eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Escuela de Máster y Doctorado

Máster en Sistemas Electrónicos Avanzados

▪ Trabajo Fin de Master ▪

Proyecto: Controlador para un inversor trifásico de potencia sobre SoC-FPGA

Jose Ángel Gumiel Quintana

Mayo 2019

Dirección:

Jon Mabe

Rafa Bárcena

Agradecimientos

Antes de nada, me gustaría aprovechar para agradecer a todos aquellos que han participado directa o indirectamente en el proyecto. Por un lado, quiero dar las gracias a IK4-Tekniker, por darme la oportunidad de realizar este TFM, y a los compañeros que allí he conocido, quienes no han tenido ningún inconveniente en dedicarme parte de su tiempo para resolver mis dudas, y de los que he aprendido algo nuevo cada día.

Por otra parte, también quiero dedicar esta página a los docentes que me han acompañado en mi etapa universitaria. Este trabajo me ha servido para darme cuenta de dos cosas. La primera de ellas es que el contenido teórico de algunas asignaturas puede acabar siendo una buena base para el autoaprendizaje, y la segunda de ellas va hacia el autoconocimiento, y es que soy de los que piensa que, al acabar la carrera, los estudiantes sabemos más de lo que creemos saber.

Durante la vida académica se suele poner el objetivo en el aprobado, y al querer ir deprisa, hay muchos temas que pasan desapercibidos. Este proyecto ha sido bastante multidisciplinar, y me ha servido para darme cuenta de las fortalezas que tenía y de los vacíos que quedaron durante esa etapa. Pero al ver la utilidad de todo lo que aprendí, hoy puedo dar las gracias a todos aquellos profesores que nos ayudaron a asentar unas buenas bases y que se molestaron en actualizar las asignaturas a la evolución tecnológica.

Para terminar, no puedo olvidarme de familia y amigos. Aquellos que me han acompañado durante este camino y con los que he compartido inquietudes, logros, problemas y días buenos y no tan buenos.

Gracias a todos.

"La imaginación es más importante que el conocimiento. El conocimiento es limitado. En cambio, la imaginación abarca todo".

- Albert Einstein

Resumen

En este proyecto se implementan diferentes algoritmos acompañados de su verificación con el objetivo de implementar una aplicación de test y puesta en marcha para un inversor trifásico de potencia. Para ello se utilizará la placa de desarrollo Terasic DE10-Nano, que incluye una FPGA (dispositivo lógico programable) y un microprocesador ARM V9. Se recurre a la adquisición, procesado y generación digital de señales, además de a la implementación en hardware de diferentes protocolos de comunicación.

Un inversor trifásico convierte corriente continua (CC) en corriente alterna (CA). El voltaje de entrada y salida, la frecuencia y la potencia dependen del diseño de la circuitería. El inversor no produce potencia, esta depende de la fuente de CC.

Hoy en día nos encontramos ante el auge de las energías renovables. Está habiendo una fuerte apuesta por la energía solar y eólica, y uno de los problemas que estas presentan es que la electricidad la generan en CC. El inconveniente principal de la CC es que transportarla desde el lugar de generación a largas distancias es muy difícil, la potencia se pierde a través de los cables. Por el contrario, la CA no solo se puede transportar a largas distancias con facilidad, si no que sus valores se pueden modificar mediante el uso de transformadores. Esto permite transportar la energía de una forma mucho más eficiente, no sólo se pueden atravesar ciudades, si no naciones. Además, en automoción, el coche eléctrico está jugando un importante papel, y también será necesario el uso de inversores de altas prestaciones, que reduzcan los tiempos de carga.

Las nuevas demandas de la industria giran en torno a la reducción de pérdidas de energía. Se están usando nuevos materiales, como el Carburo de Silicio (SiC), que es un compuesto sintético producido a partir de silicio y carbón. Las cualidades de este son su dureza, resistencia a altas temperaturas y alta conductividad eléctrica. Es una alternativa ideal a los semiconductores tradicionales, y permite aumentar la frecuencia de conmutación de los inversores de potencia.

Ante este panorama se deben buscar nuevas alternativas que estén a la altura de los cambios. Un microcontrolador (μC) tiene unas cualidades distintas a una FPGA, y si las nuevas tendencias demandan más velocidad, el paso natural será migrar los controladores hacia una plataforma que sea rápida y robusta.

Índice

AGRADECIMIENTOS III

RESUMEN V

ÍNDICE VII

LISTA DE FIGURAS Y TABLAS XIII

FIGURAS XIII

TABLAS XVI

INTRODUCCIÓN Y DEFINICIONES 1

1.1. CONTEXTUALIZACIÓN 3

1.1.1. Usos de un inversor trifásico 3

1.1.2. Funcionamiento 4

1.1.3. Hardware de control 5

1.2. ANTECEDENTES Y MOTIVACIONES 6

1.3. CONTEXTO DEL PROYECTO EN EL ÁMBITO ACADÉMICO 7

1.3.1. Cumplimiento de los requisitos 8

ESTADO DEL ARTE 11

2.1. INVERSORES DE POTENCIA 13

2.1.1. Inversores modernos 13

2.1.2. Actualidad: Hacia el SiC y el GaN 14

2.1.3. Métodos de conmutación para IGBTs 17

2.2. PROCESADORES DE CONTROL 17

2.2.1. Microcontroladores y microprocesadores 18

2.2.2. Procesadores de Señales Digitales 18

2.2.3. Field Programable Gate Array 19

2.2.4. SoC-FPGA 20

ESPECIFICACIÓN DEL DESARROLLO Y OBJETIVO 25

3.1. ESPECIFICACIÓN DEL DESARROLLO 27

3.1.1. Comunicaciones 28

3.1.2. Generación de señales de control 28

3.1.3. Monitorización 29

3.1.4. Protección 29

3.1.5. Integridad de la señal y cableado 30

3.2. OBJETIVO 30

METODOLOGÍA 31
4.1. SELECCIÓN DE LA PLACA DE DESARROLLO 33
4.1.1. Valoración de las necesidades 34
4.1.2. Estudio de tarjetas disponibles 35
4.2. DEFINICIÓN DE LA APLICACIÓN 36
4.3. HERRAMIENTAS SOFTWARE 37
4.3.1. ¿Por qué no emplear la generación automática de código? 38
4.4. SELECCIÓN DE LENGUAJES 39
4.5. DEFINICIÓN DE BLOQUES FUNCIONALES 40
4.6. IMPLEMENTACIÓN/INTEGRACIÓN 41
4.6.1. Etapa 1: Inicio, familiarización y puesta en marcha 41
4.6.2. Etapa 2: Diseño de arquitecturas SoC 42
4.6.3. Etapa 3: Real-Time Linux 42
4.6.4. Etapa 4: Aplicación y prototipo final 42
4.6.5. Etapa 5: Pruebas sobre el inversor 43
4.7. PRUEBAS UNITARIAS 43
4.8. PRUEBAS SISTEMA 44
4.9. SOBRE LA METODOLOGÍA 44

PLANIFICACIÓN 45
5.1. INTRODUCCIÓN 47
5.2. ALCANCE 47
5.3. ESTRUCTURA DE DESCOMPOSICIÓN DEL TRABAJO (EDT) 47
5.4. DIAGRAMA DE GANTT 48
5.4.1. Investigación 49
5.4.2. Implementación 50
5.4.3. Análisis de resultados 50
5.4.4. Documentación 51
5.5. HITOS 51
5.6. GESTIÓN DE RIESGOS 52
5.6.1. Identificación de riesgos 52
5.6.2. Plan de contingencia 53
5.7. ESTIMACIÓN DEL TIEMPO 53

IMPLEMENTACIÓN: DISEÑOS FPGA 55
6.1. INTRODUCCIÓN 57
6.2. PRUEBA DE ELEMENTOS DE LA PLACA 57
6.2.1. Leds e interruptores 57
6.2.2. Reloj 57
6.2.3. Pulsador 58
6.2.4. GPIO 58

6.3. ADQUISICIÓN DE DATOS 61
6.3.1. Muestreo 61
6.3.2. Cuantificación 62
6.3.3. Codificación 63
6.3.4. Procesado 63
6.3.5. Conversor Analógico-Digital (On-Board) 64
6.4. PULSE-WIDTH MODULATION (PWM) 70
6.5. PROTOCOLOS DE COMUNICACIÓN 72
6.5.1. Especificación del protocolo I2C 72
6.5.2. Implementación del protocolo I2C 74
6.5.3. Especificación del protocolo SPI 77
6.5.4. Implementación del protocolo SPI 78
6.6. JUSTIFICACIÓN DE LA ETAPA 81

IMPLEMENTACIÓN: SOC-FPGA 83

7.1. LINUX (DE SERIE) 85
7.1.1. Primeros pasos 85
7.2. DISEÑO DE UN SOC 86
7.2.1. Consideraciones previas 87
7.2.2. Preparación inicial 87
7.2.3. Diseño y construcción del sistema 88
7.2.4. Compilación del sistema 91
7.2.5. Compilación de la aplicación en C 93
7.2.6. Ejecución del programa 97
7.3. HACIA SOC MÁS COMPLEJOS 97
7.3.1. Siguiendo el ejemplo 97
7.3.2. Un nuevo paradigma 100

IMPLEMENTACIÓN: REAL-TIME LINUX 103

8.1. INTRODUCCIÓN 105
8.2. SISTEMAS OPERATIVOS 105
8.2.1. El Kernel 105
8.2.2. Los procesos 106
8.2.3. El planificador de procesos 108
8.3. LINUX-RT 110
8.3.1. Estudio de alternativas 111
8.3.2. Instalación del Kernel 112
8.4. COMPROBACIÓN DE LATENCIAS 115
8.4.1. Análisis de los resultados 116
8.5. MÓDULOS DEL KERNEL (INTERRUPCIONES) 119
8.5.1. ¿Qué es un módulo del Kernel? 119

8.5.2. Programación de los módulos ... *120*
8.5.3. Conociendo la gestión de interrupciones entre HPS y Linux ... *120*
8.5.4. Preparación del entorno Linux ... *122*
8.5.5. Diseño del sistema Hardware ... *125*
8.6. WEB-SERVER ... 129
8.6.1. Instalación y configuración ... *129*
8.6.2. Web básica para mostrar información ... *131*

IMPLEMENTACIÓN: EL PROTOTIPO FINAL ... 133

9.1. INTRODUCCIÓN ... 135
9.2. MODULACIÓN EN ANCHO DE PULSO (PWM) ... 136
9.3. MODULACIÓN EN ANCHO DE PULSO SINUSOIDAL (SPWM) ... 138
9.3.1. Amplitud de la señal variable ... *141*
9.4. CONTROL DE TENSIÓN Y CORRIENTE ... 144
9.4.1. Funcionamiento ... *144*
9.4.2. Comprobación ... *146*
9.5. APLICACIÓN ... 147
9.6. ADECUACIÓN DE LAS SEÑALES ... 150
9.7. EL PROTOTIPO ... 155

PRUEBAS SOBRE EL INVERSOR ... 157

10.1. ESCENARIOS ... 159
10.1.1. Escenario 1. Sin carga. ... *159*
10.1.2. Escenario 2. Baja tensión. ... *160*
10.1.3. Escenario 3. Alta tensión ... *161*
10.2. PWM ... 162
10.2.1. Sin carga ... *162*
10.2.2. Con carga ... *163*
10.3. SPWM ... 165
10.3.1. Sin carga ... *165*
10.3.2. Con carga ... *166*
10.3.3. Alta tensión ... *168*
10.4. SISTEMAS DE SEGURIDAD: MONITOR DE CORRIENTE Y TENSIÓN ... 170
10.4.1. Sobreintensidad ... *170*
10.4.2. Sobretensión ... *171*

CONCLUSIONES Y LÍNEAS FUTURAS ... 175

11.1. VISIÓN GENERAL Y LÍNEAS ABIERTAS ... 177
11.2. EL PROTOTIPO ... 178
11.3. RETROSPECTIVA ... 178
11.4. LABOR INVESTIGADORA ... 179

11.5. TRABAJO FUTURO 180

ANEXO A: LISTADO DE PROYECTOS GENERADOS 187

LÓGICA PROGRAMABLE 187
INTERRUPCIONES 187
SOC-FPGA 188

Lista de Figuras y Tablas

FIGURAS

Figura 1.1 Esquema del funcionamiento de un inversor de potencia. 4
Figura 1.2 Esquema de conexionado de un circuito de IGBTs. 5
Figura 2.1 Comparativa entre las características de Si, SiC y GaN [10]. 14
Figura 2.2 Selección de tecnología en función del voltaje y la aplicación. 15
Figura 2.3 Esquema de un sistema de control. 17
Figura 2.4 Esquema de la configurabilidad de una FPGA. 19
Figura 2.5 Diagrama de bloques de un SoC FPGA de Altera [32]. 21
Figura 4.1 Lista de categorías a investigar para escoger una FPGA [34]..... 33
Figura 4.2 Interfaces de comunicación y conexión de dispositivos entorno a la CPU. 34
Figura 4.3 Arquitectura del controlador en SoC-FPGA. 37
Figura 4.4 Esquema de bloques funcionales. 40
Figura 4.5 Algoritmo “divide y vencerás”. 41
Figura 5.1 EDT del proyecto. 48
Figura 5.2 Diagrama de Gantt del proyecto. 49
Figura 5.3 Hitos/Roadmap del proyecto. 51
Figura 6.1 Módulo “pruebaLeds”. 57
Figura 6.2 Módulo “pruebaLedsCont”. 58
Figura 6.3 Módulo “pruebaKey”. 58
Figura 6.4 Módulo “pruebaEscrituraGPIO”. 59
Figura 6.5 Generación de señal en FPGA y salida a través de GPIO. 59
Figura 6.6 Módulo “pruebaLecturaGPIO”. 59
Figura 6.7 Adquisición y repetición de una señal cuadrada. 60
Figura 6.8 Módulo “pruebarw-GPIO”. 60
Figura 6.9 Esquema del proceso de conversión analógico-digital. 61
Figura 6.10 Efecto aliasing. 62
Figura 6.11 Señal sinusoidal muestreada en 4 bits en complemento a 2. 64
Figura 6.12 Patillaje del cabezal 2x5 correspondiente al ADC. 64
Figura 6.13 Conexiones entre el cabezal, el ADC y el controlador diseñado. 65
Figura 6.14 Esquema del módulo “LTC2308_cyclic”. 66
Figura 6.15 Pantalla para configurar el IP-Core. 67
Figura 6.16 Creación del sistema en el Platform Designer (antes Qsys). 68
Figura 6.17 Módulo “pruebaAnalogPWM”. 71
Figura 6.18 Señal PWM controlada por potenciómetro. Duty-Cycle:15%. 71
Figura 6.19 Esquema funcionamiento del I2C mediante resistencias de pull-up. 73
Figura 6.20 Esquema de conexionado del bus I2C con múltiples dispositivos. 73
Figura 6.21 Módulo “pruebaI2C_RW”. 75

Figura 6.22 Conexión de la placa de evaluación a la DE10-Nano. 76
Figura 6.23 Trama I2C de escritura y lectura generada por la FPGA. 76
Figura 6.24 Operación de lectura. 76
Figura 6.25 Esquema de conexionado del bus SPI con múltiples dispositivos. 77
Figura 6.26 Módulo “pruebaSPI_RW”. 78
Figura 6.27 Tarjeta de evaluación con ADC. Ubicación de los “test points”. 79
Figura 6.28 ADC conectado a la fuente de alimentación y a la placa de desarrollo. 80
Figura 6.29 Trama obtenida a través del analizador lógico. 80
Figura 6.30 Trama con frecuencia variable obtenida a través del analizador lógico. 81
Figura 7.1 Compilación y ejecución de un programa. 86
Figura 7.2 Comprobación del puerto y conexión en PuTTY. 88
Figura 7.3 Selección de componentes a instanciar en System Builder. 89
Figura 7.4 Diseño de la arquitectura del sistema en “Platform Designer”. 90
Figura 7.5 Plantilla de código para la instanciación. Incompleta, pero ahorra tiempo. 90
Figura 7.6 Parte del código. Se usó la plantilla y se editó. 91
Figura 7.7 Los dos Scripts Tcl que hay que ejecutar. 92
Figura 7.8 Bootloader cargando el sistema SoC diseñado. 93
Figura 7.9 Generación de cabeceras. 95
Figura 7.10 Compilación del programa en C y envío del fichero a la placa. 96
Figura 7.11 Creación del sistema en Platform Designer. 98
Figura 7.12 Programa en funcionamiento. 98
Figura 7.13 HPS leyendo y escribiendo datos. 99
Figura 7.14 FPGA y HPS trabajando en conjunto. 100
Figura 8.1 Anillos de privilegio. 106
Figura 8.2 Estructura de un proceso en memoria. 107
Figura 8.3 Información del sistema. Kernel 4.5.0. 112
Figura 8.4 Compilación completada con éxito. 114
Figura 8.5 Imagen del Kernel lista. 114
Figura 8.6 Información del nuevo Kernel. 114
Figura 8.7 Ejecución del test en el Kernel nativo. 115
Figura 8.8 Ejecución del test en el Kernel Preempt-RT. 115
Figura 8.9 Esquema de comunicación circular. 116
Figura 8.10 Comparativa del tiempo de respuesta medio en distintos escenarios. 118
Figura 8.11 Comparativa del tiempo de respuesta máximo en distintos escenarios. 118
Figura 8.12 Esquema de la gestión de interrupciones. 120
Figura 8.13 Esquema del controlador de interrupciones integrado en el ARM. 121
Figura 8.14 Arquitectura del GIC. 121
Figura 8.15 Creación del fichero en la ruta descrita. 122
Figura 8.16 Comando y resultado de la compilación. 123
Figura 8.17 Resultado de la compilación del módulo. 124
Figura 8.18 Instanciación y deshabilitación del módulo. 125

Figura 8.19 Lectura del fichero “/proc/interrupts”. 125
Figura 8.20 Configuración de las señales. 127
Figura 8.21 Esquema del IP Core generado. 127
Figura 8.22 Sistema diseñado en Platform Designer. 128
Figura 8.23 Recepción de señales a través del módulo. 128
Figura 8.24 Recepción de IRQ y respuesta. 129
Figura 8.25 Programa en ejecución y obtención de datos vía web. 132
Figura 9.1 Arquitectura del controlador en SoC-FPGA. 135
Figura 7.1 Esquema parcial del diseño del generador de pulsos PWM. 136
Figura 9.2 Representación gráfica de las señales generadas en VHDL. 136
Figura 9.3 Consola para el cambio de parámetros de la señal PWM. 137
Figura 9.4 Captura de una señal PWM. 137
Figura 9.5 Representación gráfica de la simulación del SPWM. 139
Figura 9.6 Generación de PWM por comparación en una de las fases. 140
Figura 9.7 Esquema de un filtro RC. 140
Figura 9.8 Señales PWM filtradas. 141
Figura 9.9 Cambio de amplitud sin offset. 142
Figura 9.10 Cambio de amplitud con offset fijo al 50%. 143
Figura 9.11 Parada de IGBTs y activación del chopper por sobreintensidad. 146
Figura 9.12 Parada de IGBTs y activación del chopper por sobretensión. 147
Figura 9.13 Esquema del diseño del prototipo final. 148
Figura 9.14 Menú principal de la aplicación software. 149
Figura 9.15 Montaje de pruebas. Placa de desarrollo conectada a interfaz. 150
Figura 9.16 Quartus II. Menú de configuración del dispositivo. 150
Figura 9.17 Quartus II. Selección del voltaje de las E/S del dispositivo. 151
Figura 9.18 Pulso On sin resistencia. 151
Figura 9.19 Pulso Off sin resistencia. 151
Figura 9.20 Pulso On con resistencia de 24Ω. 152
Figura 9.21 Pulso Off con resistencia de 24Ω. 152
Figura 9.22 Pulso On con resistencia de 50Ω. 152
Figura 9.23 Pulso Off con resistencia de 50Ω. 152
Figura 9.24 Primer zócalo para conectar la placa de desarrollo a la interfaz. 152
Figura 9.25 Conexión entre Terasic DE10-Nano y tarjeta interfaz. 153
Figura 9.26 Quartus II. Selección de slew-rate en el Pin Planner. 154
Figura 9.27 Quartus II. Configuración de los pines no utilizados. 154
Figura 10.1 Conexión del inversor trifásico sin carga. 159
Figura 10.2 Conexión del inversor trifásico a baja tensión. 160
Figura 10.3 Uno de los primeros montajes en el laboratorio. 160
Figura 10.4 Conexión del inversor trifásico a alta tensión. 161
Figura 10.5 Carga RL a la salida del inversor trifásico. 162
Figura 10.6 Conmutación de una pareja de IGBTs en vacío. 163

Figura 10.7 Conmutación a 4kHz de un IGBT a 30 VDC. 164
Figura 10.8 Conmutación a 16kHz de una pareja de IGBTs a 30 VDC. 165
Figura 10.9 Conmutación de uno de los IGBTs. 166
Figura 10.10 Conmutación de uno de los IGBTs. 166
Figura 10.11 Medida de las tres corrientes en el inversor. 167
Figura 10.12 Salida de tensión del inversor. 167
Figura 10.13 Medición de las corrientes en alta tensión. 168
Figura 10.14 Medición de las corrientes en alta tensión. 169
Figura 10.15 Conexión de la placa de desarrollo al inversor es escenario de alta tensión. . 169
Figura 10.16 Verificación en sobreintensidad. 170
Figura 10.17 Verificación del control de tensión. Estado 0. 171
Figura 10.18 Verificación del control de tensión. Estado 2. 172
Figura 10.19 Verificación del control de tensión. Estado 2. 172
Figura 10.20 Verificación del control de tensión. Estado 3. 173

TABLAS

Tabla 2.1 Comparativa entre configuraciones del procesador Nios II. 22
Tabla 4.1 Comparativa de dispositivos en el mercado que podrían servir. 36
Tabla 5.1 Tiempo estimado para cada conjunto de tareas. 53
Tabla 8.1 Listado de sistemas operativos RT. 111
Tabla 8.2 Resultados obtenidos. 117
Tabla 8.3 Resultados más significativos. 118
Tabla 9.1. Valores de referencia del transductor de corriente. 145
Tabla 9.2. Valores de referencia del transductor de voltaje. 145
Tabla A.1. Listado de proyectos sobre lógica programable. 187
Tabla A.2. Listado de proyectos sobre la gestión de interrupciones. 187
Tabla A.3. Listado de proyectos implementados sobre SoC-FPGA. 188

1

Introducción y definiciones

Este primer capítulo sirve para contextualizar el proyecto. Se hace una breve introducción para conocer qué es un inversor trifásico de alta potencia, en qué consiste, cuáles son sus aplicaciones...

1.1. Contextualización

Antes de explicar las motivaciones del proyecto, conviene contextualizarlo, de forma que el lector sepa en todo momento qué se va a hacer, para qué, cómo y con qué finalidad.

El proyecto se enmarca en el campo de la electrónica de potencia, en concreto se va a trabajar sobre un inversor trifásico de potencia. Un inversor es un dispositivo eléctrico que transforma las propiedades eléctricas de entrada. A partir de una fuente de energía de CC [1], ésta se convierte a CA con las magnitudes y frecuencias deseadas. Se compone principalmente por dispositivos electrónicos de potencia, que trabajan como interruptores operando en corte y saturación. De este modo, se obtienen tensiones de salida simétrica y balanceada.

Uno de los componentes fundamentales de estos equipos es el controlador. Este se va a encargar de generar las señales de encendido y apagado en los dispositivos semiconductores, y garantizará su buen y correcto funcionamiento. El controlador es un sistema electrónico que sirve para actuar sobre el inversor y monitorizarlo.

1.1.1. Usos de un inversor trifásico

Puede que se desee usar un inversor por muchos motivos. En los siguientes puntos se enumeran algunos de ellos:

- Obtener un voltaje particular a partir de CC: Dada una tensión en CC, se puede convertir a CA y obtener un voltaje determinado. Un ejemplo de uso podría darse en la batería de un coche tradicional. Generalmente, una batería de automóvil suele tener entre 12 y 24V_{CC}, a través de un inversor se puede obtener otro voltaje en CA y conectar a ésta otros dispositivos eléctricos.
- Fuentes de alimentación interrumpidas: Es el funcionamiento de un UPS (Uninterruptable Power Supply) o es castellano SAI (Sistema de Alimentación Ininterrumpida). Es un dispositivo que se conecta entre una toma de corriente y otro dispositivo final. Tiene una batería interna que está en estado de carga, pero las baterías funcionan con una reacción química que provoca una transferencia de electrones unidireccional, es decir, el voltaje sale en CC. En caso de interrupción eléctrica entra en funcionamiento el SAI. Éste toma la alimentación de la batería, y por eso tiene un inversor de corriente, para transformarla en CA.
- Control de velocidad en motores eléctricos: Hay inversores que están diseñados para generar una salida variable de voltaje. Estos son usados a menudo con un controlador de velocidad para motores. La salida del voltaje del inversor determinará la velocidad de giro del motor.
- Transmisión de energía HVDC: El acrónimo HVDC hace referencia a (High Voltage Direct Current). Es una forma de transportar energía eléctrica disminuyendo las

pérdidas por la resistencia de los cables. En el otro extremo debe haber un inversor para convertir esa energía en CA.

- Sistemas generadores de energía: Usado en estaciones generadoras de electricidad, las compañías eléctricas tienen que convertir la electricidad generada de CC a CA, de modo que puedan introducirla en la red eléctrica. Para esto hacen falta inversores de alta potencia. Además, con el auge de las fuentes de energías renovables, se hace necesario su uso. Los paneles solares o los aerogeneradores almacenan su energía en forma de CC, pero tiene que haber un elemento que la transforme a CA, el inversor.

1.1.2. Funcionamiento

El propósito de un inversor es transformar la energía de un suministro de CC en una corriente alterna. Esto se hace con parejas de IGBT-drivers conectados a un bus de CC [2]. Los IGBT son transistores de alta potencia, y sus siglas significan (Transistor Bipolar de Puerta Aislada). Una de las formas para modular una señal en el inversor es a través de la modulación en ancho de pulso (PWM), para generar estas señales hace falta que haya un dispositivo electrónico de control que gobierne el encendido y apagado de los IGBTs (Como podría sr un µC, DSP, FPGA…). Mediante la conmutación de los IGBTs se puede transformar una entrada de CC en CA. Para esto es necesario una frecuencia de conmutación de la magnitud de decenas de kilohercios y un patrón repetitivo mediante el que se consiga la frecuencia de CA deseada.

En la siguiente imagen se muestra una imagen simplificada de lo que es el funcionamiento de un inversor de potencia.

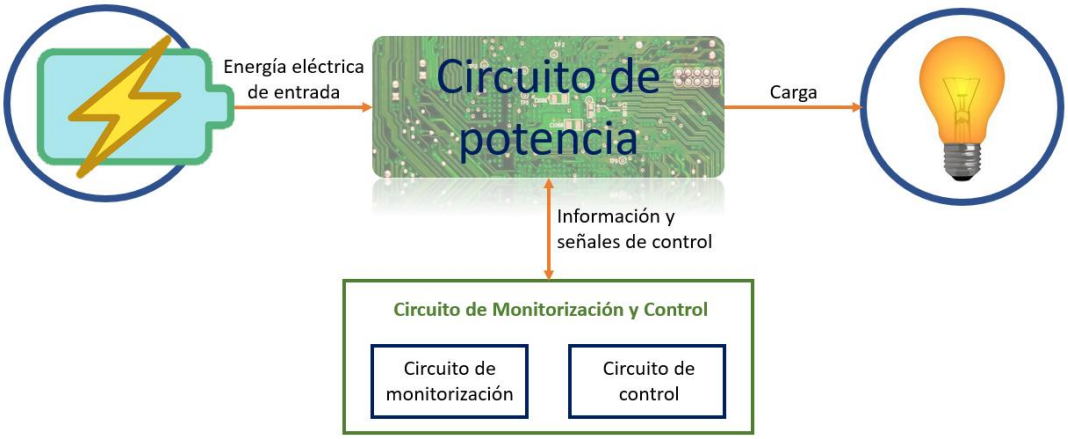


Figura 1.1 Esquema del funcionamiento de un inversor de potencia.

Yendo hacia un punto de vista particular, el inversor con el que se va a trabajar es trifásico. Esto significa que tendrá tres parejas de IGBTs conmutando, y como resultado se obtienen tres tensiones de salida, con su frecuencia fundamental desplazada en cada una de ellas 120º con respecto a su anterior.

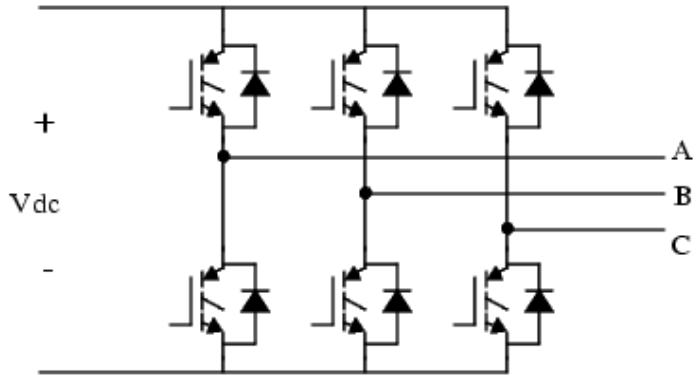


Figura 1.2 Esquema de conexionado de un circuito de IGBTs.

En un inversor trifásico el objetivo del empleo de la modulación de ancho de pulso es modelar y controlar la tensión trifásica de salida en magnitud y frecuencia a partir de una tensión de entrada en CC constante. Una única onda triangular, también llamada portadora o *carrier*, se compara con tres tensiones de control sinusoidales desfasadas entre sí 120º, para conseguir a la salida un sistema de tensiones trifásico equilibrado empleando la conmutación PWM. En los capítulos de implementación se detalla más en profundidad el funcionamiento de esta técnica.

1.1.3. Hardware de control

En la figura 1.2 se intuye la necesidad de un circuito de monitorización y control con el fin de enviar señales de control al circuito de potencia y de intercambiar información sobre el estado del inversor.

Hay que tener en cuenta que el controlador es responsable de realizar dos funciones al mismo tiempo. Por un lado, el hardware de control es el encargado de enviar señales al circuito de potencia con el objetivo de conmutar los IGBTs para obtener la tensión de salida de la forma deseada. Por otra parte, este controlador también tiene que monitorizar el estado del equipo. Esto último implica un proceso continuo de adquisición de datos y de la toma de decisiones en función de los resultados obtenidos. Lo ideal es hacer ambas funciones en paralelo.

La ejecución de las instrucciones de un algoritmo en un µC es lineal, esto significa que no puede hacer dos cosas al mismo tiempo. Aunque haya también µP multinúcleo, éstos están en condición de desventaja. Los tiempos de desarrollo son largos y la reusabilidad es baja. La generación de las señales PWM y el lazo de control requiere de frecuencias de muestreo y resoluciones muy altas, y consume casi todos los recursos del controlador. Sólo queda un tiempo limitado para el control de otras funciones secundarias. Un SoC-FPGA satisface estas demandas [3].

Hoy en día hay arquitecturas muy variadas en las que implementar un control de estas características, pero cabe hacer una especial mención a SoC-FPGA. Los SoC o System-on-Chip son la integración de un µC, un µP y otros elementos periféricos dentro de un mismo chip. En

el caso de SoC-FPGA, se añade una FPGA, por lo que existe dentro de un único circuito integrado un área dedicado a la lógica programable y otro dedicado al procesamiento.

1.2. Antecedentes y motivaciones

Al inicio de este proyecto, se parte de un inversor trifásico que utiliza IGBTs de silicio. Su funcionamiento ha sido probado a través de una aplicación de testeo y de puesta en marcha. Esta aplicación de control y monitorización se ejecuta sobre una arquitectura de µC, concretándolo más, se estaba usando una tarjeta de desarrollo NXP FRDM-K64F, que cuenta con un procesador ARM® Cortex®-M4.

En la actualidad se cuenta con nuevos materiales semiconductores, que están posibilitando hacer las mismas operaciones de antes, pero a una velocidad de conmutación mayor, permitiendo así alcanzar mayor eficiencia en la conversión. Ante este nuevo escenario, los sistemas de control basados en µC no tienen la capacidad computacional suficiente para sacar el máximo rendimiento de esta tecnología.

El desafío que se propone es cambiar los IGBT de silicio de un inversor de potencia trifásico por otros de carburo de silicio (SiC). Este material tiene ventajas sobre su predecesor, ya que permite unas frecuencias de conmutación más altas, tiene una mejor conductividad térmica y son más eficientes energéticamente [4]. Para acompañar a este nuevo material y aprovechar sus ventajas se pretende cambiar la arquitectura de control, y pasar de un µC a un SoC-FPGA.

Un SoC-FPGA permite obtener lo mejor de un µC y de una FPGA. Esto es debido a que se puede tener un procesamiento rápido en FPGA e implementar de forma sencilla la interfaz de usuario y las comunicaciones. Además, los SoC tienen ya algunos dispositivos periféricos conectados, lo que, dependiendo del diseño, podría ahorrar espacio y evitar retardos. También cabe destacar que suelen tener una muy buena eficiencia energética, debido a su bajo consumo.

Diseñar el controlador sobre un SoC FPGA permite lo siguiente:

- Realizar diferentes operaciones en paralelo.
- Disponer de un controlador reconfigurable. Se pueden añadir nuevas características y funcionalidades en caso de que surja una nueva necesidad.
- Periféricos integrados. Los periféricos pueden estar integrados en la misma placa que la FPGA, de modo que suponga un ahorro de espacio.
- Reparto de tareas. Tener un SoC con un µP y una FPGA permite combinar ambos elementos y repartir las tareas para aprovechar lo mejor de cada arquitectura.
- Ejecución de un sistema operativo o de aplicaciones bare-metal.

1.3. Contexto del proyecto en el ámbito académico

La realización de este proyecto también tiene una justificación en el marco académico. Se ha cursado un Grado en Ingeniería Informática en la línea curricular de Ingeniería de Computadores. Esto aporta un conocimiento sobre arquitecturas hardware y de computadores. Por otra parte, se conoce la gestión interna de los recursos de un sistema operativo y se han adquirido conocimientos sobre arquitecturas paralelas. Estos conocimientos pasados, entre otros, serán puestos en práctica en el proyecto.

Adicionalmente, también se van a emplear los conocimientos adquiridos en los estudios de postgrado. Se ha cursado el Máster de Sistemas Electrónicos Avanzados. En esta etapa se han adquirido conocimientos sobre descripción de hardware en HDL y de diseño de circuitos, se emplearán las buenas prácticas adquiridas en la implementación del código. También ha habido asignaturas que han tratado sobre programación de µC, y se ha llegado a programarlos en un entorno Linux RT. Es otra de las cosas que se van a hacer en el proyecto, por lo que servirá de base.

Otras materias han basado sus ejemplos de aplicación en inversores trifásicos, y en el curso se hicieron simulaciones de plantas. Estos conocimientos dan una visión general del funcionamiento de un inversor. Sí que se va a cambiar el enfoque de la aplicación y el modo de programarlo, ya que la aplicación no va a ser sobre un motor, ni se va a usar Simulink o un generador de código automatizado.

Algunas asignaturas también han tratado sobre arquitecturas de computadores, protocolos de comunicación o de verificación del hardware. Este proyecto también trata esos temas. Hay que conocer la arquitectura sobre la que se va a diseñar y a programar. También hay que estudiar los protocolos de comunicación, para posteriormente, programarlos en lógica programable. En cuanto a lo último, hay doble significado. Por un lado, se verifica el hardware diseñado mediante simulaciones y material de laboratorio, y por el otro lado, el prototipo resultante es para testear y poner en marcha otro dispositivo, el inversor.

En otro campo, pero no menos importante, se han estudiado técnicas de compatibilidad electromagnética. Aplicándolo al proyecto, sirve para obtener señales sin ruido y adaptarlas, de modo que se puedan acoplar dos dispositivos. Esto va a ser necesario, ya que la FPGA va a ir conectada a una tarjeta interfaz. Las señales tendrán que ser adaptadas y habrá que reducir los transitorios que la placa de desarrollo introduce en los flancos de subida y bajada. Por otra parte, el prototipo va a estar en un ambiente ruidoso, habrá que evitar que estas perturbaciones afecten al funcionamiento.

Las técnicas de gestión de proyectos, estudiadas tanto en el grado como en el máster, también se emplearán para garantizar la buena finalización del proyecto y el cumplimiento de los objetivos.

Por estas razones, se considera que este proyecto reúne las características necesarias para que sea un TFM acorde a la formación académica recibida y a los estudios de postgrado.

1.3.1. Cumplimiento de los requisitos

El TFM exige al alumno cumplir con una serie de requisitos. Éste es lo que se denomina un "*proyecto de desarrollo*". Se ha realizado en IK4-Tekniker, mediante un convenio de colaboración educativa con la Escuela de Máster y Doctorado (EMD).

El proyecto se realiza en el centro de investigación tecnológica, situado en Éibar (Gipuzkoa). Y, como se ha dicho anteriormente, están tratando de poner en funcionamiento un inversor trifásico de potencia basado en SiC. Para ello se va a implementar una aplicación de testeo y puesta en marcha, que irá sobre una arquitectura SoC-FPGA. Es decir, el proyecto va a finalizar con la obtención de un prototipo funcional, tal y como lo pide el máster.

Se va a mezclar tanto hardware como software. Y esto tiene dos vertientes, va a haber hardware implementado en código HDL, pero también van a existir diseños o construcciones sobre placas perforadas de prototipado, es decir, va a haber un apartado donde se va a trabajar con componentes electrónicos (resistencias, condensadores, bobinas…).

En el apartado de metodología se habla de los lenguajes, y de la diferencia entre un lenguaje de descripción de hardware y un lenguaje de programación. Aquí también se hace un inciso. La FPGA se programa a través de un HDL, esto quiere decir que se definen circuitos mediante código, y como sus propias siglas indican, esto es hardware. Por otro lado, la placa de desarrollo tiene un lado que es HPS, es decir, tiene un µP y llevará también un sistema operativo. La programación para esta plataforma se hará en C, que es un lenguaje de programación, y a esta parte sí que es software. En este sentido, tanto hardware como software pueden tener líneas de código.

Yendo hacia un punto de vista más físico, el proyecto sigue teniendo un componente de diseño electrónico. Por un lado, se va a trabajar con placas de evaluación, con las que se desea efectuar una comunicación y obtener un intercambio de información. También se va a usar una placa interfaz, que hace de puente entre la placa de desarrollo y el inversor. Habrá que comunicarse también con ella. En los dos casos, esto exige una lectura y correcta interpretación de los esquemas de los circuitos electrónicos.

Continuando con el hardware, en este proyecto se hacen también circuitos de filtrado, se verá durante la implementación. Y también se trata de reducir el ruido y mejorar las propiedades de las señales generadas. En el informe sobre el desarrollo se verá el análisis que se ha hecho y cómo se ha dado con una solución.

Otro problema ha sido también el hacer una estructura sencilla para conectar la placa de desarrollo con la interfaz, para este caso, se ha hecho un zócalo. Y se ha tenido en cuenta el ruido electromagnético, aplicando técnicas de aislamiento, y la adaptación de impedancias, mediante resistencias terminadoras.

La metodología que se ha utilizado es la propia de este tipo de centros, hay una tendencia investigadora y las acciones realizadas tienen una justificación. Es por ello que, en varios apartados, ha habido que hacer una investigación previa. La observación también juega un

papel fundamental, porque el análisis de los resultados también implica poder realizar mejoras o correcciones.

A la hora de realizar las mediciones, el modo de hacerlas ha tenido importancia. Tanto en las fases de verificación previas al montaje del prototipo en el inversor, como en la verificación final, se ha tratado de reducir el ruido lo máximo posible.

A pesar de ser un proyecto de desarrollo, también tiene una parte de investigación. Se han estudiado diferentes modos de hacer las cosas, y para ello hay que conocer tanto la arquitectura de la tecnología con la que se trabaja, como la labor de otros en este campo.

El trabajo realizado en la empresa, en conjunto con la documentación y código generados, sirven a ésta como punto de partida de futuras líneas de investigación. Y eso queda reflejado al final de este documento, acompañado de las conclusiones. Ahí se presenta un listado de campos que quedan abiertos para seguir trabajando, y un balance sobre el desarrollo del proyecto.

Se concluye este apartado asegurando que, el proyecto es transversal a las enseñanzas del Máster en Sistemas Electrónicos Avanzados, y que cumple con los requisitos que la EMD exige para la entrega de este TFM.

2

Estado del arte

Este capítulo sirve al lector como introducción a los conceptos teóricos requeridos para comprender el proyecto. Se estudia la evolución de las tecnologías utilizadas, y se analizan las nuevas demandas de la industria en la actualidad y las perspectivas de futuro que tiene.

2.1. Inversores de potencia

El inversor de potencia es un equipamiento que transforma la electricidad de CC en CA. Los primeros inversores eran complejos dispositivos electromecánicos. Usaban imanes y piezas móviles (incluyendo resortes) para realizar la transformación de CC a CA. El término "inversor" data de 1925, lo acuñó el ingeniero eléctrico David Prince en un artículo donde explica todas las funciones básicas del dispositivo. Su publicación contenía casi todos los elementos más importantes necesarios en un inversor moderno [5].

Parece ser que el término inversor derivó del "rectificador invertido" de Alexanderson. En 1922, este ingeniero estaba investigando métodos alternativos de uso de válvulas de vacío para la electrificación del ferrocarril. En un experimento usó un rectificador que producía 10kV en CC, esta tensión era posteriormente transformada en alterna a través de un circuito de válvulas, y reducida a través de un transformador para alimentar un motor eléctrico. El ingeniero llamó a este proceso "rectificación invertida" [6].

Aunque el inversor fue desarrollado y probado durante los años 30, su potencial no llegó a explotarse, debido a las limitaciones existentes en los rectificadores de la época. El tiratrón, un tipo de válvula termoiónica, y el ignitrón, un tipo de rectificador, estaban sujetos a tiempos de recuperación prolongados, necesarios para volver a obtener su capacidad dieléctrica. Además, eran sensibles a la temperatura ambiente, a la orientación y a las vibraciones mecánicas. Hubo que esperar a la llegada del tiristor en 1957 para que se abriera el campo hacia las aplicaciones masivas de electrónica de potencia [5].

2.1.1. Inversores modernos

Con la llegada del silicio se ha pasado de usar interruptores mecánicos a eléctricos. La circuitería de los inversores modernos emplea transistores y semiconductores, que reemplazan a las piezas móviles que se usaban en los inversores primitivos.

A diferencia de sus predecesores, los inversores modernos pueden operar a altos voltajes (entre 130 y 600V) y altas capacidades de salida (más de 500VA) [7].Además, la reducción de tamaño, los tiempos de conmutación más pequeños y la menor pérdida de energía fueron sus ventajas más destacables. Estos avances proporcionaron más flexibilidad, con un mayor rendimiento y a un tamaño menor.

Actualmente, la electrónica de potencia basada en semiconductores de Silicio se encuentra en un nivel muy maduro. Se continúa mejorando los dispositivos, pero los avances tecnológicos de los semiconductores de Silicio consiguen mejoras en sus características relativamente pequeñas. Se puede decir que, prácticamente, se han alcanzado los límites físicos del material [8].

El impacto de las energías renovables, que cada vez tienen una mayor penetración, o las máquinas eléctricas de alta velocidad, están propiciando una evolución tecnológica y proponiendo a la industria nuevos retos.

Un ejemplo claro se puede encontrar en el sector automovilístico. Toyota ja producido para sus coches híbridos un inversor de alta densidad de potencia de 17kW/L, y han llegado a un prototipo de hasta 26kW/L, lo que es extremadamente alto para un inversor basado en tecnología de silicio. Esto es una muestra de cómo se están alcanzando los límites de este material, y de cómo existe una necesidad de investigar en nuevas tecnologías [9].

2.1.2. Actualidad: Hacia el SiC y el GaN

Si la aparición del transistor de silicio provocó el reemplazo de los sistemas analógicos por otros digitales [10], en la actualidad, se está produciendo la transición de los IGBT de silicio (Si) hacia los de carburo de silicio (SiC). Este nuevo material aporta nuevas prestaciones y presenta varias ventajas que se basan en el incremento de la velocidad de conmutación. Por lo que renovar la electrónica y dar el salto hacia los SoC FPGA es un paso esencial para poder estar a la vanguardia de las nuevas tecnologías.

Por si fuera poco, vivimos en una época concienciada medioambientalmente, y parece ser que el futuro va hacia el uso de energías renovables y vehículos eléctricos. Para ambos casos, son necesarios los inversores de potencia, y se debe de tratar que sean lo más eficientes posibles y que tengan las menores pérdidas.

Dado que se está cerca de alcanzar los límites del silicio [11] y que han aparecido semiconductores de banda ancha prohibida (SiC y Nitruro de Galio - GaN), surge el objetivo de mejorar los actuales convertidores de potencia, y se habla ya de una nueva generación. Aunque estos dos materiales no se encuentren en una etapa tan madura como el Silicio, son sin duda alternativas prometedoras.

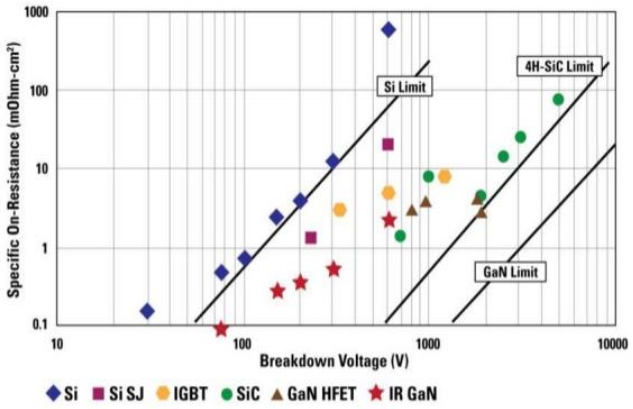


Figura 2.1 Comparativa entre las características de Si, SiC y GaN [12].

En la imagen superior se presenta un gráfico que sitúa estos 3 componentes. Los dispositivos SiC y GaN pueden trabajar a mayores frecuencias, y además tienen la ventaja de ser más eficientes y de reducir el tamaño de los convertidores. La reducción de tamaño viene asociada a que, al conmutar a altas frecuencias, los elementos de filtrado asociados a la aplicación reducen su tamaño. Las diferencias principales entre el SiC y el GaN es que este último no permite una tensión de bloqueo por encima de los 650V, aunque tiene un mejor campo de ruptura. Debido a este motivo, es por lo que se espera que sea el SiC el que esté

destinado a aplicaciones de alta tensión/potencia, ya que tiene muy buenas propiedades térmicas y un alto campo de ruptura. En resumen, el GaN puede conmutar a mayores frecuencias que el SiC, pero no puede soportar tanta tensión. De todas formas, ambos tienen ventajas sobre el silicio.

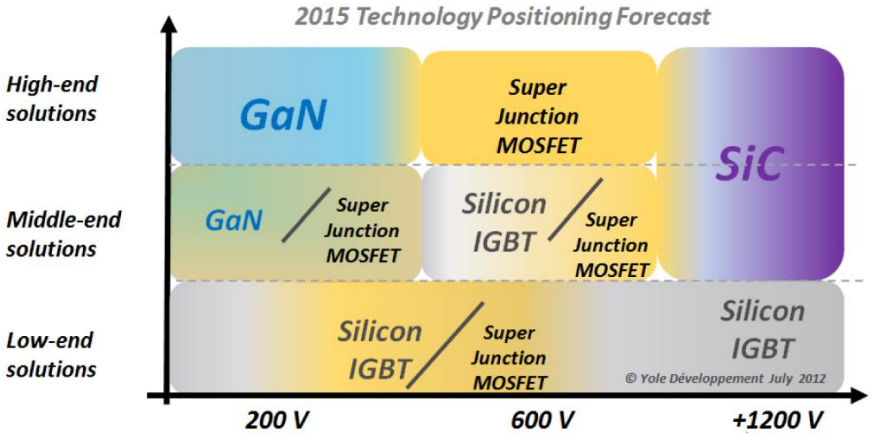


Figura 2.2 Selección de tecnología en función de voltaje y aplicación. Fuente: Yole Développment.

En la imagen superior se muestra un gráfico de posicionamiento, que indica qué tecnología se debería elegir en función de la aplicación.

El material SiC supera en 10 veces la capacidad de soportar un campo eléctrico frente al material semiconductor de Si estándar y en teoría, para un mismo espesor, el SiC es capaz de soportar una tensión 10 veces superior. Esta propiedad permite reducir la cantidad de material de SiC, mejorando la conductividad eléctrica y térmica del semiconductor, lo que repercute en menores pérdidas de calor y mayores densidades de corriente con respecto al Si [13]. Además, se reduce la constante dieléctrica del semiconductor, lo que reduce la energía almacenada durante las conmutaciones. Otra propiedad interesante del SiC es el alto valor de energía térmica necesario para que un electrón salte de la banda de valencia a la de conducción, lo que significa una capacidad de trabajar a temperaturas superiores al Si. La combinación de todos estos factores permite que los dispositivos de SiC sean capaces de operar a frecuencias de trabajo superiores al Si, en teoría entre 50 kHz y 1 GHz [14].

Actualmente, los dispositivos de SiC están diseñados para aplicaciones de media-alta tensión (1200-1700 VDC) con un rango de potencias elevado. Por ello, este tipo de dispositivos estarán destinados principalmente a aplicaciones de control de motores, fuentes de potencia y transmisión de energía.

A pesar de las ventajas que ofrece el SiC, y aunque ya es posible encontrar dispositivos a nivel comercial, todavía esta tecnología se encuentra en una etapa inicial de desarrollo. Tanto es así que algunas de sus ventajas no han sido exploradas en su totalidad debido a limitaciones tecnológicas y/o a diseños no optimizados.

En cuanto a las potenciales aplicaciones de los convertidores de potencia basados en SiC hay varios sectores interesados. Aún tienen un precio elevado y poca disponibilidad, pero eso está ya cambiando y todo apunta hacia una buena alternativa para el futuro cercano. A continuación, se citan los sectores que se harán uso de esta tecnología:

Sector transporte: Dentro de las aplicaciones relacionadas con el transporte, es el coche eléctrico e híbrido (EV/HEV) el que se posiciona como una de las más prometedoras, y hay dos vías claras. Por un lado, está el desarrollo de cargadores embarcados de alta frecuencia, usando SiC/GaN se puede reducir el tamaño y mejorar la eficiencia, y, por otro lado, el desarrollo de cargadores de alta eficiencia y tamaño reducido para el *powertrain* [15] [16].

La introducción del EV/HEV exige el desarrollo de nuevos convertidores de potencia capaces de trabajar en entornos de alta temperatura ambiente y grandes densidades de potencia. En concreto, en el caso de los vehículos híbridos, actualmente es necesario introducir dos sistemas de refrigeración independientes ya que la electrónica de potencia tradicional precisa temperaturas del líquido refrigerante inferiores a los 80ºC mientras que el sistema primario de refrigeración del motor térmico fácilmente puede alcanzar los 105ºC. Los nuevos semiconductores son capaces de trabajar a más de 175ºC, por lo que con su uso se puede recurrir al empleo de un único sistema de refrigeración, que disminuiría el volumen del sistema y aumentaría su fiabilidad, además de reducir costes.

Desde el punto de vista de los cargadores rápidos y ultrarrápidos, la introducción de la tecnología SiC permitirá reducir su tamaño y peso [17] [18].

Dentro del mismo sector, otro beneficiario que también está presente es el transporte ferroviario. Hitachi ya ha desarrollado sistemas que permiten reducir el volumen de los convertidores tradicionales basados en Si en un 40% [19]. Además, existe la iniciativa europea "Shift2Rail", que apoya la integración de los nuevos semiconductores en los sistemas electrónicos de los trenes, incluyendo también la electrónica de potencia [20].

Sector eólico: En el sector eólico hay que transformar la energía que viene de la turbina, lo cual es otra área de aplicación atractiva para estos conversores. Los aerogeneradores de velocidad variable están jugando un papel importante en el mercado de la generación eólica, y actualmente más del 75 % de las unidades vendidas requieren convertidores de potencia para controlar la energía generada [21]. Los dispositivos basados en SiC mejorarían la eficiencia e incrementarían la densidad de potencia [22] [23] [24].

Sector solar: Ocurre algo parecido al caso anterior. Ya hay compañías que están cerca de disponer de prototipos de inversores para placas fotovoltaicas basados en SiC y en GaN [25]. Además, el hecho de aumentar la frecuencia de trabajo permite reducir el tamaño de los convertidores, hasta tal punto que se podría llegar a integrar directamente en los paneles [26].

2.1.2.1. Otras ventajas de los semiconductores de banda ancha prohibida

El uso de estos dispositivos semiconductores ofrece algunas virtudes muy atractivas y que son aplicables a las máquinas eléctricas de alta velocidad.

Con la implementación de estos semiconductores, se puede incrementar la eficiencia de un motor. Esto se produce debido a que, para un mismo nivel de pérdidas en el convertidor, la frecuencia de conmutación puede ser elevada sustancialmente, reduciendo así la distorsión

armónica en las corrientes del motor, y por tanto, las pérdidas que ocasionan esos harmónicos. El tamaño y el peso del motor, también se verían reducidos, ya que el calor a disipar es menor [27].

Disponer de frecuencias de conmutación elevadas permite rediseñar las máquinas de alta velocidad. Se podría ampliar su número de pares de polos, reduciendo así el material magnético del estator, lo que también conlleva una reducción de peso y volumen.

Resumiendo, los dispositivos de banda ancha prohibida permiten reducir el tamaño de la máquina eléctrica y del convertidor [28], aumentando la eficiencia de ambos. Es aplicable en Máquina Herramienta, donde se podría conseguir un aumento de las prestaciones dinámicas.

2.1.3. Métodos de conmutación para IGBTs

En la actualidad, los métodos digitales son los más acordes a la hora de diseñar un generador de señales moduladas en ancho de pulso (PWM). A través de dos señales PWM complementarias se puede hacer conmutar una pareja de IGBTs. Tienen la ventaja de tener una menor sensibilidad a los ambientes con ruido eléctrico. Los métodos digitales suelen emplear circuitos con contadores y comparadores.

Las FPGAs han demostrado ser uno de los dispositivos preferidos a la hora de generar señales PWM para aplicaciones de electrónica de potencia. Se pueden conseguir señales de una frecuencia de conmutación muy alta [29] y tienen la ventaja de ser reprogramables.

En los apartados [6.4], [9.2] y [9.3] del presente documento se trata más en profundidad este tema.

2.2. Procesadores de control

Un sistema de control consiste en la interconexión de diferentes componentes para conseguir un propósito determinado. Hoy en día se da importancia a la eficiencia energética, el rendimiento y el coste para cumplir las especificaciones del diseño. El control es uno de los componentes que ayuda a aumentar la eficiencia de la conversión. Un controlador efectivo requiere por parte del ingeniero la comprensión del sistema, ya que tendrá que ser modelado.



Figura 2.3 Esquema de un sistema de control.

Hoy en día hay diferentes dispositivos electrónicos sobre los que se puede implementar un control. Se puede utilizar µP, µC, DSP, FPGA, ASIC, SoC… Hay multitud de plataformas diferentes, y cada una de ellas tiene sus ventajas e inconvenientes. Hay que conocer el

problema inicial y sus necesidades, y a partir de ahí se puede hacer un estudio sobre qué sistema embebido es el que mejor se va a adaptar a los requisitos particulares.

2.2.1. Microcontroladores y microprocesadores

Durante muchos años, sólo los microcontroladores y los microprocesadores se usaban como la única forma eficiente de implementar sistemas embebidos, debido a que eran programables.

La arquitectura hardware de los µP es fija y genérica, lo que los convierte en una plataforma barata. Estos dispositivos ejecutan secuencias de instrucciones que, generalmente, suelen estar almacenadas en una memoria ROM o FLASH.

En el caso de los µC, suelen venir acompañados de memoria, y con algunos periféricos digitales y analógicos integrados con un núcleo de procesamiento en un único chip. Con el objetivo de que sean baratos, existen µC con longitudes de palabra pequeña, memoria muy limitada y una velocidad de reloj muy baja. Estos cumplen perfectamente para algunos diseños que no necesiten de altas prestaciones. Por otra parte, hoy en día se pueden encontrar µC muy baratos, con longitudes de palabra de hasta 64 bits y con una velocidad de reloj del orden de cientos de MHz e incluso que llegan al GHz. Estos últimos tienen una potencia computacional suficiente como para hacer la tarea de DSP en algunos casos [30].

Con el desarrollo de las tecnologías de fabricación, que son más avanzadas y eficientes, el uso de los procesadores tiende más hacia el área de la computación. En los sistemas embebidos los µP han sido reemplazados por µC que, en la actualidad, también pueden traer consigo una unidad de procesamiento o CPU. Un ejemplo de esto último se puede encontrar en placas de desarrollo como la Raspberry Pi o la FRDM-K64F, ambas cuentan con un ARM que incluye µP y µC.

2.2.2. Procesadores de Señales Digitales

Los procesadores de señales digitales o DSP se han diseñado para tener multiplicadores y bloques de procesamiento, que permiten hacer operaciones aritméticas complejas. En comparación con los µC, los DSP tienen la ventaja de poder hacer una multiplicación en un único ciclo. Además, estos dispositivos tienen la capacidad de procesar datos en paralelos y de integrar bloques de memoria, lo que mejora en gran medida la velocidad de procesado.

Algunos DSP están optimizados para aplicaciones de control, y se les denomina *Digital Signal Controllers* o DSC. Tienen periféricos orientados al control, como pueden ser generadores de PWM, *watchdog timers* o interrupciones de respuesta rápida.

De todas formas, el inconveniente de los DSPs es que son más caros en comparación con las FPGAs. Normalmente se usan para el procesamiento de imagen y sonido, cuando un µC no tiene la capacidad computacional suficiente para hacer esta tarea en los tiempos estipulados. Dentro de la industria, los DSPs también se usan para el control de motores [30].

2.2.3. Field Programable Gate Array

Una FPGA es un circuito integrado que puede ser configurado tras su fabricación. Se programa a través de un lenguaje de descripción de hardware (HDL). En su interior contienen bloques de lógica programable (puertas lógicas) y una jerarquía de interconexiones reconfigurables que unen los bloques.

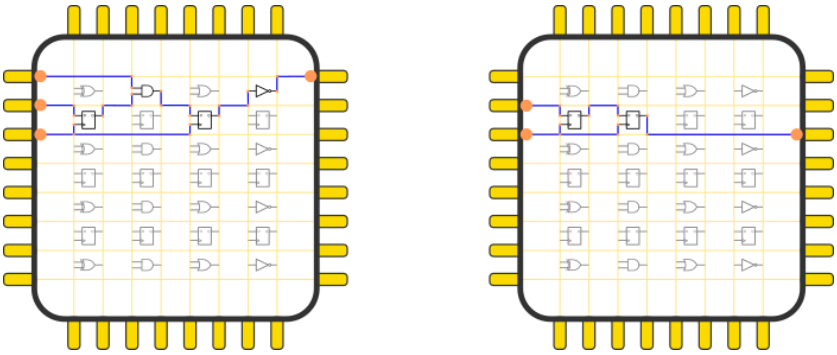


Figura 2.4 Esquema de la configurabilidad de una FPGA.

En comparación con los µC, son más rápidas, paralelizables y tienen una gran flexibilidad de diseño, ya que son reconfigurables. Por el contrario, tienden a ser más caras y difíciles de usar, aunque hoy en día se han reducido los precios y podrían a llegar a competir con algunos µC [30].

En un sistema empotrado, una FPGA se puede usar de dos formas, mediante la implementación de las funcionalidades directamente sobre la lógica digital o mediante la implementación de una arquitectura de microprocesador, lo que se denomina *SoftCore*. En los últimos años ha aparecido un nuevo escenario, y es que las FPGAs se están utilizando como aceleradores hardware, lo que permite su uso para implementar un hardware personalizado para una tarea concreta, como podría ser el paralelismo de datos. Operaciones que tardarían mucho tiempo realizarlas en un procesador pueden ser enviadas a una FPGA, preparada para resolverlas, y además en paralelo.

2.2.3.1. Principales fabricantes de FPGA

En el campo de las FPGAs los dos grandes fabricantes son Xilinx e Intel/Altera. La primera de ellas es la líder en cuanto a cuota de mercado. Pero desde que Intel compró Altera en 2015, esta segunda ha aumentado su cuota [31]. Aunque Xilinx tiene más mercado, entre los dos abarcan el 90% del total. En cuanto a cuál de las dos es la mejor, es algo que aún no se puede cuantificar, ya que en la actualidad no hay gran diferencia entre el rendimiento de las FPGAs de Xilinx y las de Intel [32]. Las razones son las siguientes [33]:

- Ambos ofrecen al mercado una amplia gama de dispositivos con sus respectivos precios, llegando así a distintos públicos.
- Ofrecen productos similares con poca diferencia de precio.
    - Procesos de fabricación a los mismos nm.
    - Bajo consumo energético.

  - Frecuencias máximas de reloj del mismo orden.
  - Numerosos IP-Cores.
  - Herramientas software (ISE/Vivado y Quartus II).
- Bancos de prueba con resultados controvertidos. Ambos afirman ser los mejores y tener un mejor rendimiento sobre su principal competidor.

Lo que sí que existe entre ambos fabricantes es una rivalidad a largo plazo, y esto es algo positivo, ya que hay una motivación y esto hace que siga habiendo avances e investigación por sacar los mejores productos.

Otros fabricantes de renombre son Lattice Semiconductors y Actel (actualmente Microsemi). Estas tratan de cubrir otro nicho de mercado, enfocando sus productos hacia un bajo consumo y un precio barato.

2.2.4. SoC-FPGA

Se denomina SoC-FPGA a todo sistema basado en una CPU, memoria y otros periféricos incluidos en un dispositivo lógico programable (FPGA).

Cuando la industria empezó a integrar FPGAs con CPUs (2002-2004), los sistemas SoC FPGA tuvieron un éxito muy limitado, sin embargo, el uso de *softcores* programados en la FPGA fue algo muy aceptado, mostrando de esta forma que el mercado tenía el deseo de que ambas tecnologías FPGA y CPU se uniesen [34]. Algunos de los factores fundamentales para introducir estos dispositivos en la industria son los siguientes:

- Eficiencia energética en el cómputo. La evolución de la computación está orientada hacia el paralelismo, y las FPGAs pueden servir como aceleradores Hardware para las CPUs. Un sistema SoC-FPGA puede maximizar la eficiencia energética y la flexibilidad del particionado de software, ya que puede soportar cientos de señales de datos para conectar diferentes áreas funcionales, esto proporciona un ancho de banda muy grande con latencias muy bajas.
  Dado que los aceleradores de hardware pueden tener más de 1000 veces la eficiencia energética que las CPU, diseñar para SoC-FPGA aporta una mejora significativa en el ámbito de la computación eficiente, más allá de lo que puede lograr el paralelismo multinúcleo.
- Reducción de tamaño en el proceso de fabricación (nm). Comparando los microprocesadores de Intel con las FPGAs de Altera de entonces se obtiene que, en el año 2000, las FPGAs tenían un proceso de fabricación de 130nm, y en las CPUs del mismo año era de 90nm. Sin embargo, en el 2011 las FPGAs se fabricaban a 28nm, cosa que en muy pocos procesadores se conseguía hacer. De hecho, en 2011 las mejores CPU de Intel bajo la denominación “Sandy Bridge” se fabricaban a 32nm [35] o por encima, fue en 2012, cuando llegó “Ivy Bridge” que se redujo el proceso hasta los 22nm [36].

- Adopción de las FPGAs para sistemas embebidos. La reducción de costes de esta tecnología propició su entrada en el mercado.
- La ley de Moore se vuelve cara. Hacer nuevos semiconductores es caro, sale más barato utilizar dispositivos programables. En este caso un SoC-FPGA tiene la ventaja de poder servir para diferentes mercados.
- Consolidación de la arquitectura de CPU. El término "procesamiento embebido" cubre un espectro de aplicaciones muy amplio, partiendo de sencillos procesadores de 4 bits hasta sofisticadas máquinas multinúcleo de 64 bits. Desde el año 2000 hasta el 2011 ha habido un cambio abismal, y el mercado está solicitando procesadores más rápidos y capaces. Las arquitecturas ARM, MIPS, PowerPC y x86 se están consolidando, y esto es debido a su compatibilidad y a la posibilidad de reutilizar software. Como resultado, un SoC-FPGA que incorpore una de estas arquitecturas de CPU puede apuntar hacia un mercado más amplio, incentivando a los proveedores para que inviertan en esta clase de semiconductores.

En el caso de los SoCs, no son como un µP, lo importante no es tanto el número de transistores a integrar, si no las funcionalidades que se agrupan en el mismo chip. Y hoy en día, la industria ya se ha fijado en las bondades que ofrece y lo integra en sus diseños.

Cuando se integra una FPGA en un SoC se le está dando al diseñador muchísima libertad, y con sus habilidades, podrá crear multitud de sistemas para diversas aplicaciones, el límite está en su imaginación. Con esta unión se pueden hacer infinidad de aplicaciones diferentes, desde diseñar aceleradores específicos que permitan mejorar algoritmos y hacerlos más rápidos y eficientes hasta crear interfaces programables específicas para comunicarse con el mundo exterior.

2.2.4.1. Hardcore (HPS ARM Cortex-A9)

En este proyecto se usa una FPGA Intel Cyclone V, que en realidad es un SoC que integra FPGA, CPU y periféricos. Se puede diferenciar en dos partes, el HPS y la FPGA.

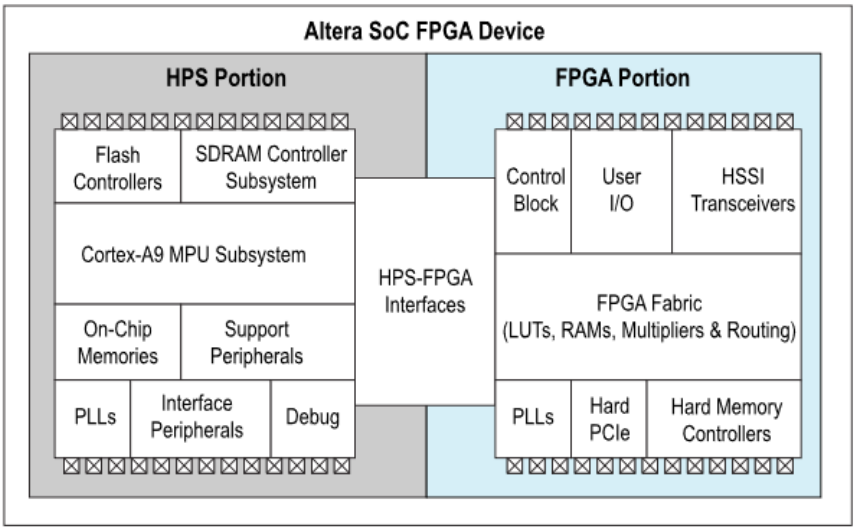


Figura 2.5 Diagrama de bloques de un SoC FPGA de Altera [37].

El HPS contiene una unidad de microprocesador (MPU) con un ARM Cortex-A9 de doble núcleo, además de memorias e interfaces de periféricos entre otros componentes. En el otro lado está la FPGA. Y para comunicar ambos dispositivos hay unas interfaces o "puentes" que hay que configurar. De esta forma se puede trabajar de forma independiente en la FPGA o en el HPS o tener a ambos dispositivos en funcionamiento. Eso depende ya del usuario y la aplicación que le quiera dar.

2.2.4.2. Softcore (Nios II)

Un microprocesador *SoftCore* o *soft processor* es aquel cuyo núcleo se puede implementar por completo usando síntesis lógica. De este modo puede ser implementado en diferentes dispositivos semiconductores que contengan lógica programable (FPGA, ASIC, CPLD…).

En la mayoría de los sistemas es suficiente tener un único SoftCore, aunque no quiere decir que no se pueda hacer. En un dispositivo se pueden instanciar tantos SoftCores como permita el número de puertas del dispositivo. Aun así, en el caso de hacer un sistema multi-núcleo, algunos recursos se pueden compartir entre todos los núcleos en un clúster. Es también una forma de implementar un paralelismo masivo en computación.

El procesador Nios II es un "*SoftCore*" que ha sido diseñado por Altera utilizando un lenguaje de descripción de hardware. Este procesador sigue los principios básicos de una arquitectura RISC (*Reduced Instruction Set Computer*) y usa un conjunto de instrucciones pequeño y optimizado [38].

Altera permite elegir hasta tres configuraciones diferentes para el componente Nios II. La opción más sencilla es la que se encuentra en la versión gratuita, se trata del Nios II/e, el núcleo económico. Está diseñado para un tamaño óptimo, pero no tiene ni arquitectura pipeline ni caché. Las otras dos versiones son más completas y añaden funcionalidades adicionales, sin embargo, sólo están disponibles en la versión de pago. La versión Nios II/s es el núcleo estándar, está diseñado para tener un tamaño pequeño a la vez que se mantiene un buen rendimiento. Consta de un pipeline de cinco etapas, caché de instrucciones y predictor de saltos estático. Por último, la versión más completa es la Nios II/f. Es el núcleo más rápido, diseñado para obtener el mejor rendimiento. Tiene un pipeline de seis etapas, caché de instrucciones y de datos y predicción dinámica de saltos. En la siguiente imagen se observa una tabla con las diferencias:

Nios II/e	Nios II/s	Nios II/f
RISC **32-bit**	RISC 32-bit **Instruction Cache** **Branch Prediction** **Hardware Multiply** **Hardware Divide**	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide **Barrel Shifter** **Data Cache** **Dynamic Branch Prediction**
Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache

Tabla 2.1 Comparativa entre configuraciones del procesador Nios II.

El procesador Nios II puede ser utilizado con otros componentes para completar un sistema más grande. A través de la herramienta Qsys se puede diseñar un sistema completo. Altera ofrece distintos módulos predefinidos o IP cores. Algunos ejemplos pueden ser una CPU, memorias, UART o incluso otros elementos de la FPGA, como controladores de red o interfaces PCI. También existe la posibilidad de usar módulos previamente diseñados por el programador.

Ya sea usando *HardCore* o *SoftCore*, la era del SoC FPGA está sólo empezando. Estos dispositivos han alcanzado un punto de inflexión debido a factores económicos, técnicos y de mercado, y su uso debe ser tomado en consideración por los arquitectos de sistemas [34]. Se deben evaluar los beneficios de estas plataformas, la reusabilidad de los módulos de propiedad intelectual (IP) y la tecnología de las FPGAs para las nuevas soluciones.

3

Especificación del desarrollo y objetivo

En este apartado se especifica la aplicación que se va a implementar. Se pretende mostrar de una forma clara y concisa en qué va a consistir el prototipo a desarrollar y el objetivo de este proyecto.

3.1. Especificación del desarrollo

Se va a obtener una aplicación de testeo y de puesta en marcha del inversor trifásico, se implementará sobre una arquitectura SoC-FPGA. Con esta aplicación se consigue facilitar la puesta en marcha de estos equipos y monitorizar las variables más importantes del sistema. Esta solución aportará un sistema robusto y paralelo, pero a la vez podrá ser configurable a través de una aplicación software que correrá sobre Linux RT.

La aplicación se encarga de gestionar el acceso a todos los periféricos disponibles en el hardware de control, pudiendo desde ella modificar los parámetros de seguridad de éstos. El software dispone de una interfaz de consola, que permite al usuario hacer los cambios sobre el inversor a través de comunicación serie.

Este desarrollo parte de un anterior diseño de una unidad de control para un inversor trifásico de potencia, que se hizo sobre un µC. Dados los nuevos requerimientos y retos que propone la industria, se desea utilizar nuevos materiales semiconductores en el inversor, concretamente SiC. Este material permite unas frecuencias de conmutación hasta tres veces superiores para la misma potencia respecto al Si. Para poder aprovechar las cualidades del SiC es necesario mejorar el controlador, por lo que se va a migrar la aplicación de testeo y de puesta en marcha a una arquitectura SoC-FPGA. El motivo viene dado a que un microcontrolador no tiene la suficiente velocidad para ejecutar los algoritmos de control cuando se usan estos semiconductores.

El nuevo controlador deberá hacer las mismas funciones que su predecesor, pero requiere de una mayor velocidad de procesamiento. Para poder ser usado con SiC, y aprovechar las ventajas de este material, la aplicación debe ser capaz de gestionar bajos tiempos de procesado. Poder hacer operaciones en paralelo es una gran ventaja, ya que se puede estar generando las señales al mismo tiempo que se están realizando adquisiciones sobre el estado del inversor. Para la nueva arquitectura se va a diseñar una serie de módulos que puedan realizar estas operaciones por separado y a la vez.

Al tratarse de una arquitectura SoC, existirá un doble control. Será la lógica programable la que se encargue de proteger al inversor, ya que es la forma más rápida. Por otra parte, las configuraciones y la monitorización se harán a través de software, ya que el tiempo no es tan primordial. Además, la aplicación dispondrá de un mecanismo de control, para que la puesta en marcha se haga de una forma segura. Esto consiste en asegurar que al inicio de la aplicación no habrá señales de disparo, y se hace de dos formas, desactivando la señal "*enable*" y poniendo las señales de conmutación a '1', de forma que el circuito de IGBTs quede abierto. En la placa interfaz existe una línea que conduce la señal "*enable*", los IGBTs sólo podrán conmutar cuando ésta esté activa.

El prototipo se probará sobre el inversor trifásico, verificando así su funcionamiento. Para ello se usará instrumentación de laboratorio.

La aplicación de testeo, además de controlar el inversor, tiene que ser capaz de monitorizar su estado, y de este modo garantizar su correcto funcionamiento. Esto significa que, a través de unos sensores, se tiene que conocer su corriente y tensión actual. Es un sistema de adquisición de datos y de generación de señales.

3.1.1. Comunicaciones

El controlador se tiene que comunicar con el inversor, tanto para enviar señales como para recibir información. Esto se hace a través de una placa interfaz, que se sitúa entre el inversor trifásico y el controlador. Esta placa es la encargada de adaptar magnitudes eléctricas y físicas a la escala del controlador o dispositivo embebido, además de integrar otros periféricos. Es un elemento necesario, ya que en el sistema hay componentes eléctricos y electrónicos de distinta naturaleza.

Para acceder a los periféricos de la tarjeta interfaz, y comunicarse con ellos, es necesario la implementación de protocolos de comunicación (I2C, SPI…) y de *drivers* específicos para esos dispositivos. Este procedimiento trae consigo cuatro etapas: especificación, diseño, desarrollo y verificación.

Las comunicaciones son una parte muy importante del sistema, ya que en la arquitectura del inversor existen sensores que proporcionan al controlador información sobre su estado. En determinadas circunstancias, como por ejemplo en casos de sobretensión o sobreintensidad, el controlador deberá tomar una acción con la finalidad de proteger la integridad del inversor.

3.1.2. Generación de señales de control

El controlador también tiene que ser capaz de enviar señales de control para testear el funcionamiento del inversor trifásico. Para ello será necesario la generación de señales PWM y SPWM. Al igual que antes, se hará un seguimiento de las cuatro etapas mencionadas, concluyendo en la verificación sobre el inversor trifásico.

La señal PWM será la encargada de actuar sobre el chopper. Esto elemento es una cuarta pareja de IGBTs que entra en conmutación como freno de emergencia. Cuando hay una sobretensión y el nivel de voltaje empieza a subir, éste entra en funcionamiento con el fin de reducirlo y hacer que el sistema vuelva a la normalidad. Sirve también como protección a las otras tres parejas de IGBTs.

La señal PWM, que gobierna el chopper, deberá ser de frecuencia y ciclo de trabajo variable. Además, la aplicación modulará de forma automática el ciclo de trabajo cuando entre en sobretensión. Lo hará en función de cómo de grande sea el sobrevoltaje respecto al máximo permitido. Cuanto más se acerque el voltaje al máximo, mayor será el *duty-cycle*.

Respecto a la señal de SPWM, es la que se encargará de conmutar las 3 parejas de IGBTs para obtener del inversor una señal sinusoidal trifásica. Tendrá que ser regulable en

frecuencia y en amplitud. Además, también tiene que ser posible cambiar la frecuencia de conmutación.

En los capítulos de implementación dedicados a las señales de PWM y SPWM se puede encontrar más información sobre la generación de estas señales.

3.1.3. Monitorización

Anteriormente se ha hablado sobre las comunicaciones y de cómo se recogían los datos. Una vez que se obtiene la información hay que interpretarla y darle una utilidad.

A través de los periféricos de la placa interfaz se accede a los sensores de corriente y tensión. Estos datos son capturados y traducidos a una magnitud que pueda ser interpretada por el usuario, en este caso, Amperios y Voltios.

En la aplicación, existe un apartado de monitorización, a través del cual, el usuario puede saber el estado de la tensión y corriente a la que trabaja el inversor. En caso de entrar en un estado de aviso, alarma o emergencia, el usuario es alertado.

Desde la aplicación, el usuario puede configurar cuales son los valores a partir de los cuales el sistema empieza a avisar, y cuáles son los máximos permitidos.

En la monitorización también existe un control automático. Y es que, se están analizando estos valores de forma constante. Cuando el sistema entre en unos umbrales que el usuario ha considerado peligrosos, la aplicación dará una respuesta, que puede ir desde encender el chopper e ir variando su ciclo de trabajo, hasta apagar por completo todos los IGBTs y dejar de conmutar.

3.1.4. Protección

Como se ha adelantado en el anterior apartado, existe un control automático basado en la monitorización del sistema, esto sirve para proteger al equipo ante situaciones de excepción.

Cuando la corriente está por encima de la máxima estipulada por el usuario, los IGBTs dejan de conmutar y entra en acción el chopper.

Otro método de protección se da con el sobrevoltaje. Cuando el voltaje es elevado y llega al estado de alarma, se activa el chopper, y su ciclo de trabajo es variable en función de lo grande que sea ese sobrevoltaje. Cuando llega al estado de emergencia, se abren las tres parejas de IGBTs y sólo queda el chopper conmutando al máximo *duty-cycle* posible. El chopper sólo deja de funcionar cuando las tensiones vuelven a la normalidad, se considera que esto ocurre cuando el voltaje está por debajo del valor de “aviso”.

Por otra parte, la conmutación de los IGBTs estará desactivada por hardware hasta que el usuario de la orden. De este modo se evita que los IGBTs entren en conmutación accidentalmente al arrancar la aplicación. También se han inicializado los parámetros por defecto a unos valores seguros.

Respecto a la conmutación de los IGBTs, también se ha añadido otro mecanismo de seguridad, el de los "tiempos muertos". Se ha añadido un tiempo de guarda entre las transiciones de '1' a '0', evitando de este modo que se pudiera dar el caso en el que dos IGBTs hagan un cortocircuito. De todas formas, también hay un mecanismo físico que evita este caso, pero no está de más tener otro método adicional de protección.

3.1.5. Integridad de la señal y cableado

No existe aún una placa interfaz cuyo zócalo coincida para integrar la placa de evaluación en ella. Por esta razón, el conexionado de pines se hará mediante cables.

Se va a hacer un estudio para analizar los transitorios que genera la FPGA en los flancos de subida y bajada de las señales que genera. Se pretende adaptar impedancias entre la placa de desarrollo y la tarjeta interfaz.

Por otro lado, se va a trabajar en un ambiente ruidoso, hay que evitar las interferencias y hacer un sistema provisional para conectar la placa de desarrollo a la tarjeta interfaz de forma sencilla y segura. Por ello se van a utilizar pares trenzados en los que uno de los cables de cada pareja irá conectado a GND, y el conjunto de cables irá apantallado, además de unas resistencias terminadoras para adaptar impedancias. En los dos extremos habrá unas tiras de pines, para hacer de zócalo en las dos placas.

3.2. Objetivo

El objetivo es la obtención de una aplicación de testeo y de puesta en marcha para un inversor trifásico de 110kVA de tecnología SiC. Las operaciones que tendrá que realizar el prototipo son las siguientes:

- Generar modulaciones PWM monofásico y trifásico, pudiendo modificar el ciclo de trabajo, la amplitud, frecuencia fundamental y frecuencia de la portadora.
- Monitorizar el estado del inversor.
- Proteger el inversor ante casos de sobretensión y sobreintensidad.

4

Metodología

En este capítulo se explican las pautas que se siguen para hacer el proyecto. Desde la selección de la tarjeta de desarrollo, hasta las pruebas para verificar el diseño de la implementación, pasando por el análisis del problema planteado y la forma de resolverlo.

4.1. Selección de la placa de desarrollo

Se ha decidido implementar este proyecto sobre una arquitectura SoC-FPGA. En el estado del arte se ha visto que hay dos grandes competidores y que ambos disponen de diferentes gamas, en función de la aplicación.

El primer paso para acertar al seleccionar una FPGA es recabar los requisitos del proyecto y ordenarlos por importancia. No consiste en obtener la mejor FPGA del mercado, si no la mejor para el proyecto en concreto. Para tomar una buena decisión se proponen estos seis criterios:

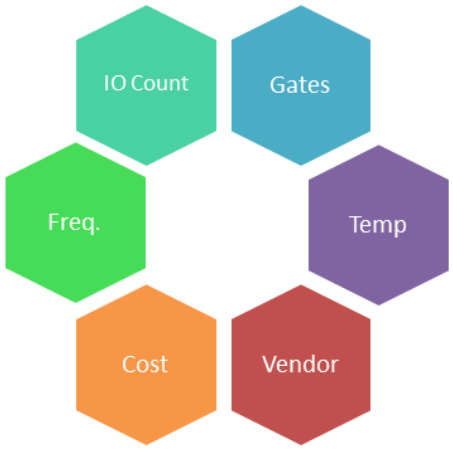


Figura 4.1 Lista de categorías a investigar para escoger una FPGA [39].

El requisito más sencillo es el número de pines de E/S. Hay que calcular o estimar cuantos pines va a necesitar el proyecto. La placa anterior tenía 76 pines de E/S, y había algunos que no se llegaban a utilizar. Así que se va a tener en cuenta esta referencia. Sí que es cierto que, siempre es buena idea que sobre algún pin, en caso de que haya que hacer alguna modificación o ampliación futura.

Otro factor a tener en cuenta es el número de puertas lógicas. Esto es difícil de estimar en una etapa temprana de desarrollo, pero la idea es que las FPGAs con muchas puertas, cuestan bastante más, por lo que la FPGA se debería de adaptar a cómo de grande sea el diseño. Una FPGA muy grande para un diseño reducido se traduce en una mala inversión.

La frecuencia de operación también es importante. Al igual que antes, mayor frecuencia suele ir ligado a un mayor coste. De todas formas, sabiendo la aplicación y que en la FPGA se pueden hacer diferentes tareas de forma simultánea, se puede estimar si la frecuencia de reloj va a ser suficiente para la aplicación a desarrollar o no.

En la lista de categorías aparece la temperatura. En este caso podría ser un factor a tener en cuenta, ya que, aunque esté bien ventilado, el inversor genera calor.

El coste suele ser uno de los factores que más se tienen en cuenta, sobre todo si el diseño se va a comercializar y hay que comprar muchas pastillas, ya que ese coste a su vez encarecería el producto. En este caso tiene importancia, pero no va a estar presente en un producto comercial.

Por último, pero no menos importante, están los vendedores. A la hora de seleccionar una FPGA conviene saber si el fabricante aporta una buena documentación, si el producto ha sido bien recibido y tiene una comunidad de desarrolladores detrás que aportan información, si el servicio de soporte contesta de manera rápida y útil…

4.1.1. Valoración de las necesidades

Para este proyecto se necesita disponer de una buena cantidad de pines (alrededor de 70) de propósito general de E/S (GPIO), podría ser bueno que trajera un ADC, ya que se realizarán medidas de valores analógicos, además es probable que el sistema vaya conectado a la red.

La arquitectura anterior se basa en un microcontrolador Kinetis, y éste es su esquema. Se desea hacer lo mismo, pero con una FPGA.

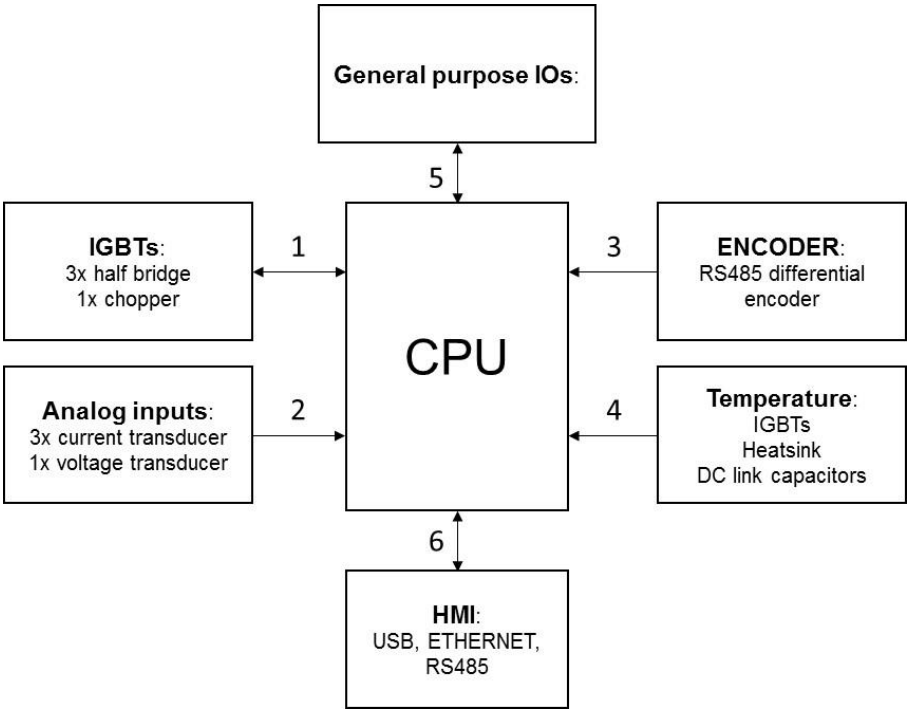


Figura 4.2 Interfaces de comunicación y conexión de dispositivos entorno a la CPU.

Para concretar más, estas son las especificaciones del proyecto:

1. Entradas analógicas:
    a. 4 elementos conectados a un bus SPI compartido con 4 CS (Chip Select).
2. Encoder:
    a. 3 señales de encoder en formato single-ended (A, B y S), a pines específicos del micro.
    b. 1 salida que unifica las alarmas de los 3 pares del encoder.
3. Medidas de temperatura:
    a. Bus SPI dedicado con 2 CS (Chip Select) para manejar 2 integrados con esta interfaz.
4. General purpose IOs (GPIOs):

a. 2 entradas 0-24 V lentas (máxima frecuencia de conmutación: 100 Hz).
b. 2 entradas 0-24 V lentas (máxima frecuencia de conmutación: 100 Hz) libres de potencial.
c. 2 salidas 0-24 V con carga máxima de 500 mA.

Señales TOTALES IOs: 2 salidas + 4 entradas

5. HMI (Human Machine Interface), buses de campo y comunicaciones
   a. USB.
   b. ETH.
   c. UART.

En resumen, sería apropiado encontrar una placa de desarrollo que tuviera como mínimo 70 pines de E/S, puerto Ethernet, USB, UART y un ADC. Además, el precio y el tamaño también serán tomados en consideración, se desea usar algo pequeño, con capacidad suficiente y que no sea demasiado caro.

4.1.2. Estudio de tarjetas disponibles

En el mercado existen multitud de placas de desarrollo con una FPGA. La lógica programable se está volviendo algo cada vez más común para construir sistemas electrónicos, y ahora no es raro integrar *SoftCores* o procesadores *HardCore* dentro del mismo chip. Esta evolución está desplazando a los procesadores de propósito general y a los ASICs. Además, hoy en día casi todos los sistemas de alto rendimiento tienden a ser implementados sobre FPGA.

Los dos gigantes de las FPGAs son Altera (Intel desde 2015) y Xilinx. Ambas compañías fueron fundadas en la primera mitad de los años 80 y entre las dos se llevan casi el 90% del mercado. De la misma época también es Lattice Semiconductors o Actel (actualmente Microsemi), que también son otras con renombre y tratan de cubrir otro nicho de mercado, enfocando sus productos hacia un bajo consumo y un precio barato.

A la hora de decantarse por una FPGA, resulta conveniente ver el soporte que dan los fabricantes. Por ejemplo, fijarse en qué aplicaciones trae, si son gratuitas, que ventajas y limitaciones ofrece… Otro punto de importancia son los manuales, guías y tutoriales, además de la comunidad que hay detrás, ya que escoger un dispositivo poco común puede derivar en tener más problemas a la hora de buscar información determinada sobre éste. Y lo más importante, sus pines, ya que a través de ellos se conectan periféricos y se realiza la generación y adquisición de señales.

Para elegir una placa de desarrollo conocer la FPGA que monta es vital. En el caso del presente proyecto, se busca un SoC, por lo que en la búsqueda se descartarán todas aquellas que no tengan un µP. Es también necesario saber qué componentes se van a montar y las interfaces de comunicación.

Se ha realizado un estudio evaluando diferentes placas de desarrollo con arquitectura SoC-FPGA. Aunque se han examinado más modelos, esta tabla es representativa de la oferta:

	 DE10-Nano	 ZynqBerry	 Raspberry PI 3 Model B	 Beaglebone Black	 Snickerdoodle
FPGA Device	Intel Cyclone® V SE 5CSEBA6U23I7N	Xilinx Zynq XC7Z010	-	-	Xilinx Zynq XC7Z010
ARM Processor	Dual Cortex-A9	Dual Cortex-A9	Quad Cotex A53	Cortex-A8	Dual Cortex-A9
Logic Elements	110K	28K	-	-	28K
Memory	1GB DDR3	512 MB DDR3	1 GB LPDDR2	512 MB DDR3	512 MB LPDDR2
Flash	MicroSD card	16MB/MicroSD card	MicroSD card	4GB On-Board / MicroSD card	MicroSD card
USB	USB OTG x1	USB x4	USB x4	USB x2	USB x1
Ethernet	Gigabit	10/100M	10/100M	10/100M	-
UART	USB to Uart	USB to Uart	-	-	USB to Uart
Video In	-	CSI-2 Connector	CSI-2 Connector	-	
Video Out	HDMI	HDMI/RCA/DSI	HDMI/RCA/DSI	Micro-HDMI	
Audio	-	HDMI/RCA	HDMI/RCA	Micro-HDMI	
FPGA Expansion	40-Pin GPIO x2/ Arduino expansion header x1	40-Pin header x1	-	-	100 I/O
HPS Expansion	LTC Connector	-	40-Pin header x1	46-Pin Header x2	54 I/O
Sensor	G-Sensor	-	-	-	
Other Interface	9 LED / 5 Buttons / 4 Switches	-	-	-	5 LED/ 2 Buttons
User LED/Button /Switch/7-SEG	ADC Conveter				
Price	130(Industrial)/99(Academic)	120	35	55	95

Tabla 4.1 Comparativa de dispositivos en el mercado que podrían servir.

La placa seleccionada finalmente ha sido la Terasic DE-10 Nano. Cumple con los requisitos solicitados y, además, Intel dispone de bastantes tutoriales y cursos, por lo que la documentación ha jugado a su favor. Experiencias pasadas, ajenas a este proyecto, han demostrado que tanto Intel como Terasic responden a las preguntas y peticiones de forma rápida a través de email. Por otro lado, el foro que tenía Altera no ha sido eliminado, si no que ha sido trasladado al dominio de Intel, de forma que hay información relevante que sigue siendo accesible.

4.2. Definición de la aplicación

La aplicación especificada se va a dividir en dos partes, la de lógica programable y la de software. Se ha pensado desde un punto de vista de optimización de recursos. La FPGA tiene en su propia naturaliza el paralelismo de tareas, es por ello puede controlar la adquisición de datos, las comunicaciones con periféricos y la generación de señales al mismo tiempo, sin ningún tipo de retardo. Por otro lado, el µP ARM es multihilo y tiene doble núcleo. Como se desea poder controlar la aplicación y monitorizar al mismo tiempo, se lanzarán dos hilos. Es la mejor manera de aprovechar las propiedades de ambos dispositivos con un fin común.

En la siguiente figura aparece un esquema de la división entre FPGA y HPS y la conexión de los periféricos:

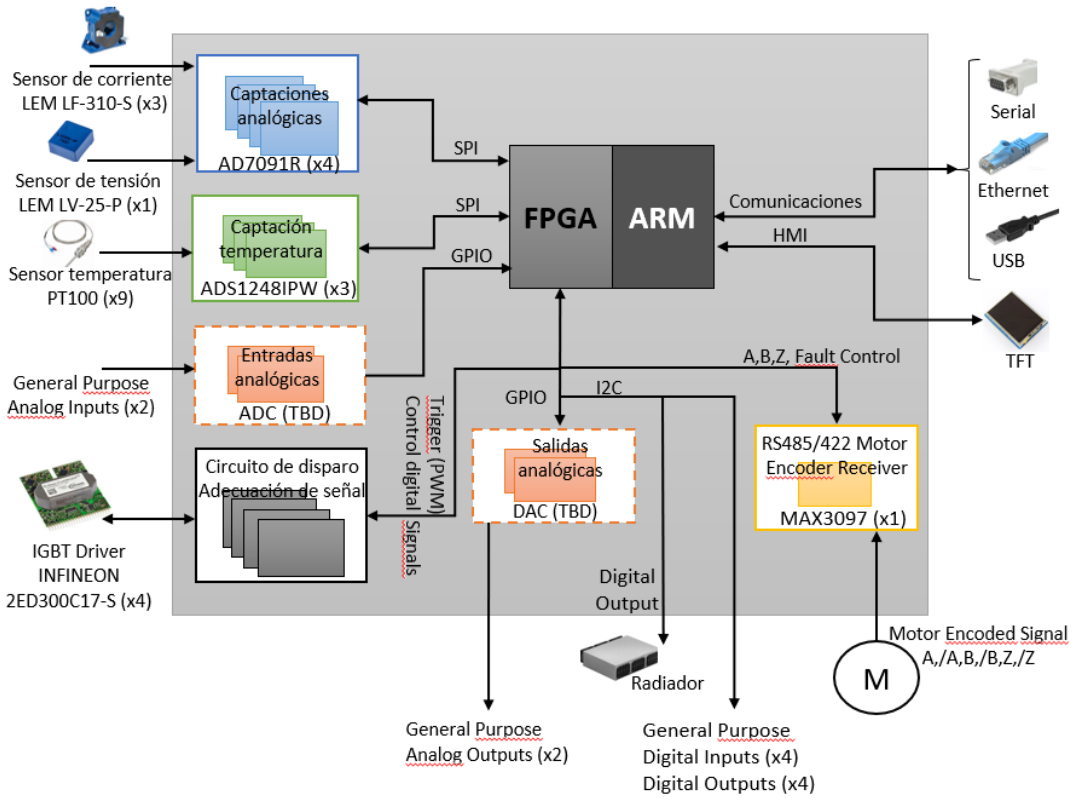


Figura 4.3 Arquitectura del controlador en SoC-FPGA.

La FPGA va a ser la encargada de generar las señales de control PWM y SPWM para la conmutación de IGBTs en el inversor. Al mismo tiempo, ésta tendrá que hacer un proceso de adquisición de datos, para conocer el estado del inversor. Para ello se implementará en lógica programable el protocolo SPI.

Esta división se produce porque la FPGA tiene la ventaja de poder estar haciendo las dos operaciones al mismo tiempo, y en caso de detectar una anomalía, al ser la primera en saberlo puede ser la primera en actuar, y los tiempos son algo fundamental en esta aplicación.

La parte del HPS tendrá en ejecución un sistema operativo en tiempo real Linux RT, y por encima habrá un software en ejecución que proporcione al usuario un entorno gráfico sencillo. A través de esta interfaz, el usuario podrá gobernar sobre el comportamiento del inversor, cambiando los parámetros de la aplicación de test, y también podrá conocer y monitorizar su estado, ya que el controlador avisará ante una incidencia.

4.3. Herramientas Software

Para la realización de este Proyecto van a ser necesarias algunas herramientas software. Al haber elegido una placa de desarrollo con una FPGA de Intel, se va a utilizar el entorno de desarrollo gratuito que proporciona esta compañía. A continuación, se expone un listado del conjunto de programas utilizado:

- Intel SoC FPGA Embedded Design Suite (EDS)
    - ModelSim 10.6d
    - Quartus Prime 18.1
    - SoC EDS Command Shell
    - Intel FPGA Monitor Program
- Mindjet MindManager 17

Intel SoC FPGA Embedded Design Suite es un conjunto de programas destinado al diseño, programación, verificación y síntesis para SoC FPGA. Se ha trabajado con la versión gratuita.

Para trabajar con Nios II y ARM se ha utilizado Intel FPGA Monitor Program. Sirve para compilar, ensamblar, descargar y compilar programas para esas dos plataformas. Además, permite la depuración del código en ejecución.

También se ha usado el software MindManager para el apartado de planificación, ya que esta herramienta permite hacer diagramas, y se ha utilizado para crear el EDT, el diagrama de Gantt y el *roadmap*.

4.3.1. ¿Por qué no emplear la generación automática de código?

En estos momentos es posible que el lector se pregunte por qué no usar una generación automática de código para este proyecto. Hoy en día hay herramientas como MATLAB, que combinándolo con SimuLink y, en el caso de FPGAs de Intel, HDL Coder, permiten hacer una generación automática de código. Esto podría ser una forma de afrontar el problema, pero no ha sido el camino a seguir. En su lugar se ha optado por la programación manual en VHDL, y las razones se exponen a continuación.

La primera razón es que la programación requiere talento. Hacer un programa puede parecer algo sencillo, pero hacer bien un programa exige muchos factores que deben ser tomados en consideración. Algunos de estos requisitos son la eficiencia, legibilidad, reusabilidad, que esté abierto a futuras ampliaciones o modificaciones… En resumen, hacerlo bien es una tarea complicada, un reto al que el programador se tiene que enfrentar. Y para ello es necesario conocer tanto el lenguaje, como el sistema de cómputo y sus posibilidades. Programar no es algo mecánico, y esa actitud es la razón de que muchos programas sean ineficientes, desestructurados, imposibles de mantener y, simplemente, erróneos [40].

En segundo lugar, la generación automática de código podría ser interesante para un prototipo inicial, pero en el caso de este proyecto se busca un alto rendimiento y un bajo consumo. Además, debe de abrir paso a futuros desarrollos, en los que podrá haber diferentes periféricos e incluso diferentes aplicaciones para el inversor.

Los generadores de código, a día de hoy, no están optimizados, los tiempos de respuesta tienden a ser más lentos y consumen más espacio en la FPGA [41] [42] [43]. Además, generalmente, necesitan mucha ayuda del desarrollador para obtener buenos resultados. Un programador puede conseguir que su código sea más eficiente respecto al automatizado, y aprovechar mejor los recursos del dispositivo embebido.

El único motivo que se podría tener en cuenta para hacer este proyecto es el de tener un prototipado rápido, pero eso no soluciona el problema en el tiempo. Es por ello, que se va a hacer todo el desarrollo programando la FPGA de forma manual.

4.4. Selección de lenguajes

Para el desarrollo del presente proyecto, se van a usar principalmente dos lenguajes, VHDL y C, eventualmente podrá salir algo en Verilog, pero será eventual y tendrá su traducción a VHDL.

Hay que diferenciar dos cosas, VHDL y C son dos lenguajes, pero el uno es de descripción de hardware y el segundo es de programación. VHDL describe el funcionamiento de un circuito digital, no algoritmos ni instrucciones de manipulación de datos. Hacer un código en VHDL siguiendo un estilo de programación software tiende a dar malos resultados. Describir un hardware requiere, en primera instancia, comprender y tener una imagen del hardware que se quiere generar.

En ningún caso, y menos en HDL, se puede pensar que el código lo es todo. Cualquier diseño tiene una fase previa antes de ser implementado. Los arquitectos de hardware tienen que tener la imagen mental del diagrama de bloque que quieren implementar, y, en caso de haberla, de la máquina de estados. Esto conlleva una etapa previa de diseño, que se recomienda hacer en papel. Los algoritmos son solo una parte de todo el trabajo que conlleva construir un hardware funcional. Adicionalmente, a la hora de hacer hardware, hay que tener muy en cuenta los tiempos, la sincronía y las señales de reloj.

Por otro lado, a la hora de ejecutar una aplicación en el lado del HPS se usará el lenguaje de programación C. Éste es, de lejos, el lenguaje de propósito general de más bajo nivel, e independiente de la arquitectura. Más bajo que C estaría ensamblador, pero ese ya es dependiente de la arquitectura.

En este caso, sí que se puede hablar, sin ninguna duda, de software. Y la programación software para este proyecto, también necesita a su vez de la comprensión del problema. Se debe considerar que el software va a hacer cálculos matemáticos para liberar a la FPGA de esta carga computacional. Por otro lado, C permite paralelismo de datos y de tareas, en este caso se va a implementar lo segundo. Se desea poder monitorizar el inversor y al mismo tiempo poder modificar sus parámetros de funcionamiento. Y, además, tiene otro añadido, y es que se va a hacer en tiempo real, utilizando Linux RT.

Con esto se pretende exponer las dificultades y la envergadura que presenta este proyecto. Que no se esté haciendo un circuito físico no implica que no haya un diseño de hardware. Al igual que, implementar una solución en software, no implica que no se estén tomando en cuenta las arquitecturas de hardware existentes y cómo hacer una programación de forma óptima para aprovechar todos los recursos que presenta el sistema. En este caso habrá que tener en cuenta la arquitectura del *Cyclone V* que trae la placa Terasic DE10-Nano.

4.5. Definición de bloques funcionales

Para la realización de este proyecto es necesario la implementación de bloques funcionales. Un bloque funcional es una parte que compone un dispositivo y se considera, a su vez, como si fuera un dispositivo totalmente independiente.

Se ha diseñado el siguiente esquema, cada cuadro representa un bloque funcional. Con esto se pretende indicar que cada módulo hace una función por sí sólo, pero cuando actúan en conjunto, se consigue que trabajen todos ellos para un fin determinado.

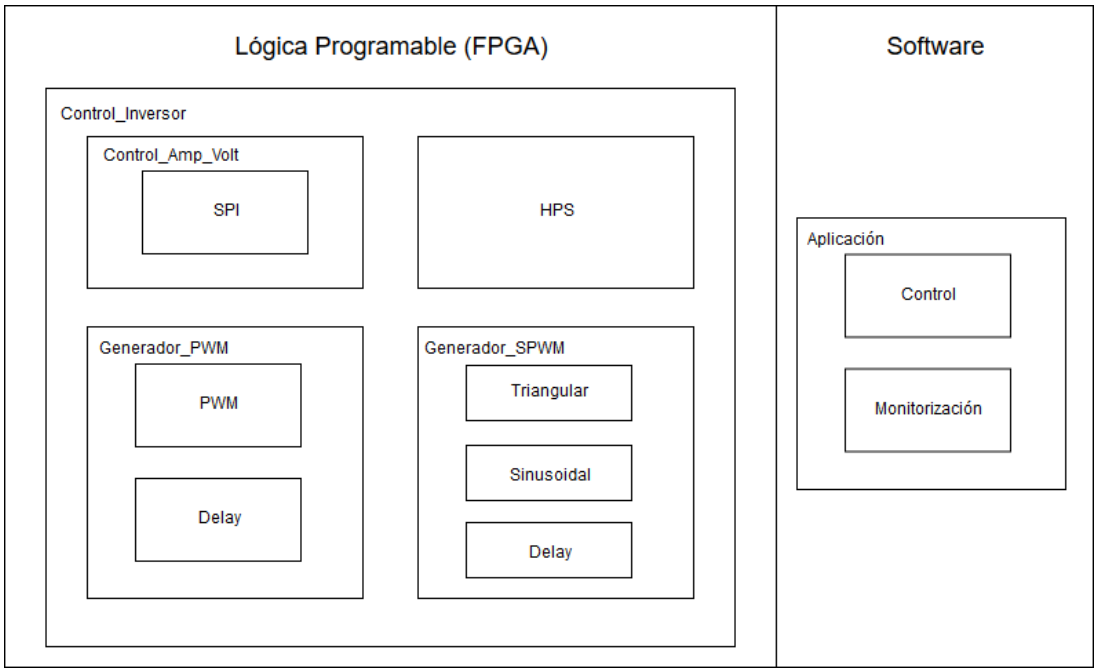


Figura 4.4 Esquema de bloques funcionales.

Los bloques deben de ser lo más genéricos y parametrizables posibles. Esto aporta modularidad y reusabilidad. En el caso del bloque "SPI", éste va a implementar la lógica del protocolo. Si se diseña desde un punto de vista genérico se consigue que el mismo módulo se pueda utilizar con otros dispositivos que usen el mismo protocolo, o al menos, simplifica la elaboración de un *driver* nuevo.

Al mismo tiempo, conocer los bloques funcionales necesarios permite dividir el problema principal en partes. Esto da una visión general del trabajo a realizar, sirve para fijar objetivos y ayuda a comprender el diseño.

Desde el punto de vista de la programación, se puede ver esta división como el algoritmo "divide y vencerás". Dado un problema, en este caso particular la aplicación de testeo y puesta en marcha del inversor, se pretende dividirlo en subproblemas. Si el nuevo problema sigue siendo complejo, se vuelve a aplicar el mismo mecanismo de división tantas veces como sea necesario. Cuando ya se han determinado cuales van a ser los problemas pequeños, se resuelven, y las soluciones convergen. Al final de este procedimiento se deberá obtener una solución al problema principal.

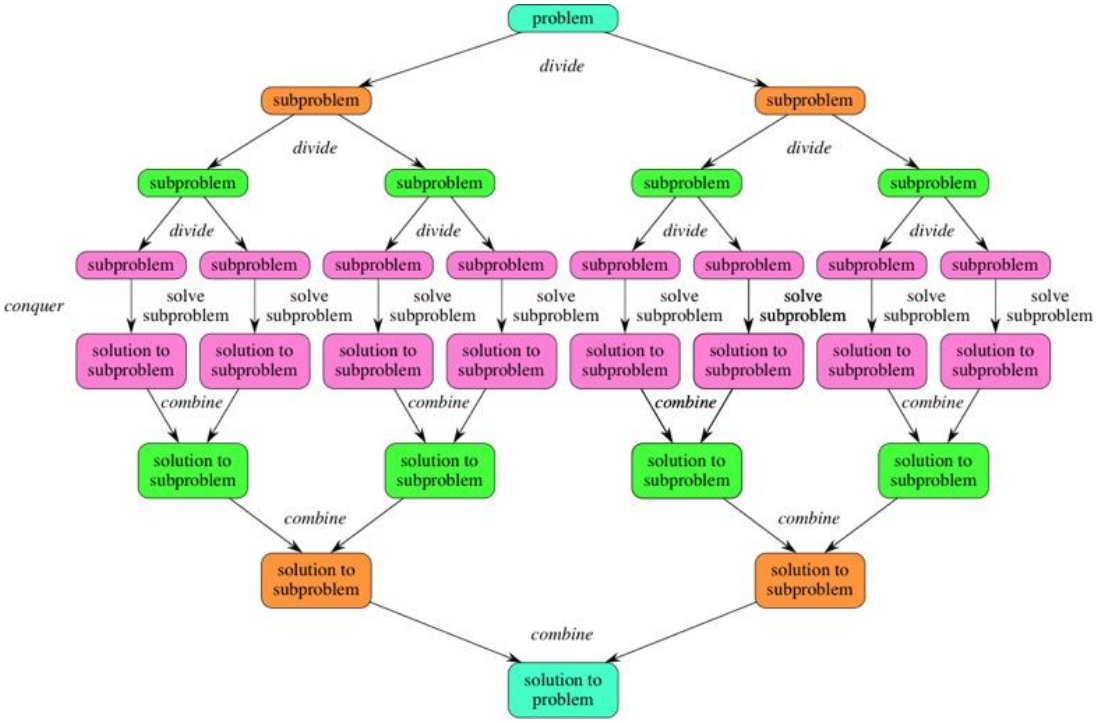


Figura 4.5 Algoritmo “divide y vencerás”.

4.6. Implementación/Integración

Como se ha visto en el apartado anterior, el problema principal se divide en conjuntos de subproblemas. La implementación de este proyecto es un proceso gradual, y se va a dividir en etapas. Antes de poder hacer un diseño complejo hay que conocer las herramientas, tecnologías y forma de trabajo. Se va a aplicar de nuevo el algoritmo de la figura 4.4, y se va a dividir el proceso de implementación en etapas.

4.6.1. Etapa 1: Inicio, familiarización y puesta en marcha

Es la fase que representa el inicio del proyecto. Existe una placa de desarrollo que no se ha utilizado antes y herramientas de laboratorio nuevas que hay que aprender a utilizar. También conviene estudiar más a fondo el entorno de desarrollo y ver qué posibilidades, aparte de la programación y sintetización del código, ofrece. En esta etapa se van a elaborar diseños exclusivamente en la parte de la FPGA.

Los desarrollos y las pruebas que se hagan no serán al azar, tienen una justificación. Habrá pruebas en las que se encienda un led, otras en las que se acceda a los interruptores o pulsadores de la placa de desarrollo, se probará a generar señales de reloj y PWM, en otros casos se escribirá o leerá a través de los GPIO… Adicionalmente, también se probará el *SoftCore* Nios II, dejando la puerta abierta a futuros usos fuera del proyecto.

El motivo de estas pruebas es que, en el proyecto, la placa tiene que comunicarse con otros periféricos. Hacerlas es pertinente, ya que con ellas se aprende a generar señales y a hacer lecturas y escrituras. Esto es lo mínimo de cualquier sistema de control.

Esta etapa culminará con el diseño de dos protocolos de comunicación, el I2C y el SPI. Para la implementación de estos dos diseños es necesario poner en práctica el trabajo anterior, ya que hay que generar señales de reloj a una velocidad a la que el dispositivo pueda responder, hay que leer y escribir a través de los pines GPIO y hay que hacer un control para que la transmisión ocurra en los tiempos estipulados y enviar y recibir correctamente los mensajes.

4.6.2. Etapa 2: Diseño de arquitecturas SoC

Esta etapa va a requerir de un aprendizaje previo, ya que será la primera que se trabaje con este tipo de arquitecturas.

El prototipo final está definido para que tenga una parte implementada sobre la lógica programable y otra sobre una aplicación software. Esto implica que debe de haber una fase de adquisición de conocimientos.

Las pruebas aquí van a girar en torno a quién gobierna el hardware. Se pretende probar el lado del HPS, el de la FPGA y luego unir ambos para que trabajen conjuntamente. Va a ser la antesala para diseñar sistemas más complejos.

4.6.3. Etapa 3: Real-Time Linux

En este punto del desarrollo se va a pasar a un sistema operativo de tiempo real. Va a haber un proceso de investigación para decidir qué sistema operativo utilizar o qué Kernel y módulos hay que compilar.

Por otra parte, el cambio de un SO o Kernel ha de estar justificado. No serviría de nada cambiarlo y que vaya igual o peor que su predecesor. Por eso se hará un banco de pruebas, observando en qué condiciones es mejor usar tiempo real o hacer una buena planificación del scheduler.

En esta fase también se va a probar el tiempo que se tarda en propagarse una señal en un sistema SoC, y se implementarán las interrupciones. Esto último pretende dejar el camino abierto para su futuro uso.

4.6.4. Etapa 4: Aplicación y prototipo final

Con todo el aprendizaje anterior, en esta etapa se pretende utilizarlo para desarrollar el prototipo final. Es el momento donde los subproblemas ya se han solucionado y convergen hacia una solución final

Algunos módulos se van a poder reutilizar o adaptar, pero también habrá que hacer módulos nuevos. Esta etapa está sobre todo orientada al control del inversor trifásico, por lo que también tendrá un apartado de pruebas.

Se van a probar tres módulos de forma independiente, y son el PWM, SPWM y el control de corriente y tensión. Tras hacer las pruebas por separado y comprobar su funcionamiento, se unirán en lo que será el prototipo final, y habrá una aplicación software para controlarlo, tal y como se ha especificado con anterioridad.

En esta etapa se tratarán otros problemas, como el cableado de la FPGA con el inversor y la mejora de la calidad de las señales, tratando de evitar ruidos u "*overshoots*".

4.6.5. Etapa 5: Pruebas sobre el inversor

En este punto ya existe el prototipo, es el momento de efectuar las pruebas sobre el inversor trifásico y validarlo. Para ello se acoplará la placa de desarrollo a la tarjeta interfaz y se probará contra el inversor.

Para las pruebas habrá tres escenarios, sin carga, con baja tensión (30-50 VDC) y alta tensión (500 VDC). Se registrarán los resultados obtenidos, demostrando así su funcionamiento. Se efectuarán las medidas del voltaje del inversor y sus corrientes en las tres fases. Para ello se usará material de laboratorio.

4.7. Pruebas unitarias

En programación, una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Esto sirve para asegurar que cada unidad funcione correcta y eficientemente por separado, además de verificar que el código cumple con su función. A la hora de hacerlo en hardware hay que tener muy en cuenta los tiempos y la posición en la que se hacen las operaciones de lectura y escritura.

La idea es escribir casos de prueba para cada bloque funcional, de forma que cada uno sea independiente del resto. Posteriormente se hacen las Pruebas de Integración, que consisten en juntar más de un módulo, para que funcionen en conjunto, y asegurar así el correcto funcionamiento de la nueva entidad. Por ejemplo, para probar el control de corriente y tensión, primero hay que garantizar que el módulo que implementa el protocolo SPI funciona, posteriormente, se podrá integrar con otro módulo y comprobar su funcionamiento. Lo mismo ocurre para el resto de los bloques funcionales.

Para hacer estas pruebas se va a utilizar ModelSim. De todas formas, esta aplicación no tiene únicamente la funcionalidad de probar el código generado, sino que también sirve para depurar y resolver problemas.

4.8. Pruebas sistema

Las pruebas unitarias son necesarias, y ahorran tiempo, ya que si algo no funciona en simulación tampoco va a funcionar cuando se programe el hardware. De todas formas, esto no es recíproco, puede funcionar en una simulación y dar fallos en el hardware.

Se está trabajando con periféricos ajenos a la FPGA y existe un cableado. ModelSim no es la única forma de probar la integridad del sistema, ya que desconoce la existencia de estos componentes. Por ejemplo, el protocolo I2C en ModelSim puede funcionar bien con una frecuencia de reloj muy alta, pero esto no quiere decir que vaya a funcionar en la práctica, ya que tanto los cables utilizados como la placa de evaluación soportan unas frecuencias de reloj máximas.

Las pruebas sobre el hardware se analizarán con material de laboratorio. Se usará polímetro, generador de funciones, fuente de alimentación, osciloscopio, analizador lógico y demás herramientas de trabajo. Se van a diferenciar en dos etapas.

Durante la primera etapa se usarán placas de evaluación que contengan los mismos circuitos integrados que la tarjeta interfaz. La tarjeta interfaz tiene periféricos, como un expansor de GPIO que usa el protocolo I2C o unos ADCs que se comunican mediante SPI. Una vez analizado su correcto funcionamiento, se probará en la tarjeta interfaz.

La segunda etapa consiste en diseñar un sistema que agrupe todas las funcionalidades y, una vez comprobado su funcionamiento sobre la tarjeta interfaz, se probará en el laboratorio con el inversor de potencia. Estas pruebas se dividen en tres escenarios que se explican con mayor detenimiento en el capítulo 10, pero se resume a tomar medidas de corriente y tensión con el inversor sin carga, con baja tensión (30-50 VDC) y en un escenario de alta tensión (500 VDC).

Por último, al concluir las pruebas sobre el inversor, se analizarán los datos obtenidos. En caso de observar alguna anomalía, se planteará una hipótesis y se tratará de dar solución al problema, volviendo a repetir el proceso de prueba y comprobación de resultados.

4.9. Sobre la metodología

Al igual que en los estudios de postgrado cursados, se ha empleado una metodología jerárquica. Se empieza por la selección de un dispositivo adecuado y se analiza el problema, planteando varias fases o etapas, y se desgrana en tareas de menor tamaño y complejidad. Los problemas se van resolviendo desde el más pequeño hacia el más grande, convergiendo en un sistema mayor. Posteriormente, se simula el trabajo realizado, y si éste es correcto, se prueba en un entorno pequeño. Finalmente, cuando se ha comprobado su funcionamiento y se han hecho las correcciones oportunas, se prueba en el entorno final, en este caso, el inversor.

5

Planificación

Apartado dedicado a la gestión del tiempo a invertir para completar el proyecto con éxito. Esto incluye la identificación de las tareas a realizar y su descomposición en subconjuntos de tareas más pequeñas, la estimación del tiempo, los hitos y el plan de contingencia ante posibles riesgos durante la vida del proyecto.

5.1. Introducción

El trabajo se va a realizar a lo largo del curso 2018-19. Se inicia en Octubre de 2018, y se avanzará de forma constante hasta el cierre del proyecto, que está previsto para Mayo. Tiene un contexto teórico-práctico, y en más de un punto va a ser necesaria una investigación previa para la adquisición de ciertos conocimientos.

Para un correcto desarrollo del proyecto se ha elaborado la siguiente planificación. Servirá como guía para conocer el estado del proyecto, el tiempo invertido y si se están aprovechando los recursos de forma apropiada, o si, por el contrario, hay que realizar ajustes en la planificación del trabajo.

5.2. Alcance

El proyecto tendrá una duración de 8 meses, desde Octubre de 2018 hasta Mayo de 2019. Se pretende tener acabado el proyecto para finales de Abril, quedando abierto el último mes ante posibles imprevistos, documentación o mejoras.

El objetivo principal es disponer, al cierre del proyecto, de un prototipo funcional que pueda realizar las operaciones necesarias para enviar señales de control y obtener información del inversor trifásico de potencia. Esto consiste en un sistema SoC-FPGA que interconecte los *drivers* y módulos de generación y adquisición de señales diseñados, y en una aplicación, a través de la cual el usuario pueda modificar los parámetros del inversor mediante comunicación serie. Para ello se deberán de cumplir las tareas representadas en el EDT.

5.3. Estructura de Descomposición del Trabajo (EDT)

El EDT es una herramienta que consiste en la descomposición jerárquica del trabajo a realizar. El propósito de éste es organizar y definir el alcance del proyecto. El dividir las tareas permite tener una visión más precisa del conjunto del proyecto, y es también una forma de comprobar el progreso. La estructura utilizada para el desarrollo de este Trabajo Fin de Máster es la siguiente:

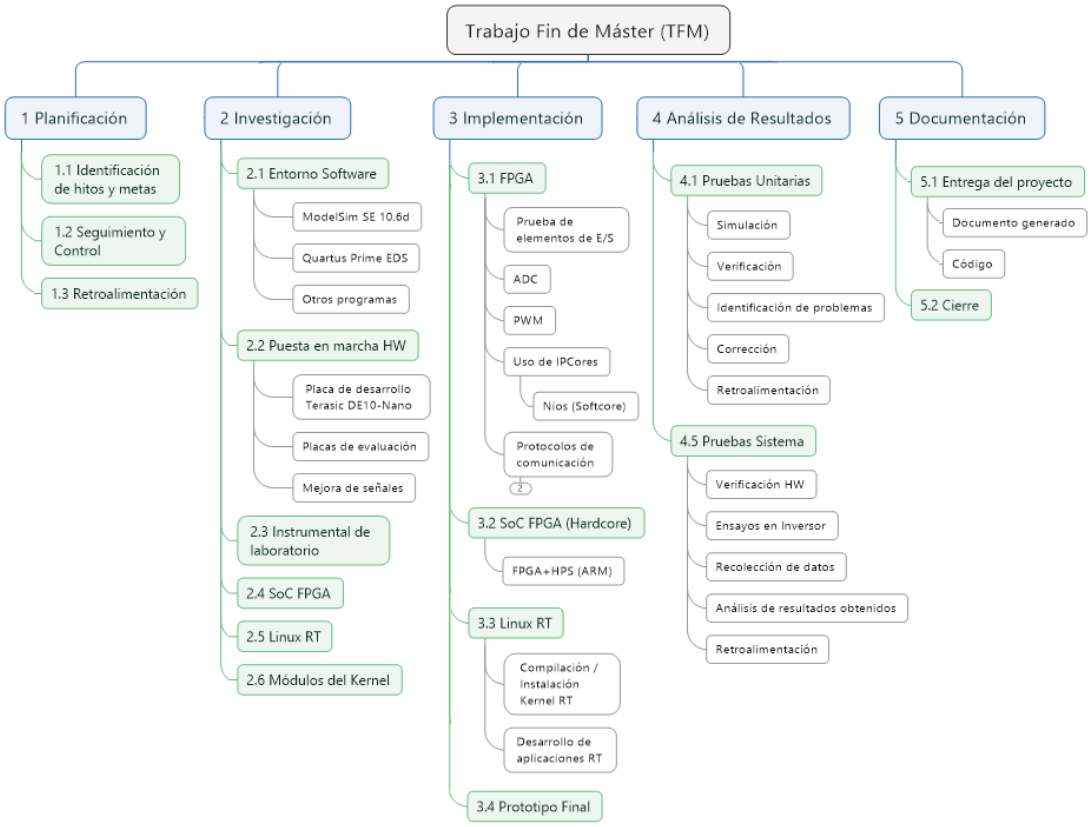


Figura 5.1 EDT del proyecto.

5.4. Diagrama de Gantt

El diagrama de Gantt es una herramienta gráfica que tiene como finalidad exponer el tiempo de dedicación previsto para realizar las tareas a lo largo de un tiempo determinado. Se muestra en la figura 5.2.

Lo primero que se ha hecho es una planificación estimada de las tareas. Servirá para definir bien los objetivos de cada fase y llevar un control del tiempo. La planificación dura todo el proyecto, ya que, como se observa en el EDT, hay una tarea de seguimiento y control y retroalimentación. Consiste en analizar el trabajo que se realiza y tratar de corregir las posibles desviaciones. Ante este caso, habrá que volver a planificar.

Al finalizar el día, se debe hacer un balance del trabajo realizado, y pensar dónde continuar al día siguiente. Si ha ido bien, habrá que pensar en la próxima tarea. Si por el contrario queda algo pendiente, habrá que analizar cómo abordar el problema, si seguir la misma línea o cambiar de estrategia.

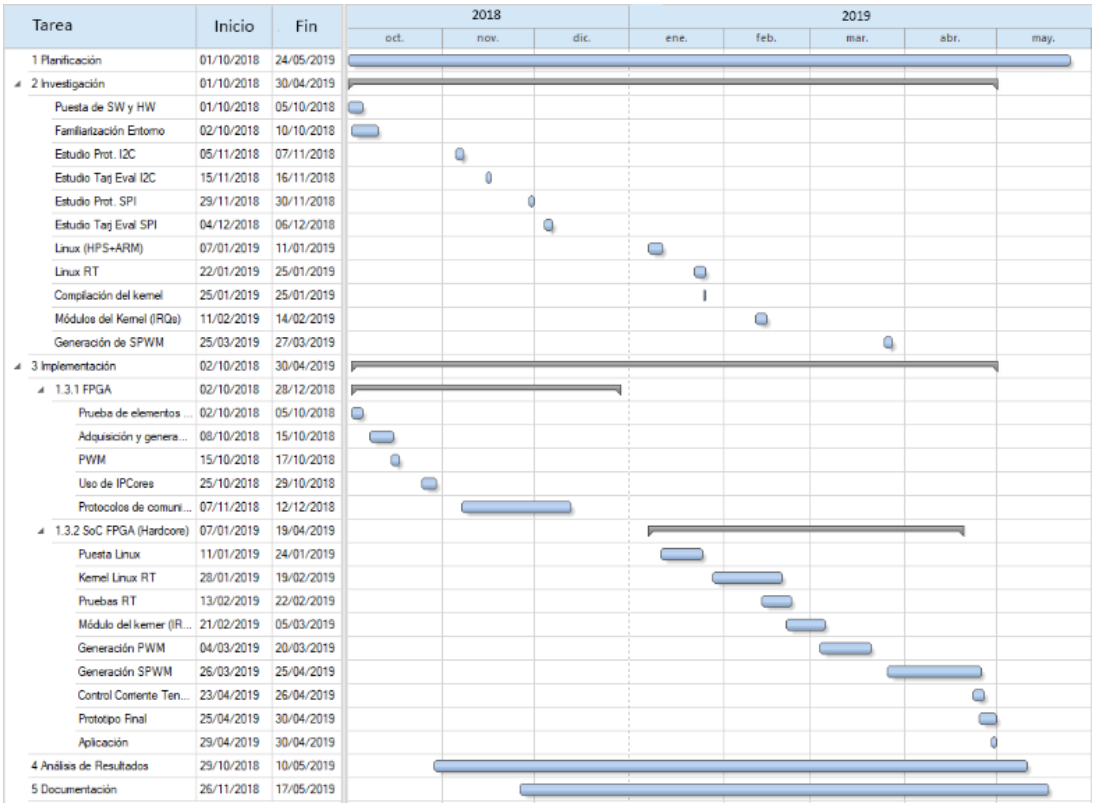


Figura 5.2 Diagrama de Gantt del proyecto.

5.4.1. Investigación

Es una tarea que durará todo el proyecto, pero no es una actividad continuada, por eso se ha dividido en subtareas. Habrá momentos en los que la labor investigadora tenga más peso, ya que existirá un desconocimiento y la investigación es algo que precede a la implementación. Primero hay que saber qué se pretende hacer y cómo hacerlo, y posteriormente afrontar el problema aplicando el conocimiento.

La puesta de SW y HW consiste en analizar qué dispositivo del mercado elegir para el proyecto y qué recursos de software serán necesarios. En cuanto a la familiarización con el entorno va dirigida al aprendizaje de las herramientas de trabajo. Por ejemplo, un IDE puede parecer fácil de utilizar, pero habrá herramientas que no se hayan usado con anterioridad y que habrá que aprender a utilizarlas. No se trata de manejar el entorno, si no de ser capaz de sacarle el máximo partido a sus funcionalidades. Lo mismo es aplicable a las herramientas de laboratorio, cuanto mejor se conozcan más provecho se obtendrá.

Por otra parte, están los protocolos de comunicaciones que se abordarán en este proyecto. El dispositivo se va a tener que comunicar con otros componentes periféricos, y lo hará a través de I2C y SPI. No se cuenta con ningún *driver* hecho de antemano o proporcionado por el fabricante para tal propósito. En este caso, se trata de implementar por completo los protocolos, por lo que habrá que tener los conocimientos de su funcionamiento a nivel físico y ser capaz de traducir esto a un lenguaje de descripción de hardware.

La siguiente fase del trabajo consistirá en usar el procesador ARM integrado. Para ello también habrá nuevamente subtareas, ya que se pretende tener funcionando tanto FPGA como µP al mismo tiempo y que se puedan comunicar entre ellos. Posteriormente, el desafío, consistirá en que eso ocurra en tiempo real, lo que llevará al diseño y compilación de un Kernel RT para Linux. Esto tendrá también un estudio previo de alternativas disponibles.

También se pretende probar las interrupciones, esto es algo que no se ha hecho con anterioridad en una arquitectura SoC-FPGA, y necesitará de su tiempo de investigación y de puesta en funcionamiento.

Para terminar, la generación de señales para el inversor tiene que ser de una manera precisa, y tienen que poder ser de amplitud y frecuencia variable. Habrá que hacer un estudio para conocer la forma óptima para implementar una señal SPWM con estas características.

5.4.2. Implementación

Consiste en llevar la labor investigadora a la práctica. Se ha dividido en dos partes, de Octubre a Diciembre se trabajará sobre el lado de la FPGA, y a partir de Enero se tratará de combinar la FPGA con el HPS, de forma que se aprovechará la tecnología SoC. Aunque la implementación vaya en paralelo a la investigación, es dependiente, no se puede diseñar hardware sin tener una base teórica.

La implementación se hará en pequeños bloques funcionales. En última instancia, se podrán juntar esos bloques para crear un sistema de mayor tamaño y complejidad. Hay que tener presente que las FPGAs funcionan de forma concurrente, así que diferentes módulos de hardware podrán estar funcionando al mismo tiempo y realizando diferentes tareas.

5.4.3. Análisis de resultados

La implementación debe ir siempre acompañada de una simulación. Por ello, durante el desarrollo de las descripciones de hardware, se efectuará un análisis del componente. Esto se hace mediante un banco de pruebas que abarque todas las casuísticas posibles, y comprobando que el componente pasa el test. Por otra parte, la simulación también sirve durante la fase de desarrollo para depurar y conocer lugares del código en los que puede haber errores o fallos de diseño.

Aparte de las simulaciones, también se harán verificaciones usando hardware real e instrumentación de laboratorio. Es la prueba de fuego para saber que además de en la simulación, el diseño funciona en el mundo real. Para ello se usarán placas de evaluación, polímetro, generador de señales, osciloscopio, analizador lógico…

5.4.4. Documentación

La documentación va en paralelo a todas las tareas anteriormente descritas, ya que todos los pasos realizados quedarán plasmados en texto e imagen. Al igual que con la investigación, no será algo que sea continuado, habrá días en los que tome mayor relevancia que en otros. Por ejemplo, al concluir uno de los hitos, la documentación puede cobrar una figura especialmente significativa, mientras que, en otros momentos, la documentación quedará reducida a la toma de apuntes o notas.

5.5. Hitos

Se han propuesto una serie de hitos o "milestones". Éstos sirven para marcar y conocer el progreso, ya que representan el haber conseguido un logro importante dentro del proyecto. Son una referencia, y permiten comprobar fácilmente si se ha cumplido en la fecha estimada inicialmente, por lo que son complementarios al diagrama de Gantt.

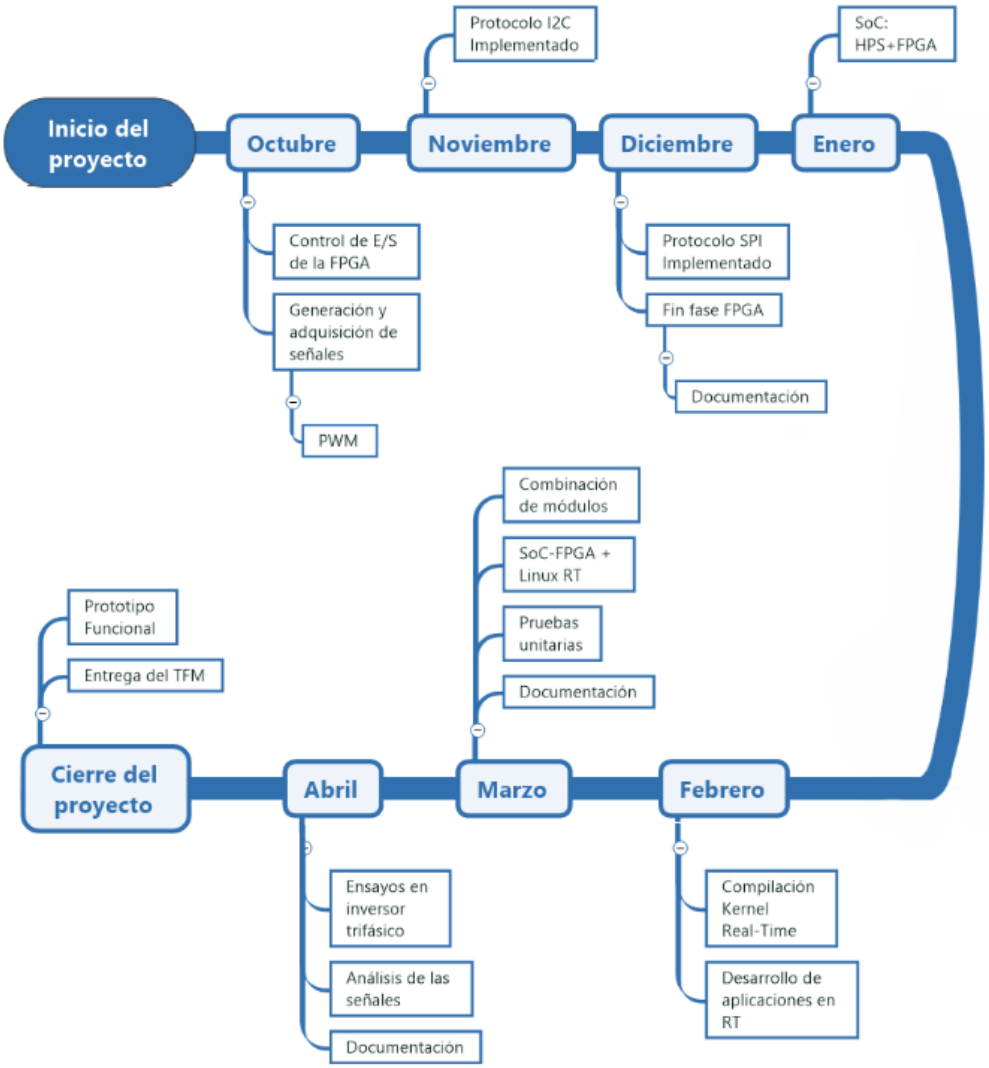


Figura 5.3 Hitos/*Roadmap* del proyecto.

En esta figura se pueden ver los hitos más representativos del proyecto, y también sirve como "*roadmap*". Muestra qué es lo que se desea tener acabado y sobre qué fecha.

Al principio del proyecto se desea empezar a trabajar con la FPGA y familiarizarse con ella. Los hitos giran en torno a la adquisición y generación de señales, completando la etapa con un PWM. Durante Noviembre y Diciembre se van a implementar los protocolos de comunicación. Para esto, es necesario la etapa previa de adquisición de conocimientos. La etapa de programación dedicada exclusivamente a FPGA terminará cuando se logre tener los dos protocolos funcionando.

En Enero empieza la arquitectura SoC. El primer paso para avanzar es conseguir tener un programa que use las dos partes, FPGA y HPS. Esto será otro hito más del proyecto.

En el mes de Febrero se pretende dotar a la placa de desarrollo con un sistema operativo en tiempo real. Hay que realizar un estudio previo, y los hitos son compilar un Kernel que añada esta funcionalidad y desarrollar algunas aplicaciones de muestra para aprender las bases de la programación en tiempo real.

Los meses de Marzo y Abril son para el diseño orientado al prototipo final. Sin duda, el hito más importante y representativo será empezar a probar el prototipo bajo el inversor de potencia. Finalmente, el proyecto finalizará con un prototipo funcional que sirva para testear y poner en marcha el inversor y con la entrega de la documentación.

5.6. Gestión de riesgos

Cualquier proyecto tiene riesgos que pueden afectar a la duración, desarrollo, calidad o viabilidad de este. Para evitar que un evento pueda influir en el proyecto hay que hacer dos cosas: Identificar los riesgos y elaborar un plan de contingencia.

5.6.1. Identificación de riesgos

- Planificación demasiado optimista.
- Tiempo de implementación.
- Recursos no disponibles al momento.
- Incertidumbre técnica.
- Herramientas desconocidas.

En esta lista se han recopilado 5 de los riesgos que probablemente surjan durante el desarrollo del proyecto. Ser consciente de lo que puede ocurrir permite la anticipación ante los problemas que puedan causar.

5.6.2. Plan de contingencia

Anteriormente se han identificado dos riesgos que pueden tener una misma solución. Una planificación demasiado optimista no es real, surgirán problemas o se retrasarán fechas estimadas. Lo mismo ocurre con la implementación, por norma general, las cosas no suelen funcionar a la primera, y algo que parece sencillo puede esconder una complicación mayor. Para evitar estos dos riesgos lo que se va a hacer es poner unas estimaciones con un margen de tiempo. En el caso de que se acabe la tarea antes de la fecha estimada, se puede aprovechar el tiempo restante para tratar de mejorar el trabajo realizado, hacer unas pruebas más robustas, documentar o empezar con la siguiente tarea.

Otro riesgo es que los recursos no estén disponibles. El proyecto se desarrolla dentro de una empresa y el material de laboratorio es de uso compartido. Si otros compañeros están haciendo pruebas y usando el material, habrá que esperar a que se libere. Lo mismo ocurre con los recursos software, y es que, para instalar un programa, tiene que hacerlo el administrador del sistema. El plan para abordar estos riesgos es tener más de una actividad a realizar y que no tengan dependencia entre sí. De este modo, si uno de los equipos del laboratorio o un programa informático no está disponible, se podrá realizar otra labor.

Los dos últimos riesgos detectados son la incertidumbre técnica y las herramientas desconocidas. Este proyecto supone un desafío, ya que el autor se va a encontrar con implementaciones que no había realizado con anterioridad y con arquitecturas nuevas. Esto supone una investigación previa sobre temas aún desconocidos. Por otra parte, la información disponible sobre algunos temas que se abarcan tampoco es numerosa. Ante este riesgo, el plan es recopilar información de interés y tener un sistema de información ordenado. Además de recurrir a tutoriales y hojas de características de fuentes fidedignas, como pueden ser los propios fabricantes.

5.7. Estimación del tiempo

El fin del proyecto está estimado para la última semana de Abril, quedando un margen de seguridad en caso de que se dé algún tipo de imprevisto. En el caso de que eso ocurra y que quede algo por hacer, comprobar, corregir o mejorar, se estima que tres semanas sea un tiempo suficiente para poder solventarlo.

Tareas	Tiempo (h)
Planificación	4h.
Investigación	70h.
Implementación	145h.
Análisis	25h.
Documentación	50h.
Total	**294h.**

Tabla 5.1 Tiempo estimado para cada conjunto de tareas.

6

Implementación: Diseños FPGA

En las siguientes páginas se recopilará toda la información relativa a la implementación del proyecto. Se explicará el trabajo realizado, y cómo se ha hecho. Además, habrá un análisis de los resultados obtenidos. Este es el primer capítulo dedicado a la implementación y trata del trabajo realizado exclusivamente sobre FPGA, es decir, la lógica programable.

6.1. Introducción

Esta ha sido la fase inicial del proyecto, que ha consistido en una breve adaptación al entorno de trabajo. Se ha instalado en el ordenador el software Quartus II acompañado de ModelSim. El primero de ellos se usa para la programación y síntesis del código, y el segundo para hacer simulaciones de los módulos a través de un banco de pruebas. Esta parte resulta de gran importancia, ya que las simulaciones reducen los tiempos de diseño, permitiendo detectar problemas y errores de antemano.

Para probar la placa se han seguido principalmente dos documentos, la "Guía de usuario [44]", el "Getting started [45]" y otros tutoriales de Intel/Altera. Se han hecho también algunos ejemplos sencillos que se expondrán a continuación.

6.2. Prueba de elementos de la placa

Se ha decidido probar el funcionamiento de algunos elementos básicos de entrada y salida de la placa de desarrollo, que vienen a ser los diodos led, los interruptores, los pulsadores y el reloj.

6.2.1. Leds e interruptores

Se ha escrito un módulo en VHDL que enciende los cuatro últimos diodos led en función de la combinación introducida por el usuario en los interruptores deslizantes. Con ello se comprueba su funcionamiento, además de que hay que leer la "Guía del usuario" para conocer con qué pin de la FPGA se corresponde, ya que hay que dejarlo indicado en el "pin planner" o "planificador de pines", que se encuentra en Quartus.

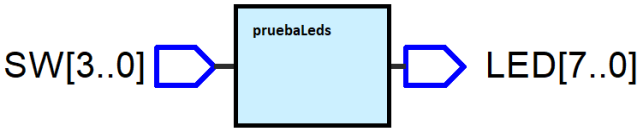


Figura 6.1 Módulo "pruebaLeds".

6.2.2. Reloj

Otro ejemplo para probar el funcionamiento del reloj fue generar un contador de segundos. Se trata de un programa que tiene dos contadores, uno para definir cuándo ha pasado un segundo y otro que funciona como acumulador, para contar el tiempo que ha pasado desde el inicio. El tiempo se representa a través de los leds de la placa en forma binaria, y una vez llegado al máximo, se reinicia.

Para trabajar en hardware también va a ser necesario la generación de relojes adicionales, por lo que también se han estudiado métodos eficientes para este cometido.

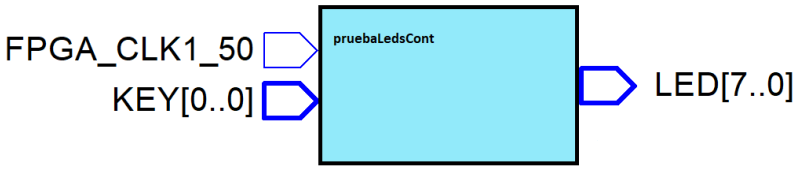


Figura 6.2 Módulo “pruebaLedsCont”.

6.2.3. Pulsador

En este caso se quiere probar el funcionamiento de los botones. Para ello se han creado un par de programas que los utilizan. El primero de ellos es muy simple, el pulsador se utiliza a modo de “toggle” o conmutador. De este modo aparece una salida por los leds cuando el botón está sin pulsar y otra diferente cuando está pulsado. En el segundo ejemplo que se ha realizado se usa el pulsador como botón de reset, y se va a usar mucho en posteriores ejemplos.

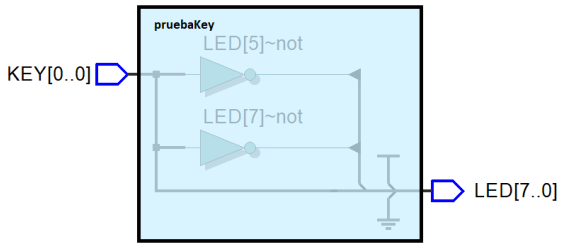


Figura 6.3 Módulo “pruebaKey”.

6.2.4. GPIO

El acrónimo GPIO significa “General Purpouse Input/Output” o Entrada/Salida de Propósito General. Esto significa que un GPIO es un pin genérico que puede ser controlado por el usuario en tiempo de ejecución.

Los pines de GPIO no tienen ningún propósito especial definido, y no se usan de forma predeterminada. En algunos casos es posible encontrar fabricantes que los añadan en algunos chips. Por ejemplo, Realtek los incluye en algunos de sus CODECs de audio, de forma que un tercero puede usar esos pines para añadir otras características al hardware.

En el caso de la FPGA, u otros dispositivos programables o placas de desarrollo, es habitual contar con estos pines, ya que no tienen una funcionalidad predeterminada. Con estos dispositivos se pueden hacer infinidad de cosas distintas, dependiendo del ámbito de uso.

Para comprobar el funcionamiento de estos puertos, se han diseñado en primera instancia y de forma separada las operaciones de escritura y lectura. Comprobar el funcionamiento de la salida de GPIO puede ser tan sencillo como escribir un ‘1’ lógico y comprobar el voltaje en el pin con un polímetro. Aunque complicándolo un poco más se puede sacar cualquier tipo de señal y comprobarla con la ayuda de un osciloscopio o analizador lógico. A continuación, se muestran unas imágenes con su correspondiente explicación:

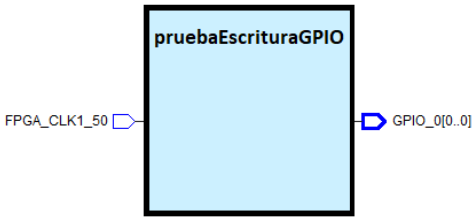


Figura 6.4 Módulo “pruebaEscrituraGPIO”.

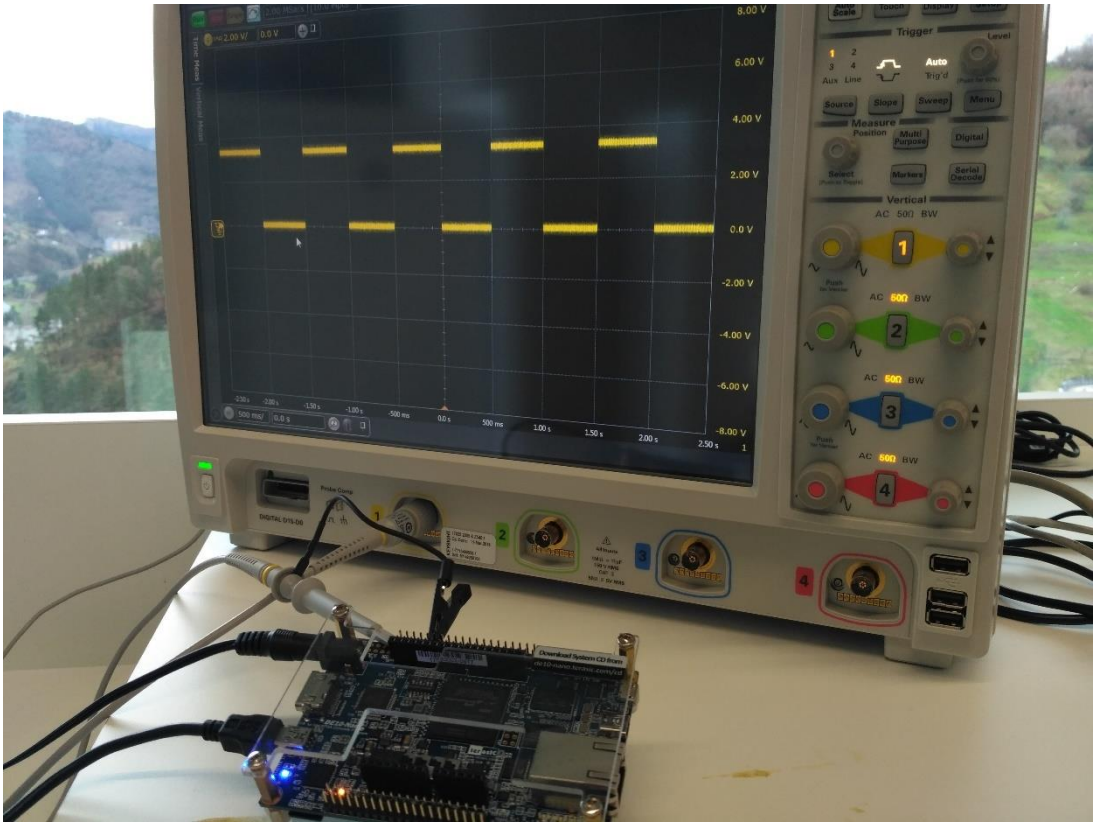


Figura 6.5 Generación de señal en FPGA y salida a través de GPIO.

En la imagen superior se encuentran la placa de desarrollo y un osciloscopio. Se ha diseñado un módulo en VHDL que genera una señal cuadrada periódica de 1Hz, con un ciclo de trabajo del 50%.

El siguiente paso es conseguir leer un valor a través de uno de los pines de GPIO. Para ello se ha diseñado un módulo que estará haciendo una adquisición de datos a uno de los pines. Para este ejemplo se ha establecido que en cada ciclo del reloj principal se muestree el canal, y que represente el valor leído a través de uno de los leds de la placa de desarrollo.

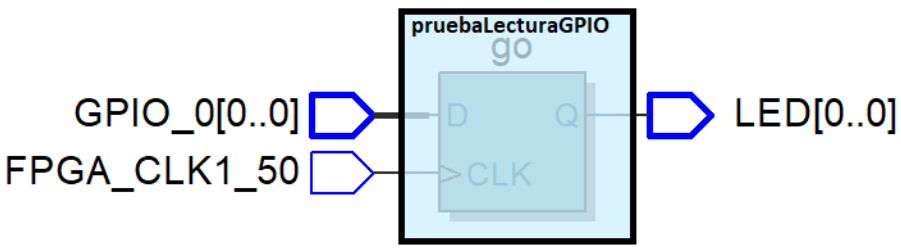


Figura 6.6 Módulo “pruebaLecturaGPIO”.

Se observa que lo que genera internamente el sintetizador a través del código generado en VHDL es un flip-flop D, es decir, su salida depende de su entrada.

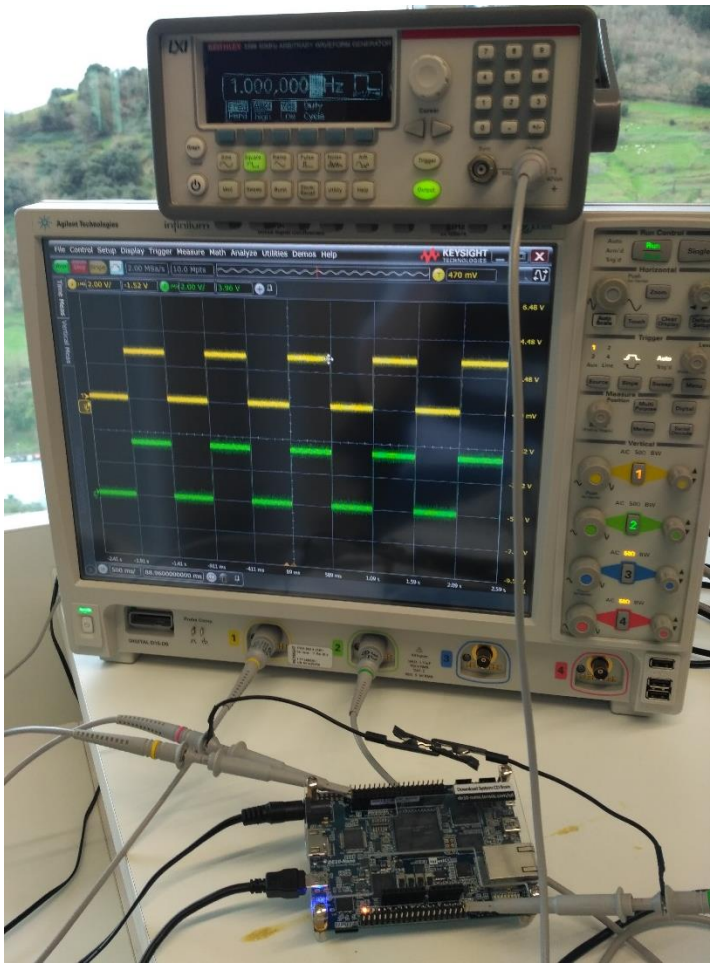


Figura 6.7 Adquisición y repetición de una señal cuadrada.

En esta otra figura se ha añadido un generador de señales. Se ha creado una onda cuadrada de 1Hz y la FPGA la está adquiriendo. Del mismo modo que la recibe, la vuelve a sacar por otro de sus pines replicándola. Se he introducido un valor tan bajo porque también tiene un led que parpadea cada vez que la señal está a '1'.

Estas dos aplicaciones que se muestran tienen un trasfondo. Generar una salida, o hacer una lectura de una señal, es necesario para realizar tareas de más complejidad. Por ejemplo, para hacer un PWM o un protocolo I2C, es necesario saber cómo generar una señal simple. Posteriormente, aplicándole otras normas o transformaciones, se puede aplicar este conocimiento para implementar otro tipo de aplicaciones.

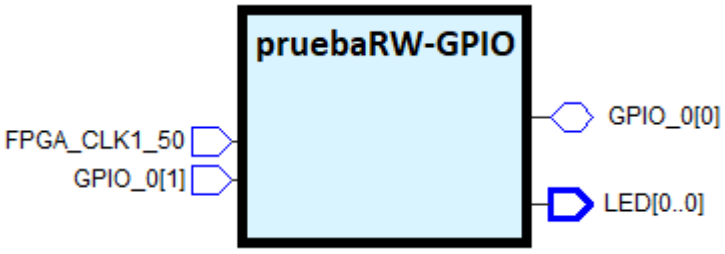


Figura 6.8 Módulo "pruebarw-GPIO".

6.3. Adquisición de datos

Una parte del proyecto está basado en el procesado digital de señales, en este caso se trabaja tanto con señales analógicas, que hay que transformarlas a digital para interpretarlas, como digitales. El procesado digital de señales consiste en la modificación matemática de una señal. Se trabaja en dominios discretos, generalmente en el dominio del tiempo y de la frecuencia.

Para trabajar con señales analógicas en un sistema digital hay que discretizarlas, de modo que sean compatibles con el procesamiento digital. Para ello se realiza un proceso que consta de tres partes: muestreo, cuantificación y codificación.

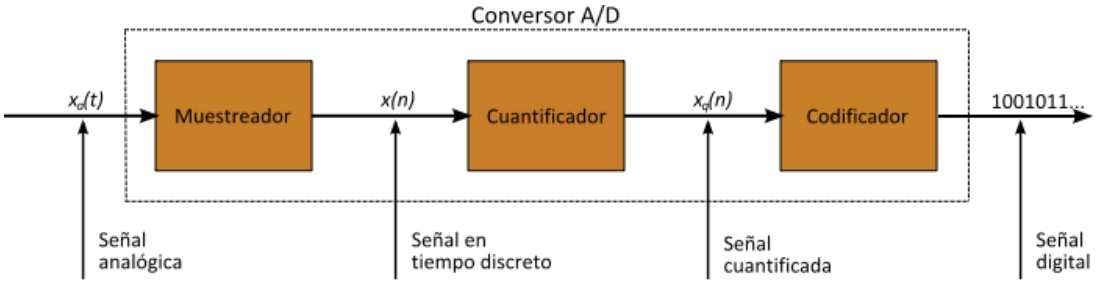


Figura 6.9 Esquema del proceso de conversión analógico-digital.

6.3.1. Muestreo

Una señal analógica puede tomar infinitos valores según la precisión con la que se quiera representar. En el mundo digital no se pueden representar con total exactitud, pero sí que se puede realizar una representación bastante precisa de la forma de la señal. El muestreo es una de las partes del proceso de digitalización de las señales. Se obtienen muestras de una señal analógica a una frecuencia constante aplicando el Teorema de Nyquist. Posteriormente se aplica la cuantificación, que asigna un valor binario a la amplitud de esa muestra.

6.3.1.1. Teorema de Nyquist

Es un teorema fundamental de la teoría de la información. Éste demuestra que la reconstrucción exacta de una señal periódica continua es matemáticamente posible si la señal está limitada en banda y la tasa de muestreo es al menos dos veces el ancho de banda (AB) de la señal analógica.

A la frecuencia *2*AB* se la denomina "razón de muestreo de Nyquist". Si ésta no se cumple, existirán frecuencias cuyo muestreo coincidirán con otras, a este fenómeno se le conoce como "aliasing" o solapamiento. El aliasing impide recuperar correctamente la señal cuando las muestras de ésta se obtienen a intervalos de tiempo demasiado largos. La forma de la onda recuperada presenta pendientes muy abruptas y aparecen frecuencias que no guardan parecido con la onda original. Éste fenómeno afecta más cuando se están digitalizando señales de alta frecuencia y no se identifica de manera correcta la frecuencia de muestreo.

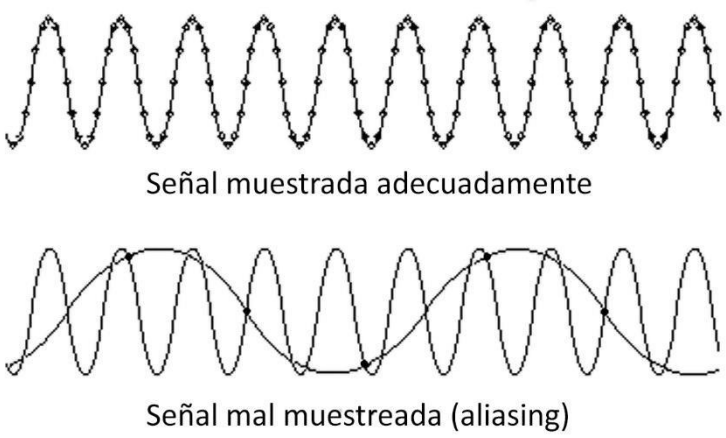


Figura 6.10 Efecto aliasing.

En la anterior imagen se observa el muestreo de una señal sinusoidal. La primera se está muestreando a una frecuencia adecuada, por lo que la forma de la señal resultante es una reconstrucción exacta. Sin embargo, en el segundo caso se está muestreando a una frecuencia insuficiente, por lo que la señal resultante dista mucho del aspecto de la original. Esto significa que no es representativa, y que está mal muestreada.

6.3.2. Cuantificación

El muestreo de una señal continua-analógica da como resultado una señal discreta en el tiempo. Mediante el anterior proceso se ha discretizado el tiempo, sin embargo, las amplitudes de las muestras obtenidas pueden tener infinitos valores. El proceso de cuantificación consiste en transformar el rango continuo de las amplitudes de las muestras de la señal de entrada en un conjunto finito de amplitudes. Normalmente, este conjunto está uniformemente distribuido (incremento constante) y se habla de cuantificación uniforme.

Durante el proceso de cuantificación se mide el nivel de tensión de cada una de las muestras obtenidas, y se les atribuye un valor finito de amplitud, seleccionado por aproximación dentro de un margen de niveles previamente fijado. En este proceso uno de los factores importantes es la resolución, la cual indica el número de valores discretos que puede producir sobre el rango de valores analógicos.

El proceso de cuantificación es un proceso con pérdidas. Debido al hecho de que se le asigna el mismo valor de reconstrucción a cualquier valor de entrada que pertenezca al mismo intervalo de cuantificación, se produce un error que depende de la señal de la señal de entrada y del valor de reconstrucción. Es un error intrínseco a la cuantificación, siempre que se introduce una etapa cuantificadora se produce este denominado "*Error de Cuantificación"*. La señal resultante tras este proceso es diferente a la señal eléctrica analógica y añade ruido. Si el ruido de cuantificación se mantiene por debajo del ruido analógico de la señal a cuantificar, el proceso cuantificador no tendrá ninguna consecuencia sobre la señal de interés.

La resolución determina la magnitud del error de cuantificación, y es que cuanto mayor sea la resolución menor será la diferencia en voltios entre un valor digital y su inmediatamente posterior. Con más resolución se consigue más precisión con respecto a la onda original.

Pese a que la señal ha sido digitalizada, aún no se ha traducido al sistema binario, esto se hace en la siguiente etapa, codificación.

6.3.3. Codificación

La codificación digital consiste en la traducción de los valores de tensión eléctrica analógicos, que ya han sido cuantificados, al sistema binario mediante códigos preestablecidos. Esta traducción es el último de los procesos de la conversión analógico-digital. El resultado es un sistema binario.

Normalmente se utilizan números en complemento a dos para representar todo el rango de valores discretos de la cuantificación uniforme, que suele ser una potencia de dos, y por ello, la resolución es expresada en bits. Para este proyecto se está usando un ADC con una resolución de 12 bits, por lo que hay 4.096 valores para representar la amplitud de la señal cuantificada.

6.3.4. Procesado

Una vez que la señal ha sido digitalizada llega el turno de efectuar el procesado o modificación de la señal.

El procesado puede tratar de aplicar filtros o mejorar la señal original. Para esta aplicación igual no es necesario una precisión de 12 bits, ya que un valor muy pequeño podría variar mucho los bits menos significativos, se podría aplicar por ejemplo un redondeo para aportar estabilidad al sistema.

En el caso de hacer operaciones matemáticas y comparaciones para efectuar cambios, hay que conocer cuál es el sistema de representación numérica con el que se está trabajando. Por ejemplo, se podría estar trabajando en complemento a 2 y representar una parte en negativo. O, por otra parte, se podría estar trabajando en binario natural, en el que se representa el número más bajo con todos los bits a ‘0’ y el más alto con todos los bits a ‘1’. En la siguiente figura se ilustra con un ejemplo un sistema de representación en el cual se aparece en una

gráfica cuantificada una señal sinusoidal con una precisión de 4bits en complemento a 2.

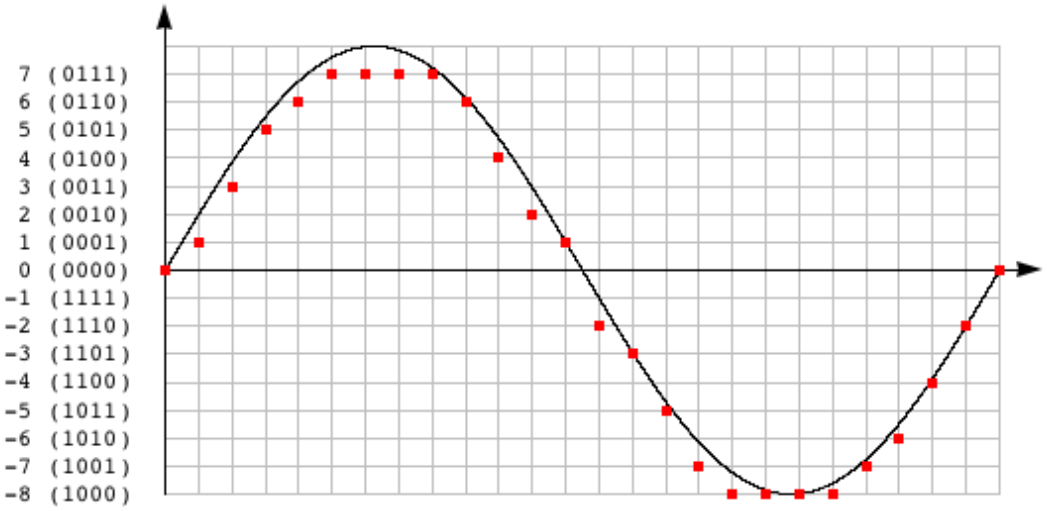


Figura 6.11 Señal sinusoidal muestreada en 4 bits en complemento a 2.

6.3.5. Conversor Analógico-Digital (On-Board)

La placa de desarrollo de Terasic tiene integrado un ADC o conversor analógico-digital LTC2308. Tiene 8 canales, su precisión es de 12 bits y puede muestrear a una velocidad de 500ksps. Para trabajar con el CAD se ha hecho de dos formas distintas.

En la placa se encuentra un cabezal de 2x5 pines que se corresponde con las entradas analógicas y su correspondiente alimentación. En la siguiente imagen se puede ver cómo es el conexionado:

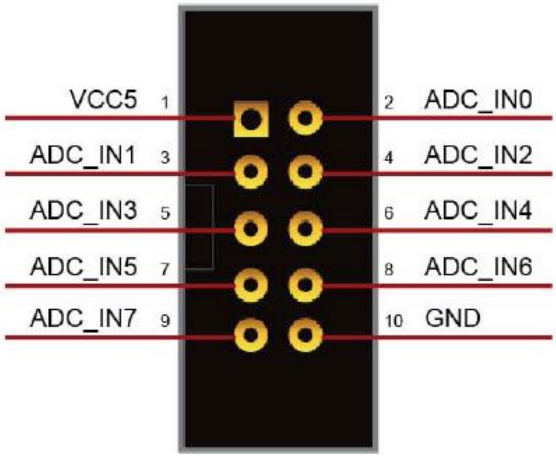


Figura 6.12 Patillaje del cabezal 2x5 correspondiente al ADC.

El ADC ha dado bastante juego, y por eso se han hecho tres implementaciones distintas, y una cuarta, que se explicará más adelante. En primer lugar, se ha implementado mediante Código VHDL, posteriormente se ha hecho a través de un IP Core que ofrece Quartus, utilizando Verilog y VHDL, y la tercera, se ha hecho con el IP Core y el procesador *SoftCore* Nios II. En el caso de la cuarta, se adelanta que va a ser, comunicándolo con el HPS.

El motivo de hacer esto es por cuestión investigadora. Con este ejemplo se pretende abrir camino a diferentes tecnologías que existen en Quartus y también a diferentes técnicas de trabajo. Por un lado, utilizar un módulo VHDL a modo de *driver* da como resultado un controlador específico para ese hardware concreto, y haciéndolo lo más genérico posible, podría ser reutilizable con mayor o menor esfuerzo. Por otra parte, Intel proporciona IP Cores. Estos son de una fuente de confianza y son robustos y fiables. Utilizar IP Cores, cuando están disponibles, ahorra tiempo de programación. Por último, con la introducción de Nios, se pretende mostrar cómo se puede introducir código en C en la FPGA para ser ejecutado a través de un *SoftCore*.

6.3.5.1. ADC por HDL

En la siguiente imagen se observa cómo se comunica el ADC con los elementos externos conectados a los pines, y cómo se comunica con la FPGA, la cual tiene el módulo LTC2308_cyclic programado en VHDL. Éste se encarga de la configuración previa del conversor y de la recepción de las señales.

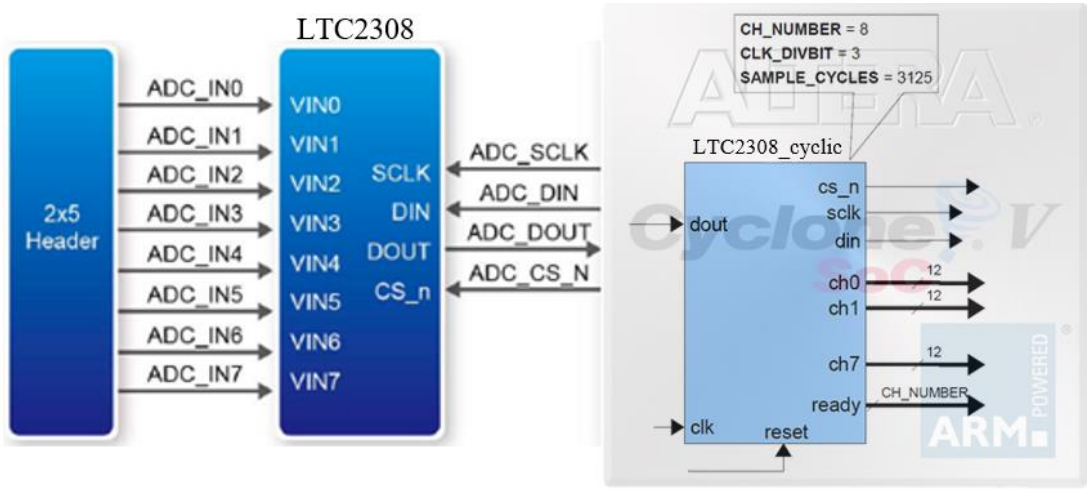


Figura 6.13 Conexiones entre el cabezal, el ADC y el controlador diseñado.

La imagen superior representa, por un lado, las entradas analógicas del cabezal, y por otro lado se observa el CAD intercambiando señales con el módulo LTC2308_cyclic, que actúa como controlador. Las entradas son las siguientes, ADC_SCLK (le llega una señal de reloj), ADC_DIN (entrada de datos digital), ADC_CS_N (señal de control para la selección de canal). La señal de salida es ADC_OUT (salida de datos digital). En la siguiente imagen se observa a mayor escala el módulo que actúa como controlador, presente en la FPGA.

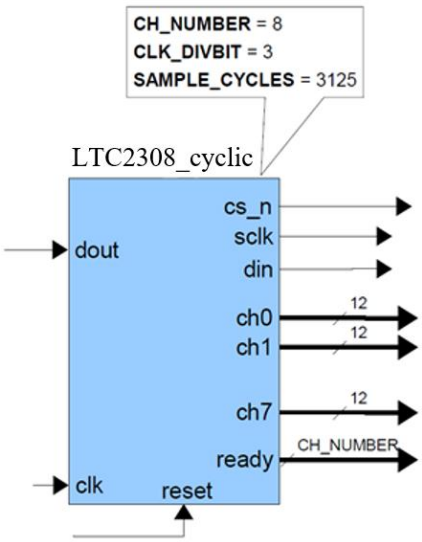


Figura 6.14 Esquema del módulo "LTC2308_cyclic".

Los parámetros para configurar el CAD son CH_NUMBER (número de canales que se van a utilizar), CLK_DIVBIT (el factor por el que se divide la frecuencia de CLK, al generar SCLK. Es un valor entre 1 y 11) y SAMPLE_CYCLES (Periodo de muestreo, en ciclos de SCLK).

En la entrada está la señal dout, (la línea de entrada), y en la salida están las señales cs_n (el flanco de bajada que indica el comienzo de una captura), sclk (reloj para el CAD), din (línea serie de salida), ch*N* (último dato capturado en el canal *N*) y ready (indica que se ha terminado de capturar el número de canales seleccionados por el parámetro CH_NUMBER).

Se ha reutilizado un módulo en VHDL capaz de interactuar con el ADC. Se trata de un módulo que se utilizó en el Trabajo Fin de Grado (TFG). En aquel entonces se hizo un pedal de efectos de sonido digital, y se utilizó el ADC como método para modificar parámetros y efectuar cambios sobre la señal.

6.3.5.2. ADC por IP-Core

En este otro caso se usa el IP Catalog para incluir el controlador ADC en un proyecto de hardware. La versión del controlador del IP Catalog permite acceder a entre dos y ocho canales, con sus valores actualizándose de manera automática.

Para instanciar el controlador se hace a partir del IP Catalog. Hay que buscar el componente que se desea instanciar. En este caso se ha seleccionado el controlador de ADC, y el asistente pregunta por el dispositivo sobre el que se va a trabajar. Hay que darle un nombre a la entidad y configurar los valores con los que se desea trabajar. En este caso se está trabajando con la Terasic DE10-Nano, por lo que en Device hay que poner el identificador correcto de la FPGA que lleva integrado, y en el desplegable habrá que elegir la placa correcta. Se le ha asignado una frecuencia de 12,5MHz y se usan los 8 canales.

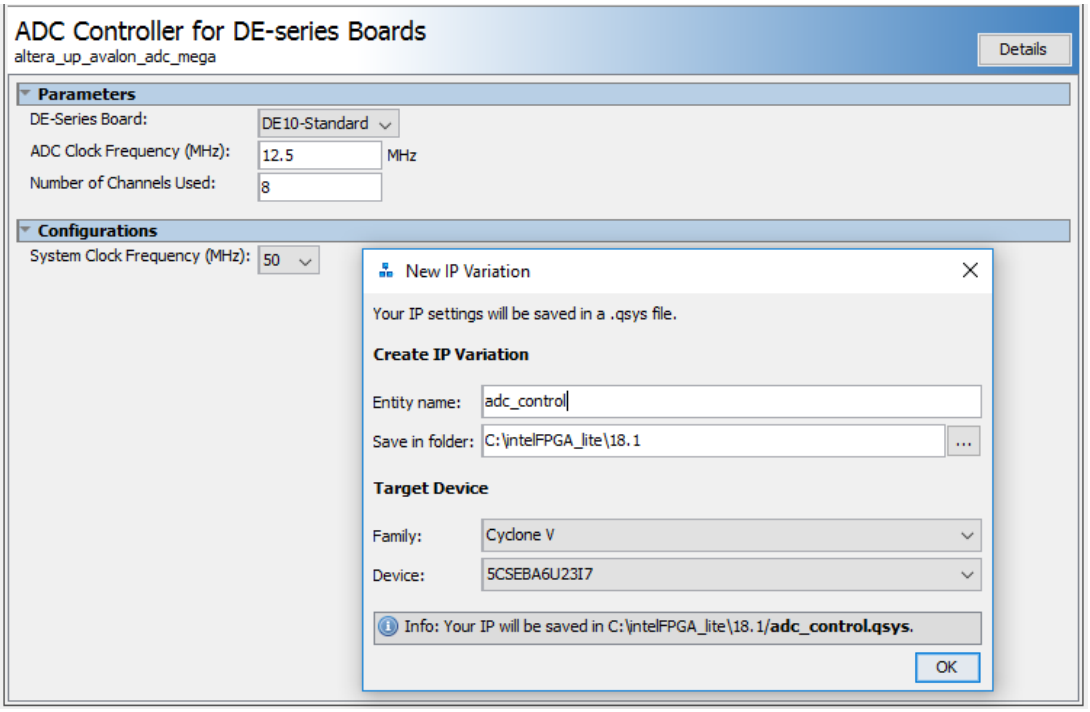


Figura 6.15 Pantalla para configurar el IP-Core.

Para generar el código del controlador se ha seleccionado Verilog. De todas formas, también se ha desarrollado otra versión del mismo código en VHDL. Ambas son igual de válidas. En cuanto a los parámetros, se ha seleccionado en el desplegable la placa con la que se está trabajando "DE10-Nano", y una frecuencia de reloj de 3,2Mhz.

A continuación, hay que crear una entidad que invoque al módulo. Hay que tener presente los nombres que da la generación automática a las señales, para eso hay que mirarlo en el archivo de código Verilog que se ha generado. En este caso se ha generado el fichero "adc_control.v", que se encuentra en la carpeta "..\adc_control\synthesis".

El código de la entidad de más alto nivel es el siguiente:

```
module pruebaADC_IP (CLOCK_50, KEY, SW, LED, ADC_SCLK,
        ADC_CS_N, ADC_SDAT, ADC_SADDR);

        input CLOCK_50;
        input [0:0] KEY;
        input [2:0] SW;
        output [7:0] LED;

        input ADC_SDAT;
        output ADC_SCLK, ADC_CS_N, ADC_SADDR;

        wire [11:0] values [7:0];

        assign LED = values [SW][11:4];

        adc_control ADC (
            .CLOCK (CLOCK_50),
            .RESET (!KEY[0]),
            .ADC_SCLK (ADC_SCLK),
            .ADC_CS_N (ADC_CS_N),
            .ADC_DIN (ADC_SADDR),
```

```
            .ADC_DOUT (ADC_SDAT),
            .CH0 (values[0]),
            .CH1 (values[1]),
            .CH2 (values[2]),
            .CH3 (values[3]),
            .CH4 (values[4]),
            .CH5 (values[5]),
            .CH6 (values[6]),
            .CH7 (values[7])
        );
endmodule
```

6.3.5.3. ADC por Software (NIOS)

En este caso se va a controlar el ADC por software. Para sistemas complejos en los que se desea usar un procesador y software, el ADC puede incluirse como un componente QSys, que es compatible con Nios II y con el procesador ARM.

Se ha generado un sistema utilizando la herramienta "Platform Designer", conocida en versiones anteriores de Quartus como "Qsys tool". El sistema será controlado por un procesador y el software, y los datos obtenidos y digitalizados por el ADC se mostrarán a través de los leds.

Hay cuatro componentes principales en el sistema, que son el controlador del ADC, el procesador Nios II, memoria on-chip y los leds. Además, será necesario el reloj del sistema. Una vez diseñado, la arquitectura queda del siguiente modo:

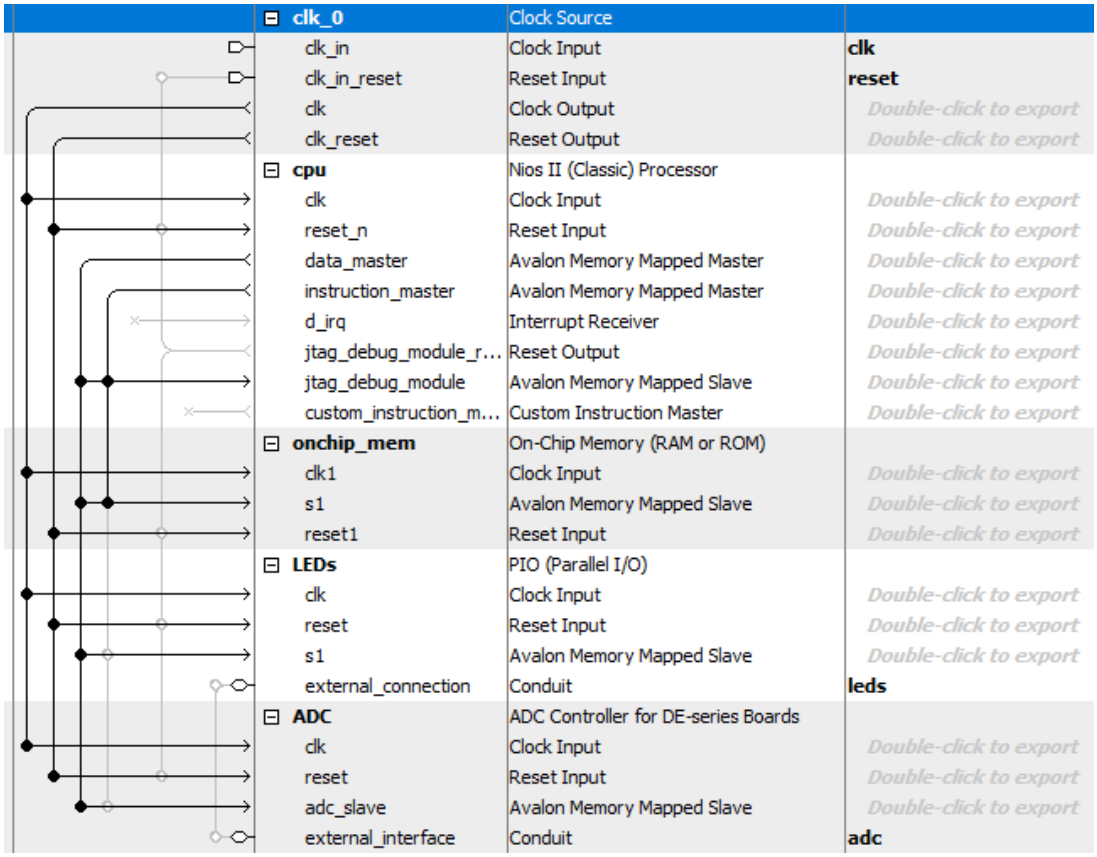


Figura 6.16 Creación del sistema en el Platform Designer (antes Qsys).

A partir de aquí se genera de forma automática el sistema, pero es necesario crear un módulo de alto nivel en un lenguaje de descripción de hardware (HDL). Se ha añadido el siguiente código en Verilog:

```
module adc_demo (CLOCK_50, KEY, LED, ADC_SCLK,
        ADC_CS_N, ADC_SDAT, ADC_SADDR);

        input CLOCK_50;
        input [0:0] KEY;
        output [7:0] LED;
        input ADC_SDAT;
        output ADC_SCLK, ADC_CS_N, ADC_SADDR;

        nios_system NIOS (
            .clk_clk (CLOCK_50),
            .reset_reset_n (KEY[0]),
            .leds_export (LED),
            .adc_sclk (ADC_SCLK),
            .adc_cs_n (ADC_CS_N),
            .adc_dout (ADC_SDAT),
            .adc_din (ADC_SADDR)
        );
endmodule
```

El módulo sirve coloquialmente para "unir los cables" entre los periféricos y las definiciones de Qsys. Aparte, desde Quartus también se definen los pines a los que van conectados en la FPGA.

Por otra parte, hay que crear un programa en C que permita usar el ADC. Cada periférico hay que declararlo como "volatile int *", es un puntero hacia la dirección base del componente, tal y como se definió en Qsys.

```
#define ADC_ADDR 0x00005000     //Direccion base del ADC en Qsys
#define LED_ADDR 0x00005020     //Dirección base de los leds en Qsys

int main (void){
    volatile int* adc = (int*)(ADC_ADDR);
    volatile int* led = (int*)(LED_ADDR);
    unsigned int data;
    int count;
    int channel;
    data = 0;
    count = 0;
    channel = 0;

    while(1){
        *(adc) = 0;              //Comienza la lectura del ADC
        count += 1;
        data = *(adc+channel);   //Toma el valor del canal seleccionado
        data = data/16;          //Ignora los últimos 4 bits (LSB)
        *(led) = data;           //Muestra el valor en los leds.
        if(count==500000){
            count = 0;
            channel = !channel;
        }
    }
    return 0;
}
```

Para programar todo el sistema en la FPGA hay que usar el Intel Monitor Program. Es un software que permite cargar el diseño en la placa y tiene funciones de depuración.

6.3.5.4. Algunas conclusiones

En este apartado dedicado al ADC interno se ha visto como existen diferentes formas de acceder a un mismo periférico. No hay una forma mejor que otra, todo depende de la aplicación que se desee desarrollar o del tiempo que se quiera invertir en el diseño.

6.4. Pulse-Width Modulation (PWM)

La modulación en ancho de pulso o PWM es una técnica en la que se modifica el *duty-cycle* o ciclo de trabajo de una señal periódica. Su funcionalidad puede ser la de transmitir información a través de un canal de comunicación o controlar la cantidad de energía que se envía a una carga.

El PWM tiene muchas aplicaciones, como pudieran ser el control de fuentes conmutadas o el control de motores. Su principal ventaja es que la pérdida de energía es muy pequeña, en el caso de los motores, regularlo por PWM en lugar de por resistencias es más eficiente, genera menos calor y se conserva el par motor.

Otra de sus ventajas es que la modulación en ancho de pulso funciona bien en los controles digitales, ya que, de por sí, tienen en su propia naturaleza el concepto de encendido/apagado (1/0), y pueden ajustar el ciclo de trabajo necesario de un modo muy sencillo.

Para la implementación de este PWM en concreto, se ha utilizado un IP core de código abierto y libre uso. Un IP core o "Intellectual Property core" es un módulo ya diseñado por un tercero. Generalmente suele haber IP cores de aquellos módulos que son muy usados o de los que hay demanda. La ventaja del IP core es la reusabilidad, no hay nadie que quiera reinventar la rueda, así que, si el módulo deseado está disponible, se puede utilizar y ahorrar tiempo.

Ese ahorro de tiempo, en ocasiones se puede traducir a dinero, y es que no todos los IP-Cores son gratuitos ni su código fuente está disponible para el usuario final, por lo que pudiera darse el caso de que hubiera que comprarlo. Otra alternativa es el proyecto OpenCores, donde se pueden encontrar IP Cores de libre uso y gratuitos, aportados por la comunidad.

En los dos últimos apartados se ha descrito el funcionamiento de los pines GPIO y del ADC. Mezclando ambos se puede hacer una señal PWM que pueda alterar su ciclo de trabajo a través de una resistencia variable, y es lo que se ha hecho. En la siguiente figura se muestra el esquema del módulo.

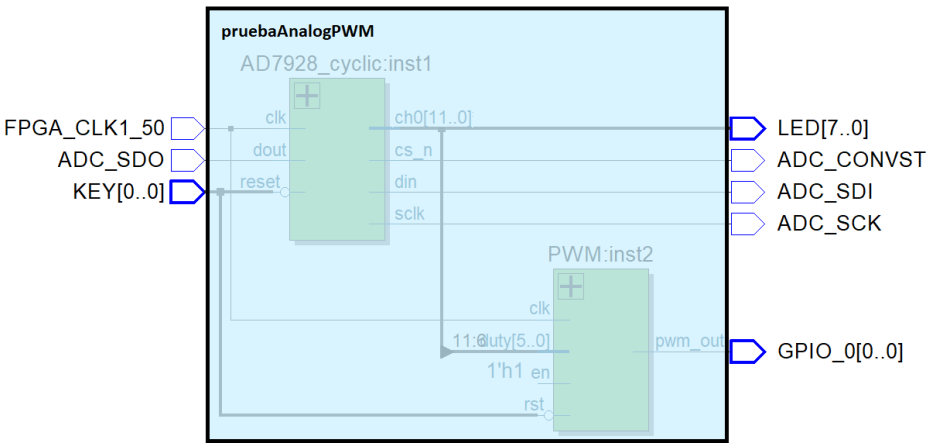


Figura 6.17 Módulo "pruebaAnalogPWM".

A través de este módulo ya se empieza a ver cómo en VHDL se pueden crear bloques de forma independiente y luego instanciarlos bajo otra entidad. En este caso se ha combinado el PWM con el ADC. La unión de ambos módulos forma un nuevo sistema, que en este caso permite una interacción física por parte del usuario para cambiar uno de los parámetros de la señal. En la siguiente imagen se muestra el montaje y el resultado en la pantalla:

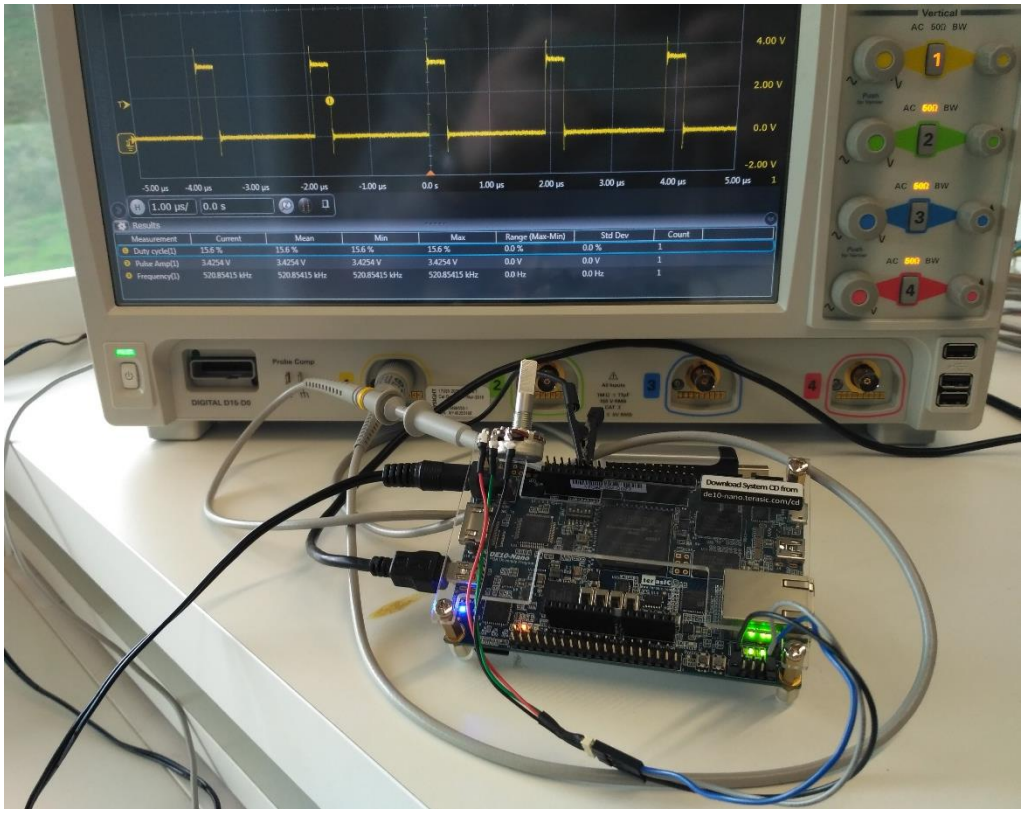


Figura 6.18 Señal PWM controlada por potenciómetro. *Duty-Cycle*: 15%.

Se comprobó, con ayuda del osciloscopio, que el *duty-cycle* de la señal generada se correspondiera con los valores introducidos en el potenciómetro. La misma prueba se hizo asignando valores fijos programados de forma manual.

6.5. Protocolos de comunicación

En informática y telecomunicaciones, un protocolo de comunicación es un sistema de reglas que permiten que dos o más dispositivos se comuniquen entre ellos para transmitir información por medio de cualquier tipo de variación de una magnitud física. Se trata de las reglas o el estándar que define la sintaxis, semántica y sincronización de la comunicación, así como también los posibles métodos de recuperación de errores. Los protocolos pueden ser implementados por hardware, por software, o por una combinación de ambos.

Los sistemas de comunicación utilizan formatos bien definidos (protocolos) para intercambiar mensajes. Cada mensaje tiene un significado exacto destinado a obtener una respuesta de un rango de posibles respuestas predeterminadas para esa situación en particular. Los protocolos de comunicación tienen que estar acordados por las partes involucradas. Para llegar a dicho acuerdo, un protocolo puede ser desarrollado dentro de estándar técnico.

Algunos protocolos para comunicar dispositivos siguen estando a la orden del día, pese a ser ya bastante antiguos. Dos de los más usados son SPI e I2C.

6.5.1. Especificación del protocolo I2C

El I2C es un bus serie para la transferencia de datos. Se usa para comunicar dos partes de un circuito, como podría ser un controlador con otros circuitos periféricos. Utiliza sólo tres cables, uno por donde el maestro envía la señal de reloj (SCL), otro bidireccional, por donde se transfieren los datos (SDA) y por último una tierra común.

Para que la comunicación funcione tiene que haber un protocolo, ya que no se puede emitir por el mismo cable a la vez, mientras un dispositivo está enviando información, el otro ha de estar escuchando, y viceversa.

6.5.1.1. Concepto teórico

El I2C funciona como un circuito de colector abierto, esto significa que el chip puede poner su salida a baja o dejar la línea abierta (alta impedancia), es decir, puede poner la línea a tierra, pero no puede ponerla a alta. Para que las líneas de SCL y SDA puedan tener un "1 lógico" será necesario poner resistencias de "*pull-up*". En caso de trabajar con más de un dispositivo, sólo hará falta poner una resistencia para cada línea (SCL y SDA), no por cada dispositivo [46].

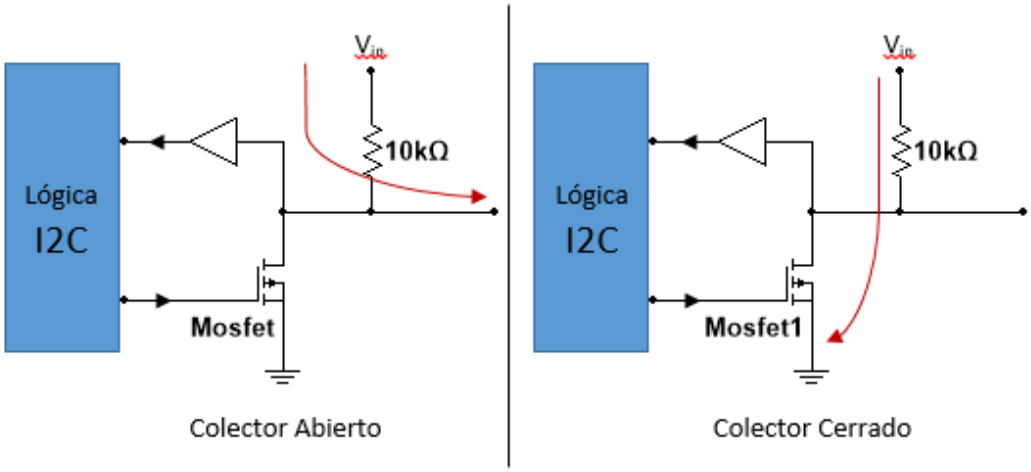


Figura 6.19 Esquema funcionamiento del I2C mediante resistencias de *pull-up*.

El esquema se puede ver en la imagen superior. El transistor Mosfet funciona como un interruptor. Cuando está abierto, deja pasar V_{in}, poniendo en el cable esa tensión, y cuando está cerrado, conduce la línea a tierra, lo que se traduce en una tensión de 0V. De esta forma es como se generan los "1" y "0" en este bus.

El estándar I2C permite tener múltiples dispositivos conectados al mismo bus, pero para que se entiendan es necesario un protocolo de comunicación. Todos los dispositivos estarán escuchando hasta que detecten un cambio particular en las líneas que indique el comienzo de transferencia. A continuación, el maestro enviará una dirección para referirse a uno de los dispositivos y comenzar la comunicación con él.

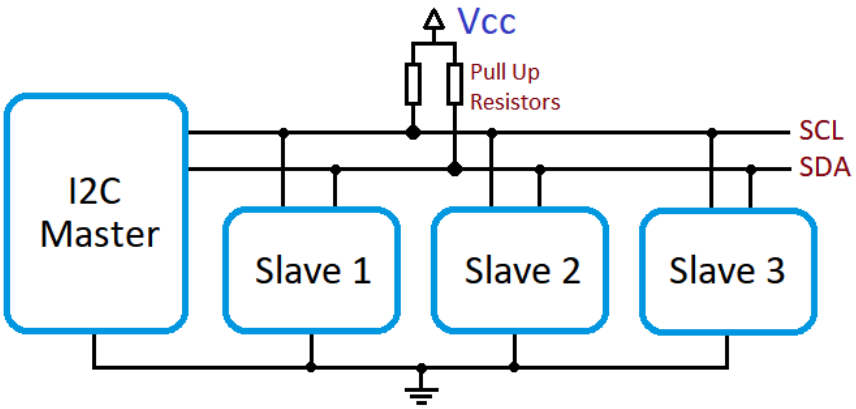


Figura 6.20 Esquema de conexionado del bus I2C con múltiples dispositivos.

6.5.1.2. Protocolo

El protocolo permite dos operaciones distintas, la escritura y la lectura, se comenzará con la explicación de la primera de ellas.

En un inicio, cuando el sistema está inactivo, las dos líneas están en alta impedancia ("Z"), por lo que al tener conectadas las resistencias "*pull-up*", las dos líneas están en "alta". Para indicar el comienzo de una operación (condición de inicio), la línea de datos tiene que ponerse a '0' antes que la del reloj. El reloj empezará a funcionar marcando el ritmo de la transmisión.

Para terminar la transmisión (condición de parada), hay que poner el canal SCL a '1' antes que el SDA. Seguirán ambos a '1' hasta que se vuelva a hacer la condición de inicio.

En el protocolo hay que tener en cuenta que, cada 8 bits, se manda un ACK. Lo primero que se hace es enviar la dirección del dispositivo (7bits) seguido de un bit de lectura/escritura. El '0' indica que se desea escribir y el '1' que se desea leer.

Para la escritura, después del ACK se envía la dirección del registro donde se desea escribir (1Byte), y se espera al ACK por parte del esclavo. El ACK consiste en poner la línea a '0' si todo va bien, y a '1' en caso contrario. Por último, se envía el dato que se desea escribir (1Byte), se espera al ACK y se termina con la condición de parada. En caso de que se quiera escribir en dos o más registros consecutivos, se enviaría el siguiente dato después del ACK y se terminaría con la condición de parada.

El caso de la lectura es un poco diferente, pero sigue siendo sencillo. Se empieza la trama como si fuera una lectura, con su condición de inicio, la dirección del dispositivo seguido del bit de escritura y después del ACK se envía la dirección del registro. Se espera al ACK. Aquí viene lo diferente. Después del ACK se vuelve a repetir la condición de inicio, y se envía de nuevo la dirección del dispositivo seguido del bit de lectura, el esclavo escribirá el bit de ACK y el maestro se quedará escuchando. El esclavo está enviando el contenido del registro que se le ha solicitado con anterioridad, el maestro debe de leerlo y almacenarlo, ya que le están respondiendo. Si se desea seguir leyendo registros consecutivos se le envía al esclavo un ACK a '0', en caso de querer finalizar la comunicación se le manda un NACK y se efectúa la condición de parada.

6.5.2. Implementación del protocolo I2C

6.5.2.1. Implementación

Una vez estudiado el funcionamiento del protocolo se ha implementado en VHDL. Cuando se programa para otros dispositivos, como pueden ser microcontroladores, existen librerías predefinidas para facilitar la comunicación con los dispositivos y ahorrar trabajo. Sin embargo, en este caso se trata de bajar de nivel de abstracción y escribir en hardware todo el protocolo de comunicación. Para ello hay que tener en cuenta el funcionamiento del protocolo, la generación de las señales, la importancia de la sincronía, el envío y recepción de datos y la interconexión de los circuitos.

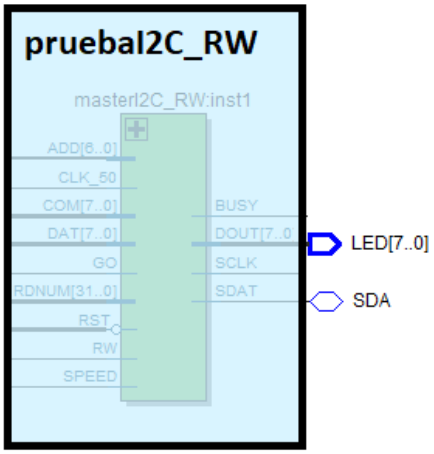


Figura 6.21 Módulo "pruebaI2C_RW".

El proceso de desarrollo del protocolo I2C ha sido lento y laborioso. Ha habido que hacer un estudio del protocolo y leer detenidamente las indicaciones del fabricante para entender el funcionamiento del dispositivo. Se ha desarrollado un módulo de I2C genérico, al que se le ha llamado "masterI2C_RW". Para la prueba actual, se ha instanciado este módulo sobre un envoltorio o "*wrapper*". A través de una entidad superior se puede programar el dispositivo para enviarle las tramas de adquisición y generación de señales.

6.5.2.2. Montaje

Para probar el protocolo sobre un hardware real se ha usado una placa de evaluación que funciona como expansor de GPIO. Se ha tenido que hacer un estudio previo, conocer sus circuitos internos, mover algunos jumpers para configurarla… Tenía bastante más trabajo que el conectarla y esperar a que funcionara. Las pruebas que se han realizado han sido depuradas a través de ModelSim, y verificadas con la ayuda de un osciloscopio y un analizador lógico. De esta forma se ha podido capturar toda la trama y comprobar su correcto funcionamiento.

La placa Terasic DE10-Nano cuenta con un zócalo para Arduino, así que, por mantener la convención, se han aprovechado esos pines, ya que hay uno destinado a SCL y SDA. De todas formas, estos no dejan de ser pines de GPIO, y también se podría haber sacado la señal por cualquier pin GPIO de los que se han usado con anterioridad. Que tenga el nombre de SCL no quiere decir que genere una señal de reloj por sí solo, es el usuario el que tiene que configurarla.

Se ha tenido que hacer un estudio previo de la placa de evaluación y su controlador para saber cómo configurar los jumpers, conocer el *pin-out* y saber cómo hacer el intercambio de datos [47] [48]. Desde la placa se ha obtenido una tensión de 3.3V para alimentar el periférico y se han configurado los dos puertos digitales para las señales de SCL y SDA. La alimentación son los cables rojo y negro, el verde corresponde a la línea SCL y el azul es para la SDA. La placa de evaluación tiene por detrás replicadas algunas de sus líneas, por lo que, en vez de utilizar una placa de prototipado, se han montado las resistencias de "*pull-up*" en el mismo módulo. El valor utilizado es de 10kΩ [49].

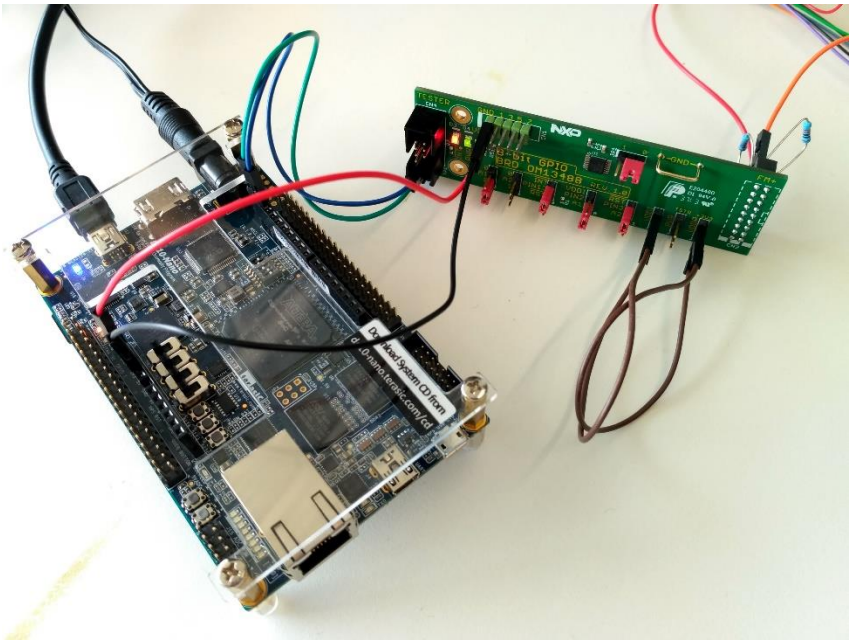


Figura 6.22 Conexión de la placa de evaluación a la DE10-Nano.

6.5.2.3. Comprobación

Tras hacer las pruebas, los datos obtenidos son satisfactorios. Se puede enviar una señal de GPIO a través del periférico y manipularla desde la FPGA. Si se conecta un generador de funciones y se configura como entrada, se puede leer esa línea del periférico a través de la FPGA. Y si se escribe en los registros de la tarjeta expansora, también pueden ser leídos en la FPGA y comprobar que la escritura se ha hecho correctamente. Estas son las capturas que se han obtenido:



Figura 6.23 Trama I2C de escritura y lectura generada por la FPGA.

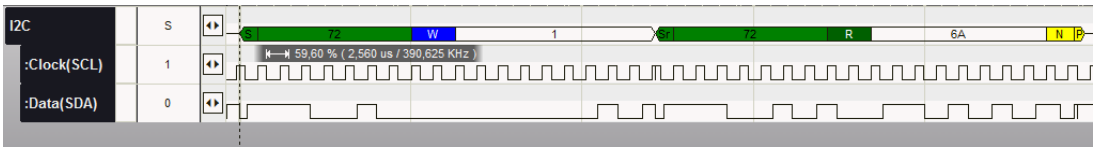


Figura 6.24 Operación de lectura.

Ambas figuras son complementarias. En la primera se ve la trama completa y la segunda es una ampliación de esta para hacer hincapié en la operación de lectura. En la primera trama de escritura se entra al registro de configuración y se escribe en ese registro de memoria qué pines son de lectura y cuáles de escritura. En la segunda trama se accede al registro de escritura, y se graba un valor. Esto define qué pines están sacando un '1' (voltaje) y cuáles no. En el caso de que se le asigne un '1' a un pin que es de lectura, no lo tendrá en cuenta, seguirá siendo de lectura. En la última trama se hace una operación de lectura, y como dicta el protocolo, tiene un "Start Repeated". Se accede a uno de los registros para leer su dato, y se observa que el dato que la placa de evaluación devuelve se corresponde con el dato que había

sido previamente enviado. Por lo tanto, se confirma que la comunicación es buena y que el protocolo está funcionando de forma correcta. Otra opción hubiera sido introducir un valor a través de un generador de funciones y leerlo, pero el montaje de pruebas se complica más.

6.5.3. Especificación del protocolo SPI

El bus SPI (*Serial Peripheral Interface*) es un estándar de comunicaciones, se usa para transferir información entre circuitos integrados en equipos electrónicos [50]. Consta como mínimo de 4 líneas, que son:

- SCL: Serial Clock (Generado por el maestro)
- MOSI: Master Out Slave In (Datos enviados por el maestro)
- MISO: Master In Slave Out (Datos enviados por el esclavo)
- $\overline{SS}$: Slave Select (Generado por el maestro, generalmente se usa lógica negada)

Tres de esas cuatro señales son compartidas por todos los dispositivos en el bus, pero todos tienen si propia línea de SS. El maestro pondrá la línea SS del dispositivo a '0' para indicarle que comienza la comunicación. A diferencia del I2C, este bus no tiene control de errores ni de flujo. Se transmiten los datos de forma continua hasta que se indica el fin de la comunicación.

SPI es un protocolo de comunicación muy simple. Carece de protocolo de alto nivel, por lo que no tiene esa sobrecarga de elementos adicionales, como pudieran ser los bits de control. Además, funciona a altas frecuencias de reloj y, al ser Full-Dúplex, el intercambio de datos puede ser más rápido en algunos dispositivos. Por el contrario, a medida que aumenta el número de dispositivos la complejidad se incrementa también, ya que se necesitan 3 líneas + N, siendo N el número de esclavos. Lo que se traduce en más cables.

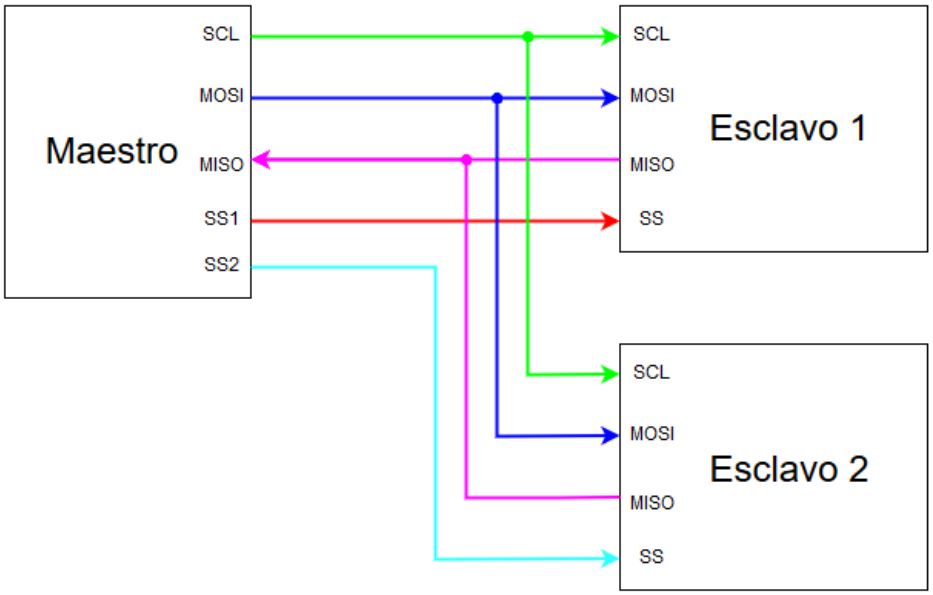


Figura 6.25 Esquema de conexionado del bus SPI con múltiples dispositivos.

El esquema de conexionado entre un maestro y varios dispositivos es el que se observa en la imagen superior. Ahí se ve de forma gráfica cómo comparten las tres líneas y que cada dispositivo tiene su propio SS. El maestro tiene que tener tantas líneas de "Slave Select" como esclavos tenga.

6.5.4. Implementación del protocolo SPI

6.5.4.1.Implementación

Al igual que antes, se ha utilizado el zócalo de Arduino para hacer el SPI, aunque podría haber valido cualquier otro pin de GPIO. Se ha realizado el diseño del módulo en VHDL y el esquema ha quedado de la siguiente forma. Se ha usado también un "wrapper" para instanciarlo.

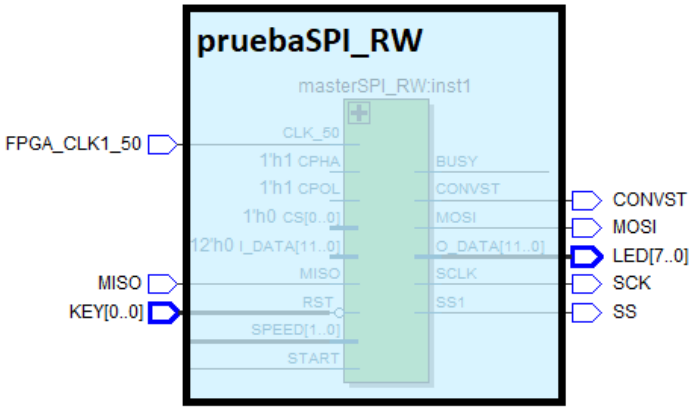


Figura 6.26 Módulo "pruebaSPI_RW".

6.5.4.2. Montaje

La configuración inicial de la placa tampoco fue sencilla, aunque los principales problemas fueron más por ser un hardware nuevo y por desconocimiento.

- La placa tiene unos "*Test Points*" o puntos de prueba. En principio, esas tomas permiten acceder a las señales con el propósito de analizarlas y depurar errores. En este caso, por esos puntos es por donde se va a introducir la señal de entrada y se va a obtener la salida del DAC.
- La alimentación se puede hacer a través de dos fuentes de alimentación o con una entrada de Jack DC de 9V. Al no tener transformador de 9V ni conector macho se conectó mediante los pines ambas fuentes de alimentación a 3.3V.
- La placa también tiene algunos jumpers. El primero que dio problemas fue el de la alimentación. El jumper LK6 sirve para seleccionar si la fuente va a ser alimentada a través del Jack o a través de las otras tomas. Al conectar una fuente de alimentación externa a las dos tomas de VDD y VDRIVE, ésta no mostraba amperaje. Al cambiar el jumper de posición ya funcionaba de forma correcta. La placa carece de leds que

indiquen si está siendo alimentada o funcionando, por lo que el problema no se aprecia a simple vista.

- Otro de los jumpers añade una resistencia a la entrada de la señal analógica. En un principio se obtuvo 5V de uno de los pines de la Terasic DE10-Nano, y se implementó un divisor de voltaje con una resistencia y un potenciómetro para que pudiera regularlo entre 0-2.5V, que es el rango de funcionamiento del ADC. Se observó que el valor que daba el ADC no se correspondía con el voltaje que se le estaba aplicando, y al medirlo con un polímetro se observó que el voltaje era más bajo que el que se le estaba asignando. Al final, buscando fallos o posibles causas se dio con el error. Hay un jumper que añade una resistencia en la entrada del ADC, al quitarlo se desactiva y funciona correctamente.

El montaje del ADC y la conexión con la placa DE10-Nano quedó del siguiente modo:

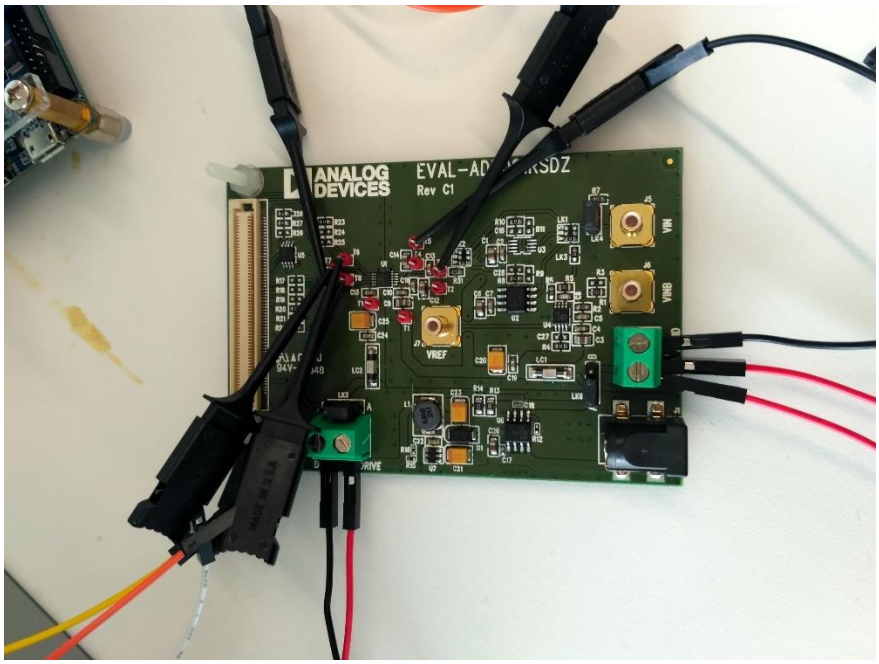


Figura 6.27 Tarjeta de evaluación con ADC. Ubicación de los "*test points*".

En esta primera imagen se observa el resultado de un trabajo previo. Se estudió la hoja de características [51] [52], se hizo un listado con los puntos de prueba y la señal a la que se corresponde cada uno, se descartaron aquellos que no eran necesarios para la aplicación y se hizo una guía sobre los colores y puntos, ya que al haber bastante cable es preferible tenerlo todo ordenado e identificado.

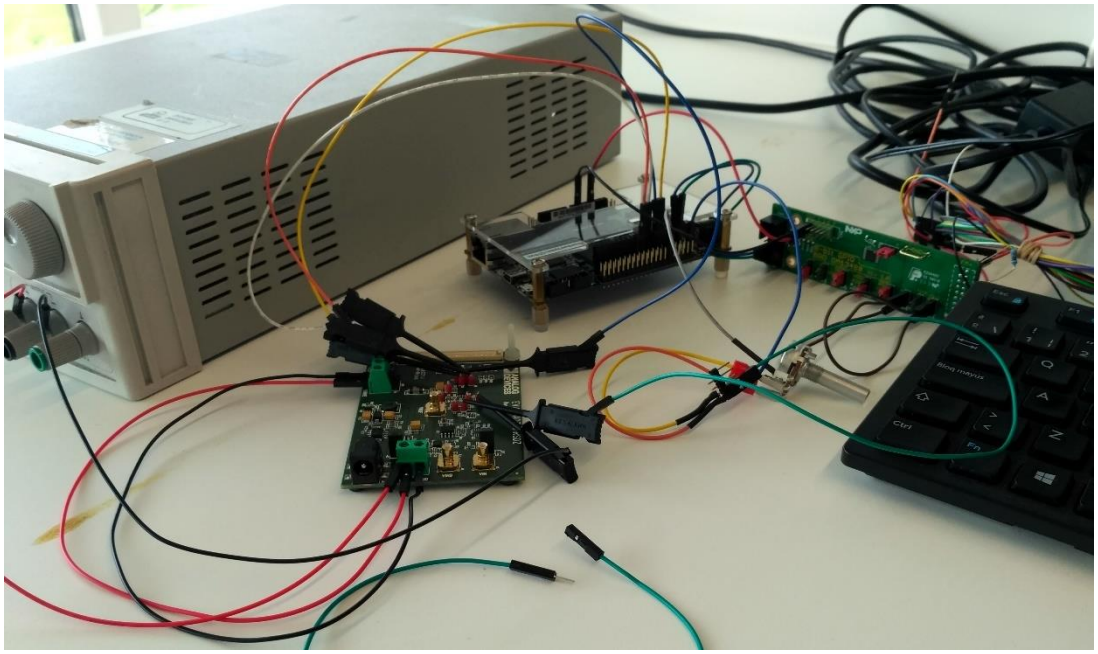


Figura 6.28 ADC conectado a la fuente de alimentación y a la placa de desarrollo.

En esta imagen se observa el conjunto ya montado. El ADC está alimentado y conectado a la FPGA. Está usando los 4 pines del protocolo (CLK, MOSI, MISO, CS) y otro para hacer el inicio de la conversión (CONVST). Tiene un potenciómetro conectado a 5V con un divisor de voltaje para obtener 2,5V, va a ser utilizado para enviarle un voltaje al ADC y comprobar que funciona. Realmente, en este caso, el MOSI no es necesario, ya que es un ADC y el maestro no tiene que escribir nada en él. Es el propio dispositivo el que se comunica cuando detecta un cambio en la señal CS.

6.5.4.3. Comprobación

Los resultados obtenidos han sido satisfactorios, y se han comprobado de dos formas. La primera ha sido visual, y es que se han representado los bits más significativos en los leds de la DE10-Nano. De esta forma, al variar el valor del potenciómetro, los leds cambian representando un valor binario. Por otra parte, se han obtenido capturas de la trama del protocolo a través del analizador lógico. Se está probando sobre un ADC, por lo que la comunicación es unidireccional, sólo se puede leer.

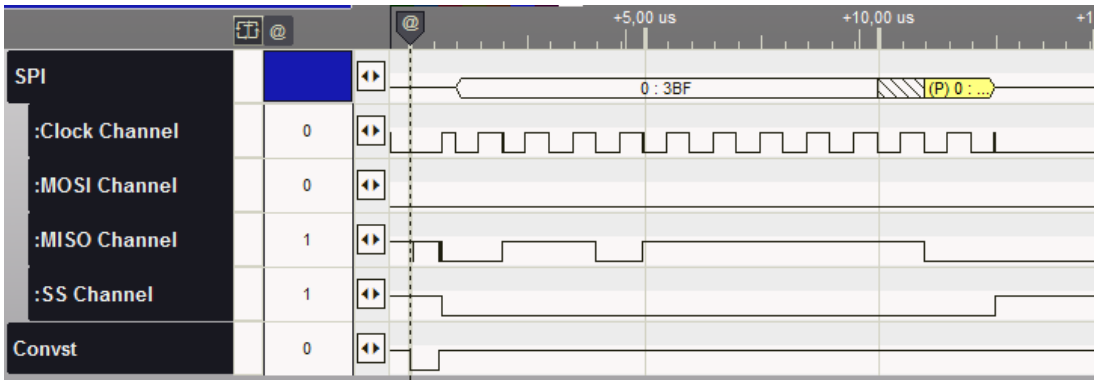


Figura 6.29 Trama obtenida a través del analizador lógico.

En este caso se está obteniendo un dato del ADC. Se baja la línea de CONVST para indicarle que inicie la conversión. Tiene que estar un tiempo mínimo a '0' antes de subir, y el CS se baja unos ns después. Comienza la transmisión y el maestro, es decir, la FPGA, tiene que saber dónde leer para obtener el dato correcto.

Con la ayuda de un polímetro se puede saber si el valor es correcto o no, y esto se ha comprobado. Se ha puesto el potenciómetro en los dos extremos y se han hecho medidas de voltaje y se ha analizado el valor que da el ADC. Posteriormente se ha puesto en el punto medio y se ha vuelto a medir. Los resultados obtenidos tienen una relación lineal, por lo que se dan los resultados por válidos.

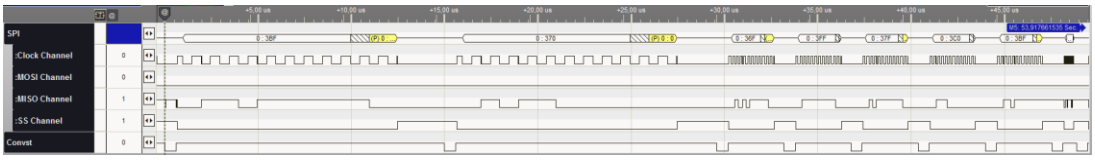


Figura 6.30 Trama con frecuencia variable obtenida a través del analizador lógico.

En esta segunda trama se ha hecho lo mismo, pero cambiando la frecuencia del reloj. Se trata de comprobar que el protocolo funciona también a velocidades más altas. El problema aquí está en los cables utilizados, que al haber distancia, pueden introducir errores.

6.6. Justificación de la etapa

Como se ha podido apreciar, la complejidad del trabajo realizado en la anterior etapa ha ido en aumento. Se ha pasado de encender unos diodos led o de cambiar el valor de uno de los GPIO a la generación de señales PWM o a la implementación de protocolos. Este trabajo ha tenido una justificación.

Por un lado, cuando se tiene una placa de desarrollo con la que no se ha trabajado antes, conviene hacer pruebas simples sobre su funcionamiento. Esas pruebas pueden dar lugar a nuevos diseños más complejos. Por otra parte, el adquirir datos mediante ADC, generar señales de PWM o comunicarse con periféricos a través de protocolos de comunicación, va a servir para el diseño final del controlador.

Por lo tanto, esta etapa es totalmente necesaria para el correcto desarrollo de todo lo que está por venir.

7

Implementación: SoC-FPGA

Este capítulo tratará sobre la implementación de soluciones SoC. Al igual que en el anterior, se sigue haciendo referencia a la implementación del proyecto, así que se recogerá la información relativa a esta parte. Se explicarán los pasos que se han dado y se estudiarán los resultados de los mismos.

7.1. Linux (de serie)

La placa de desarrollo permite la ejecución de un sistema operativo Linux. De por sí, el kit viene con una tarjeta de memoria micro-SD con una distribución de Angstrom Linux lista para funcionar, con los *drivers* correspondientes instalados. De todas formas, es posible trabajar con otras distribuciones disponibles.

Para trabajar sobre Linux se puede acceder mediante conexión serie por la UART, por SSH con un cable de red o por VNC. En este caso se ha accedido en primera instancia mediante conexión serie y se ha cambiado la dirección IP de la interfaz de red para poderlo conectar al mismo PC por SSH. Para ambas acciones se ha utilizado el software "Putty".

Trabajar con SSH aporta más ventajas. Desde el punto de vista de la conexión, el envío de datos es cifrado, aunque para el entorno de trabajo actual, no es que resulte de especial interés. Se ha elegido SSH porque también permite utilizar el protocolo FTP para la transferencia de ficheros. Programas como Notepad++ tienen *plug-ins* de FTP, lo que permite abrir ficheros del dispositivo, manipularlos desde un entorno gráfico y guardarlos. En el caso de trabajar con una conexión serie y escribir código, habría que hacerlo íntegramente desde la consola, lo que es menos eficiente, sobre todo cuando existe código que se desea copiar en lugar de redactar. Llegado el caso, SSH también podría servir como método para una transferencia de ficheros en caso de realizar una compilación cruzada.

7.1.1. Primeros pasos

Cuando se trabaja con un entorno nuevo, lo primero que se desea es conocer el entorno. Intel proporciona tutoriales para conocer el funcionamiento básico, y a partir de ahí, el usuario ya puede adentrarse en el desarrollo de sus propios diseños.

7.1.1.1. Hello World

Para mostrar la compilación nativa se escribirá y compilará el programa "Hello World". El código en C es el siguiente:

```
#include <stdio.h>

int main(void){
    printf("Hello World!\n");
    return 0;
}
```

Para compilar este programa se usará la línea de comandos de Linux, a la que se accede a través de SSH. Se usará GCC (*GNU C Compiler*), se trata de un compilador de código abierto ampliamente usado en entornos Linux y distribuido por la FSF (*Free Software Fundation*). El comando gcc invoca al compilador, y tiene dos argumentos, el primero de ellos se corresponde con el nombre del fichero que contiene el código fuente, y el parámetro "-o",

seguido de un nombre de fichero, define la salida del ejecutable. Una vez compilado el programa se puede ejecutar, los pasos a seguir se pueden observar en la siguiente imagen:

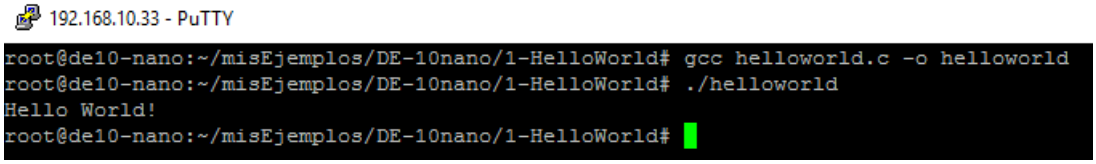


Figura 7.1 Compilación y ejecución de un programa.

7.1.1.2. Accediendo a dispositivos Hardware de la FPGA desde software en Linux

Los programas que se ejecutan en el procesador ARM pueden acceder a los periféricos que están implementados en la FPGA. El procesador puede acceder a la FPGA mediante el "HPS-to-FPGA bridge" o el "*Lightweight HPS-to-FPGA bridge*". Estos puentes son regiones mapeadas en el espacio de memoria del procesador ARM. Cuando un componente de la FPGA (como un IP core) está conectado a uno de los puentes, los registros del componente que estén mapeados en memoria estarán disponibles para la lectura y escritura por el ARM dentro de la región de memoria del puente.

Para probar el funcionamiento se ha probado un programa que accede a los leds de la placa de desarrollo. Cada vez que el programa se ejecuta incrementa el valor de los leds en '1'. Se podría definir como un contador binario de las ejecuciones del programa, que queda representado a través de los leds.

La placa Terasic DE10-Nano tiene 8 leds que van conectados a la FPGA a través de un puerto paralelo. Este puerto está también conectado al puente "lwhps2fpga", el cual está mapeado en el procesador ARM en la dirección 0xFF200000. Los puertos de entrada y salida están mapeados en la dirección de memoria del puente con diferentes "offsets", de modo que para acceder a un dispositivo concreto se hace a través de la dirección 0xFF200000 + offset. Los leds tienen un offset de 0x3000, por lo que su dirección es 0xFF200000 + 0x3000 = 0xFF203000.

Teniendo la dirección de memoria, si se le da un valor, como puede ser un *integer*, se representa en los leds. En este caso se incrementa en cada ejecución, pero si se cambia el código y se le asigna al puntero un valor estático, se representa el número de tipo *integer* en binario.

7.2. Diseño de un SoC

Diseñar un sistema SoC tiene su complejidad y su curva de aprendizaje. Se va a empezar diseñando un sistema sencillo para poder posteriormente llegar a sistemas más grandes y complicados. También cabe comentar que no hay demasiada información sobre esta tecnología y su implementación.

7.2.1. Consideraciones previas

Hay dos imágenes oficiales para HPS, una requiere todos los periféricos (versión de escritorio), mientras que la versión de consola, carente de entorno gráfico, no requiere ningún recurso de la FPGA. El sistema operativo que viene de serie en la tarjeta micro-SD es la versión con escritorio, y esto es importante. Si se desea realizar un diseño SoC particular, únicamente con aquellos periféricos que sean necesarios, y tratamos de cargar ese diseño en la versión de escritorio, el programa no va a funcionar. Por el contrario, en la versión de consola, se puede hacer que el sistema SoC arranque desde el encendido, de forma que se activen sólo aquellos componentes que pertenezcan al sistema diseñado.

En la misma línea, se recomienda grabar la imagen de la versión de consola en la tarjeta. Hay que recordar que también es aconsejable hacer una copia de seguridad antes. Eso ya es decisión del diseñador.

7.2.2. Preparación inicial

Se ha buscado la imagen de Angstrom apropiada, es decir, la que no tiene entorno gráfico. También es necesario disponer de una aplicación como PuTTY o Hyperterminal para comunicarse con la tarjeta a través de la línea de serie. Se recomienda PuTTY, ya que también permite otras comunicaciones, como SSH. Además, hará falta un software que permita grabar imágenes en tarjetas de memoria, Win32DiskImager o Rufus pueden servir para ese cometido, además de para hacer copias de seguridad.

7.2.2.1. Preparar la micro-SD con la imagen

Una vez descargada la imagen, se descomprime, ya que viene en un fichero zip. Se inserta la tarjeta micro-SD en el PC y se Win32DiskImager. Sólo hay que seleccionar la ruta de la imagen y la unidad de la tarjeta, es decir, el origen y el destino, y que realice la operación de escribir. En menos de 5 minutos estará la imagen grabada en la tarjeta y lista para funcionar en la placa. En el caso de querer hacer una copia de seguridad, se selecciona una ruta dónde se quiera almacenar y un nombre, y se hace la solicitud de leer. Creará un archivo con extensión img, y se podrá volcar de nuevo en la tarjeta cuando se desee.

Para pasos posteriores, se va a realizar una comunicación entre el PC (Host) y la placa a través de Ethernet, así que se va a dejar ya configurada. Se inserta la tarjeta micro-SD en su ranura y se conecta un cable mini-USB a la UART de la placa y al puerto USB del PC. Se pretende establecer una comunicación serie para acceder a la consola de comandos de Linux.

7.2.2.2. Levantar la red

Se efectuará la comunicación seria a través de PuTTY. Ante un sistema operativo Windows, se comprueba en el administrador de dispositivos el puerto de comunicaciones que está siendo usado por la placa. En este caso particular es el COM5, pero en otro PC, podría estar

usando otro diferente. En PuTTY hay que establecer conexión con este puerto de comunicación serie a una velocidad de 115200 baudios. En cuanto estén comunicados, y el sistema operativo cargado, aparecerá una terminal de Linux donde solicitará un inicio de sesión. Por defecto el usuario es root y no tiene contraseña, en caso de que pida contraseña, lo más seguro es que sea "terasic".

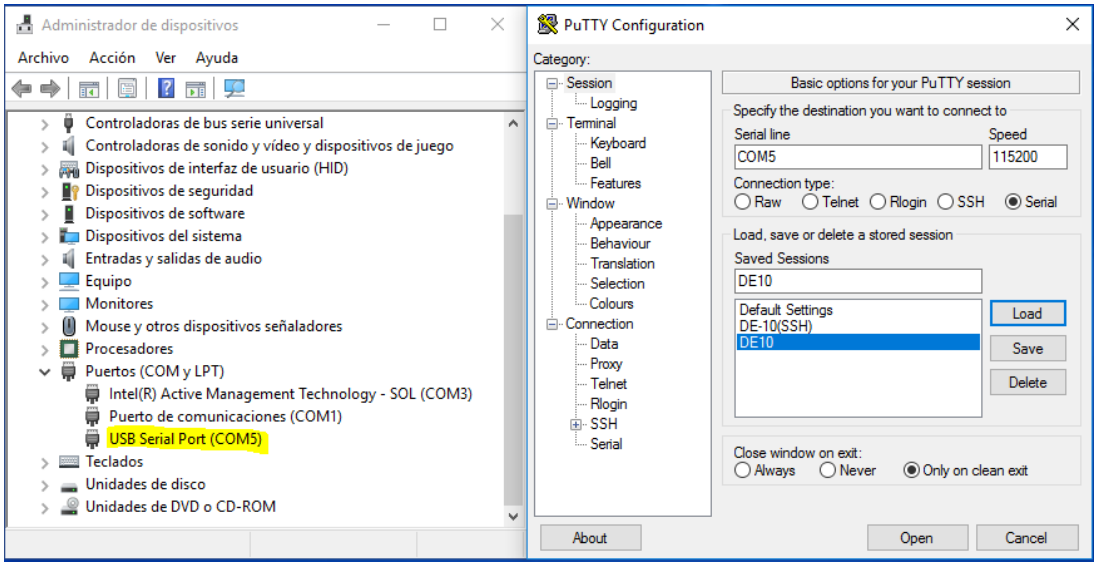


Figura 7.2 Comprobación del puerto y conexión en PuTTY.

Se conecta la placa de desarrollo al PC "host" usando un cable de red. En el PC tiene que haber disponible un puerto RJ45. Se le asigna una dirección IP del mismo rango que la tarjeta de red del PC. Para eso se usa el siguiente comando: `ifconfig [Interfaz] [IP]`. Un ejemplo particular podría ser el siguiente:

```
ifconfig eth0 192.168.10.33
```

Ya se puede acceder a la consola a través de SSH o acceder a los ficheros por FTP. Para realizar la conexión, también se podría conectar a un *switch* o *router* en caso de no disponer de un puerto en el PC.

7.2.3. Diseño y construcción del sistema

Se ha utilizado la aplicación de Terasic "System Builder". No es imprescindible, sólo tiene la ventaja de que genera un proyecto de Quartus para una placa y un código en Verilog con parte de las definiciones de forma automática, pero luego habrá que borrar partes innecesarias. Esta aplicación se puede encontrar en el CD de Terasic, pero también se puede descargar desde la web.

El código Verilog también ha sido traducido a VHDL, se pueden usar ambos. La razón para usar uno u otro puede tener factores geográficos, por ejemplo, en Europa el VHDL está bastante más extendido que el Verilog, mientras que en Norte América ocurre lo contrario.

Continuando con el System Builder, esta aplicación permite que el usuario seleccione qué componentes de la placa va a utilizar y se le crea un sistema más o menos a su medida. En

este caso se han elegido tres componentes para instanciar, que son el reloj del sistema, los leds y el HPS. En la imagen se puede ver la selección y arriba aparece la ruta donde se puede encontrar.

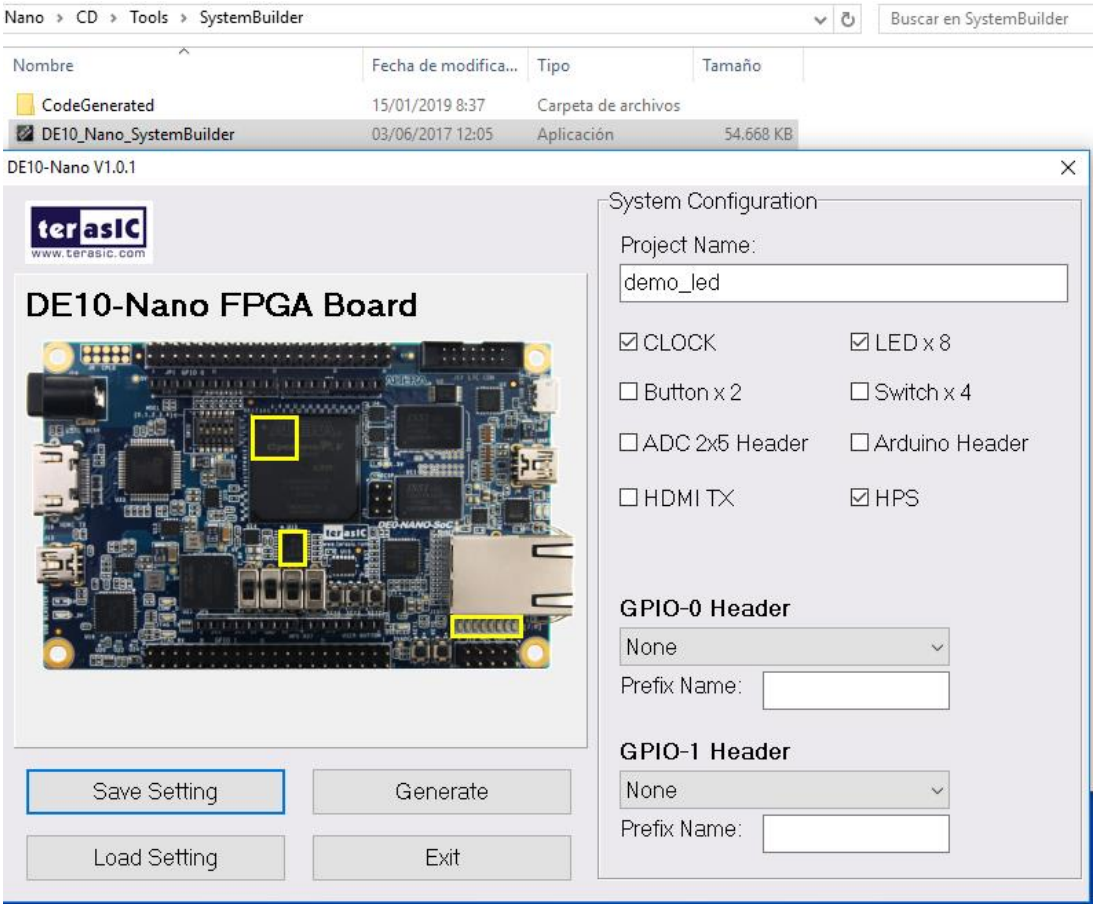


Figura 7.3 Selección de componentes a instanciar en System Builder.

La aplicación solicitará una ruta para guardar el proyecto generado. Para continuar diseñando el sistema, hay que abrir el proyecto con Quartus e ir al Platform Designer (anteriormente conocido como Qsys). El proyecto va a consistir en que el HPS acceda a los leds de la FPGA, y que decida cuales están activos o inactivos. En este caso se requiere de un reloj, el HPS y acceso a PIO, es decir, en este caso particular, los LEDs.

Para hacer la configuración del HPS se recomienda tener a mano el "Golden Hardware Reference Design" (GHRD). Lo proporciona Altera, y viene incluido en el CD de Terasic. El caso es que configurar el HPS requiere de muchos parámetros, y a cada chip hay que aplicarle sus respectivos valores de configuración. (Diferentes placas de desarrollo, pese a poder llevar un "*Cyclone V*", montan diferentes *pastillas*, y hay que aplicarles diferentes valores). Hay que conocer el hardware extremadamente bien para configurarlo adecuadamente, o se puede recurrir al diseño de Altera, que va a ser la solución más factible y rápida. El diseño del sistema quedaría como en la siguiente imagen:

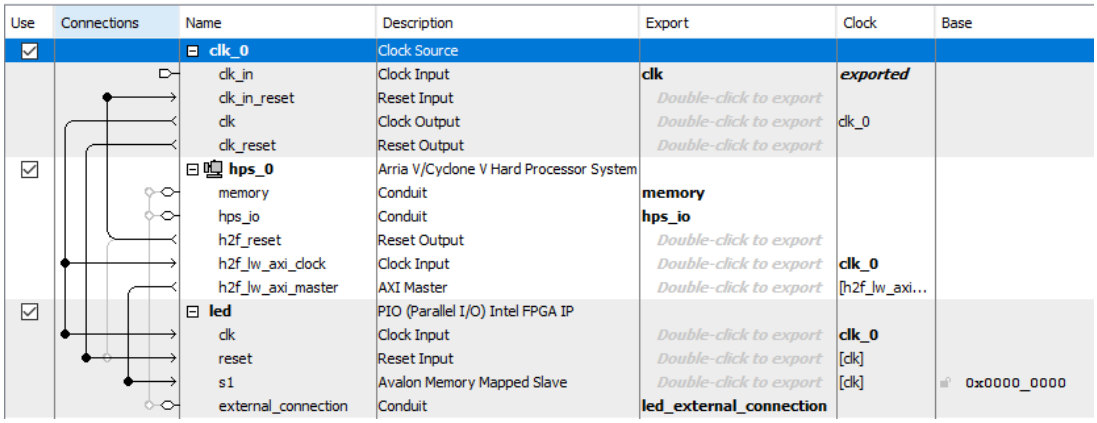


Figura 7.4 Diseño de la arquitectura del sistema en “Platform Designer”.

Como se ve en el esquema, se ha conectado el reloj del sistema a los dos dispositivos, y la señal de reset a los leds. El HPS tiene una señal de reset distinta. Por último, se comunica el HPS con los leds a través del bus “lightweight”. El HPS se ha configurado internamente con los valores del GHRD, y los como dispositivo de salida. Por otra parte, hay que exportar los leds y el HPS como conexión externa. Otro dato más, aquí sólo hay un dispositivo, por lo que no hay conflictos con las direcciones base, si existieran más de uno, probablemente habría que asignarlas manualmente para que fueran diferentes o solicitar al programa que las asigne de forma automática.

Con el diseño ya terminado en el Platform Designer, se puede guardar y exportar el código. Como la generación en el System Builder se ha hecho en Verilog, se seguirá usando ese lenguaje. Además, desde el programa también se puede obtener una plantilla, sólo hay que ir a Generate-> Show Instantiation Template. De este modo, se evita el escribir todo el código a mano. Al igual que antes, habrá líneas que no sean necesarias y se puedan borrar.

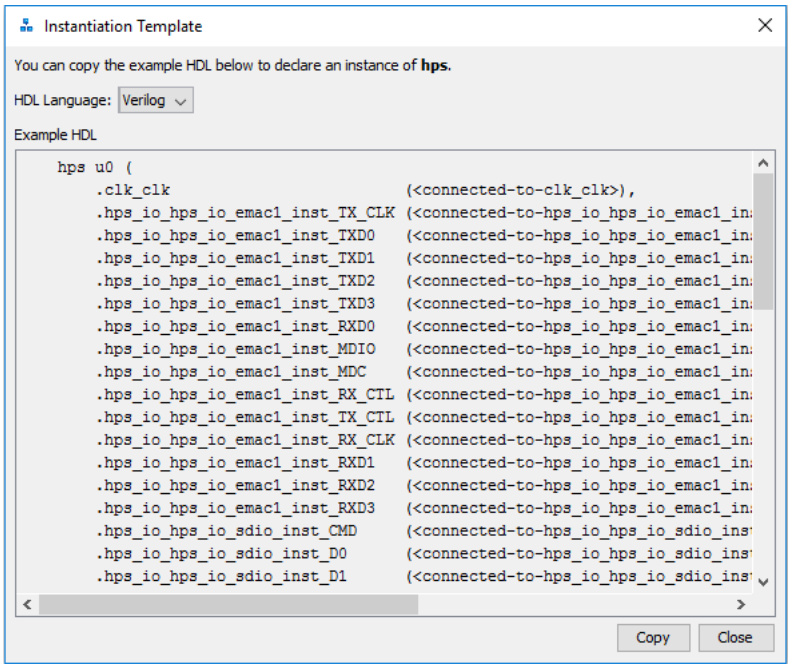


Figura 7.5 Plantilla de código para la instanciación. Incompleta, pero ahorra tiempo.

Ahora lo que hay que hacer es “conectar los cables”. En la figura anterior, a la izquierda están los “pines” del módulo, y a la derecha está incompleto y pone el texto de “(<connected-to-…>)”. Eso hay que sustituirlo, hay que poner los nombres que se han definido en la entidad principal, es decir, en el bloque que ha creado el System Builder. Se tienen que conectar ambos lados. Quedaría algo similar a esto:

```
35      output                  HPS_ENET_TX_EN,
36      output                  HPS_SD_CLK,
37      inout                   HPS_SD_CMD,
38      inout           [3:0]   HPS_SD_DATA,
39      input                   HPS_UART_RX,
40      output                  HPS_UART_TX,
41      input                   HPS_USB_CLKOUT,
42      inout           [7:0]   HPS_USB_DATA,
43      input                   HPS_USB_DIR,
44      input                   HPS_USB_NXT,
45      output                  HPS_USB_STP,
46      ////////// LED //////////
47      output          [7:0]   LED
48  );
49
50
51  //=======================================================
52  //  Structural coding
53  //=======================================================
54
55      hps u0 (
56          .clk_clk                        (FPGA_CLK1_50),     //                    clk.clk
57          .memory_mem_a                   (HPS_DDR3_ADDR),    //                 memory.mem_a
58          .memory_mem_ba                  (HPS_DDR3_BA),      //                       .mem_ba
59          .memory_mem_ck                  (HPS_DDR3_CK_P),    //                       .mem_ck
60          .memory_mem_ck_n                (HPS_DDR3_CK_N),    //                       .mem_ck_n
61          .memory_mem_cke                 (HPS_DDR3_CKE),     //                       .mem_cke
62          .memory_mem_cs_n                (HPS_DDR3_CS_N),    //                       .mem_cs_n
63          .memory_mem_ras_n               (HPS_DDR3_RAS_N),   //                       .mem_ras_n
64          .memory_mem_cas_n               (HPS_DDR3_CAS_N),   //                       .mem_cas_n
65          .memory_mem_we_n                (HPS_DDR3_WE_N),    //                       .mem_we_n
66          .memory_mem_reset_n             (HPS_DDR3_RESET_N),   //                       .mem_reset_n
```

Figura 7.6 Parte del código. Se usó la plantilla y se editó.

7.2.4. Compilación del sistema

Este apartado es necesario y no es nada a obviar, ya que tiene su complicación. El primer paso es comprobar la sintaxis, de todas formas, se adelanta que, aunque ésta sea correcta, fallará en la asignación de pines. Así que se en Quartus se inicia la compilación o se le da a “Start analysis and elaboration”, cualquiera de las dos opciones es válida.

Hay que usar scripts tcl, ya que ahorraran tiempo. Aquí hay que comentar que, si se ejecutan los scripts saltándose el paso anterior, dará error y fallarán. Tras este inciso y con el análisis y elaboración completados, se ejecutan los siguientes scripts, que servirán para que Quartus asigne de forma automática los pines que se corresponden con el HPS. Ello mismo tendrá en cuenta el dispositivo para el que está siendo programado, y sabrá asignarlos correctamente. Hay que ejecutar los dos que aparecen en la siguiente imagen:

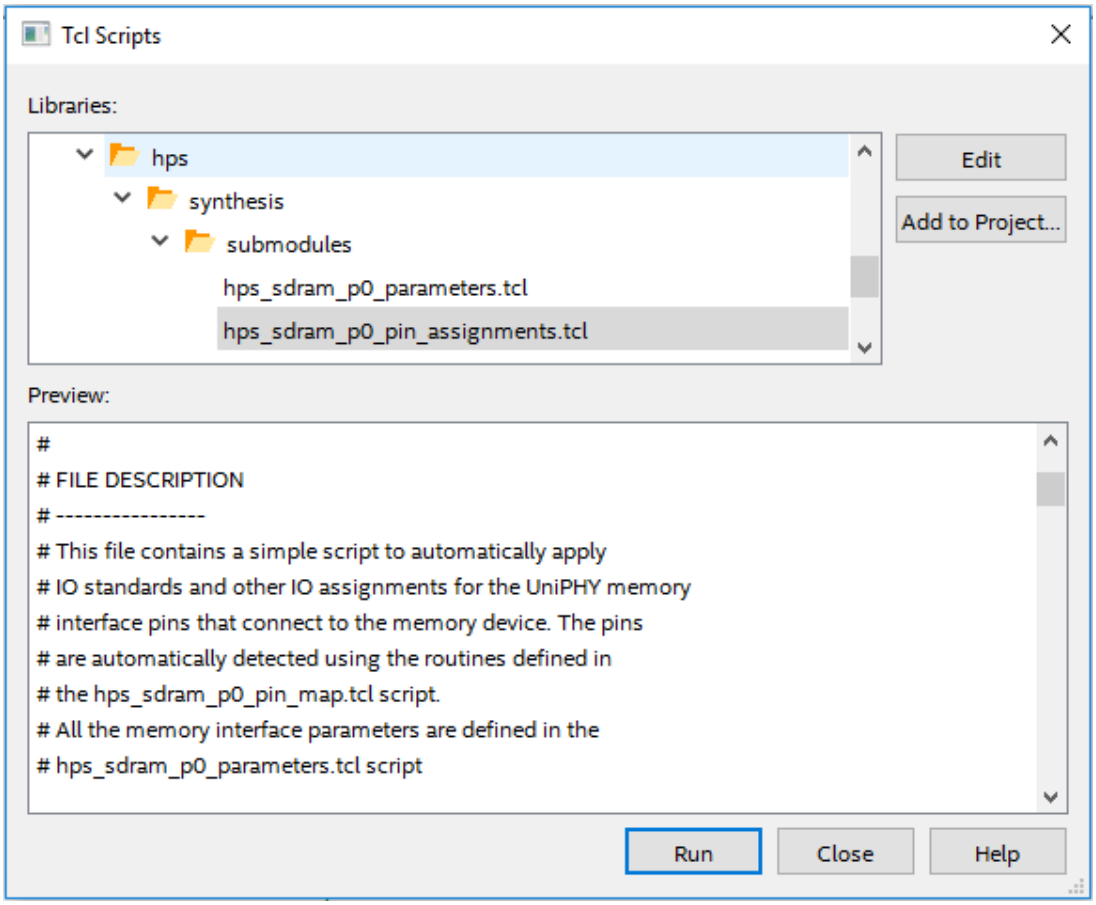


Figura 7.7 Los dos Scripts Tcl que hay que ejecutar.

Desde el "Pin Planner" se asignarán manualmente aquellos pines que no correspondan al HPS, porque esos no los ponen los scripts. Elementos como el reloj, los leds u otros que se hayan utilizado tendrán que ser correctamente referenciados. Los pines que no se usen, en el caso de haberlos, pueden ser eliminados. Ahora ya está todo listo para la compilación de verdad. Al finalizar se generará un fichero con extensión sof.

7.2.4.1. Creando el fichero de arranque (de .sof a .rbf)

Tras la compilación en Quartus, se ha dicho que se obtiene un fichero .sof. Éste contiene la información necesaria para programar e inicializar la FPGA desde Quartus. En este caso se desea inicializar la FPGA desde el arranque, es decir, desde un estado de apagado. Para eso es necesario convertir el archivo sof (SRAM Object File) en RBF (Raw Binary File).

En el CD se puede encontrar un script de Windows con el nombre de "sof_to_rbf.bat", hay que abrirlo con un editor de texto y modificarlo para que se correspondan los nombres de los ficheros generados, de modo que se pueda hacer la llamada de forma correcta. Se guardan los cambios y se ejecuta el script. Al finalizar aparecerá por pantalla si el proceso ha ido bien o si por el contrario ha habido algún error. El más común es que no se correspondan los nombres o que no se defina bien la ruta. Para que no haya problemas de ruta, se recomienda que el script y el fichero a convertir estén en el mismo directorio. Si todo ha ido bien, se habrá creado un nuevo archivo con extensión .rbf en la carpeta.

7.2.4.2. Configurando el arranque en la placa

Con la placa de desarrollo apagado se extrae la tarjeta para insertarla en el PC. Habrá una partición que pueda ser leída desde Windows, es de tipo FAT32 y ahí es donde se encuentra el "bootloader", o lo que es lo mismo, lo que lee la tarjeta para saber qué cargar en el arranque.

Se inserta la tarjeta de memoria en el PC y se copia el archivo rbf generado. Se observa que hay un archivo llamado "u-boot.src", éste es el que se encarga de decir qué es lo que se va a arrancar al inicio, así que hay que editarlo y decirle que se desea arrancar el rbf que se acaba de copiar, se hace con un editor de texto, sólo hay que cambiar el nombre del rbf que lee.

7.2.4.3. Comprobación

Se vuelve a insertar la tarjeta de memoria en la placa y se enciende. Con la terminal abierta y leyendo la línea serie, se puede ver qué es lo que está haciendo desde el arranque. En este caso, se ha obtenido la siguiente captura, que muestra cómo, efectivamente, está cargando el .rbf que se ha especificado:

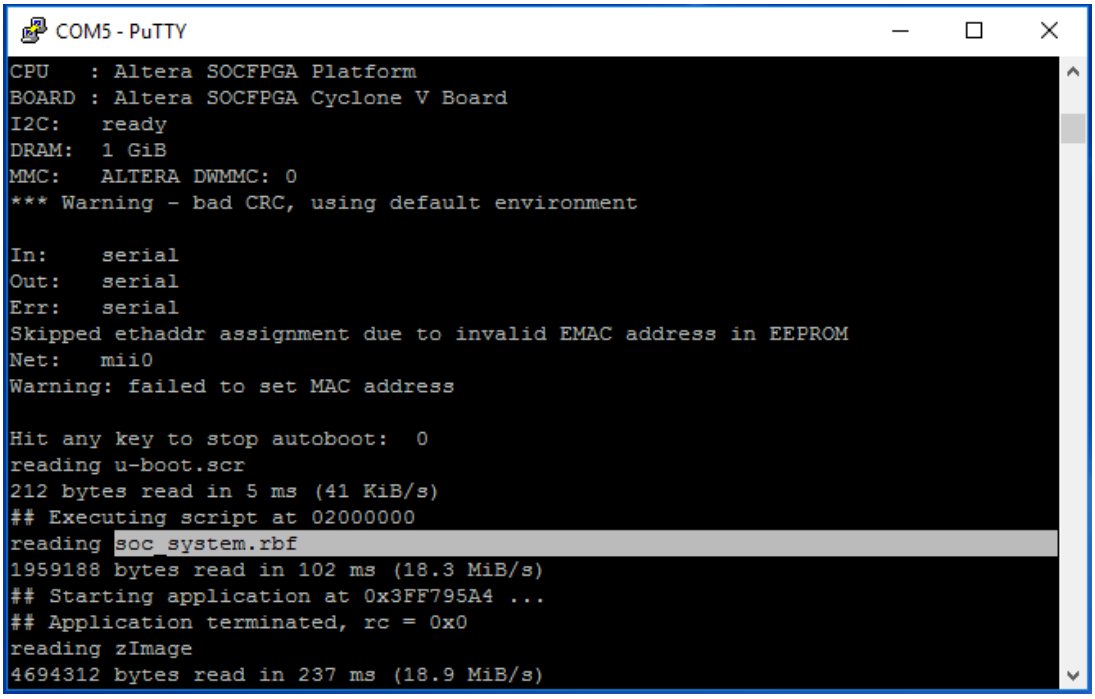


Figura 7.8 Bootloader cargando el sistema SoC diseñado.

7.2.5. Compilación de la aplicación en C

Con el sistema diseñado y funcionando en la placa, se desea hacer una aplicación en C que se ejecute sobre el HPS y que pueda acceder a elementos de la FPGA. Este ejemplo consiste en un programa que va a hacer un barrido de leds. En otras palabras, va a encender los leds

consecutivamente de derecha a izquierda, apagando el anterior. Hará un efecto de movimiento. El código del programa es el siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/mman.h>
#include "hps_0.h"

#define REG_BASE 0xff200000
#define REG_SPAN 0x00200000

void *base;
uint32_t *led;
int fd;

void handler(int signo){
    *led=0;
    munmap(base, REG_SPAN);
    close(fd);
    exit(0);
}

int main(){
    int i=0;
    fd=open("/dev/mem",(O_RDWR|O_SYNC)); //Abro la memoria del
sistema.
    if(fd<0){
        printf("Can't open memory. \n");
        return -1;
    }
    base=mmap(NULL,REG_SPAN,(PROT_READ|PROT_WRITE),MAP_SHARED,fd,REG_BASE);
    if(base==MAP_FAILED){
        printf("Can't MAP memory. \n");
        close(fd);
        return -1;
    }
    led=(uint32_t*)(base+LED_BASE);
    signal(SIGINT, handler);
    *led=0x1;
    while(1){
        usleep(250000);
        if(i<7) i++;
        else i=0;
        *led=1<<i;
    }
}
```

El programa lo que hace es, en primer lugar, acceder a la dirección de memoria y comprobar que haya algo. En segunda instancia, mapea la memoria y comprueba que la operación haya ido bien. Y por último, si existe ese dispositivo, puede escribir en su dirección de memoria y cambiará su valor, eso ya es el bucle principal del programa.

Se observa que hay una librería llamada “hps_0.h” y que hay una variable llamada “LED_BASE” que no parece estar declarada. El motivo es que a partir del sistema que se ha generado en el “Platform Designer”, se puede generar la cabecera, y ésta contendrá las

direcciones de todos los dispositivos que hayan sido instanciados. A continuación, se explica el procedimiento.

7.2.5.1. Generación automática de cabecera

Tras la generación del sistema en el "Platform Designer" sale como resultado un fichero con extensión "sopcinfo", que guarda en su interior toda la información sobre los componentes que forman el sistema y sus direcciones asignadas. El objetivo actual es traducir este fichero a otro con una extensión .h.

Hay que escribir el script "generate.sh". Al igual que otros archivos, también se puede encontrar un ejemplo de este en el CD de Terasic, pero habrá que abrirlo y editarlo para cambiar el nombre del archivo .sopcinfo del que se deseen obtener sus cabeceras.

Para ello se utiliza el programa "Intel FPGA Embedded Command Shell". Básicamente es un intérprete de comandos basado en Linux. Se siguen los siguientes pasos

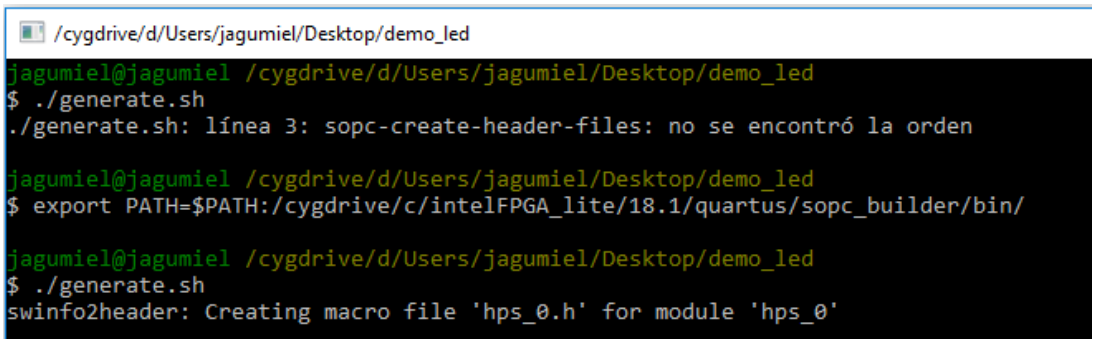


Figura 7.9 Generación de cabeceras.

Aquí se observa un error bastante habitual. Si no está la ruta para acceder al compilador en las variables de entorno de Windows ocurre lo que se ve en la tercera línea, que no se encuentra la llamada al compilador. Una alternativa para hacerlo sin modificar las variables de entorno es mediante el comando de "export PATH" que se ve en la quinta línea. Tras definir la ubicación del compilador, se puede generar la cabecera. Y se observa cómo en el directorio del proyecto ha aparecido el nuevo fichero.

7.2.5.2. Compilación del programa

Ya está por un lado el programa en C escrito, y por otra parte, están las cabeceras necesarias para la compilación, lo único que falta ahora es un fichero "Makefile", que indica cómo se compila el programa. Aquí surgieron problemas y se tuvo que analizar la situación. El fichero de Makefile ha cambiado, tras la adquisición de Altera por Intel, ahora hay que llamar a otras librerías, cuando antes no era necesario. Se hizo una pequeña investigación y se aprendió a hacerlo de forma correcta. Aquí está el código que se implementó para el Makefile:

```
#
TARGET = led

#
```

```
CROSS_COMPILE = arm-linux-gnueabihf-
ALT_DEVICE_FAMILY ?= soc_cv_av
#Nota: CFLAGS es línea única, pero no cabe en el ancho de la página.
CFLAGS = -g -Wall -I
${SOCEDS_DEST_ROOT}/ip/altera/hps/altera_hps/hwlib/include -
D$(ALT_DEVICE_FAMILY)
LDFLAGS = -g -Wall
CC = $(CROSS_COMPILE)gcc
ARCH= arm


build: $(TARGET)
$(TARGET): main.o
    $(CC) $(LDFLAGS)    $^ -o $@
%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -f $(TARGET) *.a *.o *~
```

Ya están los tres ficheros, es decir, el código, las cabeceras y el Makefile. Ahora sólo hay que asegurarse de que estén en el mismo directorio y ejecutar este último. Se hace a través de la consola de Intel:

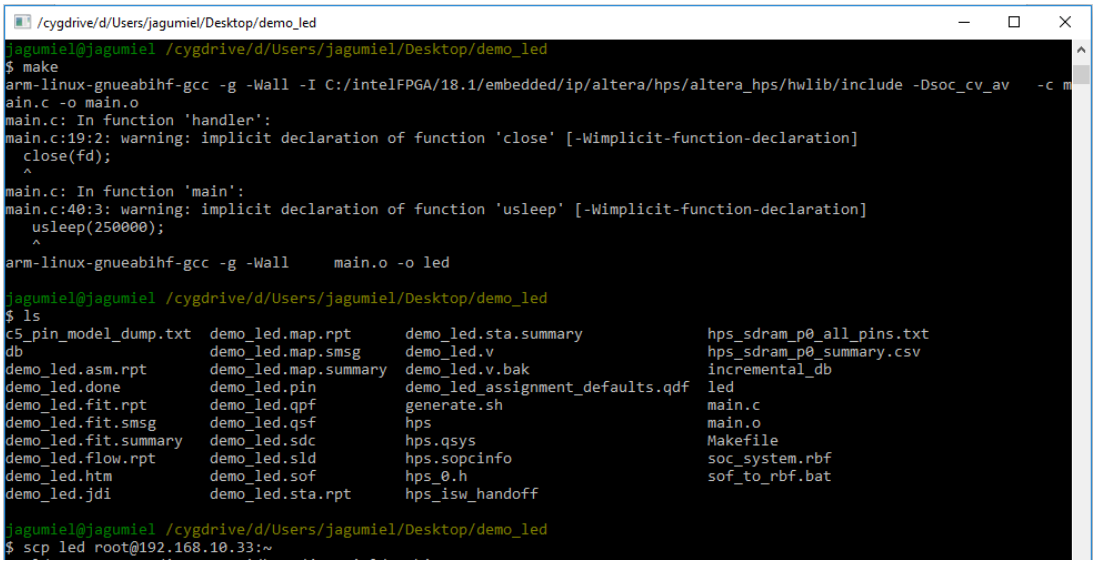


Figura 7.10 Compilación del programa en C y envío del fichero a la placa.

El programa ha compilado correctamente, salen algunas advertencias, pero nada de lo que preocuparse. Con el comando `scp [nombreFichero] [IP]:[Ubicación]` se puede enviar el fichero por SSH. Tiene que estar la placa encendida y conectada al PC por Ethernet. En el caso de pedir contraseña, ésta será "terasic" por defecto. Si no envía nada o devuelve error, se puede ir a la consola de la placa e introducir el comando de "`ssh localhost`", se inicia sesión y se vuelve a cerrar con el comando "`exit`". Luego se vuelve a probar a transferir el fichero.

7.2.6. Ejecución del programa

Desde la terminal de Linux de la placa, ya sea por línea serie o SSH, ejecutamos el programa. Lo más seguro es que haya que darle permisos de ejecución, se puede hacer con el comando "`chmod +x [nombreFichero]`". Después se puede ejecutar como cualquier archivo de Linux, "`./[nombreFichero]`". Se observará como el programa funciona y mueve los leds. Si se ejecuta el mismo programa sin cargar el sistema, no funcionará.

7.3. Hacia SoC más complejos

En el apartado anterior se ha mostrado cómo se construye un SoC básico. El sistema diseñado no tiene una utilidad real, salvo la de aprendizaje. Como se ha comentado con anterioridad, hay poca información respecto a cómo diseñar e implementar un SoC, y describir el trabajo anterior como si de una guía se tratase, puede aportar información relevante para aquellos que quieran aventurarse en tales lindes.

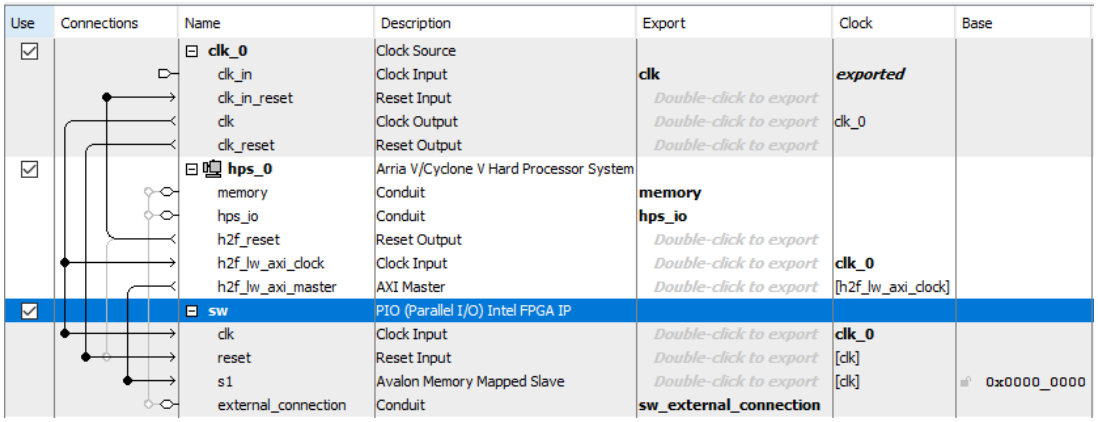
En este apartado se va a tratar de hacer sistemas que hagan uso de ambas partes, con mayor complejidad. Todo el trabajo anterior se puede mezclar. Además, se ha escrito esta vez en VHDL.

7.3.1. Siguiendo el ejemplo

A partir del tutorial anterior, y tal y como se hizo al principio del proyecto en la FPGA, es interesante hacer pequeños proyectos con los elementos básicos de la placa. En este apartado se pretende explicar tres casos similares, pero diferentes, y con dos elementos muy sencillos, los leds y los interruptores.

7.3.1.1. Interruptores controlados por el HPS

Como en el primer caso se accedió a los leds, la segunda prueba era hacer el paso contrario, es decir, antes el HPS modificaba los leds escribiendo en su registro de memoria, ahora en vez de escribir, se leerán los interruptores y se sacará su combinación binaria, traducida a decimal, por pantalla.



Use	Connections	Name	Description	Export	Clock	Base
☑		**clk_0**	Clock Source			
		clk_in	Clock Input	**clk**	***exported***	
		clk_in_reset	Reset Input	*Double-click to export*		
		clk	Clock Output	*Double-click to export*	clk_0	
		clk_reset	Reset Output	*Double-click to export*		
☑		**hps_0**	Arria V/Cyclone V Hard Processor System			
		memory	Conduit	**memory**		
		hps_io	Conduit	**hps_io**		
		h2f_reset	Reset Output	*Double-click to export*		
		h2f_lw_axi_clock	Clock Input	*Double-click to export*	**clk_0**	
		h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_axi_clock]	
☑		**sw**	PIO (Parallel I/O) Intel FPGA IP			
		clk	Clock Input	*Double-click to export*	**clk_0**	
		reset	Reset Input	*Double-click to export*	[clk]	
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0000
		external_connection	Conduit	**sw_external_connection**		

Figura 7.11 Creación del sistema en Platform Designer.

Este es el sistema. En vez de leds se ponen interruptores, que son también un PIO, pero configurado como entrada.

En cuanto a la programación, cambia el bloque principal y el nombre de alguna variable en la librería, ya que han sido renombradas.

```
while(1){
    value=*sw;
    if(value!=last_value)
        printf("Numero = %d\n", value);
    usleep(250000);
    last_value=value;
}
```

Se almacena el contenido del registro de memoria en la variable “value”. Se comprueba que el valor haya cambiado desde la última comprobación, y de ser cierto, se imprime el nuevo valor por pantalla. Al finalizar se vuelve a actualizar la variable “last_value” para seguir haciendo comprobaciones y saber si hemos cambiado los interruptores.

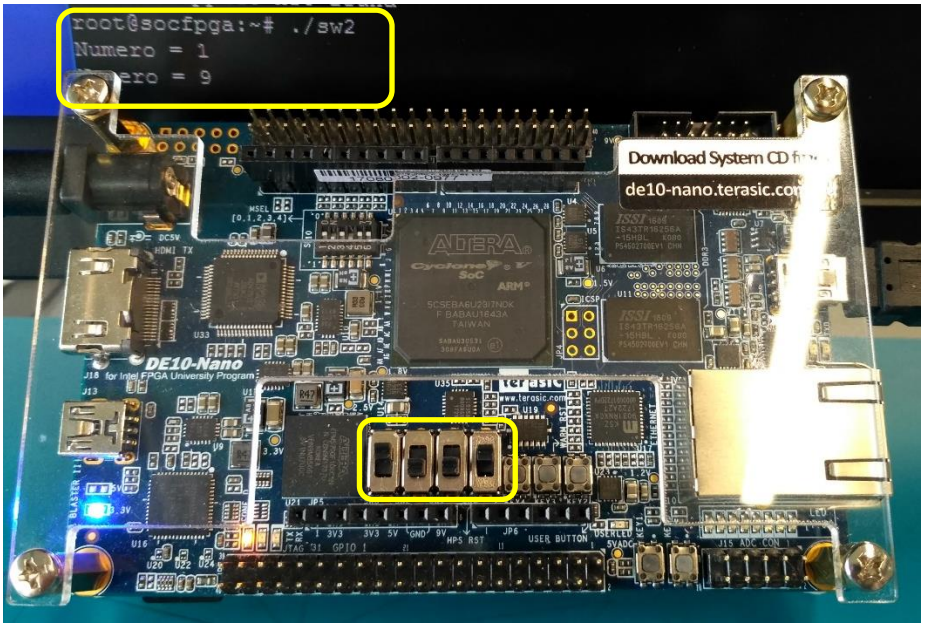


Figura 7.12 Programa en funcionamiento.

En un estado inicial los interruptores estaban en off, es decir, a '0', y el programa tiene inicializadas sus variables a 0. Al ejecutar el programa se ha desplazado el interruptor de más a la derecha hacia arriba, creando la secuencia binaria "0001". El programa lo lee, comprueba que es distinto al anterior y lo saca por pantalla, después actualiza el último valor.

La siguiente prueba ha sido desplazar el switch de la izquierda, y se crea la secuencia "1001", que en decimal es un 9. Como ha cambiado de valor, lo muestra por pantalla.

Se comprueba de este modo que el HPS tiene acceso al registro de memoria de los interruptores, y que puede leer su posición.

7.3.1.2. Interruptores y leds controlados por el HPS

En el anterior apartado se ha visto como el HPS puede leer el estado de un periférico que está conectado a la FPGA. En este caso se pretende mostrar cómo también es posible la lectura de un periférico y la escritura sobre otro, siendo ambos propios de la FPGA.

En este ejemplo se ha tomado como referencia los interruptores y los leds. Los interruptores se usarán como periférico de entrada, y los leds como periférico de salida. La combinación que se introduzca manualmente en los interruptores será leída por el HPS, y en caso de que se cambie el dato representado, se actualizará en los leds.

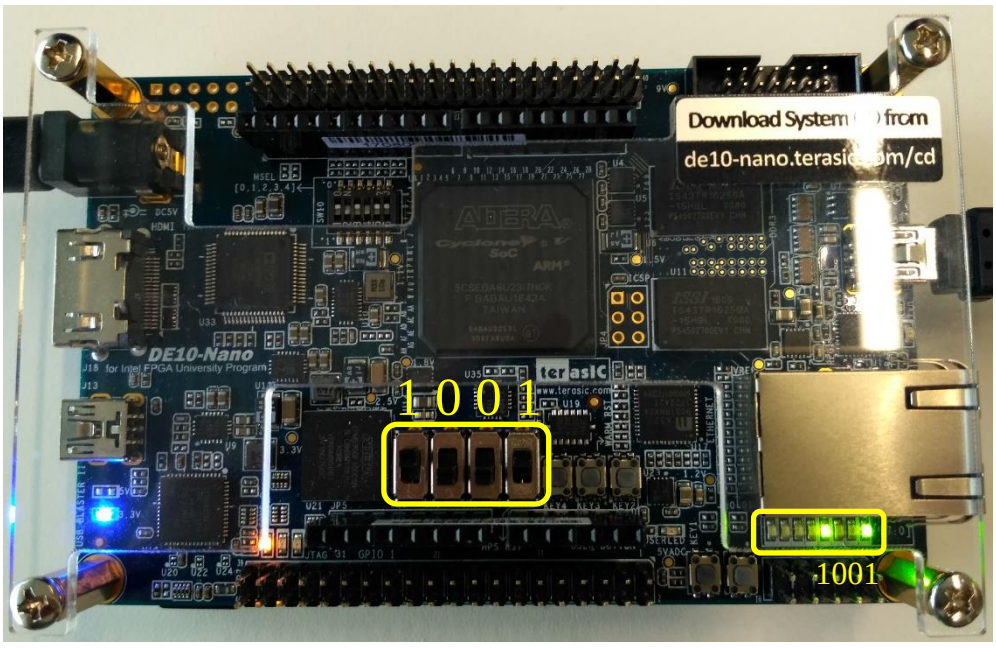


Figura 7.13 HPS leyendo y escribiendo datos.

7.3.1.3. FPGA y HPS funcionando conjuntamente

El siguiente paso ha sido comprobar que la FPGA puede funcionar de forma independiente al HPS. Lo que se ha hecho es combinar el código que se usó para probar el ADC por IP Core con el que se usó para leer los interruptores por HPS. De este modo está la FPGA recogiendo datos por el ADC y mostrándolo en los leds, y si se ejecuta una aplicación sobre Linux, se podrán ver los datos del ADC por pantalla.

En resumen, se ha creado un sistema nuevo, que en su base tiene a la FPGA funcionando y adquiriendo datos. A través de una aplicación en C se podrá acceder al puntero donde se escriben los datos del ADC, de forma que se tienen dos sistemas funcionando a la vez.

Una aplicación práctica podría ser tener a la FPGA recogiendo datos y crear una salida en paralelo con otros datos modificados por el procesador. Por ejemplo, se podría tener un canal con los datos en bruto y el procesador podría aplicar un filtro y sacar por otra salida una señal modificada.

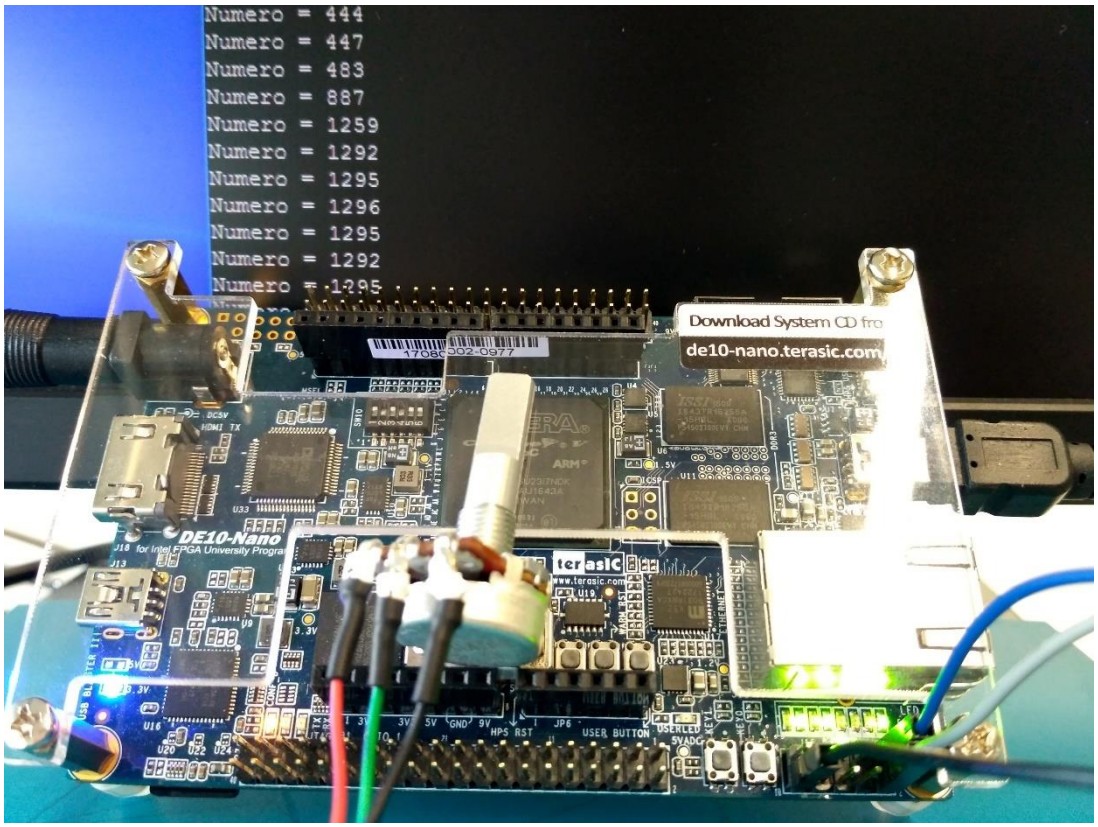


Figura 7.14 FPGA y HPS trabajando en conjunto.

En la figura superior se observa cómo se ha conectado un potenciómetro al ADC. Esté el programa encendido o apagado, la FPGA funcionará independiente, y lo que controla la FPGA es el ADC y los leds. Desde el arranque del sistema la FPGA activa el ADC, recoge los datos del canal '0' y saca los 8 bits más significativos a través de los leds.

Si se ejecuta el programa en C, se accede a un registro de memoria dónde se envían los datos del ADC actualizados, por lo que se puede sacar por pantalla, como en este caso, o hacer otras operaciones distintas.

7.3.2. Un nuevo paradigma

En este recorrido se ha visto como partiendo de un ejemplo simple de base se pueden crear sistemas más completos, hasta llegar a un nuevo paradigma de programación hardware. Observando la trayectoria se ha visto como en un inicio se ha trabajado única y

exclusivamente con la FPGA, a través del ejemplo se ha visto como trabajar con un microprocesador, y desde ese ejemplo se ha llegado al uso de ambos en conjunto, lo que permite una mayor flexibilidad y aprovechamiento de los recursos de integración actuales.

El próximo paso es llevar esta unión bajo un entorno de tiempo real, lo que lleva a hacer un estudio de las alternativas disponibles y su implementación para poder empezar a diseñar sobre este entorno. Esto se verá en el siguiente capítulo.

8

Implementación: Real-Time Linux

Un sistema operativo siempre tiene latencias y nunca responde en el acto. Usando un Kernel en tiempo real se pueden reducir esas latencias y alterar la prioridad de los procesos. Además, existen extensiones que pueden mejorar esos tiempos aún más. En este capítulo se abordará el tema de los sistemas operativos en tiempo real, y se explicará cómo implementarlo en el sistema.

8.1. Introducción

Los sistemas operativos en tiempo real dan importancia a los procesos, no al usuario. Están destinados a unas aplicaciones muy específicas y que suelen ser deterministas [53] [54]. Algunos ejemplos de uso pueden ser el control del tráfico aéreo o el control ferroviario, automoción, producción y distribución de energía eléctrica o incluso en las bolsas de valores. Son ámbitos diferentes, y también con aplicaciones muy distintas, pero con una particularidad en común, el tiempo es crucial. La máquina tiene que tomar las decisiones lo más rápido posible, en cuanto le llegue la nueva información, ésta tendrá que ser procesada en el acto y generar inmediatamente una respuesta. Y por supuesto, tiene que ser robusto y responder a tiempo, sin perder ni un solo suceso.

8.2. Sistemas operativos

Antes de ahondar en el tema, conviene explicar brevemente algo de teoría sobre sistemas operativos. Hay que conocer cómo funciona un sistema operativo, como se ejecutan los procesos, cómo se hace la gestión de los recursos del sistema, las diferentes políticas de planificación, las condiciones de carrera…

Un sistema operativo es un software que administra el hardware del ordenador, además de proporcionar un entorno para ejecutar programas. Probablemente el aspecto más visible de un sistema operativo sea la interfaz que proporciona al usuario para comunicarse con el computador, se podría decir que actúa de intermediario entre el usuario y el hardware [55].

Un aspecto a tener en cuenta de los sistemas operativos es que varía en función de la finalidad. No es lo mismo un sistema operativo para un dispositivo embebido, que para un ordenador personal o para un dispositivo móvil. En el primer caso se busca optimizar la utilización del hardware, en el segundo el sistema tiene que soportar aplicaciones de todo tipo, y en el tercero prima la interfaz con el usuario y el ahorro energético.

En resumen, un sistema operativo tiene que estar diseñado para sacar el mayor provecho a los recursos del computador, y asegurar que se va a procesar la máxima cantidad de trabajo de la forma más eficiente posible. Aunque un sistema operativo no pueda incrementar la velocidad del sistema, sí que puede maximizar el uso de los recursos, haciendo de este modo que parezca más rápido, permitiendo hacer más tareas en un periodo de tiempo determinado [56].

8.2.1. El Kernel

El Kernel es el núcleo del sistema operativo. Contiene las funciones más frecuentemente usadas, y en cierto momento, otras porciones del sistema operativo en uso.

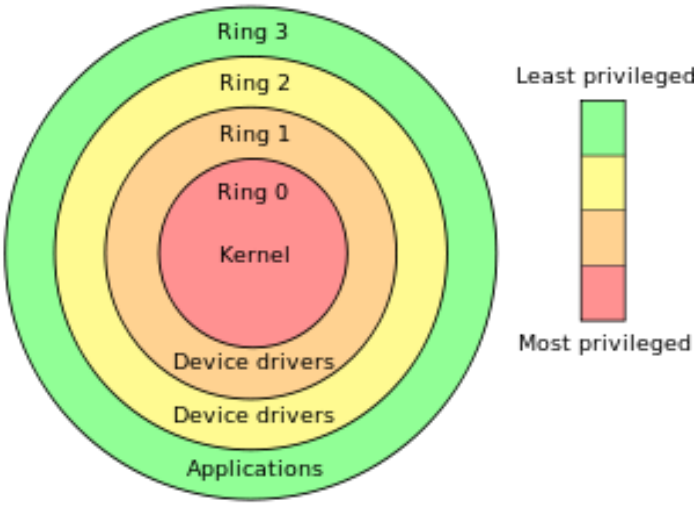


Figura 8.1 Anillos de privilegio.

Los sistemas operativos proporcionan diferentes niveles de acceso a los recursos [57]. El Kernel está en el nivel 0, por lo que tiene los máximos privilegios. Tiene acceso al hardware y puede hacer de todo. Por el contrario, el tercer anillo no puede ejecutar ciertas instrucciones ni tampoco escribir en ciertos registros [55] [58]. Para concretar, a continuación, se listarán algunas de las cosas que no se pueden hacer desde el nivel 3:

- No puede cambiar de anillo. Si pudiera cambiar, podría pasar al nivel 0 y tomar el control, de forma que la jerarquía sería inútil.
- No puede modificar las tablas de paginación. Esto significa que evita que un proceso vea la memoria de otros procesos. Hay razones de seguridad detrás.
- No se puede registrar el manejo de interrupciones. Se configuran mediante la escritura en ciertos registros de memoria. Estas se ejecutan en el anillo 0.
- No puede hacer operaciones de Entrada/Salida. Esto evita el acceso al hardware. Como ejemplo, si se pudiera hacer esto desde el nivel 3, los permisos de lectura y escritura serían inservibles, ya que entones cualquier programa podría acceder al disco. Para concretar más, el sistema operativo puede acceder a las instrucciones de E/S, lo que no puede es, el propio nivel 3, darse a sí mismo los permisos para hacerlo si no se le habían dado en un primer lugar.

Para resumir, el Kernel controla todas las interacciones entre la CPU y el mundo exterior, y determina que programas compartirán tiempo de proceso y en qué orden. Administra la memoria limitada tan bien que cientos de procesos pueden compartir el sistema de forma eficiente, y organiza las transferencias de datos de forma que la CPU no esté esperando más tiempo del necesario para los discos relativamente lentos [59].

8.2.2. Los procesos

En los primeros ordenadores sólo se permitía que un programa estuviese en ejecución al mismo tiempo. Este programa tenía el control total del sistema y tenía acceso a todos los

recursos del sistema. Eso ha cambiado, y los sistemas permiten cargar varios programas en memoria y ejecutarlos de forma concurrente. Esta evolución requería un control más firme y una mayor compartimentación de los diversos programas; y estas necesidades dieron como resultado la noción de un "proceso", que es un programa en ejecución. Un proceso es la unidad de trabajo en un sistema moderno de tiempo compartido.

Cuanto más complejo es el sistema operativo, más se espera que haga por sus usuarios. Si bien su principal cometido es la ejecución de los programas del usuario, también debe encargarse de varias tareas del sistema que es mejor dejar fuera del propio núcleo. Por lo tanto, un sistema consiste en una colección de procesos [60]: procesos del sistema y procesos del usuario. Potencialmente, todos estos procesos pueden ejecutarse simultáneamente, con la CPU multiplexada entre ellos. Al cambiar la CPU entre procesos, el sistema operativo puede hacer que la computadora sea más productiva. Si la CPU no quedara libre hasta que el proceso la liberara, sería monotarea.

Una pregunta que surge al discutir los sistemas operativos implica cómo llamar a todas las actividades de la CPU. Un sistema por lotes ejecuta trabajos, mientras que un sistema de tiempo compartido tiene programas de usuario o tareas. Incluso en un sistema de un solo usuario, éste puede ejecutar varios programas a la vez: un procesador de textos, un navegador web, un reproductor de música... Y aunque sólo se pudiera ejecutar un programa a la vez, como podría darse el caso en un dispositivo embebido que no admitiera tareas múltiples, es posible que el sistema operativo deba admitir sus propias actividades programadas internas, como la administración de memoria. En muchos aspectos, todas estas actividades son similares, por eso los llamamos procesos.

La definición informal de proceso es la de "programa en ejecución", pero realmente, el término va mucho más allá. Un proceso es más que el código del programa, también conocido como "sección de texto", sino que también incluye el valor del "contador de programa" y el contenido de los registros del procesador. Por otra parte, también incluye una pila, que contiene datos temporales (parámetros de la dirección, direcciones de retorno, variables locales…) y una sección de datos con las variables globales. Un proceso también podría incluir un "*heap*", que es memoria asignada de forma dinámica durante su tiempo de ejecución. En la siguiente imagen se puede observar la estructura de un proceso:

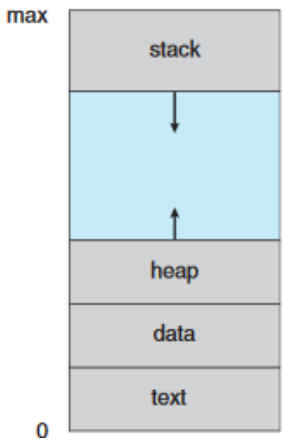


Figura 8.2 Estructura de un proceso en memoria.

Hay que enfatizar que un programa por sí mismo no es un proceso. Un programa es una entidad pasiva, como un fichero que contiene una lista de instrucciones almacenadas en disco (fichero ejecutable). En contraste, un proceso es una entidad activa con un contador de programa especificando la próxima instrucción a ejecutar y un set de recursos asociados. Un programa se convierte en proceso cuando un ejecutable es cargado en memoria.

Aunque dos procesos puedan estar asociados con el mismo programa, se consideran secuencias de ejecución diferentes. Por ejemplo, si se abren dos instancias de un explorador web, los dos son procesos distintos. Compartirán secciones de texto equivalentes, pero los datos, el "*heap*" y la pila serán diferentes, y cada proceso tendrá los suyos propios.

8.2.3. El planificador de procesos

Un sistema operativo debe asignar los recursos del computador entre las necesidades potencialmente competitivas de múltiples procesos. En el caso del procesador, el recurso que se debe asignar es el tiempo de ejecución en el procesador. La forma de asignarlo es la planificación. La función de planificación debe estar diseñada para satisfacer varios objetivos que incluyen equidad, ausencia de inanición de cualquier proceso, uso eficiente del tiempo del procesador y poca sobrecarga. Además, la función de planificación puede necesitar tener en cuenta diferentes niveles de prioridad o plazos de tiempo real para el inicio o finalización de ciertos procesos. Existen diferentes algoritmos para la planificación de procesos. Hoy en día, la investigación sobre la planificación se centra en la explotación de sistemas multiprocesador, particularmente para aplicaciones multihilo, y en la planificación de tiempo real.

El planificador de procesos, o "scheduler", es un componente funcional muy importante de los sistemas operativos multitarea y multiproceso, y es esencial en los sistemas operativos en tiempo real. En un instante dado, en el ordenador pueden existir diversos procesos listos para ser ejecutados, sin embargo, solamente uno puede ser ejecutado en cada microprocesador o núcleo. El planificador está para gestionar la cesión de recursos del sistema.

El funcionamiento básico de un planificador de procesos es muy sencillo, aunque por debajo se puede complicar bastante más. En esencia el "scheduler" asigna un tiempo (denominado "slice" o "quantum") al proceso para que haga uso de la CPU y los recursos del sistema. Acabado ese tiempo, el planificador recibe una interrupción y despacha al proceso para darle el turno a otro. Dicho así parece sencillo, pero hay que tener en cuenta otros factores. No se puede expulsar un proceso sin terminar de cualquier forma; hay que guardar el estado de ejecución del programa, lo que se denomina "contexto". Si esto no se hace, sería imposible continuar la ejecución, ya que los siguientes procesos que entren pueden modificar registros, inutilizando el progreso del primero. Se define al contexto como el conjunto mínimo de datos utilizado por una tarea que debe ser guardado para permitir su interrupción en un momento dado, y una posterior continuación desde el punto en el que fue interrumpida.

Además del contexto, también tiene más complicaciones. Existen diferentes políticas de planificación y algoritmos. Hay que destacar que no todos los procesos tienen por qué tener el mismo quantum, ni la misma prioridad. Pero cómo se ha dicho antes, se trata de conseguir una equidad y que ningún proceso se quede sin acceder a la CPU (inanición). Eso hay que tenerlo en cuenta a la hora de planificar, ya que un proceso de poca prioridad en algún momento se tiene que ejecutar, no puede estar permanentemente esperando en la cola.

8.2.3.1. Niveles de planificación

El objetivo de la planificación de procesos es asignar procesos a ser ejecutados por el procesador o procesadores a lo largo del tiempo, de forma que se cumplan los objetivos del sistema tales como el tiempo de respuesta, el rendimiento y la eficiencia del procesador. En muchos sistemas, esta actividad de planificación se divide en tres funciones independientes: planificación a largo, medio, y corto plazo. Los nombres sugieren las escalas de tiempo relativas con que se ejecutan estas funciones.

- Planificación a largo plazo. La decisión de añadir un proceso al conjunto de procesos a ser ejecutados. El planificador a largo plazo determina qué programas se admiten en el sistema para su procesamiento. De esta forma, se controla el grado de multiprogramación. Una vez admitido, un trabajo o programa de usuario se convierte en un proceso y se añade a la cola del planificador a corto plazo. En algunos sistemas, un proceso de reciente creación comienza en la zona de intercambio, en cuyo caso se añaden a la cola del planificador a medio plazo.
- Planificación a medio plazo. La decisión de añadir un proceso al número de procesos que están parcial o totalmente en la memoria principal. Es parte de la función de intercambio. Con frecuencia, la decisión de intercambio se basa en la necesidad de gestionar el grado de multiprogramación. En un sistema que no utiliza la memoria virtual, la gestión de la memoria es también otro aspecto a tener en cuenta. De esta forma, la decisión de meter un proceso en la memoria tendrá en cuenta las necesidades de memoria de los procesos que están fuera de la misma.
- Planificación a corto plazo. La decisión por la que un proceso disponible será ejecutado por el procesador. El planificador a corto plazo se invoca siempre que ocurre un evento que puede conllevar el bloqueo del proceso actual y que puede proporcionar la oportunidad de expulsar al proceso actualmente en ejecución en favor de otro. Algunos ejemplos de estos eventos son:
    - Interrupciones de reloj.
    - Interrupciones de E/S.
    - Llamadas al sistema.
    - Señales (por ejemplo, semáforos).
- Planificación de la E/S. La decisión por la que un proceso que está pendiente de una petición de E/S será atendido por un dispositivo de E/S disponible.

En términos de frecuencia de ejecución, el planificador a largo plazo ejecuta con relativamente poca frecuencia y toma la decisión de grano grueso de admitir o no un nuevo proceso y qué proceso admitir. El planificador a medio plazo se ejecuta más frecuentemente para tomar decisiones de intercambio. El planificador a corto plazo, conocido también como activador, ejecuta mucho más frecuentemente y toma las decisiones de grano fino sobre qué proceso ejecutar el siguiente.

8.2.3.2. Políticas de planificación

Existen diferentes políticas de planificación y algoritmos, pero lo más habitual es utilizar políticas mixtas. Generalmente, el planificador a corto plazo utiliza *Round-Robin*, mientras que el planificador a largo plazo utiliza varias colas FIFO. Cada una de estas colas corresponde a una prioridad diferente.

- Round-robin
- Round-robin con pesos.
- Prioridades monótonas en frecuencia (RMS (Rate-monotonic scheduling))
- Menor tiempo de respuesta primero (EDF (Earliest deadline first scheduling))
- FIFO - También conocido como FCFS "First Come, First Served".
- LIFO.
- SJF - Shortest Job First.
- CFS - Completely Fair Scheduler (ó Planificador Completamente Justo)
- SRT - Shortest Remaining Time
- SPT - Shortest Process Time
- Planificación mediante colas multinivel.

8.3. Linux-RT

Un sistema operativo en tiempo real se caracteriza por garantizar que todo programa se ejecutará en un límite máximo de tiempo. El planificador debe comportarse de manera que esto sea cierto para cualquier proceso.

En estos casos, la finalidad del planificador es balancear o equilibrar la carga del procesador, impidiendo que un proceso monopolice el procesador o que sea privado de los recursos de la máquina. En entornos de tiempo real, el planificador también impide que los procesos se paren o interrumpan a otros que esperan que se realicen ciertas acciones. Su labor resulta imprescindible para mantener el sistema estable y funcionando.

Para decidir qué sistema operativo usar se ha realizado un estudio, y aquí hay que tener en cuenta diferentes puntos. Se ha de buscar un sistema operativo que sea de tiempo real,

que las latencias máximas de respuesta sean bajas, que tenga soporte a largo plazo, que sea compatible con el dispositivo que se está utilizando y que no sea muy complicado de instalar.

8.3.1. Estudio de alternativas

Las FPGAs cada día están más presentes y están apareciendo nuevos campos de aplicación. Además, en la actualidad, no es extraño que vengan en un SoC acompañadas de un microprocesador. Esta combinación permite ejecutar un sistema operativo.

Debido a la creciente popularidad de estos dispositivos están apareciendo nuevos sistemas operativos para soportar ese hardware. Un buen ejemplo es el que está dando Fedora. La famosa distribución de Linux, gratuita y de código abierto, basada en Red Hat, está ya dando soporte a FPGAs en su versión 29. Es un paso importante, ya que estos dispositivos están teniendo hoy un papel importante en la industria. La inteligencia artificial, el aprendizaje automático ("Machine learning") o la blockchain son tecnologías novedosas en las que se están usando dispositivos lógicos programables. Que una distribución tan importante como Fedora apueste por dar soporte a las FPGAs es una buena noticia.

Continuando con el estudio se ha hecho un listado con posibles sistemas operativos en tiempo real a utilizar.

Distribución	Gratuito
Angstrom	Sí
Fedora 29	Sí
mAbassi	Limitado/No
ThreadX	No
VxWorks	No
Nucleus	No
OpenCV	Sí

Tabla 8.1 Listado de sistemas operativos RT.

Tras analizar los sistemas operativos listados, por el momento, se desea utilizar una versión gratuita. En el caso de mAbassi la versión gratuita o está muy limitada en características o está limitada en tiempo. Así que ha quedado descartada por no ser completamente funcional. También hay que comentar que está más orientada hacia arquitecturas paralelas, que no es el área en el que se está trabajando en este proyecto.

OpenCV era también una buena alternativa, está basado en el proyecto Yocto. Tiene interfaz gráfica, pero es algo que no se necesita y que lo que va a hacer es consumir más recursos. Además, otro punto en contra es que no hay una imagen lista para el dispositivo, habría que compilarla.

Angstrom es la distribución que viene de seria con la tarjeta de evaluación. La versión por defecto es la de interfaz gráfica, pero para que funcione bien, hubo que descargar la versión de consola. También está basada en Yocto, pero no es real-time. Debido al soporte que ofrece, es una buena alternativa.

En cuanto a Fedora, también tiene muchos puntos a favor. Es muy conocida, tiene un buen soporte, una comunidad de usuarios muy grande… Finalmente se decidió probarla, pero no se consiguió crear la imagen debido a errores de compilación, por lo que se dejó al margen.

Tras el estudio y las pruebas realizadas, se escogió Angstrom. Y sí, se ha dicho que no es real-time, pero sí que se puede cambiar el Kernel por otro de tiempo real. Además, es la propia Altera-Intel la que mantiene ese Kernel con "preempt_rt", y entre sus ventajas está que es de código abierto y está actualizado.

8.3.2. Instalación del Kernel

El Kernel que se va a instalar va a llevar el parche "preempt_rt". Y se explica a continuación en qué consiste:

Dentro del scheduler, la función de selección determina qué proceso, entre los todos los que están listos para tomar el control de la máquina, se selecciona para su ejecución. Entre las políticas de planificación, existe el modo de decisión, que especifica los instantes de tiempo en que se ejecuta la función de selección. Se pueden resumir en dos categorías generales:

- Sin expulsión (non-preemtive). Una vez que el proceso está en ejecución, se continúa ejecutando hasta que se termina o hasta que se bloquea para esperar E/S o para solicitar algún servicio al sistema operativo.
- Con expulsión (preemptive). El proceso que se está ejecutando puede ser interrumpido y devuelto a la lista de procesos preparados para ejecutar. Esta decisión de expulsar el proceso puede ser tomada cuando llega un nuevo proceso, cuando llega una interrupción que cambia el estado de un proceso de bloqueado a preparado, o periódicamente, basándose en las interrupciones del reloj. Cómo se ha explicado antes, esto implica guardar el contexto para evitar problemas.

Como se va a trabajar en tiempo real, se desea que un proceso de urgencia pueda interrumpir a otro de menor prioridad, por lo que el sistema operativo tendrá una directiva "preemptive".

Lo primero de todo es saber de qué versión del Kernel se parte. Para eso se recurre al comando uname –a, que sirve para imprimir por pantalla la información del sistema.

```
root@socfpga:/kernel/linux-socfpga-socfpga-4.14.73-ltsi-rt# uname -a
Linux socfpga 4.5.0-00185-g3bb556b #3 SMP Wed Aug 16 15:56:01 CST 2017 armv7l GN
U/Linux
```

Figura 8.3 Información del sistema. Kernel 4.5.0.

Antes de empezar ningún proceso de compilación, es necesario tener en cuenta dos cosas. La primera es que el sistema debe estar actualizado, y la segunda, que para compilar será necesario tener instaladas ciertas utilidades y librerías.

8.3.2.1. Pasos previos a la compilación

Se actualizan los repositorios de Linux y posteriormente se comprueba que el sistema esté actualizado. Esto se hace a través de un gestor de paquetes, y en la distribución que se está usando, ese gestor es opkg.

```
opkg update
opkg upgrade
```

Ahora hay que instalar las herramientas y librerías necesarias para el proceso de compilación, algunas es probable que ya estén instaladas.

```
opkg install build-essential gcc make libncurses5-dev libssl-dev
opkg install bison flex qt5-default qt5-qmake fakeroot
opkg install qtbase5-dev qtbase5-dev-tools
```

8.3.2.2. Obtención del Kernel

Cuando se compila un Kernel de RT, lo normal sería buscar un parche de RT de la versión que se quiera instalar y buscar el Kernel que se corresponde a ese parche. Este paso es debido a que hay más versiones de Kernel que de parches. Esto se haría a través de la web "kernel.org".

Para evitar complicaciones en esta etapa, se ha recurrido al Kernel RT que proporciona Intel-Altera [61] [62]. Lo tienen publicado en la plataforma GitHub. El usuario es "Altera-Opensource" y el repositorio es "Linux-socfpga". Aquí hay que tener especial cuidado, ya que el repositorio contiene "branches" o ramificaciones. Es decir, el mismo proyecto tiene diferentes versiones. En este caso se ha elegido el branch "socfpga-4.14.73-ltsi-rt", ya que es el Kernel RT más actual que proporcionan, y además es ltsi, que hace referencia a "Long Term Support for the Industry", lo que significa que es una versión fiable, robusta y que va a tener soporte durante al menos 2 años.

Tras descargar el Kernel desde la fuente oficial en formato "zip", se ha descomprimido en una partición de la memoria microSD con tamaño suficiente. Posteriormente se ha insertado la tarjeta en la placa de desarrollo y se ha montado la partición.

```
mount /dev/[dispositivo] [ruta]
```

8.3.2.3. Compilación del Kernel

Ya están instaladas las aplicaciones necesarias y el código fuente está listo, así que ya está todo preparado para comenzar el proceso de compilación.

```
make socfpga_defconfig
make –j12 zImage 2>&1 | tee ../linux-rt-log.txt
```

Con estas dos sentencias se inicia la compilación. En la segunda se está compilando el Kernel y obteniendo una imagen comprimida del mismo, bajo el nombre de "zImage". Interesa desviar la salida de errores por la salida estándar, pero mandándola hacia un fichero de texto. Así, en caso de que surja algún problema durante la compilación, se podrá revisar.

El fichero “zImage” es una versión comprimida del Kernel de Linux, y es auto-extraíble. En esencia son los binarios del Kernel compilado. Es lo que el bootloader o cargador de arranque, tratará de ejecutar al inicio del sistema.

La compilación puede tardar cerca de dos horas. Durante ese proceso apareció este error:

```
arm-angstrom-linux-gnueabi-ld: --pic-veneer: unknown option
arm-angstrom-linux-gnueabi-ld: use the --help option for usage
information
```

El problema es que la distribución de Angstrom usa por defecto el “Gold Linker”. El Kernel de Linux puede mostrar problemas a la hora de enlazar módulos basados en Vybrid cuando se usa este “linker”. La solución recomendada que dan ellos mismos, es la de revertir los cambios y usar el “bfd linker”. Para ello se usó el siguiente comando que también proporcionaban.

```
make -j3 zImage LD=${CROSS_COMPILE}ld.bfd | tee build.log
```

Tras un rato la compilación al fin terminó. En la siguiente captura de pantalla se observa como el compilador informa de que la imagen del kernel está lista.

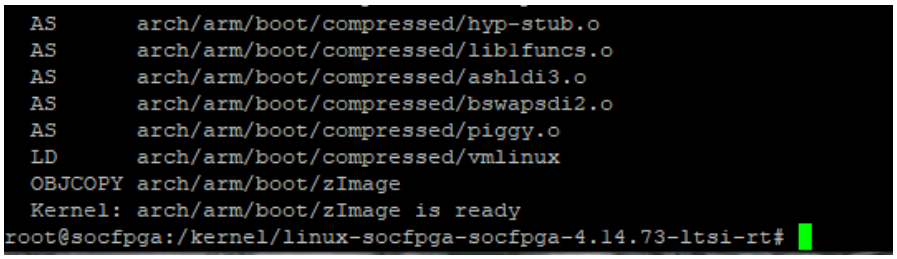


Figura 8.4 Compilación completada con éxito.

Se comprueba que la imagen esté en el destino indicado. Para eso, simplemente se comprueba los ficheros existentes en la ruta. (Nota: el escenario actual será necesario más tarde. No se debe borrar si se desea programar módulos del Kernel).

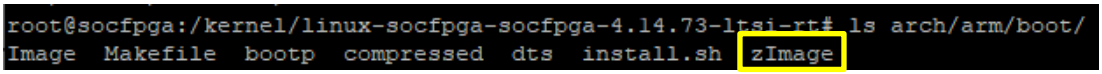


Figura 8.5 Imagen del Kernel lista.

8.3.2.4. Puesta en marcha y pruebas

Lo que se hace a continuación es obtener el fichero zImage y copiarlo a la partición de arranque. En este caso se ha obtenido insertando la tarjeta en un entorno de Linux, accediendo a la ruta para obtener el fichero y copiándolo en la partición de arranque. Se sustituye este nuevo fichero por el anterior, que es el del Kernel primitivo. El último paso es arrancar la placa de desarrollo y ver qué sucede.

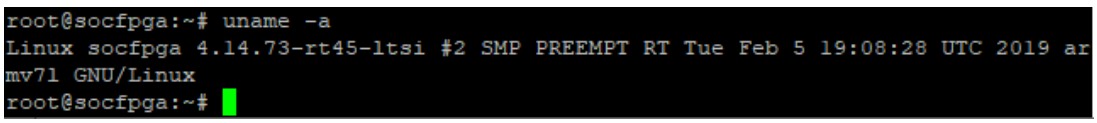


Figura 8.6 Información del nuevo Kernel.

Se observa en la figura que la versión del Kernel ha cambiado. Ahora está instalado una versión más nueva y de tiempo real. En las siguientes páginas se explicará la programación de

aplicaciones en tiempo real, se probará y se hará un análisis de los resultados en más profundidad, pero de momento, lo que sí se puede hacer, es ver el funcionamiento de ambos Kernels con unos programas de testeo, tal y como se muestra en las dos figuras que aparecen a continuación.

```
root@socfpga:~/rt-tests# uname -a
Linux socfpga 4.5.0-00185-g3bb556b #3 SMP Wed Aug 16 15:56:01 CST 2017 armv7l GN
U/Linux
root@socfpga:~/rt-tests# ./cyclictest -m -t1 -p 80 -n -i 500 -l 100000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.01 0.12 0.08 1/66 1359

T: 0 ( 1359) P:80 I:500 C: 100000 Min:      7 Act:    7 Avg:    7 Max:      118
root@socfpga:~/rt-tests# ./cyclictest -m -t1 -p 80 -n -i 500 -l 100000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.23 0.21 0.10 1/67 1355

T: 0 ( 1355) P:80 I:500 C: 100000 Min:      7 Act:    7 Avg:    7 Max:       38
root@socfpga:~/rt-tests#
```

Figura 8.7 Ejecución del test en el Kernel nativo.

```
root@socfpga:~/rt-tests# uname -a
Linux socfpga 4.14.73-rt45-ltsi #2 SMP PREEMPT RT Tue Feb 5 19:08:28 UTC 2019 ar
mv7l GNU/Linux
root@socfpga:~/rt-tests# ./cyclictest -m -t1 -p 80 -n -i 500 -l 100000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.98 0.53 0.22 2/82 1357

T: 0 ( 1357) P:80 I:500 C:  99871 Min:      9 Act:   10 Avg:   10 Max:       29
root@socfpga:~/rt-tests# ./cyclictest -m -t1 -p 80 -n -i 500 -l 100000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 1.03 0.61 0.27 1/82 1359

T: 0 ( 1359) P:80 I:500 C:  99870 Min:      9 Act:   10 Avg:   10 Max:       25
root@socfpga:~/rt-tests#
```

Figura 8.8 Ejecución del test en el Kernel Preempt-RT.

Los tiempos se miden en µs, y se observa que, aunque el tiempo mínimo sea inferior en el primer Kernel, el tiempo de la ejecución que más ha tardado es inferior, lo que significa que el Preempt-RT logra ser más determinista.

Además de este Kernel, también se ha probado a compilar el Kernel 4.19.15 con el correspondiente parche de Preempt-RT y otro con Xenomai, pero han dado peores resultados y han sido descartados.

8.4. Comprobación de latencias

Se acaba de explicar la compilación del Kernel Preempt-RT y se han hecho unos tests básicos, el siguiente paso es ponerlo a prueba y efectuar otro tipo de comparativas. Se desea saber si el nuevo Kernel aporta ventajas para la implementación del sistema de control, así que habrá que probarlo en diferentes situaciones, obtener resultados fiables y analizar los datos.

Se ha diseñado un escenario real para probar estos retardos. El µP se debe comunicar con la FPGA, es decir, va a haber un intercambio de datos bidireccional. Es por ello que se ha querido medir la latencia que tiene una estructura de anillo, en el que la FPGA envía un dato

al HPS, lo analiza y genera otra respuesta que le llega a la FPGA. Esto queda mejor explicado en el siguiente esquema:

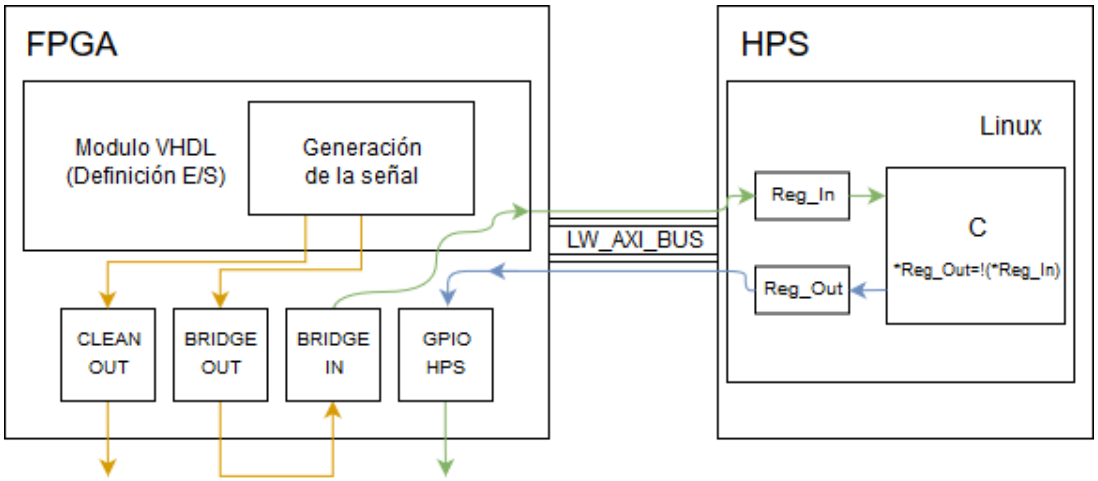


Figura 8.9 Esquema de comunicación circular.

En el lado de la FPGA existe un módulo escrito en VHDL que define las entradas y salidas de los periféricos y un generador de pulsos. Al principio la señal se generaba manualmente con la ayuda de una de las teclas de la placa de desarrollo, pero se ha decidido automatizar el proceso para que sea más simple y menos tedioso.

Para medir la latencia hay que comparar el tiempo entre la salida de la señal original y la salida de la señal modificada por el HPS, ya que este último invierte la señal. Se pretende también en este escenario simular la adquisición de datos, por eso, la señal de salida es duplicada y leída de nuevo por la FPGA, se está haciendo un "puente" y de ahí el nombre asignado a las señales. Tras la lectura del dato por la FPGA, éste se enviará al HPS, viajando por el bus de comunicaciones hasta uno de los registros. El µP se encargará de leer el dato y modificarlo, para ponerlo de nuevo en el registro. La FPGA se encargará de adquirir el dato y sacarlo al exterior a través de otro de los GPIO disponibles. Con un osciloscopio se han analizado las dos salidas y se han medido los tiempos con la función de Δt, que mide el tiempo que hay entre la aparición de dos eventos.

Este experimento se puede repetir múltiples veces y hallar valores como el número de iteraciones, la media aritmética, los valores máximo y mínimo, el rango entre ambos y la desviación típica. Además, lo interesante es medirlo en diferentes condiciones, por un lado, existen dos imágenes del Kernel, la nativa y la versión Preemt-RT, por otro lado, se pueden efectuar pruebas cambiando la prioridad del programa o introduciendo otros procesos (fundamentalmente pruebas de estrés, que saturen el sistema), y ver su comportamiento. No sólo se desea que obtener unas bajas latencias, si no también garantizar que no habrá problemas de inanición.

8.4.1. Análisis de los resultados

En la siguiente tabla se han recopilado los resultados de las pruebas realizadas. El procesador tiene dos núcleos, por lo que puede ejecutar procesos en paralelo, es por ello por lo que

cuando marca “con carga x2” hace referencia a que hay dos aplicaciones de stress en ejecución además de la de adquisición de datos [63].

Condición	Media	Min	Max	Desv. Típ.	N.º Itr.
Kernel nativo (sin carga, 1º plano)	798ns	393ns	1,21µs	226ns	100
Kernel nativo (sin carga, 2º plano)	818ns	393ns	1,20µs	217ns	100
Kernel nativo (con carga, 1º plano)	824ns	390ns	1,20µs	222ns	100
Kernel nativo (con carga x2, 1º plano)	5,41µs	400ns	187µs	25,9µs	100
Kernel nativo (con carga x2, 2º plano)	17,7µs	399ns	911µs	108,7µs	100
Preempt-RT (Sin carga, 1º plano)	768ns	414ns	1,15µs	220ns	100
Preempt-RT (Sin carga, 2º plano)	818ns	392ns	1,21µs	232ns	100
Preempt-RT (Sin carga, 1º plano, Sched: RR, p1)	844ns	398ns	1,21µs	223ns	100
Preempt-RT (Sin carga, 1º plano, Sched: RR, p99)	780ns	394ns	1,21µs	235ns	100
Preempt-RT (con carga, 1º plano)	904ns	403ns	11,1µs	1,05µs	100
Preempt-RT (con carga x2, 1º plano)	10,4µs	399ns	274µs	41,2µs	100
Preempt-RT (con carga x2-cyclic p70, 1º plano, Sched: RR p90)	1,85µs	394ns	6.9µs	1,92µs	100

Tabla 8.2 Resultados obtenidos.

Se observa que para que la ejecución de los procesos sea correcta y se ejecute con unas latencias máximas asumibles hay que hacer una buena planificación. El Preempt-RT bien configurado, como es el caso en el que se le da un scheduling prioritario, mejora considerablemente las latencias máximas en contraposición con el Kernel nativo.

Hay que añadir que durante las pruebas se han dado casos de inanición, y aunque la aplicación seguía en ejecución porque así se podía ver en el osciloscopio, se ha perdido la comunicación con el sistema, tanto mediante SSH como por puerto serie. Esto indica una mala planificación, y en el futuro, cuando se haga el sistema definitivo habrá que hacer pruebas más intensas, ya que las actuales son solamente una aproximación para ver el funcionamiento. Las configuraciones en las que ha habido inanición se han descartado y no han sido incluidas, ya que son consideradas erróneas.

Se pretende también hacer un enfoque en estos cinco casos de prueba, ya que se consideran significativos. Lo que se pretende mostrar son las diferencias entre el kernel original y el que ha sido parcheado con Preempt-RT, además de cómo se comporta al asignarle diferentes prioridades o al añadirle sobrecarga.

Condición	Media (ns)	Max (µs)	Desv. Tip. (ns)
Kernel nativo (sin carga, 1º plano)	798	1,21	226
Preempt-RT (Sin carga, 1º plano, Sched: RR, p1)	844	1,21	223
Preempt-RT (Sin carga, 1º plano, Sched: RR, p99)	780	1,21	235
Kernel nativo (con carga x2, 1º plano)	541000	187	259000
Preempt-RT (con carga x2-cyclic p70, 1º plano, Sched: RR p90)	185000	6,9	192000

Tabla 8.3 Resultados más significativos.

Con estos resultados, se pueden obtener unas gráficas para ver con más detalle las diferencias existentes. Hay que tener en cuenta que la escala es logarítmica:

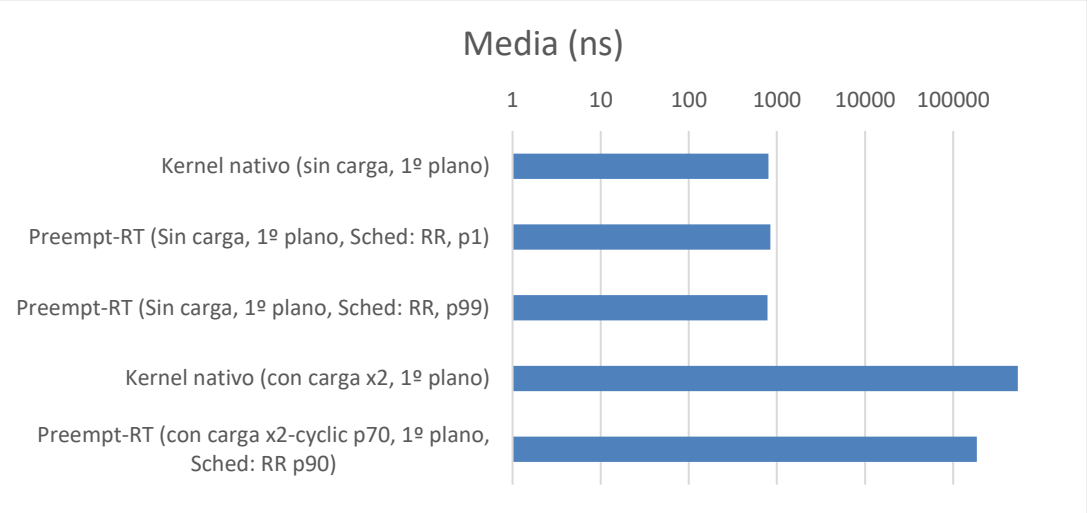


Figura 8.10 Comparativa del tiempo de respuesta medio en distintos escenarios.

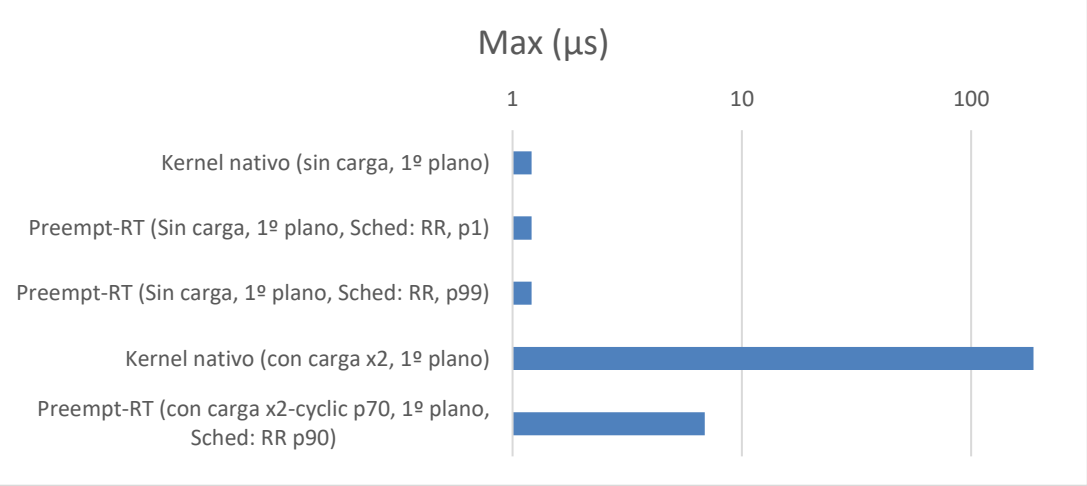


Figura 8.11 Comparativa del tiempo de respuesta máximo en distintos escenarios.

En estas gráficas se ve como el parche de Preempt-RT puede mejorar los tiempos de respuesta. Analizando la media, se observa que, sin carga computacional adicional, mejora algo respecto al Kernel original siempre y cuando se le asigne una alta prioridad. Pero donde

más se nota es con carga, ya que de media el Kernel parcheado responde en 1,85µs con una prioridad de nivel 70, mientras que el Kernel nativo lo hace en 5,40µs.

El parche de Preempt-RT y la asignación de una alta prioridad hacen que, ante una alta carga de trabajo, las latencias máximas para la respuesta se sigan manteniendo bajas, por debajo de los 7µs. Con el Kernel nativo no se pueden conseguir esos tiempos de respuesta, y la espera máxima es de casi 190µs, considerablemente superior.

En resumen, cuando se requiere un sistema fiable, que responda en unos márgenes cortos de tiempo, se debe recurrir a un Kernel de tiempo real.

8.5. Módulos del Kernel (Interrupciones)

En este apartado se van a estudiar e implementar las interrupciones. La motivación para dar este paso es que hasta ahora se estaba programando la intercomunicación mediante el mecanismo de "polling" o encuesta, lo que resulta bastante ineficiente. El cambio a una política de interrupciones podría mejorar el rendimiento y la eficiencia, trayendo consigo una mejora en las latencias.

Para poder usar este método va a ser necesario crear un módulo del Kernel. Antes de resolver el problema, conviene hacer una breve explicación de qué son y para qué sirven.

8.5.1. ¿Qué es un módulo del Kernel?

Los módulos son fragmentos de código que pueden ser habilitados o deshabilitados en el Kernel bajo demanda [64]. Sirven para extender la funcionalidad del Kernel sin tener que reiniciar el sistema. Un buen ejemplo podría ser el controlador de un dispositivo, que lo que hace es permitir que el Kernel acceda al hardware conectado al sistema. Sin módulos habría que construir Kernels monolíticos y añadir directamente las nuevas funcionalidades en su imagen. Ésto tendría la desventaja de que, además de ocupar más espacio, habría que recompilarlo de nuevo y reiniciar el sistema cada vez que se quisiera añadir una nueva funcionalidad. Por otra parte, a continuación, se enumeran algunas de sus ventajas:

- Enfoque modularizado: Dado que cualquier módulo puede ser habilitado o deshabilitado mientras el sistema está en ejecución, los programadores de sistemas tienen que introducir interfaces de software bien definidas para acceder a las estructuras de datos que manejan los módulos. Esto facilita el desarrollo de nuevos módulos.
- Independencia de plataforma: Aunque un módulo pueda depender de alguna característica específica de hardware, no depende de una plataforma fija. Por ejemplo, un módulo que controle un protocolo SPI estándar lo hará igual de bien en un PC como en la Cyclone V [59].

- Uso austero de la memoria principal: La habilitación y deshabilitación es muy útil cuando se trabaja con pequeños sistemas embebidos. Además, al deshabilitar el módulo cuando no se necesita, se ahorra memoria RAM, ya que es donde se guarda la imagen del Kernel en ejecución.
- No hay penalización de rendimiento: Una vez que el módulo se ha habilitado, su código objeto es equivalente al que está estático en el Kernel. No es necesario recurrir al pase explícito de mensajes cuando una de las funciones del módulo es invocada, al igual que ocurre con las funciones del Kernel. Sí que hay una pequeña penalización al habilitar o deshabilitar módulos, pero no es sustancial.

8.5.2. Programación de los módulos

Los módulos son programas y, por tanto, hay que tener en cuenta algunas cosas a la hora de programarlos. Se desea que el código de un módulo pueda utilizar cualquier función del núcleo, pero este código no ha sido enlazado en tiempo de compilación con el kernel, por lo que las referencias a las funciones del núcleo no están resueltas, para ello se utilizan "macros".

Hace falta una macro que indique al Kernel qué función es la que debe usar en la inicialización del módulo, ésta añade una sección especial en el código objeto del módulo. Cuando se haya cargado el módulo, los recursos del sistema utilizados serán liberados. Del mismo modo hace falta otra macro que indique qué función se ejecutará para extraer el módulo.

Todas las funciones se definirán como estáticas ("*static*"), ya que no deben ser vistas fuera del fichero.

Hay que tener en cuenta que los módulos son ficheros objeto especiales (.ko). Los ficheros objeto (.o) se pueden convertir en módulos compilándolos con un fichero Makefile con una sintaxis especial, creado por GNU, con el fin de facilitar la tarea de compilar los módulos.

8.5.3. Conociendo la gestión de interrupciones entre HPS y Linux

Para aclarar el funcionamiento del sistema hay que apoyarse en el siguiente esquema. Éste muestra como un controlador, presente en el Kernel de Linux, gestiona las peticiones de interrupción generadas por un módulo en VHDL que se encuentra en la FPGA [65].

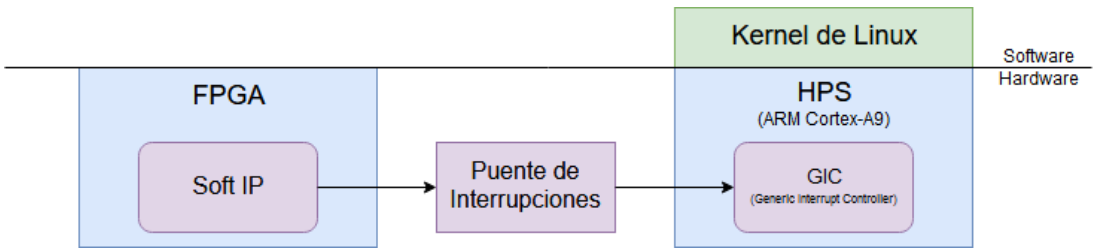


Figura 8.12 Esquema de la gestión de interrupciones.

La Cyclone V de Intel tiene en su interior un ARM Cortex-A9, que es el que va a recibir las interrupciones de la FPGA. El ARM tiene un controlador de interrupciones, denominado GIC (*Generic Interrupt Controller*). Su integración se puede ver en la siguiente imagen:

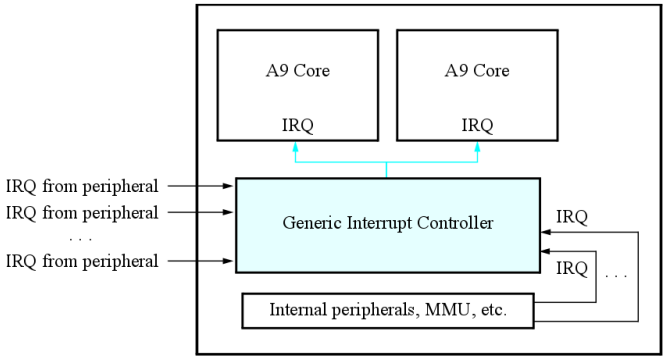


Figura 8.13 Esquema del controlador de interrupciones integrado en el ARM.

El GIC tiene dos tareas principales. En primer lugar, actúa como distribuidor, repartiendo de forma eficiente las interrupciones entre los dos núcleos del sistema (el ARM tiene una arquitectura *dual-core*). Y, en segundo lugar, otorga a cada núcleo una interfaz de CPU. Esto último, entre otras cosas, permite a la CPU consultar la fuente de la interrupción.

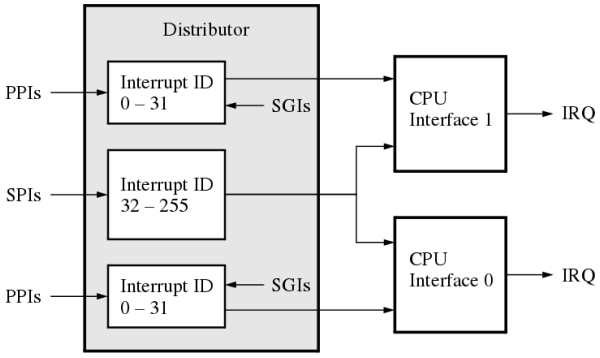


Figura 8.14 Arquitectura del GIC.

El GIC permite interrupciones privadas de periféricos (PPI) e interrupciones generadas por software (SGI), destinadas únicamente a una CPU en concreto, mientras que difunde las interrupciones compartidas de periféricos (SPI) a ambos núcleos.

En el caso del controlador integrado en el Cyclone V, puede soportar hasta 180 fuentes de interrupción, de las cuales, 64 están dedicadas a los módulos sintetizables en la FPGA. Los números asignados para esas interrupciones van del 72 al 135. Esto es importante, ya que a la hora de diseñar en Quartus, la interrupción de valor "0" que se le asigne a un módulo en HDL, se corresponderá con la interrupción número "72" en Linux.

Otra cosa a tener en cuenta es que ahora el paso a dar es que Linux reconozca una interrupción de un periférico, y para ello habrá que modificar el "árbol de dispositivos" o "*device tree*". Antes conviene explicar lo que es.

8.5.3.1. El árbol de dispositivos

El "device-tree" es una estructura de datos que describe los componentes hardware de un sistema en particular, de modo que el Kernel del sistema operativo pueda administrarlos. Esto incluye CPUs, memoria, buses y periféricos.

Dado un árbol de dispositivos válido, un mismo Kernel puede soportar diferentes configuraciones de hardware. Los Kernels de Linux para arquitecturas ARM, X86 o MicroBlaze leen esta estructura de datos. De hecho, para ARM es obligatorio para todos los nuevos SoCs desde el año 2012 [66]. Esto puede ser visto como una solución al inmenso número de "*forks*" o bifurcaciones de Kernels que se han creado para soportar diferentes placas de ARM.

El propósito del "device-tree" es el de desplazar una parte significativa de la descripción de hardware fuera de los binarios del Kernel. Esto evita que haya que hacer modificaciones en el Kernel para dar soporte a diferentes placas o a revisiones futuras de las mismas, además de que evita el proceso de compilación, ya que el mismo Kernel es válido para diferentes dispositivos.

8.5.4. Preparación del entorno Linux

En este apartado se pretende explicar qué hay que hacer en Linux para poder recibir las interrupciones. Lo primero que se necesita es el código fuente del Kernel. En las páginas anteriores se explica cómo se compiló un Kernel de Linux con el parche de Preempt-RT, ese escenario será necesario para realizar el actual desarrollo.

8.5.4.1. Actualización del "device tree"

Se debe de actualizar el "device-tree" con la descripción para el módulo Soft IP. Dado que se está utilizando la placa Terasic DE10-Nano, se ha utilizado el archivo con extensión ".dts" del Golden Hardware Reference Design (GHRD). Este fichero contiene el árbol de dispositivos de la placa y se puede encontrar en el CD o en los recursos de la web.

En la misma ubicación donde el Kernel fue compilado, hay que copiar el fichero mencionado en la siguiente ruta parcial: "arch/arm/boot/dts". Ahí también se creará un fichero llamado "socfpga_test_int.dts", con el siguiente código en su interior:

```
root@debian:/home/jose/linux-socfpga-socfpga-4.14.73-ltsi-rt# touch arch/arm/
boot/dts/socfpga_test_int.dts
```

Figura 8.15 Creación del fichero en la ruta descrita.

```
#include <dt-bindings/interrupt-controller/irq.h>
#include <dt-bindings/interrupt-controller/arm-gic.h>
#include "socfpga_cyclone5_de10_sockit.dts"

/*Extends the SoC with a soft component.*/
/ {
    soc {
        mysoftip {
            compatible = "altr , socfpga-mysoftip";
```

```
            interrupts = <GIC_SPI 40 IRQ_TYPE_EDGE_RISING>;
        };
    };
};
```

A continuación, se compila el fichero utilizando el siguiente comando desde la raíz del Kernel:

```
make socfpga_test_int.dtb
```

```
root@debian:/home/jose/linux-socfpga-socfpga-4.14.73-ltsi-rt# make socfpga_te
st_int.dtb
  CHK     scripts/mod/devicetable-offsets.h
  DTC     arch/arm/boot/dts/socfpga_test_int.dtb
```

Figura 8.16 Comando y resultado de la compilación.

Tras la compilación, habrá que copiarlo en la partición FAT32 de la SD, que es la que contiene el Bootloader. El nombre que hay que darle para que cargue ese árbol es "socfpga.dtb", aunque también se puede modificar el fichero u-boot para especificarle el nombre del "device-tree" que se desea cargar. Es muy recomendable tener copias de seguridad antes de alterar la configuración de arranque.

8.5.4.2. Programación del controlador (Módulo del Kernel)

El módulo del Kernel es la parte intermedia entre el Hardware y el Software. Se va a encargar de recoger la interrupción que le llega. En esencia es lo que se denomina un controlador o "*driver*".

```
#include <linux/module.h> /*Necesario para todos los modulos*/
#include <linux/kernel.h> /*Necesario para KERN_INFO*/
#include <linux/init.h>   /*Necesario para las macros*/
#include <linux/interrupt.h>
#include <linux/sched.h>
#include <linux/platform_device.h>
#include <linux/io.h>
#include <linux/of.h>

#define DEVNAME "test_int"

static irq_handler_t __test_isr(int irq, void *dev_id, struct pt_regs *regs){
    printk (KERN_INFO DEVNAME ": ISR\n");
    return (irq_handler_t) IRQ_HANDLED;
}

static int __test_int_driver_probe(struct platform_device* pdev){
    int irq_num;
    irq_num = platform_get_irq(pdev, 0);
    printk(KERN_INFO DEVNAME ": La IRQ %d va a ser registrada!\n", irq_num);
    return request_irq(irq_num, (irq_handler_t) __test_isr, 0, DEVNAME, NULL);
}

static int __test_int_driver_remove (struct platform_device *pdev){
    int irq_num;
    irq_num = platform_get_irq (pdev, 0);
    printk(KERN_INFO "test_int: Abandonando la captura de la IRQ %d !\n",
irq_num);
    free_irq(irq_num, NULL);
```

```
    return 0;
}

static const struct of_device_id __test_int_driver_id[] = {
    {.compatible = "altr , socfpga-mysoftip"},
    {}
};

static struct platform_driver __test_int_driver = {
    .driver= {
        .name = DEVNAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr (__test_int_driver_id),
    },
    .probe = __test_int_driver_probe,
    .remove = __test_int_driver_remove
};

module_platform_driver (__test_int_driver);

MODULE_LICENSE("GPL");
```

test_int.c

```
obj-m += test_int.o
KERNELPATH=/home/jose/linux-socfpga-socfpga-4.14.73-ltsi-rt/

all:
    make -C $(KERNELPATH) M=$(PWD) modules

clean:
    make -C $(KERNELPATH) M=$(PWD) clean
```

Makefile

Tras la elaboración del programa y del fichero Makefile, se ha compilado el módulo. Se ha listado el directorio para comprobar que los ficheros se han generado correctamente.

```
root@debian:/home/jose/driver# make
make -C /home/jose/linux-socfpga-socfpga-4.14.73-ltsi-rt/ M=/home/jose/driver
 modules
make[1]: se entra en el directorio '/home/jose/linux-socfpga-socfpga-4.14.73-
ltsi-rt'
  CC [M]  /home/jose/driver/test_int.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/jose/driver/test_int.mod.o
  LD [M]  /home/jose/driver/test_int.ko
make[1]: se sale del directorio '/home/jose/linux-socfpga-socfpga-4.14.73-lts
i-rt'
root@debian:/home/jose/driver# ls
Makefile       Module.symvers        test_int.c   test_int.mod.c  test_int.o
modules.order  socfpga_test_int.dtb  test_int.ko  test_int.mod.o
```

Figura 8.17 Resultado de la compilación del módulo.

A continuación, hay que pasar los nuevos ficheros generados al sistema. Al igual que en otras ocasiones, este paso se ha realizado mediante el protocolo SFTP. Por último, se comprueba que el programa funcione. Hay que decir que aún no se ha programado el VHDL,

por lo que no se está provocando ningún evento, pero el módulo instanciado debería dar el siguiente mensaje:

```
root@socfpga:~/driverInterrupt# insmod test_int.ko
[  299.414891] test_int: La IRQ 41 va a ser registrada!
root@socfpga:~/driverInterrupt# rmmod test_int.ko
[  308.383320] test_int: Abandonando la captura de la IRQ 41 !
```

Figura 8.18 Instanciación y deshabilitación del módulo.

Otra prueba más de que el módulo está en funcionamiento es acceder al fichero "/proc/interrupts" mientras está instanciado. Con el comando de Linux "more" se puede leer.

```
 39:          0          0     GIC-0 160 Level     ffb40000.usb, ffb40000.usb, d
wc2_hsotg:usb1
 41:          0          0     GIC-0  72 Edge      test_int
IPI0:         0          0  CPU wakeup interrupts
IPI1:         0          0  Timer broadcast interrupts
--More--(0%)
```

Figura 8.19 Lectura del fichero "/proc/interrupts".

Se observa que la interrupción 41 está asociada a la 72 que se ha comentado con anterioridad, se captura en el borde del flanco y la está registrando el módulo test_int.

8.5.5. Diseño del sistema Hardware

En este punto ya está preparado el entorno de Linux para recibir interrupciones provenientes de la FPGA, ahora es el turno de elaborar un sistema que envíe las interrupciones cuando sea necesario [67]. El desarrollo que se va a realizar es un poco especial, ya que en primer lugar se va a diseñar un módulo en HDL y posteriormente se va a integrar en el sistema como un IP Core.

8.5.5.1. Diseño del módulo en HDL

Se ha implementado un módulo en lenguaje VHDL. En él se describen las interfaces de entrada y salida y la lógica. El módulo envía una interrupción cuando sucede un evento, en este caso de prueba, esto ocurre cuando un switch cambia de valor.

En el lado software, cuando se reciba la interrupción, se debe de actuar e informarle al hardware de que lo ha leído y ha actuado, esto se hace mediante una señal de "read". El hardware entonces actualizará su valor. El código es el siguiente:

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

ENTITY entrada IS
    PORT(
        FPGA_CLK1_50: IN STD_LOGIC;
        reset       : IN STD_LOGIC;
        switches    : IN STD_LOGIC_VECTOR(3 downto 0);
        avl_irq     : OUT STD_LOGIC;
        avl_read    : IN STD_LOGIC;
        avl_readdata: OUT STD_LOGIC_VECTOR(3 downto 0)
```

```
    );
END entrada;

ARCHITECTURE MAIN of entrada IS

    signal cur_inputs  : std_logic_vector(3 downto 0):="0000";
    signal last_inputs : std_logic_vector(3 downto 0):="0000";
    signal irq : std_logic:='0';

begin

    avl_irq <= irq;
    avl_readdata <= last_inputs;

    process(FPGA_CLK1_50)
    begin
        if(reset='0')then
            cur_inputs <= "0000";
            last_inputs <= "0000";
            irq <= '0';
        end if;
        cur_inputs <= switches;
        if(cur_inputs/=last_inputs)then
            irq <= '1';
            if(avl_read='1')then
                irq <= '0';
                last_inputs <= cur_inputs;
            end if;
        end if;
    end process;
end MAIN;
```

El programa tiene el valor de los interruptores de la placa inicializados a cero. Si detecta que hay un cambio, envía una interrupción, y se queda a la espera de que el HPS le informe de que la ha recibido y que ha hecho la operación pertinente. Es entonces cuando la FPGA actualiza los valores, una vez que ha recibido una confirmación. Este módulo será convertido a IP Core y formará parte de un sistema.

8.5.5.2. Diseño del sistema

Ahora hay que diseñar un sistema con el "Platform Designer". Como se desea utilizar el módulo anterior como un IP Core, será necesario crear un "nuevo componente".

En el editor de componentes habrá que darle un nombre al nuevo elemento; se puede añadir también otro tipo de metadatos, como autoría, versión, documentación, etc. En el apartado de archivos, habrá que seleccionar el código fuente, es decir, el fichero VHDL del módulo. Pero la parte más interesante llega en los apartados de señales e interfaces.

Component Editor - entrada_hw.tcl*

File Templates Beta View

Component Type | Block Symbol | Files | Parameters | Signals | Interfaces

About Signals

Name	Interface	Signal Type	Width	Direction
switches	**conduit_end**	**writebyteenable_n**	**4**	**input**
reset	**reset**	**reset**	**1**	**input**
avl_readdata	**avl**	**readdata**	**8**	**output**
avl_read	**avl**	**read**	**1**	**input**
avl_irq	**interrupt_sender**	**irq**	**1**	**output**
FPGA_CLK1_50	**clock_sink**	**clk**	**1**	**input**

Figura 8.20 Configuración de las señales.

Hay que asegurarse de que la señal avl_irq está asociada a una interfaz de tipo “Interrupt Sender”, y que el tipo de señal es “IRQ”.

En el apartado de interfaces hay que asegurarse de que todo esté bien conectado, pero teniendo en cuenta la referencia de la imagen superior, es bastante intuitivo, ya que el programa avisa de las conexiones erróneas o de las que faltan por realizar. Una vez realizados todos los pasos se guarda el componente y se añade al sistema. El esquema del IP Core resultante es el siguiente:

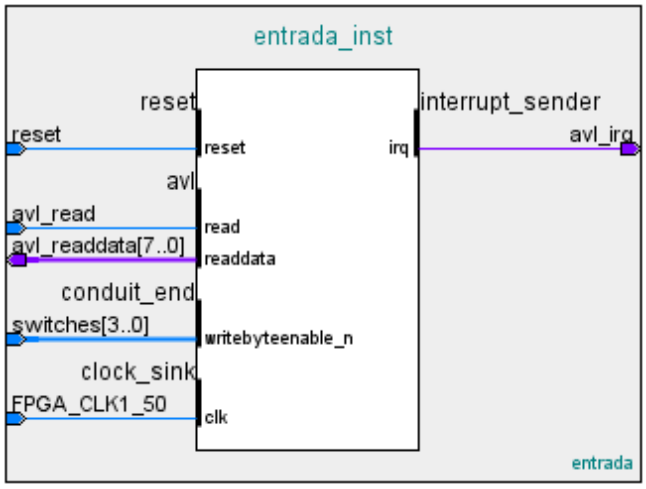


Figura 8.21 Esquema del IP Core generado.

El sistema va a ser parecido a los que se han desarrollado con anterioridad, la principal diferencia va a ser que en vez de usar un IP Core de PIO genérico, se va a utilizar un módulo propio. Por lo que quedará del siguiente modo:

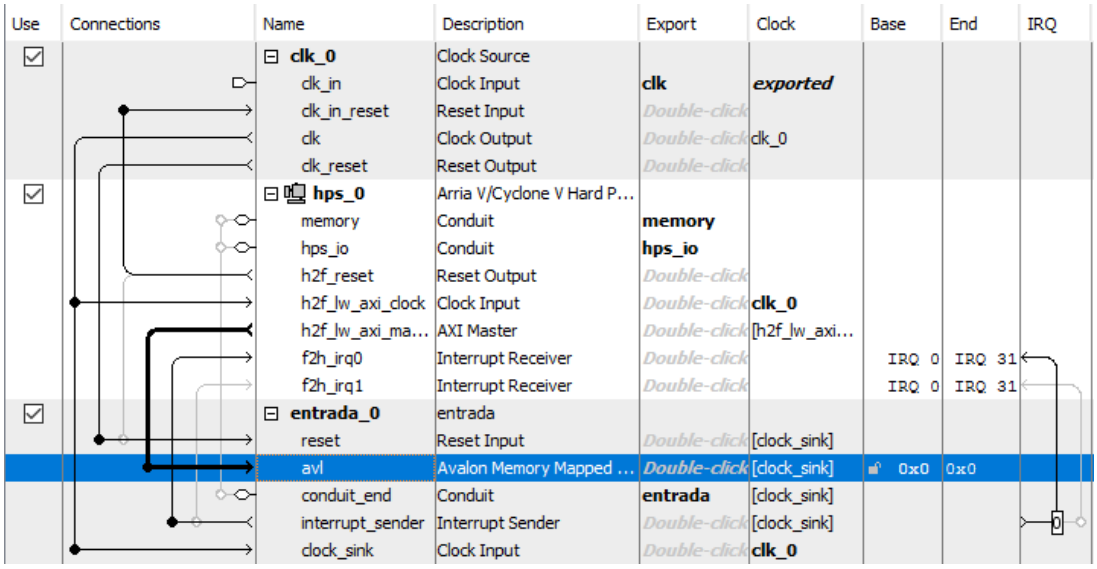


Figura 8.22 Sistema diseñado en Platform Designer.

A la hora de diseñar el sistema, hay que tener en cuenta que las interrupciones provenientes de la FPGA tienen que estar habilitadas en el HPS. Hay que marcar la opción "Enable FPGA-to-HPS Interrupts". Por otro lado, hay que recordar también que para configurar el HPS hay que basarse en el GHRD.

Una vez terminado el diseño se guarda y se genera el código, se ha elegido VHDL. El proceso de compilación es el mismo que en otras ocasiones, hay que hacer el análisis y síntesis, ejecutar los Tcl Scripts para los pines del HPS, configurar el Pin Planner para los periféricos, terminar la compilación y generar el archivo .rbf a partir del .sof. Los pasos fueron explicados en el capítulo 7.

Faltaría hacer un software para atender la rutina de interrupción, pero si se ejecuta el módulo, sí que se puede ver cómo la FPGA envía una interrupción cuando se ha cambiado el valor de los switches.

```
root@socfpga:~/driverInterrupt# insmod test_int.ko
[  112.223775] test_int: loading out-of-tree module taints kernel.
[  112.231214] test_int: La IRQ 41 va a ser registrada!
root@socfpga:~/driverInterrupt# [  119.121334] test_int: ISR
[  119.123967] test_int: ISR
[  125.054468] test_int: ISR
[  125.057097] test_int: ISR

root@socfpga:~/driverInterrupt# rmmod test_int
[  149.984559] test_int: Abandonando la captura de la IRQ 41 !
root@socfpga:~/driverInterrupt#
```

Figura 8.23 Recepción de señales a través del módulo.

Cabe destacar que ahora mismo se está en un entorno de pruebas, por lo que es interesante mostrar por pantalla datos como el arranque y parada del módulo o la interrupción que se está recibiendo, pero cuando se trate de una implementación final, no será necesario imprimir esa información por pantalla, cómo mucho podría hacerse en un log.

```
root@socfpga:~/driver_# insmod test_int.ko
[   24.680561] test_int: loading out-of-tree module taints kernel.
[   24.688890] test_int: La IRQ 41 va a ser registrada!
[   24.693859] Offset 0: 0
[   24.696305] Offset 3: 0
root@socfpga:~/driver_# [   28.085557] test_int: ISR
[   28.088179] IRQ=41
[   28.090185] input_state=1
[   34.893148] test_int: ISR
[   34.895770] IRQ=41
[   34.897776] input_state=9
[   96.758101] test_int: ISR
[   96.760722] IRQ=41
[   96.762728] input_state=1
[   97.682029] test_int: ISR
[   97.684649] IRQ=41
[   97.686655] input_state=0
[   99.606011] test_int: ISR
[   99.608632] IRQ=41
[   99.610640] input_state=4
```

Figura 8.24 Recepción de IRQ y respuesta.

La programación y configuración del módulo ha sido exitosa. Cada vez que se cambia el estado de los switches se produce una interrupción, y desde software se recoge el valor que tienen en ese momento para dejárselo en el registro compartido a la FPGA.

Las interrupciones pueden desencadenar cualquier tipo de acción que se desee. Sólo hay que interceptarla y programar una respuesta a esa interrupción.

8.6. Web-server

Un servidor web es un software almacena archivos que pueden ser accedidos a través del protocolo HTTP. Es un modelo cliente/servidor. En este caso puede resultar útil para hacer una monitorización en remoto. Se pueden recoger datos estadísticos, analizarlos, y mostrarlos de una forma fácil y ordenada.

Se va a realizar una primera aproximación a lo que pudiera ser una interfaz remota para mostrar información. Se pretende demostrar cómo es posible representar los datos obtenidos de una manera gráfica e intuitiva. Esta podría ser una futura línea de mejora del proyecto.

8.6.1. Instalación y configuración

El servidor web por excelencia es Apache, a día de hoy sigue siendo el más utilizado. Es de código abierto, fácil de instalar, robusto, potente, altamente configurable y en constante desarrollo, debido a su gran comunidad.

La instalación en un entorno Linux es muy sencilla, tan solo hay que actualizar los repositorios e instalar el paquete, en el caso de angstrom se ha hecho de la siguiente forma:

```
opkg update
opkg install apache2
```

Puede que convenga además instalar otros paquetes, por ejemplo, si se va a trabajar en el lado del servidor, habrá que instalar PHP. En el caso de querer trabajar con bases de datos, se podría instalar, por ejemplo, MySQL.

```
opkg install mysql-server
opkg install php-pear php-fpm php-dev php-zip php-curl php-xmlrpc
php-gd php-mysql php-mbstri
```

Una vez instalado es hora de configurarlo. Como casi todo en Linux, esto se hace a través de un fichero que está ubicado en el directorio "/etc/apache2/httpd.conf". Se trata de un archivo de unas 500 líneas, pero bien comentado y con ejemplos, por este motivo, habrá que buscar y cambiar lo que se cita a continuación.

Httpd es lo que se conoce en Linux como un "demonio". Un demonio es un programa que se ejecuta como proceso en segundo plano. En este caso, httpd estará a la espera de que le llegue una solicitud, y es por ello que tiene que estar "escuchando". Los puertos comunes para http son el 80, 8080 y el 443. En el fichero hay que escribir la siguiente línea:

```
Listen 80
```

Hay que especificarle también el nombre del servidor. En este caso se le ha dado la IP de la máquina y el puerto 80. De todas formas, si en el fichero de "/etc/hosts" se le ha asignado un nombre a esa dirección, se podría usar el nombre en lugar de la IP.

```
ServerName 192.168.137.100:80
```

Se pretende que desde fuera se pueda acceder al contenido del servidor, por lo que hay que darle permisos:

```
<Directory />
  AllowOverride none
  Options none
  Order allow,deny
  Allow from all
</Directory>
```

Hay que especificar también cuál va a ser la raíz del servidor. Por motivos de seguridad, no se puede permitir el acceso a cualquier ruta. Hay que poner una a partir de la cual los archivos que cuelguen de ella sean visibles.

```
DocumentRoot "/usr/share/apache2/htdocs"
<Directory "/usr/share/apache2/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Y ahora hay que establecer una página por defecto, que será la que se muestre por defecto al introducir la dirección del servidor.

```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

8.6.2. Web básica para mostrar información

Como se ha descrito al principio del apartado, por ahora sólo se pretende obtener algunos datos por remoto y demostrar cómo se puede hacer a través de web.

Para una primera prueba se ha decidido usar HTML y JavaScript, ya que están del lado del cliente. Lo que se está haciendo en esencia, simplificándolo mucho, es leer de un fichero de texto. Por un lado, está la FPGA funcionando, por otro, el HPS está leyendo los datos a través del bus que comunica las dos partes. Hay un programa en C que lee el registro en el que se almacenan los datos, y éste lo muestra por pantalla. Además, se le ha añadido la funcionalidad de que escriba en un fichero. El código JavaScript lo que hace es leer en ese fichero.

```
<html>
    <head>
        <script>
    function readTextFile(){
            var rawFile = new XMLHttpRequest();
            rawFile.open("GET", "valor.txt", false);
            rawFile.onreadystatechange = function (){
                if(rawFile.readyState === 4){
                   if(rawFile.status === 200 || rawFile.status == 0){
                        var allText = rawFile.responseText;
                        document.getElementById("valorId").innerHTML
= "El valor del ADC es: "+allText;
                    }
                }
            }
            rawFile.send(null);
        }
        var intervalID = setInterval("readTextFile()", 10);
        </script>
    </head>

    <body>
        <h1>Webserver en Angstrom:</h1>
        <p id="valorId"></p>
    </body>
</html>
```

En el código C se ha incluido la siguiente función que permite escribir en ficheros.

```
int imprimirValorFich(int valor){
    FILE *fptr;

    fptr = fopen("/usr/share/apache2/htdocs/valor2.txt", "w");
    if(fptr == NULL){
        printf("Error!");
        exit(1);
    }

    fprintf(fptr,"%d", valor);
    fclose(fptr);

    return 0;
}
```

El resultado se puede ver en la siguiente imagen:

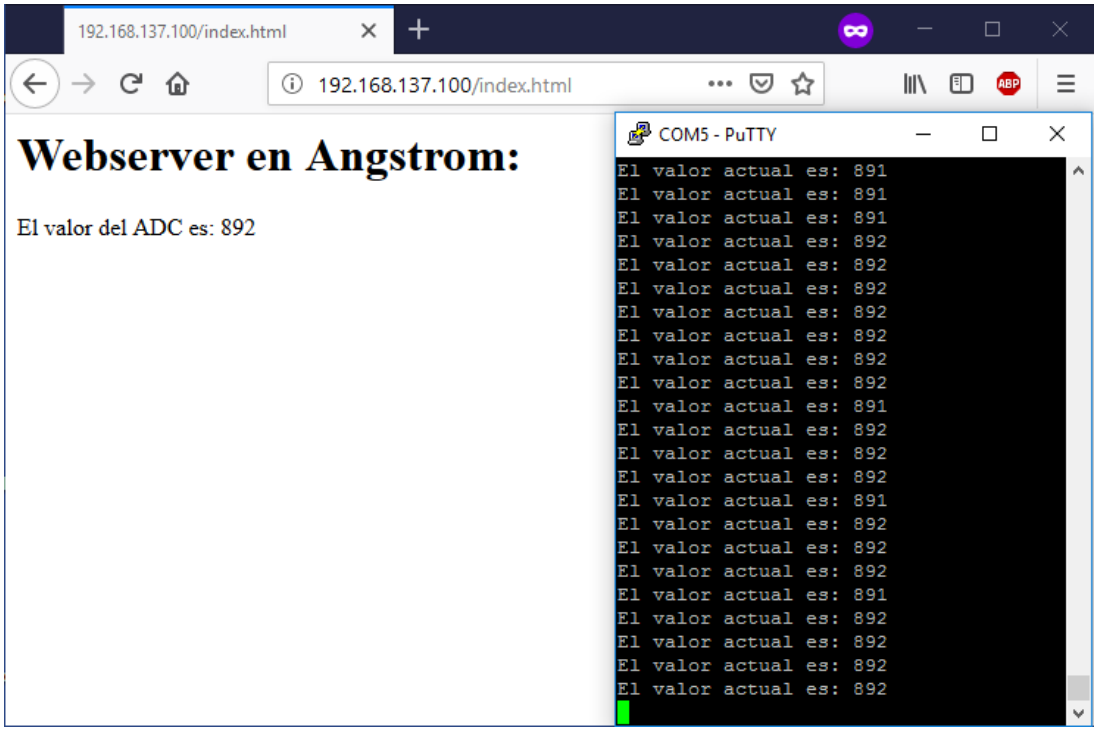


Figura 8.25 Programa en ejecución y obtención de datos vía web.

Por un lado, está la consola mostrando los resultados del ADC, y por el otro, se están mostrando los valores actualizados en un navegador web. Éste último se refresca de forma automática, sin intervención del usuario.

9

Implementación: El prototipo final

Este es el último capítulo de la implementación. Hasta ahora se ha explicado una progresión de tareas previas que eran necesarias para dar este paso final. Este capítulo consiste en la unión del aprendizaje y de los desarrollos previos, con la finalidad de implementar un prototipo final y poder conectarlo al inversor trifásico.

9.1. Introducción

En anteriores capítulos se ha visto como se han hecho diferentes implementaciones. Se han hecho pruebas para conocer la placa, su funcionamiento y las opciones que ofrece, se han diseñado *drivers* e implementado protocolos de comunicación, se ha dado el salto a Linux RT y se ha comprobado su rendimiento, se han creado sistemas SoC-FPGA… Ahora llega el momento de juntar todo ese conocimiento adquirido y aplicarlo al diseño, implementación y verificación del controlador.

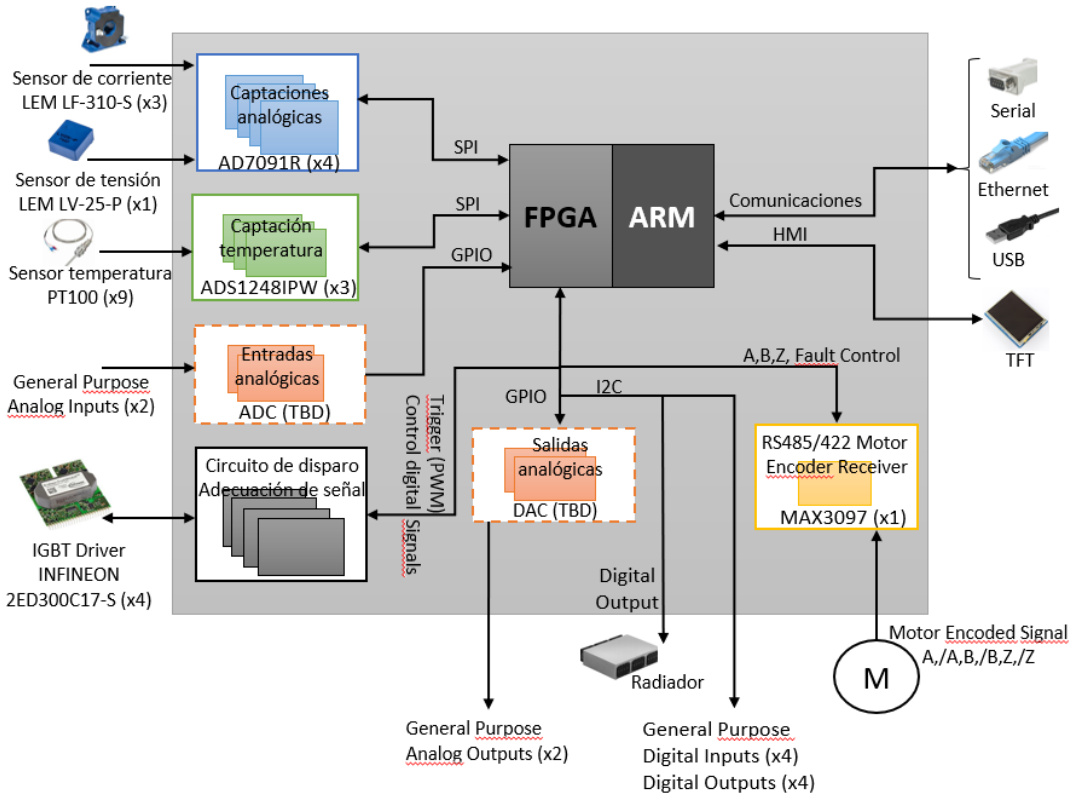


Figura 9.1 Arquitectura del controlador en SoC-FPGA.

En la figura anterior se define la arquitectura del controlador. Va a ser un sistema SoC-FPGA y estará conectado a múltiples periféricos. Por un lado, estará conectado a sensores de corriente, tensión y temperatura, que garantizarán la seguridad del hardware, y, por otro lado, estará la generación de señales de PWM para el control del inversor. Disponer de una arquitectura SoC-FPGA permitirá configurar y monitorizar el controlador por software, sin añadir carga de trabajo y dejando que la FPGA se encargue de controlar el inversor. Además, conociendo ambas arquitecturas, se puede delegar en el µP tareas u operaciones matemáticas que serían complejas de realizar en una FPGA.

Estudiando la arquitectura interna del ARM-A9 se puede comprobar que integra 8 unidades funcionales, entre las que se encuentra un divisor. Realizar este tipo de operaciones por software será menos costoso que en la FPGA.

9.2. Modulación en ancho de pulso (PWM)

Para empezar con la fase de pruebas de este proyecto, se ha diseñado un sistema en VHDL, compuesto de varios módulos, para enviar una señal PWM y su inversa a un equipo que contiene dos IGBTs.

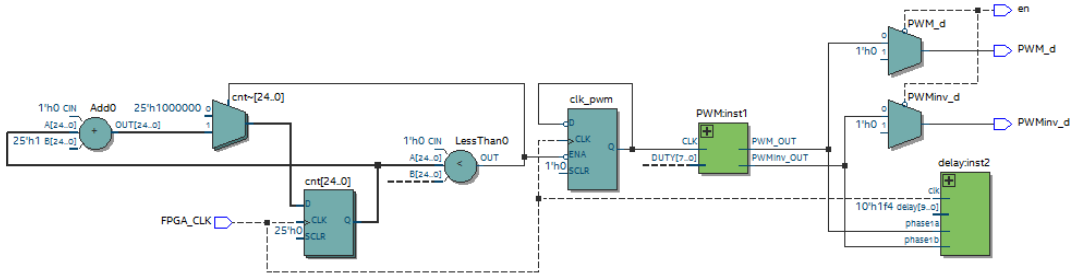


Figura 7.1 Esquema parcial del diseño del generador de pulsos PWM.

En la figura superior se ha omitido el módulo del HPS, que es el que representa la unión de la FPGA con el HPS.

El modelado del hardware se ha simulado con la herramienta ModelSim, y se le ha añadido un módulo que permite retrasar la señal cuando se dé la condición de un flanco de subida. Además, la frecuencia y el ciclo de trabajo son configurables.

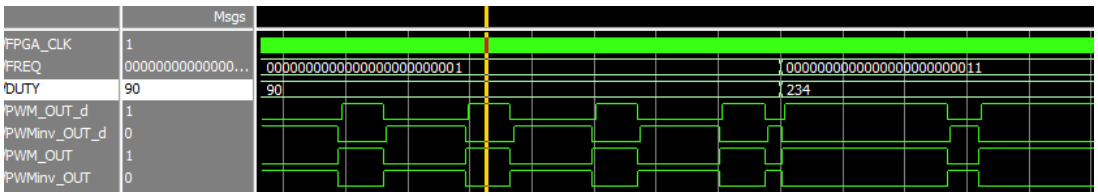


Figura 9.2 Representación gráfica de las señales generadas en VHDL.

En la imagen se puede observar la diferencia entre las señales de PWM con y sin retardo.

Posteriormente, este sistema se ha pasado a una arquitectura SoC. De forma que se puedan configurar los parámetros de frecuencia y ciclo de trabajo a través de una aplicación programada en C. Esto facilita su uso por parte del usuario, ya que la aplicación calcula el valor correcto para los parámetros solicitados. Se podrá configurar la frecuencia introduciendo su valor en Hercios, tendrá que ser un valor numérico por debajo de 100.000. Por otra parte, el *duty-cycle* será un número comprendido entre 0 y 100.

Se ha utilizado un vector de 8 bits para definir el ciclo de trabajo. Para la aplicación actual es una precisión más que suficiente, y permitiría incluso definir un ciclo de trabajo con posición decimal. En el caso de reducir este vector en un bit, se podría llegar a utilizar frecuencias más altas.

La aplicación diseñada de momento es simple, en esta imagen se puede observar en ejecución:

```
root@socfpga:~# ./pwm_simple
  Menu del PWM:

****************************************
Seleccione la opcion que quiere utilizar:
                              -1) Cambiar frecuencia.
                              -2) Cambiar Duty-Cycle.
                              -3) Salir de la aplicacion.

                              Elija una opcion:
 1
Esta es la seleccion: 1

 Introduzca la frecuencia en Hz.
10000
2
Esta es la seleccion: 2

 Introduzca el duty-cycle.
50
```

Figura 9.3 Consola para el cambio de parámetros de la señal PWM.

Si se conecta un osciloscopio a las salidas de la placa de desarrollo, se pueden ver las dos señales complementadas. El osciloscopio proporciona también medidas sobre la frecuencia, y se puede observar cómo, pese a no ser completamente exactas, la aproximación es lo suficientemente buena. En la siguiente imagen se ha generado una señal PWM de 100Hz y del 35% de *duty-cycle*, (el 35% es el tiempo en el que el IGBT está activo, y se usa lógica negada). Observando los valores que da el osciloscopio, aparecen unos resultados de 100,05Hz de frecuencia y 35,2% de ciclo de trabajo, es por eso que se dice que, aun no pudiendo conseguir siempre valores exactos, la aproximación es buena.

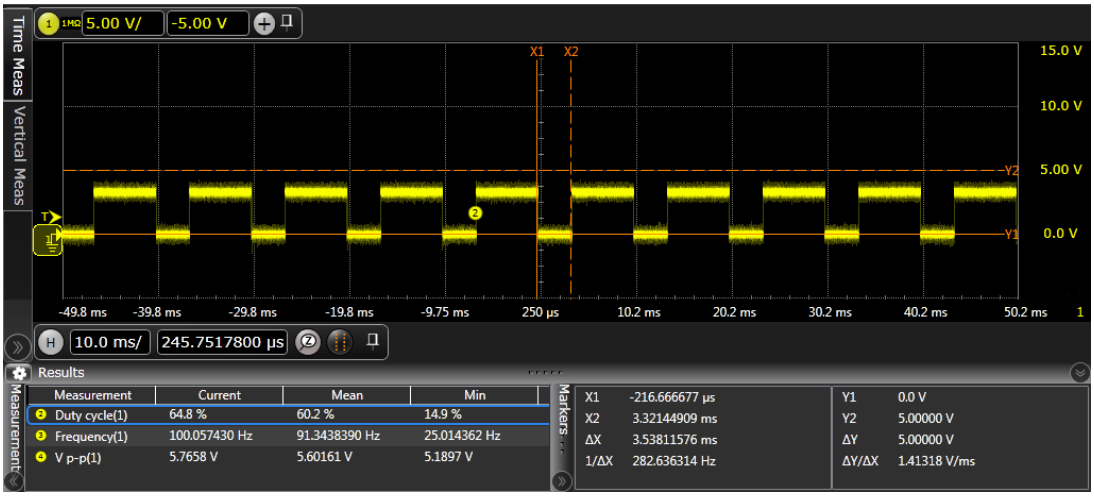


Figura 9.4 Captura de una señal PWM.

Hay que añadir que una FPGA no es buena para hacer operaciones como las divisiones, pero la arquitectura del microprocesador sí que tiene unidades funcionales dedicadas a este tipo de operaciones. Dado que hacer divisiones en el procesador tiene menor coste computacional, es preferible hacerlas ahí antes que en la FPGA. Así que para hallar los valores

que hay que aplicar a las señales, para que se trabaje con los parámetros solicitados por el usuario, se efectuarán las operaciones en el ARM, y se enviarán a la FPGA a través del bus Lightweight AXI.

Para trabajar a una frecuencia determinada, la FPGA tiene que saber cuántos ciclos de reloj principal debe esperar. Lo que se está haciendo en VHDL es generar una señal de reloj más lenta para pasársela al módulo de PWM. Además, hay que tener en cuenta que el módulo de PWM también va a perder más ciclos para elaborar el *duty-cycle*, y al haberle dado una precisión de 8 bits, va a hacerlo en base a 256 pasos extra. La fórmula matemática a utilizar es la siguiente:

$$\frac{CLK_{FPGA}}{2 * \left(Res_{Duty}\right) * Freq} = nº\ ciclos\ a\ esperar$$

$$\frac{50MHz}{2 * (256) * Freq} = \frac{50MHz}{512 * Freq} = nº\ ciclos\ a\ esperar$$

Siendo "Freq" la frecuencia solicitada por el usuario en Hz. Para que la operación dé el valor deseado todas las unidades se traducirán a Hercios.

En el caso del *Duty-Cycle* es sencillo. Se tienen 256 valores y se quiere traducir a una escala que tiene de rango [0-100], así que la fórmula utilizada es la siguiente:

$$\frac{Duty * Res_{Duty}}{nº\ valores} = nº\ ciclos\ con\ la\ señal\ a\ '1'$$

$$\frac{Duty * 256}{100} = nº\ ciclos\ con\ la\ señal\ a\ '1'$$

Siendo *Duty* el valor del ciclo de trabajo deseado por el usuario.

9.3. Modulación en ancho de pulso sinusoidal (SPWM)

Anteriormente ya ha aparecido el concepto de PWM. La modulación en ancho de pulso permite variar los periodos de encendido y apagado de los componentes del inversor, en este caso, de los IGBTs. Modelando una señal sinusoidal en ancho de pulso, se puede conseguir transformar una entrada de corriente continua en una salida en alterna, y si además se hace esto con tres sinusoides con un desplazamiento de 120º y 240º respecto al seno, se puede obtener una salida en alterna y trifásica. Es uno de los métodos más populares.

El modelado de una sinusoide en PWM consiste en tres tipos de ondas, la sinusoidal, la triangular y la cuadrada resultante. La referencia la da la señal sinusoidal, y se interpola con una señal triangular denominada portadora. Comparando el valor de los puntos de la señal triangular con los de la sinusoidal se puede codificar el PWM, ya que este variará en función de la amplitud de la sinusoide. La portadora sirve para evaluar la posición y medir la amplitud, se toma como una referencia para el muestreo.

Para el desarrollo se ha obtenido un IP Core de código fuente abierto, cuya función es la de actuar como un generador de señales, y se han hecho unas primeras aproximaciones. Este módulo permitía representar numéricamente una señal triangular y sinusoidal, pero tenía una resolución muy escasa, sólo llegaba a 8 bits, por lo que la señal de PWM que se obtenía no era demasiado precisa.

Se realizó un breve estudio para comprender el funcionamiento interno del algoritmo y se realizaron mejoras sobre el mismo. Ahora, además de tener una resolución de 16 bits, se toman en cuenta otros factores para adaptar la señal a la frecuencia deseada, sin perder tiempos, y que el crecimiento y decrecimiento de los valores que componen la señal se produzca de la forma más lineal posible.

El algoritmo usa una "*Lookup Table*" (LUT) para generar la señal sinusoidal y la triangular. Esto consiste en una tabla numérica con los valores que irá tomando la señal para tomar la forma deseada, cuantos más valores tenga, más precisión se puede conseguir. A través de un índice se asigna el valor en cada instante de tiempo. Como se desea obtener tres señales sinusoidales desplazadas entre sí 120º, se usan tres índices. Si se contara con un conversor de digital a analógico (DAC), se le podría enviar los datos en binario y capturar la función que genera, pero en su lugar, se han obtenido los datos de la simulación y se ha exportado a un fichero de texto. A través de una hoja de cálculo se puede hacer una gráfica que muestra el resultado obtenido. El resultado se puede ver en la siguiente figura:

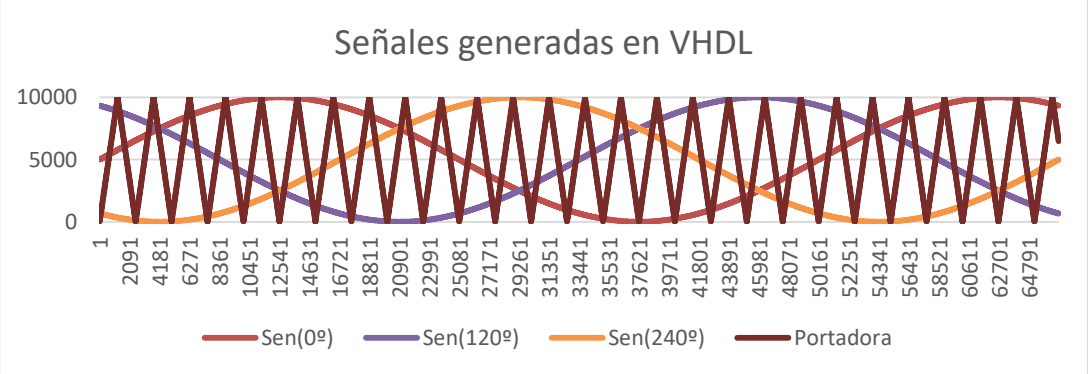


Figura 9.5 Representación gráfica de la simulación del SPWM.

En la siguiente gráfica se compara la señal sinusoidal con la PWM generada. Se puede apreciar cómo cambia el valor del PWM. Estos valores se han obtenido comparando la portadora con el seno generado por la LUT. Se observa como el ciclo de trabajo disminuye cuando el seno toma los valores superiores y como aumenta cuando toma los valores inferiores.

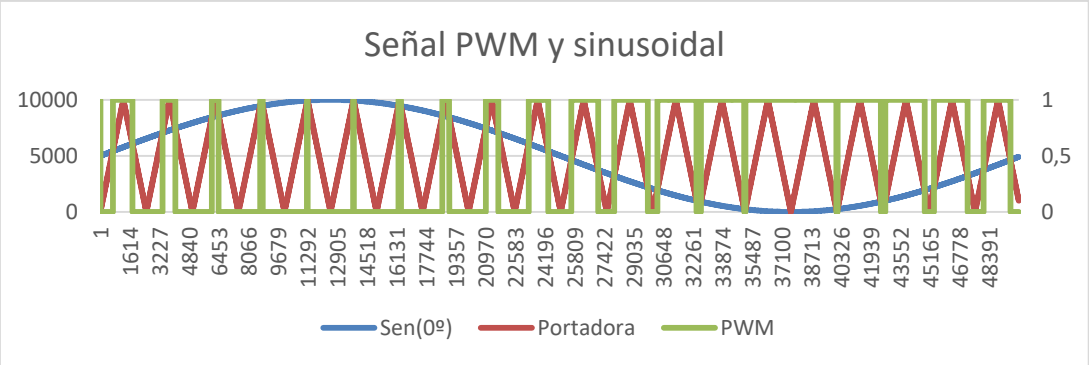


Figura 9.6 Generación de PWM por comparación en una de las fases.

A la hora de pensar en el controlador, hay que tener en cuenta que las parejas de IGBTs no pueden bajo ningún concepto estar en estado de conducción al mismo tiempo, ya que se produciría un cortocircuito. Para evitar esto hay que añadir un retardo cuando una señal tenga un flanco de subida. Habitualmente se hace por hardware, y la propia controladora trae consigo unos condensadores para añadir ese tiempo muerto. De todas formas, se ha implementado también en VHDL. En caso de querer usarlo bastaría sólo con añadir el módulo que se ha diseñado.

Tras realizar la implementación y comprobar los resultados en la simulación, se ha decidido analizar el resultado de la señal generada a través de un osciloscopio. Para ello se ha construido un filtro pasabajo RC, que convierte la señal cuadrada de PWM en sinusoidal.

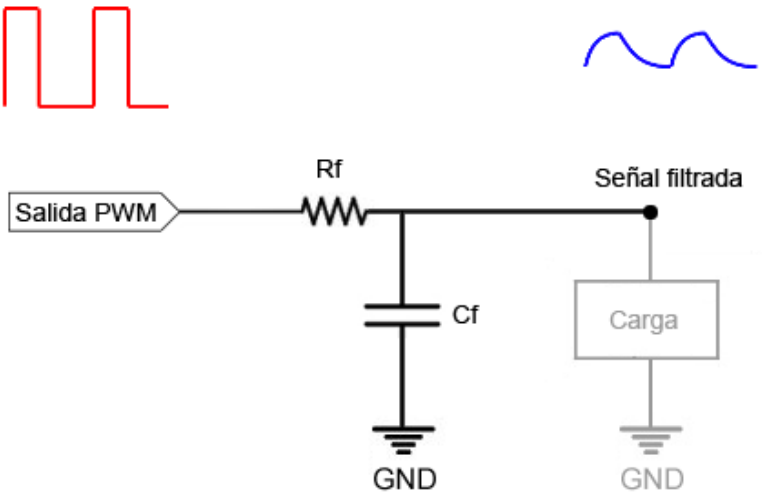


Figura 9.7 Esquema de un filtro RC.

Los valores de Rf y Cf son de 1kΩ y de 100nF respectivamente, sirve para filtrar a una frecuencia de 1,5kHz. El resultado que se ha obtenido de este experimento se observa a continuación:

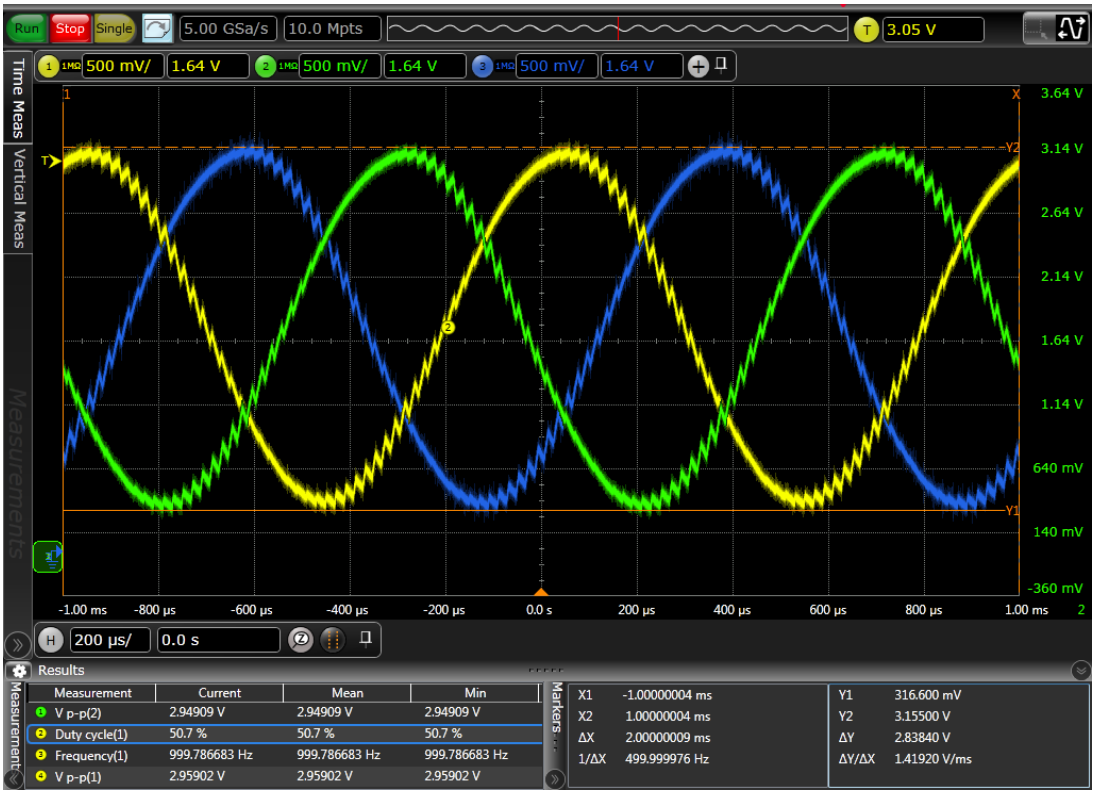


Figura 9.8 Señales PWM filtradas.

En la imagen superior se pueden ver las tres señales sinusoidales. Para obtener esta salida se han introducido en la aplicación los parámetros deseados para generar una señal PWM, y ésta ha sido filtrada.

Por otra parte, y añadiendo más complejidad al módulo, se desea que además de poder variar la frecuencia se pueda también variar la amplitud de la señal. Para que esto sea posible hay que establecer un índice de modulación.

9.3.1. Amplitud de la señal variable

Para conseguir modificar la amplitud de la señal se toman como referencia la señal triangular, que no varía, y la sinusoidal. Resulta lógico pensar que multiplicando los valores que va tomando la señal sinusoidal en cada momento por un número comprendido entre 0 y 1, se estará haciendo la señal de PWM más pequeña. Esto es cierto, pero hay más factores que influyen y se exponen a continuación.

En primer lugar, hay que tener en cuenta que se está trabajando con una FPGA. Realizar divisiones va a ser costoso, y utilizar números decimales no es fácil de por sí. Lo que se ha tenido que hacer es acondicionar el algoritmo anterior y hacer una serie de conversiones para poder hacer multiplicaciones.

En una FPGA todo se puede reducir a vectores de bits. Pero depende de la interpretación que se haga de un vector, puede tener un significado u otro. En este momento se va a tratar

el tema de la representación numérica, ya que para hacer este paso se ha recurrido al sistema numérico de coma fija.

Un sistema numérico es un conjunto de símbolos y reglas que se utilizan para representar datos numéricos o cantidades. Se caracterizan por su base, que indica el número de símbolos distintos que utiliza, y que además es el coeficiente que determina cuál es el valor de cada símbolo dependiendo de la posición que ocupe. Estas cantidades se caracterizan por tener dígitos enteros y fraccionarios.

Mediante la representación en coma fija se hace posible la multiplicación por un número fraccional y además se puede mantener una buena precisión numérica. En la versión gratuita de Quartus no hay ninguna librería para usar este sistema de notación, así que se ha recurrido a la librería en coma fija de David Bishop [68], que está contrastada, documentada y es de código abierto y libre. Además, está incluida en el estándar VHDL-2008 [69].

La amplitud deseada es definida por el usuario, que puede introducir un valor comprendido entre el 0 y el 100%. Los cálculos para hallar el índice de modulación se hacen en el ARM, que le ahorra ese trabajo a la FPGA. La fórmula matemática es la siguiente:

$$Amplitud(\%) = \frac{Amplitud}{100} * Factor\ de\ Conversión$$

Con este cambio se consigue modificar la amplitud, sin embargo, como se verá a continuación, aquí no termina el proceso, y es que la señal debe de estar siempre centrada, es decir, tiene que haber un offset y este tiene que ser del 50%.

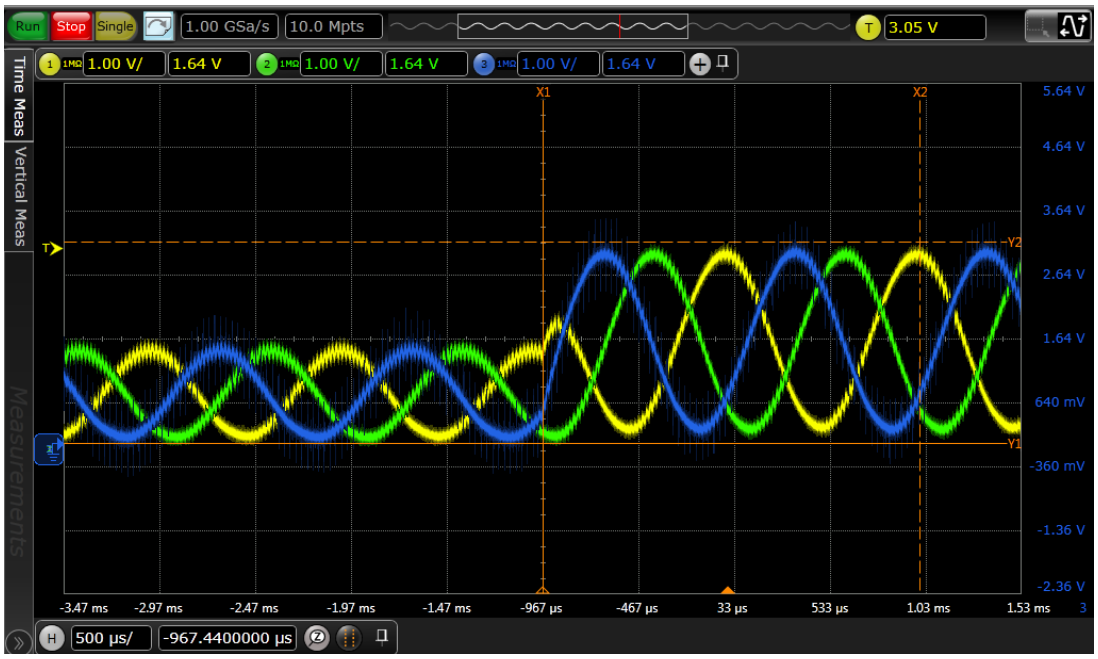


Figura 9.9 Cambio de amplitud sin offset.

En la imagen anterior se observa cómo se ha capturado un cambio en la señal sinusoidal, pasando de un 50% de amplitud a un 100%. En este caso la señal va siempre de 0V a V_{max}, estando V_{max} acotado por el índice de modulación. Para un sistema trifásico, la señal tiene que estar centrada, ya que la suma de los tres voltajes en un instante de tiempo cualquiera

tiene que ser igual a cero. Hay que añadir que, en la figura anterior, se aprecia un leve desfase. Su corrección se verá reflejada en la siguiente imagen.

Observados estos problemas, el siguiente paso es conseguir que la señal quede centrada y corregir el desfase. Para hacer esto ha habido que aplicar ciertos cambios en el módulo de VHDL. El más significativo ha sido el cambio de notación; se ha pasado de usar la coma fija sin signo a usar coma fija con signo. Esto es debido a que se van a hacer nuevas operaciones matemáticas y el resultado puede dar también negativo.

Lo primero que se ha hecho ha sido definir el punto medio. Dado que para hacer la señal sinusoidal con la LUT se han utilizado 10.000 puntos, se ha establecido en 5.000. Y la fórmula a aplicar es la siguiente:

$$Pos_{Actual} = offset - (Mod_{index} * (Pos - offset))$$

Dado que en este caso el offset es fijo y es siempre 5.000, salvo que se cambie la LUT, la función queda del siguiente modo:

$$Pos_{Actual} = 5000 - (Mod_{index} * (Pos - 5000))$$

Se ha vuelto a probar el módulo en el osciloscopio filtrando la señal de PWM, y esta vez queda de la siguiente forma:

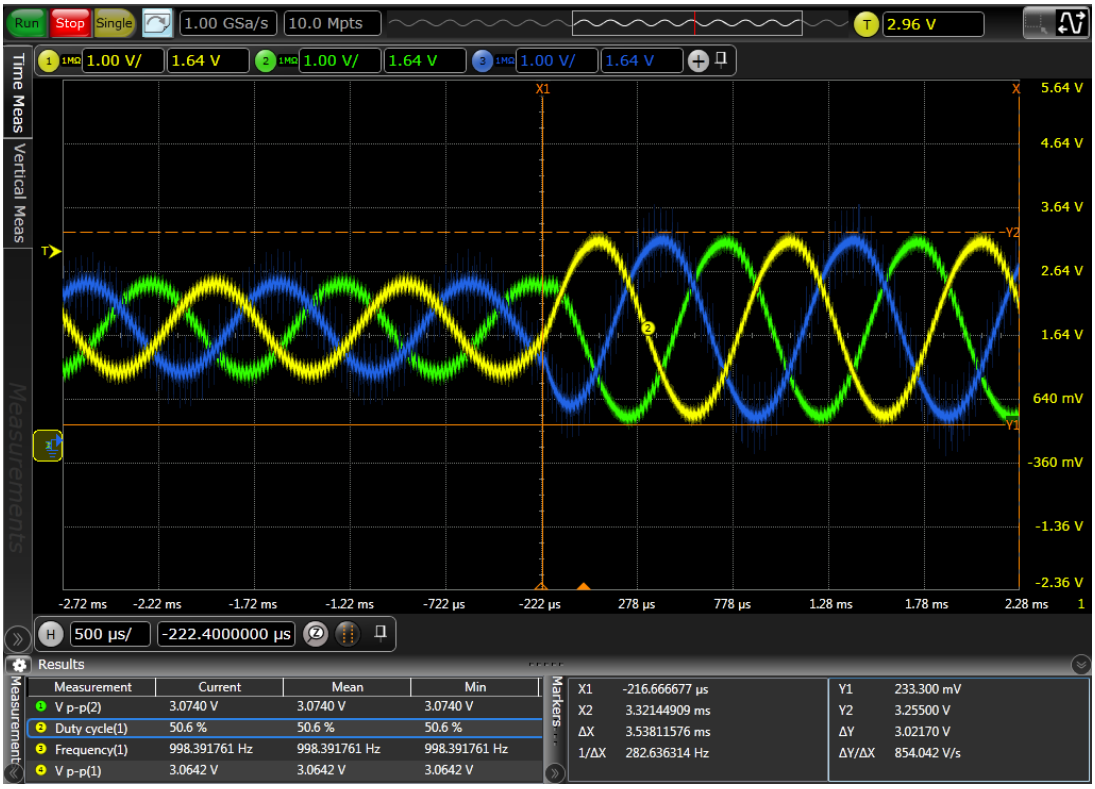


Figura 9.10 Cambio de amplitud con offset fijo al 50%.

La señal trifásica ya está lista para el inversor. Se ha corregido el offset y el desfase ahora es también el correcto, entre cada una de las fases hay 120º de diferencia. Sin embargo, durante el análisis de la señal, se determinó que se podría mejorar la salida de los pulsos

generados por la FPGA, por lo que se va a realizar un acondicionamiento de la señal, se explicará más adelante.

9.4. Control de tensión y corriente

En el capítulo de implementación se trataron dos temas que cobran ahora su relevancia y significado. Por un lado, se trabajó de distintas formas con los ADC, y, por otro lado, se hizo un protocolo SPI y un *driver* para el ADC de Analog Devices AD7091. En aquellas pruebas se utilizó una placa de evaluación, ahora estos mismos ADC están integrados en una tarjeta interfaz.

El cometido de esta tarjeta es la de adaptar magnitudes eléctricas y físicas a la escala de la FPGA, además de integrar otros componentes electrónicos.

9.4.1. Funcionamiento

Hay que controlar en todo momento el estado de la tensión y de las corrientes. Sobrepasar los límites de funcionamiento podría poner en peligro el inversor. Se ha decidido establecer la lógica en la FPGA, ya que es la que actuaría de una forma más rápida en caso de emergencia. Por otro lado, la configuración y monitorización de los valores se hará a través del HPS.

A continuación, se va a exponer el funcionamiento lógico del controlador, es decir, se van a explicar los diferentes estados y qué es lo que se hace en cada uno de ellos.

- Estado 0. En este estado el funcionamiento del inversor es el correcto. Las corrientes y la tensión no representan ningún peligro para la vida del inversor.
- Estado 1 (Aviso). Los niveles no suponen ningún riesgo, pero están algo por encima de lo normal. Si el estado anterior era el 0, el chopper no se activará, si por el contrario, el estado anterior era el 2, entonces el chopper sí que estará activo hasta que los niveles de corriente vuelvan a estar por debajo de la franja de aviso. En este estado hay histéresis.
- Estado 2 (Alarma). Los niveles empiezan a ser preocupantes, están por encima del nivel de aviso. En este estado entra en funcionamiento el chopper. Se trata de un cuarto IGBT que entra en actividad para estabilizar el sistema.
- Estado 3 (Emergencia). En este estado el sistema corre peligro. Los niveles de tensión o corriente son demasiado altos. Por seguridad se apagan todos los IGBTs a excepción del chopper, que sigue funcionando para que el sistema vuelva a ser estable.

A la hora de implementar el módulo, hay que tener en cuenta que la tarjeta interfaz ya está diseñada, por lo que el código tiene que ser adaptado a esa tarjeta. El diseñador de esa tarjeta ha proporcionado los esquemas, y en ellos aparece una tabla para la correcta lectura de los datos recibidos por el ADC. Un dato importante es que existe un offset de 2,5V.

LEM LF310-S. Transductor de corriente.				
Ipri (A)	Isec (A)	VLOAD (V)	Vadc (V)	Digital (HEX)
+500 A	+250 mA	+5.025 V	>+5 V	0xFFF
+497.51 A	+248.75622 mA	+5V	+5 V	0xFFF
+300 A	+150 mA	+3.015 V	+4.0075 V	0xCD2
0 A	0 mA	0 V	+2.5 V	0x7FF
-300 A	-150 mA	-3.015 V	+0.9925 V	0x32D
-497.51 A	-248.75622 mA	-5 V	0 V	0x000
-500 A	-250 mA	-5.025 V	NO DISPONIBLE	

Tabla 9.1. Valores de referencia del transductor de corriente.

LEM LV25-600/SP2. Transductor de tensión.					
Vin(V)	Ipri (A)	Isec (A)	VLOAD (V)	Vadc (V)	Digital (HEX)
+923.07692 V	+15.38462 mA	+38.46154 mA	+5 V	+5 V	0xFFF
+900 V	+15 mA	+37.5 mA	+4.875V	+4.9375 V	0xFCC
+600 V	+10 mA	+25 mA	+3.25 V	+4.125 V	0xD33
0 V	0 mA	0 mA	0 V	+2.5 V	0x7FF
-600 V	-10 mA	-25 mA	-3.25 V	+0.875 V	0x2CC
-900	-15 mA	-37.5 mA	-4.875 V	0.0625 V	0x033
-923.07692 V	-15.38462 mA	-38.46154 mA	-5 V	0 V	0x000

Tabla 9.2. Valores de referencia del transductor de voltaje.

La aplicación tiene que estar preparada para que cualquier persona pueda configurarla sin tener que memorizar números hexadecimales ni tener que hacer cuentas. Es por ello que el usuario introducirá un número decimal para delimitar los valores, y el software hará los cálculos para adaptarlo al hardware y para hallar su complementario. Ya que el umbral va entre un número positivo y otro negativo. Las fórmulas aplicadas han sido las siguientes:

$$NUMBITS = (2^{12}) - 1 = 2047$$

$$Volt_{Hex} = \left(\frac{Volt_{Dec} * NUMBITS}{MaxV_{ref}}\right) + 2048$$

$$Amp_{Hex} = \left(\frac{Amp_{Dec} * NUMBITS}{MaxA_{ref}}\right) + 2048$$

Donde $MaxV_{ref}$ y $MaxA_{ref}$ equivalen a los valores máximos que interpreta el ADC, es decir 923,07692 y 497,51 respectivamente.

Tras hacer estas operaciones, el valor hexadecimal se almacena en un registro al que accede la FPGA. Ésta hallará su complementario para tener también el mínimo, y hará comparaciones entre el valor almacenado y el valor recogido de los ADCs a través del bus SPI. De este modo se puede aplicar la lógica de estados.

En el diseño de este módulo se han incluido unas señales internas que facilitarán la activación y desactivación de las señales de SPWM y PWM. Es decir, estas señales permitirán que los módulos de PWM y SPWM entren en funcionamiento o no.

9.4.2. Comprobación

El módulo desarrollado se ha probado con la placa interfaz. Para ello se ha usado en primera instancia una fuente de alimentación, un osciloscopio y la monitorización a través de la interfaz gráfica de la aplicación software. El objetivo es comprobar que los valores introducidos de forma analógica se corresponden con la digitalización. Por otro lado, a través del osciloscopio, se pretende comprobar que se realiza la histéresis de forma correcta, ya que se puede observar como las señales de parada o de chopper se activan, y en qué momento lo hacen.

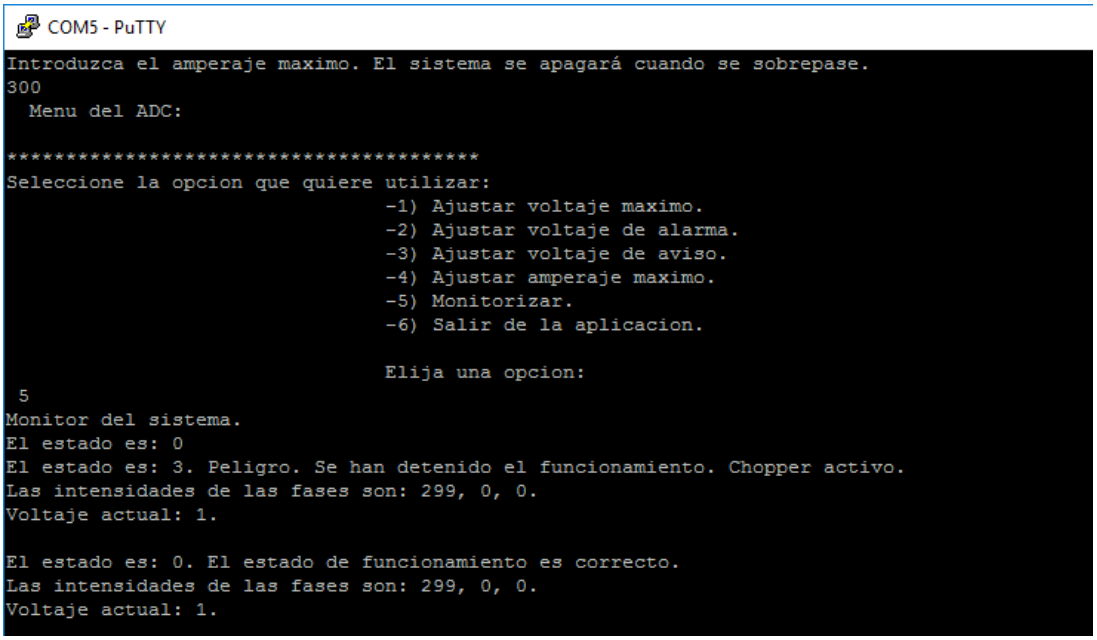


Figura 9.11 Parada de IGBTs y activación del chopper por sobreintensidad.

En la imagen superior se puede ver el menú de ajuste para el ADC. Se ha establecido el amperaje máximo en 300 A. En pantalla se muestra que el estado inicial es 0, es decir, todos los niveles están en un umbral correcto. Se ha alimentado el ADC para una de las entradas del sensor de corriente. Se le está aplicando un voltaje de forma manual, con una fuente de tensión, sin el transductor. Se observa que cuando se supera el nivel establecido se pasa directamente al estado 3.

Con que uno de los sensores detecte un valor por encima de ese umbral, el chopper se activará y los IGBTs se desconectarán. Se ha examinado los niveles de las líneas para la señal de entrada del chopper y para la de desactivación de los IGBTs y estas hacen la transición de '0' a '1'.

La siguiente prueba consiste en poner a prueba el control de tensión. Para ello se ha hecho el mismo proceso. En primer lugar, se han introducido los valores de referencia. El aviso lo

dará a partir de 600 V, la alarma saltará a partir de 750 V y el estado de emergencia se producirá cuando se llegue a los 900 V. En este apartado hay que observar también las salidas de activación del chopper y de apagado de IGBTs. Se comprueba con el osciloscopio que aparece el estado de histéresis, y de lo mismo se informa por la consola.

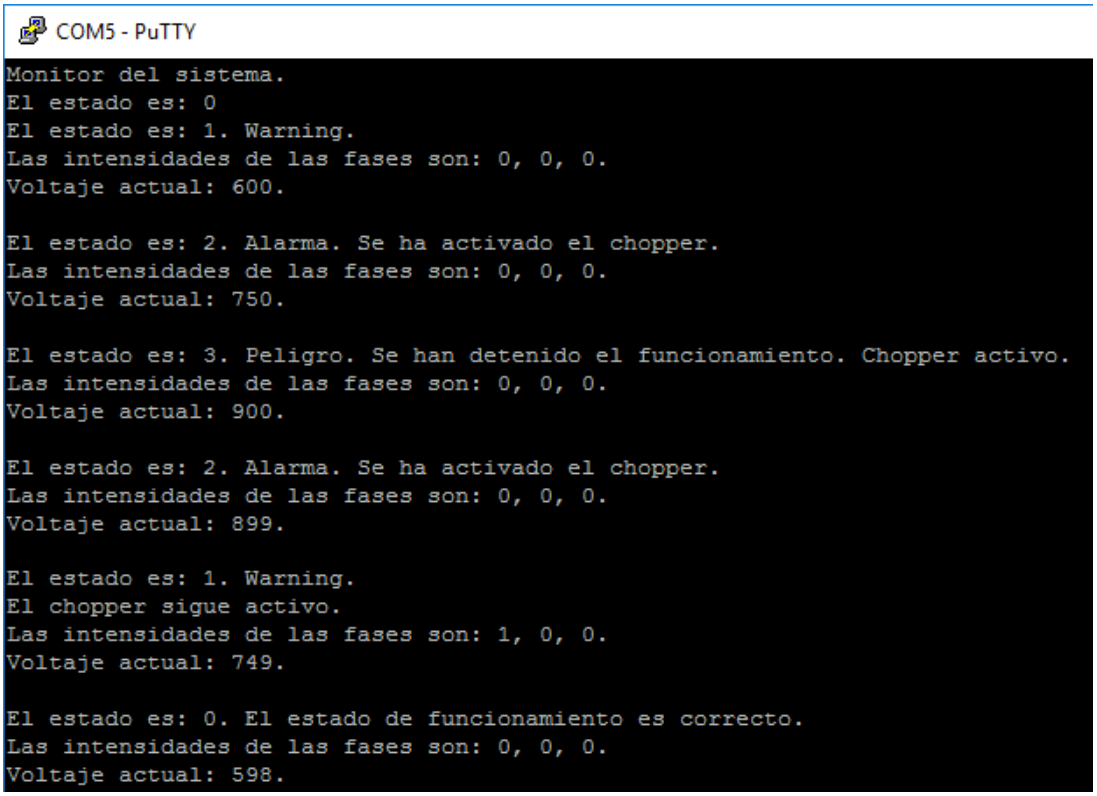


Figura 9.12 Parada de IGBTs y activación del chopper por sobretensión.

Los resultados obtenidos son satisfactorios. Se ha puesto a prueba el funcionamiento del módulo y cumple con los requisitos.

Hasta ahora se ha visto como se ha implementado un PWM, un SPWM y un sistema de supervisión y actuación sobre la tensión y la corriente. El siguiente paso es juntar estos módulos y realizar una aplicación funcional que permita controlar de manera eficaz el inversor.

9.5. Aplicación

En los anteriores apartados se ha mostrado el diseño y verificación de las diferentes piezas que compondrán la aplicación final. Llegado este momento, es hora de juntar todo el trabajo realizado en una única entidad, de forma que se obtenga el primer prototipo funcional para controlar el inversor trifásico.

A la hora de realizar este trabajo se ha tenido en cuenta el esquemático que se diseñó para la arquitectura de µC, de forma que las salidas de las señales de la FPGA tengan el mismo

nombre que en el caso de su predecesor. Esto se hace para garantizar un orden y una coherencia.

Los módulos que se han evaluado anteriormente estaban diseñados sobre una arquitectura FPGA-SoC. Ahora hay que diseñar un nuevo sistema para realizar los intercambios de datos entre FPGA y HPS. Quitando el HPS, la estructura es la siguiente:

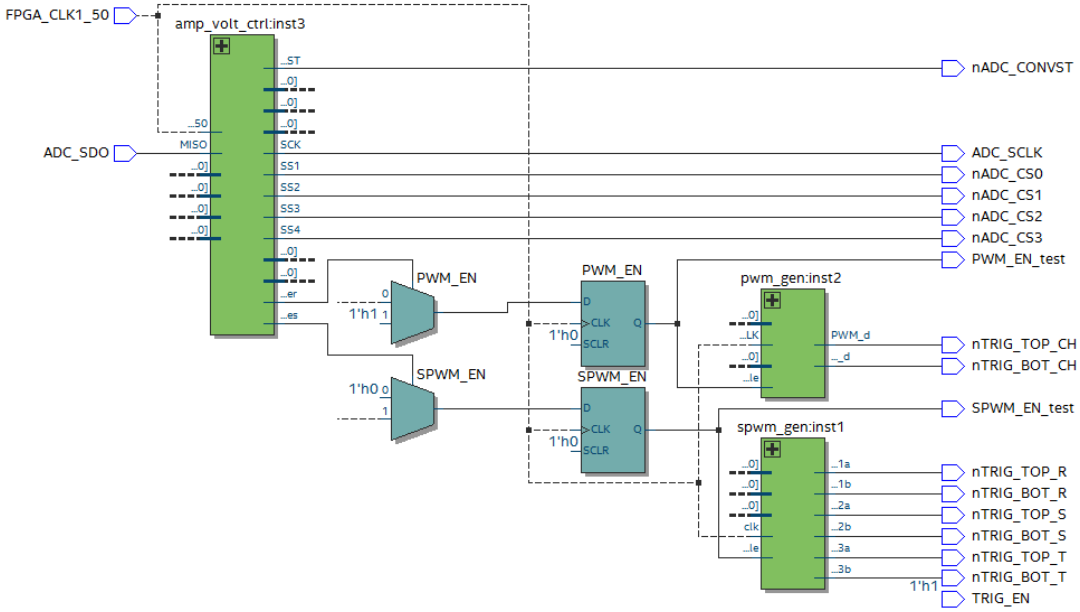


Figura 9.13 Esquema del diseño del prototipo final.

En esta imagen se observa parte de la arquitectura que se ha diseñado. Por un lado, está el ADC que recibe los datos acerca de los valores de tensión y corriente. Ese mismo módulo se configura a través del HPS y aplica la lógica para asegurar el correcto funcionamiento del inversor y evitar posibles daños por sobretensión o sobrecorriente. Por otro lado, el controlador también permite activar o desactivar la generación de señales PWM y SPWM. Para esto se tiene en cuenta el estado del inversor y la acción del usuario. Por ejemplo, el usuario podrá encender el chopper si así lo desea, pero no podrá apagarlo si está en estado de alarma o emergencia. A la derecha se pueden ver las salidas que genera el controlador.

En cuanto a la aplicación software, se ha implementado una primera versión, pero se están teniendo en cuenta bastantes mejoras a corto plazo. El trabajo sobre este diseño no acabará con el TFM, sino que continuará, esta vez en una fase de mejora. Durante la elaboración de este proyecto ha habido también una labor investigadora, y las vías están abiertas para continuar por ese camino. De esta forma, una de las mejoras a aplicar sería la programación de la aplicación en tiempo real y la división de ésta en *threads*, de modo que se puede dedicar un hilo a la configuración y otro hilo a la monitorización.

La aplicación actual permite la configuración de la señal SPWM en frecuencia de la portadora, frecuencia de la señal que se quiere generar y amplitud, además de su encendido y apagado. También permite generar una señal PWM para la cuarta pareja de IGBTs, el chopper, de forma que se pueda configurar su frecuencia y ciclo de trabajo. Por último, se pueden establecer manualmente los valores de tensión a los cuales saltará la advertencia, la

alarma o el estado de emergencia y el valor máximo de corriente. También se permite la monitorización del estado del inversor. En la siguiente imagen se puede ver la consola de la aplicación:

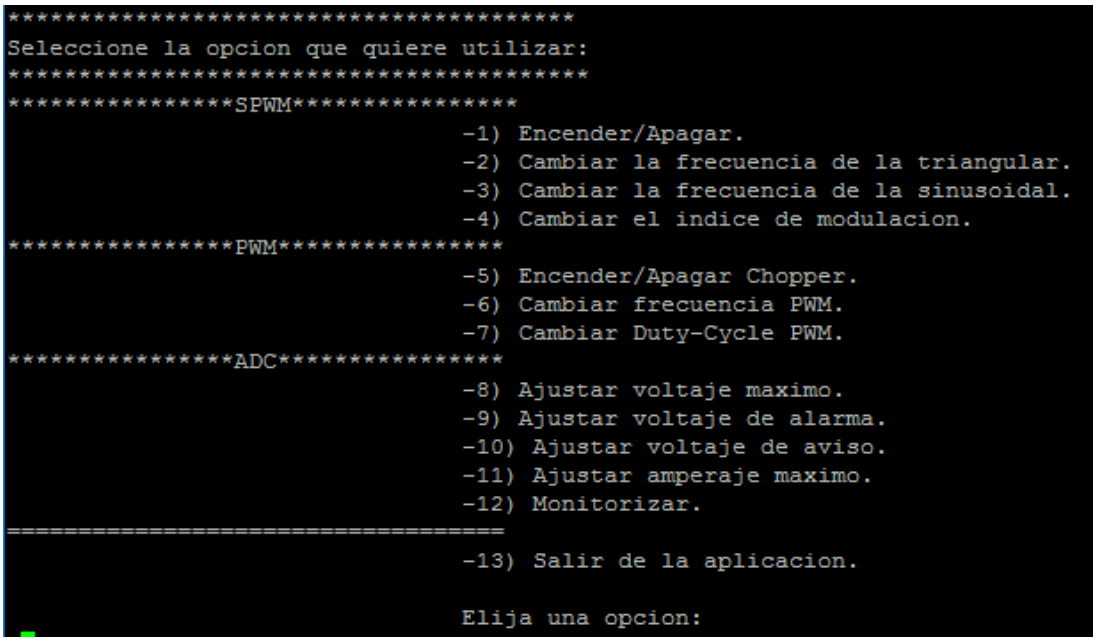


Figura 9.14 Menú principal de la aplicación software.

De cara al futuro se ha tratado de hacer el diseño y la aplicación de la forma más modular posible, de forma que no sea difícil reutilizar módulos o ampliar el sistema. Los *drivers* que se han implementado utilizando protocolos de comunicación se han ideado para ser lo más genéricos posible, de forma que, en caso de querer añadir otro dispositivo que use el mismo protocolo no sea difícil el adaptarlo.

Para efectuar las pruebas correspondientes a esta etapa se ha utilizado la tarjeta de desarrollo con la placa interfaz. Se ha verificado utilizando instrumentación de laboratorio, que ha sido con una fuente de tensión, un generador de funciones y un osciloscopio. En la siguiente imagen se muestra el montaje:

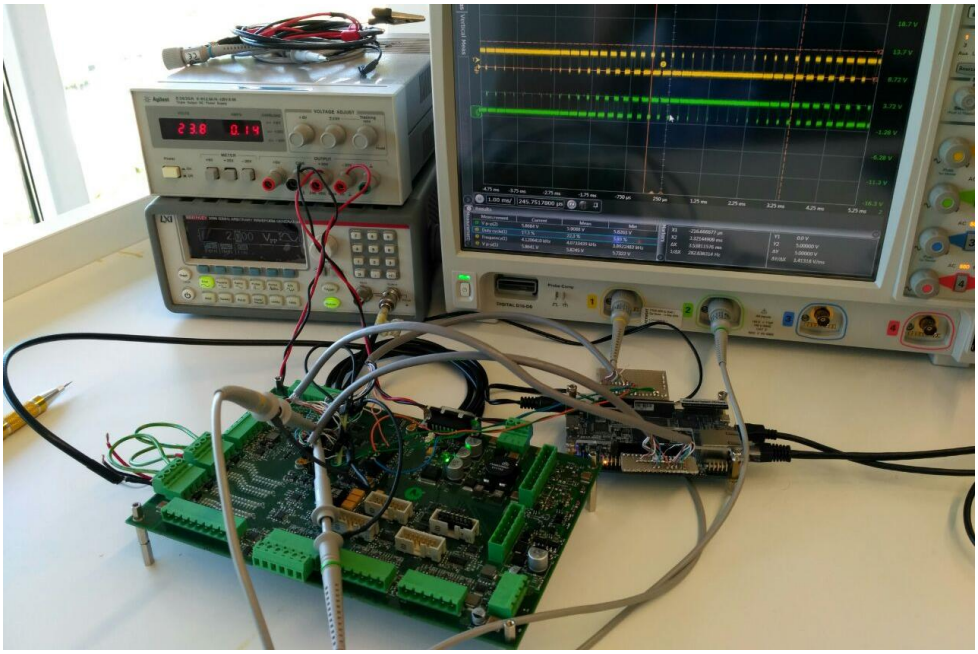


Figura 9.15 Montaje de pruebas. Placa de desarrollo conectada a interfaz.

9.6. Adecuación de las señales

Lo primero que se ha tenido en cuenta es que el trabajo que se está realizando consiste en la migración de un controlador para un inversor basado en µC a FPGA. La placa interfaz se diseñó para ser usada con una tarjeta de evaluación FRDM-K64F. Ésta generaba unas señales con un voltaje de 3.3V, por lo que las salidas de la FPGA tienen que generar el mismo tipo de señal.

Quartus permite configurar los pines de la FPGA, por lo que se les puede asignar diferentes modos, depende de la aplicación y del proyecto a realizar.

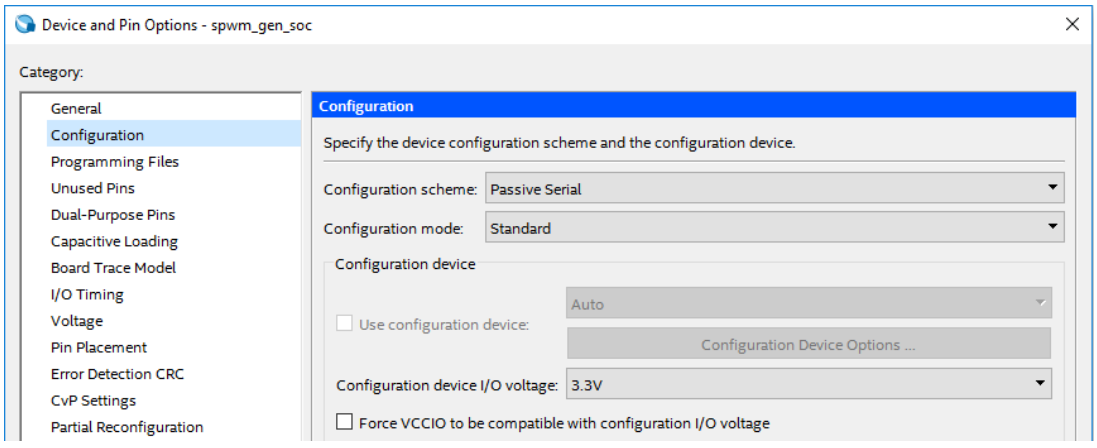


Figura 9.16 Quartus II. Menú de configuración del dispositivo.

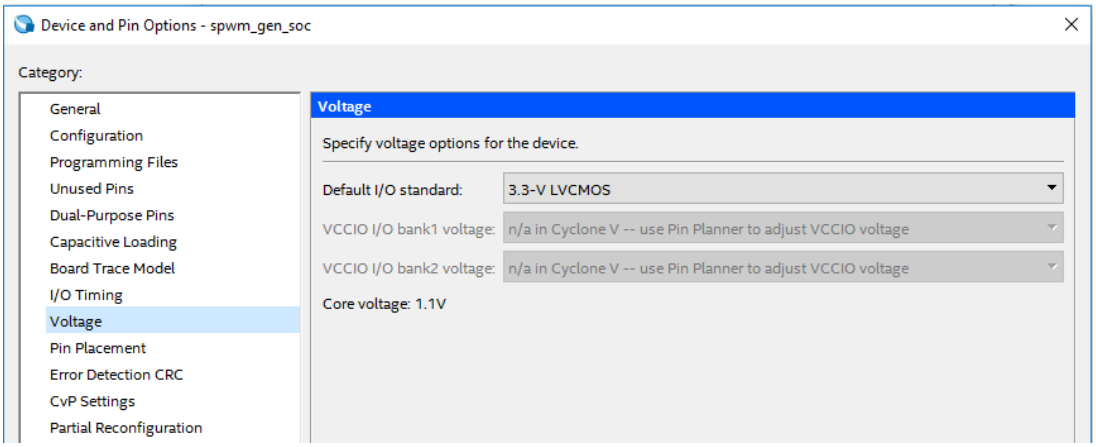


Figura 9.17 Quartus II. Selección del voltaje de las E/S del dispositivo.

Por otra parte, hay que tener en cuenta que el rendimiento de los sistemas electrónicos se ha incrementado, por lo que los retos de los diseñadores de hardware son ahora más complicados. [70]

Uno de los mayores desafíos de diseño que trata la generación de señales de E/S es que su forma de transmisión puede tener efectos significativos en los datos que se desea transmitir. [70]. Esto afecta seriamente a la respuesta de la señal y daña su integridad. Estos problemas pueden solventarse mediante buenas técnicas de diseño.

Durante la fase de implementación se ha podido observar que las señales que está generando la FPGA tienen un sobreimpulso (u *overshoot*) demasiado alto. Esto sucede tanto en los flancos de subida como en los de bajada (*undershoot*). Este fenómeno podría degradar la electrónica y reducir su vida útil, además de que se desea tener unas señales lo más limpias posibles.

La solución que se ha propuesto para contrarrestar este efecto ha sido añadir resistencias terminadoras en serie a las salidas de los pines de la FPGA. Estas resistencias deberían de amortiguar el sobreimpulso que se produce en las transiciones de los flancos. Determinar el valor óptimo de las resistencias puede variar, ya que, en algunos casos, el mejor valor depende de si se está haciendo una transición de activo a bajo o a la inversa. El uso de las resistencias está normalmente relacionado con una adaptación de impedancias. Su utilidad depende mucho de la señal que se esté generando. Para la generación de relojes externos, como puede ser para los protocolos I2C y SPI o para la generación de señales PWM para el inversor, se considera necesario usar las resistencias terminadoras.

Para elegir el valor de la resistencia se ha realizado un breve estudio, probando con diferentes valores.

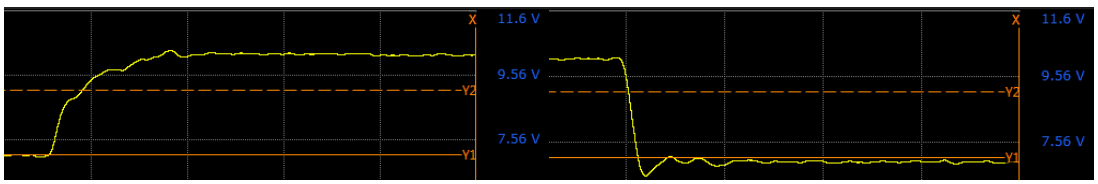


Figura 9.18 Pulso On sin resistencia.

Figura 9.19 Pulso Off sin resistencia.

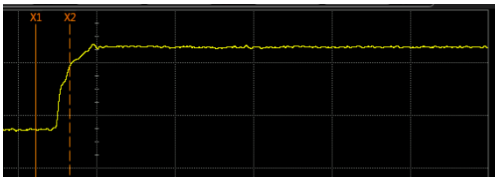


Figura 9.20 Pulso On con resistencia de 24Ω.

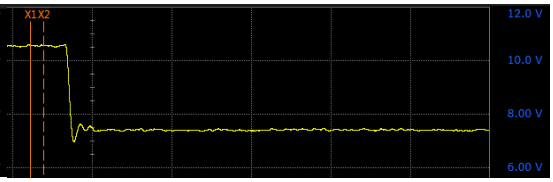


Figura 9.21 Pulso Off con resistencia de 24Ω.

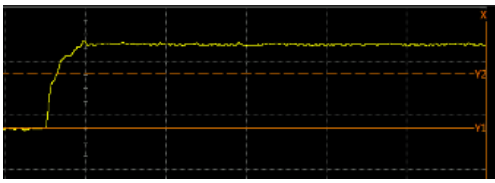


Figura 9.22 Pulso On con resistencia de 50Ω.

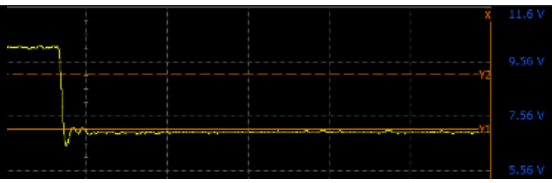


Figura 9.23 Pulso Off con resistencia de 50Ω

Se comprueba que cuanta mayor resistencia se aplica, menor es el ruido y los rebotes que se obtienen. Por el contrario, la resistencia también hace que la señal se retrase, por lo que un valor muy alto tampoco es la opción correcta. Lo que se dedujo claramente del estudio es que las resistencias terminadoras eran necesarias, y se decidió usar un valor intermedio entre 25Ω y 50Ω. Finalmente, las resistencias utilizadas fueron de 33Ω.

Se implementó un pequeño prototipo para insertar las resistencias y hacer más sencilla la inserción del cableado en la placa interfaz. El motivo fue el de poder hacer las pruebas de disparo de IGBTs en el inversor a alta tensión. El resultado fue el siguiente:

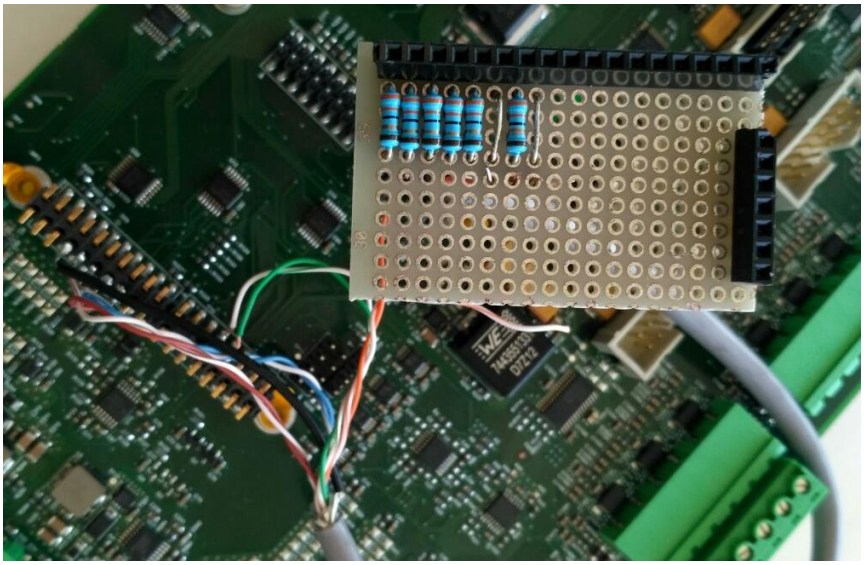


Figura 9.24 Primer zócalo para conectar la placa de desarrollo a la interfaz.

En este prototipo de pruebas se ha utilizado un cable de red Ethernet. El motivo es que es un cable de par trenzado, tiene un enmallado que se conecta a GND, y además, hay suficientes cables como para conectar las señales que se requieren. Al prevenir en gran parte el ruido eléctrico y minimizar los efectos de "*cross-talk*" o interferencias entre cables, se considera una buena solución.

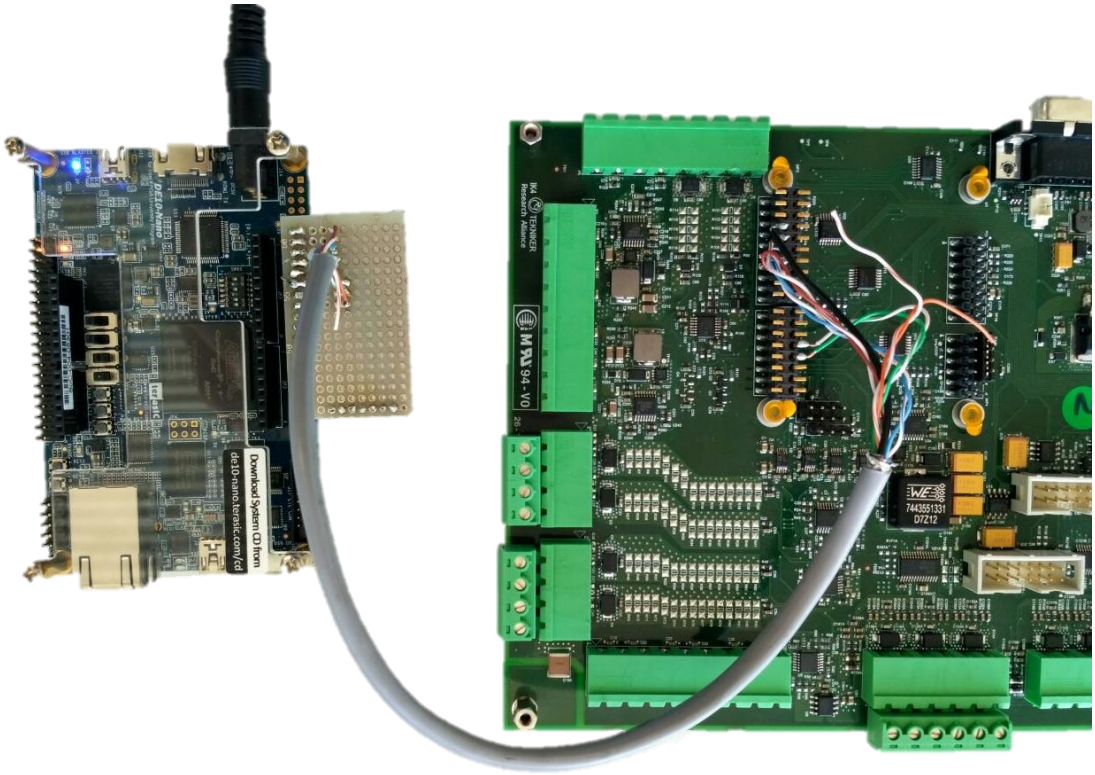


Figura 9.25 Conexión entre Terasic DE10-Nano y tarjeta interfaz.

En esta imagen se puede ver la conexión entre la placa de desarrollo y la placa interfaz. Habrá que conectarla del mismo modo en el inversor.

Esta investigación tiene un aporte y repercusión en el proyecto. Desde Tekniker, y fuera del alcance del presente proyecto, se va a diseñar otra placa interfaz para montar la Terasic DE10-Nano sobre ella. Conocer este detalle permite que se añadan las resistencias terminadoras en la placa interfaz para evitar este problema.

Desde Quartus también se pueden añadir resistencias terminadoras, pero sólo permite usar este modo cuando se está usando voltaje diferencial. En este proyecto, como se ha explicado con anterioridad, se está usando 3.3V LVCMOS.

Otra de las opciones que permite Quartus, y que sí que se ha aplicado en este proyecto, es la de elegir el tipo de *slew rate* deseado. El término slew rate en electrónica digital hace referencia al tiempo de transición entre dos puntos de medida. Suelen ser el 20% y el 80% de la tensión respectivamente.

La FPGA es un dispositivo muy rápido, tener un *slew rate* pequeño puede hacer que las señales sean más vulnerables al ruido, además de que una pendiente de señal demasiado acusada puede también generar ruido en otras señales cercanas.

En el diseño del hardware hay que tener en cuenta también las conexiones a GND. Los dispositivos de Altera incluyen múltiples pines de alimentación y de GND, esto permite al diseñador la opción de ubicar pines de alta velocidad cerca de pines de GND, con la intención de reducir los efectos de que varias salidas de la FPGA estén cambiando de estado de forma simultánea.

En la siguiente imagen se muestra una captura de pantalla del "*Pin Planner*". En Quartus hay que especificar qué pines van a utilizar las señales de E/S. En este caso se están mostrando las salidas por donde sale la señal de PWM que disparará los IGBTs. El *slew rate* por defecto es el '1', se ha cambiado a '0', que es el más lento.

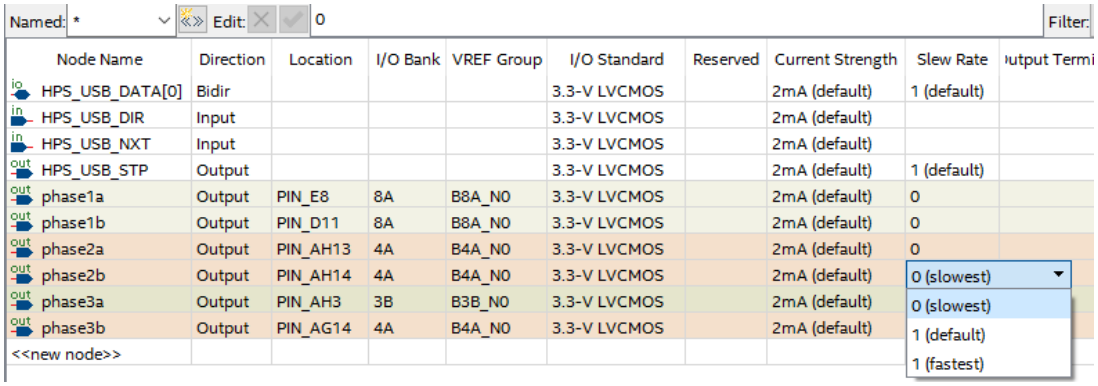


Figura 9.26 Quartus II. Selección de *slew-rate* en el *Pin Planner*.

Otra cosa que hay que considerar a la hora de configurar la FPGA es qué hacer con los pines que no se han utilizado. La FPGA tiene muchos pines disponibles, dependiendo del proyecto es muy probable que sobren pines. La pregunta es qué se debe hacer con esos pines.

En el proyecto actual se utiliza una placa interfaz, que es donde irá conectada la placa de desarrollo Terasic DE10-Nano. Para hacer las pruebas se está conectando mediante cables, pero en el futuro se diseñará de nuevo la placa interfaz con un zócalo donde se pueda conectar la placa de desarrollo que contiene la FPGA.

Quartus permite establecer 5 configuraciones distintas para los pines sin utilizar, se muestra en la siguiente imagen:

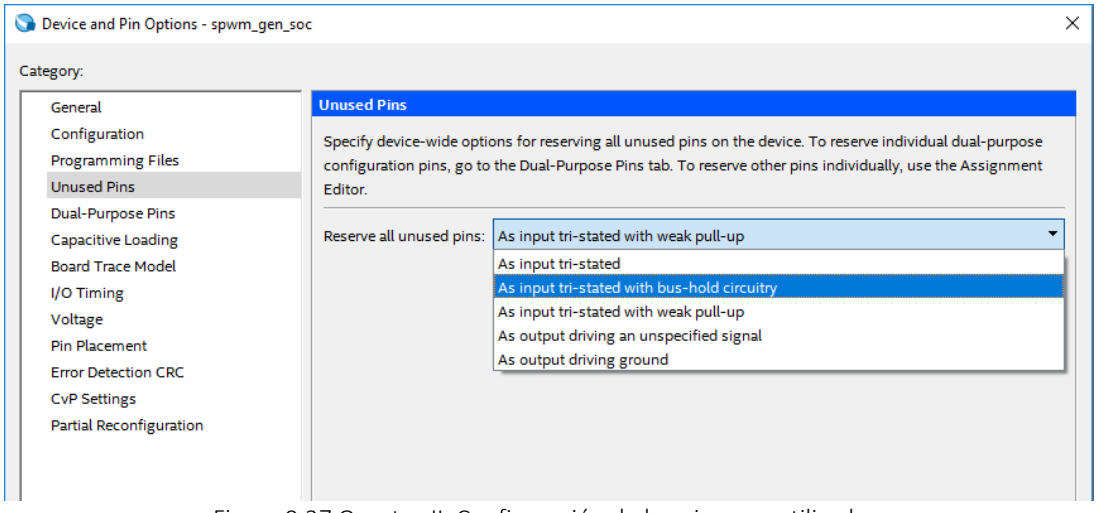


Figura 9.27 Quartus II. Configuración de los pines no utilizados.

La opción que viene por defecto es como entrada tri-estado con *pull-up*. En el pasado, la opción predeterminada era la de configurar esos pines como salida a GND, sin embargo, se cambió porque podría tener consecuencias fatales en caso de que al combinar la FPGA con otro hardware se conectara esa salida con algo distinto a GND.

Se ha concluido que para el presente proyecto se configurarán esos pines como entradas tri-estado con *pull-up* o entradas tri-estado en caso de disponer de las resistencias de *pull-up* de forma externa.

9.7. El prototipo

El prototipo final, que se muestra en esta implementación, es el resultado de todo el conjunto de actividades realizadas desde el capítulo 6 hasta éste. Todos los pasos y decisiones que se tomaron durante el proyecto han concluido en este desarrollo final.

Para ello ha habido tanto desarrollo de hardware, como de software, y se han tomado las líneas de investigación pasadas. No debe, de ningún modo, engañar la finalidad del dispositivo, dado que su uso va a ser el testeo y puesta en marcha del inversor. Todo el trabajo realizado puede ser reutilizable para otros proyectos con el inversor.

El desarrollo de drivers y protocolos de comunicación en HDL, al igual que la generación de señales de control, puede resultar útil para otro tipo de proyectos que vayan a ser implementados sobre una plataforma FPGA. Del mismo modo, abrir el camino hacia el uso de SoC-FPGA, permite ir hacia otro tipo de proyectos con unas exigencias de velocidad y de determinismo mayores.

Por otro lado, el estudio realizado sobre los transitorios va a ser tomado en consideración para adaptar la nueva tarjeta interfaz.

Este presente proyecto es un avance en el proceso investigador que hay en la CAPV respecto a los nuevos materiales semiconductores de banda ancha prohibida y sus aplicaciones en la industria. Tener una aplicación de test y puesta en marcha, sirve para conocer el funcionamiento de los IGBTs basados en SiC y de sus características, y permite continuar con el estudio.

10

Pruebas sobre el inversor

Los bloques funcionales han sido verificados y uno a uno y el prototipo está ya construido. Se han hecho las pruebas pertinentes antes de llevarlo al inversor, y los resultados han sido los esperados. Llega el momento de probarlo en el inversor. Este capítulo expone qué pruebas se han hecho, de qué forma y los resultados que se han obtenido.

10.1. Escenarios

En las siguientes páginas se expondrán las pruebas que se han realizado sobre el inversor. El objetivo es comprobar la robustez de los algoritmos y del hardware que se ha diseñado. Se dividen en tres fases. La primera de ellas consiste en probar el Sistema en vacío y posteriormente con baja tensión. La segunda fase consiste en realizar las comprobaciones en el armario, a alta tensión, se aplicarán 500 V. Por último, la tercera fase será la prueba de un prototipo funcional que permita controlar y monitorizar el funcionamiento del inversor.

10.1.1.Escenario 1. Sin carga.

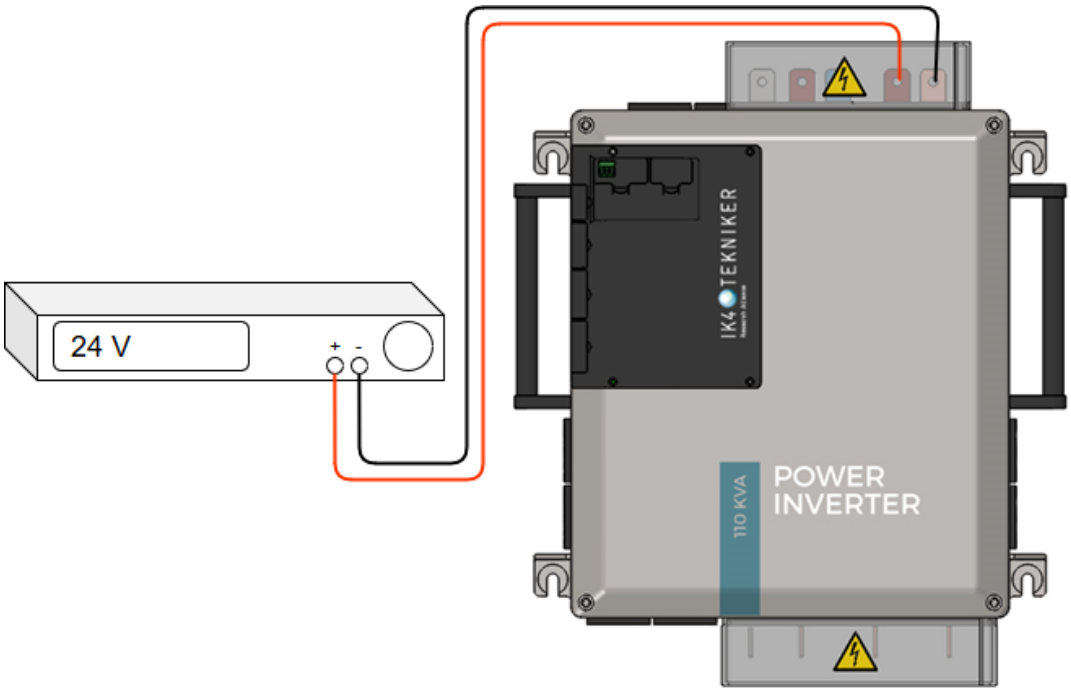


Figura 10.1 Conexión del inversor trifásico sin carga.

El inversor trifásico está únicamente conectado a una fuente de tensión que alimenta la placa interfaz. Se ha conectado además la placa de desarrollo para poder hacer las pruebas en vacío, sin riesgos.

10.1.2.Escenario 2. Baja tensión.

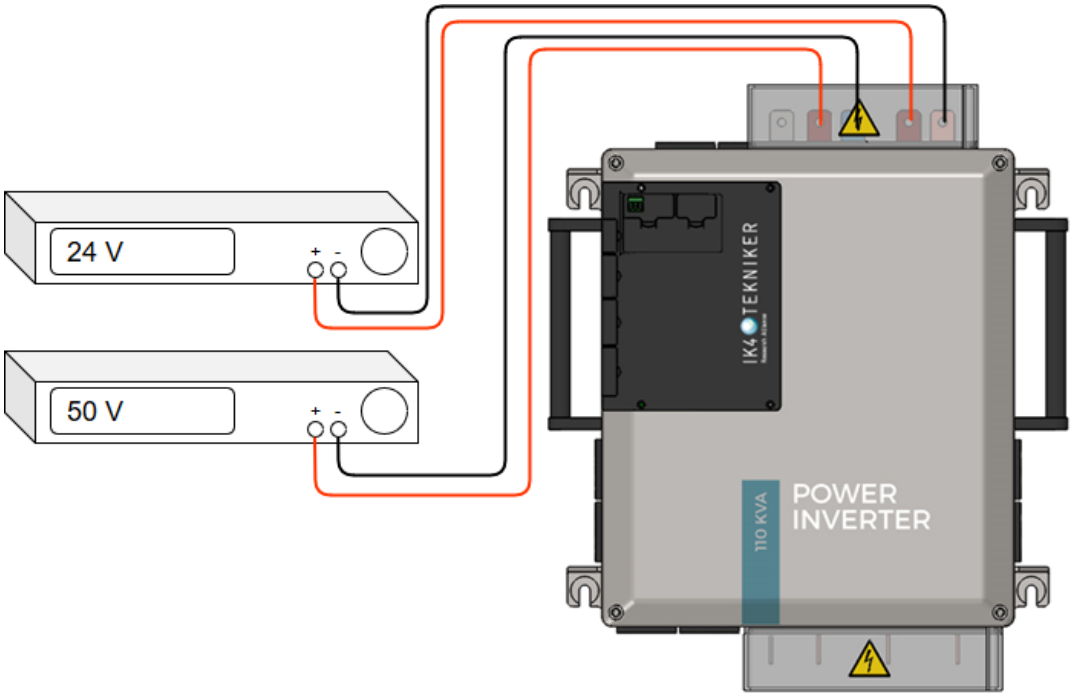


Figura 10.2 Conexión del inversor trifásico a baja tensión.

Se han utilizado en este caso dos fuentes de tensión. Una de ellas sirve, como en el primer escenario, para alimentar la electrónica. La otra fuente se utiliza para introducir en el inversor una tensión en corriente continua controlada. Los valores que se han usado para efectuar las pruebas se encuentran entre los 30-50 VDC.

En esta imagen se puede ver cómo se ha situado el inversor trifásico sobre una mesa de trabajo y se ha conectado la tarjeta de desarrollo, con la FPGA, para hacer las pruebas en el laboratorio:

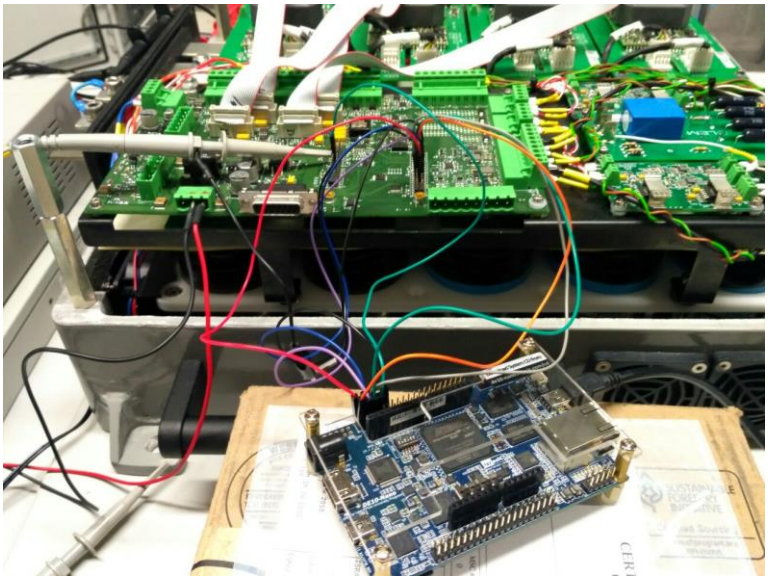


Figura 10.3 Uno de los primeros montajes en el laboratorio.

10.1.3.Escenario 3. Alta tensión.

Una vez hecha la verificación del diseño con el inversor trifásico en vacío y a baja tensión se puede pasar a efectuar las pruebas en un entorno de alta tensión. Esta vez el voltaje de entrada es de 500V. La configuración es la siguiente:

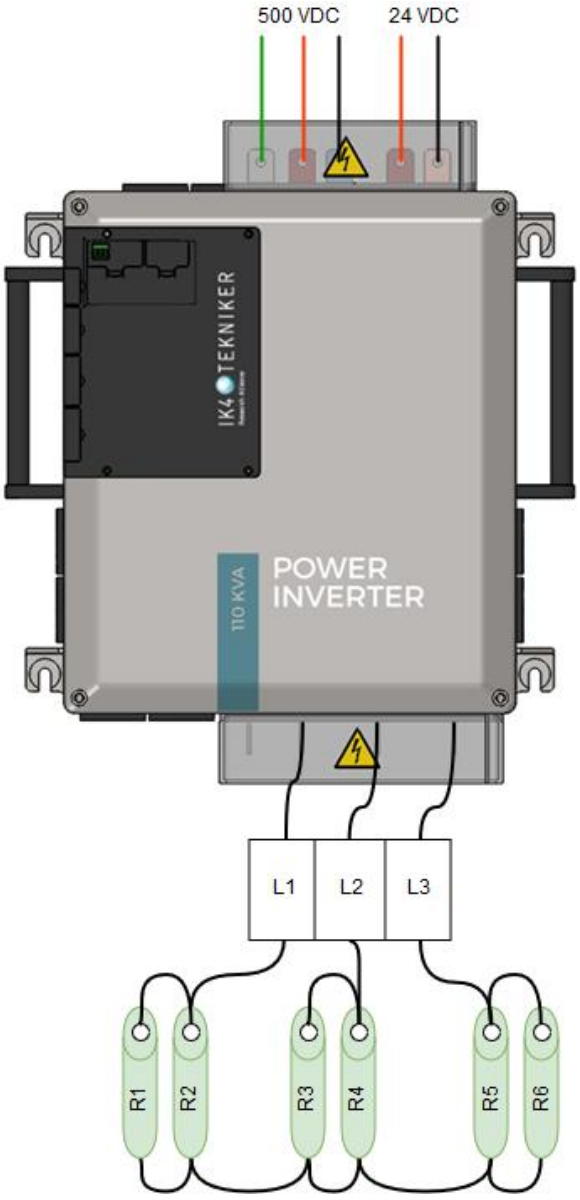


Figura 10.4 Conexión del inversor trifásico a alta tensión.

Para probar el inversor trifásico a alta tensión se ha hecho el siguiente conexionado, donde se añade a la salida una carga RL con topología de estrella. Esto permite la disipación de la energía en forma de calor.

En esta imagen se puede ver físicamente la colocación de la carga RL a la salida del inversor.



Figura 10.5 Carga RL a la salida del inversor trifásico.

10.2. PWM

Se pretende comprobar el funcionamiento del algoritmo generador de PWM. En el diseño de éste se ha especificado que tanto la frecuencia como el ciclo de trabajo tienen que poder ser configurables, además de tener una señal complementaria.

Con estas pruebas se pretende verificar que la señal generada se ajusta a las especificaciones, además cuenta con un tiempo de guarda de 600ns, con el fin de evitar que dos parejas de IGBTs puedan estar en conducción al mismo tiempo.

10.2.1.Sin carga

El primer paso ha sido probar el algoritmo de generación de señal PWM en el inversor sin carga. El diseño ya fue verificado inicialmente en un osciloscopio. Durante las primeras pruebas no se logró que la pareja de IGBTs elegida conmutara con esta señal, posteriormente se dio con el motivo. La placa interfaz requiere de una señal de activación o “enable” para permitir que los IGBTs estén operativos. No se tuvo en cuenta en el diseño, así que hubo que generar esta señal.

Una vez añadida al sistema dicha señal, se han efectuado las pruebas. Se ha probado la conmutación de una pareja de IGBTs configurando una señal PWM de 4kHz, con un *duty-cycle* del 10% del tiempo encendido. En la siguiente captura se observa la conmutación de los dos IGBTs.

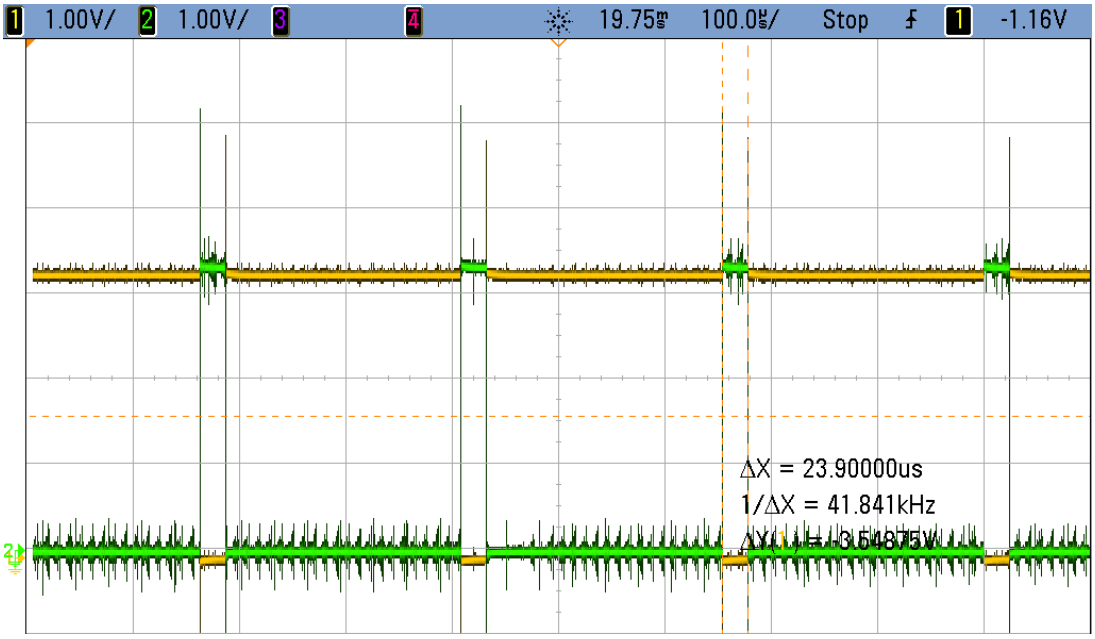


Figura 10.6 Conmutación de una pareja de IGBTs en vacío.

Se observa que las señales son complementarias, por lo que el primer paso de la verificación es correcto. Se ha conseguido que haya una conmutación.

10.2.2.Con carga

En este escenario se está probando la conmutación de la misma pareja de IGBTs aplicándole una tensión. En esta etapa se le aplica baja tensión. En este caso se están obteniendo la siguiente información:

- Canal 1 (amarillo): $V_{GE\,B}$. La diferencia de potencial medida entre la puerta y el emisor del IGBT "bottom".
- Canal 2 (verde): $V_{CE\,B}$. La caída de tensión entre el colector y el emisor del IGBT "bottom".
- Canal 4 (rosa): $I_{C\,B}$. Corriente que circula por el colector en el IGBT "bottom".

Se ha configurado la señal de PWM para que tenga una frecuencia de 4kHz y un *duty-cycle* del 5% para este IGBT. Hay que recordar que funcionan en pareja, por lo que el ciclo del trabajo para el "top" es que su tiempo de encendido sea del 95%. En el lado del inversor, se le ha aplicado una corriente de 30 VDC.



Figura 10.7 Conmutación a 4kHz de un IGBT a 30 VDC.

Los resultados obtenidos aparecen en la figura superior. Para entenderla bien hay que decir que los valores obtenidos en $V_{GE\,B}$ oscilan entre ±15V. A 15V el IGBT se pone en estado de conducción y a -15V se abre el circuito. Cuando se mide en $V_{CE\,B}$ se ve lo que hace el IGBT. Se usa lógica negada, por lo que cuando el $V_{GE\,B}$ está a +15V se observa como el IGBT se cierra, y observando $I_{C\,B}$ se comprueba cómo se produce el paso de corriente. Por el contrario, cuando el IGBT se cierra con la puesta de $V_{GE\,B}$ a -15V, se observa como ya no circula la corriente.

El trabajo que se ha realizado gira en torno al controlador, por lo que la verificación no sólo consiste en ver si los IGBTs conmutan o no, sino en ver que se cumplen los tiempos que se predefinieron a través de la consola de la aplicación.

Con el osciloscopio se ha tomado una medida del tiempo que está en estado de conducción, que son 12,5µs. Si eso es un 5%, al multiplicar ese valor por 20 y convertir el resultado de tiempo a frecuencia se deberían obtener 4000Hz. Se comprueba de la siguiente forma:

$$12{,}5\mu s \cdot 20 = 250\mu s;$$

$$F = \frac{1}{T} \rightarrow \frac{1}{250 \cdot 10^{-6}} = 4000Hz = 4kHz;$$

Se comprueba de este modo que la generación de la señal de PWM es correcta y se adecúa a los parámetros introducidos por el usuario.

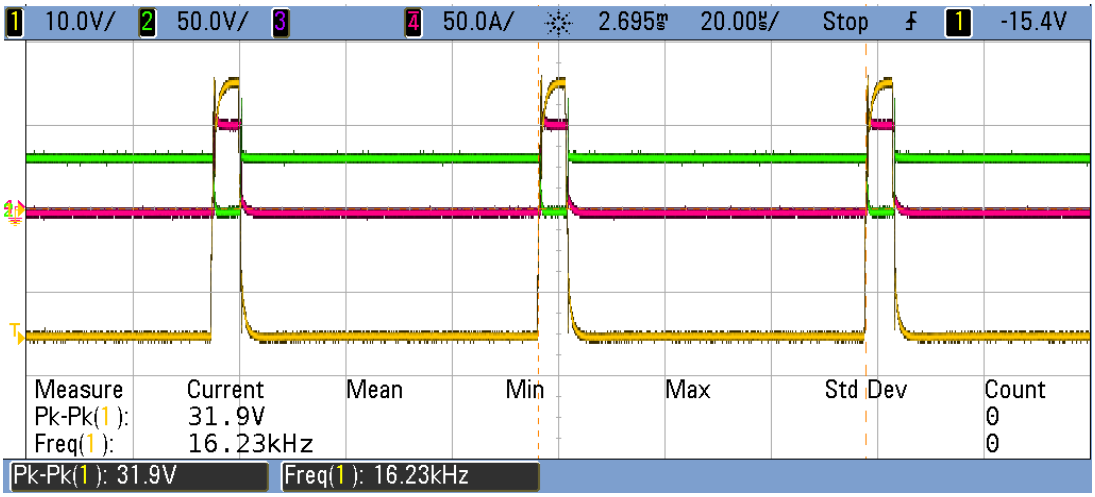


Figura 10.8 Conmutación a 16kHz de una pareja de IGBTs a 30 VDC.

En esta segunda imagen se ha repetido el experimento, pero se ha aumentado la frecuencia de conmutación a 16kHz. Se observa cómo se sigue produciendo la conmutación. Ampliando la imagen se podría apreciar que existe una pequeña sobretensión. Esto no es debido al control, es por una inductancia parásita, que depende del área entre los condensadores y los IGBTs. Cuanto menor sea el área, mejor, pero es un efecto que no se puede evitar, siempre va a aparecer algo.

10.3. SPWM

Al igual que se ha hecho con el PWM, hay que verificar la generación del SPWM en el inversor. Antes de probarlo en el laboratorio, se han hecho todas las pruebas oportunas para garantizar su corrección.

10.3.1.Sin carga

Se ha comprobado el funcionamiento de la SPWM sin carga. Para ello se ha generado la señal y se ha recogido la información del voltaje de salida y de la conmutación de uno de los IGBTs.

En el primer caso de prueba se ha generado una señal sinusoidal de 100Hz, la frecuencia de la portadora (o de conmutación) es de 4kHz y el índice de modulación es del 10%

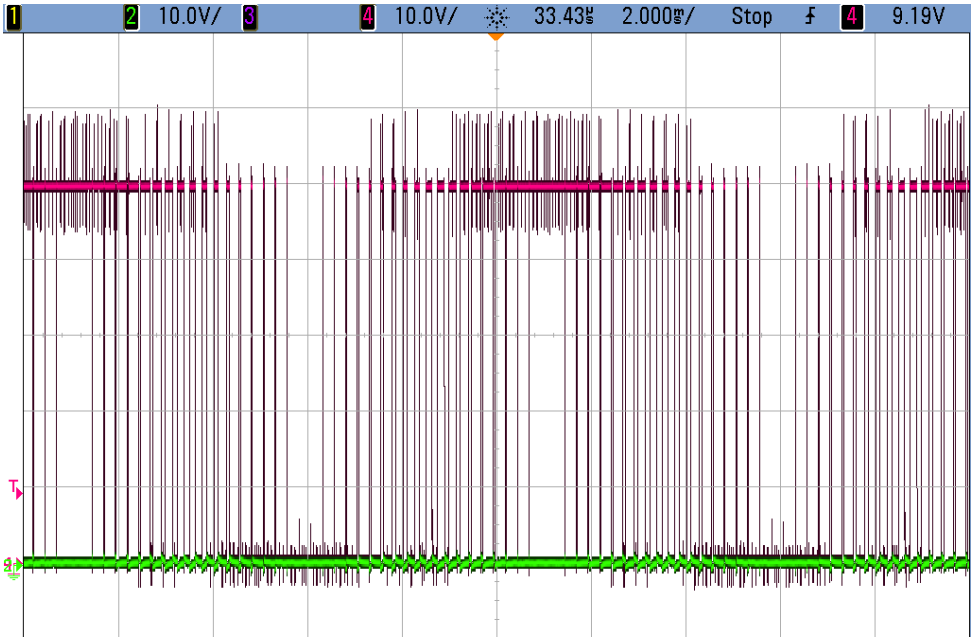


Figura 10.9 Conmutación de uno de los IGBTs.

Se ha ejecutado un segundo caso de prueba, esta vez se han mantenido los mismos parámetros, pero se ha establecido el índice de modulación al 50%.

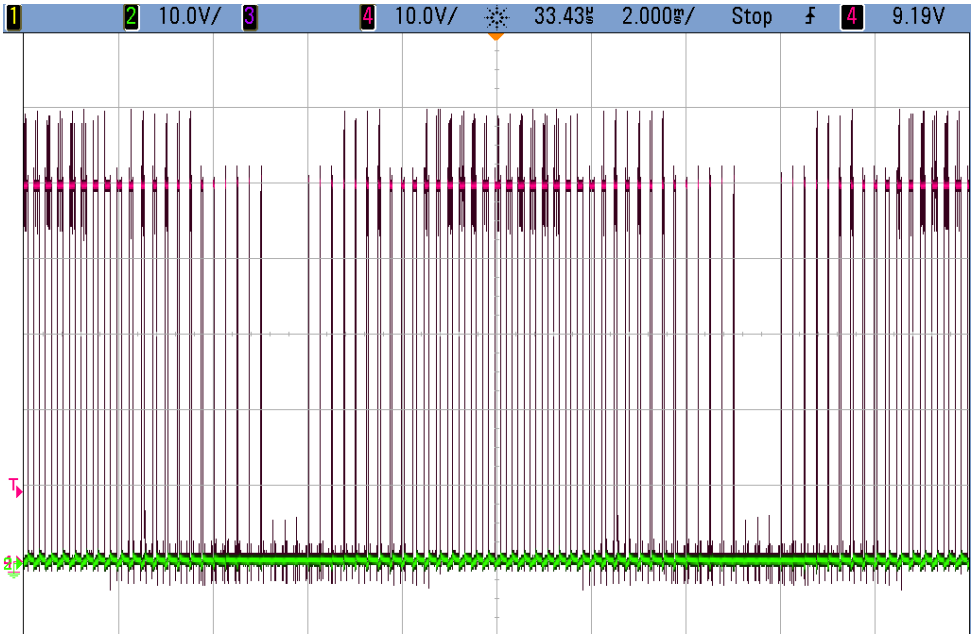


Figura 10.10 Conmutación de uno de los IGBTs.

10.3.2.Con carga

Para esta prueba se ha conectado una fuente de tensión al inversor trifásico a 50 VDC. El valor de la señal generada es el mismo que en el anterior escenario, 100Hz para la sinusoidal, 4kHz para la portadora y un 10% de índice de modulación.

La primera prueba efectuada consiste en medir las tres corrientes en el inversor. Para ello se han usado unas sondas.

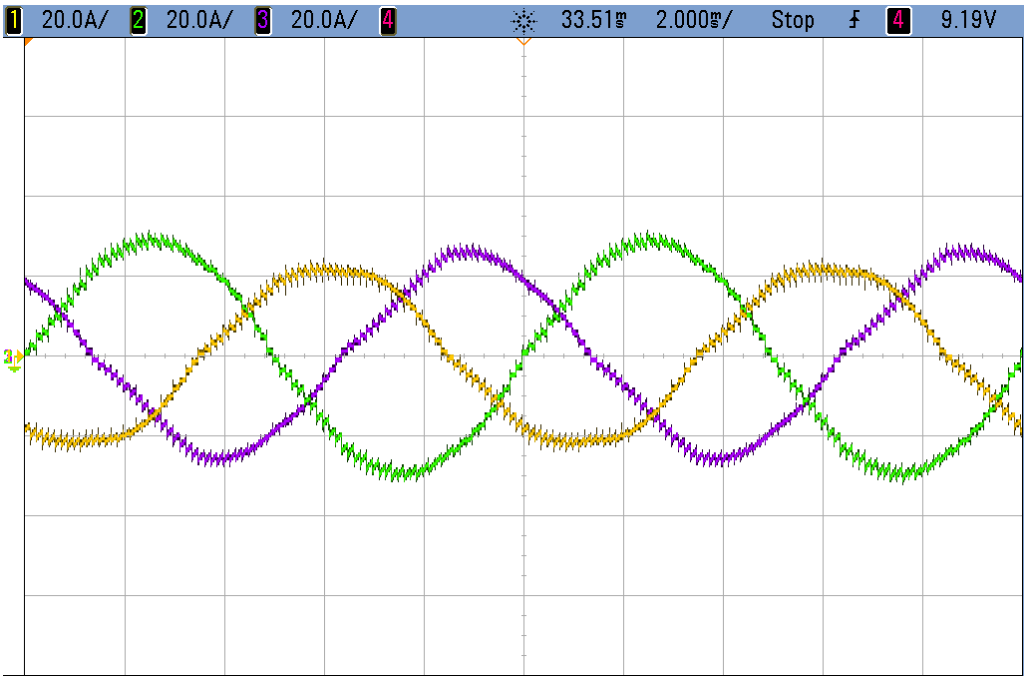


Figura 10.11 Medida de las tres corrientes en el inversor.

En esta imagen se pueden ver las corrientes de las tres fases. Se detectó que el desfase era incorrecto, y esto fue debido a un error en la generación de la señal SPWM. Este fallo pasó desapercibido en las fases previas, pero la solución fue sencilla. En la siguiente prueba, con el inversor en alta tensión, se va a ver como la señal está corregida y tiene el desfase correcto, a 120º respectivamente entre sí.

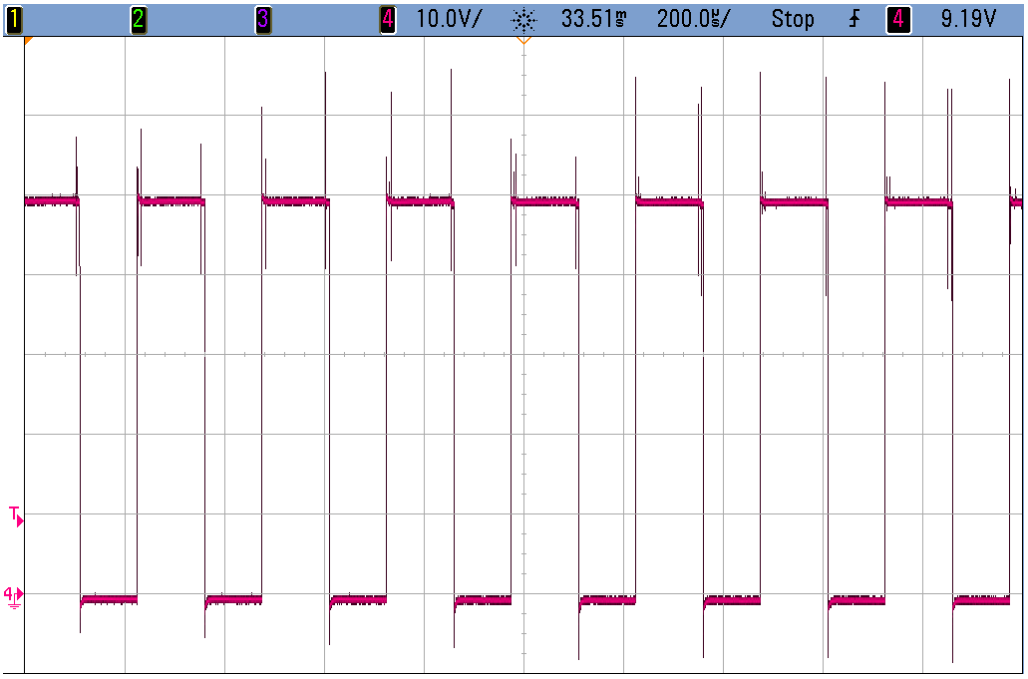


Figura 10.12 Salida de tensión del inversor.

En esta imagen se ve la salida de voltaje en la conmutación de uno de los IGBTs. Se observa como varía entre 0-50V.

10.3.3.Alta tensión

Esta es la última prueba para asegurar el correcto funcionamiento de la SPWM en el inversor. Esta vez se va a probar el control de la FPGA con el inversor de corriente instalado en el armario. Para ello se la está introduciendo un voltaje de 500 VDC. Como carga se ha colocado un circuito RL en estrella.

Esta vez se han hecho tres pruebas, pero se van a presentar únicamente las dos capturas del osciloscopio más representativas. En primer lugar, se ha empezado con la generación de una señal de 50Hz, la frecuencia de conmutación es de 4kHz y el índice de modulación es del 10%. Se han obtenido las corrientes y el resultado ha sido el esperado.

Al haber salido bien la prueba anterior, se ha decidido aumentar el índice de modulación al 25%. El resultado de las corrientes en las tres fases se expone a continuación:

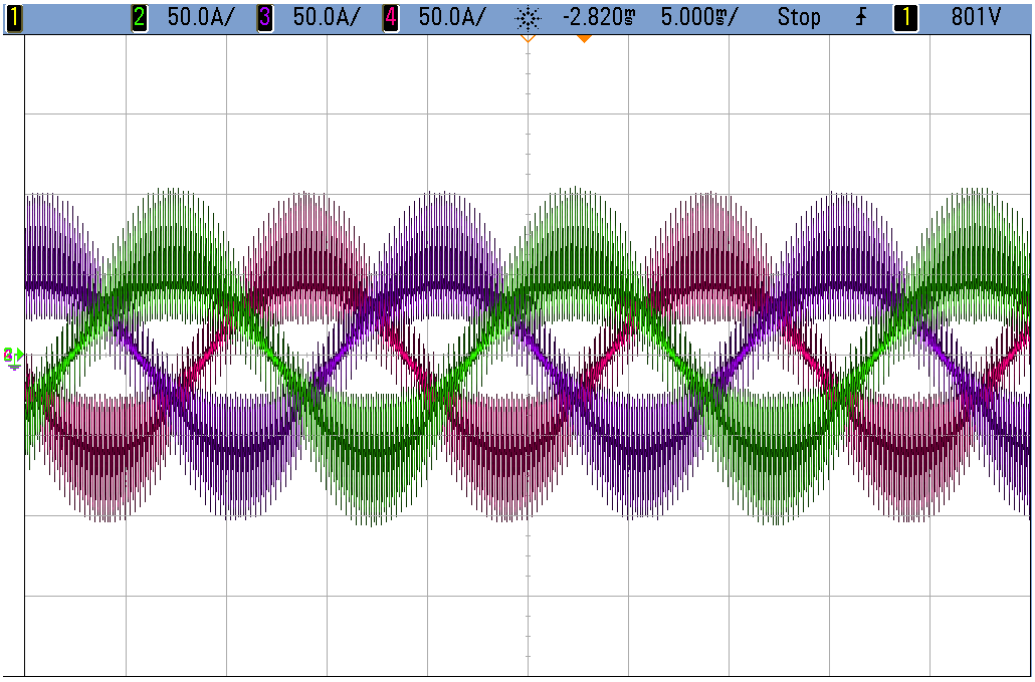


Figura 10.13 Medición de las corrientes en alta tensión.

Como última prueba, se ha aumentado en casi cuatro veces la frecuencia de conmutación, ahora la señal portadora va a 15kHz. El resto de los parámetros se mantienen igual:

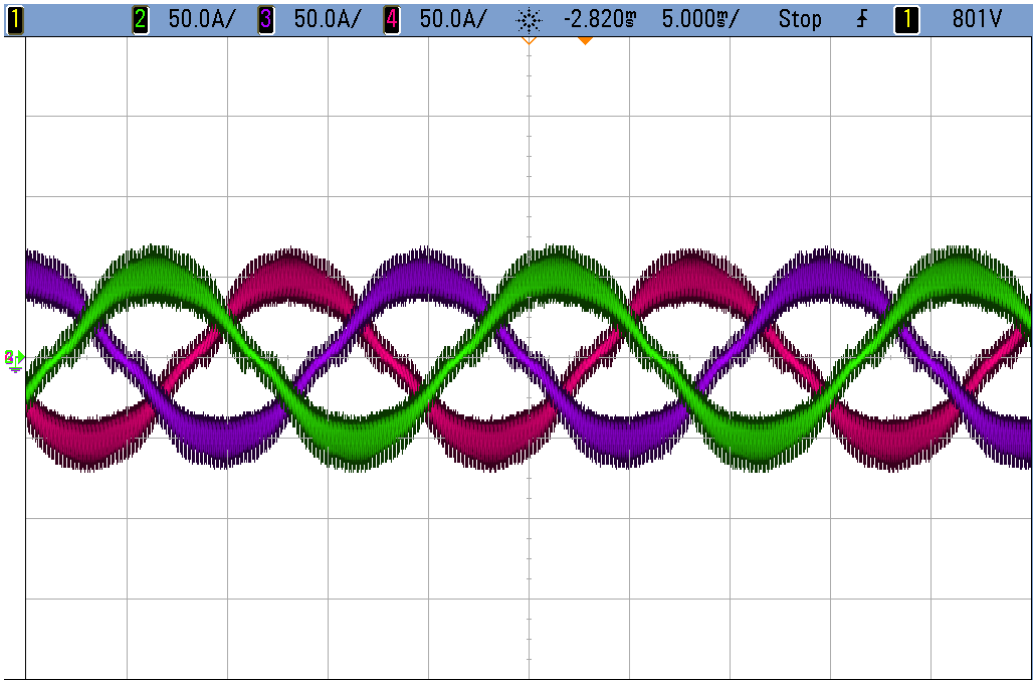


Figura 10.14 Medición de las corrientes en alta tensión.

En la siguiente imagen se puede ver cómo se ha conectado la placa de desarrollo al inversor trifásico de potencia. Como la tarjeta interfaz fue diseñada para otra tarjeta de desarrollo, se ha montado un zócalo provisional para hacer las conexiones:

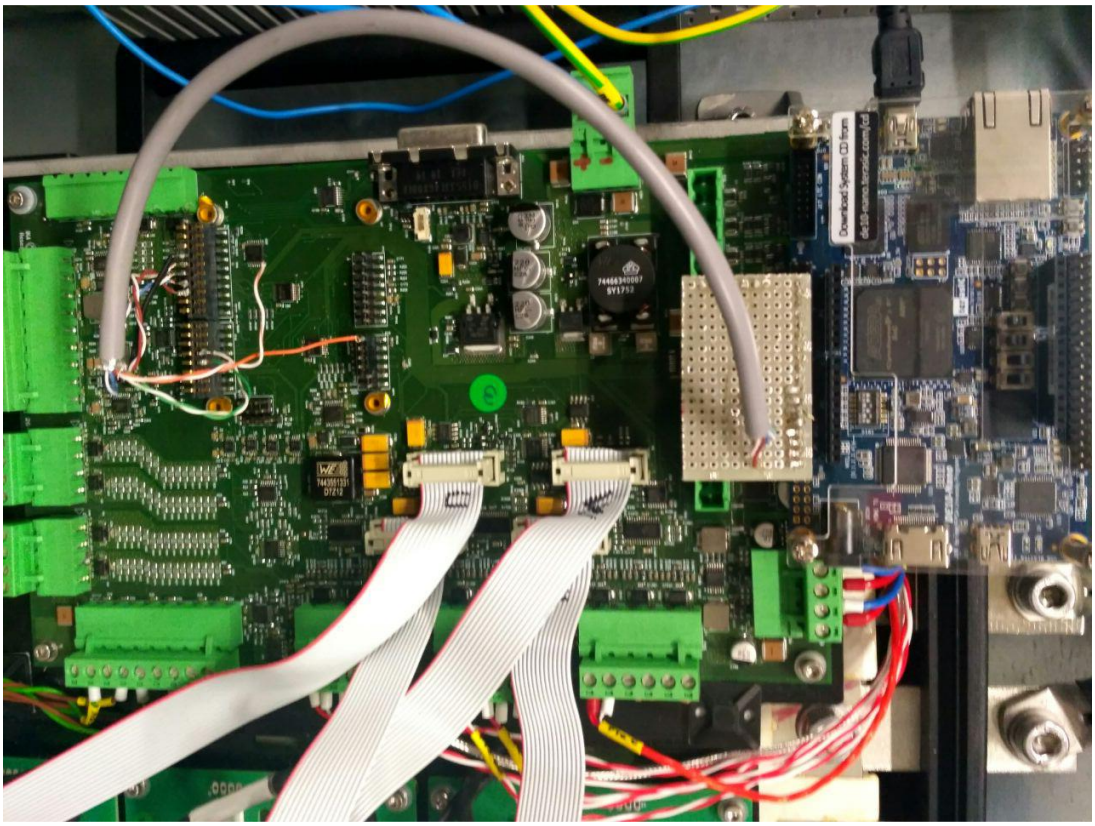


Figura 10.15 Conexión de la placa de desarrollo al inversor es escenario de alta tensión.

10.4. Sistemas de seguridad: Monitor de corriente y tensión

Como se ha comentado con anterioridad, el prototipo tiene también una parte de control. Esta se encarga de tomar decisiones para proteger al equipo. En este apartado se va a verificar el funcionamiento del control de corriente y de tensión.

El inversor tiene tres sensores de corriente, que miden la intensidad en cada una de las fases, y uno de voltaje. En caso de sobreintensidad, el controlador tiene que parar la señal de SPWM y activar una señal PWM para poner en funcionamiento el chopper. Como no se desea poner en riesgo la integridad del inversor, se ha probado el control del prototipo a través de la placa interfaz y con un generador de funciones. Los resultados se muestran a continuación.

10.4.1.Sobreintensidad

Se va a verificar el controlador en caso de que haya un pico de corriente. Para ello se está generando una señal sinusoidal cuya amplitud varía entre ±5V y cuyo centro está en valor medio se encuentra en 0V. Esta señal se ha conectado en el zócalo de uno de los sensores de corriente, y trata de simular los posibles valores que llegarán desde un sensor real. Con ayuda de un osciloscopio se han capturado las señales de SPWM de una de las fases y se está recogiendo también información sobre el chopper. El objetivo es ver si el chopper entra en acción cuando la corriente que circula por el inversor está fuera de los umbrales permitidos.

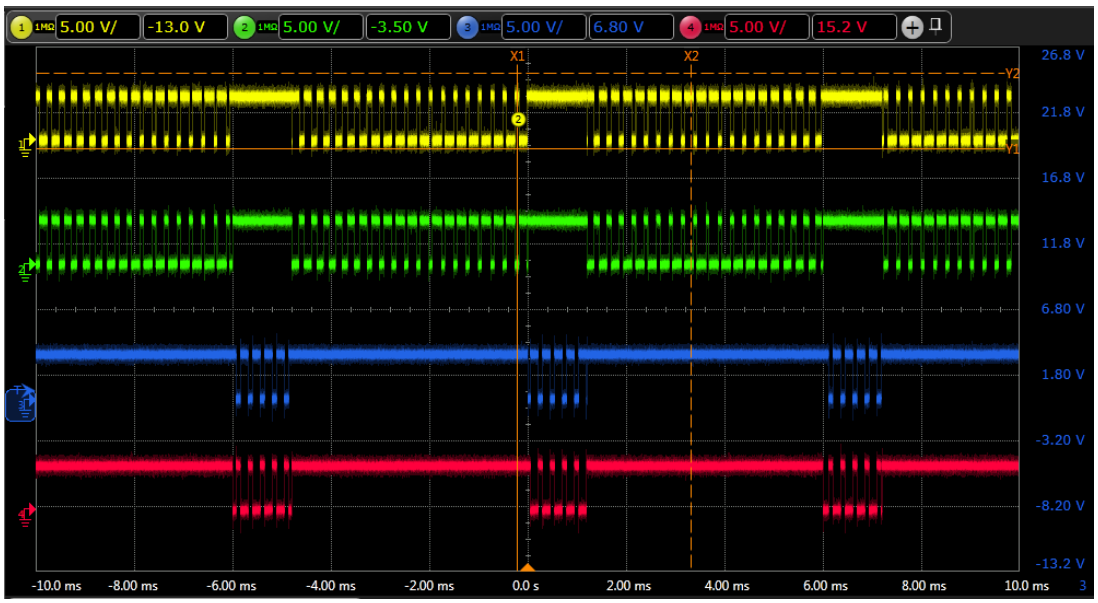


Figura 10.16 Verificación en sobreintensidad.

Se observa en esta captura como cuando la tensión que lee el ADC es superior a la que se ha configurado en el menú del ADC el control deja las dos parejas de IGBTs abiertas y entra en funcionamiento el chopper. Cuando los valores vuelven a la normalidad, la pareja de IGBTs que forman el chopper se quedan abiertos, y se vuelve a poner en funcionamiento la señal de SPWM.

10.4.2.Sobretensión

Como se ha explicado con anterioridad, en este control existen tres estados. El de aviso, que simplemente muestra por pantalla que se ha alcanzado un determinado nivel, el de alarma, que activa el chopper y lo regula en función de la sobre tensión y el de emergencia, que mantiene el chopper y desactiva el resto de IGBTs para protegerlo.

Se van a efectuar una serie de pruebas para verificar su funcionamiento. El escenario va a ser similar al anterior. Se van a configurar los parámetros con unos valores ya conocidos, los de la tabla 7.2, de este modo se sabe a qué tensión saltará el estado. Al igual que antes, se va a introducir una señal por la entrada correspondiente al sensor de tensión.

En la configuración del control, se va a definir el aviso a 500V, y la alarma y el estado de emergencia a 600V y 900V respectivamente. Estos valores se han tomado simplemente porque ya son conocidos, y resulta más cómodo usarlos a la hora de aplicar un voltaje a través de la fuente de alimentación o del generador de funciones.

En primer lugar, se ha probado con una fuente de alimentación, ya que para esta prueba, se puede ver mejor la respuesta del ciclo de trabajo del PWM en función del voltaje aplicado. Las señales que se van a generar son una SPWM de 100Hz conmutando a 4kHz y la frecuencia del PWM, cuando este se active, será de 4kHz.

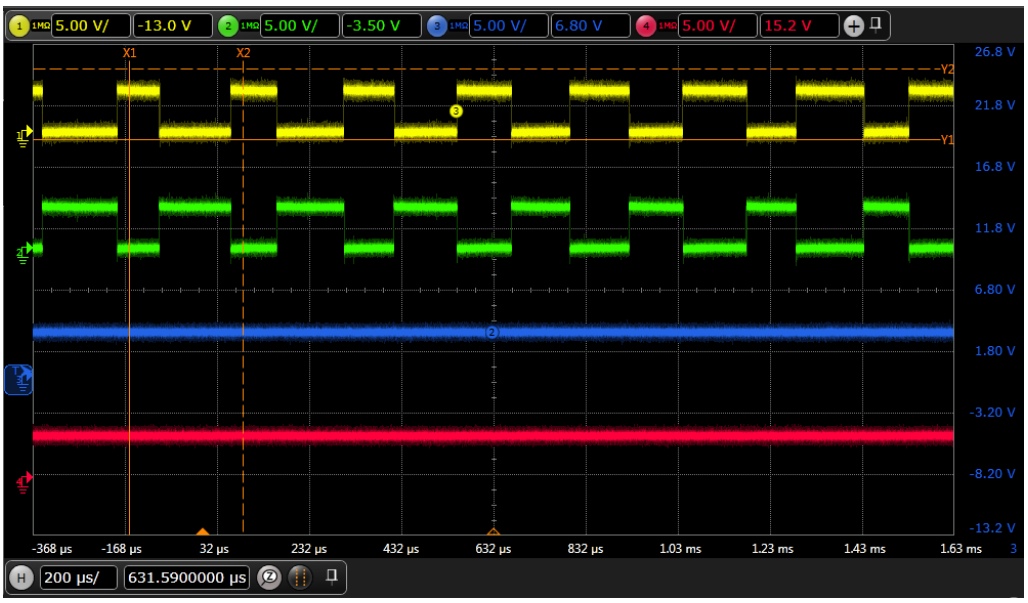


Figura 10.17 Verificación del control de tensión. Estado 0.

En esta captura se muestra el sistema en lo que se ha denominado “estado 0”. En este momento la tensión está por debajo del umbral de alarma. En el estado 1, se vería una imagen parecida, sólo que aparecería un mensaje por la consola de la aplicación. La señal de SPWM se mantiene activa y no ha saltado aún el PWM. La tensión que se le ha aplicado es de 2V, que se encuentra por debajo de los umbrales especificados.



Figura 10.18 Verificación del control de tensión. Estado 2.

En la imagen anterior se ha aplicado una tensión de 3,28V. Si se observa la tabla 7.2, se observa que el valor equivalente a 600V es 3,25V. En este caso se ha superado levemente el umbral, y como se observa en la imagen ha entrado en funcionamiento la PWM. Esta tiene un ciclo de trabajo del 3%, y en la siguiente imagen se verá cómo al aumentar el voltaje, aumenta también el *duty-cycle* de la señal.

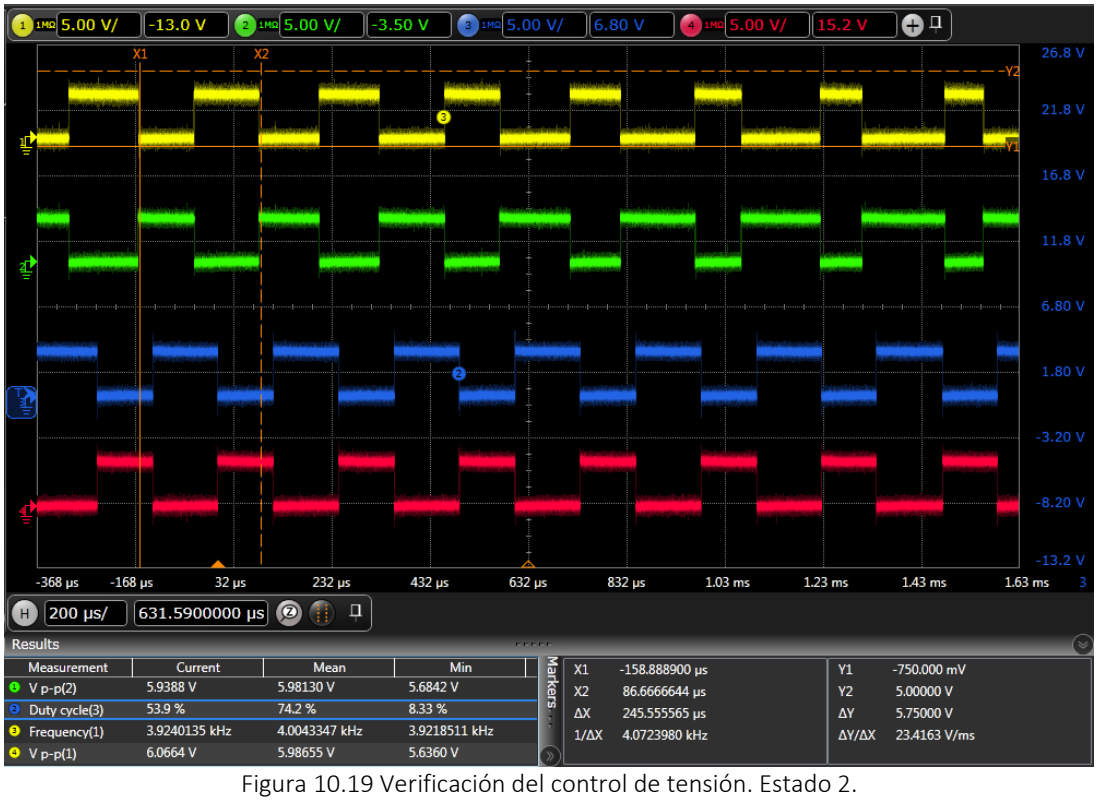


Figura 10.19 Verificación del control de tensión. Estado 2.

A 4V el ciclo de trabajo ha aumentado con respecto a la anterior captura. Se sitúa ya en un 46%. Se observa que el control está funcionando y tiene la respuesta lineal en función de la tensión, tal y como ha sido programada. Como aún no se ha llegado al límite, la señal de SPWM sigue inalterada.

Podría haber otras políticas a la hora de aumentar o disminuir el ciclo de trabajo, pero se consideró que una progresión lineal podría ser una buena estrategia a seguir.



Figura 10.20 Verificación del control de tensión. Estado 3.

En esta imagen se ve qué es lo que sucede cuando se llega al estado de emergencia. Como se puede comprobar, la señal PWM se para y se dejan los IGBTs en circuito abierto, es una medida de protección. Por otra parte, la señal PWM para el chopper se mantiene activa y en conmutación. El ciclo de trabajo es ya del 99%.

Además de esta prueba, se ha efectuado también otra adicional introduciendo la señal sinusoidal. Las imágenes obtenidas no aportan más información de la aquí mostrada, pero sí que sirvieron para ver que el algoritmo seguía funcionando de igual manera cuando se le aplicaba una tensión negativa.

11

Conclusiones y líneas futuras

En este último capítulo se hará un análisis y evaluación del trabajo realizado. Servirá para explicar las complicaciones que han surgido durante el desarrollo del proyecto, así como las lecciones aprendidas que podrían ser útiles para futuros proyectos.

11.1. Visión general y líneas abiertas

Esta memoria recopila un trabajo que ha llevado 8 meses. Se trata de un proyecto ambicioso que no solo ha cumplido con la finalización de un primer prototipo funcional, sino que además ha dejado abiertas varias líneas de investigación por las que poder continuar.

En este recorrido, se ha empezado por la selección de un hardware adecuado a una necesidad específica, ha habido un proceso de puesta en marcha y de documentación, se han probado diferentes herramientas y formas de trabajar con una SoC-FPGA, y se ha llegado a la implementación de diferentes algoritmos que han convergido en una aplicación final, con su parte hardware y software. En resumen, el proyecto cumple con los objetivos propuestos al inicio.

Desde un punto de vista global a una perspectiva específica, se podría decir que se ha abierto camino para las siguientes líneas de investigación.

- Diseño de algoritmos en FPGA.
- Generación de señales.
- Comunicación con dispositivos periféricos que usen los protocolos I2C/SPI.
- Implementación de *drivers*.
- Uso de IP Cores.
- Uso de FPGA con software en ensamblador o C a través del *SoftCore* Nios.
- Desarrollo de aplicaciones para Linux en tiempo real.
- Implementación de arquitecturas SoC-FPGA.
- Aplicaciones web a través de un servidor Apache que permitan controlar la FPGA o monitorizarla.
- Adecuación y mejora de las señales generadas.
- Desarrollo de nuevos proyectos sobre SoC-FPGA.
- Paralelismo a través de hardware.
- Paralelismo de tareas usando hilos o "*threads*".

Aparte de estas líneas que quedan abiertas para futuros proyectos, no hay que olvidar que las FPGAs están teniendo en estos momentos mucho peso, y aunque este proyecto no abarque lo que se va a mencionar a continuación, las FPGAs están siendo utilizadas como coprocesadores para realizar operaciones complejas. Intel ya ha sacado al mercado procesadores con FPGA integrada [71]. Combinar un dispositivo lógico programable con una CPU, aúna dos mundos distintos pero complementarios. Y no se puede dejar de lado la importancia que tiene hoy la computación paralela, se podría llegar a unir la programación en openMP con openCL, y hay artículos en los que se juntan ambas con MPI [72], permitiendo hacer un clúster de máquinas trabajando en paralelo con las tres tecnologías.

11.2. El prototipo

Todo el trabajo realizado ha sido necesario para desembocar en el diseño, implementación y verificación de un prototipo funcional. Desde el primer momento ha habido un asentimiento de los conocimientos previos y un aprendizaje constante, descubriendo las tecnologías disponibles y su forma de funcionamiento.

Además de los entornos software, para efectuar las pruebas, ha habido que adecuar la placa de desarrollo a la placa interfaz. Para ello se ha construido un zócalo provisional y se han tenido en cuenta buenas prácticas para evitar el ruido electromagnético en la medida de lo posible. Por otro lado, también se ha observado durante la verificación la forma de las señales que genera la FPGA, y se ha mejorado la salida mediante el uso de resistencias terminadoras al final de los pines de salida. Sobre este tema, se ha elaborado también un estudio, y se ha intentado que las mediciones obtenidas fueran lo más veraces posibles.

Este prototipo ha sido verificado con la ayuda de un osciloscopio y se han efectuado diversas pruebas de adquisición de datos con la tarjeta interfaz, de forma que se comprueba que los algoritmos de control implementados responden de forma correcta a la información que reciben. Finalmente se ha montado sobre el inversor trifásico y se han obtenido pruebas de su funcionamiento, es decir, el prototipo ha sido validado en condiciones reales.

11.3. Retrospectiva

En el segundo capítulo se especificaba el alcance y objetivo de este proyecto. Ahora es el momento de hacer balance y analizar si se ha cumplido con lo especificado. El proyecto ha dado como resultado un prototipo funcional, que es una arquitectura SoC-FPGA configurable a través de una aplicación software que se ejecuta sobre un entorno RT Linux.

Se ha seguido una metodología, y se ha dividido el problema principal en un conjunto de problemas más pequeños a resolver. Finalmente, la resolución de esos problemas ha concluido en la aplicación de test y puesta en marcha, que es lo que se especificó al inicio de este documento.

Respecto a las fechas marcadas en el *roadmap* (Figura 5.3), se han cumplido. Esto es indicador de una buena planificación. También refleja el conocimiento sobre la dificultad y las posibles complicaciones del proyecto. Mirando el diagrama de Gantt (Figura 5.2), se estimó que la aplicación estaría terminada para el 30 de Abril. En la realidad, acabó en la segunda semana de Mayo, pero también se planificó pensando en los riesgos, por lo que hubo tiempo de terminar el proyecto en el plazo estipulado.

En este proyecto, si bien es cierto que se partía con una base y experiencia sobre FPGAs, ha habido un autoaprendizaje constante y muy enriquecedor. Como se puede ver en esta memoria, el proyecto, aunque con un fin específico, abarca numerosos temas. Se han

programado módulos en VHDL, implementado protocolos, diseñado *drivers*… Posteriormente se ha llegado a hablar de sistemas operativos, de su funcionamiento, de qué supone el tiempo real y cómo hacer programas para aprovechar esta característica. Incluso se ha bajado al nivel del Kernel y a la programación de módulos, que acabó en el diseño de un controlador de interrupciones. Y por supuesto, se ha aprendido a diseñar arquitecturas SoC y hacer un sistema completo a partir de pequeños subsistemas, combinando la descripción de hardware con la programación en software. En resumen, ha sido un camino largo, pero bonito.

11.4. Labor investigadora

Este proyecto ha tenido un trabajo de investigación en dos sentidos, por un lado, el de contextualización, y por el otro, de desarrollo.

Es la primera vez que se desarrolla un trabajo en el campo de la electrónica de potencia. En los estudios de grado cursados, como es normal en una ingeniería informática, no se tratan temas sobre sistemas eléctricos. Se ha tenido que hacer una investigación previa sobre su funcionamiento para saber qué tipo de señales generar.

Lo mismo ocurre con la lectura e interpretación de esquemáticos. En el proyecto ha habido que comprender los esquemas electrónicos de las placas con las que se ha trabajado. Esto incluye las placas de evaluación y la tarjeta interfaz.

Respecto a la investigación realizada para la implementación del prototipo, existía un conocimiento previo de desarrollo en VHDL y se había trabajado con el softcore Nios II. Sin embargo, hasta la fecha, no se había trabajado en una arquitectura SoC-FPGA.

En diversos puntos de la implementación ha habido problemas respecto a las fuentes. No hay mucha información sobre cómo diseñar una arquitectura SoC-FPGA. Exactamente igual pasó con la gestión de interrupciones, había muy poca información, y no estaba del todo completa.

Es por esto por lo que, en algunos apartados de esta memoria, se ha contado el proceso con bastante detenimiento y detalle. Ante la falta de información, se pretende que esta documentación también pueda servir para replicar desarrollos similares. Por estas razones, se estima de interés dar las explicaciones oportunas de los procedimientos realizados.

Se considera que en este trabajo se han tocado muchos puntos diferentes de una arquitectura hardware, y se ha trabajado a múltiples niveles de abstracción. Se ha ido desde una capa de aplicación hasta el Kernel de Linux. Casi todo lo que se podía tocar, se ha tocado, y todos los pasos han servido para añadir valor al proyecto.

11.5. Trabajo futuro

Durante la última etapa del proyecto, desde el departamento de electrónica y comunicaciones de Tekniker, se está trabajando en el diseño de una nueva placa interfaz para conectar a ella la placa de desarrollo utilizada.

Esto permite poder montar la Terasic DE10-Nano sobre la placa interfaz y dejarla fijada. Para ello habrá que ver los nuevos esquemáticos de la tarjeta interfaz y adecuar los pines de la tarjeta de desarrollo para que coincidan con las especificaciones.

Por otra parte, el desarrollo de la aplicación continuará, y se añadirán nuevos periféricos. Esto supone seguir con el desarrollo de *drivers*, y la integración de módulos de control y monitorización en HDL y software.

También se ha visto en el presente documento cómo es recomendable adaptar las señales de la FPGA con resistencias terminadoras, esto se va a tener en cuenta a la hora de hacer el nuevo diseño de la placa interfaz.

La aplicación de testeo y puesta en marcha del inversor, va a servir para estudiar el comportamiento de éste con los nuevos materiales semiconductores basados en SiC, y su viabilidad dentro del tejido industrial. Es una forma de generar el conocimiento tecnológico necesario sobre una nueva generación de convertidores, que cuentan con los semiconductores más prometedores del futuro.

Referencias

[1] IEEE Press, «The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition,» 2000, p. 588.

[2] J. Eriksson, «DC to AC inverter card with IGBTs,» Uppsala Universitet, Uppsala, 2012.

[3] A. Ahamed y F. Banu, «FPGA based control single phase three level inverter,» *International Journal of Research in Engineering & Technology,* vol. 5, nº 6, pp. 1-18, 2017.

[4] A. Elasser y T. P. Chow, «Silicon carbide benefits and advantages for power electronics circuits and systems,» *Proceedings of the IEEE,* vol. 90, nº 6, 2002.

[5] E. L. Owen, «History [origin of the inverter],» *IEEE Industry Applications Magazine,* vol. 2, nº 1, pp. 64-66, 1996.

[6] J. E. Britain, «Vacuum tube electronics: 1913-1930,» *Advances in Electronics and Electron Physics,* vol. 50, pp. 416-430, 1980.

[7] B. G. Lipták, Instrument Engineers' Handbook,(Volume 2) Third Edition: Process Control, 1995.

[8] T. Kobayashi, H. Abe, . Y. Niimura , T. Yamada, A. Kurosaki, T. Hosen y T. Fujihira, «High-voltage power MOSFETs reached almost to the silicon limit,» de *Proceedings of the 13th International Symposium on Power Semiconductor Devices & ICs*, Osaka, 2001.

[9] X. Wen, T. Fan, P. Ning y Q. Guo, «Technical ApproachesTowards Ultra-High Power Density SiC Inverterin Electric Vehicle Applications,» *CES Transactions on Electrical Machines and Systems,* vol. 1, nº 3, pp. 231-237, 2017.

[10] M. Macellari, A. Schiaratur y L. Schirone, «Digital Control of a Power Inverter,» Cambridge, 2009.

[11] G. Mattiussi y D. Zupancic, «The SiC revolution – reliable, efficient, & cost effective,» 2018. [En línea]. Available: https://www.infineon.com/dgdl/Infineon-Article_SiliconCarbide_The_SiC_revolution_PSD_Mattiussi_Zupancic-ART-v01_00-EN.pdf?fileId=5546d46261ff5777016205252a8d7d01.

[12] T. McDonald, «Impact of Commercialization of GaN based Power Devices on PV Solar Power Generation,» ARPA-E (United States Department of Energy), 2011.

[13] B. J. Baliga, Advanced High Voltage Power Device Concepts, NY Springer, 2012.

[14] K. Li, «Wide Bandgap (SiC/GaN) Power Devices Characterization and Modeling: Application to HF Power Converters,» Université Lille Nord-de-France, 2014.

[15] C. Zhu, «High-Efficiency High-Density GaN-Based 6.6kW Bidirectional On-board Charger for PEVs,» ARPA-E (United States Department of Energy), 2015.

[16] K. Itoh, S. Inoue, M. Ishigaki, T. Sugiyama y M. Sugai, «Loss Estimation of an Isolated Three-port DC-DC Converter for Automotive Applications,» de *Energy Conversion Congress and Exposition (ECCE)*, Montreal, 2015.

[17] S. Haghbin, «Design considerations of a 50 kW compact fast charger stations using nanocrystalline magnetic materials and SiC modules,» de *11th International Conference on Ecological Vehicles and Renewable Energies (EVER)*, Mónaco, 2016.

[18] S. Srdic, C. Zhang, X. Liang, W. Yu y S. Lukic, «A SiC-based power converter module for medium-voltage fast charger for plug-in electric vehicles,» de *IEEE Applied Power Electronics Conference and Exposition (APEC)*, Long Beach, CA, 2016.

[19] Hitachi, «Hitachi-Rail,» 2012. [En línea]. Available: http://www.hitachi-rail.com/products/on-board/sic/.

[20] European Commission, «Shift2Rail: driving innovation on railways,» 2014. [En línea]. Available: https://ec.europa.eu/research/press/jti/factsheet_s2r-web.pdf.

[21] F. Blaabjerg, D. Anca, F. Iov y L. H. Hansen, «Review of contemporary wind turbine concepts and their market penetration,» *Wind Engineering,* vol. 28, nº 3, pp. 247-263, 2004.

[22] H. Zhang y L. M. Tolbert, «Efficiency Impact of Silicon Carbide Power Electronics for Modern Wind Turbine Full Scale Frequency Converter,» *IEEE Transactions on Industrial Electronics,* vol. 58, nº 1, pp. 21-28, 2011.

[23] J. Colmenares, D.-P. Sadik, P. Hilber y H.-P. Nee, «Reliability Analysis of a High-Efficiency SiC Three-Phase Inverter,» *IEEE Journal of Emerging and Selected Topics in Power Electronics,* vol. 4, nº 3, pp. 996-1006, 2016.

[24] H. Zhang y L. M. Tolbert, «SiC's potential impact on the design of wind generation system,» de *34th Annual Conference of IEEE Industrial Electronics*, Orlando, 2008.

[25] T. Nozawa, «World's 1st SiC-based PV Inverter,» 2015. [En línea]. Available: https://tech.nikkeibp.co.jp/dm/english/NEWS_EN/20140731/368247/?ST=msbe&P=1.

[26] A. Oliva, J. Gonzalez-Llorente, L. Garcia-Rodriguez, V. Jones, J. C. Balda y E. Lindstrom, «Design of a GaN-based microinverter for photovoltaic systems,» de *5th International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, Galway, 2014.

[27] A. Morya, M. Moosavi, M. C. Gardner y H. A. Toliyat, «Applications of Wide Bandgap (WBG) devices in AC electric drives: A technology status review,» de *2017 IEEE International Electric Machines and Drives Conference (IEMDC)*, Miami, 2017.

[28] I. C. Kizilyalli, E. P. Carlson, D. W. Cunningham, J. S. Manser, Y. ". Xu y A. Y. Liu, «Wide Band-Gap Semiconductor Based Power Electronics for Energy Efficiency,» ARPA-E (United States Department of Energy), Washington, 2018.

[29] J. I. Tamboli, S. R. Jagtap y A. R. Sutar, «Pulse width modulation implementation using FPGA and CPLD ICs,» *International Journal of Scientific & Engineering Research,* vol. 3, nº 8, pp. 1-5, 2012.

[30] N. Waghmare y T. Mohite-Patil, «Comparison of Embedded System and FPGA Application for Universal Multi-Resource Bus,» *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. 1, nº 7, pp. 23-29, 2012.

[31] H. Chauhan, «Intel Is Gaining Against Its Biggest Rival in This Lucrative Market,» 14 08 2018. [En línea]. Available: https://www.fool.com/investing/2018/08/14/intel-is-gaining-against-its-biggest-rival-in-this.aspx.

[32] K. Morris, «Xilinx vs. Altera: Calling the Action in the Greatest Semiconductor Rivalry,» 25 02 2014. [En línea]. Available: https://www.eejournal.com/article/20140225-rivalry/.

[33] O. Šćekić, «FPGA Comparative Analysis,» University of Belgrade-School of Electrical Engineerin, 2004.

[34] Intel-Altera, «Strategic Considerations for Emerging SoC FPGAs,» 2011. [En línea]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01157-embedded-soc.pdf.

[35] Intel, «Comparación técnica de los procesadores de la familia Sandy Bridge,» [En línea]. Available: https://ark.intel.com/content/www/us/en/ark/compare.html?productIds=53478,52237,53476,50067,54644,53472,61275,54617,53469,53470,54615,54642,54611,53464,54618,53463,52219,54610,52231,54616,54645,52215,52214,52213,54620,65647,50072,54619,52229,54647,53456,5.

[36] Intel, «Comparación técnica de los procesadores de la familia Ivy Bridge,» [En línea]. Available: https://ark.intel.com/content/www/us/en/ark/compare.html?productIds=71096,64887,70846,64889,65525,65524,65523,65719,70847,64891,72015,71258,64898,71460,71670,71458,71459,64900,65709,67356,64901,65710,64899,65711,65712,71255,72054,64893,65713,65714,65704,6.

[37] M. Ruíz y A. Carpeno, «Using Quartus and Buildroot for building Embedded Linux Systems with DE1-SoC,» E.T.S.I. y Sistemas de Telecomunicación (UPM), Madrid, 2017.

[38] P. P. Chu, Embedded SoPC Design with Nios II Processor and VHDL Examples, Wiley, 2011.

[39] Hardwarebee, «How to Choose an FPGA for your Design,» 09 11 2018. [En línea]. Available: http://hardwarebee.com/choose-fpga-design/.

[40] A. Hunt y D. Thomas, The Pragmatic Programmer, Addison Wesley, 1999, pp. 171-172.

[41] M. A. Tétrault, «Two FPGA Case Studies Comparing High Level Synthesis and Manual HDL for HEP applications,» de *21st IEEE NPSS Real Time Conference*, Williamsburg, 2016.

[42] M. Pelcat, C. Bourrasset, L. Maggiani y F. Berry, «Design Productivity of a High Level Synthesis Compilerversus HDL,» de *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, Agios Konstantinos, 2016.

[43] M. D. Zwagerman, «High Level Synthesis, a Use Case Comparison with Hardware Description Language,» Grand Valley State University, Allendale, 2015.

[44] Terasic, DE10-Nano: User Manual, 2017.

[45] Terasic, DE10-Nano: Getting Started Guide, 2017.

[46] NXP Semiconductors, «I2C-bus specification and user manual,» 2014. [En línea]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[47] NXP, «OM13488 8-bit GPIO Daughter Card User Manual,» 2013. [En línea]. Available: https://www.nxp.com/docs/en/user-guide/UM10751.pdf.

[48] NXP, «PCA9538: A Low-voltage 8-bit I2C-bus I/O port with interrupt and reset,» 2012. [En línea]. Available: https://www.nxp.com/docs/en/data-sheet/PCA9538A.pdf.

[49] Resistor Guide, «Pull up resistor / Pull down resistor,» [En línea]. Available: http://www.resistorguide.com/pull-up-resistor_pull-down-resistor/.

[50] Motorola, Inc., «SPI Block Guide,» 2000. [En línea]. Available: https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf.

[51] Analog Devices, «AD7091 Datasheet,» 2012. [En línea]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad7091.pdf.

[52] Analog Devices, «AD7091R User Guide,» 2012. [En línea]. Available: https://www.analog.com/media/en/technical-documentation/user-guides/ug-409.pdf.

[53] Ecured, «Sistema Operativo de Tiempo Real,» [En línea]. Available: https://www.ecured.cu/Sistema_Operativo_de_Tiempo_Real.

[54] D. W. Harder, J. Zarnett, V. Montaghami y A. Giannikouris, A practical introduction to real-time systems for undergraduate engineering, Waterloo: University of Waterloo, 2018.

[55] A. Silberschatz, P. B. Galvin y G. Gagne, Operating System Concepts, Addison-Wesley, 2015.

[56] IBM, «What are mainframe operating systems?,» 2008. [En línea]. Available: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zmainframe_book.pdf.

[57] BrokenThorn, «Operating Systems Development - Kernel: Basic Concepts Part 1,» 2009. [En línea]. Available: http://www.brokenthorn.com/Resources/OSDev12.html.

[58] J. Carretero, F. García, P. De Miguel y F. Pérez, Sistemas Operativos: Una visión aplicada, McGraw-Hill, 2001.

[59] D. P. Bovet y M. Cesati, Understanding the Linux Kernel: From I/O Ports to Process Management, O'Reilly Media, Inc., 2005.

[60] W. Stallings, Sistemas operativos: Aspectos internos y principios de diseño, Pearson Educación, 2005.

[61] V. Bridgers, «Altera SoC LTSI Kernel with Preempt-RT,» 24 03 2015. [En línea]. Available: https://rocketboards.org/foswiki/Documentation/AlteraSoCLTSIRTKernel.

[62] Intel, «Compiling u-boot and Linux Kernel for Cyclone V SoC,» 7 03 2013. [En línea]. Available: https://fpgawiki.intel.com/wiki/Compiling_u-boot_and_Linux_Kernel_for_Cyclone_V_SoC.

[63] Arch Linux, «Realtime kernel patchset,» 2018. [En línea]. Available: https://wiki.archlinux.org/index.php/Realtime_kernel_patchset.

[64] P. J. Salzman, M. Burian y O. Pomerantz, «The Linux Kernel Module Programming Guide,» 2007. [En línea]. Available: http://tldp.org/LDP/lkmpg/2.6/html/index.html.

[65] P. Favrod, «How to configure Linux to receive IRQs from the FPGA?,» 14 03 2016. [En línea]. Available: https://wiki.epfl.ch/prsoc/documents/Cyclone_V_SoC_Linux_Interrupt-2.pdf.

[66] T. Petazzoni, «Your new ARM SoC Linux support check-list!,» de *Embedded Linux Conference Europe*, Barcelona, 2012.

[67] H. Mao, «Exploring the Arrow SoCKit Part X - Sending and Handling Interrupts,» 24 05 2014. [En línea]. Available: https://zhehaomao.com/blog/fpga/2014/05/24/sockit-10.html.

[68] D. Bishop, «Floating point package user's guide,» 2016.

[69] IEEE, «IEEE 1076-2008 - IEEE Standard VHDL Language Reference Manual,» 2009.

[70] Intel, «Signal Integrity Basics,» [En línea]. Available: https://www.intel.com/content/www/us/en/programmable/support/support-resources/support-centers/signal-power-integrity/learn.html.

[71] Anandtech, «Intel shows Xeon Scalable Gold 6138P with integrated FPGA,» 17 05 2018. [En línea]. Available: https://www.anandtech.com/show/12773/intel-shows-xeon-scalable-gold-6138p-with-integrated-fpga-shipping-to-vendors.

[72] A. Gorobets, F. X. Trias y A. Oliva, «A parallel MPI + OpenMP + OpenCL algorithm for hybrid supercomputations of incompressible flows,» *Computers & Fluids,* vol. 88, pp. 764-772, 2013.

Anexo A: Listado de proyectos generados

Para el desarrollo final de este proyecto, se han ido generando pequeños bloques funcionales. Algunos de ellos tenían el cometido de probar el funcionamiento de alguna parte de la FPGA, otros, han sido reutilizados en la implementación del prototipo final.

Este proyecto incluye parte del código generado. Se ha ordenado por fases y se ha recopilado con la fecha de finalización y su ruta.

Lógica Programable

Estos proyectos se corresponden a la fase de lógica programable. Como se ha explicado en la memoria, en esta etapa se probó el funcionamiento de la placa para la generación y adquisición de señales. La fase concluye con los módulos de I2C y SPI, que se incluyen en la aplicación final.

Módulo	Ruta	Fecha Fin
pruebaLeds	/fpga/pruebaLeds	02/10/2018
pruebaLedsCont	/fpga/pruebaLedsCont	02/10/2018
pruebaKey	/fpga/pruebaKey	03/10/2018
pruebaSumaKey	/fpga/pruebaSumaKey	03/10/2018
pruebaEscrituraGPIO	/fpga/pruebaEscrituraGPIO	04/10/2018
pruebaLecturaGPIO	/fpga/pruebaLecturaGPIO	04/10/2018
sistemaGPIO	/fpga/sistemaGPIO	04/10/2018
pruebaRW-GPIO	/fpga/pruebaRW-GPIO	05/10/2018
pruebaOnBoardADC	/fpga/pruebaOnBoardADC	05/10/2018
pruebaPWM	/fpga/pruebaPWM	08/10/2018
pruebaQsys	/fpga/pruebaQsys	25/10/2018
pruebaADC_Qsys	/fpga/pruebaADC_Qsys	29/10/2018
pruebaAnalogPWM	/fpga/pruebaAnalogPWM	29/10/2018
pruebaADC_IP	/fpga/pruebaADC_IP	31/10/2018
pruebaI2C_Write	/fpga/pruebaI2C_Write	14/11/2018
pruebaI2C_Write	/fpga/pruebaI2C_Write	16/11/2018
pruebaI2C_RW	/fpga/pruebaI2C_RW	23/11/2018
pruebaSPI_RW	/fpga/pruebaSPI_RW	12/12/2018

Tabla A.1. Listado de proyectos sobre lógica programable.

Interrupciones

Este apartado se corresponde a las pruebas que se hicieron para aprender el tratamiento de interrupciones entre la FPGA y el HPS. Se generaron dos proyectos:

Módulo	Ruta	Fecha Fin
pruebaInterrupcion	/IRQ/pruebaLeds	07/03/2019
pruebaInterrupcionSWLed	/IRQ/pruebaLedsCont	08/03/2019

Tabla A.2. Listado de proyectos sobre la gestión de interrupciones.

SoC-FPGA

Aquí se muestra el conjunto de pruebas que se hicieron para conocer el funcionamiento de una arquitectura SoC-FPGA. Finalmente, concluyó en el desarrollo de diversas aplicaciones, y en los módulos de control y monitorización de la aplicación final

Módulo	Ruta	Fecha Fin
demo_led	/soc-fpga/demo_led	15/01/2019
demo_sw	/soc-fpga/demo_sw	24/01/2019
demo_led_sw_FPGA	/soc-fpga/demo_led_sw_FPGA	24/01/2019
demo_led_sw_hps	/soc-fpga/demo_led_sw_hps	24/01/2019
hps_adc	/soc-fpga/hps_adc	25/01/2019
hps_adc_IPCore	/soc-fpga/hps_adc_IPCore	28/01/2019
hps_fpga_key	/soc-fpga/hps_fpga_key	13/02/2019
pingPong	/soc-fpga/pingPong	18/02/2019
pingPong_automatizado	/soc-fpga/pingPong_automatizado	18/02/2019
spwm_gen_soc	/soc-fpga/spwm_gen_soc	01/04/2019
PWM_Simple_Configurable_Retardo	/soc-fpga/PWM_ Conf_Retardo	09/04/2019
spwm_gen_soc_variable	/soc-fpga/spwm_gen_soc_var	08/05/2019
amp_volt_ctrl_soc	/soc-fpga/amp_volt_ctrl_soc	07/05/2019

Tabla A.3. Listado de proyectos implementados sobre SoC-FPGA.

El código de la aplicación final no se incluye en esta lista, pero se puede ver la generación del PWM, SPWM y el control y monitorización de la corriente y tensión, que se corresponden a los tres últimos proyectos de la tabla. Para la aplicación final se hizo una unión de esos tres proyectos.