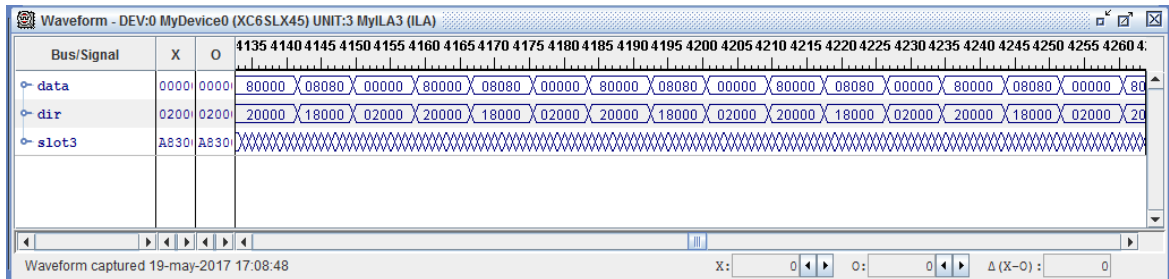


Figura 130: Output_frame

La opción “*bus plot*” se encuentra en la parte izquierda del analizador lógico de ChipScope y al hacer doble clic sobre ella abre la ventana de la Figura 131. En esta ventana se puede seleccionar la señal a representar, la forma de representación (si es lineal o con puntos) y los puntos máximos y mínimos en el eje a representar.

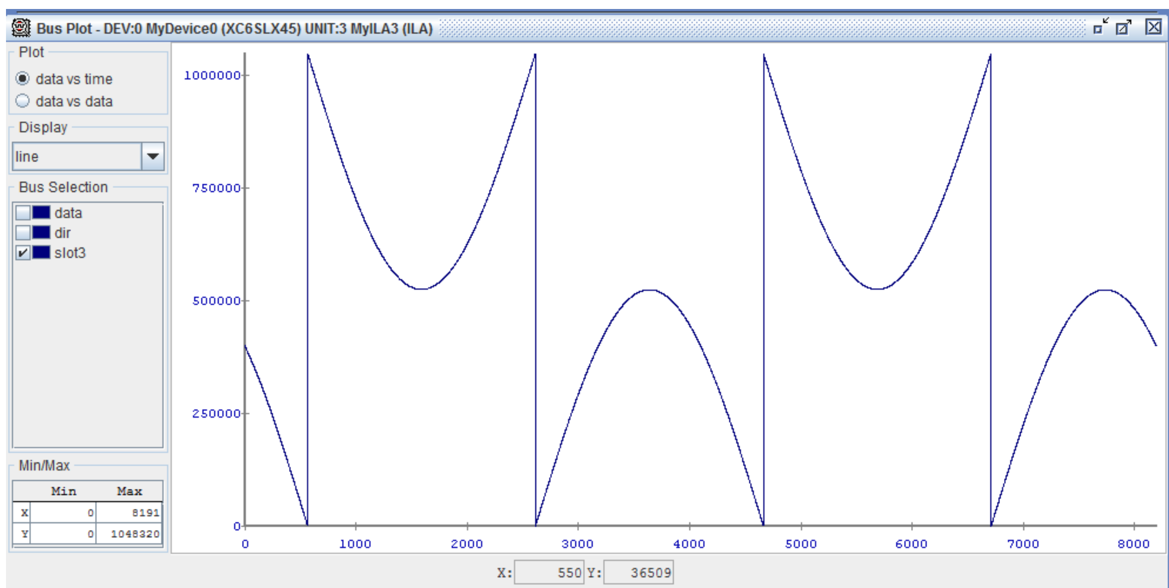


Figura 131: muestras de audio pertenecientes al SLOT3

Se puede apreciar como el resultado obtenido no es una señal seno. Esto es debido a que no se ha indicado el tipo de dato correcto. Hay que tener cuidado con el tipo que se está usando a la hora de visualizar un bus de datos ya que puede dar lugar a conclusiones erróneas. La imagen muestra los datos en hexadecimal, que una vez cambiados a decimal (Figura 132) obtendrá una señal sinusoidal perfecta. El tipo de dato se cambia en la ventana ‘*waveform*’.

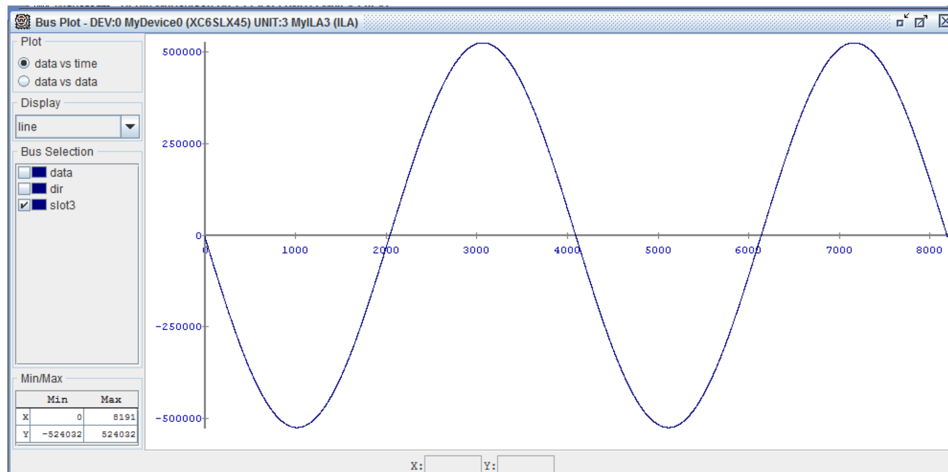


Figura 132: muestras de audio en formato decimal

Detección de flancos

La entidad ***output_frame*** se empleará como ejemplo en la detección de flancos en un puerto de entrada, función del núcleo VIO. La señal que se evaluará será ***r_edge_b_clk***. Esta opción permite detectar transiciones que tengan una mayor frecuencia a la frecuencia de muestreo. Para ello se conecta la señal que se desea evaluar en un puerto de entrada síncrono del núcleo VIO.

Debido a que esta opción sólo permite detectar la actividad entre muestras, para observar este efecto es necesario utilizar una señal cuya frecuencia sea mayor a la frecuencia de muestreo. En este caso como la frecuencia de muestreo es proporcionada por el prescaler y no por la señal de reloj del sistema como es lo normal, es posible ver en la Figura 133 una actividad en el puerto de la señal ***r_edge_b_clk***. Esta actividad está representada con una flecha a la derecha del valor de la señal.

El uso del detector de actividades es un fuerte recurso a la hora de detección de glitches y de problemas de sincronismo.

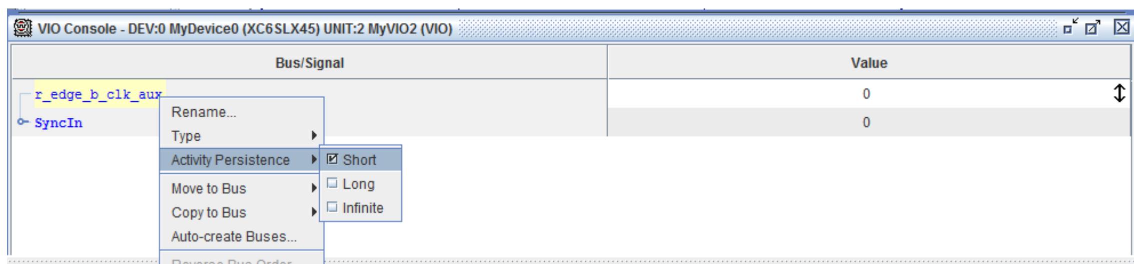


Figura 133: detector de actividad de la señal r_edge_b_clk

Capitulo 7:

Evaluación de las funciones de ChipScope

A continuación se evaluarán los distintos usos de la herramienta ChipScope en la depuración de diseños basándose en el estudio de los distintos casos previamente vistos.

Medida de tiempos:

Para el análisis de esta funcionalidad se ha utilizado el diseño de ***epp_controler***. Al principio, se utilizó una frecuencia de muestreo igual a la del diseño. Sin embargo, al analizar las señales obtenidas, se puede observar que el tiempo mostrado en la depuración no es suficiente y los ciclos se quedan incompletos. Como resultado se ha concluido que, para una correcta visualización y medición del tiempo de una señal, se debe tener en cuenta el periodo de muestreo y asegurarse de que se visualice el tiempo necesario para llevar a cabo la medición.

Debido a la medición del tiempo del protocolo EPP, es posible realizar una ejecución correcta del sistema diseñado, utilizando así procesos con un tiempo que permita que el protocolo EPP se lleve a cabo correctamente sin solapamientos.

Triggering:

La multitud de opciones que ofrece la herramienta ChipScope en cuanto a triggering se refiere, permite capturar las muestras justo en el momento deseado lo que da al diseñador flexibilidad a la hora de depurar además de optimizar en recursos.

A lo largo del proyecto de fin de grado se han ido encontrando problemas en la captura muestras a través del núcleo ILA porque la elección de la señal de disparo ha sido incorrecta. Por esta razón, es necesario asegurarse de usar una señal de disparo que se genere de modo que se pueda producir la condición necesaria para mostrar las señales deseadas del diseño.

Uno de estos problemas se puede ver en el análisis del semáforo, donde la señal ***cs*** se usó inicialmente como disparador. El analizador lógico no pudo mostrar ningún resultado debido a que dicha señal no se producía a causa del funcionamiento erróneo.

En el diseño de la memoria FIFO y la entidad ***codec_controller*** se empleó una combinación de varios puertos trigger de forma que la captura de datos sea dada por una condición específica impuesta por el diseñador.

Optimización:

Como muestra del funcionamiento de este recurso se ha decidido por su uso en la depuración del diseño del semáforo. En este caso particular su uso no es muy útil, sin embargo ha ayudado a comprender el funcionamiento de este.

Esta opción permite a los diseñadores optimizar los recursos utilizados por el dispositivo. Esto es útil cuando tienen diseños complejos en dispositivos con capacidad RAM limitada. No obstante, para el uso de esta herramienta el usuario debe tener una mínima idea de lo que debe obtenerse para que el diseño pueda ser correctamente depurado. En el caso del diseño del semáforo, la señal resultante tras el uso de la función *'storage qualification'* es incompleta, por lo tanto era difícil entender el comportamiento del sistema.

Output trigger:

El puerto de disparo de salida del núcleo ILA incluye varias funciones de las cuales se ha estudiado la conexión en cascada de varios núcleos ILA. Ha sido difícil encontrar un ejemplo para aplicar esta función a causa de que los diseños utilizados no tienen una gran complejidad. Por esta razón solo es necesario el uso de un único núcleo ILA.

El mejor ejemplo posible se ha mostrado en el diseño del controlador del códec de audio el cual incluye varias entidades. Como resultado, se ha obtenido el análisis de las señales internas que intervienen en las diferentes entidades utilizando como desencadenante la condición de disparo resultante únicamente de las señales internas de una de las entidades.

La utilidad de esta función se debe a la posibilidad de obtener la visualización de las señales de ambos módulos en el mismo momento de la ejecución. Esto no sería posible de otro modo ya que el analizador debería ejecutarse manualmente lo que conlleva una pérdida de tiempo por la reacción humana por la que ambas señales no se estarían capturando en el mismo instante de tiempo. Por otro lado, este puerto tiene la desventaja de tener un retardo de 10 ciclos de la señal del reloj, el cual sigue siendo más rápido que la intervención humana.

Bus plot:

Esta opción dada por el núcleo ILA permite observar la forma de onda de un bus de datos en el analizador lógico. A través de la ventana '*waveform*' del analizador solo es posible verificar el valor de las muestras sin embargo, es más fácil verificar el comportamiento de ciertas señales de una manera visual. Este es el caso de la entidad ***output_frame*** estudiada, donde se ha mostrado el seno enviado con el que se genera la señal de audio. Es posible observar cómo la señal enviada coincide exactamente con la forma de onda seno, que de otra forma se tendría que comprobar cada valor de muestra individualmente lo que conllevaría la pérdida de una gran cantidad de tiempo.

Tren de pulsos:

El uso del tren de pulsos es útil cuando es necesaria la simulación de una serie de valores específicos en la entrada de un módulo a depurar. El problema de esta función es el hecho de que solo es posible obtener un tren de pulsos de duración de 16 ciclos de reloj.

En este proyecto, el uso del tren de pulsos se ha utilizado como muestra de su funcionalidad en la memoria FIFO. El tiempo que pasa entre el momento el que se lanza el tren de pulsos y comienza la captura de muestras de datos en el analizador es demasiado largo, por lo que el tren de pulsos no se ha podido evaluar correctamente. Sin embargo, ha sido posible mostrar su inserción y un posible uso que se le puede atribuir.

Detector de actividades:

El detector de actividad permite detectar problemas técnicos sin consumir recursos del dispositivo. Los fallos de sincronización en los diseños son la causa más común de este tipo de problemas, produciendo un error en el sistema completo.

El detector de actividad se pone a prueba en la entidad ***output_frame***, donde se debe evaluar la señal ***r_edge_b_clk***. Está en una señal que no está sincronizada con la señal de reloj. Se ha elegido esta señal de forma que sea posible comprobar la actividad que se produce entre las muestras capturadas, únicamente para mostrar el funcionamiento del detector de actividad ya que es posible saber de antemano que se produce actividad entre muestras en la señal ***r_edge_b_clk***. De esta manera, la consola del núcleo VIO detecta actividades indicándolo a través de un '1' lógico.

Capitulo 8: Conclusiones

En el estudio realizado sobre la herramienta ChipScope se han encontrado algunas dificultades debido a que no se ha encontrado una gran cantidad de material para este producto. La mayor parte del trabajo se ha basado en la documentación provista por la compañía de la herramienta (Xilinx) y en las conclusiones obtenidas mientras se ha estado trabajando en los distintos diseños y funciones.

A primera vista y sin conocimiento de ChipScope Pro, esta herramienta parece complicada de usar debido a las diferentes funciones que tiene. Sin embargo, una vez que el usuario puede entender cada una de estas opciones y el funcionamiento de la aplicación, ChipScope es una herramienta simple y que incluye una multitud de ventajas.

El uso de esta herramienta conlleva un nivel adecuado de conocimiento en el campo del diseño y la depuración de sistemas. El diseñador debe saber exactamente qué desea ver para usar ChipScope de manera óptima y adecuada, optimizando los recursos del dispositivo y utilizando las funciones apropiadas. Esto no es sencillo para usuarios con un bajo nivel de conocimiento ya que no obtendrán el máximo beneficio.

La incorporación de una manera simple y rápida de la herramienta le da al diseñador una manera de depurar los diseños que de otro modo sería difícil, o incluso imposible de lograr. El conocimiento obtenido en este documento se puede extrapolar a la depuración de otras FPGAs de diferentes fabricantes que utilicen diferentes entornos de desarrollo. Hoy en día la mayoría de los entornos incluyen software que permite depurar el diseño sin tener que invertir en equipos de alto costo. Ejemplos de estos son la herramienta SignalTap en el caso de los FPGA de Altera o FS2 Logic Navigator ™ de Atmel. En este documento, se ha estudiado la herramienta ChipScope debido al uso del dispositivo FPGA Spartan 6 para su realización, fabricado por Xilinx. Las ventajas de estas herramientas son similares, estando la diferencia en

la placa utilizada para la realización del diseño. Dependiendo del diseñador y las necesidades del diseño, se elegirá una placa u otra.

Como conclusión se ha llegado a que el uso de software de depuración en el diseño con FPGA es de gran utilidad, además de fácil y rápido de administrar y lograr. Hoy en día los software de depuración son una herramienta común en el ámbito de la electrónica, que facilitan al diseñador la tarea de comprobar y finalizar un diseño.

Capitulo 9:

Pliego de condiciones

La herramienta ChipScope es un software que ofrece Xilinx para la depuración de dispositivos FPGAs. Para el estudio de la herramienta ChipScope en este trabajo de fin de grado se ha empleado la versión ISE Design Suite 14.2. Este entorno soporta los dispositivos FPGA Spartan®-6, Virtex®-6, and CoolRunner™, así como las familias FPGAs anteriores.

En la siguiente tabla (Tabla 1) se detallan los dispositivos específicos que pueden soportar las diferentes ediciones de ISE.

	ISE WebPACK Tool	ISE Design Suite (All Other Editions)
Zynq® Device	Zynq-7000 Device • XC7Z010, XC7Z020, XC7Z030	Zynq-7000 Device • All
Virtex® FPGA	Virtex-4 FPGA • LX: XC4VLX15, XC4VLX25 • SX: XC4VSX25 • FX: XC4VFX12 Virtex-5 FPGA • LX: XC5VLX30, XC5VLX50 • LXT: XC5VLX20T - XC5VLX50T • SXT: None • FXT: XC5VFX30T Virtex-6 FPGA • LXT: XC6VLX75T Virtex-7 FPGA • None	Virtex-4 FPGA • All Virtex-5 FPGA • All Virtex-6 FPGA • All Virtex-7 FPGA • All non-SSIT devices
Kintex™ FPGA	Kintex-7 FPGA • XC7K70T, XC7K160T	Kintex-7 FPGA • All
Artix™ FPGA	Artix-7 FPGA • XC7A100T, XC7A200T	Artix-7 FPGA • All
Spartan® FPGA	Spartan-3 FPGA • XC3S50 - XC3S1500(L) Spartan-3A/-3AN/-3E FPGA • All Spartan-3A DSP FPGA • XC3SD1800A Spartan-6 FPGA • XC6SLX4 - XC6SLX75T	Spartan-3 FPGA • All Spartan-3A/-3AN/-3E FPGA • All Spartan-3A DSP FPGA • All Spartan-6 FPGA • All
CoolRunner™ XPLA3, CoolRunner-II, XC9500 CPLD	• All	• All

Tabla 1. Dispositivos soportados por ISE Design Suite

Los sistemas operativos requeridos para esta versión del software abarcan Windows XP/7/Server y Linux. Para el sistema operativo Windows 10 se debe actualizar el software a la versión ISE Design Suite 14.7, que únicamente soporta el dispositivo Spartan-6.

Para las nuevas familias Virtex®-7, Kintex®-7, Artix®-7, and Zynq®-7000 Xilinx recomienda el entorno de desarrollo Vivado® Design Suite.

Ahora bien, sabiendo los distintos dispositivos soportados por el entorno de desarrollo hay que tener en cuenta cuales de ello permiten el uso de ChipScope Pro. En la Tabla 2 mostrada a continuación se enumeran los dispositivos con los cuales es posible emplear ChipScope.

Supported Device Families

ChipScope Pro	ChipScope Pro Serial I/O Toolkit
Artix®-7 Zynq®-7000 Kintex®-7 FPGAs (All) Virtex®-7 FPGAs (All) Virtex-6 FPGAs (All) Virtex-5 FPGAs (All) Virtex-4 FPGAs (All) Spartan®-6 FPGAs (All) Spartan-3A/-3AN/-3A DSP/-3E/-3 FPGAs (All)	Kintex-7 FPGAs, Virtex-6 FPGAs (LXT, SXT, and HXT) Virtex-7 FPGAs Virtex-5 FPGAs (LXT, and FXT) Virtex-4 FPGAs (FX) Spartan-6 FPGAs (LXT)

Tabla 2. Familias de dispositivos soportados por ChipScope

En referencia a la comunicación de la placa con el PC, ChipScope utiliza las cadenas Boundary Scan en el JTAG del dispositivo. A continuación se listan una serie de cables usados para la comunicación que la herramienta ChipScope Pro Analyzer soporta.

- Platform Cable USB-II
- Platform Cable USB
- Parallel Cable IV
- Digilent JTAG-SMT1 and JTAG-HS1 USB-to-JTAG download cables
- ByteTools Catapult EJ-1 Ethernet-to-JTAG cable

Para tener una buena comunicación y ser capaz de conectarse con el dispositivo FPGA es necesario tener los drivers adecuados instalados en el PC. La programación del dispositivo FPGA se puede llevar a cavo de dos formas, a través de iMPACT o a través de la herramienta Digilent Adept. Para ello se debe instalar el siguiente software:

- Digilent Plugin for Xilinx Tools.
- Digilent Adept software.

Capitulo 10: Presupuesto

A continuación se analizará el presupuesto que se emplearía en la depuración de un diseño empleando ChipScope Pro. El presupuesto se dividirá en la parte hardware y la parte software.

Hardware

En el caso de este proyecto de fin de grado se ha empleado la placa Atlys Spartan-6, sin embargo, dependiendo de las necesidades del diseño se elegirá el dispositivo más adecuado para su realización. En el apartado anterior "Pliego de condiciones" se comentan las familias de dispositivos que permiten el uso de ChipScope.

Concepto	cantidad	Precio unidad	Coste total
PC	1	650€	650€
Placa Atlys Spartan-6 [4]	1	396,95€	396,95€
Xilinx Platform Cable USB II [5]	1	182,27€	182,27€
USB-JTAG Programming Cable [6]	1	121,52€	121,52€
Total sin IVA			1067,08€
IVA (21%)			283,65€
Total			***1350,74€***

Tabla 3. Costes de hardware

[4] https://store.digilentinc.com/atlys-spartan-6-fpga-trainer-board-limited-time-see-nexys-video/
[5] https://www.xilinx.com/products/boards-and-kits/hw-usb-ii-g.html
[6] https://store.digilentinc.com/xup-usb-jtag-programming-cable/

Software

Dependiendo del dispositivo empleado en la realización del diseño se trabajará con un software u otro. En la siguiente tabla se muestra el precio de los distintos entornos de desarrollo empleados en este trabajo de fin de grado.

Concepto	Precio unidad	Coste total
Emacs	Gratuito	Gratuito
ISE Design Suite	Gratuito	Gratuito

Tabla 4. Costes de software

Mano de obra

A la hora de calcular los costes de mano de obra se debe de tener en cuenta dos posibles opciones. El ingeniero responsable del proyecto puede ser un trabajador por cuenta ajena o autónomo.

Según el Boletín Oficial del Estado a diplomados y titulados de 1.er ciclo universitario se les aplica un salario anual de 17.544,24€, teniendo en cuenta que se trabaja una media de 1800 horas efectivas al año en España se obtiene un precio de aproximadamente 10€/hora. Evaluando esto, el trabajo realizado en este proyecto de fin de grado conllevaría un costo a la empresa de 5000€, reflejado en la Tabla 5.

Concepto	Horas	Precio/Hora	Coste total
Estudio y análisis de la herramienta	120	10€	1200€
Desarrollo de los ejemplos de estudio	200	10€	2000€
Documentación	180	10€	1800€
Total sin IVA			3950€
IVA (21%)			1050€
Total			***5000€***

Tabla 5. Costes de mano de obra

Ahora bien, si se tiene un proyecto realizado por un ingeniero autónomo hay que tener en consideración los gastos que esto conlleva como son seguridad social, impuestos, seguro, material, etc. De los que se estaría suponiendo un gasto de entre 1500€-2000€ mensuales, esto es entre 18000€-24000€ anuales. Si se desea obtener un salario de 18000€ al igual que un trabajador por cuenta ajena se deberá ganar un total de entre 36000€-42000€ anuales. Al ser autónomo el número de horas facturables se reduce por lo que se supondrá una media de entre 1000-1400 horas. En este caso el precio por hora rondaría entre 30€-50€. Para este proyecto de fin de grado se ha decidido por un precio/hora de 40€ lo que supondría un coste de mano de obra de 20000€.

Concepto	Horas	Precio/Hora	Coste total
Estudio y análisis de la herramienta	120	40€	4800€
Desarrollo de los ejemplos de estudio	200	40€	8000€

Documentación	180	40€	7200€
Total sin IVA			15800€
IVA			4200€
Total			***20000€***

Tabla 6: costes de mano de obra autónomo

Teniendo en ceunta los precios calculados, el presupuesto total empleado para la realización del trabajo de fin de grado sería el mostrado en la Tabla 7.

HARDWARE	1350,74€
SOFTWARE	GRATUITO
MANO DE OBRA (Autónomo)	20000€
TOTAL	21350,74€

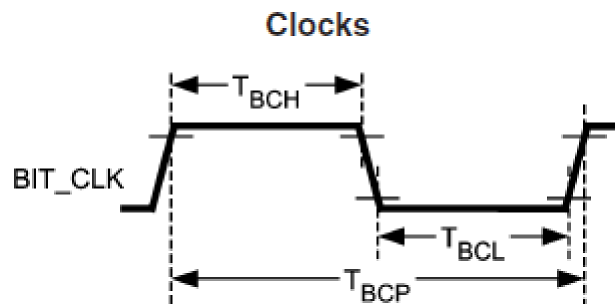
Tabla 7: presupuesto final

ANEXOS

ANEXO I: Códec de audio LM4550

BIT_CLK

La señal ***Bit_clk*** es generada por el propio códec de audio con una frecuencia de 12.288 MHz.



F_{BC}	BIT_CLK frequency		12.288		MHz
T_{BCP}	BIT_CLK period		81.4		ns
T_{CH}	BIT_CLK high	Variation of BIT_CLK duty cycle from 50%		±20	% (max)

Figura 134: señal bit_clk del códec

Reset:

El códec de audio LM4550 genera dos reset distintos. El "cold reset" que se genera cuando se produce un nivel bajo en el pin 11 con una duración superior a 1us. Esto produce un restablecimiento de los valores por defecto de todos los registros internos del códec y borra los modos ATE and Vendor Test. El diagrama de tiempos del reset frio se muestra en la Figura 135.

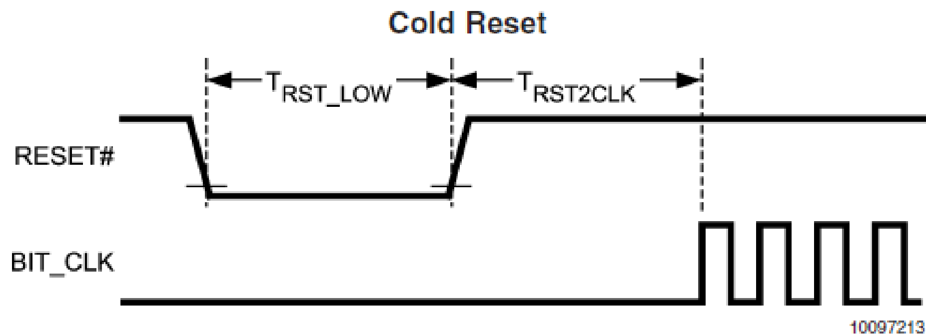


Figura 135: señal de "cold reset" del códec

El "warm reset" se produce cuando se da un nivel alto en el pin 10 (señal SYNC) durante un periodo mayor a 1us y además la interfaz digital AC Link del códec se encuentra desconectada (PR4 = 1, Powerdown Control / Status register, 26h). Este reset es utilizado para restablecer

PR4 y habilitar la interfaz digital AC Link. Sin embargo, no modifica ningún otro registro ni lógica del códec.

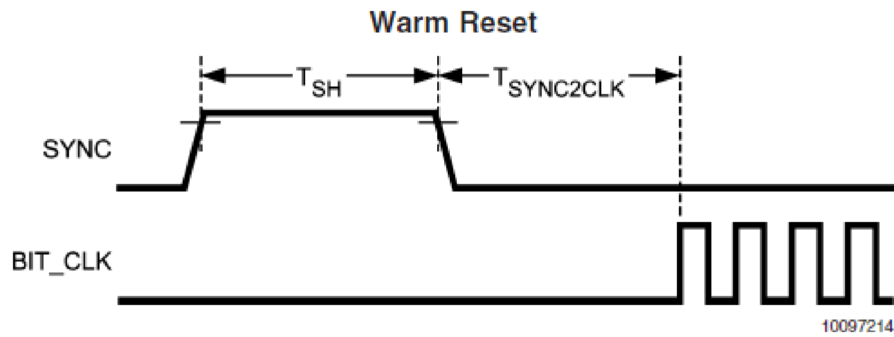
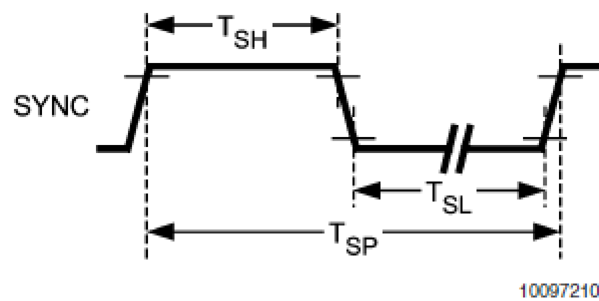


Figura 136: señal de "warm reset" del códec

T_{RST_LOW}	RESET# active low pulse width	For Cold Reset		1.0	μs (min)
$T_{RST2CLK}$	RESET# inactive to BIT_CLK start up	For Cold Reset	TBD	162.8	ns (min)
T_{SH}	SYNC active high pulse width	For Warm Reset	1.3	TBD	μs (min)
$T_{SYNC2CLK}$	SYNC inactive to BIT_CLK start up	For Warm Reset	TBD	162.8	ns (min)

SYNC



F_{SYNC}	SYNC frequency		48		kHz
T_{SP}	SYNC period		20.8		μs
T_{SH}	SYNC high pulse width		1.3		μs
T_{SL}	SYNC low pulse width		19.5		μs

Figura 137: señal SYNC del códec

La señal ***SYNC*** permanece a nivel alto una duración de 1.3us, esto es un total de 16 pulsos de bit_clk.

Trama

Con cada flanco de subida en la señal SYNC se envía una nueva trama de salida al códec de audio. Cada trama está compuesta por varios slots de 20 bits mandando un total de 256 bits,

siendo el primer slot en enviar de 16 bits (Figura 138). Este primer slot 0 es el encargado de validar la trama, llamado Tag Phase. El primer bit recibido es el llamado “Valid Frame”. Si el bit es 1 indica que al menos uno de los slots pertenecientes a la trama es válido.

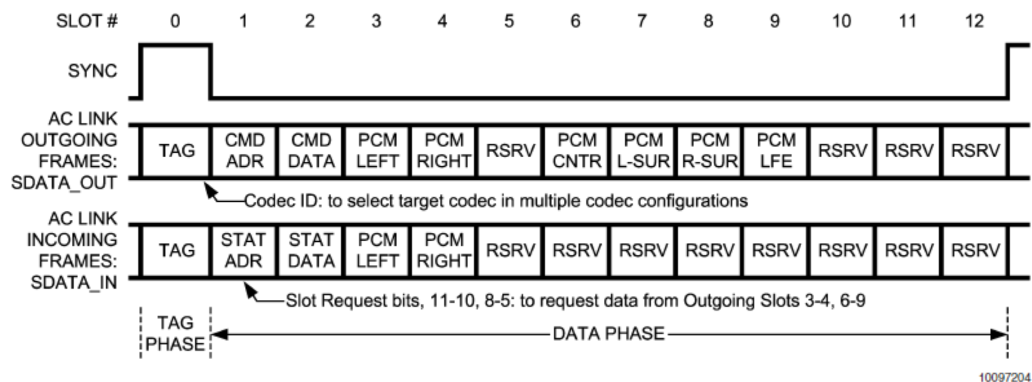


Figura 138: trama a enviar para la configuración del códec de audio LM4550

El códec de audio solo puede recibir datos de cuatro slots pertenecientes a una trama por lo que solo puede validar 4 Slots, esto es debido a que es un códec de dos canales. Para modo primario los Slots a validar en el Slot 0 son los siguientes:

bit	description	comment
15	Valid frame	1 = Valid data in at least one slot.
14	Control register address	1 = Valid Control Address in Slot 1 (Primary codec only)
13	Control register data	1 = Valid Control Data in Slot 2 (Primary codec only)
12	Left DAC data in slot 3	1 = Valid PCM Data in Slot 3 (Primary & Secondary 1 modes; Left Channel audio)
11	Right DAC data in slot 4	1 = Valid PCM Data in Slot 4 (Primary & Secondary 1 modes; Right Channel audio)
10:0	Not Used	Secondary mode

El Slot 1 tiene la función de controlador. Indica el registro con el cual el códec tiene que operar y se va a realizar una operación de lectura o escritura. Los bits del Slot 1 se describen de la siguiente forma:

bit	description	comment
19	Read/Write	1 = Read 0 = Write
18:12	Register address	Identifies the Status/Command register for read/write
11:0	Reserved	controller should set to "0"

Cuando se realiza la operación de escritura de un registro el Slot 2 es utilizado para transmitir los 16 bits de datos de control del registro en el códec de audio. El Slot 2 es descrito a continuación:

bit	description	comment
19:4	Control register write data	controller should stuff with zeros if operation is "read"
3:0	Reserved	Set to "0"

Por último, los Slots 3 y 4 son los encargados de mandar los datos de audio a reproducir a los canales derecho e izquierdo.

bit	description	comment
19:0	PCM audio data (Left/Right channels)	Slots used to stream data to DAC when codec is in Primary or Secondary 1 modes. Set unused bits to "0"

Los Slots 5, 10, 11 y 12 no son usados por el códec de audio LM4550 y el resto son usados para el modo Secundario, por lo que se deben rellenar con "0".

Registros

El sistema a diseñar utilizará únicamente la salida analógica del códec por lo que los registros a configurar serán PCM Out Volume, General Purpose y Master Volume (18h, 20h y 02h respectivamente). Estos registros se puede observar en la Figura 139 como son los encargados de configurar el códec para la salida LINE_OUT.

Block Diagram

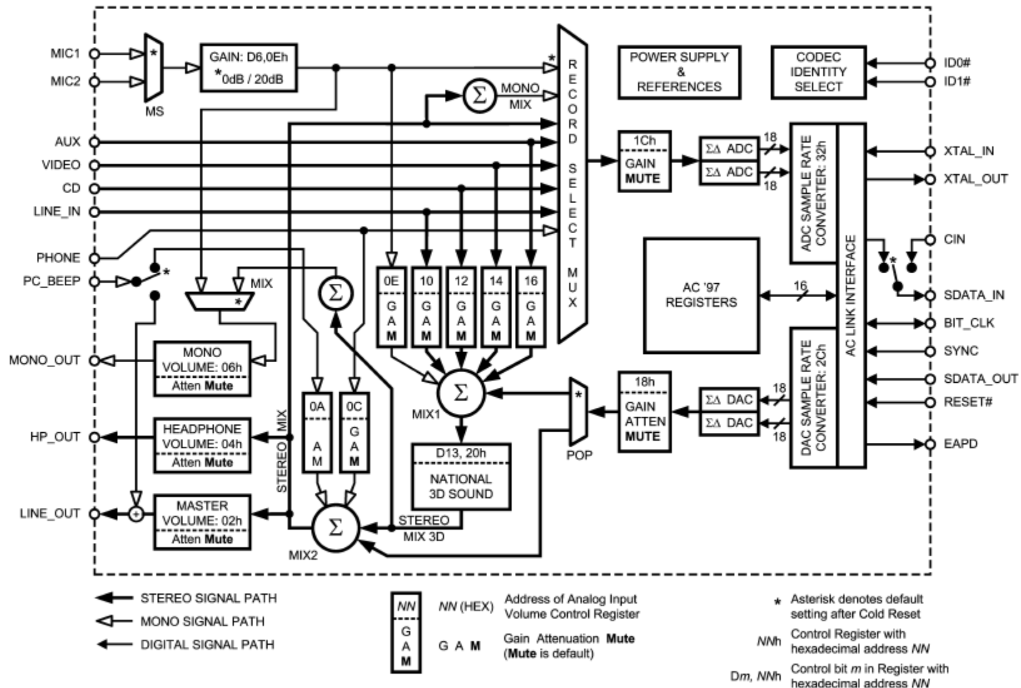


Figura 139: diagrama de bloques del códec de audio LM4550

Cada registro está compuesto por 16 bits encargados de la configuración del códec, el mapa de registros de puede ver en la Figura 140.

LM4550 Register Map

	REG	Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	Default
	00h	Reset	X	0	0	0	1	1	0	1	0	1	0	1	0	0	0	0	0D50h
Output Volume	02h	Master Volume	Mute	X	X	ML4	ML3	ML2	ML1	ML0	X	X	X	MR4	MR3	MR2	MR1	MR0	8000h
	04h	Headphone Volume	Mute	X	X	ML4	ML3	ML2	ML1	ML0	X	X	X	MR4	MR3	MR2	MR1	MR0	8000h
	06h	Mono Volume	Mute	X	X	X	X	X	X	X	X	X	X	MM4	MM3	MM2	MM1	MM0	8000h
Input Volume	0Ah	PC_Beep Volume	Mute	X	X	X	X	X	X	X	X	X	X	PV3	PV2	PV1	PV0	X	0000h
	0Ch	Phone Volume	Mute	X	X	X	X	X	X	X	X	X	X	GN4	GN3	GN2	GN1	GN0	8008h
	0Eh	Mic Volume	Mute	X	X	X	X	X	X	X	X	20dB	X	GN4	GN3	GN2	GN1	GN0	8008h
	10h	Line In Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	8808h
	12h	CD Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	8808h
	14h	Video Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	8808h
	16h	Aux Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	8808h
	18h	PCM Out Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	8808h
ADC Sources	1Ah	Record Select	X	X	X	X	X	SL2	SL1	SL0	X	X	X	X	X	SR2	SR1	SR0	0000h
	1Ch	Record Gain	Mute	X	X	X	GL3	GL2	GL1	GL0	X	X	X	X	GR3	GR2	GR1	GR0	8000h
	20h	General Purpose	POP	X	3D	X	X	X	MIX	MS	LPBK	X	X	X	X	X	X	X	0000h
	22h	3D Control (Read Only)	X	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0101h
X	24h	Reserved	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0000h
	26h	Powerdown Ctrl/Stat	EAPD	PR6	PR5	PR4	PR3	PR2	PR1	PR0	X	X	X	X	REF	ANL	DAC	ADC	000Xh
	28h	Extended Audio ID	ID1	ID0	X	X	X	X	AMAP	0	0	0	X	X	0	X	0	VRA	X201h
	2Ah	Extended Audio Control/Status	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	VRA	0000h
	2Ch	PCM DAC Rate	SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0	BB80h
	32h	PCM ADC Rate	SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0	BB80h
X	5Ah	Vendor Reserved 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0000h
	74h	Chain-In Control	X	X	X	X	X	X	X	X	X	X	X	X	X	X	ID1	ID0	000Xh
X	7Ah	Vendor Reserved 2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0000h
	7Ch	Vendor ID1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	4E53h
	7Eh	Vendor ID2	0	1	0	0	0	0	1	1	0	1	0	1	0	0	0	0	4350h

Figura 140: mapa de registros del códec de audio LM4550

A continuación se explicarán cada uno de los registros a configurar empezando por el registro Master Volume (02h). El registro Master Volume es el encargado de modificar el nivel de salida del canal estéreo LINE_OUT. Este puede ser silenciado o atenuado en un rango de 0 dB a 46.5 dB. Cada canal estéreo es controlado por 5 bits de manera individual, actuando el bit mute para ambos simultáneamente.

Mute	Mx4:Mx0	Function
0	0 0000	0dB attenuation
0	1 1111	46.5dB attenuation
1	X XXXX	*mute
Default: 8000h		

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Mute	0	0	ML4	ML3	ML2	ML1	ML0	0	0	0	MR4	MR3	MR2	MR1	MR0

El valor de MLx y MRx corresponden con el valor del volumen que se desea, siendo para este diseño el mismo en ambos canales estéreos.

Los registros de volumen de entrada del mezclador (0Ch-18h) los cuales ajustan los niveles de volumen en los mezcladores MIx1 y MIX2, que pueden ser ajustados en un rango de 12dB a 34.5dB. Ambos puertos serán silenciados si se pone el MSB a 1.

Mute	Gx4:Gx0	Function
0	0 0000	+12dB gain
0	0 1000	0dB gain
0	1 1111	34.5dB attenuation
1	X XXXX	*mute
Default:	8008h (mono registers) 8808h (stereo registers)	

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Mute	0	0	GL4	GL3	GL2	GL1	GL0	0	0	0	GR4	GR3	GR2	GR1	GR0

El registro General Purpose (20h) controla diversas de las funciones que se pueden implementar con el códec de audio LM4550, las cuales se muestran a continuación.

BIT	Function		
POP	PCM Out Path:	*0 =	3D allowed
		1 =	3D bypassed
3D	National 3D Sound:	*0 =	off
		1 =	on
MIX	Mono output select:	*0 =	Mix
		1 =	Mic
MS	Mic select:	*0 =	MIC1
		1 =	MIC2
LPBK	ADC/DAC Loopback:	*0 =	No Loopback
		1 =	Loopback
Default: 0000h			

D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
POP	ST	3D	LD	0	0	MIX	MS	LPBK	0	0	0	0	0	0	0

ANEXO II: Código

- Ejemplo ILA: Contador

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cont_32bits is
  port (
    rst          : in  std_logic;
    clk          : in  std_logic;
    ce           : in  std_logic;
    u_d          : in  std_logic;      --UP/DOWN
    tc           : out std_logic;      --FFF&UP || 000&DOWN
    ceo          : out std_logic;      --tc&ce
    counter_out  : out std_logic_vector(3 downto 0));
end cont_32bits;


architecture for_cont of cont_32bits is

  signal counter : unsigned(3 downto 0);
  signal fin : std_logic;
  signal enable : std_logic;
  signal ceo_aux : std_logic;
  signal up_down : std_logic;


  -- chipscope signals
  signal control0 :       std_logic_vector(35 downto 0);
  signal ila_data :       std_logic_vector(7 downto 0);
  signal trig0    :       std_logic_vector(3 downto 0);

begin

---------- chipscope-------------------------------------------

  my_icon:entity work.ICON
    port map (
          CONTROL0 => control0);

  my_ila:entity work.ILA
    port map (
         CONTROL => control0,
         CLK     => clk,
         DATA    => ila_data,
         TRIG0   => trig0);

---------------------------------------------------------------

  process (clk, rst)
  begin  -- process
    if rst = '1' then
        counter      <= (others => '0');
    elsif clk'event and clk = '1' then
      if ce = '1' then
        if u_d = '1' then
          counter <= counter+1;
```

```
        else
          counter <= counter-1;
        end if;
      end if;
    end if;
  end process;

  counter_out <= std_logic_vector(counter);

  process ( u_d, counter)
  begin  -- process
    if u_d = '1' then
       if counter = x"F" then
          fin <= '1';
       else
          fin <= '0';
       end if;
    else
       if counter = x"0" then
          fin <= '1';
       else
          fin <= '0';
       end if;
    end if;
  end process;

  ceo_aux <= '1' when (fin='1' and ce='1') else '0';

  tc      <= fin;
  ceo     <= ceo_aux;
  enable  <= ce;
  up_down <= u_d;

  ila_data(3 downto 0)  <= std_logic_vector(counter);
  ila_data(4)           <= enable;
  ila_data(5)           <= up_down;
  ila_data(6)           <= fin;
  ila_data(7)           <= ceo_aux;

  trig0(0)           <= enable;
  trig0(3 downto 1)  <= (others => '0');

end for_cont;
```

- ## Ejemplo VIO: ALU

```
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;

ENTITY alu IS
PORT(
 clk          : in  std_logic;
-- op1 : IN std_logic_vector(7 DOWNTO 0);--entrada 1
-- op2 : IN std_logic_vector(7 DOWNTO 0);--entrada 2
proceso : IN std_logic_vector(1 DOWNTO 0);--qué hará la alu
 res : OUT std_logic_vector(8 DOWNTO 0));
END alu;

ARCHITECTURE synth OF alu IS

 signal op1 :  std_logic_vector(7 DOWNTO 0);--entrada
 SIGNAL a,b:UNSIGNED(op1'range);
SIGNAL c:UNSIGNED(res'range);

   -- chipscope signals
signal control0 :       std_logic_vector(35 downto 0);
signal control1 :       std_logic_vector(35 downto 0);
signal ila_data :       std_logic_vector(26 downto 0);
signal trig0    :       std_logic_vector(1 downto 0);
signal vio_out  :       std_logic_vector(15 downto 0);

signal proc       :         std_logic_vector(1 DOWNTO 0);

    BEGIN
 ----------------------- chipscope----------------------------------
-----
my_icon:entity work.ICON
  port map (
       CONTROL0 => control0,
       CONTROL1 => control1);

my_ila:entity work.ILA
  port map (
       CONTROL => control0,
       CLK     => clk,
       DATA    => ila_data,
       TRIG0   => trig0);

my_vio:entity work.VIO
  port map (
      CONTROL  => control1,
     ASYNC_OUT => vio_out);
  ------------------------------------------------------------------
    PROCESS (a, b, proceso)
     BEGIN
       CASE proceso IS
         WHEN "00" => c <= RESIZE((a + b),c'length);
         WHEN "01" => c <= RESIZE((a - b),c'length);
         WHEN "10" => c <= RESIZE((a OR b),c'length);
         WHEN "11" => c <= RESIZE((a AND b),c'length);
         WHEN OTHERS => null;
       END CASE;
     END PROCESS;
```

```
     -- a <= UNSIGNED(op1);
     -- b <= UNSIGNED(op2);
      a <= UNSIGNED(vio_out(7 downto 0));
      b <= UNSIGNED(vio_out(15 downto 8));
      res <= std_logic_vector(c);

   proc <= proceso;
 ila_data(7 downto 0)   <= std_logic_vector(a);
 ila_data(15 downto 8)  <= std_logic_vector(b);
 ila_data(24 downto 16) <= std_logic_vector(c);
 ila_data(26 downto 25) <=proc;

 trig0(1 downto 0) <= proc;


 END synth;
```

- ### Ejemplo Core Inserter: Registro

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity registro is

  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    ce          : in  std_logic;
    data_out    : out std_logic_vector(4 downto 0));

end registro;

architecture fun_reg_des of registro is

  constant N          : integer := data_out'length;
  signal data_out_i : std_logic_vector(4 downto 0);
  signal counter     : std_logic;
  signal data_in     : std_logic;

begin

process (clk, rst)
  begin  -- process
    if rst = '1' then
      counter <= '0';
    elsif clk'event and clk = '1' then
      counter <= not counter;
    end if;
  end process;

data_in <= counter;

  process (clk, rst)
  begin  -- process
    if rst = '1' then
      data_out_i<=(others=>'0');
    elsif clk'event and clk = '1' then
      if ce='1' then
       data_out_i<=data_out_i(N-2 downto 0) & data_in;
      end if;
    end if;
  end process;

  data_out <= data_out_i;

end fun_reg_des;
```

- ## Semáforo

En la depuracion del semáforo se ha optado por la inserción independiente de los núcleos de ChipScope dentro de cada módulo. Como no es posible insertar más de un núcleo ICON por diseño, el código empleado para insertar la herramienta Chipscope está comentado.

Main file:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity state_chart is
    Port ( clk          : in  STD_LOGIC;
           rst          : in  STD_LOGIC;
           PSH_BUTTON   : in  STD_LOGIC;
           shifter_out  : out std_logic_vector(7 downto 0));
end state_chart;

architecture Behavioral of state_chart is

  type estadoFSM is (espera, contar);
  signal estado, prox_est : estadoFSM;

  signal counter          : unsigned(2 downto 0);
  signal pres_out         : std_logic;
  signal final            : std_logic;
  signal flanco           : std_logic;
  signal cs               : std_logic;
  signal shf_aux          : std_logic_vector(7 downto 0);
  signal clk_sys          : std_logic;
  signal psh              : std_logic;

   -- using ChipScope signals
  signal control0 : std_logic_vector(35 downto 0);
  signal ila_data : std_logic_vector(15 downto 0);
  signal trig0    : std_logic_vector(3 downto 0);
  signal trig1    : std_logic_vector(3 downto 0);

begin

 my_prescaler:entity work.prescaler
   port map (
    clk      => clk,
    rst      => rst,
    pre_out  => pres_out);

 my_contador:entity work.contador
   port map (
    clk     => clk,
    rst     => rst,
    cs      => cs,
    pre_out => pres_out,
    fn      => final,
    count   => counter);

 my_edge_detector:entity work.edge_detector
   port map (
    clk => clk,
    rst => rst,
```

```
    button => PSH_BUTTON,
    r_edge => flanco);
----------------- chipscope ---------------------------------
 my_icon:entity work.ICON
   port map (
    CONTROL0 => control0);

 my_ila:entity work.ILA
    port map (
     CONTROL => control0,
     CLK     => clk,
     DATA    => ila_data,
     TRIG0   => trig0,
     TRIG1   => trig1);
-------------------------------------------------------------
     clk_sys <= clk;
     psh     <= PSH_BUTTON;

-----------------------------------------------------------------------
------------------------- MAQUINA DE ESTADOS -------------------------
-----------------------------------------------------------------------
  process (counter, final, flanco , estado)

  begin  -- process
    case estado is
      when espera =>
        if   flanco  = '1' then
          prox_est <= contar;
          cs       <= '1';
        else
          prox_est <= espera;
          cs       <= '0';
        end if;
      when contar =>
        --if counter = "101" then -- original code
        if (final = '1' and counter = "101" )then --Modified code
          prox_est <= espera;
          cs       <= '0';
        else
          prox_est <= contar;
          cs       <= '1';
        end if;
    end case;
  end process;

  process (clk, rst)
  begin  -- process
    if rst = '0' then
      estado   <= espera;
    elsif clk'event and clk = '1' then
        estado <= prox_est;
    end if;
  end process;

----------------------------------------------------------------
        -- encender leds--  0RAG_00RG(C/P)
 ----------------------------------------------------------------

  process (counter)
  begin  -- process
    case counter is
```

```
      when "000"  => shf_aux <= "00010010";--cg+pr--12
      when "001"  => shf_aux <= "00100010";--ca+pr--22
      when "010"  => shf_aux <= "00100001";--ca+pg--21
      when "011"  => shf_aux <= "01000001";--cr+pg--41
      when "100"  => shf_aux <= "01000010";--cr+pr--42
      when "101"  => shf_aux <= "00010010";--cg+pr--12
      when "110"  => shf_aux <= "00010010";--cg+pr--12
      when "111"  => shf_aux <= "00010010";--cg+pr--12
      when others => null;
    end case;

  end process;

   shifter_out <= shf_aux;

    ila_data(0)          <= clk_sys;
    ila_data(1)          <= pres_out;
    ila_data(4 downto 2)  <= std_logic_vector(counter);
    ila_data(12 downto 5) <= shf_aux;
    ila_data(13)         <= cs;
    ila_data(14)         <= psh;
    ila_data(15)         <= flanco;

    trig0(0) <= psh;
    trig0(3 downto 1) <= (others => '0');
    trig1(0) <= cs;
    trig1(3 downto 1) <= (others => '0');

end Behavioral;
```

Prescaler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity prescaler is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           pre_out : out  STD_LOGIC);
end prescaler;

architecture Behavioral of prescaler is

 -- constant CLKDIV       : integer := 50e6; --4Hz=0.25sg
  constant CLKDIV       : integer :=5; --50nsg --simulacion
  signal counter_reg    : integer range 0 to CLKDIV-1;


------------------------------------------------------------
-- signal clk_sys        : std_logic;
-- signal psh                        : std_logic;

--  signal control0 : std_logic_vector(35 downto 0);
--  signal ila_data : std_logic_vector(15 downto 0);
--  signal trig0  : std_logic_vector(3 downto 0);


begin

--  my_icon:entity work.ICON
```

```
--    port map (
--   CONTROL0 => control0);
--
--   my_ila:entity work.ILA
--    port map (
--   CONTROL => control0,
--    CLK => clk,
--    DATA => ila_data,
--    TRIG0 => trig0);

---------------------------- prescaler------------------------------
--------------------------------------------------------------------
 process (clk,rst)
  begin  -- process
    if rst = '0'      counter_reg   <= 0;
    elsif clk'event and clk = '1' then
      if counter_reg = CLKDIV-1 then
        counter_reg <= 0;
      else
        counter_reg <= counter_reg+1;
      end if;
    end if;
  end process;

    pre_out   <= '1' when  counter_reg = CLKDIV-1 else '0';

 ------chipscope

--  psh <= '1' when counter_reg = "00000000" else '0';
--  clk_sys <= clk;
--
--
--  ila_data(0) <= clk_sys;
--  ila_data(8 downto 1) <= std_logic_vector(counter_reg);
--  ila_data(9) <= psh;
--  ila_data(15 downto 10) <= (others => '0');
--
--  trig0(0) <= clk_sys;
--  trig0(3 downto 1) <= (others => '0');

end Behavioral;
```

Contador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity contador is
    Port ( clk     : in  STD_LOGIC;
           rst     : in  STD_LOGIC;
           cs      : in  STD_LOGIC;
           pre_out : in  STD_LOGIC;
           fn      : out  STD_LOGIC;
           count   : out  unsigned(2 downto 0));
end contador;

architecture Behavioral of contador is

  signal counter         : unsigned(2 downto 0);
  signal fin         : std_logic;
```

```
--  signal countt           : unsigned(2 downto 0);
--  signal clk_sys          : std_logic;
--  signal pre_in           : std_logic;
--  signal cs_in            : std_logic;
--  signal control0         : std_logic_vector(35 downto 0);
--  signal ila_data         : std_logic_vector(15 downto 0);
--  signal trig0            : std_logic_vector(3 downto 0);

begin
--
--  my_icon:entity work.ICON
--    port map (
--          CONTROL0 => control0);
--
--   my_ila:entity work.ILA
--    port map (
--          CONTROL => control0,
--          CLK => clk,
--      DATA => ila_data,
--      TRIG0 => trig0);
 ---------------------------------------------------
               -- contador --
 ---------------------------------------------------
  process (clk, rst)
    begin  -- process

  if rst = '0' then
      counter     <= (others => '0');
  elsif (clk'event and clk = '1') then
      if cs = '1' then  --si se ha apretado el boton
         if pre_out = '1' then
           counter <= counter+1;
         end if;
      else
         counter <= (others => '0');
      end if;
   end if;
  end process;

fin   <= '1' when counter = "101" else '0';
count <= counter;
fn    <= fin;

--  countt <= counter;
--  clk_sys <= clk;
--  cs_in <= cs;
--  pre_in <= pre_out;
--
--  ila_data(0) <= clk_sys;
--  ila_data(1) <= cs_in;
--  ila_data(2) <= pre_in;
--  ila_data(3) <= fin;
--  ila_data(6 downto 4) <= std_logic_vector(countt);
--  ila_data(15 downto 7) <= (others => '0');
--
--  trig0(0) <= cs;
--  trig0(3 downto 1) <= (others => '0');

end Behavioral;
```

Edge_detector

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity edge_detector is
    Port ( clk    : in  STD_LOGIC;
           rst    : in  STD_LOGIC;
           button : in  STD_LOGIC;
           r_edge : out  STD_LOGIC);
end edge_detector;

architecture Behavioral of edge_detector is

  signal Q_add           : std_logic;
--  signal psh           : std_logic;
--  signal edge          : std_logic;
--  signal f_edge        : std_logic;
--     -- using ChipScope signals
--  signal control0 : std_logic_vector(35 downto 0);
--  signal ila_data : std_logic_vector(15 downto 0);
--  signal trig0    : std_logic_vector(3 downto 0);

begin

--  my_icon:entity work.ICON
--    port map (
--   CONTROL0 => control0);
--
--   my_ila:entity work.ILA
--    port map (
--   CONTROL => control0,
--    CLK => clk,
--    DATA => ila_data,
--    TRIG0 => trig0);

  process (clk, rst) --biestable D
  begin
    if rst = '0' then
      Q_add <= '0';
    elsif clk'event and clk = '1' then
      Q_add <= button;
    end if;
  end process;

     r_edge <= button and (not Q_add) ;

--   psh <=button;
--   edge <= button and (not Q_add) ;
--
--  ila_data(0) <= Q_add;
--  ila_data(1) <= psh;
--  ila_data(2) <= edge;
--  ila_data(3) <= f_edge;
--  ila_data(15 downto 4) <= (others => '0');
--
--  trig0(0) <= psh;
--  trig0(3 downto 1) <= (others => '0');

end Behavioral;
```

- **Epp_controller**

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

entity cnt_epp is
  port (
    CLK      : in    std_logic;
    ila_clk  : in    std_logic;
    RST      : in    std_logic;
    ASTRB    : in    std_logic;
    DSTRB    : in    std_logic;
    DATA     : inout std_logic_vector(7 downto 0);
    PWRITE   : in    std_logic;
    PWAIT    : out   std_logic;
    DATO_RD  : in    std_logic_vector(7 downto 0);
    CE_RD    : out   std_logic;
    DIR      : out   std_logic_vector (7 downto 0);
    DIR_VLD  : out   std_logic;
    DATO     : out   std_logic_vector (7 downto 0);
    DATO_VLD : out   std_logic);
end;

architecture rtl of cnt_epp is

---------------------- ChipScope  -----------------------------------
component icon
  PORT (
    CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
end component;

component ila
  PORT (
    CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
    CLK     : IN STD_LOGIC;
    DATA    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    TRIG0   : IN STD_LOGIC_VECTOR(7 DOWNTO 0));
end component;

  signal control0 :       std_logic_vector(35 downto 0);
  signal ila_data :       std_logic_vector(11 downto 0);
  signal trig0    :       std_logic_vector(7 downto 0);
  -------------------------------------------------------------------

  signal Q_add, ce_add, Q_dat, ce_dat : std_logic;
  signal ce_rd_aux                    : std_logic;

begin

 sys_icon : icon
   port map (
    CONTROL0 => control0);


 sys_ila : ila
   port map (
    CONTROL => control0,
    CLK     => ila_clk,
    DATA    => ila_data,
    TRIG0   => trig0);
```

```vhdl
---------------------------------------------------------------------

  PWAIT <= (not ASTRB) or (not DSTRB);

-- detector de flanco de subida de ASTRB
  process (RST, CLK) --biestable D
  begin
    if RST = '1' then
      Q_add <= '1';
    elsif (CLK'event and CLK = '1') then
      Q_add <= ASTRB;
    end if;
  end process;

--cuando PWRITE indique escritura(0) y haya un flanco subida en ASTRB
--> ce_add=1
  ce_add <= ASTRB and (not Q_add) when PWRITE = '0' else '0'; --enable
address

-- si hay un flanco de subida en ASTRB entonces validamos direccion y
metemos la se□DATA,
-- que sera la direccion, en DIR
  process (RST, CLK)
  begin
    if RST = '1' then
      DIR_VLD    <= '0';
      DIR        <= (others => '0');
    elsif (CLK'event and CLK = '1') then
      if ce_add = '1' then
        DIR_VLD <= '1';
        DIR     <= DATA;
      else
        DIR_VLD <= '0';
      end if;

    end if;
  end process;

--detector flanco de subida DSTRB
  process (RST, CLK) --biestable D
  begin
    if RST = '1' then
      Q_dat <= '1';
    elsif (CLK'event and CLK = '1') then
      Q_dat <= DSTRB;
    end if;
  end process;

--cuando PWRITE indique escritura(0) y haya un flanco subida en DSTRB
--> ce_dat=1
  ce_dat <= DSTRB and (not Q_dat) when PWRITE = '0' else '0'; --enable
data

-- si hay un flanco de subida en DSTRB entonces validamos el dato a
escribir y metemos la se□DATA,
-- que sera el dato que queremos escribir, en DATO
  process (RST, CLK)
  begin
    if RST = '1' then
      DATO_VLD    <= '0';
      DATO        <= (others => '0');
```

```
    elsif (CLK'event and CLK = '1') then
      if ce_dat = '1' then
        DATO_VLD <= '1';
        DATO     <= DATA;
      else
        DATO_VLD <= '0';
      end if;
    end if;
  end process;

--cuando DSTRB este a nivel bajo y PWRITE indique lectura(1) se
habilitara lectura de datos
--el dato leido de la FPGA (DATO_RD) pasara a DATA para ser leido por
PC
  ce_rd_aux <= '1'     when DSTRB = '0' and PWRITE = '1' else '0';
  CE_RD     <= ce_rd_aux;
  DATA      <= DATO_RD when ce_rd_aux = '1' else (others => 'Z');

  ila_data(0) <= ASTRB;
  ila_data(1) <= DSTRB;
  ila_data(2) <= PWRITE;
  ila_data(10 downto 3) <= DATA;
  ila_data(11) <= '0';
  trig0(0) <= ASTRB;
  trig0(1) <= DSTRB;
  trig0(2) <= PWRITE;
  trig0(7 downto 3) <= (others => '0');

end rtl;
```

- DECODER_EPP

Main

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;

entity main is
   port (
     CLK        : in   std_logic;
     RST        : in   std_logic;
     CODE       : out  std_logic_vector(7 downto 0));
end main;

architecture Behavioral of main is

        -- chipscope signals
  signal control0     : std_logic_vector(35 downto 0);
  signal control1     : std_logic_vector(35 downto 0);
  signal ila_data     : std_logic_vector(20 downto 0);
  signal trig0        : std_logic_vector(0 to 0);
  signal vio_out      : std_logic_vector(15 downto 0);
  signal vio_syn_out  : std_logic_vector(0 to 0);

  signal addr         : std_logic_vector(7 downto 0);
  signal data         : std_logic_vector(7 downto 0);
  signal valid_data   : std_logic;
  signal valid_freq   : std_logic;
  signal valid_dur    : std_logic;
  signal rest         : std_logic;
  signal str          : std_logic;
  signal ply          : std_logic;
  signal volume       : std_logic_vector(4 downto 0);
  signal code_out     : std_logic_vector(7 downto 0);

begin

my_decoder:entity work.decoder_epp

port map(
     CLK        => CLK,
     RST        => RST,
     DIR        => addr,
     DATO       => data,
     DATOS_VLD  => valid_data,
     RESTART    => rest,
     START      => str,
     PLAY       => ply,
     FREC_VLD   => valid_freq,
     DUR_VLD    => valid_dur,
     VOL_CODE   => volume,
     CODE       => code_out

);

   ----------------------- chipscope------------------------------

  my_icon:entity work.ICON
    port map (
```

```
          CONTROL0 => control0,
          CONTROL1 => control1);

  my_ila:entity work.ILA
    port map (
          CONTROL => control0,
          CLK     => CLK,
          DATA    => ila_data,
          TRIG0   => trig0);

  my_vio:entity work.VIO
    port map (
       CONTROL   => control1,
       CLK       => CLK,
      ASYNC_OUT => vio_out,
      SYNC_OUT  => vio_syn_out);
    -------------------------------------------------------------------
  addr <= vio_out(7 downto 0); --vio
  data <= vio_out(15 downto 8); --vio
  valid_data <= vio_syn_out(0); --vio

    CODE <= code_out;

  ila_data(0) <= rest;
  ila_data(1) <= str;
  ila_data(2) <= ply;
  ila_data(3) <= valid_freq;
  ila_data(4) <= valid_dur;
  ila_data(9 DOWNTO 5)   <= volume;
  ila_data(17 DOWNTO 10) <= code_out;
  ila_data(20 DOWNTO 18) <= (OTHERS =>'0');
  trig0(0) <= valid_data;

end Behavioral;
```

decoder_epp

```
-- decodifica los datos enviados por el protocolo EPP de la siguiente
manera:
    -- DIR=0x11; DATA=0x11 -> restart=pulso
    -- DIR=0x22; DATA=0x22 -> start=pulso
    -- DIR=0xAA; DATA=0xAA -> play=pulso
    -- DIR=0xF0; -> frec_vld=pulso
    -- DIR=0xD0; -> dur_vld=pulso
    -- DIR=0xB0; -> vol_code=dato

library ieee;
use ieee.std_logic_1164.all;

entity decoder_epp is

  port (
    CLK       : in  std_logic;
    RST       : in  std_logic;
    DIR       : in  std_logic_vector(7 downto 0);
    DATO      : in  std_logic_vector(7 downto 0);
    DATOS_VLD : in  std_logic;
    RESTART   : out std_logic;
    START     : out std_logic;
    PLAY      : out std_logic;
    FREC_VLD  : out std_logic;
```

```vhdl
    DUR_VLD    : out std_logic;
    VOL_CODE   : out std_logic_vector(4 downto 0);
    CODE       : out std_logic_vector(7 downto 0));

end decoder_epp;

architecture rtl of decoder_epp is

begin  -- RTL

  process (CLK, RST)
  begin
    if RST = '1' then
       RESTART   <= '0';
       START     <= '0';
       PLAY      <= '0';
       FREC_VLD  <= '0';
       DUR_VLD   <= '0';
       VOL_CODE  <= (others => '0');
    elsif (CLK'event and CLK = '1') then
      if DATOS_VLD = '1' and DIR = x"11" and DATO = x"11"  then
        RESTART <= '1';
      elsif DATOS_VLD = '1' and DIR = x"22" and DATO = x"22"  then
        START<= '1';
      elsif DATOS_VLD = '1' and DIR = x"AA" and DATO = x"AA"  then
        PLAY<= '1';
      elsif DATOS_VLD = '1' and DIR = x"F0"  then
        FREC_VLD<= '1';
      elsif DATOS_VLD = '1' and DIR = x"D0"  then
        DUR_VLD<= '1';
      elsif DATOS_VLD = '1' and DIR = x"B0"  then
        VOL_CODE <= DATO(VOL_CODE'range);
      else
        RESTART <= '0';
        START   <= '0';
        PLAY    <= '0';
        FREC_VLD<= '0';
        DUR_VLD <= '0';
      end if;
    end if;
  end process;

   CODE<=DATO;


end rtl;
```

- **FIFO**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_system is
    Port ( clk_sys : in  STD_LOGIC;
           datos : out  std_logic_vector(7 downto 0));
end top_system;

architecture Behavioral of top_system is

 signal   rst_wr_i      :  std_logic;
 signal   rst_rd_i      :  std_logic;
 signal   wr_en_i       :  std_logic;
 signal   din_i         :  std_logic_vector(7 downto 0);
 signal   dout_i        :  std_logic_vector(7 downto 0);
 signal   rd_en_i       :  std_logic;
 signal   empty_i       :  std_logic;
 signal   full_i        :  std_logic;

--------------- chipscope signals--------------------------
  signal control0      :   std_logic_vector(35 downto 0);
  signal control1      :   std_logic_vector(35 downto 0);
  signal ila_data      :   std_logic_vector(19 downto 0);
  signal trig0         :   std_logic_vector(3 downto 0);
  signal trig1         :   std_logic_vector(3 downto 0);
  signal vio_out       :   std_logic_vector(3 downto 0);
  signal vio_out_syn   :   std_logic_vector(7 downto 0);
  signal vio_in        :   std_logic_vector(1 downto 0);

begin

  my_icon:entity work.ICON
    port map (
      CONTROL0 => control0,
      CONTROL1 => control1);

  my_ila:entity work.ILA
    port map (
         CONTROL => control0,
         CLK     => clk_sys,
         DATA    => ila_data,
         TRIG0   => trig0,
         TRIG1   => trig1);

  my_vio:entity work.VIO
    port map (
         CONTROL   => control1,
         CLK       => clk_sys,
         ASYNC_OUT => vio_out,
         SYNC_OUT  => vio_out_syn,
         ASYNC_IN  => vio_in);

  memo_FIFO : entity work.fifo
    port map (
        rst_wr     => rst_wr_i,
        rst_rd     => rst_rd_i,
        clk        => clk_sys,
        wr_en      => wr_en_i,
        din        => din_i,
```

```
        rd_en       => rd_en_i,
        dout        => dout_i,
        empty       => empty_i,
        full        => full_i);


    datos <= dout_i;

----------- asignacio chipscope---------------------------

  ila_data(0) <= rst_wr_i;
  ila_data(1) <= rst_rd_i;
  ila_data(2) <= wr_en_i;
  ila_data(3) <= rd_en_i;
  ila_data(11 downto 4) <= dout_i (7 downto 0);
  ila_data(19 downto 12) <= din_i(7 downto 0);

  trig0(0) <= wr_en_i;
  trig1(0) <= rd_en_i;
  trig0(3 downto 1) <= (others => '0');
  trig1(3 downto 1) <= (others => '0');

  din_i     <= vio_out_syn(7 downto 0);
  rst_wr_i  <= vio_out(0);
  rst_rd_i  <= vio_out(1);
  wr_en_i   <= vio_out(2);
  rd_en_i   <= vio_out(3);
  vio_in(0) <= empty_i;
  vio_in(1) <= full_i;

end Behavioral;
```

- Codec_controller

```
library ieee;
use ieee.std_logic_1164.all;


entity codec_controller is
  port (
    rst         : in  std_logic;
    clk         : in  std_logic;
    bit_clk     : in  std_logic;
    sync        : out std_logic;
    sdata_out   : out std_logic;
    reset       : out std_logic;
    leds        : out std_logic_vector(7 downto 0)
   );
end codec_controller;


architecture rtl of codec_controller is


  signal f_edge_b_clk     : std_logic;
  signal r_edge_b_clk_aux : std_logic;
  signal sync_counter_aux : std_logic_vector(7 downto 0);
  signal frec_code_aux    : std_logic_vector(3 downto 0);
  signal vol_code_aux     : std_logic_vector(4 downto 0);

  -- chipscope signals
  signal control0       :   std_logic_vector(35 downto 0);
  signal CONTROL_ila1   :   std_logic_vector(35 downto 0);
  signal CONTROL_ila2   :   std_logic_vector(35 downto 0);
  signal CONTROL_vio1   :   std_logic_vector(35 downto 0);
  signal CONTROL_vio    :   std_logic_vector(35 downto 0);
  signal ila_data       :   std_logic_vector(22 downto 0);
  signal trig0          :   std_logic_vector(7 downto 0);
  signal trig1          :   std_logic_vector(3 downto 0);
  signal vio_out        :   std_logic_vector(8 downto 0);
  signal trg_out        :   std_logic;



 signal reset_aux       : std_logic;
 signal bit_clk_aux     : std_logic;
 signal clk_aux         : std_logic;
 signal sync_aux        : std_logic;
 signal sdata_out_aux   : std_logic;


begin

    vol_code_aux    <= vio_out(4 downto 0);
    frec_code_aux   <= vio_out(8 downto 5);

--- chipscope-------------------------------------------

  my_icon:entity work.ICON
    port map (
          CONTROL0 => control0,
          CONTROL1 => CONTROL_ila1,
```

```
            CONTROL2 => CONTROL_vio1,
            CONTROL3 => CONTROL_ila2,
            CONTROL4 => CONTROL_vio);

  my_ila:entity work.ILA
    port map (
           CONTROL  => control0,
           CLK      => clk,
           DATA     => ila_data,
           TRIG0    => trig0,
           TRIG1    => trig1,
           TRIG_OUT => trg_out);

 my_vio:entity work.VIO1
    port map (
           CONTROL    => CONTROL_vio,
           ASYNC_OUT  => vio_out);
    ---------------------------------------------------------------
  double_edge : entity work.doble_edge_detector
    port map (
      rst          => rst,
      clk          => clk,
      bit_clk      => bit_clk,
      r_edge_b_clk => r_edge_b_clk_aux,
      f_edge_b_clk => f_edge_b_clk);

  rst_sync_g : entity work.rst_and_sync_generation
    port map (
      rst          => rst,
      clk          => clk,
      r_edge_b_clk => r_edge_b_clk_aux,
      reset        => reset_aux,
      sync         => sync_aux,
      sync_counter => sync_counter_aux,
      trg          => trg_out,
      control_ILA1 => CONTROL_ila1);

  frame : entity work.output_frame
    port map (
      clk          => clk,
      rst          => rst,
      r_edge_b_clk => r_edge_b_clk_aux,
      sync_counter => sync_counter_aux,
      sdata_out    => sdata_out_aux,
      vol_code     => vol_code_aux,
      frec_code    => frec_code_aux,
      control_ILA2 => CONTROL_ila2,
      control_VIO1 => CONTROL_vio1      );


LEDS<=vol_code_aux(4 downto 1)&frec_code_aux;

-------------------------------------------------------------------

sdata_out   <= sdata_out_aux;
reset       <= reset_aux;
sync        <= sync_aux;
clk_aux     <= clk;
bit_clk_aux <= bit_clk;
```

```
  ila_data(0)              <= reset_aux;
  ila_data(1)              <= sync_aux;
  ila_data(2)              <= sdata_out_aux;
  ila_data(3)              <= r_edge_b_clk_aux;
  ila_data(7 downto 4)     <= frec_code_aux(3 downto 0);
  ila_data(12 downto 8)    <= vol_code_aux (4 downto 0);
  ila_data(20 downto 13)   <= sync_counter_aux (7 downto 0);
  ila_data(21)             <= bit_clk_aux;
  ila_data(22)             <= clk_aux;

  trig0(7 downto 0)        <= sync_counter_aux (7 downto 0);
  trig1(0)                 <= r_edge_b_clk_aux;
  trig1(3 downto 1)        <= (others => '0');

end rtl;
```

- Rst_and_sync_generation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rst_and_sync_generation is
  port (
    rst          : in  std_logic;
    clk          : in  std_logic;
    r_edge_b_clk : in  std_logic;
    reset        : out std_logic;
    sync         : out std_logic;
    sync_counter : out std_logic_vector(7 downto 0);
    control_ILA1 : inout std_logic_vector(35 downto 0);
    trg          :   in    std_logic);

end rst_and_sync_generation;

architecture rtl of rst_and_sync_generation is

  signal cont : unsigned(7 downto 0);   -- contador de 255 pulsos para
el reset
  signal cont_rst : unsigned(7 downto 0);   -- contador de periodos de
sync

   -------------- chipscope signals ----------------------

  signal ila_data :       std_logic_vector(9 downto 0);
  signal trig0    :       std_logic_vector(0 to 0);
  signal trig1    :       std_logic_vector(0 to 0);

-------------------
signal r_edge_b_clk_aux : std_logic;
signal tr_out           : std_logic;
signal sync_aux         : std_logic;

begin  -- rtl

----------------------------------------------------------
  my_ila:entity work.ILA1
    port map (
    CONTROL => control_ILA1,
    CLK     => clk,
    DATA    => ila_data,
    TRIG0   => trig0,
    TRIG1   => trig1);

----------------------------------------------------------

   process (clk, rst)
  begin  -- process sync
    if rst = '1' then
      sync<='0'  ;
        sync_aux<='0'  ;
        cont<=x"e3";
    elsif clk'event and clk = '1' then
      if r_edge_b_clk='1' then --con el flanco de subida
        cont<=cont+1;
        if cont<14 or cont>=254 then
```

```
          sync <='1';
          sync_aux <='1';
        else
          sync <='0';
          sync_aux <='0';
        end if;
      end if;
    end if;
  end process ;

  sync_counter <= std_logic_vector(cont);

 process (clk, rst)
  begin  -- process reset
    if rst = '1' then
      reset<='0'  ;
      cont_rst<=(others=>'0');
    elsif clk'event and clk = '1' then
      if cont_rst<128 then
        reset <='0';
        cont_rst<=cont_rst+1;
      else
        reset <='1';
      end if;
    end if;
  end process ;

  ------------------------------------------------------------------
  ------------------------------------------------------------------
  r_edge_b_clk_aux <= r_edge_b_clk;
  tr_out <= trg;

  ila_data(0)<= r_edge_b_clk_aux;
  ila_data(8 downto 1) <= std_logic_vector(cont);
  ila_data(9)<= sync_aux;
  trig0(0) <= r_edge_b_clk_aux;
  trig1(0) <= tr_out;

end rtl;
```

- **Output_frame**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity output_frame is
  port (
    rst          : in  std_logic;
    clk          : in  std_logic;
    r_edge_b_clk : in  std_logic;
    vol_code     : in  std_logic_vector(4 downto 0);
    frec_code    : in  std_logic_vector(3 downto 0);
    sync_counter : in  std_logic_vector(7 downto 0);
    control_ILA2 : inout std_logic_vector(35 downto 0);
    control_VIO1 : inout std_logic_vector(35 downto 0);
    sdata_out    : out std_logic);
end output_frame;

architecture rtl of output_frame is

  component seno_core
    port (
      clka  : in  std_logic;
      addra : in  std_logic_vector(11 downto 0);
      douta : out std_logic_vector(19 downto 0));
  end component;

  signal DIR                : std_logic_vector(0 to 19);
  signal DATA               : std_logic_vector(0 to 19);
  type estadoFSM is (reg_20, reg_18, reg_02);
  signal estado, prox_est : estadoFSM;
  --
  signal mux_in             : std_logic_vector(0 to 255);
  --
  signal CLKDIV             : integer;
  signal counter_reg        : unsigned(7 downto 0);
  signal ce                 : std_logic;
  signal pre_out            : unsigned(11 downto 0);
  signal slot3              : std_logic_vector(0 to 19);
  signal SLOT3_4            : std_logic_vector(0 to 19);
  --

  signal ila_data           : std_logic_vector(59 downto 0);
  signal trig0              : std_logic_vector(0 to 0);
  signal trig1              : std_logic_vector(0 to 0);
  signal r_edge_b_clk_aux : std_logic;
  signal vio_sync_in        : std_logic_vector(3 downto 0);

begin  -- rtl
---------------------------------------------------------------------

  my_ila:entity work.ILA2
    port map (
          CONTROL => control_ILA2,
          CLK => ce,
          DATA => ila_data,
          TRIG0 => trig0,
          TRIG1 => trig1);
```

```
  my_VIO:entity work.VIO
    port map (
        CONTROL => control_VIO1,
        CLK     => clk,
        SYNC_IN => vio_sync_in );

--------------------------------------------------------------------
--      MAQUINA DE ESTADOS
-------------------------------------------------------------------
  process (sync_counter, estado, vol_code)

  begin  -- process
    case estado is
      when reg_20 =>
        if unsigned(sync_counter) = 200 then
          prox_est <= reg_18;
          DIR      <= x"18000";          --dir->reg 18
          DATA     <= x"08080";
        else
          prox_est <= reg_20;
          DIR      <= x"20000";          --dir->reg 20
          DATA     <= x"80000";
        end if;
      when reg_18 =>
        if unsigned(sync_counter) = 200 then
          prox_est <= reg_02;
          DIR      <= x"18000";          --dir->reg 02
          DATA     <= "000" & vol_code & "000" & vol_code & "0000";
        else
          prox_est <= reg_18;
          DIR      <= x"18000";          --dir->reg 18
          DATA     <= x"08080";
        end if;
      when reg_02 =>
        if unsigned(sync_counter) = 200 then
          prox_est <= reg_20;
          DIR      <= x"20000";          --dir->reg 20
          DATA     <= x"80000";
        else
          prox_est <= reg_02;
          DIR      <= x"02000";          --dir->reg 02
          DATA     <= "000" & vol_code & "000" & vol_code & "0000";
        end if;
    end case;
  end process;

  process (clk, rst)
  begin  -- process
    if rst = '1' then
      estado   <= reg_20;
    elsif clk'event and clk = '1' then
      if (r_edge_b_clk = '1')then
        estado <= prox_est;
      end if;
    end if;
  end process;


-------------------------------------------------------------------
```

```vhdl
-- REGISTRO
--------------------------------------------------------------------
  SLOT3_4       <= SLOT3(0 to 17) & "00";
  process (clk, rst)
    constant SLOT0 : std_logic_vector(0 to 15)  := x"f800";
    constant zeros : std_logic_vector(0 to 159) := (others => '0');

  begin  -- process
    if rst = '1' then
      mux_in <= (others => '0');
    elsif clk'event and clk = '1' then

      mux_in <= SLOT0 & DIR & DATA & SLOT3_4 & SLOT3_4 & zeros;
    end if;
  end process;

  --------------------------------------------------------------------
  --    MULTIPLEXOR
  -------------------------------------------------------------------

  process (mux_in, sync_counter)
  begin  -- process
    sdata_out <= mux_in(to_integer(unsigned(sync_counter)));
  end process;
----------------------------------------------------------------------
  -- conversor
----------------------------------------------------------------------
  process (frec_code)
  begin  -- process
    case frec_code is
      when "0000" => CLKDIV <= 245;
      when "0001" => CLKDIV <= 123;
      when "0010" => CLKDIV <= 82;
      when "0011" => CLKDIV <= 62;
      when "0100" => CLKDIV <= 49;
      when "0101" => CLKDIV <= 35;
      when "0110" => CLKDIV <= 25;
      when "0111" => CLKDIV <= 20;
      when "1000" => CLKDIV <= 15;
      when "1001" => CLKDIV <= 13;
      when "1010" => CLKDIV <= 10;
      when "1011" => CLKDIV <= 8;
      when "1100" => CLKDIV <= 7;
      when "1101" => CLKDIV <= 6;
      when "1110" => CLKDIV <= 5;
      when "1111" => CLKDIV <= 4;
      when others => CLKDIV <= 1;
    end case;
  end process;
```

```vhdl
--------------------------------------------------------------------
  -- prescaler
  --------------------------------------------------------------------

  process (clk, rst)
  begin  -- process
    if rst = '1' then
      counter_reg <= (others => '0');
    elsif clk'event and clk = '1' then
      if counter_reg = CLKDIV-1 then
        counter_reg <= (others => '0');
      else
        counter_reg <= counter_reg+1;
      end if;
    end if;
  end process;

  ce <= '1' when counter_reg = CLKDIV-1 else '0';

  --------------------------------------------------------------------
  -- contador
  --------------------------------------------------------------------

  process (clk, rst)

  begin  -- process
    if rst = '1' then
      pre_out   <= (others => '0');
    elsif clk'event and clk = '1' then
      if ce = '1' then
        pre_out <= pre_out+1;
      end if;
    end if;
  end process;

  --------------------------------------------------------------------
  -- memoria seno
  --------------------------------------------------------------------
  memo : seno_core
    port map (
      clka   => clk,
      addra => std_logic_vector(pre_out),
      douta => slot3 );

--------------------------------------------------------------------
----------------------------------------------------------------------

  r_edge_b_clk_aux <= r_edge_b_clk;

  ila_data(19 downto 0)  <= slot3;
  ila_data(39 downto 20) <= DIR;
  ila_data(59 downto 40) <= DATA;
  trig0(0) <= r_edge_b_clk_aux;
  trig1(0) <= ce;

  vio_sync_in(0) <= r_edge_b_clk_aux;
  vio_sync_in(3 downto 1)<=(others=>'0');

end rtl;
```

Bibliografía:

Maxfield, C.M. (2004) The design warrior's guide to FPGAs: Devices, tools, and flows. Boston: Newnes (an imprint of Butterworth-Heinemann Ltd).

Cheng, W.. and Patel, J.H. (1987) 'Testing in two-dimensional iterative logic arrays', Computers & Mathematics with Applications, 13(5-6), pp. 443–454. doi: 10.1016/0898-1221(87)90074-5.

Edward J. McCluskey and Chao-Wen Tseng . Stuck-Fault Tests vs. Actual Defects. Center for Reliable Computing Stanford University, Stanford, CA

http://crc.stanford.edu

July 05, 2005 20-152 - RTCA, Inc., Document RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware (https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentid/22211)

Xilinx, Inc. (2002) Xilinx ChipScope pro supports the industry's First integrated bus analyzer for programmable logic. Available at: http://newit.gsu.by/resources/Journals%5Cdacafe%5C2002_03%5C21556.htm (Accessed: February 2017).

O. Oltu, P. L. Milea, and A. Simion, "Testing of digital circuitry using Xilinx ChipScope logic analyzer," CAS 2005 Proceedings. 2005 International Semiconductor Conference, 2005., Dec. 2005. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1558829. Accessed: Feb. 2017.

Arshak, K., Jafer, E. and Ibala, C. (2006) 'Testing FPGA based digital system using XILINX ChipScope logic analyzer', 2006 29th International Spring Seminar on Electronics Technology, . doi: 10.1109/isse.2006.365129.

Skokan, Z.E. (1983) 'Programmable logic machine (A programmable cell array)', IEEE Journal of Solid-State Circuits, 18(5), pp. 572–578. doi: 10.1109/jssc.1983.1051996.

Aldec. DO-254 compliance - solutions. Available at: https://www.aldec.com/en/solutions/do_254_compliance (Accessed: February 2017).

paguayo (2014) FPGA (field programmable gate array). Available at: http://cursos.olimex.cl/fpga/ (Accessed: February 2017).

Universidad de Huelva. Available at:

http://www.uhu.es/rafael.lopezahumada/Cursos_anteriores/fund97_98/plds.pdf

Bozich, E.C. (2005) Introducción a los Dispositivos FPGA. Análisis y ejemplos de diseño. Universidad Nacional de la Plata

Available at: https://catedra.ing.unlp.edu.ar/electrotecnia/islyd/Trabajo%20Final.pdf (Accessed: February 2017).

Manual de Xilinx Ug750 (http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug750.pdf).

Manual de Xilinx Ug029 (http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf).

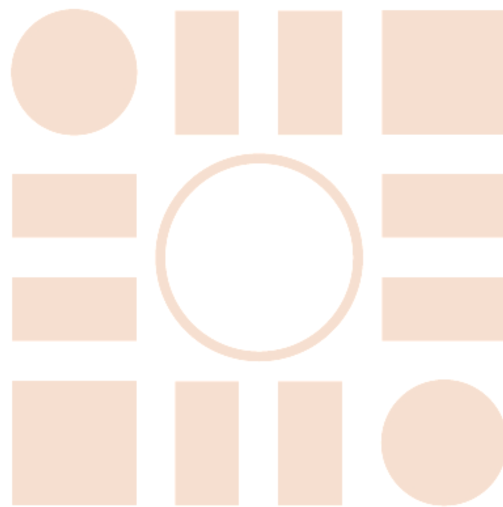
J. M. Rabaey. “Digital Integrated Circuits: A Design Perspective”, Ed. Prentice-Hall, 1996.

Estándar IEEE-1149.1. Standard Test Access Port and Boundary-Scan Architecture (http://fiona.dmcs.pl/~cmaj/JTAG/JTAG_IEEE-Std-1149.1-2001.pdf)

BOE.es (2018) (https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf)

Notes from Universidad de Alcalá.

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR