

Grado en Ingeniería Informática
2024-2025

Trabajo Fin de Grado

Microcontrolador endurecido para aplicaciones espaciales

Darío Caballero Polo

Tutora

Almudena Lindoso Muñoz
Leganés, septiembre de 2025



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

Los sistemas electrónicos destinados a ser utilizados en entornos con partículas ionizantes han de estar adaptados para resistir los efectos de estas. El espacio es uno de estos entornos, por lo que un microcontrolador diseñado para una misión espacial debe disponer de mecanismos para detectar, corregir o incluso enmascarar los errores que inevitablemente acontecerán.

En este trabajo se implementará en la FPGA presente en la placa PYNQ-Z1 un sistema tolerante a fallos basado en el procesador *softcore* MicroBlaze-V. Para ello, se utilizarán el subsistema TMR para MicroBlaze y el núcleo IP LogiCORE SEM. El objetivo principal es determinar si un sistema de estas características se puede configurar en una FPGA con un número moderado de células lógicas. Además, se comprobará la correcta compatibilidad del subsistema TMR para MicroBlaze con la versión RISC-V del *softcore*. Las herramientas *software* utilizadas serán la *Suite* de Diseño Vivado, la plataforma Vitis y el proyecto de código abierto *Project X-Ray*. Todo el proceso de desarrollo desde el diseño inicial hasta la programación del dispositivo y su depuración se hará siguiendo la metodología UltraFast de AMD. Se realizarán pruebas para comprobar el correcto funcionamiento del sistema y se analizarán las métricas de implementación. Para las pruebas relativas al subsistema TMR, se presentarán procedimientos para la inyección de errores en bits de la memoria de configuración asociados a *cells* primitivas específicas deseadas.

Los resultados muestran que el sistema realiza una utilización reducida de los recursos del dispositivo, y que es capaz de detectar, corregir y enmascarar los fallos de acuerdo a las especificaciones del subsistema TMR para MicroBlaze y el núcleo IP LogiCORE SEM.

Palabras clave: Tolerancia a fallos; Endurecimiento frente a radiación; Redundancia; TMR; SEM; SEU; FPGA; SoC; PYNQ-Z1; MicroBlaze; RISC-V; Espacio; Inyección de errores; Xilinx.

ABSTRACT

Electronic systems made for environments filled with ionizing particles must be adapted to be able to resist its effects. Space is one of those environments, so a microcontroller designed for a space mission must have the mechanisms for detecting, correcting and even masking the errors that will eventually happen.

In this thesis, a fault-tolerant system based on the softcore processor MicroBlaze-V will be implemented in the FPGA of the PYNQ-Z1 board. To do that, the TMR subsystem for MicroBlaze will be used in conjunction with the LogiCORE SEM IP. The main objective is to determine if a system of those characteristics can be configured on a FPGA with a moderate count of logic cells. Furthermore, the correct compatibility of the TMR subsystem for MicroBlaze with the RISC-V version of the softcore will be checked. The software tools used in the project will be the Vivado design suite, the Vitis platform and the open source project named *Project X-Ray*. The entire development process, from the initial design to the programming of the device and its debugging, will be performed by following the AMD UltraFast methodology. Various tests will be performed to ascertain the correct behaviour of the system and the implementation metrics will be analysed. As for the TMR-related tests, some procedures will be presented for injecting errors in configuration memory bits linked to specific desired primitive cells.

Results show that the system makes a reduced utilization of the device resources and is capable of detecting, correcting and masking the errors according to the TMR subsystem for MicroBlaze and LogiCORE SEM IP specifications.

Keywords: Fault-tolerance, Radiation-hardening, Redundancy, TMR, SEM, SEU, FPGA, SoC, PYNQ-Z1, MicroBlaze, RISC-V, Space, Error injection, Xilinx.

DEDICATORIA

Querría dedicarle este espacio a Cande, por todo el apoyo y cariño que me ha regalado durante tantos años.

ÍNDICE GENERAL

1. INTRODUCCIÓN	1
1.1. Antecedentes y motivación	1
1.2. Objetivos	2
1.3. Alcance y limitaciones.	2
1.4. Marco regulador	2
2. ESTADO DE LA CUESTIÓN	5
2.1. Microcontroladores en aplicaciones espaciales	5
2.2. Técnicas de tolerancia a fallos en FPGAs	6
2.2.1. Mitigación de <i>Soft Errors</i> (SEM)	6
2.2.2. Técnicas de protección de memorias basadas en códigos	7
2.2.3. Redundancia Modular Triple (TMR).	8
2.3. RISC-V	13
2.4. Procesadores softcore	15
2.4.1. LEON	15
2.4.2. NOEL-V	16
2.4.3. MicroBlaze	16
2.5. PYNQ-Z1 y Zynq-7000	17
3. METODOLOGÍA Y DISEÑO	21
3.1. Requisitos del sistema para aplicaciones espaciales.	21
3.2. <i>Hardware</i> utilizado.	22
3.3. <i>Suite</i> de diseño Vivado: herramientas, IPs y flujo de trabajo	23
3.3.1. IP <i>Integrator</i>	26
3.3.2. RTL <i>Analysis</i>	34
3.3.3. <i>Synthesis</i>	39
3.3.4. <i>Implementation</i>	40
3.4. Entorno de desarrollo de software: Vitis	48
3.4.1. Creación de componentes: plataforma y aplicaciones	49
3.4.2. Aplicación del Cortex-A9	50

3.4.3. Aplicación del subsistema TMR	52
3.4.4. Construcción, configuración y <i>debugging</i>	53
3.5. Metodología de pruebas	56
3.5.1. Simulación de fallos	56
3.5.2. Pruebas realizadas	72
4. RESULTADOS	77
4.1. Utilización de recursos.	77
4.2. Resultados de las pruebas	78
5. PLANIFICACIÓN Y PRESUPUESTO	87
5.1. Planificación	87
5.1.1. Planificación inicial	88
5.1.2. Planificación final	88
5.2. Presupuesto	89
5.2.1. Presupuesto inicial	89
5.2.2. Presupuesto final	90
6. CONCLUSIONES	91
6.1. Resumen de los hallazgos	91
6.2. Contribuciones	91
6.3. Cumplimiento de objetivos	91
6.4. Mejoras futuras.	92
6.4.1. Verificación del requisito RF-02	92
6.4.2. Mejora del ataque a bits específicos	92
6.4.3. Inyección de instrucciones	93
6.4.4. Simulación de los errores inducidos por un entorno radiactivo	93
6.4.5. Pruebas en un laboratorio de radiación.	93
6.4.6. Evaluación del SWaP-C de diferentes configuraciones del MicroBlaze-V .	94
6.4.7. Integración con sistemas satelitales reales	94
6.5. Impacto socioeconómico	94
6.5.1. Impacto económico	94
6.5.2. Impacto social	95
BIBLIOGRAFÍA	97

ÍNDICE DE FIGURAS

2.1	Diagrama de un sistema TMR con votantes simples	8
2.2	Círculo TMR	9
2.3	Fiabilidades de TMR con votante simple y triplicado para 4 valores de fiabilidad del votante	11
2.4	Fiabilidades de TMR con votante simple y triplicado para múltiples valores de fiabilidad del votante	12
2.5	Diagrama de un sistema TMR con votantes triplicados	12
2.6	ISA y procesadores en misiones espaciales de Europa y EEUU	14
2.7	Segmentación en 7 fases de NOEL-V	16
2.8	Fotografía de PYNQ-Z1	17
2.9	Diagrama de bloques de PYNQ-Z1	18
2.10	Diagrama de bloques del <i>overlay</i> base de PYNQ-Z1	19
2.11	Pines de configuración de PYNQ-Z1	20
2.12	Rutas de configuración de la lógica programable del Zynq-7000	20
3.1	Flujo de diseño AMD UltraFast	24
3.2	Navegador de flujo de Vivado	25
3.3	Subsistemas TMR MicroBlaze <i>Fault Tolerant</i>	28
3.4	Lógica de un Comparador TMR	29
3.5	Bloques seleccionados para triplicarse	30
3.6	Configuración de los controladores UART	33
3.7	Diagrama del puerto Pmod	37
3.8	Esquema del circuito de Pmod A y Pmod B	38
3.9	Conexión entre el módulo PmodUSBUART y Pmod A	38
3.10	Esquemas de la <i>netlists</i> RTL y sintetizada	39
3.11	Informe <i>Check Timing</i>	40
3.12	Jerarquía arquitectural de una FPGA de Xilinx	42
3.13	Mapeos entre elementos lógicos y físicos	42
3.14	Asignación física y ubicación en el dispositivo de una <i>cell</i> primitiva	43

3.15 Resumen del informe temporal del diseño conectado a una señal de reloj de 100 MHz	45
3.16 Proceso iterativo para lograr el cierre temporal	46
3.17 Resumen del informe temporal del diseño conectado a una señal de reloj de 50 MHz	46
3.18 Implementación en el dispositivo del diseño	47
3.19 Grado de utilización de cada recurso del dispositivo	48
3.20 Informe energético del diseño	48
3.21 Flujo de trabajo para el desarrollo de aplicaciones de software embebidas	49
3.22 Propiedades del bloque <i>axi_gpio_status_1</i>	52
3.23 Configuración de lanzamiento de la aplicación del Cortex-A9	54
3.24 Configuración de lanzamiento de la aplicación del subsistema TMR	55
3.25 Configuración de lanzamiento de la aplicación conjunta	56
3.26 BEL D6LUT mapeado a una <i>cell</i> del sub-bloque MB3	69
3.27 Utilización de la ayuda del script <i>tile2injection.py</i>	72
3.28 Flujo de las pruebas del subsistema TMR	73
3.29 Interfaces para la realización de las pruebas	76
 4.1 Comparación de utilización de recursos	77
4.2 Inyección de un error simple	82
4.3 Inyección de un error doble adyacente	83
4.4 Inyección de un error séxtuple distribuido de forma unitaria por <i>frames</i> distintos	84
4.5 Inyección de un error óctuple distribuido en parejas adyacentes por <i>frames</i> distintos	84
4.6 Inyección de un error doble adyacente localizado en dos palabras	85
4.7 Inyección de un error doble no adyacente	86
 6.1 Esquema de la inyección de instrucciones	93

ÍNDICE DE TABLAS

3.1	Formato de los requisitos	22
3.2	Utilización de recursos en SoCs Zynq-7000 para cada configuración del MicroBlaze-V	27
3.3	Clasificación de las <i>cells</i> primitivas en grupos, subgrupos y tipos	36
3.4	Valor del comando de inyección de errores en esquema de direccionamiento LFA	58
3.5	Valor del comando de inyección de errores en esquema de direccionamiento PFA	58
3.6	Posibles mensajes de informe de cambio de estado	59
3.7	Posibles mensajes de informe de cambio de <i>Flag</i>	60
3.8	Señal de estado del subsistema TMR	64
3.9	Comandos de inyección para atacar el BEL deseado	71
4.1	Relación del uso de cada recurso en el sistema endurecido con respecto al inicial	78
4.2	Resultados mayoritarios de las pruebas TMR	79
4.3	Ejemplo de inyección del test del requisito RF-07	81
4.4	Ejemplo de inyecciones del test del requisito RF-07	82
4.5	Ejemplo de inyecciones del test del requisito RF-08	83
4.6	Ejemplo de inyecciones del test del requisito RF-09	84
4.7	Ejemplo de inyecciones del test de un error doble adyacente localizado en dos palabras distintas	85
4.8	Ejemplo de inyecciones del test de un error doble no adyacente	86
5.1	Presupuesto inicial	90
5.2	Presupuesto final	90

LISTA DE ABREVIATURAS

AEE	Agencia Espacial Española
AMD	<i>Advanced Micro Devices, Inc.</i>
APSoC	<i>All Programmable System on a Chip</i>
APU	<i>Application Processing Unit</i>
ARM	<i>Advanced RISC Machine</i>
ASCII	<i>American Standard Code for Information Interchange</i>
ASIC	<i>Application Specific Integrated Circuit</i>
AXI	<i>Advanced eXtensible Interface</i>
BCH	<i>Bose–Chaudhuri–Hocquenghem</i>
BEL	<i>Basic Element of Logic</i>
BRAM	<i>Block Random Access Memory</i>
BUFG	<i>Global clock BUFFer</i>
CISC	<i>Complex Instruction Set Computer</i>
CLB	<i>Configurable Logic Block</i>
COTS	<i>Commercial Off-The-Shelf</i>
CRC	<i>Cyclic Redundancy Check</i>
CTS	<i>Clear To Send</i>
DSP	<i>Digital Signal Processing</i>
ECC	<i>Error Correction Code</i>
ECSS	<i>European Cooperation for Space Standardization</i>
ELF	<i>Executable and Linkable Format</i>
EMIO	<i>Extended Multiplexed I/O</i>
ESA	<i>European Space Agency</i>
FASM	<i>FPGA Assembly</i>
FF	<i>Flip-Flop</i>
FPGA	<i>Field-Programmable Gate Array</i>
FS	<i>Fail Safe</i>
FSR	<i>Fabric Sub Region</i>
FT	<i>Fault Tolerant</i>
GND	<i>GrouND</i>
GPIO	<i>General Purpose Input/Output</i>
HDL	<i>Hardware Description Language</i>
ICAP	<i>Internal Configuration Access Port</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
I/O	<i>Input/Output</i>
IOB	<i>Input/Output Block</i>
IOP	<i>Input/Output Peripherals</i>
IP	<i>Intellectual Property</i>

IPython	<i>Interactive Python</i>
ISA	<i>Instruction Set Architecture</i>
LED	<i>Light-Emitting Diode</i>
LFA	<i>Linear Frame Address</i>
LMB	<i>Local Memory Bus</i>
LUT	<i>Look-Up Table</i>
LUTRAM	<i>Look-Up Table Random Access Memory</i>
MF	<i>Maximum Frame</i>
MIO	<i>Multiplexed I/O</i>
MMCM	<i>Mixed-Mode Clock Manager</i>
NASA	<i>National Aeronautics and Space Administration</i>
ODS	<i>Objetivos de Desarrollo Sostenible</i>
PC	<i>Program Counter</i>
PCAP	<i>Processor Configuration Access Port</i>
PFA	<i>Physical Frame Address</i>
PIP	<i>Programmable Interconnect Point</i>
PL	<i>Programmable Logic</i>
Pmod	<i>Peripheral module</i>
PS	<i>Processing System</i>
PYNQ	<i>PYthon productivity for zyNQ</i>
RAM	<i>Random Access Memory</i>
RF	<i>Requisito Funcional</i>
RISC-V	<i>Reduced Instruction Set Computer - Five</i>
RN	<i>Requisito No funcional</i>
RTL	<i>Register Transfer Level</i>
RTS	<i>Request To Send</i>
Rx	<i>Receiver</i>
SDC	<i>Synopsys Design Constraints</i>
SEFI	<i>Single-Event Functional Interrupt</i>
SEM	<i>Soft Error Mitigation</i>
SEU	<i>Single Event Upset</i>
SLR	<i>Super Logic Region</i>
SoC	<i>System-on-Chip</i>
SPARC	<i>Scalable Processor ARChitecture</i>
SRAM	<i>Static Random Access Memory</i>
SSI	<i>Stacked Silicon Interconnect</i>
SST	<i>SEUs Simulation Tool</i>
SWaP-C	<i>Size, Weight, Power and Cost</i>
TMR	<i>Triple Modular Redundancy</i>
TNS	<i>Total Negative Slack</i>
TTEthernet	<i>Time-Triggered Ethernet</i>
Tx	<i>Transmitter</i>

UART	<i>Universal Asynchronous Receiver-Transmitter</i>
UNE	<i>Una Norma Española</i>
URSO	<i>Union Registry of Space Objects</i>
USB	<i>Universal Serial Bus</i>
VCC	<i>Voltage Common Collector</i>
VHDL	<i>VHSIC (Very High Speed Integrated Circuit) HDL</i>
WNS	<i>Worst Negative Slack</i>
XADC	<i>Xilinx Analog to Digital Converter</i>
xdc	<i>Xilinx Design Constraints</i>
XSA	<i>Xilinx Support Archive o Xilinx Shell Archive</i>

1. INTRODUCCIÓN

En este capítulo se expondrá la problemática abarcada por el trabajo y su motivación. Además, se presentarán los objetivos del proyecto y sus limitaciones. Por ultimo, se establecerá el marco regulador en el que se encuadra.

1.1. Antecedentes y motivación

Los efectos de la radiación sobre circuitos electrónicos han sido objeto de estudio durante décadas: desde rayos cósmicos provocando anomalías en satélites de comunicación [1] a los efectos de la atmósfera sobre la aviación [2]. Estos errores inducidos por el impacto de una partícula ionizante se denominan *Single Event Upsets* (SEUs) [3]. La primera teorización sobre la posibilidad de la ocurrencia de un SEU se produjo en 1962 [4], y desde entonces ha quedado demostrada la relevancia de este fenómeno [5]. La sonda Cassini, lanzada en octubre de 1997, sufrió alrededor de 280 errores de un bit diarios durante los primeros dos años y medio [6]. Los SEUs se producen por dos mecanismos: deposición de carga y recolección de carga. A su vez, la deposición de carga se puede dar mediante la ionización directa por la partícula entrante o por ionización indirecta, consecuencia de partículas secundarias creadas por reacciones nucleares entre la partícula entrante y el dispositivo atacado [5].

En este trabajo se utilizará una matriz de puertas lógicas programable en campo (FPGA—*Field-Programmable Gate Array*) para implementar el diseño del microcontrolador. Las FPGA son circuitos integrados reprogramables. Esto es, se puede reescribir su configuración —generalmente mediante lenguajes de descripción de *hardware* (HDL) como VHDL o Verilog— para modificar la lógica implementada por el *hardware*. Para ello, establecen una matriz de bloques lógicos configurables (CLBs —Configurable Logic Blocks—) interconectada por una red reprogramable. Tanto los bloques como la interconexión se controlan mediante células de memoria, generalmente de tipo SRAM —pues permiten un gran número de ciclos de reconfiguración— [7]. Un impacto de una partícula cargada sobre alguno de los transistores que componen una célula de memoria SRAM puede inducir corrientes en los demás transistores de la célula que acaben derivando en la inversión del bit guardado [5]. Como las FPGAs son tan susceptibles a los SEUs, es necesario implementar mecanismos de tolerancia a fallos tanto en *software* como en *hardware*, especialmente en entornos sensibles como la astronáutica [1] y la aeronáutica [2]. En este trabajo se explorará una solución a nivel de *hardware*.

1.2. Objetivos

El objetivo principal de este trabajo es diseñar un microcontrolador apto para aplicaciones espaciales, lo que conlleva dotarlo de mecanismos de tolerancia a fallos. En concreto, se quiere explorar la viabilidad de un microcontrolador redundante basado en la arquitectura de conjunto de instrucciones (ISA) RISC-V utilizando el procesador *softcore* MicroBlaze-V. Se quiere determinar si es posible programar dicho diseño en la lógica programable de una placa PYNQ-Z1, con un número moderado de células lógicas. Por último, se pretende hacer de esta memoria una guía accesible para quienes quieran construir sistemas redundantes similares, concentrando toda la información básica necesaria en un mismo lugar.

1.3. Alcance y limitaciones

Este trabajo abarca la creación de un prototipo de microcontrolador endurecido. Por lo tanto, si bien se construirá un sistema acorde al objetivo principal, no se realizarán las pruebas exhaustivas que se requerirían para un despliegue real. Se plantearán las bases para hacer pruebas de inyecciones de errores localizadas, pero no se simularán los efectos de un entorno radiactivo ni se harán ensayos en un laboratorio de radiación.

1.4. Marco regulador

Un proyecto de índole espacial ha de ajustarse rigurosamente a los estándares y legislación vigentes para garantizar el éxito de misiones cuyo fallo puede derivar en accidentes fatales y grandes pérdidas económicas.

A nivel nacional, la Asociación Española de Normalización (UNE —Una Norma Española—), plantea multitud de normas relativas a la industria aeroespacial. Estas son elaboradas por el comité CTN 28, el cual ha creado 3878 de ellas. El comité se compone de las secciones SC1 (Equipo de tierra para aeronaves), SC 2 (Sistemas aéreos no tripulados) y SC 3 (Sistemas aeroespaciales). Así, el comité más relevante para este caso es el CTN 28/SC3, que ha redactado 3610 normas, entre las que se destacan las siguientes [8]:

- **UNE-EN 16603-20-40:2023** Ingeniería espacial: Ingeniería ASIC, FPGA e IP Co-re.
- **UNE-CEN/TR 17602-60-02:2021** Aseguramiento de los productos espaciales: Manual de técnicas para la mitigación de los efectos de la radiación en ASIC y FPGA.
- **UNE-EN 16602-60-02:2014** Aseguramiento de los productos espaciales: Desarro-llo ASIC y FPGA.

A nivel europeo, la Cooperación Europea para la Estandarización del Espacio (ECSS —*European Cooperation for Space Standardization*—), una iniciativa conjunta de la industria espacial europea, establece los estándares para las actividades espaciales europeas. La Agencia Espacial Europea (ESA —European Space Agency—) juega un papel central en la ECSS y adopta sus estándares [9]. En lo relativo a este proyecto, existen estándares sobre el desarrollo y aseguramiento de sistemas en FPGAs, así como de aspectos relativos a la radiación:

- **ECSS-E-ST-20-40C ASIC, FPGA and IP Core engineering** [10].
- **ECSS-Q-ST-60-03C ASIC, FPGA and IP Core product assurance** [11].
- **ECSS-E-ST-10-12C Methods for the calculation of radiation received and its effects, and a policy for design margins** [12].
- **ECSS-Q-ST-60-15C Rev.1 Radiation hardness assurance** [13].

En relación a los lenguajes de descripción de *hardware* utilizados para configurar FPGAs, el IEEE plantea estándares sobre los mismos:

- **IEEE 1076-2019 IEEE Standard for VHDL Language Reference Manual** [14].
- **IEEE 1800-2023 IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language** [15].

Más allá de los estándares, se han iniciado procesos para la elaboración de legislaciones específicas para el espacio tanto en España como en la Unión Europea. El 25 de junio de 2025, la Comisión Europea presentó una Propuesta de Reglamento para la seguridad, resiliencia y sostenibilidad de las actividades espaciales en la Unión (procedimiento 2025/0335/COD). Así, se busca establecer una base normativa espacial conjunta a nivel europeo que garantice la seguridad y facilite la transmisión de información entre países. Un ejemplo de este último punto es la creación de un registro de objetos espaciales de la Unión (URSO —Union Registry of Space Objects—) que unifique los registros que hasta ahora se realizan a nivel nacional. En España, el Real Decreto 278/1995, de 24 de febrero, por el que se crea en España el Registro previsto en el Convenio de 12 de noviembre de 1974 de la Asamblea General de las Naciones Unidas, establece la creación de un Registro de Objetos Lanzados al Espacio Ultraterrestre de aquellos objetos cuyo lanzamiento se haya realizado o haya sido promovido por el Estado Español, o de aquellos que se hayan lanzado desde España o instalaciones españolas. Asimismo, establece los mecanismos de comunicación con las Naciones Unidas para informar sobre las adiciones al registro español de nuevos objetos, que han de ser añadidos al registro de las Naciones Unidas. La propuesta de la Comisión Europea, sin embargo, ha de pasar por el Parlamento Europeo y el Consejo Europeo, por lo que se prevé que entre en vigor el 1 de enero de 2030, con una implementación efectiva 2 años después: el 1 de enero de 2032. En el caso de España, el

Real Decreto 158/2023, de 7 de marzo, por el que se aprueba el Estatuto de la Agencia Espacial Española (AEE), indica que uno de los objetivos de la AEE es la elaboración de una propuesta de anteproyecto de Ley del Espacio, de la que todavía no se conocen detalles.

2. ESTADO DE LA CUESTIÓN

El conocimiento previo del campo en el que se trabaja es condición *sine qua non* para la realización de un buen trabajo. Por ello, este capítulo realizará un repaso del estado de la técnica de diferentes aspectos y tecnologías implicados en el proyecto.

2.1. Microcontroladores en aplicaciones espaciales

El uso de microcontroladores en sistemas espaciales ha evolucionado significativamente en las últimas décadas, impulsado por la necesidad de lograr operaciones autónomas, prolongadas y resilientes en entornos extremos donde la intervención humana es imposible o limitada. A diferencia de otros dominios como la aviación comercial, donde los sistemas pueden ser mantenidos regularmente, los sistemas embarcados en naves espaciales deben operar sin fallos durante meses o incluso años, sin posibilidad de reparación física directa y con un gasto energético mínimo. En concreto, están limitados por tamaño, peso, energía y coste (SWaP-C —*Size, Weight, Power and Cost*—) [16]. Esto impone requisitos rigurosos en cuanto a la fiabilidad, tolerancia a fallos y eficiencia energética de los microcontroladores utilizados [17].

En las aplicaciones espaciales, los microcontroladores forman parte de subsistemas críticos como propulsión, energía, telecomunicaciones, control de actitud y manejo de datos. Cada subsistema está diseñado con redundancia interna, y se integran estrategias de tolerancia a fallos para mitigar errores tanto transitorios como permanentes. Esta redundancia puede manifestarse a nivel de componentes, módulos o incluso sistemas completos, como en el caso de la Redundancia Modular Triple (TMR), donde tres instancias idénticas del sistema procesan la misma información y se utiliza lógica de votación para determinar el resultado correcto.

Un aspecto clave del diseño de estos sistemas es el concepto de «modo seguro», un estado al que se transfiere la nave automáticamente tras la detección de un fallo grave que requiere intervención remota [18]. En este modo, los microcontroladores desconectan cargas no esenciales, orientan los paneles solares para maximizar la obtención de energía, y esperan instrucciones desde tierra para reconfigurar o reiniciar sistemas [17].

La tendencia moderna tras el advenimiento de la filosofía del *NewSpace* es el uso de procesadores comerciales modificados (COTS —*Commercial Off-The-Shelf*—) o de estándares abiertos junto con técnicas avanzadas de tolerancia a fallos, con el fin de reducir costes sin comprometer la fiabilidad. Sin embargo, la adopción de COTS implica desafíos potencialmente no considerados durante el ciclo de desarrollo del producto —concebido para aplicaciones terrestres— [17]. El *NewSpace* surge tras la emergencia de empresas privadas en el sector con una marcada vocación comercial que les empuja hacia

la reducción de costes y la adopción de metodologías ágiles [19].

Recapitulando, los microcontroladores utilizados en el ámbito espacial constituyen una parte central de las misiones modernas, y deben cumplir simultáneamente con criterios de alta fiabilidad, capacidad de recuperación y eficiencia energética. Su papel es crucial para garantizar que las misiones cumplan sus objetivos científicos y operacionales incluso en presencia de eventos adversos.

2.2. Técnicas de tolerancia a fallos en FPGAs

Las técnicas *hardware* de mitigación de SEUs se articulan en tres enfoques: a nivel de sistema, a nivel de circuito y a nivel de tecnología. Las técnicas a nivel de sistema abordan el diseño de la arquitectura. Aquellas a nivel de circuito modifican el diseño físico del mismo para reducir la sensibilidad SEU. Por último, aquellas a nivel de tecnología implican un cambio en los métodos de fabricación del circuito integrado [5].

Dada una FPGA, no se pueden aplicar técnicas a nivel de tecnología, pues el diseño que se programe en ella no modificará la tecnología de fabricación de su propio *hardware*. Las FPGA, sin embargo, son idóneas para investigar técnicas de mitigación de SEUs a nivel de sistema, ya que su reconfigurabilidad permite establecer arquitecturas de sistemas tolerantes a fallos. A su vez, se puede modificar el *floorplanning* de los componentes para lograr cierta mitigación a nivel de circuito. En este trabajo se explorará la mitigación a nivel de sistema planteando una arquitectura tolerante a fallos. Los dos sistemas más utilizados en esta categoría son los circuitos de detección y corrección de errores y la redundancia modular triple, que puede ser aplicada a nivel de circuito, placa o módulo [5]. Los códigos de detección y corrección de errores se utilizan para mitigar errores transitorios (*soft errors*), por lo que se catalogan como SEM (*Soft Error Mitigation*). En cambio, la TMR permite detectar y enmascarar el primer error permanente —no corregible por el SEM— que se sufra sin comprometer la disponibilidad del sistema. A su vez, también permite enmascarar los *soft errors* desde su acontecimiento hasta su corrección por técnicas SEM.

2.2.1. Mitigación de *Soft Errors* (SEM)

Unas de las principales técnicas para detectar y corregir *soft errors* son los códigos de corrección de errores (ECC —Error Correction Code—) y la verificación de redundancia cíclica (CRC —Cyclic Redundancy Check—), que se utilizan a menudo de forma combinada para maximizar la fiabilidad. Esto es así debido a que la CRC es robusta para la detección de errores y los ECC permiten identificar la posición del error con más precisión, además de poder corregirlo [20]. Las técnicas de SEM se utilizan en FPGA para mantener inalterada la memoria de configuración, la cual define el diseño programado en matriz. Esto se puede realizar mediante dos enfoques: reconfiguración y *scrubbing*.

La reconfiguración, en su forma más básica, consiste en reprogramar la FPGA al completo, parando la operatividad del sistema en el proceso, para cargar la configuración original tras la detección de un error [21]. Implementaciones más modernas logran lo mismo realizando reconfiguraciones parciales sin detener la actividad del sistema [22]. La reconfiguración permite corregir un número arbitrario de errores en posiciones arbitrarias, por lo que es muy efectiva [22]. Sin embargo, depende de que se tenga una copia externa [22] o redundante [21] de la configuración inicial. En caso de que esa copia se viese comprometida, el error se cargaría en la memoria de configuración sin ninguna forma de corregirlo. Para llevar a cabo esta técnica se utilizan códigos de detección de errores.

El *scrubbing* consiste en la comprobación periódica de la memoria de configuración para la detección y corrección de errores mediante el uso de ECC y, opcionalmente, códigos de detección de errores complementarios. La periodicidad se define como *scrub rate*, y se ha de ajustar de acuerdo a la razón de aparición de errores prevista [23]. En este caso, no se reprograma la FPGA total ni parcialmente, sino que se corrigen los bits afectados de forma aislada. La ventaja sobre la reconfiguración es que no se detiene la operatividad del sistema y se puede realizar de forma autónoma, sin depender de una copia externa. Sin embargo, el *scrubbing* no es capaz de corregir un número arbitrario de errores en posiciones arbitrarias, ya que tanto el número como la posición de los errores determina si el algoritmo de ECC es capaz de corregirlos [21]. Otra desventaja es que algunos bits de la memoria pueden estar fuera del alcance del *scrubber* y que al sufrir un *soft error* deriven en una interrupción funcional de evento único (SEFI —*Single-Event Functional Interrupt*—) [21] [24]. Un SEFI es una interrupción no destructiva producida por el impacto de un ion que provoca que un componente se reinicie, cuelgue o entre a un modo de operación diferente [25]. Las técnicas de reconfiguración que no paralizan el funcionamiento del sistema también pueden catalogarse como *scrubbers* [21].

2.2.2. Técnicas de protección de memorias basadas en códigos

Como se ha indicado, los códigos de detección y corrección de errores son una pieza esencial en el endurecimiento de las FPGA, pudiéndose destacar los códigos ECC y CRC.

Códigos de Corrección de Errores (ECC)

Los ECC nacieron en 1950 de mano del matemático Richard Hamming, quien propuso códigos de detección única de errores, de corrección única de errores, y de corrección única más detección doble de errores [26]. Versiones más modernas de los códigos de Hamming son capaces de corregir con una probabilidad del 100 % errores de hasta 5 bits [27]. Otros ECC ampliamente utilizados son los Bose–Chaudhuri–Hocquenghem (BCH), los cuales —a pesar de ocupar una mayor área [28]— permiten la corrección de múltiples bits. Operan sobre Campos de Galois [29].

Verificación de Redundancia Cíclica (CRC)

Los códigos de CRC, a pesar de que solo se utilizan para detectar errores, destacan por la simpleza de su implementación y por su capacidad de detección de errores en ráfaga —múltiples bits continuos modificados—. Al igual que los BCH, se definen sobre Campos de Galois y, en este caso, cualquier código generado por un polinomio generador de grado $n - k$ es capaz de detectar ráfagas de errores de longitud $n - k$, siendo k la longitud del mensaje binario original y $k - n$ el número de bits de comprobación que se añaden [30].

2.2.3. Redundancia Modular Triple (TMR)

La redundancia modular triple es una concreción de la redundancia N-modular en la que se replican tres módulos funcionalmente idénticos cuyas salidas son evaluadas por una lógica de votación mayoritaria. De este modo, si uno de los módulos falla, los otros dos determinan la salida correcta, enmascarando el fallo [31]. En la figura 2.2 se aplica TMR a la salida de 1 bit del módulo triplicado. Se decide cuál es la salida correcta mediante un votante, y a su vez cuál es la instancia errática del módulo mediante un comparador.

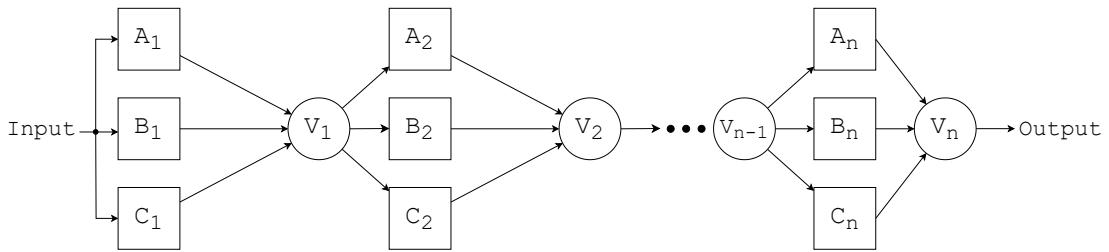


Fig. 2.1. Diagrama de un sistema TMR con n módulos triplicados. Los votantes se representan como V_n [31].

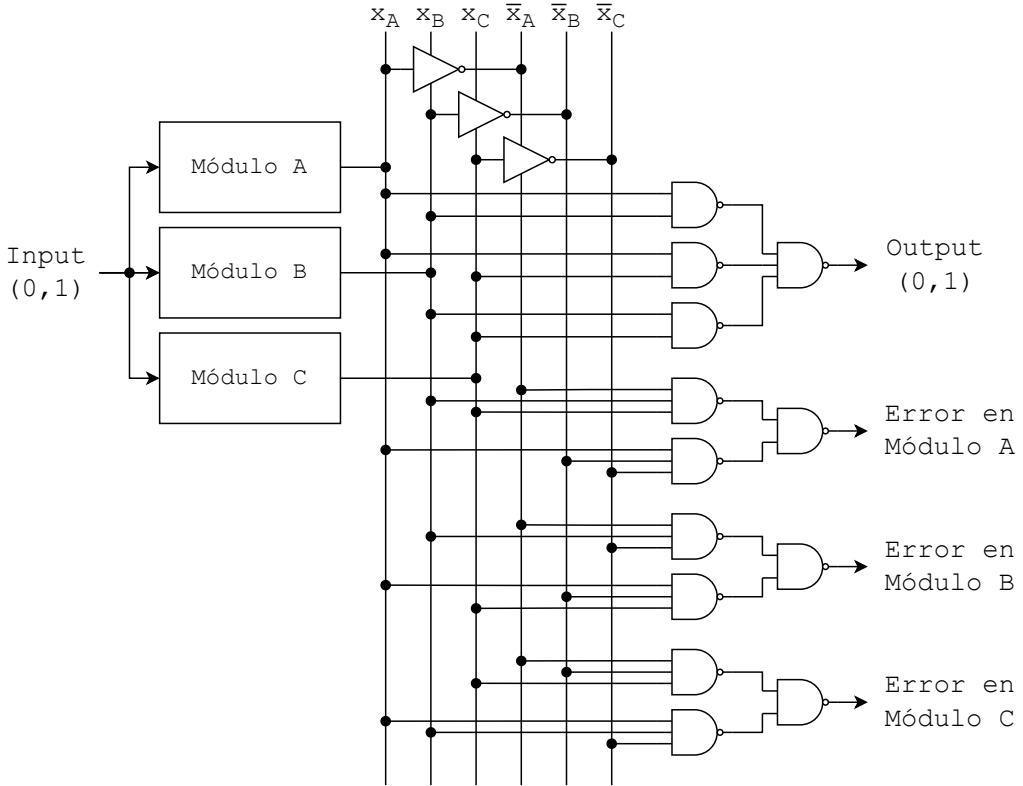


Fig. 2.2. Circuito que implementa TMR para una señal de 1 bit, incluyendo la detección de la instancia afectada [31]. Módulo A, Módulo B y Módulo C son instancias del mismo módulo.

La triple redundancia modular se puede aplicar a distintos niveles: desde una sección amplia del circuito que implemente una lógica compleja, hasta un simple biestable [32]. Se puede incluso aplicar TMR a nivel de microcódigo mediante la implementación de una misma microinstrucción con tres algoritmos diferentes. Esos algoritmos preferiblemente se han de ejecutar de forma paralela y han de utilizar circuitería independiente. Las microinstrucciones son las operaciones fundamentales de un procesador —como la suma, complemento o desplazamientos—, las cuales se implementan mediante microcódigo [31].

La fiabilidad de un sistema TMR (R_{TMR}) depende tanto de la fiabilidad del módulo triplicado (p_c) como de la fiabilidad del votante (p_v) [31].

$$R_{TMR} = p_v(3p_c^2 - 2p_c^3) \quad (2.1)$$

Donde:

$R_{TMR} \equiv$ Fiabilidad del sistema TMR

$p_v \equiv$ Fiabilidad del votante

$p_c \equiv$ Fiabilidad del módulo triplicado

Por lo tanto, si se quiere evitar la existencia de un único punto de fallo, se puede aplicar TMR al propio votante, triplicándolo. La fiabilidad del sistema sería entonces $R_{TMR} \times R_{voter}$ [31].

$$R_{TMR} \times R_{voter} = (3p_c^2 - 2p_c^3) \times (3p_v^2 - 2p_v^3) \quad (2.2)$$

Donde:

$R_{TMR} \equiv$ Fiabilidad del conjunto de módulos

$R_{voter} \equiv$ Fiabilidad del conjunto de votantes

$p_v \equiv$ Fiabilidad del votante triplicado

$p_c \equiv$ Fiabilidad del módulo triplicado

Todos los cálculos de fiabilidades se realizan considerando que el sistema falla cuando no existe un mínimo de dos elementos triplicados concordantes. Además, la fiabilidad se define como la probabilidad de que el sistema funcione de forma adecuada durante un determinado tiempo bajo las condiciones de entorno especificadas [33]. Atendiendo pues a las ecuaciones de fiabilidad 2.1 y 2.2, se puede comparar la fiabilidad del sistema con y sin redundancia del votante. En las figuras 2.3 y 2.4 se aprecia que para valores de fiabilidad del votante superiores a 0.5, la fiabilidad del sistema mejora tras incorporar la redundancia a nivel de votante. Sin embargo, para valores de fiabilidad del votante inferiores a 0.5, la fiabilidad del sistema disminuye, pues se estarían añadiendo componentes que perjudican la fiabilidad. De forma similar, un TMR tan solo mejora la fiabilidad si el módulo triplicado tiene una fiabilidad mayor a 0.5. En caso contrario, la perjudicaría.

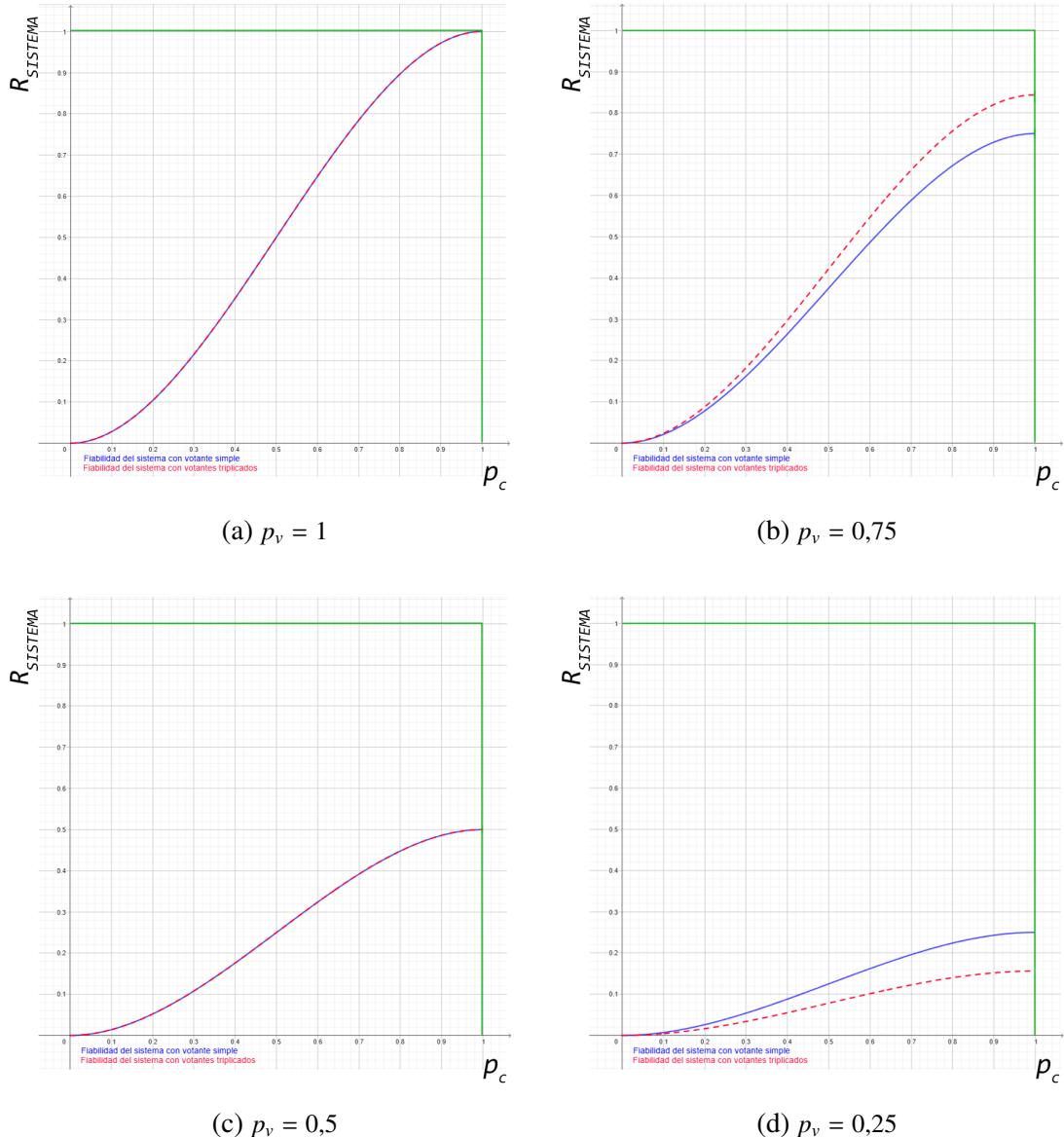


Fig. 2.3. Fiabilidades del sistema TMR con y sin redundancia del votante para cuatro valores distintos de fiabilidad del votante. En azul y línea continua: sistema con votante simple. En rojo y línea discontinua: sistema con votantes triplicados.

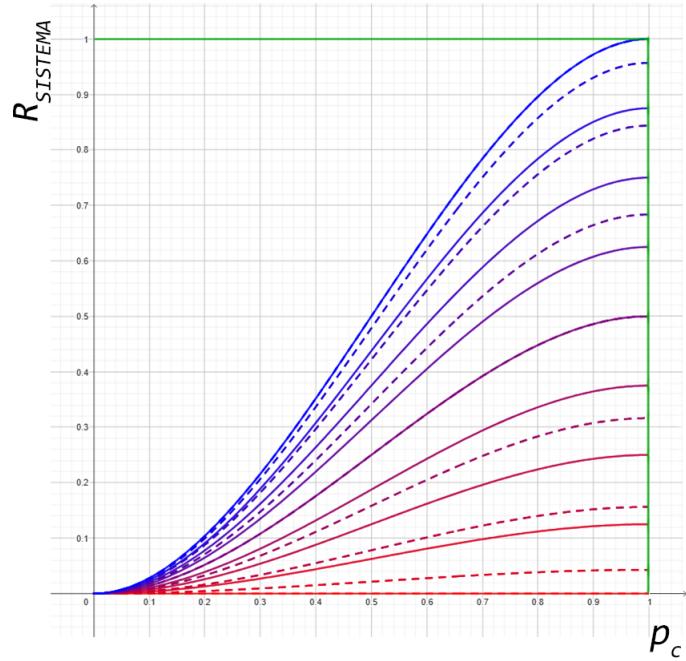


Fig. 2.4. Fiabilidades del sistema con y sin redundancia del votante para nueve valores distintos de fiabilidad del votante. En línea continua: sistema con votante simple. En línea discontinua: sistema con votante triplicado. Las fiabilidades para un mismo valor de p_v comparten color, desde azul cuando $p_v = 1$ a rojo cuando $p_v = 0$.

Analizando la figura 2.5, que muestra el diagrama de un sistema TMR con n módulos triplicados y redundancia de votantes, se observa una propiedad característica de la triplicación de los votantes: la propagación de errores a etapas más allá de la siguiente es prácticamente nula [31]. En el caso de un sistema con votantes simples, un error en un votante se propaga a todos los módulos subsecuentes, anulando cualquier beneficio de la triplicación de los módulos posteriores (ver figura 2.1). En cambio, si un votante triplicado falla, el error tan solo se propaga a uno de los módulos de la siguiente etapa. Los otros dos tendrán una entrada correcta y, por lo tanto, el error no se propaga a etapas posteriores.

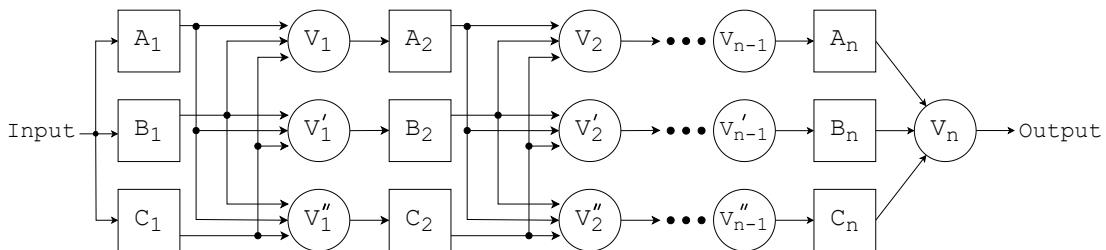


Fig. 2.5. Diagrama de un sistema TMR con n módulos y votantes triplicados [31].

Complementando la importancia de evitar y detener la propagación de errores en el sistema, se vuelve imperativo hablar de la necesidad de aplicar técnicas SEM en conjunción con la redundancia modular triple para evitar la acumulación de errores. Si bien el sistema TMR enmascara un error, a menos que este se corrija, el módulo afectado quedará inutilizado para siempre. El sistema por sí mismo no es capaz de corregir errores y los acumulará, quedando condenado a bloquearse tarde o temprano. Ahí entra la mitigación de los *soft errors*. En el caso de que se produzca un error en la memoria de configuración entre ciclos de *scrubbing*, el sistema TMR enmascarará el fallo, pero el módulo afectado quedará inoperativo y el sistema pasará de modo TMR a *lockstep* (ya no va a poder enmascarar otro error, simplemente detectarlo y bloquearse en tal caso) [46]. En el próximo ciclo de *scrubbing*, se detectará el error y se intentará corregir. Si es corregible, todos los módulos volverán a funcionar de forma correcta y se volverá a operar en modo TMR. Así, se evita que un error transitorio se acumule y perdure en el tiempo.

2.3. RISC-V

RISC-V (*Reduced Instruction Set Computer-V*) es un juego de instrucciones (ISA — *Instruction Set Architecture*) desarrollado inicialmente en la Universidad de California en Berkeley en el año 2010 [34]. Una ISA es una especificación que define aspectos relativos a la forma en la que un procesador analiza y ejecuta instrucciones [35]. Esto es, se plantea como una interfaz visible al *software*, evitando detallar aspectos relativos a la implementación *hardware*. Por lo tanto, detalla cuestiones como las instrucciones, tipos de datos, registros, modos de direccionamiento, interrupciones y arquitectura de memoria, entre otras [34].

Existen dos principales familias de ISA: CISC (*Complex Instruction Set Computer*) y RISC (*Reduced Instruction Set Computer*). Como su nombre indica, RISC-V pertenece a la familia RISC, siendo además el quinto gran diseño de ISA RISC de la UC Berkeley. Las instrucciones de una arquitectura RISC son reducidas en el sentido de que realizan acciones más atómicas: una operación con valores guardados en registros, leer de la memoria, escribir a la memoria... todas ellas se ejecutan con instrucciones diferenciadas. Sin embargo, en una arquitectura CISC las instrucciones son mucho más complejas, y realizan conjuntos de acciones que en RISC estarían divididas en múltiples instrucciones. Así, una instrucción CISC podría leer de la memoria, operar y volver a escribir a la memoria, todo ello con la misma instrucción. Esto acarrea varias desventajas. En primer lugar, las instrucciones, al realizar más acciones, tardan más ciclos de reloj en ejecutarse. Esto dificulta la segmentación de instrucciones (la cual permite implementar paralelismo a nivel de instrucción). Por otro lado, las instrucciones ocupan más espacio, lo que deriva en menos espacio disponible para registros, la memoria más rápida del procesador. Todo esto se soluciona en la arquitectura RISC, con instrucciones pequeñas que ocupan menos espacio y además permiten implementar segmentación de instrucciones, con la que aumenta las prestaciones del procesador [36]. Una desventaja de RISC sobre CISC es que

la programación en ensamblador es más compleja, ya que acciones que en CISC supondrían una instrucción, en RISC requieren varias. No obstante, esta desventaja no eclipsa las múltiples ventajas, por lo que la mayoría de arquitecturas modernas implementan una ISA RISC [37].

RISC-V nace con el objetivo principal de apoyar la investigación y enseñanza en el campo de la arquitectura de computadores. Sin embargo, evolucionó hacia una aspiración incluso más ambiciosa: convertirse en una arquitectura estándar abierta y libre de regalías [34]. Estas características —que permiten recortar gastos de desarrollo, producción y puesta en marcha— sumadas al rendimiento de las implementaciones modernas de RISC-V, la convierten en una excelente candidata para la industria aeroespacial. Por estos motivos, se están abandonando procesadores resistentes a radiación con un largo recorrido en la industria como el RAD750 o el GR712RC en favor de procesadores RISC-V tanto *hardcore* como *softcore* en FPGAs o ASICs [16]. Un ASIC (*Application Specific Integrated Circuit*) es un circuito integrado diseñado para una aplicación específica, como el procesamiento de señales. Al igual que las FPGAs, se suelen programar en lenguajes HDL, con la diferencia de que los ASICs no se pueden reconfigurar. En la figura 2.6 se ve la proporción de implantación de diferentes ISA en misiones espaciales a lo largo de los años y se aprecia que ya en 2017 se preveía la entrada de procesadores RISC-V a la industria espacial.

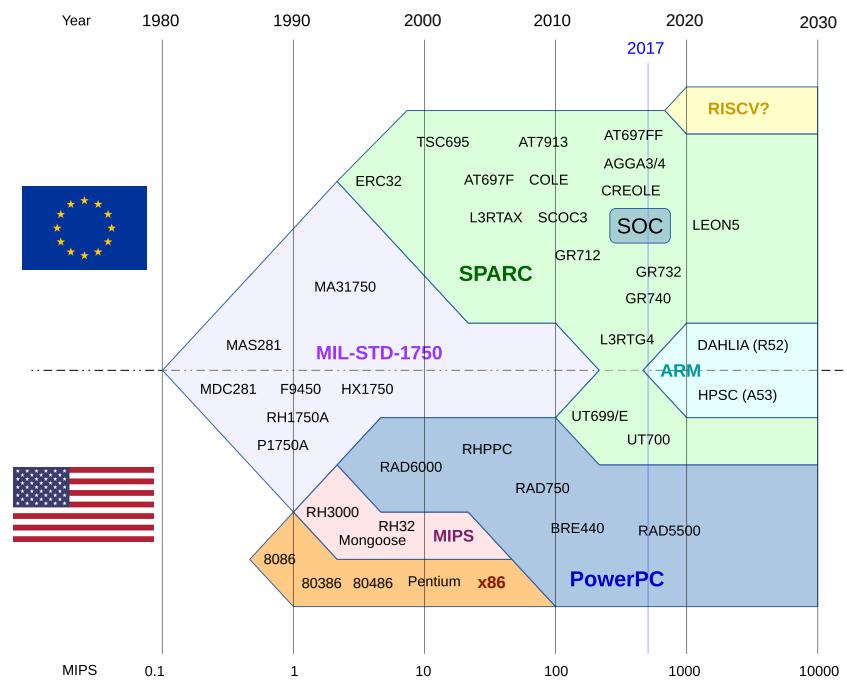


Fig. 2.6. Principales ISA y procesadores utilizados en misiones espaciales europeas y estadounidenses [38].

2.4. Procesadores softcore

Los procesadores *softcore* son aquellos que se pueden implementar mediante síntesis lógica en dispositivos de lógica programable —como FPGAs—, e incluso en ASICs. El uso de procesadores *softcore* en FPGAs le permite al diseñador añadir una unidad de procesamiento a demanda para cubrir tareas como la monitorización y la coordinación de comunicaciones entre componentes o el alivio de carga de un procesador principal *hardcore* —añadiendo un núcleo más de computación paralela—. Otorgan pues una gran versatilidad tanto en funciones y casos de uso como en características. Existen procesadores *softcore* basados en diferentes ISA, con distintos niveles de protección ante radiación, de código abierto o propietario, con licencias de pago o libres de regalías, y demás características. La amplia oferta permite escoger el *softcore* más adecuado para cada caso de uso [39].

Para aplicaciones espaciales se pueden utilizar procesadores *softcore* específicamente diseñados para resistir radiación, o se puede tomar un procesador no resistente y añadirle tolerancia a fallos. Esta segunda opción, dependiendo de si el procesador es o no de código abierto, se puede realizar a diferentes niveles. Si el código de descripción de hardware del *softcore* es público, se puede añadir redundancia de forma interna, modificando la propia arquitectura del procesador. En cambio, si no se tiene acceso a su código, el dominio de la redundancia habrá de aumentar, aplicándose sobre el procesador en su conjunto añadiendo más copias del mismo [40].

2.4.1. LEON

Los procesadores *softcore* de la familia LEON nacieron en 1997 como un proyecto de la Agencia Espacial Europea (ESA —*European Space Agency*—) para desarrollar procesadores de alto rendimiento de uso espacial. Se eligió la ISA SPARC (*Scalable Processor ARChitecture* —un juego de instrucciones RISC—) por ser abierta y libre de regalías, entre otros motivos. Posteriormente, los procesadores *softcore* LEON han sido desarrollados por Cobham Gaisler [38]. El último modelo de la familia LEON, el LEON5, mejora en un 85 % la velocidad de ejecución monohilo con respecto al LEON4 y ofrece tolerancia a fallos en la versión del producto adscrita a licencia comercial. Su versión abierta no incluye tolerancia a fallos [41]. Los procesadores LEON se han utilizado en múltiples misiones de la ESA. El cohete Ariane 6 utilizó un LEON2 en tecnología de 65 nm para controlar la red de Ethernet tolerante a fallos (TTEthernet —*Time-Triggered Ethernet*—) [42]. De forma similar, todas las grandes compañías europeas integradoras de sistemas espaciales han desarrollado y utilizado en misiones SoCs basados en LEON [43].

2.4.2. NOEL-V

NOEL-V es la apuesta de Cobham Gaisler por la ISA RISC-V. Se trata de un procesador *softcore* que complementa la línea LEON (nótese que NOEL es un anagrama de LEON), implementando una ISA más moderna. Al igual que el LEON5, tiene una arquitectura superescalar (ver figura 2.7) que le permite despachar dos instrucciones en cada ciclo de reloj, así como redundancia a fallos en su versión de licencia comercial. Sus capacidades de resistencia a la radiación han sido probadas en la órbita terrestre [44] y las evaluaciones muestran un rendimiento superior a otros procesadores RISC-V [16].

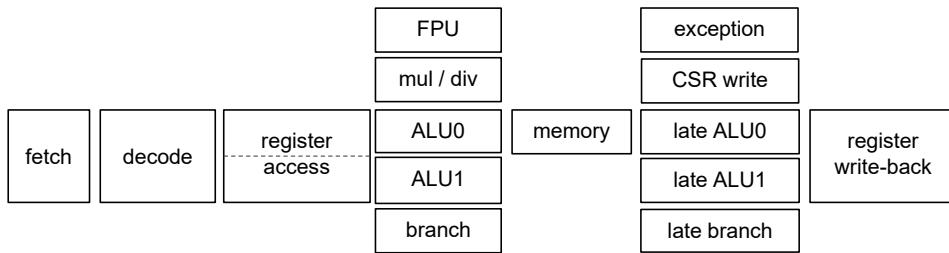


Fig. 2.7. Segmentación en 7 fases del *softcore* NOEL-V [45].

2.4.3. MicroBlaze

La familia MicroBlaze es la propuesta de AMD por ofrecer procesadores *softcore* optimizados para su línea de FPGAs y SoCs adaptables completamente integrados en el flujo de diseño de sus herramientas. Se caracterizan por su alta flexibilidad, pudiendo elegir entre configuraciones predeterminadas (microcontrolador, procesador en tiempo real y procesador de aplicaciones) e incluso modificar aspectos concretos de la arquitectura. Sin embargo, estas modificaciones se han de realizar desde el menú de configuración del IP (*Intellectual Property*) en la *Suite* de Diseño Vivado, pues MicroBlaze no es de código abierto. En lo relativo a usos espaciales, esta familia de *softcores* ofrece configuraciones en *dual core lockstep* y TMR mediante el *Triple Modular Redundancy* (TMR) LogiCORE IP [46], así como protección de la memoria de configuración mediante el *Soft Error Mitigation Controller* LogiCORE IP [24] y detección y corrección de errores en la RAM de bloque [47].

A finales de abril de 2024, AMD lanzó MicroBlaze-V, la segunda adición a la familia MicroBlaze, esta vez abandonando su ISA RISC propietaria en favor de RISC-V. Es completamente compatible a nivel de hardware con su predecesor, por lo que la migración desde MicroBlaze a MicroBlaze-V es sencilla. Como consecuencia de esa compatibilidad, el subsistema TMR para MicroBlaze es compatible con MicroBlaze-V [48], a pesar de que no se haya actualizado desde abril de 2022 [46].

2.5. PYNQ-Z1 y Zynq-7000

La placa PYNQ-Z1 es una plataforma *hardware* para el *framework* de código abierto PYNQ (*PYthon productivity for zyNQ*) desarrollada por Digilent. Cuenta con un APSoc (*All Programmable System on a Chip*) ZYNQ XC7Z020-1CLG400C (xc7z020clg400-1), de la familia Zynq-7000. Como se ve en la figura 2.8, también dispone de múltiples interfaces y módulos de I/O como Pmod, interruptores, botones o LEDs [49].

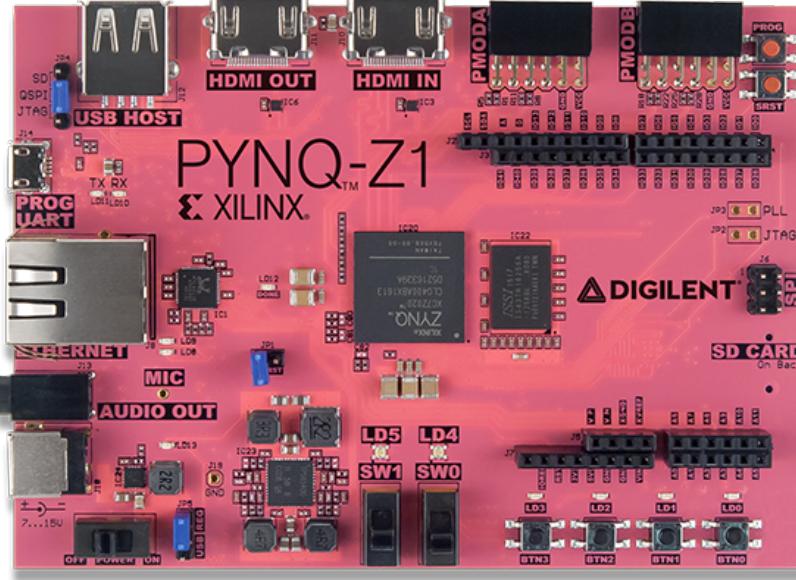


Fig. 2.8. Fotografía de la placa PYNQ-Z1 [49].

El sistema en chip Zynq-7000, como se ve en la figura 2.9, está compuesto por dos grandes bloques funcionales: el sistema de procesamiento (PS —*Processing System*)— y la lógica programable (PL —*Programmable Logic*). El PS, a su vez, se compone de la unidad de procesamiento de aplicaciones (APU —*Application Processing Unit*—), las interfaces de memoria, los periféricos de entrada y salida (IOP —*Input/Output Peripherals*—) y las interconexiones [50]. La APU del ZYNQ XC7Z020-1CLG400C es un procesador Cortex-A9 *dual-core* de 650MHz. Por otro lado, el PL es equivalente a una FPGA Artix-7, con 13300 *logic slices* —cada una con cuatro LUTs (*Look-Up Tables*) de 6 entradas y 8 biestables—, 630 KB de BRAM (*Block Random Access Memory*) rápida, 4 *tiles* de gestión de reloj, 220 *slices* DSP (*Digital Signal Processing*) y un conversor de señal analógica a digital (XADC —*Xilinx Analog to Digital Converter*) [49][50].

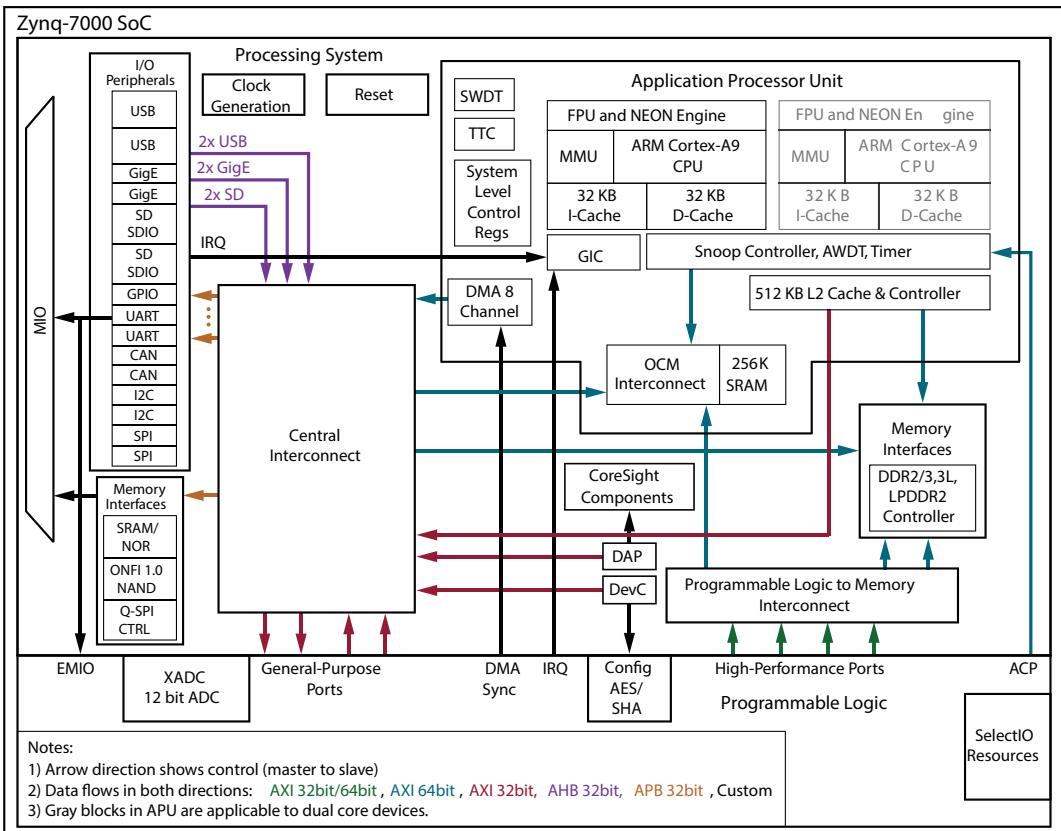


Fig. 2.9. Diagrama del Zynq-7000 SoC [50].

A diferencia de una FPGA Artix de la serie 7 —que solo cuenta con PL—, el Zynq-7000 integra un PS y un PL intercomunicados, lo que permite ejecutar código en el PS e implementar un diseño lógico en el PL que procese datos de forma paralela. La placa PYNQ-Z1, utilizando el *framework* PYNQ, se vale de esta característica para ofrecer al usuario una forma sencilla y transparente de correr programas acelerados por hardware en un lenguaje de alto nivel y productividad como Python [51]. Eligiendo el modo de arranque MicroSD [49], la placa carga desde la tarjeta SD una imagen de PYNQ, basada en Ubuntu, que contiene infraestructura de código abierto de cuadernos de Jupyter para ejecutar un *kernel* IPython (*Interactive Python*) y un servidor web en el procesador ARM del PS. Desde los cuadernos Jupyter, el usuario de la placa puede utilizar los *overlays* de PYNQ para programar el PL y utilizarlo mediante simples llamadas a funciones, todo ello sin salir del entorno de Jupyter [51].

Los *overlays* son bibliotecas de *hardware* que se cargan de forma dinámica y a demanda en el PL. PYNQ-Z1 ofrece un *overlay* base (ver figura 2.10) que permite controlar todos los periféricos desde Python. Esta forma transparente de comunicarse con el PL permite que ingenieros de *hardware* diseñen circuitos optimizados para una cierta tarea, y que los ingenieros de *software* los utilicen para procesar datos en su código con la misma facilidad que si estuvieran llamando a una función en el mismo PS [51].

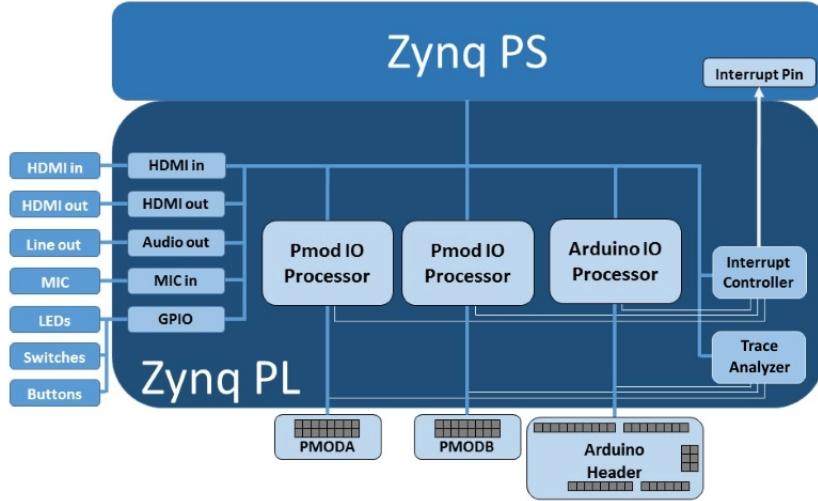


Fig. 2.10. Diagrama de bloques del *overlay* base de la placa PYNQ-Z1 [51].

A pesar de las ventajas de PYNQ para algunos casos de uso, no es idóneo para trabajar a un nivel más bajo. Afortunadamente, PYNQ-Z1 dispone de tres modos de arranque: MicroSD, Quad SPI y JTAG. MicroSD permite arrancar el SoC desde una tarjeta microSD insertada en el conector J9 con una *Zynq Boot Image* estándar creada con las herramientas de Xilinx. Quad SPI realiza el arranque desde la memoria *flash* Quad-SPI de 16 MB de la placa cargada con una *Zynq Boot Image* estándar. Finalmente, JTAG permite programar el SoC desde un ordenador externo a través del puerto JTAG. Hasta que se carga el *software*, el procesador permanece a la espera. Tras ello, se puede ejecutar directamente el programa, o realizar *debugging* línea por línea. En este modo, se puede configurar el PL directamente, independientemente del PS [49]. En el resto de modos, el PL se configura desde el PS, a través del puerto PCAP (*Processor Configuration Access Port*), lo cual es común en la fase de despliegue. Sin embargo, la configuración por el puerto JTAG es más adecuada para la fase de desarrollo, pues permite hacer *debugging*. En la figura 2.12 se muestran las diferentes rutas de configuración del PL. Además de las descritas anteriormente, existe la ruta ICAP (*Internal Configuration Access Port*) para reconfigurar el PL desde lógica ya instanciada en el propio PL [50]. Para escoger el modo de arranque de PYNQ-Z1, se ha de colocar el *jumper* de modo JP4 en la posición deseada, como se muestra en la figura 2.11 [49].

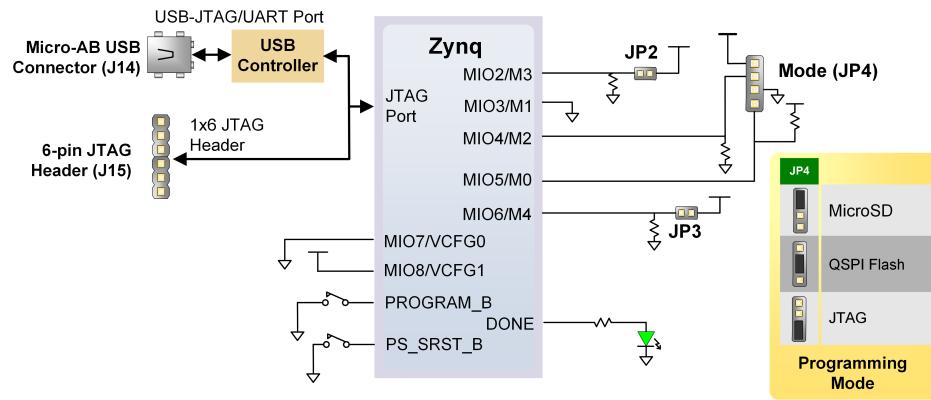


Fig. 2.11. Pines de configuración de la placa PYNQ-Z1 [49].

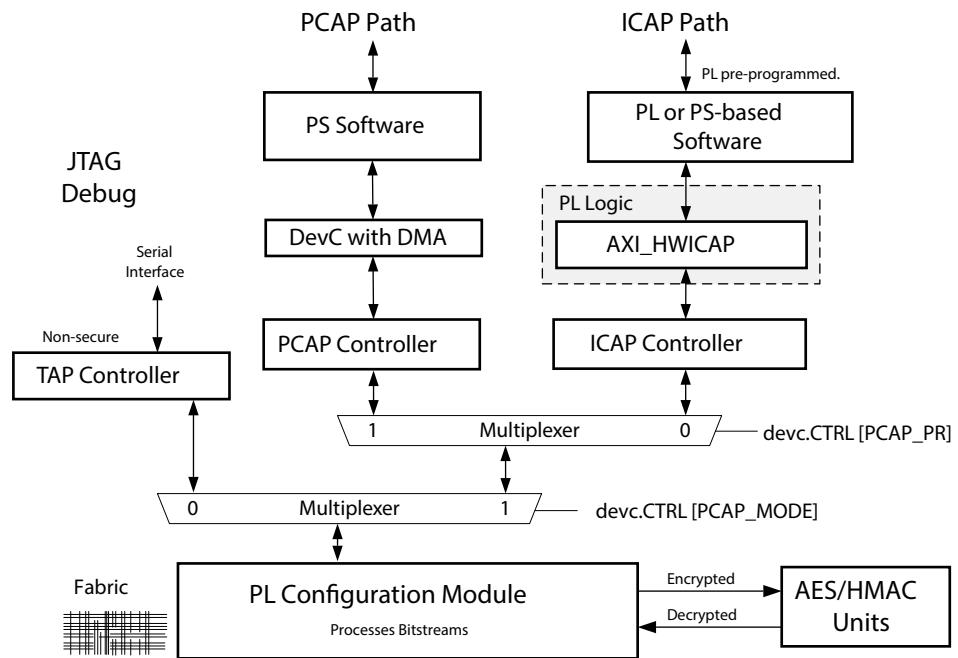


Fig. 2.12. Rutas de configuración de la lógica programable del SoC Zynq-7000 [50].

3. METODOLOGÍA Y DISEÑO

En este capítulo se detallará la metodología seguida en el desarrollo del proyecto, fundamentada principalmente en el flujo de diseño a nivel de sistema de la metodología de diseño AMD UltraFast (ver figura 3.1) [52]. A su vez, se plasmará el proceso del desarrollo hasta la concepción final del diseño.

3.1. Requisitos del sistema para aplicaciones espaciales

Los requisitos de un sistema apto para aplicaciones espaciales son de vital importancia, pues una buena planificación y estructuración del desarrollo es vital para el éxito de misiones en las que se invierten grandes sumas de dinero y en las que incluso pueden estar en riesgo vidas humanas. Por ello, y como ya se ha explicado, existen estándares como el *ECSS-E-ST-20-40C – ASIC, FPGA and IP Core engineering* [10], utilizado por la ESA, que buscan garantizar las buenas prácticas en el proceso de desarrollo. El alcance y la escala de este trabajo impiden realizar una especificación de requisitos tan extensa y detallada como la descrita en el estándar mencionado, por lo que se plasmaron aquellos más cercanos al tipo de trabajo desarrollado y al conocimiento técnico del autor. Un proyecto espacial se nutre de múltiples disciplinas que requieren de la intervención de diferentes personas. A pesar de no poder seguir el estándar, se tomaron ciertas características de sus requisitos: la criticidad y el tipo de dispositivo [10].

La criticidad indica la potencial gravedad de las consecuencias derivadas del fallo o incumplimiento del requisito. Se divide en las siguientes categorías:

1. **A** Deriva en la propagación de fallos. En cuanto a las consecuencias en la seguridad, acarrea la pérdida de la vida, heridas permanentemente discapacitantes, enfermedades profesionales permanentes, pérdida del sistema, pérdida de un sistema de vuelo tripulado, pérdida de instalaciones de lanzamiento, o daños ambientales.
2. **B** Deriva en la pérdida de la misión. En cuanto a las consecuencias en la seguridad, acarrea heridas temporalmente discapacitantes, enfermedades profesionales temporales, daños importantes a los sistemas de vuelo, daños importantes a las instalaciones terrestres, daños importantes a propiedad pública o privada, o daños ambientales importantes.
3. **C** Deriva en una importante degradación de la misión. No tiene consecuencias en la seguridad.
4. **D** Derivan en una pequeña degradación de la misión. No tiene consecuencias en la seguridad.

El tipo de dispositivo indica cuáles están afectados por el requisito. Un requisito puede afectar a uno, varios, o todos los tipos de dispositivos. Se detallan cuatro tipos principales de dispositivos:

1. **D-ASIC** Aplicable a ASICs digitales o a la parte digital de ASICs de señal mixta.
2. **A-ASIC** Aplicable a ASICs análogos o a la parte análoga de ASICs de señal mixta.
3. **FPGA** Aplicable a FPGAs.
4. **IP** Aplicable a núcleos IP digitales y análogos.

Además de las entradas mencionadas, los requisitos que se plantearon cuentan con apartados de prioridad, claridad (informa sobre la ambigüedad del requisito), verificabilidad y estabilidad (cómo de probable es que no se modifique el requisito en el futuro). Además, tienen una entrada con el texto principal, otra con notas aclaratorias, y una última con las pruebas que validan su cumplimiento. El identificador del requisito se basa en su tipo: funcional (RF) o no funcional (RN). El formato expuesto se muestra en la tabla 3.1.

Identificador: RX-##				
Dispositivo	<input type="checkbox"/> D-ASIC	<input type="checkbox"/> A-ASIC	<input type="checkbox"/> FPGA	<input type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A	<input type="checkbox"/> B	<input type="checkbox"/> C	<input type="checkbox"/> D
Prioridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Claridad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Verificabilidad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Estabilidad	<input type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Texto principal	<Texto principal del requisito>			
Notas	<Notas aclaratorias>			
Tests	<Pruebas de verificación>			

TABLA 3.1. FORMATO DE LOS REQUISITOS

Todos los requisitos establecidos se han recogido en el Anexo A.

3.2. *Hardware utilizado*

Uno de los objetivos del trabajo es determinar si un chip de lógica programable de tamaño moderado es capaz de albergar un microcontrolador endurecido. Por ello, se utilizará la placa PYNQ-Z1, con el SoC ZYNQ XC7Z020-1CLG400C de 85120 celdas lógicas [50].

La comunicación entre la placa PYNQ-Z1 y el ordenador con el que se monitorearán las pruebas se realizará a través del protocolo serial UART (*Universal Asynchronous Receiver-Transmitter*). PYNQ-Z1 dispone de un puerto USB UART (conector J14), asignado a los pines MIO (*Multiplexed I/O*) 14 y 15 —Rx y Tx, respectivamente— del PS, bajo el controlador UART0 [49]. Debido a esto, el puerto USB de la placa no se puede utilizar para comunicarse mediante UART con la lógica programable, al menos de forma directa. El controlador UART del PS permite mantener dos comunicaciones independientes al mismo tiempo utilizando UART0 y UART1. Si se le asignan pines EMIO (*Extended Multiplexed I/O*) —los cuales son accesibles desde el PL— al UART1, se logra comunicación UART entre el PS y el PL [50]. Así, se puede ejecutar un programa en el PS que cree un puente entre el UART0 (conecta el ordenador externo y el PS) y el UART1 (conecta el PS y el PL), conectando de esta forma el ordenador externo y el circuito en la lógica programable mediante UART.

Se van a utilizar tres canales de comunicación UART durante las pruebas: uno para inyectar errores y observar el estado de módulo SEM, otro para gestionar las entradas y salidas del programa en ejecución en el microcontrolador endurecido, y otro para observar el estado del subsistema TMR. Con el puente UART0-UART1 se cubre tan solo uno de esos canales, por lo que se utilizarán dos módulos PmodUSBUART de Digilent. Estos contienen una interfaz USB-UART [53] y permiten conectar las señales UART necesarias a los puertos Pmod de la placa, accesibles desde el PL.

Así, el *hardware* utilizado será el siguiente: 1 placa PYNQ-Z1, 1 ordenador externo, 3 cables USB de tipo A a micro-B y 2 módulos PmodUSBUART.

3.3. *Suite de diseño Vivado: herramientas, IPs y flujo de trabajo*

Tal y como se ha relatado anteriormente, se ha seguido la metodología AMD Ultra-Fast. En la figura 3.1 se observan tres etapas bien diferenciadas: la entrada de diseño del sistema, la implementación, y la puesta en marcha y validación del *hardware*. Las dos primeras etapas se han llevado a cabo en la *Suite de diseño Vivado*, mientras que la última se ha realizado en el entorno de desarrollo de *software* Vitis. Las herramientas expuestas forman parte de la metodología, por lo que su integración en la misma es completa. Esta sinergia entre metodología y herramientas queda reflejada de forma paradigmática en el navegador de flujo de Vivado (ver figura 3.2), un menú vertical que guía al usuario por diferentes etapas de la metodología de manera intuitiva.

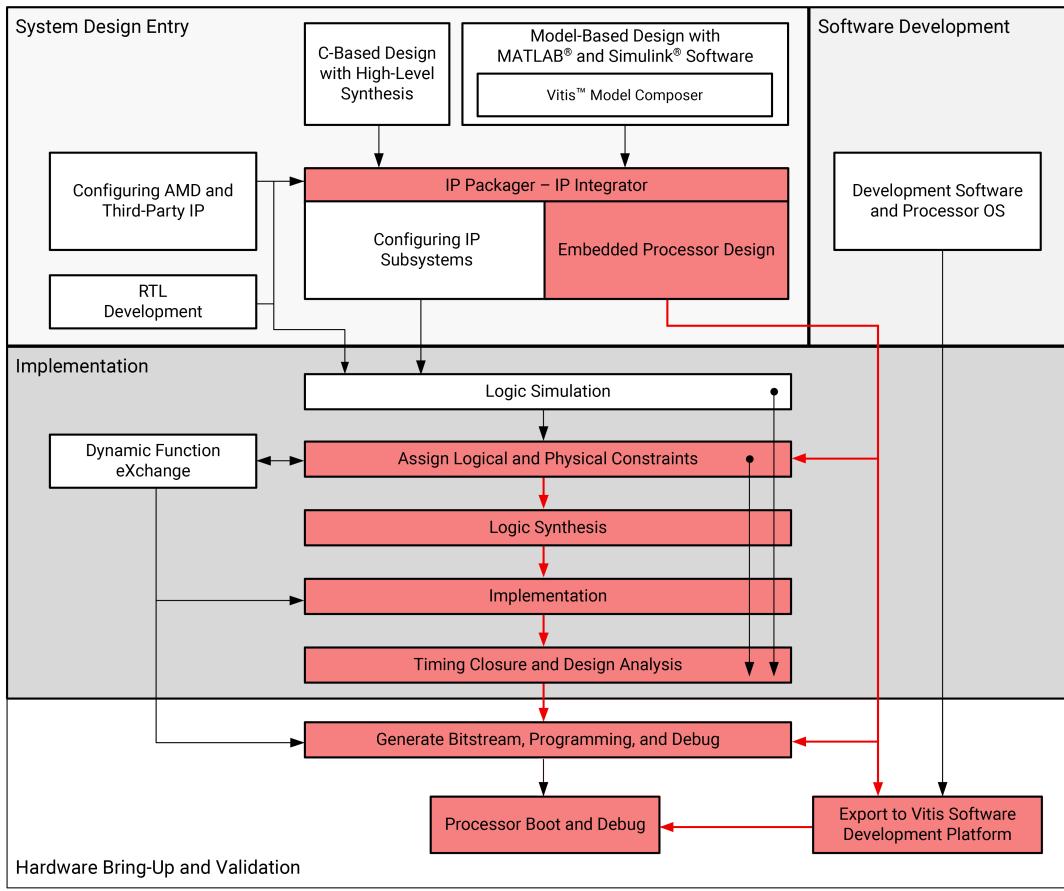


Fig. 3.1. Flujo de diseño a nivel de sistema para FPGAs y SoCs de acuerdo a la metodología AMD UltraFast [52]. Se muestran en rojo las etapas aplicables.

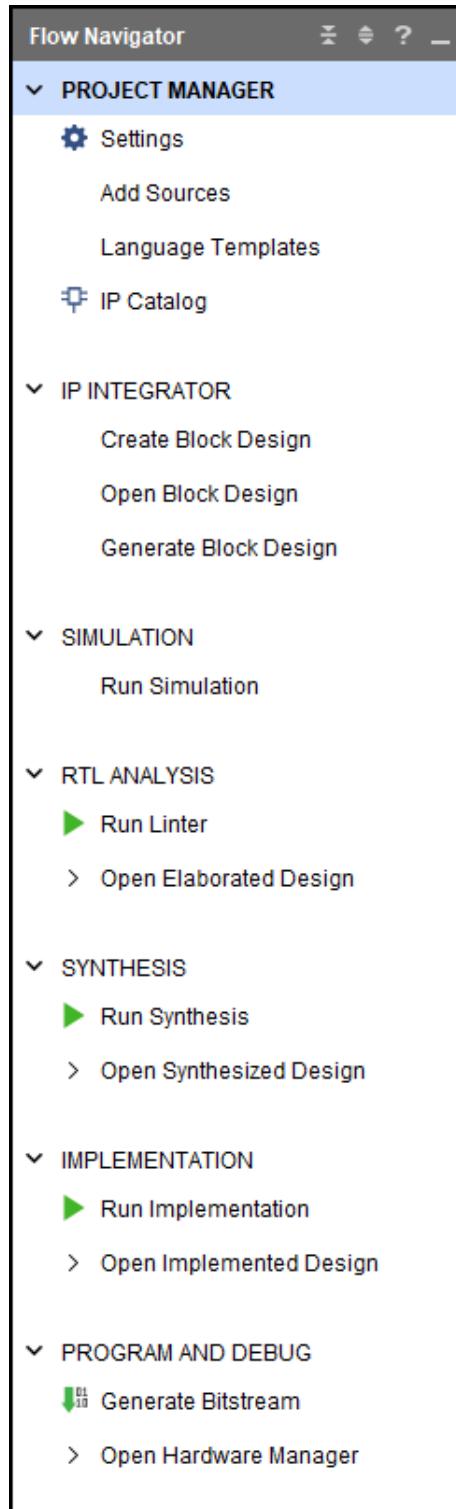


Fig. 3.2. Navegador de flujo de la *Suite* de diseño Vivado.

El primer paso antes de crear el proyecto es, sin embargo, descargar los archivos de la placa PYNQ-Z1 [54] y colocarlos en:
<Xilinx installation directory>\Vivado\<version>\data\boards\board_files, ya que estos no se encuentran en la instalación de Vivado.

3.3.1. IP Integrator

El integrador de IPs permite trabajar con un diseño de bloques en una interfaz visual e intuitiva donde la lógica e implementación interna de cada bloque se abstraen, mostrando tan solo los bloques, sus interfaces y las diferentes conexiones, puertos y otras interfaces del diseño. Estos bloques pueden ser tanto núcleos IP como módulos en HDL escritos por el usuario. Así, la herramienta facilita la creación de diseños complejos que utilicen diferentes módulos y núcleos IP. En esta fase del ciclo de diseño se define la lógica a nivel de trasferencia de registro (RTL —*Register Transfer Level*—).

Antes de crear el sistema tolerante a fallos, se ha de diseñar uno que no lo sea, pero que contenga todos los elementos básicos que más adelante se incluirán en el diseño final. Este diseño también habrá de pasar por todas las fases de la figura 3.1. Una vez se valide, se podrá continuar con su transformación a un sistema TMR. El diseño no redundante, que se encuentra en el Anexo B.1, utiliza el *softcore* MicroBlaze-V y lo conecta mediante protocolo AXI (*Advanced eXtensible Interface*) a módulos I/O mapeados en memoria: AXI GPIO (*General Purpose Input/Output*), AXI UART16550 y AXI TIMER. De esta forma, se puede interactuar con dichos módulos desde *software* que se ejecuta en el MicroBlaze-V. Los AXI GPIO se utilizan para interactuar con los botones, interruptores y LEDs de la placa; el AXI UART16550, para mantener un canal de comunicación por puerto serial con el ordenador externo; y el AXI TIMER, para implementar un reloj en *hardware* que permita realizar esperas no bloqueantes en el código sin necesidad de utilizar diferentes hilos o procesos, para lo cual se requeriría un sistema operativo. Cabe resaltar que uno de los botones de la placa se utiliza como señal de *reset*. Ese botón no está mapeado en memoria, por lo que es inaccesible desde el código, y tan solo se utiliza para reiniciar el MicroBlaze-V. Como la señal de reinicio del módulo *Processor System Reset* es *active-low*, se ha implementado el módulo NOT_GATE para invertir la señal del botón. Este módulo, de implementación trivial, es el único escrito con HDL (Verilog en este caso):

```
1 module NOT_GATE(
2     input a,
3     output y
4 );
5
6     assign y = ~a;
7
8 endmodule
```

El resto de módulos del diseño son IPs proporcionadas por Vivado.

Configuración	Recursos del Dispositivo					
	LUTs	FFs	BRAMs (36K)	DSP48	F_{\max}	MHz
Microcontroller Preset	2221	1145	0	4	164	
Real-time Preset	4101	2670	6	4	117	
Application Preset	8577	5342	16	6	115	
Maximum Frequency	1234	826	0	0	224	

TABLA 3.2. UTILIZACIÓN DE RECURSOS EN SOCS ZYNQ-7000
PARA CADA CONFIGURACIÓN DEL MICROBLAZE-V

En cuanto a la configuración del MicroBlaze-V, se ha seleccionado la indicada en la primera fila de la tabla 3.2 [48]: la configuración de microcontrolador. Como se observa en dicha tabla, ofrece un buen equilibrio entre frecuencia máxima y utilización de recursos. Escoger una configuración con un uso bajo de los recursos de la FPGA es esencial para disminuir el área utilizada y reducir por tanto la posibilidad de que se produzca un SEU en caso de impacto de una partícula cargada.

Triple Modular Redundancy (TMR) v1.0 LogiCORE IP

Para convertir el sistema no tolerante a fallos en uno que sí lo sea, se han utilizado los núcleos IP MicroBlaze TMR. Este conjunto de IPs está desarrollado específicamente para convertir diseños que utilicen un procesador *softcore* de la familia MicroBlaze en sistemas TMR. Dichos sistemas resultantes son *Fault Tolerant - Fail Safe* (FT-FS), esto es, son capaces de enmascarar por completo el primer error no corregido (FT), y de detectar el segundo (FS) [46].

Los sistemas TMR de MicroBlaze se implementan mediante cinco núcleos IP [46]:

1. **TMR Voter** Compara las salidas de un elemento triplicado y proporciona la mayoritaria, enmascarando la incorrecta en caso de que no todas coincidan.
2. **TMR Comparator** Compara las salidas de un elemento triplicado y proporciona entre qué sub-bloques hay una disparidad en caso de que exista.
3. **TMR Manager** Gestiona diversas funcionalidades del sistema TMR. Una de sus funciones principales es la gestión del estado del sistema: *Voting* (modo FT/TMR), *Lockstep* (modo FS) o *Fatal* (se detiene).
4. **TMR Inject** Implementa la funcionalidad de inyección de instrucciones. Permite cargar una instrucción arbitraria a una dirección de memoria arbitraria en el procesador triplicado deseado.
5. **TMR SEM** Encapsula el *Soft Error Mitigation Controller* LogiCORE IP, utilizado para la corrección y detección de errores de la memoria de configuración, así como para la inyección de errores en la misma.

El sistema TMR triplica el procesador *softcore*, las memorias BRAM y los periféricos I/O. De esta forma, se implementan votaciones tanto de las interfaces de los controladores de *Instruction* y *Data LMB* (*Local Memory Bus*) de la BRAM (los cuales controlan la memoria de bloque local), como de los módulos de entrada y salida. Los votantes garantizan la propiedad de *Fault Tolerant*, enmascarando el error del sub-bloque afectado. Además de estos elementos triplicados, se puede triplicar la propia BRAM, utilizando a su vez ECC de forma independiente en cada una de ellas. Si se requiere una configuración que demande menos recursos, se puede implementar una única BRAM fuera del dominio TMR, compartida por los tres sub-bloques, y protegida mediante ECC. Sin embargo, la BRAM triplicada ofrece la mayor protección [46], por lo que es la configuración elegida en este caso. En la figura 3.3 se exponen ambas configuraciones de memoria. Independientemente de la configuración, los controladores de memoria de bloque tienen votantes triplicados. En cambio, las interfaces externas tienen un solo votante para cada una de ellas, y comparadores triplicados. Si se quiere implementar la triplicación de votantes en este último caso, se debe activar la opción de *Self-checking voter* de los comparadores. Así, se añade redundancia al votante realizando el comparador la votación de forma paralela y comprobando que la salida del votante coincide con aquella de la votación efectuada por el comparador (ver figura 3.4).

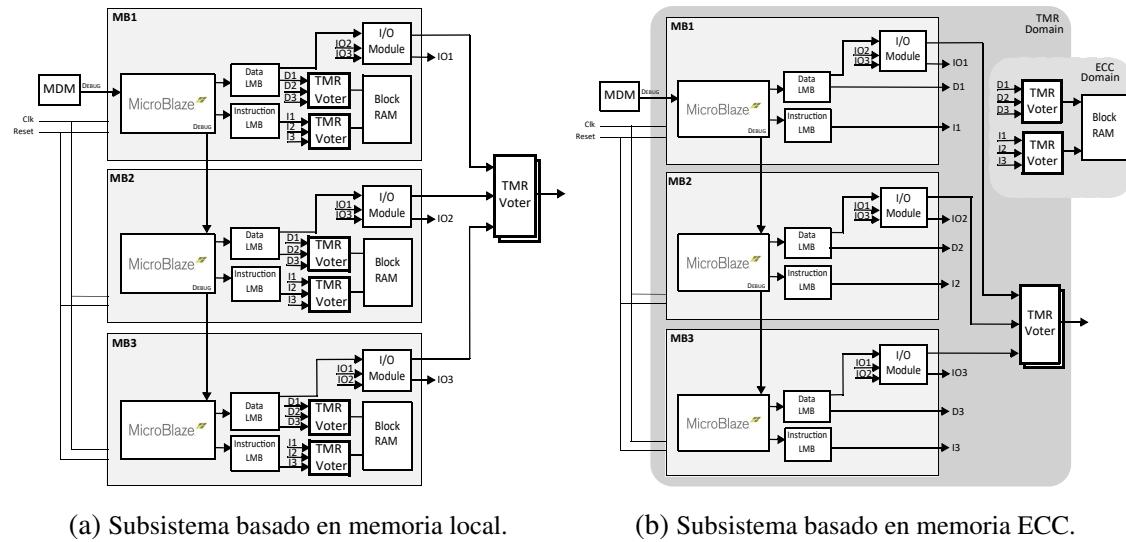


Fig. 3.3. Subsistemas TMR MicroBlaze *Fault Tolerant* basados en memoria local y memoria ECC compartida [46].

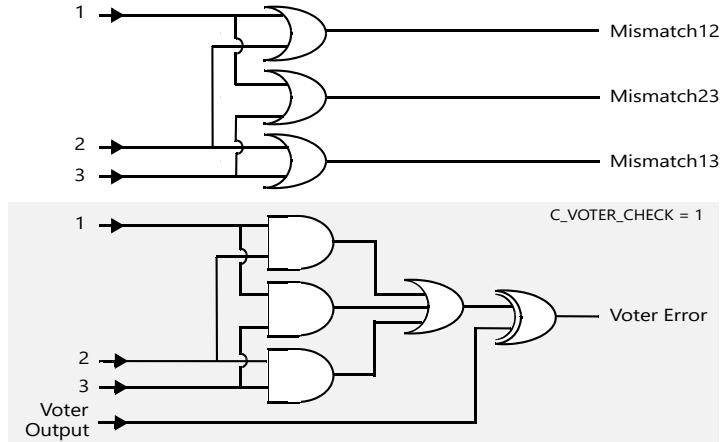


Fig. 3.4. Lógica de un Comparador TMR funcionando en modo TMR. La lógica inferior solo se añade al activar la opción de *Self-checking voter* ($C_VOTER_CHECK = 1$) [46].

En cuanto a la manera de convertir el diseño original en uno TMR, existen dos opciones: editar el diseño a mano añadiendo todos los núcleos IP necesarios y estableciendo las conexiones adecuadas, y utilizar el asistente de diseño de la automatización de bloques del integrador de IPs de Vivado para automatizar el proceso. La primera de las opciones es compleja y propensa a errores: además de tener que conocer las especificaciones de los IPs en absoluto detalle, se ha de gestionar la creación y edición de potencialmente cientos de conexiones entre bloques. Por ello, se recomienda la conversión automática a TMR. Para ello, el primer paso es asegurarse de que el diseño original pasa la validación del integrador de IPs, y crear una jerarquía que englobe la sección del diseño que se quiere triplicar. Los bloques que se añaden a la jerarquía han de compartir reloj y tener un mismo origen de las señales *reset*. Tras la creación de la jerarquía que encapsulará el nuevo subsistema TMR, se ha de añadir a la misma el IP *TMR Manager*. Al hacerlo, Vivado sugerirá utilizar el asistente de diseño para automatizar el proceso. Utilizándolo, tan solo se han de seleccionar las opciones deseadas, en este caso el modo TMR, la configuración de memoria LMB local, la interfaz SEM incluida y las opciones de inyección de errores activadas. La figura 3.5 muestra los bloques que se han seleccionado para ser triplicados. Se ha de destacar la exclusión del IP AXI UART16550. Este bloque no se ha triplicado ya que la naturaleza asíncrona del protocolo UART es incompatible con la necesidad de extrema sincronización del sistema TMR, que requiere de datos síncronos que garanticen que se procesan exactamente los mismos datos en los tres procesadores en cada ciclo de reloj. El Anexo B.2 presenta el diagrama de bloques del sistema tras la conversión a TMR. Se aprecia que todos los bloques que se seleccionaron para ser triplicados ya no aparecen, y en su lugar se encuentra el subsistema TMR (nombrado como *tmr_system*). El diagrama de bloques de dicho subsistema se halla en el Anexo B.3, y en él se observan claramente los 3 sub-bloques triplicados: MB1, MB2 y MB3. Se ha expandido el sub-bloque MB1 en el Anexo B.4. En estos sub-bloques se encuentran todos los bloques que se seleccionaron para ser triplicados.

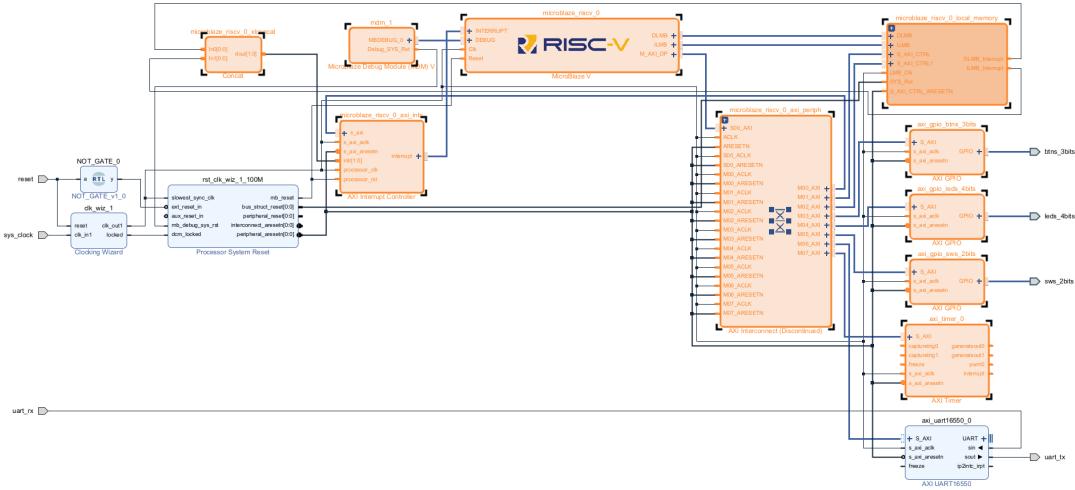


Fig. 3.5. Bloques seleccionados para triplicarse. Se han añadido a la jerarquía que encapsula el subsistema TMR generado posteriormente por el núcleo IP TMR Manager.

TMR SEM IP: Soft Error Mitigation Controller v4.1 LogiCORE IP

A lo largo del documento se ha reiterado la importancia de establecer mecanismos de mitigación de *soft errors* que complementen a la arquitectura TMR. En caso contrario, los errores se acumularían rápidamente y el sistema se detendría. El subsistema TMR MicroBlaze gestiona la detección y corrección de errores de la memoria de configuración mediante el IP TMR SEM, el cual contiene las configuraciones más comunes del *Soft Error Mitigation Controller v4.1 LogiCORE IP*. Para aplicaciones más específicas, se puede establecer un controlador SEM externo, pero para este caso se utilizará el IP proporcionado. Ofrece 5 funciones: inicialización; e inyección, detección, corrección y clasificación de errores —siendo la inicialización y la detección de errores las únicas funciones no opcionales [24]—. Los métodos de detección y corrección de errores disponibles en el *Soft Error Mitigation Controller v4.1 LogiCORE IP* son los siguientes [24]:

1. **Repair** Basado en un algoritmo ECC. Soporta la corrección de cualquier error de un bit en la memoria de configuración. Además, permite la corrección de múltiples bits si se distribuyen de forma unitaria en diferentes *frames*. Esta última característica es resultado de que cada *frame* contenga una palabra reservada para ECC. Más adelante se desarrollará la estructura de los *frames* y cómo estos componen el *bitstream*, la secuencia de bits que determina el estado de la memoria de configuración.
2. **Enhanced Repair** Basado en algoritmos ECC y CRC. Soporta la corrección de cualquier error de un bit, así como de errores dobles adyacentes, en la memoria de configuración. Además, permite la corrección de múltiples bits si estos se distribuyen de forma unitaria o adyacentes en parejas, siempre que en ambos casos sea en diferentes *frames*.

3. **Replace** Basado en la reconfiguración. Permite la corrección de errores arbitrarios. Es capaz de corregir cualquier *event upset*, independientemente del número de bits que involucre y su distribución.

El método de corrección más potente es, por lo tanto, el de reemplazo. Sin embargo, este requiere una copia externa del estado original de la memoria de configuración que no haya sido alterada y, además, no está disponible en el IP TMR SEM. De esta forma, el método más efectivo que ofrece es el de reparación mejorada, por lo que fue el elegido. Además de realizar *scrubbing* en la memoria de configuración (la cual determina el comportamiento del diseño), es igual de importante proteger los elementos de la memoria de diseño (la cual determina el estado del mismo). Las siguientes categorías de memoria pertenecen a la memoria de diseño: memoria de bloque, memoria distribuida, biestables de los CLBs, registros internos de control del dispositivo y otros elementos de estado [50]. Para lograr proteger la memoria de diseño, se han de utilizar códigos de corrección de errores. Si no se hace, el *scrubber* de la memoria de configuración no detectará ni reparará ningún error y el sistema TMR será el único capaz de enmascarar el fallo, pero lo acumulará. Por ello, se activó la opción de ECC de los controladores de las memorias de bloque en cada sub-bloque triplicado. Esta es una funcionalidad de los controladores de la LMB BRAM, y no del IP TMR SEM, que tan solo realiza el *scrubbing* en la memoria de configuración.

Para monitorizar el estado del controlador del IP TMR SEM, se puede interactuar con la *Monitor Interface* del IP mediante comandos ASCII enviados por protocolo AXI o UART [24]. En este caso, se utilizó el protocolo UART para llevar a cabo la monitorización desde un ordenador externo. No obstante, el asistente de diseño construye el subsistema TMR con la interfaz de monitorización del IP TMR SEM configurada en AXI en lugar de UART. Aunque se modifique la configuración del IP para obtener la interfaz UART, el asistente de diseño no es capaz de reorganizar la arquitectura del subsistema, por lo que la conversión se ha de realizar manualmente. Por ello, se lista la sucesión de acciones necesaria:

1. Configurar la interfaz de monitorización del IP en UART.
2. Eliminar el TMR *Voter* correspondiente al IP TMR SEM.
3. Eliminar el TMR *Comparator* correspondiente al IP TMR SEM en cada sub-bloque triplicado.
4. Eliminar todas las conexiones obsoletas, incluyendo la correspondiente a la señal *Interrupt* que el IP TMR SEM tenía al estar configurada en AXI.
5. Modificar el número de comparadores conectados al TMR *Manager* de cada sub-bloque triplicado para disminuirlo en uno.

6. Modificar el número de *Master Interfaces* del AXI *Interconnect* de cada sub-bloque triplicado para disminuirlo en uno.
7. Modificar el número de entradas del bloque de concatenación de interrupciones en cada sub-bloque triplicado para disminuirlo en uno. Si al hacerlo, alguna de las conexiones necesarias (no relativas al IP TMR SEM) se elimina, se han de reconectar.
8. Convertir los pines Rx y Tx del IP TMR SEM en puertos externos.

ZYNQ7 Processing System

Más allá de monitorizar el IP SEM desde un ordenador externo, lo cual se puede realizar directamente, hay ciertas tareas restantes que requieren un procesamiento externo al subsistema TMR. Estas tareas son la activación del puerto ICAP, la lectura del estado del subsistema TMR y la creación del puente UART ya mencionado que permita mantener tres conexiones paralelas e independientes utilizando únicamente dos módulos PmodUSBUART. Para todo ello, se hizo uso del PS del SoC. Si se quiere utilizar el PS en conjunción con el PL, se ha de añadir el IP ZYNQ7 Processing System en el integrador de IPs. En la ventana de configuración, se activaron tanto el controlador UART0 (el cual permite la comunicación serial externa con el PS mediante el puerto USB UART) como el UART1 en pines EMIO (el cual permite la comunicación serial entre el PS y el PL). En la figura 3.6 se muestra la configuración que se utilizó.

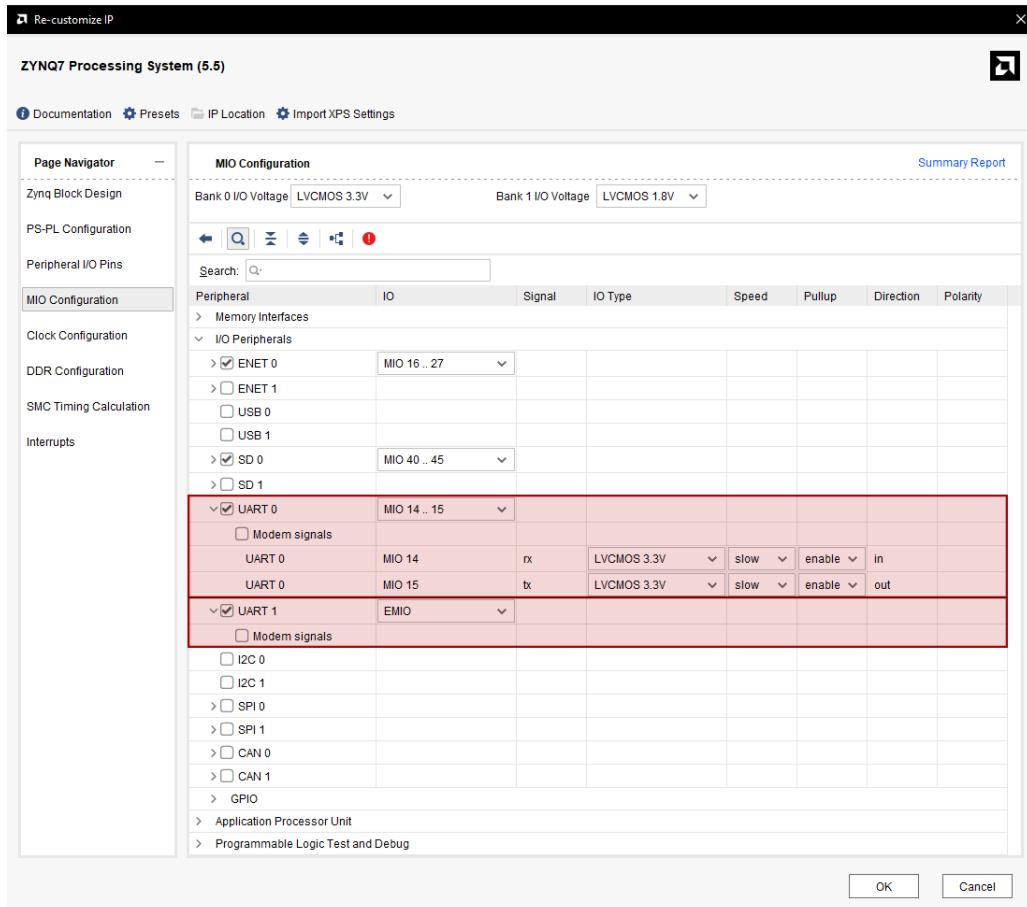


Fig. 3.6. Configuración de los controladores UART en la ventana de personalización del IP ZYNQ7000 PS.

Al activar el controlador UART1 con pines EMIO, la interfaz del bloque ZYNQ7000 PS cambia, añadiendo los pines del UART1. Dichos pines se conectaron a los correspondientes del IP AXI UART16550 que aparece en el Anexo B.2, reemplazando sus puertos externos. Se renombró el núcleo IP a *axi_uart16550_tmr* para indicar que ese bloque UART se corresponde con el canal de comunicaciones entre el ordenador externo y el código que se ejecuta en los MicroBlaze del subsistema TMR.

Para poder leer el estado del subsistema TMR, se conectaron los pines *Status* de cada TMR Manager (hay uno en cada sub-bloque) a un IP AXIGPIO (*axi_gpio_status_1*, *axi_gpio_status_2* y *axi_gpio_status_3*). Estos están conectados al PS mediante protocolo AXI y mapeados en memoria, lo que permite la lectura del estado del subsistema TMR desde código ejecutado en el PS. Para enviar las señales de estado al ordenador externo por un canal independiente, se añadió otro IP AXI UART16550, con nombre *axi_uart16550_status*, también conectado al PS con protocolo AXI. Es importante resaltar que el *axi_uart16550_tmr*, no está conectado a la red del PS, sino que se encuentra en el espacio de direcciones del subsistema TMR.

Tras estas modificaciones, la comunicación con los MicroBlaze se realiza a través del puerto USBUART de la placa, y la comunicación con el monitor SEM y la recepción del estado del subsistema TMR, a través de los módulos PmodUSBUART conectados a los puertos Pmod. El diagrama de bloques del diseño final tolerante a fallos se encuentra en el Anexo B.5. Asimismo, los correspondientes al diseño final del subsistema TMR y del sub-bloque MB1 (con la misma arquitectura que el resto de sub-bloques), se hallan en los Anexos B.6 y B.7, respectivamente.

El último paso a realizar en el integrador de IPs es la asignación de direcciones en el *Address Editor*. Este es un proceso que normalmente se lleva a cabo de forma automática. Sin embargo, en este caso el asistente de diseño no lo realizó de forma correcta, por lo que se tuvo que hacer manualmente. Es un requisito indispensable para el correcto funcionamiento del subsistema TMR que todas las direcciones de memoria asignadas entre elementos triplicados análogos sean iguales. A su vez, todos los elementos —no análogos— dentro de un mismo espacio de direcciones han de tener direcciones diferentes. Las direcciones asignadas por el asistente de diseño se solapaban (esto es, había diferentes interfaces en un mismo espacio de direcciones con la misma dirección asignada). Para solucionarlo, se desasignaron todas las direcciones y se fueron estableciendo de una en una respetando los requisitos expuestos. Es importante resaltar que tras desasignar las direcciones de los segmentos de memoria, se ha de volver a seleccionar el rango de direcciones adecuado (32K en este caso), ya que este se desconfigura. Las direcciones de memoria que se asignaron se muestran en el Anexo C.

Antes de continuar con la siguiente fase, se creó un *HDL wrapper* para el diseño de bloques. Este expone todos los puertos e interfaces externas del diseño y lo marca como *top module*. Dicho módulo es aquel con la mayor jerarquía en el diseño. En el explorador de fuentes de diseño del proyecto aparecen dos: el diseño del sistema TMR encapsulado en su respectivo *wrapper* recién creado, y el módulo NOT_GATE. Este último no es el *top module*, sino que está integrado en él.

3.3.2. RTL Analysis

La siguiente etapa de la metodología AMD UltraFast es la asignación de restricciones lógicas y físicas (ver figura 3.1), la cual se lleva a cabo en las fases *RTL Analysis* y *Synthesis* de Vivado. Además, es un proceso iterativo, por lo que se deberán cambiar de acuerdo a los informes de tiempo, consumo energético u otros análisis que se realicen en fases posteriores.

En la fase de *RTL Analysis*, se verifica el diseño y se elabora, transformando los archivos RTL a una *netlist*. La *netlist* se presenta de forma jerárquica, comenzando desde el *top module*, y se compone de diferentes elementos:

- **Nets** Las *nets* son las conexiones entre los pines de los componentes (*cells*) del circuito. Hacen referencia a bits individuales: esto es, los buses están compuestos de múltiples *nets*. Una *net* no es un cable (*wire*), ya que este es el elemento físico utilizado para la implementación de una *net*, la cual se compone de pines, puertos y cables interconectados [55].
- **Cells**
 - **Hierarchical cell** Una *cell* jerárquica es una abstracción lógica que encapsula elementos más concretos de la *netlist*, llegando finalmente a una *cell* primitiva.
 - **Primitive cell** Una *cell* primitiva, también conocida como *leaf cell*, es aquella que no encapsula una lógica más concreta, sino que es un elemento lógico indivisible. La tabla 3.3 muestra los diferentes tipos de *cells* primitivas que existen.

Se ha incluido en el Anexo B.8 un detalle del esquema de la *netlist* generado al elaborar el diseño RTL final. En concreto, se presenta el esquema de la *cell* jerárquica *tmr_design_i/tmr_system/MB1/tmr_reset_0/U0/SEQ*. El nombre de la *cell* indica el nivel de anidación desde el *top module tmr_design_i*. El resto de niveles intermedios también son *cells* jerárquicas. Se ha escogido el esquema de esta *cell* porque ejemplifica correctamente todos los elementos de la *netlist*: en verde claro aparecen las *nets*; en verde oscuro, los buses formados por múltiples *nets*; en azul, otras *cells* jerárquicas; y en amarillo, las *cells* primitivas. Entre las *cells* primitivas hay LUTs —del grupo CLB— y FDREs —del grupo REGISTER— (ver tabla 3.3 [55]).

PRIMITIVE_GROUP	PRIMITIVE_SUBGROUP	PRIMITIVE_TYPE
BLOCKRAM	BRAM	BLOCKRAM.BRAM.RAMB18E2 BLOCKRAM.BRAM.RAMB36E2
CLB	CARRY	CLB.CARRY.CARRY8
	LUT	CLB.LUT.LUT1 CLB.LUT.LUT2 CLB.LUT.LUT3 CLB.LUT.LUT4 CLB.LUT.LUT5 CLB.LUT.LUT6
	LUTRAM	CLB.LUTRAM.RAM32M CLB.LUTRAM.RAM32M16 CLB.LUTRAM.RAM32X1D
	MUXF	CLB.MUXF.MUXF7 CLB.MUXF.MUXF8
	SRL	CLB.SRL.SRL16E CLB.SRL.SRLC16E CLB.SRL.SRLC32E
	Others	CLB.others.LUT6_2
	BUFFER	CLOCK.BUFFER.BUFGCE CLOCK.BUFFER.BUFGCE_DIV
CLOCK	PLL	CLOCK.PLL.MMCME3_ADV CLOCK.PLL.PLLE3_ADV
	BSCAN	CONFIGURATION.BSCAN.BSCANE2
I/O	BDIR_BUFFER	I/O.BDIR_BUFFER.IOBUFDS
	BITSLICE	I/O.BITSLICE.BITSLICE_CONTROL I/O.BITSLICE.RIU_OR I/O.BITSLICE.RXTX_BITSLICE I/O.BITSLICE.TX_BITSLICE_TRI
	INPUT_BUFFER	I/O.INPUT_BUFFER.HPIO_VREF I/O.INPUT_BUFFER.IBUF I/O.INPUT_BUFFER.IBUFDS
	OUTPUT_BUFFER	I/O.OUTPUT_BUFFER.IOBUFE3 I/O.OUTPUT_BUFFER.OBUF I/O.OUTPUT_BUFFER.OBUFDS
OTHERS	others	others.others.others OTHERS.others.AND2B1L OTHERS.others.GND OTHERS.others.VCC
REGISTER	SDR	REGISTER.SDR.FDCE REGISTER.SDR.FDPE REGISTER.SDR.FDRE REGISTER.SDR.FDSE
RTL_GATE	buf	RTL_GATE.buf.RTL_INV
	logical	RTL_GATE.logical.RTL_AND RTL_GATE.logical.RTL_OR RTL_GATE.logical.RTL_XOR
RTL_MEMORY	ram rom	RTL_MEMORY.ram.RTL_RAM RTL_MEMORY.rom.RTL_ROM
RTL_MUX	mux	RTL_MUX.mux.RTL_MUX
RTL_OPERATOR	arithmetic	RTL_OPERATOR.arithmetic.RTL_ADD RTL_OPERATOR.arithmetic.RTL_MULT RTL_OPERATOR.arithmetic.RTL_SUB
	equality	RTL_OPERATOR.equality.RTL_EQ
	shift	RTL_OPERATOR.shift.RTL_RSHIFT
RTL_REGISTER	flop	RTL_REGISTER.flop.RTL_REG

TABLA 3.3. CLASIFICACIÓN DE LAS *CELLS* PRIMITIVAS EN GRUPOS, SUBGRUPOS Y TIPOS

Con la *netlist* generada, se pueden establecer restricciones para realizar *I/O Planning* y *Floorplanning*. La planificación de entradas y salidas consiste en relacionar los *package pins* de la placa con los puertos del diseño —esto es, con las *nets I/O* del *top module*—. Para ello, se descargó el archivo *XDC Constraints* de la placa PYNQ-Z1 desde su página de referencia [56]. Los archivos de restricciones XDC (*Xilinx Design Constraints*) se basan en la versión 1.9 del formato estándar SDC (*Synopsys Design Constraints*), pero añaden restricciones físicas propietarias de AMD [57]. El archivo contiene una línea por cada pin de la placa, indicando el pin, su estándar y el puerto al que está asignado. En el archivo de restricciones de la PYNQ-Z1, hay comentarios indicando la equivalencia entre los *package pins* y la nomenclatura utilizada en el esquema del circuito de la placa (disponible en su referencia *online* [56][58]). Todas las líneas aparecen comentadas, y tan solo hay que descomentar las referentes a los pines que se vayan a utilizar. En este caso, se descomentaron las líneas de los pines de la señal de reloj, de los interruptores, de los LEDs, de los botones y de los pines relevantes de las cabeceras Pmod A y B. A su vez, se indicó en cada línea el nombre del puerto al que se conectó cada pin. El nombre ha de coincidir con el de la *net I/O* de la *netlist*. Se ha incluido el archivo XDC en el Anexo D.

Para identificar los *package pins* relevantes de las cabeceras Pmod A y B, se observó en primer lugar el diagrama de la figura 3.7, obtenido del manual de referencia de la placa [49]. En él aparecen numerados los pines, señalando además cuáles son de señal y cuáles de VCC y GND. A continuación, se comparó la numeración de los pines con la que aparece en el esquema del circuito publicado en la página de referencia de la placa [56] (ver figura 3.8). Así, se identificó que la primera fila de pines se corresponde con la nomenclatura JA1_P, JA1_N, JA2_P y JA2_N del esquema. Con esta información, y como muestra la figura 3.9, se conectó el módulo PmodUSBUART a la primera fila de la cabecera Pmod A, y se asignaron los puertos UART_Tx_SEM y UART_Rx_SEM a los pines JA1_N (Y19) y JA2_P (Y16), respectivamente. Como se observa en el Anexo D, se hizo lo mismo para conectar el segundo módulo PmodUSBUART a la primera fila de la cabecera Pmod B y asignar sus pines a los puertos UART_Tx_Status y UART_Rx_Status. En la figura 3.9 se ha indicado la posible conexión de las señales módem (CTS —*Clear To Send*— y RTS —*Request To Send*—), pero estas no se utilizan en el diseño, por lo que no aparecen en el archivo de restricciones.

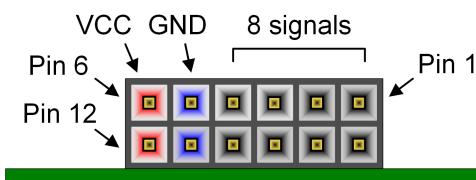


Fig. 3.7. Diagrama del puerto Pmod con la numeración de sus pines [49].

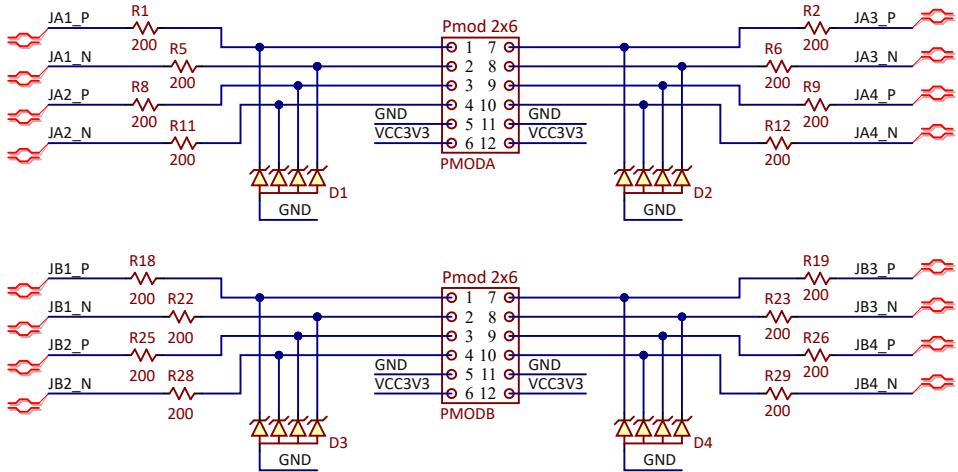


Fig. 3.8. Esquema del circuito de las cabeceras Pmod A y B de la placa PYNQ-Z1 [58].

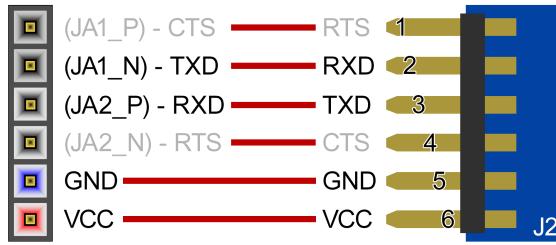
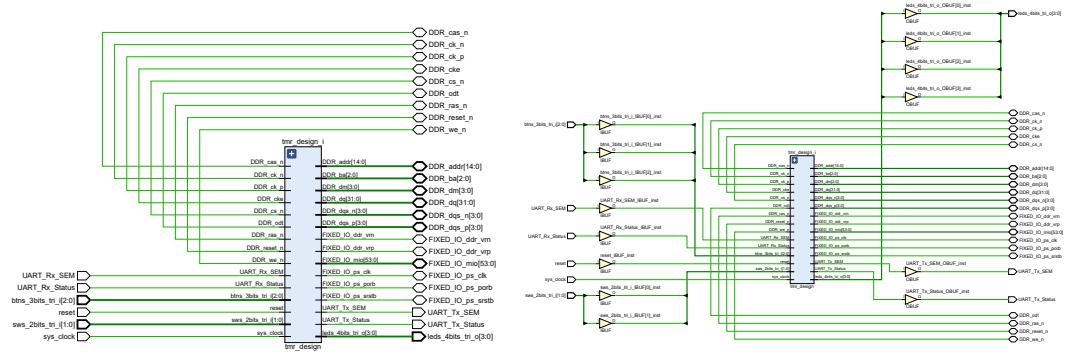


Fig. 3.9. Conexión del módulo PmodUSBUART a la primera fila de pines de la cabecera Pmod A.

Con respecto al *Floorplanning*, este consiste en modificar cómo se realizará la implementación física de la *netlist* en el *hardware* de la FPGA con el objetivo de mejorar el rendimiento reduciendo la holgura temporal de las rutas críticas con más demora o aumentando la densidad lógica del circuito. Para llevarlo a cabo, se puede guiar a la herramienta de implementación sin establecer unas restricciones rígidas, o especificar el emplazamiento detallado de cada *cell* [59]. En la primera iteración del proceso de diseño, el circuito implementado no cumplió con los requisitos temporales. Por ello, se debió regresar a esta fase para mejorar el *Floorplanning*. Se optó por guiar a la herramienta mediante el uso de Pblocks. Estos son conjuntos que agrupan *cells* con lógica relacionada. Así, se puede especificar un área donde colocar las *cells* del Pblock y la herramienta de implementación las intentará colocar dentro de esta siempre que sea posible [55]. En este caso no se especificó ningún área, y tan solo se agruparon las *cells* de cada sub-bloque del subsistema TMR para informar a la herramienta de implementación de que cada sub-bloque ha de tener una alta densidad lógica. Al no asignarle un área a cada Pblock, las agrupaciones se tomarán como sugerencias. Al final del archivo de restricciones del Anexo D se halla el código que crea y asigna los tres Pblocks mencionados: *pblock_MB1*, *pblock_MB2* y *pblock_MB3*.

3.3.3. Synthesis

La fase de síntesis lógica de la metodología AMD UltraFast consiste en generar la *netlist*, pero teniendo en cuenta ciertas restricciones establecidas durante el análisis RTL —como las de I/O—. En la figura 3.10 se compara el esquema del *top module* de la *netlist* en ambas fases. En el correspondiente a la fase de síntesis, los puertos del diseño aparecen conectados a los pines de la placa, mientras que en la *netlist* previa están desconectados. Las *cells* primitivas que se aprecian en el esquema de la *netlist* sintetizada (*Input* y *Output Buffers* —ver tabla 3.3—) ya están colocadas. Esto es, ya tienen su posición física en la placa determinada, incluso antes de la fase de implementación. Esto se debe a que las restricciones físicas de I/O definen el pin al que se conecta cada puerto, y ese pin tiene una posición fija en la placa, por lo que en esta fase ya se conoce el emplazamiento físico de esas *cells*. Por ejemplo, el *Input Buffer* relativo a un puerto conectado al *Package Pin* D20, está colocado en el *Site D20*, en el *BEL* correspondiente. Más adelante se detallará la estructura física de las FPGAS de Xilinx.



(a) Esquema del *top module* de la *netlist* RTL. (b) Esquema del *top module* de la *netlist* sintetizada.

Fig. 3.10. Comparación de las *netlist* RTL y sintetizada.

Por otro lado, en esta fase, además de poder hacer *I/O Planning* y *Floorplanning*, se puede realizar un análisis de tiempo preliminar a través del informe *Check Timing*, el cual informa sobre las restricciones temporales faltantes o rutas con restricciones que han de ser revisadas [59]. En la figura 3.11 se muestra el informe que se generó. Como se ve, advierte de que no se han especificado restricciones de retraso temporal a los puertos. Sin embargo, esto solo es relevante si se envían o reciben datos sincronizados por reloj a o desde otro dispositivo. Como no es el caso, se pueden ignorar estos avisos o silenciarlos añadiendo restricciones temporales de *False Path* a las rutas que los generan. Estas son excepciones para rutas que existen topológicamente, pero no son funcionales o no necesitan ser incluidas en los análisis de tiempo [57].

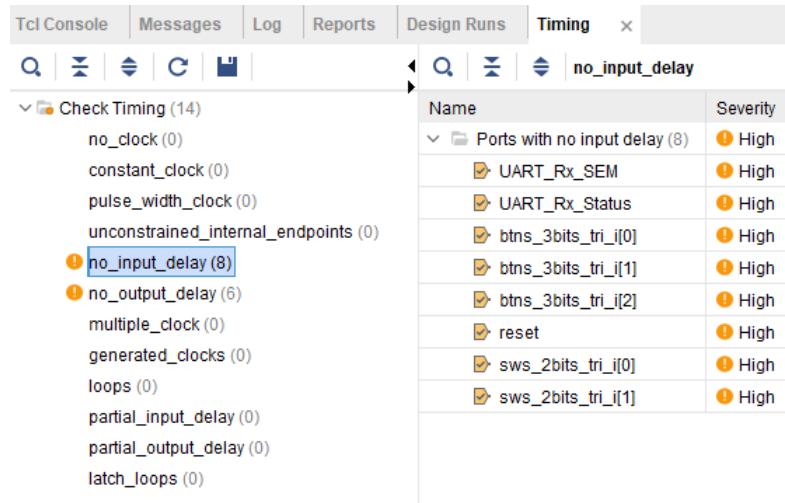


Fig. 3.11. Informe *Check Timing* que muestra avisos graves por no asignar restricciones de retraso temporal a los puertos de entrada y salida.

3.3.4. Implementation

La siguiente fase de la metodología AMD UltraFast es la implementación del diseño. Esta consiste en la asignación de cada una de las *cells* primitivas a un BEL de la placa. Para comprender cómo funciona, se ha de conocer la arquitectura de las FPGAs de Xilinx, que se organiza de forma jerárquica de la siguiente manera, ilustrada en la figura 3.12:

- **Device** El dispositivo es el nivel más elevado en la jerarquía de la arquitectura de las FPGA [60].
- **SLR (Super Logic Region)** Una SLR es una sección de un *die* de un dispositivo SSI (*Stacked Silicon Interconnect*). Estos son dispositivos FPGA de gran tamaño construidos mediante la combinación de varios componentes SLR apilados verticalmente y conectados por un intermediador. Las SLR se nombran de abajo a arriba, desde SLR0 e incrementando el índice según se asciende verticalmente. Así, en un dispositivo SSI de 4 componentes SLR, estos se identifican con SLR0, SLR1, SLR2 y SLR3, desde el inferior al superior [61].
- **FSR (Fabric Sub Region)** Las FSRs, más conocidas como *Clock Regions*, son matrices bidimensionales de *Tiles*. En dispositivos SSI, las SLR están formadas por una matriz bidimensional de FSRs. En dispositivos de un solo *die* —los cuales no contienen SLRs—, el propio dispositivo es una matriz bidimensional de FSRs. Las regiones de reloj se nombran en la forma $X\#Y\#$, donde $X\#$ indica la posición horizontal en la matriz e $Y\#$, la vertical [60]. La región de reloj $X0Y0$ se encuentra en la esquina inferior izquierda del dispositivo.

- **Tile** Las *Tiles* son los elementos que conforman las regiones de reloj formando una matriz bidimensional que abarca toda la FPGA. Los cables que conectan eléctricamente diferentes *Tiles* se denominan nodos. Los nodos se conectan para enrutar las *nets* mediante PIPs (*Programmable Interconnect Points*), multiplexores programables que se configuran para unir cables. Existen distintos tipos de *Tiles*, y cada tipo contiene un único tipo de recurso o función. Los tipos más importantes son los de interconexión izquierdos y derechos (INT_L e INT_R) y los CLBs izquierdos y derechos de lógica (CLBLL_L y CLBLL_R) y memoria (CLBLM_L y CLBLM_R). Las *Tiles* de interconexión, también llamadas *switch boxes*, albergan la mayoría de PIPs [60][62]. Los nombres de las *Tiles* se presentan en el formato <TYPE>_X#Y#, y aquella con XYOY está en la esquina inferior izquierda.
- **Site** Un *Site* es un elemento de las *Tiles* (las cuales pueden no tener ninguno, tener uno, o tener más) que agrupa principalmente tres objetos: BELs, *Site pins* y *Site wires*. Los *Site pins* son los pines externos del *Site*, mientras que los *Site wires* conectan elementos entre sí y a los *Site pins*. Los *Sites* de una *Tile* CLB se denominan *Slices*, y existen dos tipos: SLICEM —*Slices* de memoria que pueden ser configuradas como RAM distribuida— y SLICEL —*Slices* lógicas que no se pueden configurar como memoria—. Una *Tile* CLB lógica contiene dos *Slices* lógicas, mientras que una de memoria contiene una *Slice* lógica y otra de memoria. Los nombres de los *Sites* también terminan en el sufijo _X#Y#. No obstante, a diferencia de las regiones de reloj y las *Tiles*, en los cuales las coordenadas son globales —a nivel de la FPGA—, cada tipo de *Site* se rige por sus propias coordenadas [60][62].
- **BEL (Basic Element of Logic)** Un BEL es la unidad atómica de una FPGA. Es el equivalente físico del concepto lógico de *cell* primitiva (ver figura 3.13). Existen dos tipos de BELs: lógicos y de enrutado. Durante la implementación del diseño, las *cells* primitivas se asignan a BELs lógicos. El concepto de colocar una *cell* en la FPGA no es, pues, sino mapearla a un BEL lógico. Los BELs de enrutado, también llamados *Site PIPs*, no se mapean con *cells*, sino que son multiplexores programables que enrutan las señales entre los pines de BELS y/o *Site pins*. No obstante, incluso los BELs lógicos se pueden utilizar para enrutar una *net* si es necesario, lo que se conoce como *route through* o *route-thru* [60][55].

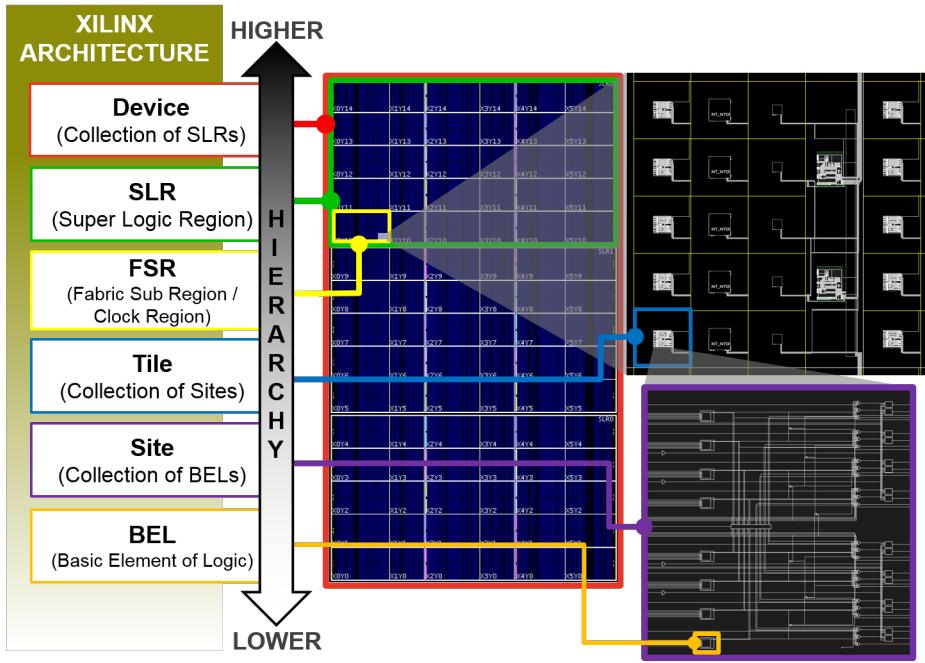


Fig. 3.12. Jerarquía arquitectural de una FPGA de Xilinx [60].

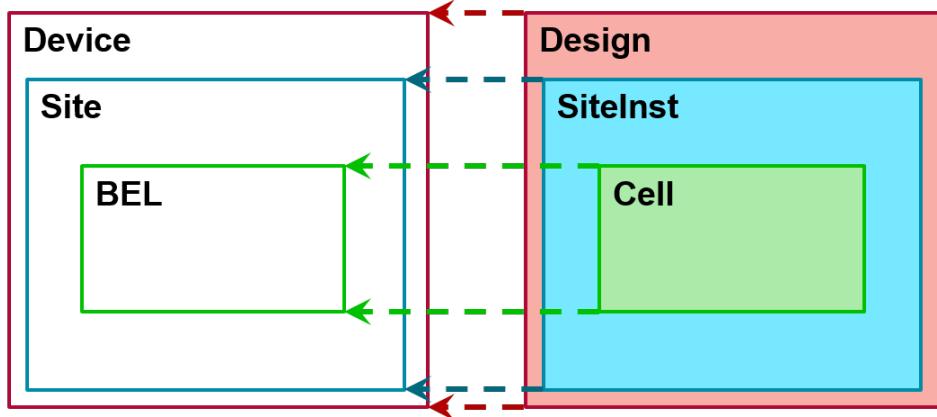


Fig. 3.13. Mapeos entre elementos lógicos (derecha) y físicos (izquierdo) [60].

La *netlist* generada en la implementación, conocida como *device netlist* o *physical netlist*, contiene por lo tanto todos los mapeos entre elementos lógicos y físicos. La primera *netlist* que se generó —en la fase de análisis RTL— era puramente lógica. La segunda, la *netlist* sintetizada, incluía los mapeos especificados en las restricciones y, finalmente, la *netlist* física implementa todos los mapeos lógico-físicos. En el Anexo B.9 se muestra el esquema de la *cell* lógica *tmr_design_i/tmr_system/MB1/tmr_reset_0/U0/SEQ*, pero tras la implementación. Por lo tanto, todas las *cells* primitivas que se ven ya están mapeadas a un *BEL* lógico. Para visualizar la implementación de una *cell*, se ha marcado *tmr_design_i/tmr_system/MB1/tmr_reset_0/U0/SEQ/pr_dec0* en rojo, al igual que en la figura 3.14. En esta, se observa la ubicación en el dispositivo del *BEL* al que se ha mapeado. En concreto, la *cell* (una *look-up table* de 4 bits) se ha implementado en el *BEL*

B6LUT (una *look-up table* de 6 bits), del *Site* SLICE_X87Y118 (una *Slice* lógica), en el *Tile* CLBLM_L_X54Y118 (un CLB de memoria izquierdo), en la región de reloj X1Y2. El resto de BELs utilizados para implementar el diseño se muestran en color celeste.



Fig. 3.14. Asignación física y ubicación en el dispositivo de la *cell* primitiva *tmr_design_i/tmr_system/MB1/tmr_reset_0/U0/SEQ/pr_dec0* (en rojo).

La elección de cómo llevar a cabo el mapeo de todos los elementos de la *netlist* sintetizada que no estén preasignados con restricciones la realiza la herramienta de implementación. Esta actúa en pasos secuenciales que se pueden ejecutar individualmente si así se desea [63]:

1. ***Opt Design*** Optimiza el diseño lógico para facilitar su cabida en el dispositivo objetivo.
2. ***Power Opt Design* (Opcional)** Optimiza elementos del diseño para reducir la demanda energética del dispositivo objetivo.
3. ***Place Design*** Coloca el diseño en el dispositivo objetivo. Esto es, realiza el mapeo de *cells* primitivas a BELs. Además, replica partes de la lógica para mejorar el rendimiento temporal.

4. ***Post-Place Power Opt Design*** (Opcional) Optimiza aún más el consumo energético tras la colocación.
5. ***Post-Place Phys Opt Design*** (Opcional) Optimiza la lógica y la colocación utilizando estimaciones temporales basadas en la colocación. Además, replica registros de señales con demasiada dispersión para reducir cuellos de botella en rutas críticas [52].
6. ***Route Design*** Enruta el diseño al dispositivo objetivo.
7. ***Post-Route Phys Opt Design*** (Opcional) Optimiza la lógica, la colocación y el enrutamiento utilizando los retrasos temporales reales de las rutas.

Al terminar la implementación, la metodología AMD UltraFast indica que se ha de cerrar el diseño mediante el análisis del rendimiento, el análisis temporal, y el análisis de potencia. Se realizó el análisis temporal del diseño y se obtuvieron los resultados expuestos en la figura 3.15. Estos se dividen en tres secciones: *Setup* —el tiempo antes del cual los nuevos datos estables han de estar disponibles antes del siguiente flanco activo del reloj para ser capturados correctamente—, *Hold* —el tiempo que los datos han de permanecer estables tras el flanco activo del reloj para evitar capturar valores indeseados— y *Pulse Width* —requisitos de periodo mínimo y máximo y de duración del pulso alto y bajo de cada pin de reloj, así como del sesgo máximo del reloj entre dos pines de reloj de una misma *leaf cell*— [52][59]. El diseño superó tanto los requisitos de *Hold* como los de *Pulse Width*. Sin embargo, existían 4684 *endpoints* que no cumplían con los requisitos de *Setup*. Si se analizan el WNS (*Worst Negative Slack*) y el TNS (*Total Negative Slack*), se observa que tienen valores muy negativos, cuando tendrían que ser cercanos a cero o incluso positivos en el caso del WNS. Este último indica la holgura temporal de los requisitos de *Setup* del camino crítico del circuito. Esto es, la de la ruta con menor holgura temporal. Una holgura positiva indica el tiempo sobrante con respecto al necesario para llegar a los requisitos de *Setup*, mientras que una negativa indica el tiempo faltante. Así, el resultado obtenido de WNS de -5,951 ns muestra que hay una ruta en el circuito que no es capaz de proveer los datos a tiempo de forma fiable por prácticamente 6 ns, lo cual es más de la mitad del periodo de la señal de reloj utilizada (100 MHz, periodo de 10 ns). Valores de WNS menores que -1 ns se consideran violaciones importantes de los requisitos temporales. El TNS suma todas las holguras negativas del diseño, por lo que no puede ser positiva y su valor óptimo sería cero. En cambio, se obtuvo un TNS de aproximadamente -15992 ns.

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): -5,951 ns	Worst Hold Slack (WHS): 0,056 ns	Worst Pulse Width Slack (WPWS): 0,000 ns
Total Negative Slack (TNS): -15992,466 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 4684	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 32960	Total Number of Endpoints: 32960	Total Number of Endpoints: 12459

Timing constraints are not met.

Fig. 3.15. Resumen del informe temporal del diseño conectado a una señal de reloj de 100 MHz.

Un informe temporal desfavorable implica realizar cambios en etapas anteriores de la metodología. Como muestra la figura 3.16, es un proceso iterativo que, si se realiza correctamente, deriva en el cierre temporal del diseño (el cumplimiento de todos los requisitos temporales). Así pues, tras los resultados negativos del informe temporal, se decidió modificar el diseño disminuyendo la frecuencia de la señal de reloj utilizada de 100 MHz a 50 MHz. El diseño no tolerante a fallos (ver Anexo B.1) cumplía los requisitos temporales utilizando un reloj de 100 MHz. Sin embargo, la triplicación de los sub-bloques y toda la infraestructura de enrutamiento que se añade en la conversión a TMR supone un aumento sustancial de la complejidad del diseño y la longitud de sus rutas. En la documentación de los núcleos IP MicroBlaze TMR se informa de que la frecuencia del subsistema TMR se puede ver reducida por el nivel lógico adicional que añaden los votantes [46]. Con respecto a la modificación de la frecuencia a la que funciona el subsistema TMR, hay que tener en cuenta que al utilizar el SEM interno, la frecuencia máxima del subsistema TMR está limitada a aquella del subnúcleo SEM: ambas han de coincidir [46]. El subnúcleo SEM admite frecuencias de 8 a 100 MHz. Para realizar el cambio de frecuencias se recomienda dejar la frecuencia del subnúcleo SEM en automático y ajustar la del reloj que utiliza el subsistema TMR para que esté en el rango válido de 8-100 MHz. Así, se redujo la frecuencia del reloj a 50 MHz y se volvió a sintetizar el diseño. Además de este cambio al diseño, se configuró el paso opcional de *Post-Route Phys Opt Design* de la herramienta de implementación.

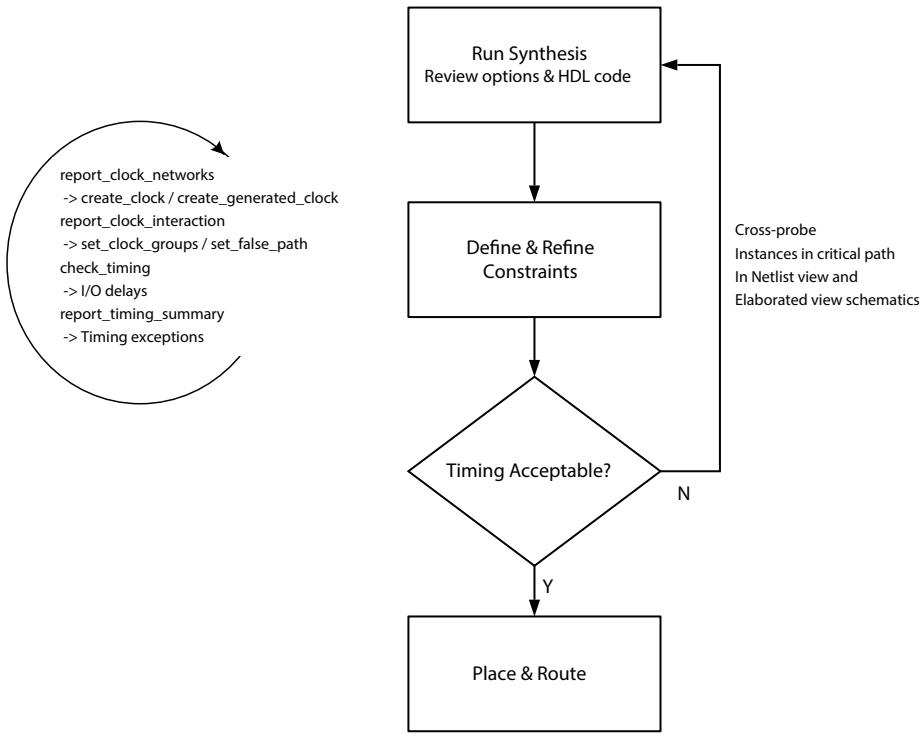


Fig. 3.16. Proceso iterativo para lograr el cierre temporal (*Timing Closure*) del diseño [52].

Con estos cambios y como se ve en la figura 3.17, el diseño superó el análisis temporal, cumpliendo todos los requisitos temporales. En la figura 3.18 se encuentra una representación del dispositivo, con los BELs utilizados marcados. Aquellos mapeados a *leaf cells* de alguno de los sub-bloques del subsistema TMR aparecen en amarillo, magenta o verde (MB1, MB2 y MB3, respectivamente). Se aprecia que las *cells* de cada sub-bloque se agrupan formando clústeres relativamente definidos, lo que es una consecuencia del la optimización realizada por la herramienta de implementación, que ha agrupado la lógica relacionada para disminuir el coste temporal de mover información entre elementos lógicos adyacentes.

Design Timing Summary		
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0,494 ns	Worst Hold Slack (WHS): 0,015 ns	Worst Pulse Width Slack (WPWS): 2,000 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 32896	Total Number of Endpoints: 32896	Total Number of Endpoints: 12426

All user specified timing constraints are met.

Fig. 3.17. Resumen del informe temporal del diseño conectado a una señal de reloj de 50 MHz.

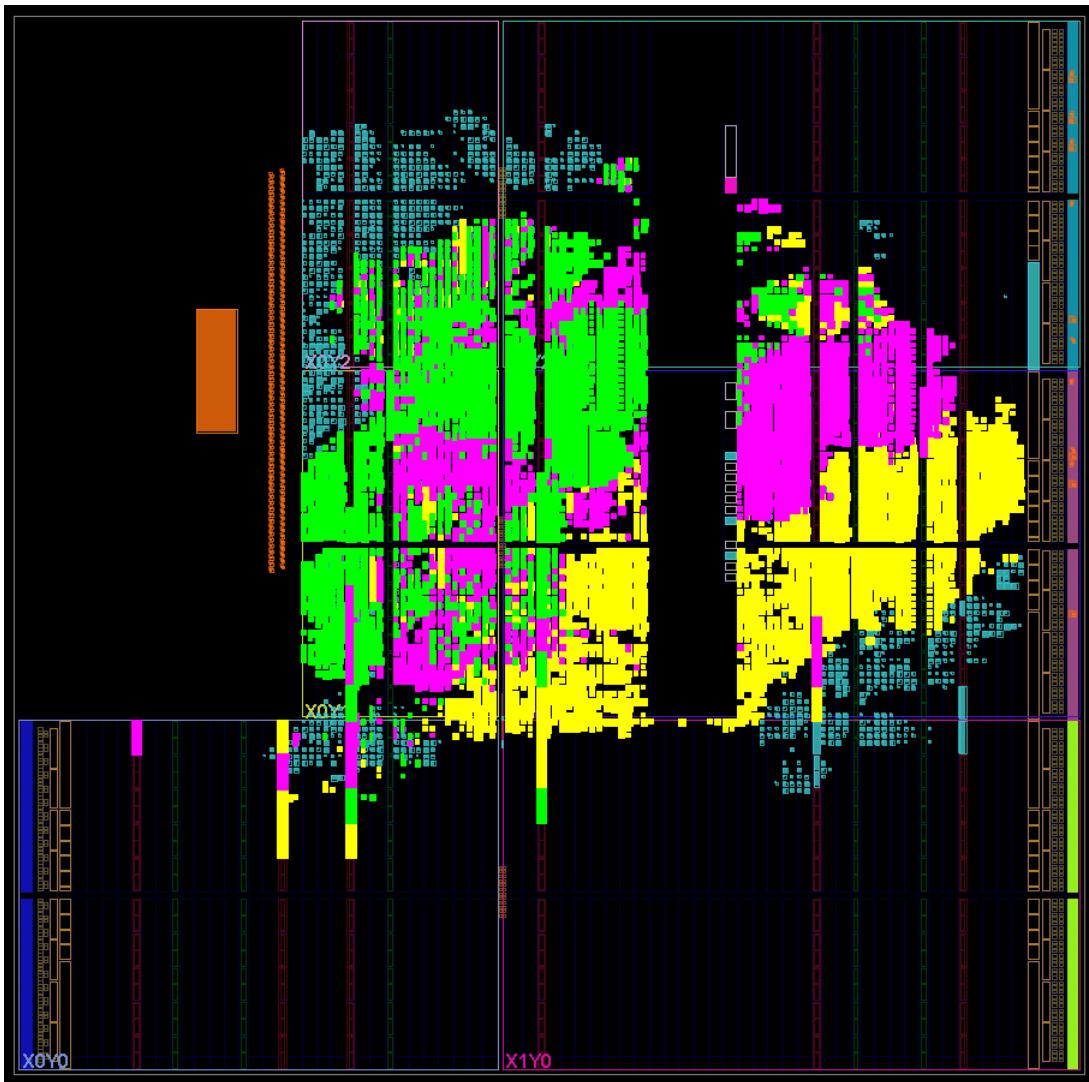


Fig. 3.18. Implementación en el dispositivo del diseño. En amarillo: BELs asociados a *cells* primitivas del sub-bloque MB1. En magenta: BELs asociados a *cells* primitivas del sub-bloque MB2. En verde: BELs asociados a *cells* primitivas del sub-bloque MB3. En celeste: BELs asociados al resto de *cells* primitivas.

El diseño final encaja con holgura en el dispositivo: ningún recurso supera el 30 % de utilización, y la mitad de ellos no llega al 15 %. El uso de los recursos del dispositivo se plasma en la figura 3.19. Con respecto al uso energético, el subsistema TMR requiere muy poca potencia. En la figura 3.20 se muestra que el responsable del mayor consumo energético es el PS del SoC, y no el PL. El Sistema de Procesamiento requiere el 79 % de la potencia total del chip.

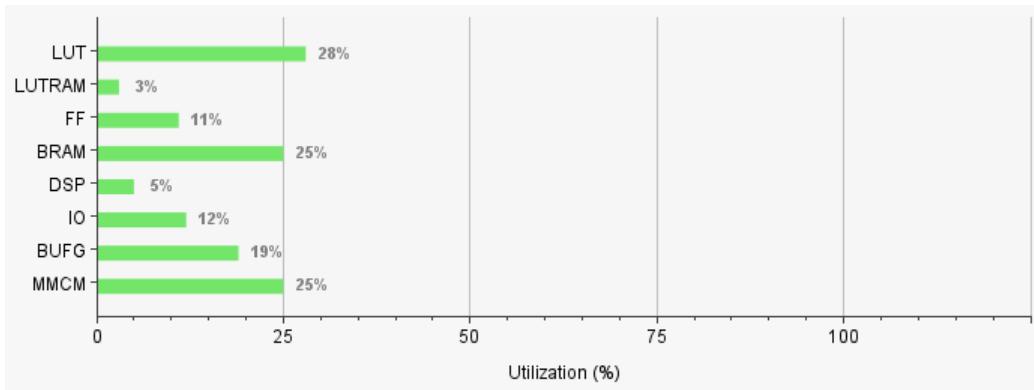


Fig. 3.19. Grado de utilización de cada recurso del dispositivo.

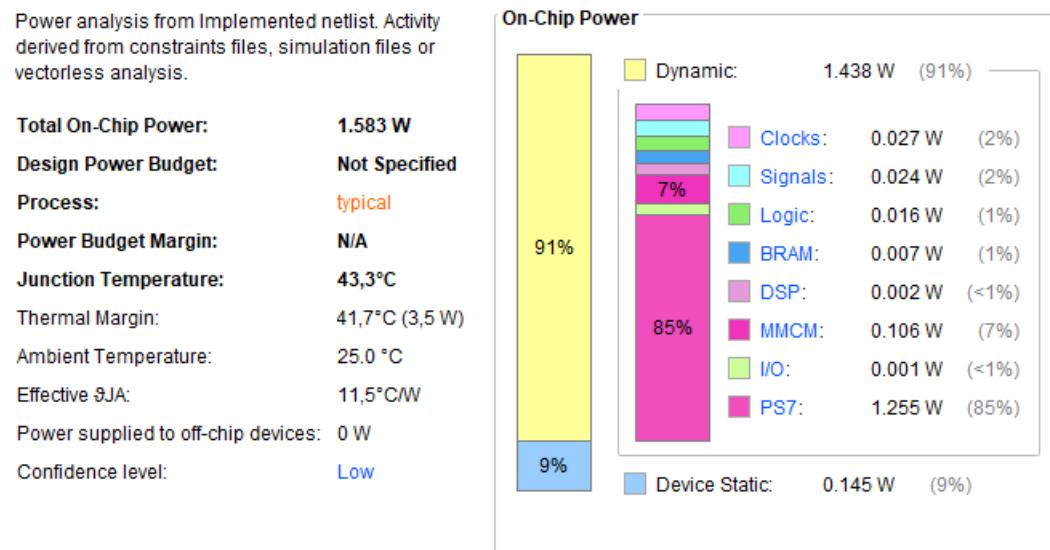


Fig. 3.20. Informe energético del diseño.

El paso sucesivo al *Timing Closure* y al análisis del diseño en la metodología AMD UltraFast es la generación del *bitstream*, la programación y el *debug*. El *bitstream* se generó desde Vivado y, tras ello, se exportó el *hardware* con el *bitstream* incluido mediante la opción *File > Export > Export Hardware*. El diseño se exporta en formato XSA (*Xilinx Support Archive* o *Xilinx Shell Archive*). Esto permite continuar con la programación y el *debug* en el entorno de desarrollo de *software* Vitis.

3.4. Entorno de desarrollo de software: Vitis

Vitis es una *suite* de herramientas de AMD para el desarrollo de aplicaciones embebidas en sus dispositivos [64]. De acuerdo con el flujo de trabajo para el desarrollo de aplicaciones de software embebidas (ver figura 3.21), antes de escribir *software* se han de crear los componentes relevantes de un proyecto Vitis. En este caso: plataforma y aplicaciones.

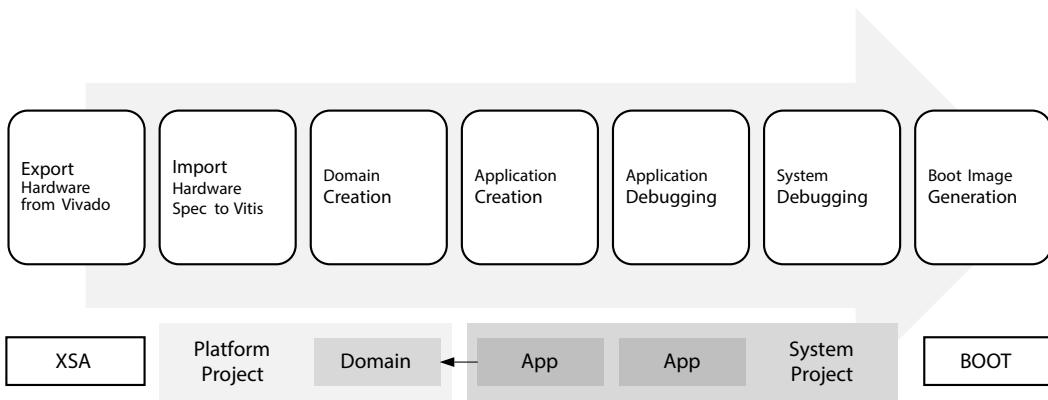


Fig. 3.21. Flujo de trabajo para el desarrollo de aplicaciones de software embebidas [65].

3.4.1. Creación de componentes: plataforma y aplicaciones

Las plataformas son componentes necesarios para la programación y uso del *hardware*. Estas se conforman a su vez de dos componentes: plataforma *hardware* y plataforma *software*. La primera de estas se construye con el archivo XSA exportado desde Vivado, y Vitis añade otros archivos relevantes. La plataforma *software* provee el entorno necesario para controlar el *hardware*: configuración del dominio, componentes de arranque, configuraciones de la plataforma *hardware*, librerías de compilación, *drivers* de los núcleos IP, y parámetros de los componentes mapeados en memoria, entre otros [64]. Para crear una plataforma, se ha de abrir el asistente de creación de componente de plataforma en *File >New Component >Platform*. En este caso, la plataforma se crea a partir de un diseño de *hardware*, por lo que se seleccionó dicha opción y se importó el archivo XSA. En la ventana *OS and Processor* se ha de seleccionar un dominio inicial. Un dominio delimita el entorno en el que se ejecutará la aplicación embebida, y debe especificar sistema operativo y procesador [66]. El diseño de este trabajo requiere dos dominios: uno para el PS y otro para el subsistema TMR. Como el asistente solo permite añadir un dominio inicial, se seleccionó uno de los núcleos del procesador Cortex-A9 (*ps7_cortexa9_0*), con sistema operativo *standalone* (la aplicación se ejecutará *barebone*, sin sistema operativo). Tras la creación de la plataforma, se agregó el segundo dominio seleccionando el MicroBlaze-V del sub-bloque MB1 (*tmr_system_MB1_microblaze_riscv_0*) con sistema operativo *standalone*. Para ejecutar la misma aplicación en los tres procesadores triplicados solo es necesario seleccionar uno de ellos (el del sub-bloque MB1 en este caso), pues todos tienen la misma configuración de *hardware* [46]. El ultimo paso que se realizó alrededor de la plataforma fue su construcción. Es importante resaltar que si la construcción de la plataforma deriva en errores no esperados, es posible que se esté trabajando en un directorio con una ruta demasiado extensa. Se recomienda por lo tanto crear el directorio de trabajo de Vitis en una ubicación lo menos anidada posible.

Atendiendo a la figura 3.21, tras la creación de los dominios se deben crear los componentes de aplicación. De forma análoga a la creación de la plataforma, se ha de utilizar el asistente de creación de aplicación, en *File >New Component >Application*. Hay que especificar la plataforma en la que se ejecutará la aplicación, así como su dominio [66]. Se creó por lo tanto una aplicación para ser ejecutada en el dominio del Cortex-A9, y otra para el subsistema TMR. Para escribir el código de las aplicaciones, se crearon los correspondientes archivos *main.c* en los directorios adecuados: <*app*>\Sources\src\<*main.c*>.

3.4.2. Aplicación del Cortex-A9

La aplicación del PS, cuyo código se halla en el Anexo E, ha de gestionar tres cuestiones:

- Configuración de la ruta ICAP** El núcleo SEM ha de poder acceder a la lógica de configuración del PL [24]. Para ello, y como se mostró en la figura 2.12, se ha de modificar la ruta de configuración de PCAP a ICAP. Dicha ruta se configura mediante un multiplexor controlado por el bit 27 (PCAP_PR) del registro de control de la configuración del dispositivo (DEVCFG_CTRL), con dirección 0xF8007000. Cuando el bit es 1, se establece la ruta PCAP y, cuando es 0, la ruta ICAP [50]. El cambio del valor del bit se realizó con el siguiente código:

```

1 #include <stdint.h>
2 #include "xil_io.h"
3
4 uint32_t reg_value;
5 reg_value = Xil_In32(0xF8007000); // Leer el registro DEVCFG_CTRL
6 reg_value &= ~((uint32_t)1 << 27); // Limpiar el bit PCAP_PR
7 Xil_Out32(0xF8007000, reg_value); // Escribir el valor de vuelta
al registro

```

- Transmisión de datos entre UART1 y UART2** El controlador UART1 recibe datos enviados por el subsistema TMR que se han de retransmitir al controlador UART0 —el cual tiene acceso al puerto USBUART— para ser enviados al ordenador externo. De forma similar, los datos enviados desde el ordenador externo que se reciben en UART0 se han de reenviar a UART1 para que los reciba el subsistema TMR. Este puente entre UART0 y UART1, como ya se ha explicado, permite utilizar un módulo PmodUSBUART menos. La implementación hace uso de las cabeceras *xuartps_hw.h* y *xuartps.h*, las cuales permiten utilizar las definiciones y declaraciones del *driver uartps*. Como se ve en el código del Anexo E, se inicializaron UART0 y UART1 con la *base address* correspondiente, los pines de las señales módem deshabilitados y un *baud rate* de 115200. En el bucle principal, se comprueba constantemente si alguno de los controladores ha recibido información y, si es el caso, se la reenvía al otro.

3. Monitorización del estado del subsistema TMR Las señales de estado se leen constantemente en el bucle principal y, si se detecta un cambio en alguna de ellas, se envía al ordenador externo por el periférico *axi_uart16550_status*. En el Anexo E se aprecia que se utilizó la cabecera *xgpio.h* para los *drivers* de los periféricos AXI GPIO. En el caso del AXI UART16550, se incluyó la cabecera *xuartns550.h*. Tanto los AXI GPIO como el AXI UART16550 se inicializaron con su *base address* correspondiente. Además, el UART se configuró en formato de datos 8N1 (8 bits, *No parity*, 1 *stop bit*), con *baud rate* de 115200 y sin señales módem.

La cabecera *xparameters.h* (incluida en el Anexo F) es un elemento de vital importancia para interactuar con los periféricos mapeados en memoria. Este contiene todas las definiciones de las direcciones de memoria de los periféricos, en especial la *base address*. Las definiciones de todos ellos se dan tanto en forma canónica como en no canónica. La forma no canónica utiliza el nombre en Vivado del bloque del periférico, con el *parent name* concatenado delante; esto es, el nombre del IP instanciado. En cambio, la definición canónica simplemente toma el nombre del IP con un índice basado en su orden de creación en caso de existir múltiples instancias del mismo IP. La ventaja de la definición canónica es que no cambia si se decide modificar el nombre de la instancia en Vivado. Sin embargo, hay que ser cauto con las modificaciones que se realizan por si estas alteran el índice. Debajo se muestran las definiciones del periférico *axi_gpio_status_1*. Se observa que la definición no canónica tiene el formato *XPAR_<NAME>_<PARENT_NAME>* (observar las propiedades en la figura 3.22), mientras que la canónica utiliza el nombre genérico del bloque seguido del índice 0, lo que indica que fue el primer bloque AXI_GPIO en añadirse en este espacio de direcciones.

```

1  /* Definitions for peripheral AXI_GPIO_STATUS_1 */
2  #define XPAR_AXI_GPIO_STATUS_1_COMPATIBLE "xlnx,axi-gpio-2.0"
3  #define XPAR_AXI_GPIO_STATUS_1_BASEADDR 0x41200000
4  #define XPAR_AXI_GPIO_STATUS_1_HIGHADDR 0x4120ffff
5  #define XPAR_AXI_GPIO_STATUS_1_INTERRUPT_PRESENT 0x0
6  #define XPAR_AXI_GPIO_STATUS_1_IS_DUAL 0x0
7  #define XPAR_AXI_GPIO_STATUS_1_GPIO_WIDTH 0x20
8
9  /* Canonical definitions for peripheral AXI_GPIO_STATUS_1 */
10 #define XPAR_XGPIO_0_BASEADDR 0x41200000
11 #define XPAR_XGPIO_0_HIGHADDR 0x4120ffff
12 #define XPAR_XGPIO_0_COMPATIBLE "xlnx,axi-gpio-2.0"
13 #define XPAR_XGPIO_0_GPIO_WIDTH 0x20
14 #define XPAR_XGPIO_0_INTERRUPT_PRESENT 0x0
15 #define XPAR_XGPIO_0_IS_DUAL 0x0

```

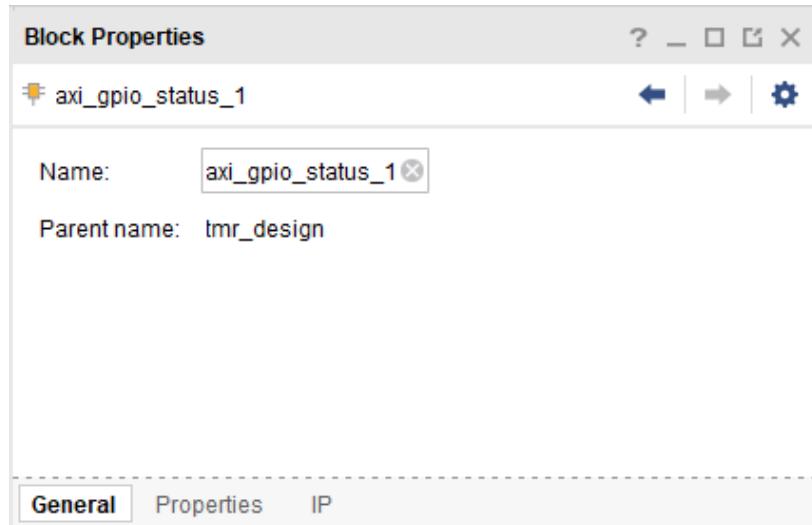


Fig. 3.22. Propiedades del bloque *axi_gpio_status_1* en el integrador de IPs de Vivado.

3.4.3. Aplicación del subsistema TMR

La aplicación del subsistema TMR, ejecutada simultáneamente en los tres MicroBlaze-V, y cuyo código se encuentra en el Anexo G, realiza tareas básicas relacionadas con la comunicación con el ordenador externo y el uso de periféricos. Se programaron dos modos (*mode0* y *mode1*), seleccionables mediante la interfaz UART del ordenador externo, que permiten elegir entre un modo de prueba de botones, interruptores y LEDs (*mode0*); y otro en el que los LEDs se accionan de forma secuencial automáticamente (*mode1*), utilizando el AXI TIMER para esperar el tiempo adecuado entre encendidos sin bloquear el bucle principal. El funcionamiento de ambos modos es el siguiente:

1. ***mode0*** Si se activan ambos interruptores, todos los LEDs se iluminan. Si se desactivan ambos interruptores, todos los LEDs se apagan. En caso de que haya un interruptor activado y otro desactivado, los LEDs permanecen apagados a menos que se mantenga pulsado su botón correspondiente (el colocado inmediatamente debajo de cada LED en la placa). Los botones solo iluminan los LEDs en este caso, en los anteriores no hacen nada. Se ha de recordar que el botón ubicado más a la derecha (BTN0, ver figura 2.8) no está mapeado en memoria, sino que proporciona la señal *reset* del sistema. Por lo tanto, pulsarlo no encenderá el LED LD0, sino que reiniciará el sistema.
2. ***mode1*** Se iluminan los LEDs de forma secuencial, en intervalos de medio segundo. La secuencia comienza en el LED ubicado más a la izquierda (LD3) y avanza hacia la derecha hasta llegar al LED LD0. Tras ello, se reinicia la secuencia.

El cambio entre modos se realiza enviando *mode0* o *mode1* por UART desde el ordenador externo y se puede realizar en cualquier momento. Tras un cambio de modo exitoso, el subsistema TMR envía un mensaje de confirmación al ordenador externo mediante UART. Si el cambio no se puede producir por encontrarse ya en ese modo, o si se recibe un comando inesperado, se envía la notificación adecuada al ordenador externo. La cabecera *xmrctr.h* se incluyó para hacer uso de los *drivers* del AXI TIMER. El archivo *xparameters.h* utilizado para obtener la *base address* de los periféricos se ha incluido en el Anexo H.

3.4.4. Construcción, configuración y *debugging*

Una vez escrito el código de las aplicaciones y antes de poder hacer *debugging*, se han de construir y establecer la configuración de lanzamiento correcta de cada una de ellas. Para construirlas, tan solo hay que seleccionar la opción correspondiente en el navegador de flujo. En cuanto a la configuración de lanzamiento, hay que implantar una distinta para cada *app*.

1. ***App del Cortex-A9*** La figura 3.23 muestra las opciones que se han de marcar en la configuración de esta aplicación. Hay que reiniciar todo el sistema, programar el PL con el *bitstream*, e inicializar el PS y dejarlo activo. Antes de descargar la aplicación en el núcleo Cortex-A9, este se reinicia y queda suspendido hasta que la recibe. La aplicación se descarga al procesador en formato ELF (*Executable and Linkable Format*). Este es un formato portable que contiene, entre otras cuestiones, cabeceras que indican información sobre el propio formato y la ISA de la máquina de destino, o información sobre el programa como su tamaño y la dirección de memoria en la que se ha de cargar. También contiene secciones con el código ejecutable y con los datos utilizados por este. Tras lanzar la aplicación del Cortex-A9, se ha de lanzar la del subsistema TMR sin parar la ejecución de la primera.

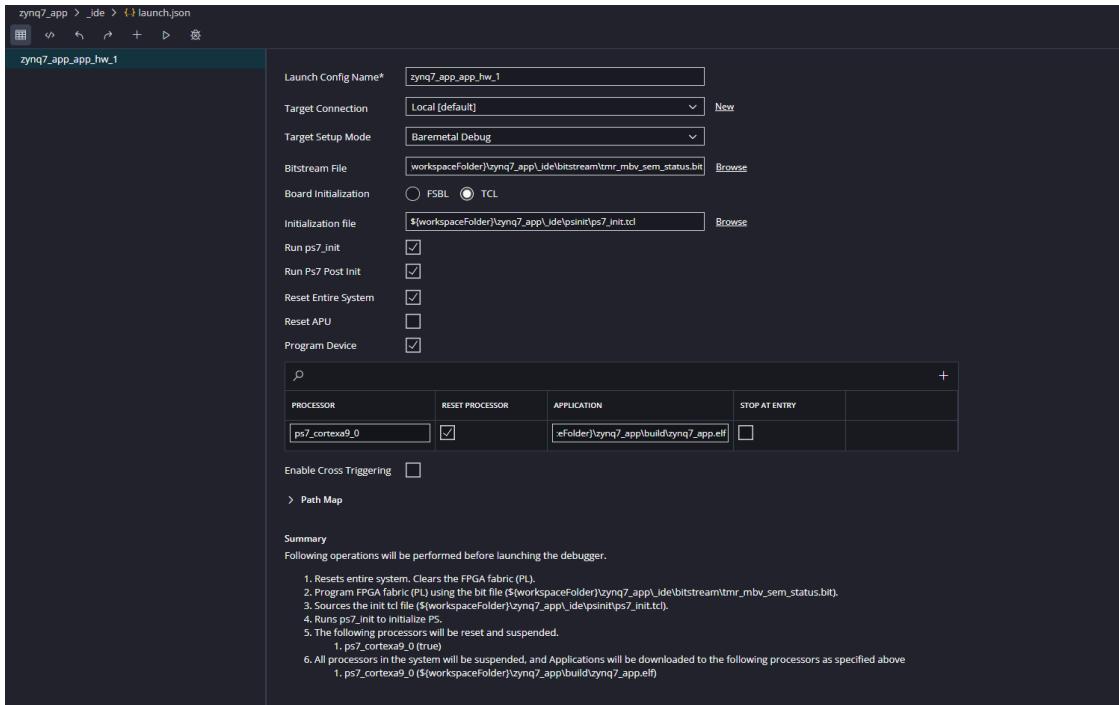


Fig. 3.23. Configuración de lanzamiento de la aplicación del Cortex-A9

2. App del subsistema TMR Como esta aplicación ha de lanzarse con posterioridad a la del Cortex-A9, y esta ya ha programado el PL e inicializado el PS, se han de desactivar todas las opciones relativas a ello. Además, se debe deseleccionar la opción del reinicio del procesador, ya que, aunque el programa se ejecute en los tres procesadores MicroBlaze-V, tan solo se reiniciaría el del sub-bloque MB1. Ello derivaría en un desajuste entre ese sub-bloque y el resto, y el subsistema TMR pasaría a modo *lockstep*. La configuración correcta se muestra en la figura 3.24.

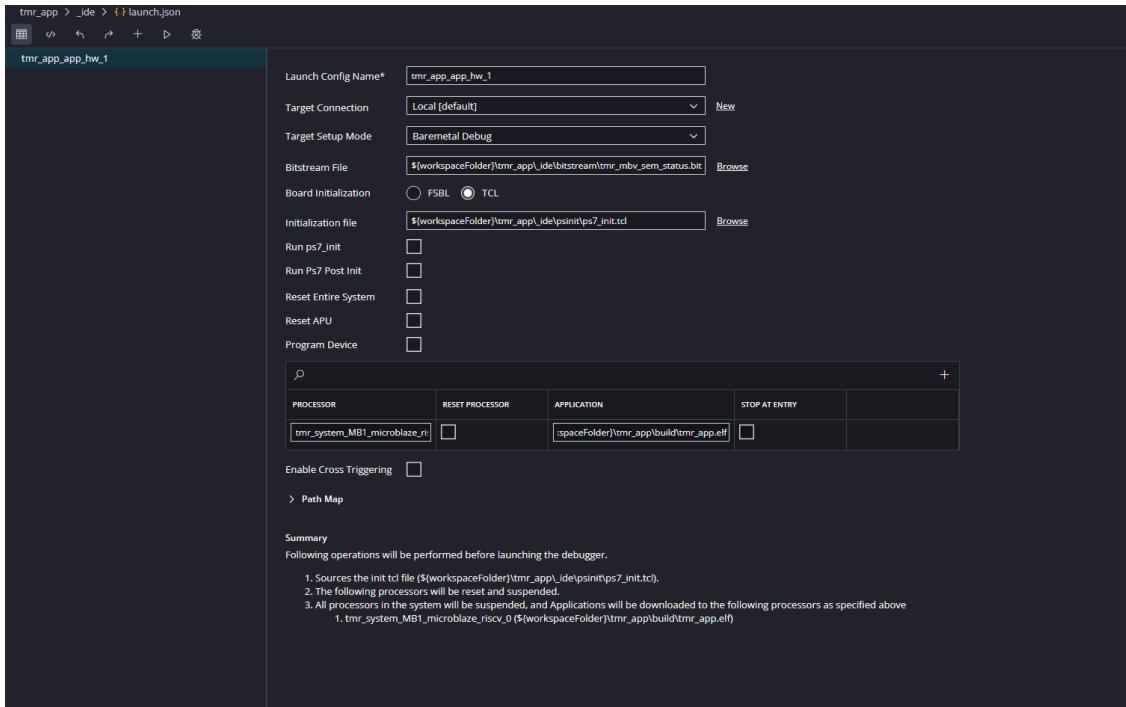


Fig. 3.24. Configuración de lanzamiento de la aplicación del subsistema TMR

Una opción más conveniente es, sin embargo, la creación de una aplicación que lance ambos programas en el orden adecuado automáticamente. Para ello, se modificó la configuración de lanzamiento de la figura 3.23 añadiendo el archivo ELF de la aplicación del subsistema TMR. Esta nueva configuración se muestra en la figura 3.25. La aplicación del Cortex-A9 se lanza primero para realizar el cambio a ICAP cuanto antes. Además, es necesaria para recibir en el ordenador externo los mensajes enviados por UART desde el subsistema TMR. Si el programa de este último se lanzase antes, se perderían sus mensajes de inicialización.

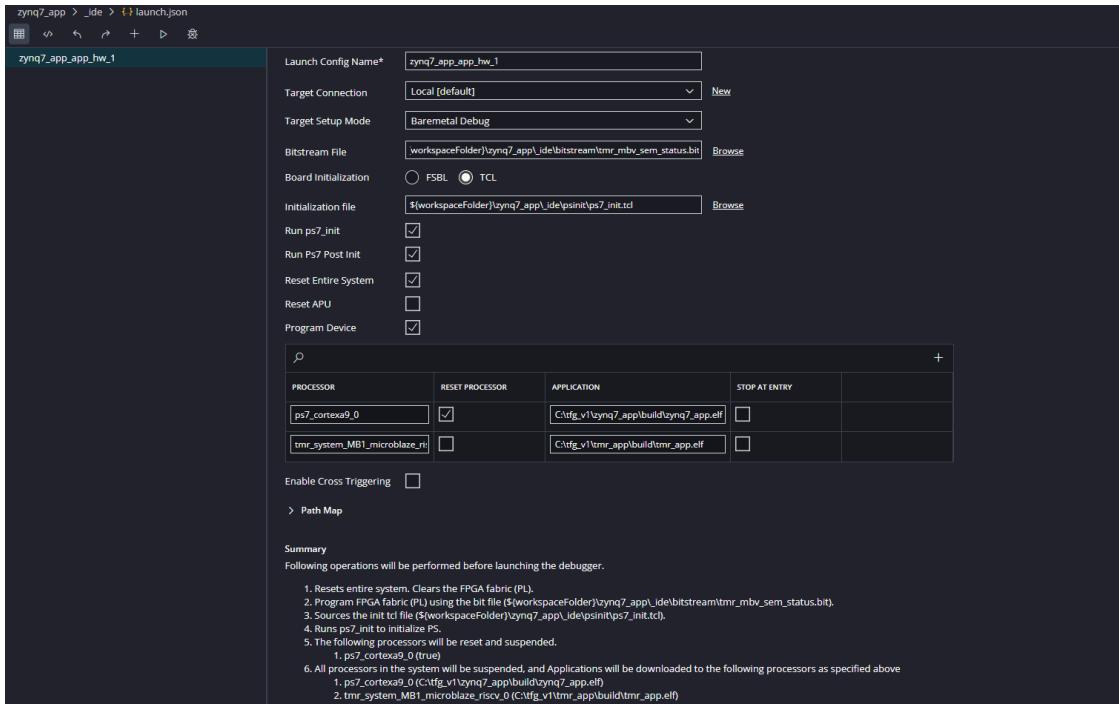


Fig. 3.25. Configuración de lanzamiento de la aplicación conjunta.

Tras establecer las configuraciones correctas, se puede lanzar la aplicación para hacer *debugging* tanto línea a línea como sin pausar el flujo de ejecución.

3.5. Metodología de pruebas

En esta sección se tratará la realización de las pruebas del dispositivo. Estas pruebas tienen como objeto determinar la correcta función de la triple redundancia modular y del controlador SEM.

3.5.1. Simulación de fallos

En un entorno espacial real, el dispositivo sería susceptible de sufrir SEUs tanto en la memoria de configuración como en la de diseño. No obstante, la simulación de errores en este trabajo se acotó a la memoria de configuración. Para plantear la metodología de inyección de errores hay que conocer el funcionamiento de la interfaz de monitorización del IP TMR SEM.

Interfaz de monitorización SEM

El primer contacto con la interfaz de monitorización llegará de la mano del informe de inicialización. Este es un mensaje informativo que se envía al inicializarse el IP tras

programar el PL con el diseño *hardware*. En este caso, se recibe a través de uno de los módulos PmodUSBUART:

```

1 X7_SEM_V4_1 # Nombre del dispositivo y version del IP
2 SC 01 State # El controlador entra en estado de inicializacion
3 FS {2-digit hex value} # Informacion sobre la configuracion de
   caracteristicas del IP
4 ICAP OK # ICAP disponible
5 RDBK OK # Readback disponible
6 INIT OK # Inicializacion completada
7 SC 02 # El controlador entra en estado de observacion

```

El informe se queda parado tras la palabra «ICAP» hasta que se realiza el cambio de PCAP a ICAP. Cuando este se lleva a cabo, el informe se completa [24]. En este punto, ya se pueden enviar comandos por la interfaz de monitorización. Los comandos disponibles son los siguientes:

1. **Directed State Changes** Los comandos de cambio directo de estado modifican el estado del controlador pasando de *Idle* a *Observation* y viceversa. En *Idle*, el controlador no detecta ningún fallo en la memoria de configuración y permite realizar las siguientes acciones mediante el uso de comandos: cambiar al estado *Observation*, generar un *Status report* e inyectar un número arbitrario de errores de un bit. En el estado de *Observation*, el controlador realiza *scrubbing* a la memoria de configuración en busca de errores. Permite pasar al estado *Idle* y generar un *Status Report*, pero no inyectar errores. Para la inyección de múltiples errores simultáneos, se deben enviar individualmente desde el estado *Idle*. Al realizar la transición al estado de observación, el controlador se encontrará con todos los errores de forma simultánea, como si se hubiesen producido en el mismo instante [24].

```

1 I # Pasa a "Idle" desde "Observation"
2 O # Pasa a "Observation" desde "Idle"

```

2. **Status report** Genera un informe del estado del controlador [24].

```

1 S # Desde "Idle" u "Observation", genera un informe de estado

```

3. **Error Injection** Realiza una lectura-modificación-escritura para invertir el bit especificado de la memoria de configuración. Los detalles de la inyección se proporcionan en un valor hexadecimal de 10 dígitos, lo que se corresponde con un valor binario de 40 bits. Dependiendo del formato de dicho valor, se utilizan dos esquemas de direccionamiento diferentes [24]:

- **Linear Frame Address (LFA)** El valor binario tiene la estructura indicada en la tabla 3.4 [24].

3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	1	0	9	8	7	6	5	4	3	2	1		
9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0				
1	1	0	0	0	0	0	0	0	0	S	S	L	L	L	L	L	L	L	L	L	L	L	L	L	W	W	W	W	W	B	B	B	B

TABLA 3.4. VALOR DEL COMANDO DE INYECCIÓN DE ERRORES EN ESQUEMA DE DIRECCIONAMIENTO LFA

Donde:

- **SS** En dispositivos SSI: índice del componente SLR (2 bits). En dispositivos no SSI: 00.
 - **LLLLLLLLLLLLLLL** *Linear Frame Address* (17 bits, [0..0xMF]).
 - **WWWWWWWW** *Word Address* (7 bits, [0..100]).
 - **BBBBBB** *Bit Address* (5 bits, [0..31]).
- **Physical Frame Address (PFA)** El valor binario tiene la estructura indicada en la tabla 3.5 [24].

3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	1	0	9	8	7	6	5	4	3	2	1	0		
9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0				
0	S	S	T	T	H	R	R	R	R	C	C	C	C	C	C	C	C	C	C	M	M	M	M	M	M	W	W	W	W	B	B	B	B

TABLA 3.5. VALOR DEL COMANDO DE INYECCIÓN DE ERRORES EN ESQUEMA DE DIRECCIONAMIENTO PFA

Donde:

- **SS** En dispositivos SSI: índice del componente SLR (2 bits). En dispositivos no SSI: 00.
- **TT** *Block type* (2 bits).
- **H** *Half Address* (1 bit).
- **RRRRR** *Row Address* (5 bits).
- **CCCCCCCC** *Column Address* (10 bits).
- **MMMMMM** *Minor Address* (7 bits).
- **WWWWWWWW** *Word Address* (7 bits, [0..100]).
- **BBBBBB** *Bit Address* (5 bits, [0..31]).

En ambos tipos de direccionamiento, los bits [39..12] seleccionan el *frame*. La *Word Address* selecciona la palabra dentro del *frame*, y la *Bit Address*, el bit dentro de la palabra. Más adelante se desarrollará la cuestión de los *frames* y su direccionamiento físico. El direccionamiento LFA mapea todos los *frames* relevantes a un espacio linear continuo. De esta forma, si se inyecta un error en cualquier bit perteneciente al rango de direcciones lineales, se está seguro de que dicho bit pertenece a un *frame* que configura el diseño. No obstante, las direcciones no proveen información de la localización física y tipo de *frame*. En cambio, el direccionamiento PFA no agrupa los *frames* relevantes, sino que estos se encuentran en sus direcciones físicas reales [24].

```
| N {10-digit hex value} # Realiza la inyección especificada por el  
| valor hexadecimal
```

4. **Software reset** Pasa del estado *Idle* a *Observation* a través del estado de inicialización, generando de nuevo un informe de inicialización [24].

```
| R {2-digit hex value} # Los 2 dígitos hexadecimales son valores  
| "I don't care".
```

En el proceso de monitorización, el controlador enviará mensajes a la interfaz. Los posibles mensajes son los siguientes [24]:

1. **Command Prompt** Si el controlador está en estado *Observation*, se muestra *O>* en la consola. Si el controlador está en estado *Idle*, se muestra *I>* en la consola [24].
2. **State change report** El informe de cambio de estado se genera cada vez que cambia el estado del controlador. Se ha de tener en cuenta que se puede pasar al estado *Fatal Error* sin recibir el informe de cambio de estado correspondiente. En cualquier caso, se mostrará el mensaje de error fatal *HLT* [24].

```
| SC {2-digit hex value} # Indica que se ha pasado al estado  
| especificado por el valor hexadecimal
```

Report String	State Name
SC 00	Idle
SC 01	Initialization
SC 02	Observation
SC 04	Correction
SC 08	Classification
SC 10	Injection
SC 1F	Fatal error

TABLA 3.6. POSIBLES MENSAJES DE INFORME DE CAMBIO DE ESTADO

3. **Flag change report** La *Flag* indicada identifica el tipo de error detectado [24].

```
| FC {2-digit hex value} # Indica que se ha detectado el tipo de  
| error especificado por el valor hexadecimal
```

Report String	Condition Name
FC 00	Correctable, Non-Essential
FC 20	Uncorrectable, Non-Essential
FC 40	Correctable, Essential
FC 60	Uncorrectable, Essential

TABLA 3.7. POSIBLES MENSAJES DE INFORME DE CAMBIO DE *FLAG*

4. **Status report** Da información adicional sobre el estado del controlador. Se genera con el comando *S* desde los estados *Idle* u *Observation*. En dispositivos no SSI, como el ZYNQ XC7Z020-1CLG400C, se crea un informe sencillo. Sin embargo, con dispositivos SSI el informe es compuesto, con subinformes para cada SLR, ordenados por el índice (número de *hardware*) SLR [24].

```

1 MF {8-digit hex value} # Frame maximo (numero de frames lineales)
2 SN {2-digit hex value} # Numero de hardware SLR
3 SC {2-digit hex value} # Estado actual
4 FC {2-digit hex value} # Flags actuales
5 FS {2-digit hex value} # Configuracion de caracteristicas del IP

```

5. **Error Detection Report** El informe de detección de errores se genera generalmente tras la corrección del error (si esta es posible), ya que el controlador intenta corregirlo lo antes posible. Se pueden dar los siguientes casos [24]:

- **Error CRC** En este caso no se puede identificar la ubicación o número de bits del error.

```

1 SC 04 # Se pasa al estado de correccion de errores
2 CRC # Error CRC

```

- **Error BFR** Error incorregible del búfer del *checksum* de reparación mejorada.

```

1 SC 04 # Se pasa al estado de correccion de errores
2 BFR # Error BFR

```

- **Error ECC de 1 bit, síndrome ECC válido** El síndrome ECC se utiliza para identificar la posición exacta del error en el *frame*. El controlador muestra pues su dirección.

```

1 SC 04 # Se pasa al estado de correccion de errores
2 SED OK # Sindrome ECC valido (Single Error Detection OK)
3 PA {8-digit hex value} # Physical Address del frame
4 LA {8-digit hex value} # Linear Address del frame

```

```
5 | WD {2-digit hex value} BT {2-digit hex value} # Word  
address y Byte Address
```

- **Error ECC de 1 bit, síndrome ECC inválido** El síndrome ECC apunta a una dirección de palabra fuera de rango. El controlador muestra la dirección del *frame*, pero no la posición exacta del bit. Tras este error se ha de reconfigurar la FPGA.

```
1 | SC 04 # Se pasa al estado de correccion de errores  
2 | SED NG # Sindrome ECC invalido (Single Error Detection No  
Good)  
3 | PA {8-digit hex value} # Physical Address del frame  
4 | LA {8-digit hex value} # Linear Address del frame
```

- **Error ECC de 2 bits** El controlador muestra la dirección del *frame*.

```
1 | SC 04 # Se pasa al estado de correccion de errores  
2 | DED # Double Error Detection  
3 | PA {8-digit hex value} # Physical Address del frame  
4 | LA {8-digit hex value} # Linear Address del frame
```

6. **Error Correction Report** El informe de corrección de errores depende de la configuración del IP y del resultado de la corrección [24]:

- **Corrección desactivada o error no corregible**

```
1 | COR # Comienzo del listado de errores corregidos  
2 | END # Fin del listado de errores corregidos
```

Seguido por:

```
1 | FC 20 # No corregible, no esencial
```

O

```
1 | FC 60 # No corregible, esencial
```

- **Error corregible**

```
1 | COR # Comienzo del listado de errores corregidos  
2 | {correction list} # Lista de errores corregidos  
3 | END # Fin del listado de errores corregidos
```

Seguido por:

```
1 | FC 00 # Corregible, no esencial
```

O

```
1 | FC 40 # Corregible, esencial
```

Donde *correction list* se conforma de una o varias líneas (una línea por bit corregido) que indican la posición en el *frame* de cada bit corregido:

```
1 | WD {2-digit hex value} BT {2-digit hex value} # Word  
      address y Byte Address
```

7. **Error Classification Report** La clasificación de errores analiza cada error en el *frame* para determinar si alguno de ellos es esencial, en cuyo caso todo el evento se considera esencial. El informe de clasificación de errores depende de la configuración del IP y del resultado de la corrección [24]:

- **Clasificación desactivada y evento corregible** Con la clasificación de errores desactivada, todos se catalogan como esenciales, puesto que el controlador no puede diferenciarlos. El IP TMR SEM tiene la clasificación de errores desactivada [46], por lo que en este caso el informe de clasificación de errores determinará que todos los eventos son esenciales. Si se quiere activar la clasificación de errores se ha de utilizar un controlador externo. No obstante, a pesar de que el informe de clasificación muestre todos los eventos como esenciales, el informe de corrección sí que determina el tipo de error correctamente como consecuencia del cambio de *flag* que alguno de los errores haya producido. No obstante, ese cambio de *flag* es a nivel de evento y por lo tanto no clasifica bit a bit, como haría el informe de clasificación de errores.

```
1 | SC 08 # Se pasa al estado de clasificacion de errores  
2 | FC 40 # Corregible, esencial
```

- **Error no corregible** Todos los errores no corregibles se catalogan como esenciales, independientemente de la configuración de clasificación del controlador.

```
1 | SC 08 # Se pasa al estado de clasificacion de errores  
2 | FC 60 # No corregible, esencial
```

■ Clasificación activada y evento corregible y no esencial

```
1 SC 08      # Se pasa al estado de clasificacion de errores
2 CLA        # Comienzo del listado de bits esenciales
3 END        # Fin del listado de bits esenciales
4 FC 00      # Corregible, no esencial
```

■ Clasificación activada y evento corregible y esencial

```
1 SC 08      # Se pasa al estado de clasificacion de errores
2 CLA        # Comienzo del listado de bits esenciales
3 {classification list}
4 END        # Fin del listado de bits esenciales
5 FC 40      # Corregible, no esencial
```

Donde *classification list* se conforma de una o varias líneas (una línea por bit esencial) que indican la posición en el *frame* de cada bit esencial:

```
1 WD {2-digit hex value} BT {2-digit hex value}    # Word
      address y Byte Address
```

Con respecto a los informes de detección, corrección y clasificación, estos se remiten en ese orden, según el controlador va pasando por los diferentes estados. En el caso de que un error no se pueda corregir, el controlador permanecerá en estado *Idle* de forma perpetua hasta que se reconfigure la FPGA [24]. Así, se podrá seguir inyectando errores, pero no volver al estado de observación hasta que se realice la reconfiguración.

Interfaz de monitorización TMR

La interfaz de monitorización TMR que se ha implementado es unidireccional: tan solo se reciben en ella los cambios en el estado del subsistema TMR, pero no permite enviar comandos. Los cambios en el estado se reciben de forma independiente desde los TMR *Manager* de cada uno de los sub-bloques, por lo que se muestran tres estados. Estos estados coinciden entre sí, a menos que el error haya afectado a la lógica del estado del algún sub-bloque o se produzca otro tipo de imprevisto. El informe de cambio de estado del subsistema TMR tiene la siguiente forma:

```
1 TMR Status 1 changed: {32-digit binary value}
2 TMR Status 2 changed: {32-digit binary value}
3 TMR Status 3 changed: {32-digit binary value}
```

Donde el valor binario de 32 dígitos se interpreta según la tabla 3.8 [46].

Bit	Campo	Descripción
0	Lockstep Mismatch	Desajuste entre los procesadores 1 y 2. Se pasa al estado de <i>Lockstep</i>
1		Desajuste entre los procesadores 1 y 3. Se pasa al estado de <i>Lockstep</i>
2		Desajuste entre los procesadores 2 y 3. Se pasa al estado de <i>Lockstep</i>
3	Recovery	Recuperación
4	Fatal Errors	Error fatal entre los procesadores 1 y 2. Se pasa al estado <i>Fatal</i>
5		Error fatal entre los procesadores 1 y 3. Se pasa al estado <i>Fatal</i>
6		Error fatal entre los procesadores 2 y 3. Se pasa al estado <i>Fatal</i>
7		Error fatal del votante
8		Error fatal incorregible ECC
9		<i>Watchdog</i> expirado
11–10	FT State	00 = Estado nominal 01 = <i>Lockstep</i> 10 = Reinicio de recuperación 11 = Error fatal
12	Processor Faults	Fallo en el procesador 1, válido cuando el campo <i>Lockstep Mismatch</i> no es cero
13		Fallo en el procesador 2, válido cuando el campo <i>Lockstep Mismatch</i> no es cero
14		Fallo en el procesador 3, válido cuando el campo <i>Lockstep Mismatch</i> no es cero
31–15	Reserved	Reservado

TABLA 3.8. SEÑAL DE ESTADO DEL SUBSISTEMA TMR

Así, un error en el sub-bloque MB3 produciría la siguiente salida en la interfaz de comunicación serial del estado, que indica el desajuste de MB3 con MB1 y MB2, la transición al estado de *Lockstep*, y el fallo en MB3:

```

1 TMR Status 1 changed: 0b00000000000000000000100010000000110
2 TMR Status 2 changed: 0b00000000000000000000100010000000110
3 TMR Status 3 changed: 0b00000000000000000000100010000000110

```

Inyección a elementos concretos de la *netlist*

Tras conocer el funcionamiento de las interfaces de monitorización, en concreto de la SEM y cómo esta se puede utilizar para provocar errores, surge la cuestión de la inyección localizada e informada de los mismos. Una inyección de errores que garantice afectar a *cells* primitivas de la *netlist* implementada es esencial para comprobar la funcionalidad del sistema TMR. Sin embargo, las herramientas de AMD no ofrecen ninguna funcionalidad con la que poder identificar qué bits configuran cada BEL. Lo más cercano es el archivo *logic location* (.ll), el cual se genera junto al *bitstream* si así se configura en *Flow >Settings >Bitstream Settings*. No obstante, proporciona una cantidad de información limitada. Por ello, se optó por utilizar *Project X-Ray*, un proyecto de código abierto que tiene el objetivo de documentar el formato del *bitstream* de las FPGAs de la serie 7 de Xilinx [62].

Project X-Ray

Project X-Ray ofrece una extensa base de datos con información técnica sobre las FPGAs de la serie 7 de Xilinx, pues su objetivo es fomentar el desarrollo de herramientas gratuitas y de código abierto para convertir código HDL a *bitstreams* de estos dispositivos. La base de datos que tiene publicada se construyó por un proceso de ingeniería inversa, generando multitud de diseños en Vivado y realizando comparaciones cruzadas para determinar el propósito de cada bit. Las herramientas para llevar a cabo este proceso se encuentran en el repositorio del proyecto, pero no son necesarias para el objeto de este trabajo, pues la base de datos ofrece toda la información relevante [62].

Antes de tratar la base de datos, hay que comprender en profundidad la organización de la memoria de configuración. Como se ha mencionado en múltiples ocasiones a lo largo del documento, los *frames* son la unidad fundamental de la memoria de configuración: su segmento direccionable más pequeño. Están compuestos de 101 palabras de 32 bits, siendo la 50^a palabra de cada *frame* su ECC [62]. El formato de las direcciones de los *frames*, de 32 bits, ya se expuso en el comando de inyección PFA. Esto es así porque, como su nombre indica, la inyección en esquema de direccionamiento de *Physical Frame Address* utiliza la dirección física del *frame*. El formato es el siguiente:

- **Reservado [31:26] (6 bits)**
- **Block type [25:23] (3 bits)** Define el tipo de bloque (bus de configuración) al que se conectan las *Tiles* configuradas por el *frame*. Los posibles bloques son: CLB, IO, CLK —al que se conectan *Tiles Interconnect* (INT)— con valor 000; block RAM content —al que se conectan *Tiles BRAM*— con valor 001; y CFG_CLB —el cual, a pesar de ser mencionado en la documentación del dispositivo, no se ha encontrado en ningún *bitstream* [62]— con valor 010 [50].

- **Half Address [22] (1 bit)** Selecciona entre la mitad superior (valor 0) o inferior (valor 1) del dispositivo. Cada mitad tiene un BUFG (*Global Clock Buffer*) diferente [50].
- **Row Address [21:17] (5 bits)** Selecciona la fila dentro de la mitad. Su numeración es creciente en el sentido que se aleja de la línea que separa las mitades. En una fila, las *Tiles* se conectan a uno o más buses de configuración dependiendo de su tipo.
- **Column Address [16:7] (10 bits)** Selecciona una columna. Su numeración aumenta de izquierda a derecha. En cada bus de configuración, las *Tiles* se agrupan en columnas. Los datos de configuración de una columna son los *frames*. La columna coincide con la *X* del nombre de la *Tile*, pero no ocurre lo mismo con la fila y la *Y* [62].
- **Minor Address [6:0] (7 bits)** Identifica un *frame* específico de la columna. En la documentación de *Project X-Ray*, también se conoce como *frame index*. Estableciendo los bits de la *Minor Address* a cero, se obtiene la *Base Address* de un *frame*. La *Base Address* es la dirección de la columna en la que se agrupa la *Tile*, y es importante porque los bits que configuran la *Tile* se distribuyen entre múltiples *frames* de la columna [62].

Se ha incluido en el Anexo I un diagrama que explica de forma visual la estructura de los *frames*. Los *frames* del diagrama se obtuvieron mediante herramientas de utilidad de *Project X-Ray*. Con el script *bit2fasm.py* se convirtió el *bitstream* del diseño a formato FASM (*FPGA Assembly*), un formato abierto de configuración de FPGAs legible por humanos [67]. Tras ello, se utilizó el script *fasm2frames.py* para obtener los *frames* del *bitstream* original.

Para relacionar una *Tile* con los *frames* que la configuran, se utilizó el archivo *tilegrid* (*database/zynq7/xc7z020/tilegrid.json*) de la base de datos. Este archivo en concreto proporciona información de las *Tiles* de los dispositivos de la familia XC7Z020 (se recuerda que la PYNQ-Z1 tiene un SoC ZYNQ XC7Z020-1CLG400C). Existe un archivo *tilegrid* para cada familia de dispositivos (no varía el formato del archivo, pero sí su contenido). Está en formato JSON, y cada entrada corresponde a una *Tile* del dispositivo. A continuación se muestra el formato de las entradas [62]:

```

1  {
2      "<TITLE_NAME>": {
3          "bits": {
4              "<CONFIGURATION_BUS>": {
5                  "baseaddr": "<BASE_ADDR>" ,
6                  "frames": "<FRAME_COUNT>" ,
7                  "offset": "<TITLE_OFFSET>" ,
8                  "words": "<WORDS_COUNT>" ,
9                  } ,
10                 "<...>" ,
11             } ,
12             "clock_region": "<CLOCK_REGION>" ,
13             "grid_x": "<GRID_X>" ,
14             "grid_y": "<GRID_Y>" ,
15             "pin_functions": {
16                 "<PIN_NAME>": "<PIN_FUNCTION>" ,
17                 "<...>" ,
18             } ,
19             "prohibited_sites": [
20                 "<SITE_TYPE>" ,
21                 "<...>" ,
22             ] ,
23             "sites": {
24                 "<SITE_NAME>": "<SITE_TYPE>" ,
25                 "<...>" ,
26             } ,
27             "type": "<TITLE_TYPE>" ,
28         }

```

Donde los campos tienen el siguiente significado:

- ***bits*** Diccionario con los diferentes *Configuration Buses* que configuran la *Tile*.
 - **<CONFIGURATION_BUS>** Diccionario que agrupa datos sobre la configuración de la *Tile* a través del bus de configuración *<CONFIGURATION_BUS>*.
 - **baseaddr** *Base address* del *frame*. En base hexadecimal.
 - **frames** Número de *frames* que configuran la *Tile*. Todos esos *frames* pertenecen a la misma columna. En base decimal.
 - **offset** Número de palabras que hay en el *frame* delante de la primera que configura la *Tile*. Se denomina *Tile offset*. En base decimal.
 - **words** Número de palabras de cada *frame* que configuran la *Tile*. En base decimal.
- ***clock_region*** Región de reloj a la que pertenece la *Tile*.
- ***grid_x*** Columna de la *Tile*. No confundir con la *Column Address*. Su numeración aumenta de izquierda a derecha.

- ***grid_y*** Fila de la *Tile*. No confundir con la *Row Address*. Su numeración aumenta de arriba abajo.
- ***pin_functions*** Diccionario que indica funciones especiales de los pines de la *Tile*. Este campo lo utilizan principalmente los bloques IOB (*I/O Blocks*).
- ***prohibited_sites*** Lista los *Sites* que no se pueden utilizar con la *Tile*.
- ***sites*** Diccionario con los *Sites* que conforman la *Tile*.
 - <*SITE_NAME*> Indica el tipo de *Site* (<*SITE_TYPE*>) al que pertenece <*SITE_NAME*>.
- ***type*** Especifica el tipo de *Tile*.

La entrada del *tilegrid* correspondiente a la *Tile* configurada por los *frames* del Anexo I es la siguiente:

```

1  "DSP_R_X25Y95": {
2    "bits": {
3      "CLB_IO_CLK": {
4        "baseaddr": "0x00400C80",
5        "frames": 28,
6        "offset": 91,
7        "words": 10
8      }
9    },
10   "clock_region": "X0Y1",
11   "grid_x": 68,
12   "grid_y": 57,
13   "pin_functions": {},
14   "prohibited_sites": [],
15   "sites": {
16     "DSP48_X2Y38": "DSP48E1",
17     "DSP48_X2Y39": "DSP48E1",
18     "TIEOFF_X28Y95": "TIEOFF"
19   },
20   "type": "DSP_R"
21 }
```

Los datos proporcionados por el archivo *tilegrid* identifican el rango de *frames* que configuran una *Tile*, pero la inyección de errores requiere de un bit específico en un *frame* determinado. Para hallar los bits concretos que configuran un BEL, se necesita también el archivo *segbits* del tipo de *Tile* correspondiente. Los archivos *segbits* son específicos de cada tipo de *Tile* y contienen la información sobre las combinaciones de bits que activan cualquiera de sus características. El nombre de estos archivos sigue el formato *segbits_<tile_type>.db* para *Tiles* configuradas por el bus predeterminado CLB_IO_CLK. Si la *Tile* es además configurable por otros buses, existen archivos *segfiles* adicionales

para esos casos, con formato `segbits_<tile_type>.<configuration_bus>.db`. El archivo está formado por tantas líneas como características configurables tenga la *Tile*. Las líneas tienen la siguiente estructura [62]:

```
1 | <feature> <bit_list>
```

Donde:

- **<feature>** Tiene la forma `<feature_name>.<feature_address>`
- **<bit_list>** Lista los bits que se han de configurar, separados por espacios. Cada bit se presenta en la forma `<frame_address_offset>_<bit_position>`. El `<frame_address_offset>` es la *Minor Address* (la cual identifica el *frame* concreto sumándose a la *Base Address*), y la `<bit_position>` indica en qué posición se encuentra el bit a partir del *Tile Offset* (ver Anexo I). Ambos se muestran en base decimal. Los bits que se han de establecer con valor 0 para configurar la característica están precedidos de una exclamación (!). El resto han de ser configurados con valor 1 si se desea activar la característica [62].

Para ilustrar el proceso de obtención de los bits de configuración de un BEL mapeado a una *leaf cell* de la *netlist*, se va a realizar el procedimiento con una *cell* del sub-bloque MB3. La *cell* primitiva en cuestión es:

`tmr_design_i/tmr_system/MB3/microblaze_riscv_0_axi_intc/U0/AXI_LITE_IPIF_I/I_SLAVE_ATTACHMENT/I_DECODER/s_axi_rdata_i[31]_i_4.`

La figura 3.26 presenta la ubicación física del BEL asociado a la *cell* mencionada. Pertenece al *Site SLICE_X42Y104*, de la *Tile CLBLM_R_X29Y104*.

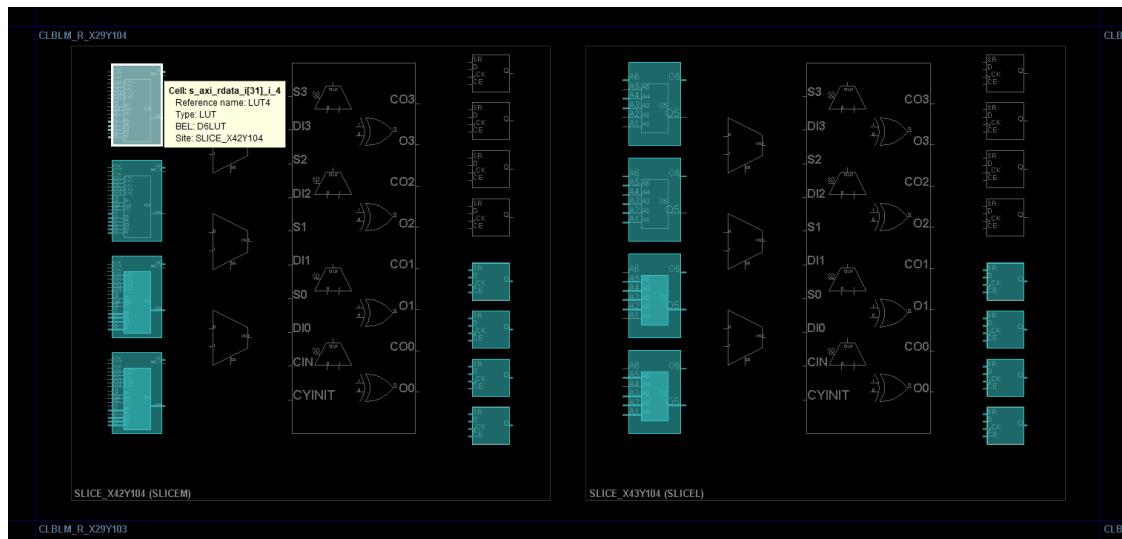


Fig. 3.26. BEL D6LUT al que está mapeado una *cell* primitiva del sub-bloque MB3.

Con esos datos, se busca en el archivo *tilegrid.json* de la FPGA utilizada la entrada correspondiente a la *Tile*:

```

1  "CLBLM_R_X29Y104": {
2      "bits": {
3          "CLB_IO_CLK": {
4              "baseaddr": "0x00000E80",
5              "frames": 36,
6              "offset": 8,
7              "words": 2
8          }
9      },
10     "clock_region": "X0Y2",
11     "grid_x": 77,
12     "grid_y": 47,
13     "pin_functions": {},
14     "prohibited_sites": [],
15     "sites": {
16         "SLICE_X42Y104": "SLICEM",
17         "SLICE_X43Y104": "SLICEL"
18     },
19     "type": "CLBLM_R"
20 }
```

De esta forma se obtienen los siguientes datos relevantes:

- ***Base Address*** 0x00000E80
- ***Tile Offset*** 8
- ***Site Type*** SLICEM
- ***Tile Type*** CLBLM_R

Tras ello, se analiza el archivo *segbits_clblm_r.db* en busca de las líneas que comiencen por *CLBLM_R.SLICEM_X0.DLUT*, ya que el BEL se encuentra en un *Tile* tipo CLBLM_R, en un *Site* tipo SLICEM, y es de tipo DLUT. Se ha de resaltar que el índice *X0* hace referencia a que es el *Site* izquierdo. En este caso no existe un SLICEM_X1 porque las *Tiles* CLB de memoria siempre tienen una SLICEM a la izquierda y una SLICEL a la derecha. Si se quisiera identificar la BEL D6LUT de la *Slice* derecha se debería buscar *CLBLM_R.SLICEL_X1.DLUT*. Estas son algunas de las líneas relevantes:

```

1 CLBLM_R.SLICEM_X0.DLUT.INIT[00] 34_63
2 CLBLM_R.SLICEM_X0.DLUT.INIT[01] 35_63
3 CLBLM_R.SLICEM_X0.DLUT.INIT[02] 34_62
4 <...>
5 CLBLM_R.SLICEM_X0.DLUT.INIT[61] 33_49
6 CLBLM_R.SLICEM_X0.DLUT.INIT[62] 32_48
7 CLBLM_R.SLICEM_X0.DLUT.INIT[63] 33_48
8 CLBLM_R.SLICEM_X0.DLUT.RAM 31_47
9 CLBLM_R.SLICEM_X0.DLUT.SMALL 01_59
10 CLBLM_R.SLICEM_X0.DLUT.SRL 30_47

```

Así, se obtienen la *Minor Address* y el *Bit Offset* de los bits relevantes, con lo que se puede calcular la dirección exacta del los bits, determinada por la *Base Address*, *Minor Address*, *Word Address* y *Bit Address*. Las dos últimas direcciones se han de calcular como muestran las ecuaciones 3.1 y 3.2, respectivamente.

$$\text{Word Address} = \text{Tile Offset} + \left\lfloor \frac{\text{Bit Position}}{32} \right\rfloor \quad (3.1)$$

$$\text{Bit Address} \equiv \text{Bit Position} \pmod{32} \quad (3.2)$$

Con todo ello, y como se observa en la tabla 3.9, se puede calcular el comando de inyección siguiendo el formato de la tabla 3.5.

Base Address	Tile Offset	Minor Address	Bit Position	Word Address	Bit Address	Injection Command Value	Injection Command
0x000000E80 (0b0000 0000 0000 0000 0000 1110 1000 0000)	8	34 (0b0100010)	63	9 (0b0001001)	31 (0b11111)	0b0 00 000000000000000011101 0100010 0001001 11111	N 0000EA213F
		35 (0b0100011)	63	9 (0b0001001)	31 (0b11111)	0b0 00 000000000000000011101 0100011 0001001 11111	N 0000EA313F
		34 (0b0100010)	62	9 (0b0001001)	30 (0b11110)	0b0 00 000000000000000011101 0100010 0001001 11110	N 0000EA213E
		33 (0b0100001)	49	9 (0b0001001)	17 (0b10001)	0b0 00 000000000000000011101 0100001 0001001 10001	N 0000EA1131
		32 (0b0100000)	48	9 (0b0001001)	16 (0b10000)	0b0 00 000000000000000011101 0100000 0001001 10000	N 0000EA0130
		33 (0b0100001)	48	9 (0b0001001)	16 (0b10000)	0b0 00 000000000000000011101 0100001 0001001 10000	N 0000EA1130
		31 (0b0011111)	47	9 (0b0001001)	15 (0b01111)	0b0 00 000000000000000011101 0011111 0001001 01111	N 0000E9F12F
		01 (0b0000001)	59	9 (0b0001001)	27 (0b11011)	0b0 00 000000000000000011101 0000001 0001001 11011	N 0000E8113B
		30 (0b0001110)	47	9 (0b0001001)	15 (0b01111)	0b0 00 000000000000000011101 0011110 0001001 01111	N 0000E9E12F

TABLA 3.9. COMANDOS DE INYECCIÓN PARA ATACAR EL BEL DESEADO

No obstante, las pruebas preliminares para intentar provocar una pérdida de la concordancia entre sub-bloques resultaron inconsistentes debido a la baja probabilidad que tiene un bit arbitrario de la configuración del BEL de afectar inmediatamente a la ejecución del programa. Para conseguir pruebas más consistentes, se optó por realizar inyecciones masivas de errores a *Tiles* a las que se mapean *leaf cells* de un solo sub-bloque. Para conseguir todos los bits que configuran una *Tile*, *Project X-Ray* proporciona un archivo de máscara para cada tipo de *Tile*. Este archivo contiene una línea por cada bit que configura la *Tile* con este formato:

```

1 | bit <frame_address_offset>_<bit_position>

```

Como se ve, indica la localización de cada bit en el mismo formato que un archivo *segbits*, esto es: <*minor_address*>_<*bit_position*>, donde *Bit Position* es la posición relativa del bit con respecto al *Tile Offset*.

Para automatizar el proceso de cálculo de comandos de inyección, y en línea con las herramientas de utilidad ofrecidas por *Project X-Ray*, se desarrolló el *script* de Python *tile2injection.py*. Este toma como entrada el nombre de una *Tile* o *Site*, el nombre del dispositivo, el bus de configuración deseado, las direcciones de los archivos *tilegrid* y *mask*, y la dirección del archivo de salida. Además, tiene una *flag verbose* para activar la escritura en consola de información adicional. Con esta información, busca los datos relevantes en los *tilegrid* y *mask*, y construye un comando de inyección para cada uno de los bits que configuran la *Tile*. Los únicos argumentos obligatorios para utilizar el *script* son el nombre de la *Tile* o *Site* y el nombre del dispositivo. Si no se especifica bus de configuración, se toma el primero que aparezca en la entrada correspondiente del *tilegrid*. Si no se especifican los archivos *tilegrid* y *mask*, se toman automáticamente los presentes en la base de datos de *Project X-Ray* en función del nombre del dispositivo provisto y del tipo de *Tile*. Por último, si no se especifica un archivo de salida, se imprimen todos los comandos de inyección en *stdout*. El *script* se ha creado respetando las interfaces de las herramientas presentes en el proyecto, por lo que, al igual que estas, permite utilizar el argumento de ayuda *--help* para obtener información sobre su interfaz (ver figura 3.27). El código del *script* se recoge en el Anexo J.

```
(env) dario@LAPTOP-4TQJ54T3:~/prjxray$ python3 ./utils/tile2injection.py
usage: tile2injection.py [-h] [--config_bus CONFIG_BUS] [--tilegrid TILEGRID] [--mask MASK] [--out OUT] [--verbose] tile_or_site part
tile2injection.py: error: the following arguments are required: tile_or_site, part
• (env) dario@LAPTOP-4TQJ54T3:~/prjxray$ python3 ./utils/tile2injection.py --help
usage: tile2injection.py [-h] [--config_bus CONFIG_BUS] [--tilegrid TILEGRID] [--mask MASK] [--out OUT] [--verbose] tile_or_site part

Generate error injection commands to use in Soft Error Mitigation Controller LogiCORE IP for a given tile

positional arguments:
  tile_or_site      Tile or site name. If a site is provided, the output will be based on its parent tile
  part             Part name

optional arguments:
  -h, --help        show this help message and exit
  --config_bus CONFIG_BUS
                    Configuration bus (also known as block type) of the tile
  --tilegrid TILEGRID
                    Path to part's tilegrid.json file
  --mask MASK
                    Path to tile's mask file
  --out OUT
                    Output file to write the error injection commands. If not specified, it will print to stdout
  --verbose         Print verbose output
```

Fig. 3.27. Utilización de la ayuda del *script* *tile2injection.py*.

3.5.2. Pruebas realizadas

Las pruebas para verificar el sistema se dividieron en dos bloques: TMR y SEM, en función del objetivo del test. Las pruebas listadas se desarrollaron para satisfacer las condiciones de verificación de los requisitos (ver Anexo A).

Pruebas TMR

Las pruebas del subsistema TMR buscan verificar la transición entre sus diferentes estados: funcionamiento nominal, *lockstep* y error fatal. Para ello, el requisito RF-10 establece realizar 20 transiciones entre estado nominal y *lockstep* atacando cada uno de los sub-bloques. A su vez, el requisito RF-10 plantea que, tras dejar fuera de funcionamiento un sub-bloque, se ataque otro distinto para provocar un error fatal, repitiendo 10 veces cada combinación ordenada del primer y segundo sub-bloque atacado. Estas pruebas pueden por lo tanto realizarse de forma conjunta, pues para desconfigurar 10 veces la misma pareja de sub-bloques en el mismo orden, se requieren 10 iteraciones de la prueba. Al existir dos posibles sub-bloques como segunda víctima, cada sub-bloque actúa como primera víctima 10 veces. Esto se puede visualizar en la figura 3.28.

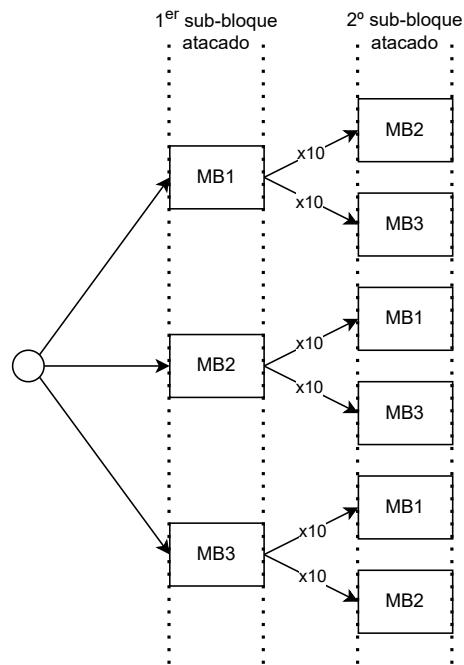


Fig. 3.28. Flujo de las pruebas del subsistema TMR.

Para atacar a un sub-bloque, se utilizó el script *tile2injection.py* para generar los comandos de inyección automáticamente, y estos se fueron remitiendo a la interfaz SEM hasta provocar la desconfiguración deseada. Las *Tiles* seleccionadas para las inyecciones se escogieron de las áreas con una agrupación exclusiva de *cells* de un mismo sub-bloque (ver figura 3.18).

Pruebas SEM

Las pruebas del IP SEM, a diferencia de las anteriores, se plantean sobre bits en ubicaciones aleatorias de los *frames* que configuran el dispositivo. Por ello, en lugar de utilizar el esquema de direccionamiento físico, se utilizó el lineal. Así, se puede seleccionar fácilmente un *frame* aleatorio del rango válido —entre 0 y MF (*Maximum Frame*)—, y se evita tener que buscar unas *Base Address* y *Minor Address* válidas. No obstante, aunque el rango válido sea [0..MF], se recomienda realizar las inyecciones en el rango [2/3 MF..MF], ya que en los *frames* con una LFA reducida tienen bastantes bits enmascarados, por lo que inyectar un error en ellos podría no hacer nada [68]. Para obtener el *Maximum Frame* del dispositivo, hay que enviar un comando de informe de estado (S). Al hacerlo, se recibió la siguiente información:

1	MF 00001F0C
2	SN 00
3	SC 02
4	FC 00
5	FS 0B

La primera línea determina que el *Maximum Frame* es 0x00001F0C, lo que es 7948 en decimal. Este dato informa sobre que la memoria de configuración del ZYNQ XC7Z020-1CLG400C se divide en 7949 *frames*. Los tests realizados son principalmente los descritos en los requisitos:

- **Test del requisito RF-06** Su objetivo es evaluar la corrección de errores simples.
 1. Selección de un *frame* lineal aleatorio en el rango [2/3 MF..MF].
 2. Selección de una palabra aleatoria en el rango [0..100].
 3. Selección de un bit aleatorio en el rango [0..31]
 4. Inyección del error en el bit seleccionado.
 5. Evaluación de los informes generados. El error se ha tenido que corregir.
 6. Repetir los pasos hasta llegar a 20 iteraciones.
- **Test del requisito RF-07** Su objetivo es evaluar la corrección de errores dobles adyacentes.
 1. Selección de un *frame* lineal aleatorio en el rango [2/3 MF..MF].
 2. Selección de una palabra aleatoria en el rango [0..100].
 3. Selección de un bit aleatorio en el rango [0..31].
 4. Selección de un segundo bit, adyacente al previamente seleccionado. Se escoge aleatoriamente entre el bit a su izquierda o derecha. Si el primer bit seleccionado se encuentra en alguno de los límites del rango, el segundo bit se escoge de forma determinista.

5. Inyección simultánea de los errores en los bits seleccionados.
 6. Evaluación de los informes generados. Los errores se han tenido que corregir.
 7. Repetir los pasos hasta llegar a 20 iteraciones.
- **Test del requisito RF-08** Su objetivo es evaluar la corrección de errores múltiples de longitud de un bit distribuidos en *frames* diferentes.
1. Selección de un número de *frames* aleatorio.
 2. Selección de tantos *frames* lineales aleatorios en el rango [2/3 MF..MF] como se hayan determinado en el paso anterior.
 3. Selección de una palabra aleatoria en el rango [0..100] para cada *frame* seleccionado.
 4. Selección de un bit aleatorio en el rango [0..31] para cada *frame* seleccionado.
 5. Inyección simultánea de los errores en los bits seleccionados.
 6. Evaluación de los informes generados. Los errores se han tenido que corregir.
 7. Repetir los pasos hasta llegar a 20 iteraciones.
- **Test del requisito RF-09** Su objetivo es evaluar la corrección de errores de múltiples parejas adyacentes distribuidas en *frames* diferentes.
1. Selección de un número de *frames* aleatorio.
 2. Selección de tantos *frames* lineales aleatorios en el rango [2/3 MF..MF] como se hayan determinado en el paso anterior.
 3. Selección de una palabra aleatoria en el rango [0..100] para cada *frame* seleccionado.
 4. Selección de un bit aleatorio en el rango [0..31] para cada *frame* seleccionado.
 5. Selección de un segundo bit para cada *frame*, adyacente al previamente seleccionado. Se escoge aleatoriamente entre el bit a su izquierda o derecha. Si el primer bit seleccionado se encuentra en alguno de los límites del rango, el segundo bit se escoge de forma determinista.
 6. Inyección simultánea de los errores en los bits seleccionados.
 7. Evaluación de los informes generados. Los errores se han tenido que corregir.
 8. Repetir los pasos hasta llegar a 20 iteraciones.
- **Test de un error doble adyacente en dos palabras** Su objetivo es evaluar el caso en el que cada bit de un error doble recae en palabras distintas.
1. Selección de un *frame* lineal aleatorio en el rango [2/3 MF..MF].
 2. Selección de una palabra aleatoria en el rango [1..99].
 3. Selección de un bit aleatorio en el set 0, 31.

4. Selección de un segundo bit, adyacente al previamente seleccionado. Si el bit anterior es el 0, seleccionar el bit 31 de la anterior palabra. Si el bit anterior es el 31, seleccionar el bit 0 de la siguiente palabra.
 5. Inyección simultánea de los errores en los bits seleccionados.
 6. Evaluación de los informes generados. Los errores se han tenido que corregir.
 7. Repetir los pasos hasta llegar a 20 iteraciones.
- **Test de un error doble no adyacente** Su objetivo es evaluar que los errores dobles no adyacentes no se corrigen.
 1. Selección de un *frame* lineal aleatorio en el rango [2/3 MF..MF].
 2. Selección de una palabra aleatoria en el rango [0..100].
 3. Selección de un bit aleatorio en el rango [0..31].
 4. Selección de un segundo bit aleatorio, no adyacente al anterior.
 5. Inyección simultánea de los error en los bits seleccionados.
 6. Evaluación de los informes generados. Los errores no han de haberse corregido.
 7. Repetir los pasos hasta llegar a 20 iteraciones.

El software ejecutado en el SoC durante las pruebas es el expuesto en los anexos E y F. Por otro lado, la monitorización UART se realizó con el *Serial Monitor* de Visual Studio Code (ver figura 3.29), que ofrece más funcionalidades que el de Vitis. Entre cada iteración de las pruebas, se reprogramó la placa para garantizar la independencia de las mismas. Asimismo, en caso de realizar una inyección en un bit enmascarado, se repitió la iteración en direcciones distintas, puesto que el bit enmascarado no permite evaluar el comportamiento del controlador.

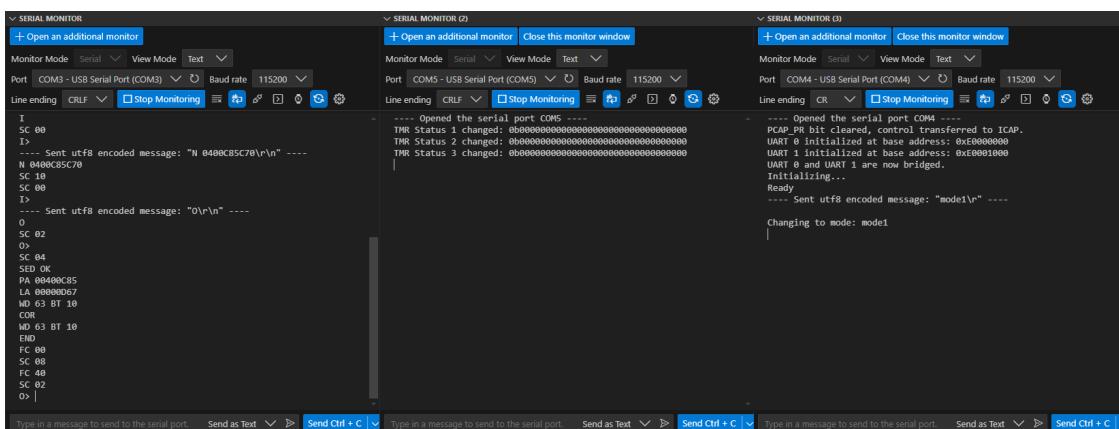
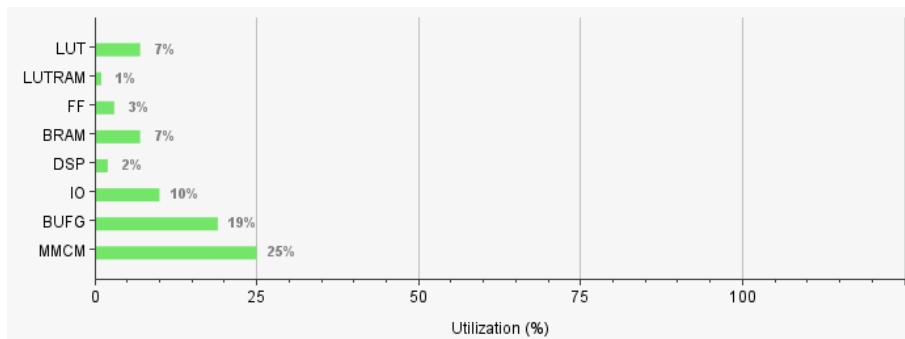


Fig. 3.29. Interfaces para la realización de las pruebas. A la izquierda: interfaz de monitorización SEM. En el centro: interfaz de monitorización del estado del subsistema TMR. A la derecha: interfaz de comunicación con el programa ejecutado en el subsistema TMR.

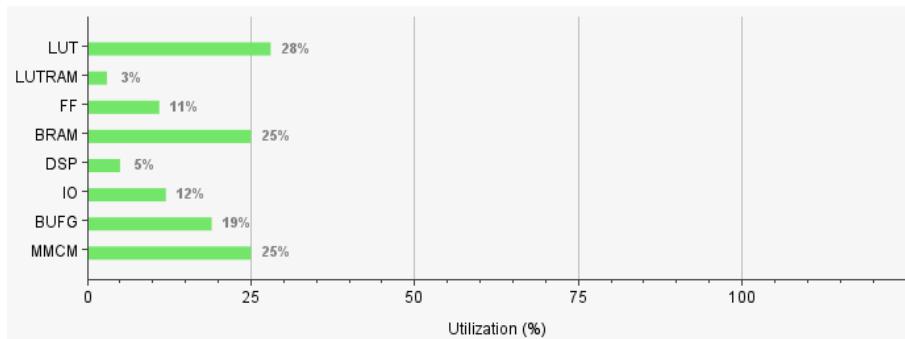
4. RESULTADOS

En este capítulo se discutirán los resultados de las pruebas expuestas en la sección anterior, así como la diferencia de utilización de recursos entre los sistemas tolerante y no tolerante a fallos.

4.1. Utilización de recursos



(a) Sistema no tolerante a fallos



(b) Sistema tolerante a fallos

Fig. 4.1. Comparación de la utilización de los recursos del dispositivo por parte del sistema no tolerante a fallos y del tolerante a fallos.

La figura 4.1 expone la utilización de diferentes recursos del dispositivo con cada uno de los diseños. Como se ve, el tolerante a fallos realiza un uso más intensivo de los recursos, pero el efecto de la triplicación no se traslada de forma equivalente entre diferentes tipos de recursos. Si se ordenan los recursos por el aumento experimentado, se obtiene lo recogido en la tabla 4.1.

Recurso	Relación de uso
LUT	x4
FF	x3,7
BRAM	x3,6
LUTRAM	x3
DSP	x2,5
IO	x1,2
BUFG	x1
MMCM	x1

TABLA 4.1. RELACIÓN DEL USO DE CADA RECURSO EN EL SISTEMA ENDURECIDO CON RESPECTO AL INICIAL

Las LUTs, que se utilizan para implementar funciones lógicas, son las que reciben el mayor aumento. A pesar de que el sub-bloque se triplique, toda la lógica de votación y sincronización necesita recursos que la implementen, por lo que es esperable el aumento x 4. Además, más allá de la lógica adicional propia del TMR, se añadieron otros IPs para la gestión de la monitorización. Los otros recursos de estado, memoria y procesamiento de señales digitales (FF, BRAM, LUTRAM, y DSP) aumentaron en valores cercanos al x 3. En cambio, los recursos globales compartidos de reloj: BUFG y MMCM mantienen una utilización constante, puesto que todo el sistema sigue estando conectado a la misma red de reloj. Por último, IO incrementa ligeramente por la adición de los puertos Pmod necesarios para la monitorización UART.

4.2. Resultados de las pruebas

■ Pruebas TMR

Las pruebas del subsistema TMR, además de verificar el correcto funcionamiento de este, revelaron que las señales de estado emitidas ante un error fatal son las mismas independientemente del orden en el que se desconfiguren los sub-bloques. Así, tan solo las señales emitidas ante la pérdida de coherencia del primer sub-bloque informan sobre la ubicación del error. La tabla 4.2, que muestra las señales de estado obtenidas en las diferentes combinaciones de ataques, agrupa por lo tanto las pruebas que en la figura 3.28 bifurcaban de cada primer bloque atacado.

1er sub-bloque atacado	2º sub-bloque atacado	Secuencia de estados
MB1	MB2 o MB3	TMR Status 1 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 2 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 3 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 1 changed: 0b0000000000000000 001 01 000000 0 011 TMR Status 2 changed: 0b0000000000000000 001 01 000000 0 011 TMR Status 3 changed: 0b0000000000000000 001 01 000000 0 011 TMR Status 1 changed: 0b0000000000000000 000 11 000111 0 011 TMR Status 2 changed: 0b0000000000000000 000 11 000111 0 011 TMR Status 3 changed: 0b0000000000000000 000 11 000111 0 011
MB2	MB1 o MB3	TMR Status 1 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 2 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 3 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 1 changed: 0b0000000000000000 010 01 000000 0 101 TMR Status 2 changed: 0b0000000000000000 010 01 000000 0 101 TMR Status 3 changed: 0b0000000000000000 010 01 000000 0 101 TMR Status 1 changed: 0b0000000000000000 000 11 000111 0 101 TMR Status 2 changed: 0b0000000000000000 000 11 000111 0 101 TMR Status 3 changed: 0b0000000000000000 000 11 000111 0 101
MB3	MB1 o MB2	TMR Status 1 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 2 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 3 changed: 0b0000000000000000 000 00 000000 0 000 TMR Status 1 changed: 0b0000000000000000 100 01 000000 0 110 TMR Status 2 changed: 0b0000000000000000 100 01 000000 0 110 TMR Status 3 changed: 0b0000000000000000 100 01 000000 0 110 TMR Status 1 changed: 0b0000000000000000 000 11 000111 0 110 TMR Status 2 changed: 0b0000000000000000 000 11 000111 0 110 TMR Status 3 changed: 0b0000000000000000 000 11 000111 0 110

TABLA 4.2. RESULTADOS MAYORITARIOS DE LAS PRUEBAS TMR

A pesar de que la gran mayoría de pruebas ofreció el resultado recogido en la tabla 4.2, se dieron dos ocasiones en las que el resultado fue diferente:

- **Inyección a *CLBLM_R_X61Y71* (MB1)** En una de las ocasiones en las que que se atacó el sub-bloque MB1, se realizaron un conjunto simultáneo de inyecciones a la *Tile CLBLM_R_X61Y71* —con presencia exclusiva de *cells* del MB1— que desencadenaron un error fatal del sistema. A continuación se muestra la salida obtenida:

```

1 TMR Status 1 changed: 0b00000000000000000000 000 00 000000 0 000
2 TMR Status 2 changed: 0b00000000000000000000 000 00 000000 0 000
3 TMR Status 3 changed: 0b00000000000000000000 000 00 000000 0 000
4
5 TMR Status 1 changed: 0b00000000000000000000 001 01 000000 0 011
6 TMR Status 2 changed: 0b00000000000000000000 001 01 000000 0 011
7 TMR Status 3 changed: 0b00000000000000000000 001 01 000000 0 011
8
9 TMR Status 1 changed: 0b00000000000000000000 000 11 000111 0 011
10 TMR Status 2 changed: 0b00000000000000000000 000 11 000111 0 011
11 TMR Status 3 changed: 0b00000000000000000000 000 11 000111 0 011

```

Como se observa, uno de los comandos de inyección enviados desconfiguró el sub-bloque MB1, pero, inesperadamente, otro de los comandos desconfiguró otro subbloque distinto y derivó en error fatal. A pesar de que todos los comandos ataqueen a bits que configuran una *Tile* con tan solo *cells* de uno de los sub-bloques, es posible que el cambio termine afectando a otros sub-bloques. Hay que comprender la configuración de la FPGA como un todo: el diseño no solo se programa con los BELs activos, sino que depende de que aquellos que no se están utilizando se mantengan de ese modo. Por lo tanto, un ataque a una *Tile* que solo contenga BELs asociados a *cells* del MB1, aunque improbable, puede provocar un cambio en el enrutamiento u otro aspecto del circuito que termine afectando a otro sub-bloque. Esto demuestra que, aunque efectivo en la mayoría de casos, este sistema de inyecciones no es infalible y no asegura un fallo en el sub-bloque deseado.

- **Inyección a *CLBLM_R_X61Y71* (MB1) y *CLBLM_R_X29Y104* (MB2)** En este segundo caso particular, se comenzó realizando una primera inyección a la *Tile CLBLM_R_X61Y71*, con *cells* del MB1. Esta inyección sucedió sin imprevistos. No obstante, al realizar la inyección a la *Tile CLBLM_R_X29Y104*, con *cells* del MB2, este sub-bloque emitió una señal de estado adicional antes de transicionar al valor esperado:

```

1 TMR Status 1 changed: 0b00000000000000000000 001 01 000000 0 011
2 TMR Status 2 changed: 0b00000000000000000000 001 01 000000 0 011
3 TMR Status 3 changed: 0b00000000000000000000 001 01 000000 0 011
4
5 TMR Status 1 changed: 0b00000000000000000000 001 01 000000 0 011
6 TMR Status 2 changed: 0b00000000000000000000 001 01 000000 0 011
7 TMR Status 3 changed: 0b00000000000000000000 001 01 000000 0 011
8
9 TMR Status 2 changed: 0b00000000000000000000 001 11 000110 0 011
10
11 TMR Status 3 changed: 0b00000000000000000000 000 11 000110 0 011
12 TMR Status 1 changed: 0b00000000000000000000 000 11 000110 0 011
13 TMR Status 2 changed: 0b00000000000000000000 000 11 000110 0 011

```

Como se ve, en la señal aislada del MB2 el campo *Processor Faults* sigue apuntando a un fallo en el procesador 1, a pesar de que el campo *FT State* ya indica un error fatal. Esto, como se ve en la señal emitida después, se corrigió. Es probable que los bits del campo *FT State* cambiaron antes de que lo hiciera el de *Processor Faults*, y que el controlador del estado lo detectase. Sin embargo, al haberse producido tan solo este caso, no es posible obtener reflexiones concluyentes. Sea como fuere, el subsistema transicionó bien entre los estados. Otra particularidad que se puede identificar al observar los cambios de estado es que en el último grupo primero se emite el del MB3, luego el del MB1 y finalmente el del MB2. Esto, sin embargo, no representa ninguna particularidad y tan solo es producto de la forma en la que el controlador de los estados escanea los cambios en las señales. Estos se comprueban en bucle infinito en el orden MB1-MB2-MB3, por lo que si el cambio de estado se produce justo después de que se haya comprobado la señal de MB2, la primera en imprimirse será la del MB3.

Se ha de resaltar que los espacios y líneas vacías de los informes de cambio de estado se han añadido en el documento para aumentar la legibilidad, separando los diferentes campos de las señales y las agrupaciones de estas.

- **Pruebas SEM** Las pruebas del IP SEM son esenciales para garantizar que en un entorno radiactivo los errores no se acumulen con el tiempo. Como ya se explicó en la metodología, en cada prueba se realizaron 20 inyecciones aleatorias independientes utilizando direccionamiento lineal en *frames* con direcciones en el rango [2/3 MF..MF].

- **Test del requisito RF-06** Los errores simples se corrigieron con éxito en todas las pruebas. En la figura 4.2 se muestra una de las pruebas como ejemplo. En ella, se inyecta el error especificado en la tabla 4.3 y se recibe un informe de corrección que indica que se ha corregido correctamente.

Linear Frame	Word	Bit	Injection Command
5436	69	26	N C00153C8BA

TABLA 4.3. EJEMPLO DE INYECCIÓN DEL TEST DEL REQUISITO RF-07

```

X7_SEM_V4_1
SC 01
FS 0B
ICAP OK
RDBK OK
INIT OK
SC 02
O>
---- Sent utf8 encoded message: "I\r\n" ----
I
SC 00
I>
---- Sent utf8 encoded message: "N C00153C8BA\r\n" ----
N C00153C8BA
SC 10
SC 00
I>
---- Sent utf8 encoded message: "O\r\n" ----
O
SC 02
O>
SC 04
SED OK
PA 0042B418
LA 0000153C
WD 45 BT 1A
COR
WD 45 BT 1A
END
FC 00
SC 08
FC 40
SC 02
O> |

```

Fig. 4.2. Interfaz de monitorización SEM
que muestra la inyección de un
error simple.

- **Test del requisito RF-07** Los errores dobles adyacentes se corrigieron con éxito en todas las pruebas. Se expone de ejemplo una de las inyecciones realizadas, descrita en la tabla 4.4 y cuyo resultado se recoge en la figura 4.3. Esta contiene el informe de corrección de errores donde se indica que ambos bits se corrigieron.

Linear Frame	Word	Bit	Injection Command
7054	100	14	N C001B8EC8E
7054	100	15	N C001B8EC8F

TABLA 4.4. EJEMPLO DE INYECCIONES DEL TEST DEL
REQUISITO RF-07

```

X7_SEM_V4_1
SC 01
FS 0B
ICAP OK
RDBK OK
INIT OK
SC 02
O>
---- Sent utf8 encoded message: "I\r\n" ----
I
SC 00
I>
---- Sent utf8 encoded message: "N C001B8EC8E\r\n" ----
N C001B8EC8E
SC 10
SC 00
I>
---- Sent utf8 encoded message: "N C001B8EC8F\r\n" ----
N C001B8EC8F
SC 10
SC 00
I>
---- Sent utf8 encoded message: "O\r\n" ----
O
SC 02
O>
SC 04
DED
PA 00421B8A
LA 00001B8E
COR
MD 64 BT 0E
MD 64 BT 0F
END
FC 00
SC 08
FC 40
SC 02
O> |

```

Fig. 4.3. Interfaz de monitorización SEM
que muestra la inyección de un
error doble adyacente.

- **Test del requisito RF-08** Los errores múltiples con los bits distribuidos en *frames* diferentes se corrigieron con éxito en todas las pruebas. En la tabla 4.5 se observan todos los bits que se atacaron simultáneamente en una de ellas. El resultado de la prueba se recoge en la figura 4.4, que se divide en cuatro columnas para acomodar el gran tamaño de la interfaz. Por lo tanto, la segunda columna ha de leerse como una continuación de la primera, y así sucesivamente. Si se analiza, se aprecian seis informes de corrección de errores —uno para cada bit— que informan sobre la exitosa corrección de estos.

Linear Frame	Word	Bit	Injection Command
7308	70	29	N C001C8C8DD
5505	92	8	N C001581B88
6588	3	30	N C0019BC07E
7144	63	1	N C001BE87E1
6438	91	26	N C001926B7A
6080	54	14	N C0017C06CE

TABLA 4.5. EJEMPLO DE INYECCIONES DEL TEST DEL
REQUISITO RF-08

X7_SEM_V4_1	SC 01	SC 00	SC 10	SC 08	MD 3F BT 01
FS 00	TCAP OK	D	----- Sent utf8 encoded message: "N C0017C06CE\r\n"	FC 40	COR
LCAP OK	RDBK OK	SC 02	----- Sent utf8 encoded message: "N C0017C06CE\r\n"	SC 02	MD 3F BT 01
INIT OK	SC 02	SC 00	----- Sent utf8 encoded message: "0\r\n"	DED	END
SC 02	O>	SC 10	----- Sent utf8 encoded message: "0\r\n"	FC 40	FC 40
----- Sent utf8 encoded message: "I\r\n"	I	SC 00	----- Sent utf8 encoded message: "0\r\n"	SC 00	SC 00
I	SC 00	D>	----- Sent utf8 encoded message: "N C001C8C800\r\n"	FC 40	FC 40
----- Sent utf8 encoded message: "N C001C8C800\r\n"	N C001C8C800	SC 10	PA 00421000	SC 02	SC 02
SC 10	SC 00	D	LA 00001501	SC 02	O>
----- Sent utf8 encoded message: "N C0015181B88\r\n"	N C0015181B88	PA 00420510	MD 5B BT 1A	COR	SC 04
SC 10	SC 00	I>	LA 000015181	MD 5B BT 1A	SED OK
----- Sent utf8 encoded message: "N C0019BC07E\r\n"	N C0019BC07E	SC 02	PA 00421300	PA 00421F1C	PA 00421F1C
SC 10	SC 00	D	LA 00001502	LA 00001C8C	LA 00001C8C
----- Sent utf8 encoded message: "N C001BE87E1\r\n"	N C001BE87E1	PA 00420508	MD 46 BT 1D	MD 46 BT 1D	MD 46 BT 1D
SC 10	SC 00	I>	LA 000017C0	COR	COR
----- Sent utf8 encoded message: "N C001BE87E1\r\n"	N C001BE87E1	SC 02	MD 36 BT 0E	MD 36 BT 0E	MD 36 BT 0E
SC 10	SC 00	D	SC 02	SC 02	SC 02
----- Sent utf8 encoded message: "N C0019266B7A\r\n"	N C0019266B7A	PA 00420502	SC 02	SC 02	O>
N C0019266B7A	FC 40	PA 00421D00	PA 00421D00	PA 00421D00	PA 00421D00

Fig. 4.4. Interfaz de monitorización SEM que muestra la inyección de un error séxtuple distribuido de forma unitaria por *frames* distintos.

- **Test del requisito RF-09** Los errores múltiples de parejas adyacentes de bits en *frames* diferentes se corrigieron con éxito en todas las pruebas. Una de esas pruebas se ha plasmado en la tabla 4.6, en la que se muestran agrupadas las inyecciones pertenecientes a la misma pareja. En la figura 4.5 se pueden ver los cuatro informes de corrección de errores —uno para cada pareja— que indican que todos se corrigieron con éxito.

Linear Frame	Word	Bit	Injection Command
5830	77	24	N C0016C69B8
5830	77	25	N C0016C69B9
7095	94	15	N C001BB7BCF
7095	94	14	N C001BB7BCE
6494	99	6	N C00195EC66
6494	99	5	N C00195EC65
5429	65	21	N C001CB7835
5429	65	21	N C001CB7836

TABLA 4.6. EJEMPLO DE INYECCIONES DEL TEST DEL REQUISITO RF-09

X7_SEM_V4_1	SC 01	SC 00	SC 10	SC 04	FC 40
FS 00	TCAP OK	D	----- Sent utf8 encoded message: "N C00195EC65\r\n"	DED	SC 00
LCAP OK	RDBK OK	SC 02	----- Sent utf8 encoded message: "N C00195EC65\r\n"	PA 00421394	FC 40
INIT OK	SC 02	SC 00	----- Sent utf8 encoded message: "0\r\n"	LA 00001505E	SC 02
SC 02	O>	SC 10	----- Sent utf8 encoded message: "0\r\n"	COR	O>
----- Sent utf8 encoded message: "I\r\n"	I	SC 00	----- Sent utf8 encoded message: "0\r\n"	MD 63 BT 05	MD 5E BT 0E
I	SC 00	D	PA 00421005	MD 63 BT 05	MD 5E BT 0E
----- Sent utf8 encoded message: "N C001C8C800\r\n"	N C001C8C800	SC 10	PA 00420508	MD 63 BT 05	MD 5E BT 0E
N C001C8C800	SC 10	I>	LA 00001502	COR	O>
----- Sent utf8 encoded message: "N C001C8C800\r\n"	N C001C8C800	SC 02	PA 00421300	PA 00421C0F	PA 00421C0F
SC 10	SC 00	D	LA 00001503	LA 00001B07	LA 00001B07
----- Sent utf8 encoded message: "N C0019BC07E\r\n"	N C0019BC07E	PA 00420502	SC 02	MD 5E BT 0E	MD 5E BT 0E
SC 10	SC 00	I>	PA 00420502	PA 00420502	PA 00420502
----- Sent utf8 encoded message: "N C0019BC07E\r\n"	N C0019BC07E	SC 02	PA 00420502	PA 00420502	PA 00420502
SC 10	SC 00	D	PA 00420502	PA 00420502	PA 00420502
----- Sent utf8 encoded message: "N C0019266B7A\r\n"	N C0019266B7A	PA 00420502	SC 02	SC 02	O>
N C0019266B7A	FC 40	PA 00421D00	PA 00421D00	PA 00421D00	PA 00421D00

Fig. 4.5. Interfaz de monitorización SEM que muestra la inyección de un error óctuple distribuido en parejas adyacentes por *frames* distintos.

- **Test de un error doble adyacente localizado en dos palabras distintas** Los errores dobles adyacentes que abarcan varias palabras se corrigieron con éxito en todas las pruebas, lo que indica que su distribución en el *frame* es irrelevante, siempre y cuando sus bits sean adyacentes. La tabla 4.7 y la figura 4.6 contienen las inyecciones de una de las pruebas y su resultado, respectivamente.

Linear Frame	Word	Bit	Injection Command
6346	21	0	N C0018CA2A0
6346	20	31	N C0018CA29F

TABLA 4.7. EJEMPLO DE INYECCIONES DEL TEST DE UN ERROR DOBLE ADYACENTE LOCALIZADO EN DOS PALABRAS DISTINTAS

```
X7_SEM_V4_1
SC 01
FS 0B
ICAP OK
RDBK OK
INIT OK
SC 02
O>
---- Sent utf8 encoded message: "I\r\n" ----
I
SC 00
I>
---- Sent utf8 encoded message: "N C0018CA2A0\r\n" ----
N C0018CA2A0
SC 10
SC 00
I>
---- Sent utf8 encoded message: "N C0018CA29F\r\n" ----
N C0018CA29F
SC 10
SC 00
I>
---- Sent utf8 encoded message: "O\r\n" ----
O
SC 02
O>
SC 04
DED
PA 00421188
LA 000018CA
COR
WD 14 BT 1F
WD 15 BT 00
END
FC 00
SC 08
FC 40
SC 02
O> |
```

Fig. 4.6. Interfaz de monitorización SEM que muestra la inyección de un error doble adyacente en el que cada bit pertenece a una palabra distinta.

- **Test de un error doble no adyacente** Los errores dobles no adyacentes no se corrigieron en ninguna de las pruebas, lo que se corresponde con el comportamiento esperado del IP. En la tabla 4.8 se muestran las inyecciones de una de las pruebas realizadas, y en la figura 4.7 se ve la lista del informe de corrección de errores vacía, lo que indica que no se pudieron corregir a pesar de que se detectaran.

Linear Frame	Word	Bit	Injection Command
6018	23	15	N C0017822EF
6018	23	8	N C0017822E8

TABLA 4.8. EJEMPLO DE INYECCIONES DEL TEST DE UN ERROR DOBLE NO ADYACENTE

```

X7_SEM_V4_1
SC 01
FS 0B
ICAP OK
RDBK OK
INIT OK
SC 02
O>
---- Sent utf8 encoded message: "I\r\n" ----
I
SC 00
I>
---- Sent utf8 encoded message: "N C0017822EF\r\n" ----
N C0017822EF
SC 10
SC 00
I>
---- Sent utf8 encoded message: "N C0017822E8\r\n" ----
N C0017822E8
SC 10
SC 00
I>
---- Sent utf8 encoded message: "O\r\n" ----
O
SC 02
O>
SC 04
DED
PA 00428C9A
LA 00001782
COR
END
FC 20
SC 08
FC 60
SC 00
I> |

```

Fig. 4.7. Interfaz de monitorización SEM que muestra la inyección de un error doble no adyacente.

5. PLANIFICACIÓN Y PRESUPUESTO

La buena planificación de un proyecto es esencial para su éxito y finalización en plazo. En contraposición a una tarea mecánica que se deba realizar por mera costumbre, ha de ser el eje rector del desarrollo del mismo. Por ello, en este capítulo se abordarán la planificación y el presupuesto del proyecto, y cómo estos difieren o no de su concepción inicial.

5.1. Planificación

Para realizar una planificación efectiva, se identificaron en primer lugar los grandes grupos de tareas a realizar. De esta forma, se define la separación entre diferentes etapas temporales o conceptuales y se sientan las bases para el desgranado en tareas más concretas:

1. **Investigación** Se planificó la investigación alrededor de tres ejes: problema, tecnología y marco regulador.
2. **Análisis del sistema** Inicialmente se planteó realizar la definición de los requisitos y la estimación del presupuesto. Más adelante se añadió una revisión de los requisitos para corregir potenciales errores anclados en etapas iniciales del desarrollo.
3. **Sistema no tolerante a fallos** Antes de desarrollar el sistema tolerante a fallos, es importante familiarizarse con las herramientas de trabajo con un sistema de escala reducida. Además, el sistema TMR se construye sobre un diseño verificado no tolerante a fallos. La planificación sigue el esquema sugerido por la metodología de diseño AMD UltraFast, pero adaptado al flujo de Vivado.
4. **Sistema tolerante a fallos** Se programaron dos iteraciones principales del desarrollo del sistema endurecido. La segunda de ellas se planteó en un espacio temporal mucho más reducido, pues su objetivo era mejorar lo ya establecido, no desarrollar de nuevo desde cero. Las tareas de ambas iteraciones reflejan la metodología UltraFast y el flujo de desarrollo de Vivado.
5. **Redacción del documento** La creación de la memoria se dividió por capítulos.
6. **Tutorización** Las tareas de tutorización abarcan las tutorías realizadas y la revisión del documento.

La planificación extendida de tareas se muestra en el Anexo K.

5.1.1. Planificación inicial

La planificación del trabajo se plasmó en un diagrama de Gantt. Este es ampliamente utilizado en la planificación de proyectos, y permite comprender visualmente las interrelaciones entre tareas y su distribución temporal. El diagrama de Gantt realizado se halla en el Anexo L. Se utilizó la herramienta GanttProject (de código abierto [69]) para su creación. Los grandes corchetes de color negro representan las categorías más amplias de tareas, y bajo ellos se establecen las tareas concretas, en color azul o rojo. Las tareas en el segundo de estos colores se corresponden con la línea base inicial del proyecto. Así, cualquier tarea que haya sufrido retrasos cuenta con una barra roja (la planificación inicial) y otra azul (la planificación final). Si la tarea no se ha retrasado, aparece únicamente en color azul. Las flechas entre tareas indican su interdependencia: aquella tarea que es señalada no se puede comenzar hasta la finalización de la que señala. Por otro lado, las tareas que se solapan verticalmente están programadas para el mismo día. En el caso de las tareas de implementación, estas se solapan ya que constituyen un proceso iterativo en sí mismo en el que se deben revisitar constantemente las subtareas que la componen. Así, se deben realizar todas ellas de forma reiterada a lo largo del mismo día. Por último, las dos secciones violeta abarcan períodos de vacaciones en los que no se trabajó en el proyecto.

El proyecto técnicamente comenzó el 28 de noviembre de 2024 con la reunión preliminar con la tutora. No obstante, no fue hasta la segunda reunión el 30 de enero —en la que se concretó el trabajo— cuando realmente empezó. La planificación inicial se realizó con el objetivo de entregar el proyecto antes del 17 de junio. Sin embargo, y como se puede ver, los retrasos en ciertas fases del desarrollo derivaron en la necesidad de retrasar las planificaciones para garantizar la correcta realización de las pruebas y la memoria.

5.1.2. Planificación final

El principal motivo del retraso es el embotellamiento en las tareas de integración de IPs. Tanto en el diseño del sistema no tolerante a fallos como en el tolerante, se alargaron de forma inesperada los tiempos de las tareas. Al no existir caminos secundarios en la planificación, estos desplazaron al futuro en su totalidad las tareas posteriores. El mayor de los retrasos se produjo con el integrador de IPs durante la primera iteración del diseño del sistema tolerante a fallos. En un comienzo se intentó realizar la triplicación de forma manual, lo que disparó el tiempo de desarrollo. Tras optar por la automatización del proceso, se recuperó el ritmo previsto de trabajo. Durante la segunda iteración, incluso se logró reducir el tiempo estimado de las tareas.

La planificación final además añade tareas de tutorización no contempladas en la inicial. En un comienzo solo se anotaron las dos reuniones que se habían tenido. Sin embargo, esta recoge las tutorías posteriores y la futura revisión del documento, para la cual se ha estimado su finalización el 4 de septiembre. Otra tarea agregada a la planificación final

fue la revisión de los requisitos tras la primera iteración del sistema tolerante a fallos. La duración de cada tarea se expone tanto en el diagrama del Anexo L como en el listado del Anexo K.

El Anexo M muestra el diagrama de recursos, en el que se detalla el tiempo dedicado a cada tarea en forma de porcentaje sobre la jornada de 8 horas. La mayoría de tareas se planificaron con una carga de trabajo semanal de 10 horas, lo que se traduce a 2 horas diarias (se reservan 2 días semanales al descanso). En el diagrama esto se representa con asignaciones del 25 % de la jornada. En el caso de las tutorías, se les asignó un 20 % tanto para el alumno como para la tutora, lo que se interpreta como aproximadamente una hora y media de reunión para ambos.

5.2. Presupuesto

El presupuesto, al igual que la planificación de las tareas, es un pilar fundamental de los proyectos. En este caso, al tratarse de un proyecto exclusivamente académico y no laboral, no se perciben los salarios que se presupuestan y por lo tanto pierde parte de su relevancia. No obstante, es importante conocer los costes que acarrearían proyectos así en un entorno laboral.

5.2.1. Presupuesto inicial

El presupuesto inicial se expone en la tabla 5.1. Este se divide en dos secciones principales: coste personal y de materiales.

- **Personal** Se incluye el salario bruto de un ingeniero júnior y una ingeniera sénior. El alumno, representado por el ingeniero júnior, realiza el grueso del proyecto, por lo que tiene la mayoría de las horas asignadas, pero su salario es inferior. En cambio, la tutora, representada por la ingeniera sénior, solo aporta sus horas de tutorización y revisión, pero tiene un salario más elevado. Las horas presupuestadas son las obtenidas en la planificación con el diagrama de Gantt.
- **Material** Se incluyen todos los gastos no personales. El portátil, al ser un bien de valor elevado con antigüedad, se amortiza. Suponiendo un periodo de amortización de 10 años, tiene una amortización del 50 %, pues se compró hace 5 años. No se ha incluido el precio de la licencia de Vivado, pues esta solo es de pago en caso de trabajar con los dispositivos de alta gama de AMD.

El coste total inicialmente estimado para el proyecto fue de 5481 €. De ellos, el 75 % lo compone el coste personal.

				Gastos
Personal				
Concepto	Horas	Tasa (€/h)		
Ingeniero Júnior	268	15		4,020.00 €
Ingeniera Sénior	3	25		75.00 €
Total personal				4,095.00 €
Materiales				
Concepto	Unidades	Coste unitario (€)	Amortización	
PYNQ-Z1	1	315		315.00 €
Cable USBA-USBC	2	3		6.00 €
PmodUSBUART	1	15		15.00 €
Portátil	1	2100	50%	1,050.00 €
Total materiales				1,386.00 €
Total				5,481.00 €

TABLA 5.1. PRESUPUESTO INICIAL

5.2.2. Presupuesto final

La tabla 5.2 muestra el presupuesto final del proyecto. Con respecto al inicial, se han aumentado las horas de trabajo debido a los retrasos y a las tareas no planificadas inicialmente (como la revisión de la documentación por parte de la tutora). Además, se sumó un módulo PmodUSBUART y un cable USB, necesarios para la monitorización del estado del subsistema TMR. Dicha monitorización se añadió en la revisión de los requisitos, por lo que los materiales necesarios no aparecían en el presupuesto inicial. En total, el presupuesto final es de 6784 €, aproximadamente un 24 % superior al inicial.

				Gastos
Personal				
Concepto	Horas	Tasa (€/h)		
Ingeniero Júnior	332	15		4,980.00 €
Ingeniera Sénior	16	25		400.00 €
Total personal				5,380.00 €
Materiales				
Concepto	Unidades	Coste unitario (€)	Amortización	
PYNQ-Z1	1	315		315.00 €
Cable USBA-USBC	3	3		9.00 €
PmodUSBUART	2	15		30.00 €
Portátil	1	2100	50%	1,050.00 €
Total materiales				1,404.00 €
Total				6,784.00 €

TABLA 5.2. PRESUPUESTO FINAL

6. CONCLUSIONES

En este capítulo se extraen y analizan los hallazgos del proyecto, evaluando la consecución de los objetivos y trazando líneas de trabajo a futuro. Por otro lado, se plantea el impacto socioeconómico del trabajo desarrollado.

6.1. Resumen de los hallazgos

Los resultados del proyecto indican que es posible configurar el PL de una PYNQ-Z1 con un diseño redundante basado en un procesador *softcore* MicroBlaze-V. Además, las pruebas han determinado que la compatibilidad del subsistema TMR para MicroBlaze con el *softcore* MicroBlaze-V es correcta. Por último, también se ha comprobado que las funcionalidades de detección y corrección de errores del IP SEM coinciden con las especificadas.

6.2. Contribuciones

Este trabajo no presenta ninguna contribución sustancial más allá de la confirmación de la viabilidad del uso de la placa comercial PYNQ-Z1 para aplicaciones que requieren diseños tolerantes a fallos. La única contribución adicional, de importancia limitada, es la metodología utilizada para comprobar el funcionamiento del subsistema TMR inyectando múltiples errores a *Tiles* con BELs pertenecientes a un único sub-bloque. No obstante, como se muestra en los resultados, este no es un método infalible para generar exclusivamente errores en el sub-bloque deseado. La mayor contribución del trabajo es, pues, el aprendizaje personal que se ha extraído de su elaboración. El proyecto ha supuesto un primer acercamiento del autor a las FPGAs, las metodologías y programas utilizados para trabajar con estas, y los retos y desafíos que plantea el espacio a los sistemas electrónicos. Por lo tanto, este trabajo no se ha planteado con el objetivo de contribuir al avance del estado de la cuestión, sino con el objeto de aprender sobre un campo no tratado en la carrera de Ingeniería Informática de la UC3M y de exponer en esta memoria los detalles del desarrollo para facilitar el acercamiento de otras personas ajenas al campo.

6.3. Cumplimiento de objetivos

Se ha logrado un cumplimiento pleno de los objetivos planteados inicialmente:

1. **Tolerancia a fallos** Los resultados obtenidos demuestran que el sistema resiste las inyecciones de errores para las cuales fue diseñado. Es capaz de detectar y corregir

los errores planteados en los requisitos y el subsistema TMR enmascara correctamente los fallos en cada uno de los sub-bloques.

2. **Uso de redundancia** Parte de la tolerancia a fallos se ha logrado gracias al subsistema TMR. Además de la redundancia, se ha implementado *scrubbing* de la memoria de configuración.
3. **Uso del *soft-core* MicroBlaze-V** Se ha utilizado el procesador *softcore* MicroBlaze-V, basado en la ISA RISC-V.
4. **Configuración en una placa PYNQ-Z1** Todo el diseño se ha implementado en la placa PYNQ-Z1. Así, se demuestra que el sistema tolerante a fallos se puede configurar en una FPGA asequible con un número moderado de células lógicas. Los resultados de utilización mostrados en la figura 3.19 indican que incluso una FPGA de esta gama es capaz de albergar un diseño de estas características con creces.
5. **Creación de una guía accesible** La redacción de esta memoria se ha realizado con la visión de que personas que no estén familiarizadas con este campo sean capaces de introducirse en él y replicar los resultados obtenidos.

6.4. Mejoras futuras

A pesar de los resultados obtenidos, el alcance del proyecto es limitado. A continuación se plantean algunas tareas pendientes y líneas de investigación y desarrollo futuras que se construyen sobre este trabajo.

6.4.1. Verificación del requisito RF-02

No se ha realizado la prueba de verificación del requisito RF-02, de prioridad inferior al resto (prioridad media). Se ha verificado la configuración del diseño de bloques tal y como indica el requisito, pero es necesario llevar a cabo las pruebas pertinentes para verificarlo empíricamente. Por lo tanto, se ha de explorar la manera de inyectar errores en la memoria de bloques.

6.4.2. Mejora del ataque a bits específicos

Si bien se ha planteado una forma de atacar bits específicos que configuran BELs mapeados a las *cells* primitivas deseadas, sería necesario una investigación más profunda en la materia para determinar el papel de cada bit en la configuración del BEL.

6.4.3. Inyección de instrucciones

El diseño se ha configurado para permitir la inyección de instrucciones arbitrarias a cualquiera de los sub-bloques para provocar la pérdida de la concordancia del mismo, lo que permite comprobar de forma sencilla el correcto funcionamiento del subsistema TMR. Esta comprobación se realizó con inyecciones a la memoria de configuración, por lo que el resultado debería ser el mismo. La figura 6.1 muestra la implementación de la inyección de instrucciones del subsistema TMR. *Address Register* almacena el Program Counter (PC) de la dirección física de la instrucción que se quiere reemplazar. Esta se compara con la dirección de memoria que realmente se ha cargado y, si coinciden, se realiza el reemplazo. La instrucción que se inyecta en su lugar se almacena en el *Instruction Register*, y el *Control Register* permite seleccionar a qué procesadores se inyecta.

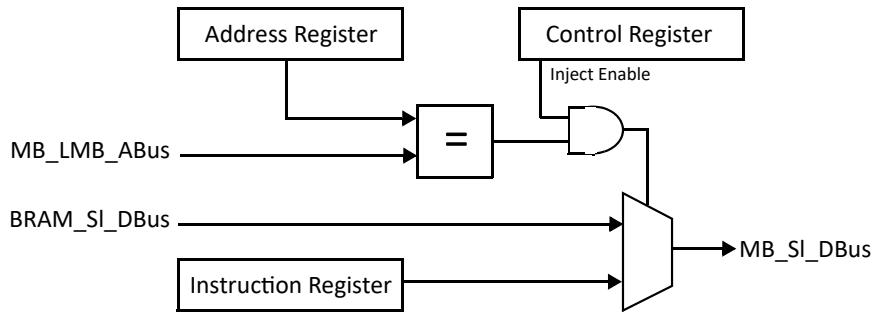


Fig. 6.1. Esquema de bloques de la implementación de la inyección de instrucciones [46].

6.4.4. Simulación de los errores inducidos por un entorno radiactivo

Los errores inyectados durante las pruebas no se asemejan a los efectos de un entorno radiactivo sobre el dispositivo. Para ello, se han de simular aspectos como la tasa de impactos o su distribución física. Se puede desarrollar una herramienta propia o utilizar otras como la *SEUs Simulation Tool* (SST) de la ESA [70].

6.4.5. Pruebas en un laboratorio de radiación

La forma más precisa de simular el efecto de un entorno radiactivo es realizar experimentos de radiación en un laboratorio. Esta es una manera costosa de evaluar la capacidad de tolerancia a fallos del diseño, pero ofrece datos reales sobre el desempeño en entornos radiactivos. Tanto este tipo de pruebas como la simulación permiten realizar comparativas entre el funcionamiento del sistema con y sin los mecanismos de tolerancia a fallos en entornos radiactivos.

6.4.6. Evaluación del SWaP-C de diferentes configuraciones del MicroBlaze-V

Además de las comparativas entre los diseños tolerante y no tolerante a fallos, se puede comparar el rendimiento de las diferentes configuraciones del MicroBlaze-V (ver tabla 3.2): microcontrolador, procesador en tiempo real y procesador de aplicaciones. Dichas comparaciones se pueden realizar para evaluar el SWaP-C del sistema. Para ello, se han de recoger los datos por megahercio tanto de la potencia como del desempeño en bancos de prueba. Se pueden utilizar tres *benchmarks* estándar: *Dhrystone 2*, *Whetstone* y *Coremark* —para enteros y cadenas; punto flotante; y matrices, estructuras de datos y enteros; respectivamente— [16].

6.4.7. Integración con sistemas satelitales reales

Una de las mejoras futuras más ambiciosas sería la integración del diseño en un sistema satelital real. Una de las opciones más factibles sería su integración en un CubeSat. Estos son sátelites de tamaño muy reducido, basados en unidades de medidas estándar denominadas 1U. Las unidades son cubos de aproximadamente 10 cm de lado y entre 1 y 1,33 kg de masa. Los CubeSat de mayor tamaño tienen un diseño equivalente a la combinación de múltiples unidades. Los costes de lanzamiento de un CubeSat 3U a órbita terrestre oscila alrededor de los 300.000 \$. No obstante, el programa *CubeSat Launch Initiative* de la NASA (*National Aeronautics and Space Administration*) financia todos los costes asociados al lanzamiento hasta la cifra mencionada, por lo que se aumenta la viabilidad económica de esta línea de mejora del proyecto. El resto de costes derivados del desarrollo y operación del satélite no están financiados por el programa, pero sigue siendo una de las opciones más factibles para llevar a cabo una integración real del diseño [71].

6.5. Impacto socioeconómico

La investigación sobre el uso de FPGAs, núcleos IP y técnicas de tolerancia a fallos para aplicaciones espaciales acarrea un impacto sustancial en diversas áreas.

6.5.1. Impacto económico

El uso de FPGAs permite reducir los costes de investigación y los tiempos de desarrollo, pues su reconfigurabilidad posibilita la iteración de múltiples diseños y sus respectivas evaluaciones. En este trabajo, se explora uno de esos posibles diseños y se describe el proceso para alcanzar un prototipo funcional en una plataforma *hardware* de coste reducido. Además, la metodología utilizada disminuye el riesgo económico de la investigación, pues permite validar los diseños antes de invertir en *hardware* más robusto pero no reconfigurable. A pesar de que el impacto económico real de este proyecto será limitado,

investigaciones en estas líneas permiten reducir los costes y mejorar la seguridad y fiabilidad de sistemas que impulsan o de los que dependen grandes sectores económicos. Algunos de esos ejemplos serían los sistemas satelitales de navegación y geolocalización, teledetección, meteorología y monitorización medioambiental, telecomunicación, o experimentación espacial y en microgravedad.

6.5.2. Impacto social

El análisis del impacto social de este trabajo se ha realizado en torno a los Objetivos de Desarrollo Sostenible (ODS) de la ONU.

1. **ODS 2 — Hambre cero** A pesar de que este proyecto no tiene un impacto directo sobre la reducción del hambre, presenta tecnología que podría ser utilizada en misiones espaciales que realicen experimentos de cultivos en microgravedad. Estos experimentos podrían ofrecer soluciones aplicables a la agricultura terrestre que ayuden a reducir el hambre.
2. **ODS 3 — Salud y bienestar** Al igual que con el anterior ODS, este trabajo no afecta de forma directa a la salud y el bienestar. Sin embargo, los experimentos en el espacio pueden producir avances en farmacología y medicina que deriven en una mejora de la salud y el bienestar.
3. **ODS 4 — Educación de Calidad** Este trabajo plantea una forma accesible de introducirse en el ámbito de la microelectrónica espacial. La utilización de un dispositivo comercial asequible, así como el hecho de que este solo sea necesario para la implementación final del diseño (se puede realizar la mayor parte del proyecto en un portátil), permite que en escuelas y universidades se pueda generar material docente y prácticas sobre el diseño en FPGA, tolerancia a fallos y sistemas embebidos con un coste reducido.
4. **ODS 8 — Trabajo decente y crecimiento económico** Este proyecto no tiene un impacto directo sobre este ODS. No obstante, la investigación en tecnología espacial genera puestos de trabajo de calidad de forma directa e indirecta y ayuda a la disminución del desempleo.
5. **ODS 9 — Industria, innovación e infraestructura** Este trabajo fomenta la innovación y el desarrollo tecnológico, el prototipado rápido, la transferencia tecnológica y la creación de infraestructuras resilientes.
6. **ODS 13 — Acción por el clima** A pesar de que este proyecto no tiene una relación directa con este ODS, los sistemas satelitales se pueden utilizar para monitorizar variables climáticas, deforestación o incendios, cambios en el suelo o desastres naturales. Los datos recabados ayudan a plantear soluciones a los problemas climáticos.

A pesar de todos estos beneficios que se pueden derivar de forma directa e indirecta del trabajo, se han de conocer los potenciales riesgos que acarrea. Los dispositivos tolerantes a la radiación son una pieza clave en la aeronáutica y la astronáutica. Por ello, y debido a la estrecha relación de estas disciplinas con la industria militar, la investigación en estas áreas puede fomentar la carrera armamentística y el desarrollo de tecnologías que supongan un riesgo para la humanidad. El uso doble (civil y militar) de estas tecnologías no es una cuestión anodina.

BIBLIOGRAFÍA

- [1] D. Binder, E. C. Smith y A. B. Holman, “Satellite Anomalies from Galactic Cosmic Rays,” *IEEE Transactions on Nuclear Science*, vol. 22, n.º 6, pp. 2675-2680, 1975. doi: [10.1109/TNS.1975.4328188](https://doi.org/10.1109/TNS.1975.4328188).
- [2] E. Normand, “Single-event effects in avionics,” *IEEE Transactions on Nuclear Science*, vol. 43, n.º 2, pp. 461-474, 1996. doi: [10.1109/23.490893](https://doi.org/10.1109/23.490893).
- [3] C. S. Guenzer, E. A. Wolicki y R. G. Allas, “Single Event Upset of Dynamic RAMs by Neutrons and Protons,” *IEEE Transactions on Nuclear Science*, vol. 26, n.º 6, pp. 5048-5052, 1979. doi: [10.1109/TNS.1979.4330270](https://doi.org/10.1109/TNS.1979.4330270).
- [4] J. T. Wallmark y S. M. Marcus, “Minimum Size and Maximum Packing Density of Nonredundant Semiconductor Devices,” *Proceedings of the IRE*, vol. 50, n.º 3, pp. 286-298, 1962. doi: [10.1109/JRPROC.1962.288321](https://doi.org/10.1109/JRPROC.1962.288321).
- [5] P. Dodd y L. Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics,” *IEEE Transactions on Nuclear Science*, vol. 50, n.º 3, pp. 583-602, 2003. doi: [10.1109/TNS.2003.813129](https://doi.org/10.1109/TNS.2003.813129).
- [6] G. Swift y S. Guertin, “In-flight observations of multiple-bit upset in DRAMs,” *IEEE Transactions on Nuclear Science*, vol. 47, n.º 6, pp. 2386-2391, 2000. doi: [10.1109/23.903781](https://doi.org/10.1109/23.903781).
- [7] E. Monmasson y M. N. Cirstea, “FPGA Design Methodology for Industrial Control Systems—A Review,” *IEEE Transactions on Industrial Electronics*, vol. 54, n.º 4, pp. 1824-1842, 2007. doi: [10.1109/TIE.2007.898281](https://doi.org/10.1109/TIE.2007.898281).
- [8] UNE. “Asociación Española de Normalización,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://www.une.org/>.
- [9] ESA. “ECSS,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://ecss.nl/>.
- [10] *ECSS-E-ST-20-40C – ASIC, FPGA and IP Core engineering*, ECSS, oct. de 2023. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://ecss.nl/standard/ecss-e-st-20-40c-asic-fpga-and-ip-core-engineering-11-october-2023/>.
- [11] *ECSS-Q-ST-60-03C – ASIC, FPGA and IP Core product assurance*, ECSS, oct. de 2023. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://ecss.nl/standard/ecss-e-st-20-40c-asic-fpga-and-ip-core-engineering-11-october-2023/>.

- [12] *ECSS-E-ST-10-12C – Methods for the calculation of radiation received and its effects, and a policy for design margins +Corr.1*, ECSS, feb. de 2017. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://ecss.nl/standard/ecss-e-st-10-12c-methods-for-the-calculation-of-radiation-received-and-its-effects-and-a-policy-for-design-margins/>.
- [13] *ECSS-Q-ST-60-15C Rev.1 – Radiation hardness assurance*, ECSS, mar. de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://ecss.nl/standard/ecss-q-st-60-15c-rev-1-radiation-hardness-assurance-20-march-2025/>.
- [14] *IEEE 1076-2019 – IEEE Standard for VHDL Language Reference Manual*, IEEE, dic. de 2019. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://standards.ieee.org/ieee/1076/5179/>.
- [15] *IEEE 1800-2023 – IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language*, IEEE, feb. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://standards.ieee.org/ieee/1800/7743/>.
- [16] S. Malone, P. Saenz y P. Phelan, “RISC-V Processors for Spaceflight Embedded Platforms,” en *2023 IEEE Aerospace Conference*, 2023, pp. 1-11. doi: [10.1109/AERO55745.2023.10115850](https://doi.org/10.1109/AERO55745.2023.10115850).
- [17] D. P. Siewiorek y P. Narasimhan, “Fault-tolerant architectures for space and avionics applications,”
- [18] T. J. Bayer, “Planning for the Un-Plannable: Redundancy, Fault Protection, Contingency Planning and Anomaly Response for the Mars Reconnaissance Orbiter Mission,” en *AIAA SPACE 2007 Conference & Exposition*, Long Beach, California: American Institute of Aeronautics y Astronautics, sep. de 2007. doi: [10.2514/6.2007-6109](https://doi.org/10.2514/6.2007-6109).
- [19] M. N. Sweeting, “Modern Small Satellites-Changing the Economics of Space,” *Proceedings of the IEEE*, vol. 106, n.º 3, pp. 343-361, 2018. doi: [10.1109/JPROC.2018.2806218](https://doi.org/10.1109/JPROC.2018.2806218).
- [20] R. Le, “Soft error mitigation using prioritized essential bits,” 2012.
- [21] M. Berg et al., “Effectiveness of internal vs. external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis,” en *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, 2007, pp. 1-8. doi: [10.1109/RADECS.2007.5205603](https://doi.org/10.1109/RADECS.2007.5205603).
- [22] J. Wu, X. Meng y N. Zhang, “Fault-tolerant technology based on FPGA: A Research of LogiCORE™ IP Soft Error Mitigation Controller,” *Journal of Physics: Conference Series*, vol. 1486, n.º 5, p. 052 030, abr. de 2020. doi: [10.1088/1742-6596/1486/5/052030](https://doi.org/10.1088/1742-6596/1486/5/052030).

- [23] M. Caffrey y A. Salazar, “Correcting single-event upsets through Virtex partial configuration,” *Xilinx Application Notes*, 2000.
- [24] *Soft Error Mitigation Controller Product Guide (PG036)*, v4.1, AMD, nov. de 2023. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/pg036_sem/.
- [25] Subcomité JC-13.4, *Test Procedures for the Measurement of Single-Event Effects in Semiconductor Devices from Heavy Ion Irradiation*, JEDEC, 2017.
- [26] R. W. Hamming, “Error detecting and error correcting codes,” *The Bell System Technical Journal*, vol. 29, n.º 2, pp. 147-160, 1950. doi: [10.1002/j.1538-7305.1950.tb00463.x](https://doi.org/10.1002/j.1538-7305.1950.tb00463.x).
- [27] S. Velayudham, S. Rajagopal, Y. Venkata Ramana Rao y S.-B. Ko, “Power efficient error correction coding for on-chip interconnection links,” *IET Computers & Digital Techniques*, vol. 14, n.º 6, pp. 299-312, 2020. doi: <https://doi.org/10.1049/iet-cdt.2019.0082>.
- [28] Z.-L. Yang, X.-H. Wang, Z.-G. Zhang, J. Liu y H. Su, “Implementation and verification of different ECC mitigation designs for BRAMs in flash-based FPGAs,” *Chinese Physics C*, vol. 40, n.º 4, p. 046 103, abr. de 2016. doi: [10.1088/1674-1137/40/4/046103](https://doi.org/10.1088/1674-1137/40/4/046103).
- [29] R. Bose y D. Ray-Chaudhuri, “On a class of error correcting binary group codes,” *Information and Control*, vol. 3, n.º 1, pp. 68-79, 1960. doi: [https://doi.org/10.1016/S0019-9958\(60\)90287-4](https://doi.org/10.1016/S0019-9958(60)90287-4).
- [30] W. W. Peterson y D. T. Brown, “Cyclic Codes for Error Detection,” *Proceedings of the IRE*, vol. 49, n.º 1, pp. 228-235, 1961. doi: [10.1109/JRPROC.1961.287814](https://doi.org/10.1109/JRPROC.1961.287814).
- [31] M. L. Shooman, “N-Modular Redundancy,” en *Reliability of computer systems and networks*, Nashville, TN: John Wiley & Sons, ene. de 2002, pp. 145-201. doi: [10.1002/047122460x.ch4](https://doi.org/10.1002/047122460x.ch4).
- [32] V. Petrovic y M. Krstic, “Design Flow for Radhard TMR Flip-Flops,” en *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, 2015, pp. 203-208. doi: [10.1109/DDECS.2015.65](https://doi.org/10.1109/DDECS.2015.65).
- [33] M. A. Durivage, “The Reliability Bathtub Curve,” en *Practical engineering, process, and reliability statistics*, 2.ª ed., ASQ Quality Press, mar. de 2022, pp. 167-170.
- [34] *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA*, 2019, ver. 20250508, Unprivileged ISA Committee, mayo de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://github.com/riscv/riscv-isa-manual>.

- [35] G. Liu, E. Li, J. Huang y Z. Guo, “Moonix: An Educational Operating System on Nezha D1-H RISC-V Development Board,” en *Computer Science and Educational Informatization*, J. Gan, Y. Pan, J. Zhou, D. Liu, X. Song y Z. Lu, eds., Singapore: Springer Nature Singapore, 2024, pp. 12-25. doi: [10.1007/978-981-99-9492-2_2](https://doi.org/10.1007/978-981-99-9492-2_2).
- [36] D. Caballero. “Hazard Elimination on a RISC-V32i 5-stage Pipelined CPU,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://github.com/Dario-CP/ELEC-5140-project/blob/main/elec5140_project_report.pdf.
- [37] I. Wienand, “Computer Architecture,” en *Computer Science from the Bottom Up*, 2022, pp. 64-98.
- [38] J. Gaisler, *25 Years of SPARC*, 2017. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://indico.esa.int/event/182/contributions/1526/attachments/1400/1625/0905_-_Gaisler.pdf.
- [39] J. G. Tong, I. D. L. Anderson y M. A. S. Khalid, “Soft-Core Processors for Embedded Systems,” en *2006 International Conference on Microelectronics*, 2006, pp. 170-173. doi: [10.1109/ICM.2006.373294](https://doi.org/10.1109/ICM.2006.373294).
- [40] A. E. Wilson y M. Wirthlin, “Neutron Radiation Testing of Fault Tolerant RISC-V Soft Processor on Xilinx SRAM-based FPGAs,” en *2019 IEEE Space Computing Conference (SCC)*, 2019, pp. 25-32. doi: [10.1109/SpaceComp.2019.00008](https://doi.org/10.1109/SpaceComp.2019.00008).
- [41] F. Gaisler. “LEON5,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://www.gaisler.com/products/leon5>.
- [42] R. Weigand, *Open Instruction Set Architectures (ISA) in Space*, nov. de 2019. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <http://microelectronics.esa.int/papers/OpenSourceISA-RolandWeigandPub-2019-11-13.pdf>.
- [43] G. Furano y A. Menicucci, “Roadmap for On-Board Processing and Data Handling Systems in Space,” en *Dependable Multicore Architectures at Nanoscale*, M. Ottavi, D. Gizopoulos y S. Pontarelli, eds. Cham: Springer International Publishing, 2018, pp. 253-281. doi: [10.1007/978-3-319-54422-9_10](https://doi.org/10.1007/978-3-319-54422-9_10).
- [44] F. Gaisler. “NOEL-V,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://www.gaisler.com/products/noel-v>.
- [45] N.-J. Wessman, “NOEL-V RISC-V Processor latest development roadmap and applications,” en *3rd RISC-V Meeting*, 2021. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://open-src-soc.org/2021-03/media/slides/3rd-RISC-V-Meeting-2021-03-30-16h15-Nils-Johan-Wessman.pdf>.
- [46] *MicroBlaze Triple Modular Redundancy (TMR) Subsystem (PG268)*, v1.0, AMD, abr. de 2022. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/pg268-tmr/>.

- [47] *MicroBlaze Processor Reference Guide (UG984)*, 2024.2, AMD, nov. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/2024.2-English/ug984-vivado-microblaze-ref/>.
- [48] *MicroBlaze V Processor Reference Guide (UG1629)*, 2024.2, AMD, dic. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1629-microblaze-v-user-guide/>.
- [49] *PYNQ-Z1 Reference Manual*, Digilent, 2023. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://digilent.com/reference/programmable-logic/pynq-z1/reference-manual>.
- [50] *Zynq 7000 SoC Technical Reference Manual (UG585)*, v1.14, AMD, jun. de 2023. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/>.
- [51] *PYNQ: Python productivity for Adaptive Computing platforms*, a056b845, AMD, 2022. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://pynq.readthedocs.io/>.
- [52] *UltraFast Design Methodology Guide for FPGAs and SoCs (UG949)*, 2024.2, AMD, dic. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug949-vivado-design-methodology/>.
- [53] *PmodUSBUART™ Reference Manual (DOC#: 502-212)*, 2024.2, Digilent, abr. de 2015. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://digilent.com/reference/_media/pmod:pmodusuart_rm.pdf.
- [54] Cathal McCabe. “PYNQ-Z1 board files for Vivado,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://github.com/cathalmccabe/pynq-z1_board_files.
- [55] *Vivado Design Suite Properties Reference Guide (UG912)*, 2024.2, AMD, dic. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug912-vivado-properties/>.
- [56] Digilent. “PYNQ-Z1,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://digilent.com/reference/programmable-logic/pynq-z1/start>.
- [57] *Vivado Design Suite User Guide: Using Constraints (UG903)*, 2024.2, AMD, dic. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug903-vivado-using-constraints/>.
- [58] *PYNQ-Z1 Schematics (DOC#: 6003-500-017)*, Rev. F.0, Digilent, jul. de 2022. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://digilent.com/reference/_media/reference/programmable-logic/pynq-z1/pynq-z1_sch.pdf.

- [59] *Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)*, 2024.2, AMD, dic. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug906-vivado-design-analysis/>.
- [60] *RapidWright Documentation*, 2025.1.0-beta, RapidWright, jun. de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://www.rapidwright.io/docs/index.html>.
- [61] *Xilinx Large FPGA Methodology Guide (UG872)*, 2024.2, Xilinx, oct. de 2012. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://docs.amd.com/v/u/en-US/ug872_largefpga.
- [62] *Project X-Ray Documentation*, 0.0-3948-gc9f02d85, F4PGA, jun. de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://f4pga.readthedocs.io/projects/prjxray/en/latest/>.
- [63] *Vivado Design Suite User Guide: Implementation (UG904)*, 2024.2, AMD, nov. de 2024. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug904-vivado-implementation/>.
- [64] *Embedded Design Development Using Vitis User Guide (UG1701)*, 2024.2, AMD, ene. de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1701-vitis-accelerated-embedded/>.
- [65] *Vitis Reference Guide (UG1702)*, 2024.2, AMD, ene. de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1702-vitis-accelerated-reference/>.
- [66] *Vitis Unified Software Platform Documentation: Embedded Software Development (UG1400)*, 2024.2, AMD, ene. de 2025. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://docs.amd.com/r/en-US/ug1400-vitis-embedded/>.
- [67] *FPGA Assembly (FASM) documentation*, 0.0.2-100-gffafe82, F4PGA, 2022. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://fasm.readthedocs.io/>.
- [68] AMD. “Adaptive SoC & FPGA Support,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://adaptivesupport.amd.com/s/question/0D54U00007raoLMSAY/are-all-linear-frame-addresses-injectable-for-a-sem-in-testing-and-mitigation-mode>.
- [69] GanttProject. “Free desktop project management software,” Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <https://www.ganttpoint.biz/>.
- [70] *Single Event Upsets Simulation Tool Functional Description*, rev. 1, ESA, jul. de 2004. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: <http://microelectronics.esa.int/asic/SST-FunctionalDescription1-3.pdf>.

- [71] *CubeSat101 Basic Concepts and Processes for First-Time CubeSat Developers*, NASA, oct. de 2017. Acceso: 28 de ago. de 2025. [En línea]. Disponible en: https://www.nasa.gov/wp-content/uploads/2017/03/nasa_csl_i_cubesat_101_508.pdf.

ANEXO A. REQUISITOS DEL SISTEMA

Identificador: RF-00				
Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP			
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D			
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Texto principal	El sistema debe implementar triple redundancia modular (TMR).			
Notas				
Tests	Verificar el diseño de bloques. La implementación del TMR se verifica con el cumplimiento de los requisitos RF-10 y RF-11.			
Identificador: RF-01				
Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP			
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D			
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja			
Texto principal	El sistema debe implementar <i>scrubbing</i> de la memoria de configuración.			
Notas	El <i>scrubbing</i> debe realizar detección y corrección de errores.			
Tests	Verificar el diseño de bloques. La implementación del <i>scrubbing</i> de la memoria de configuración se verifica con el cumplimiento de los requisitos RF-06, RF-07, RF-08 y RF-09.			

Identificador: RF-02

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input type="checkbox"/> Alta <input checked="" type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input type="checkbox"/> Alta <input checked="" type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El sistema debe implementar corrección de errores en la memoria de bloques.
Notas	Memoria de bloques: BRAM.
Tests	Verificar el diseño de bloques. Inyectar un error en un bit de la memoria de bloques y comprobar que se corrige.

Identificador: RF-03

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El sistema debe permitir la comunicación bidireccional del subsistema TMR con un ordenador externo.
Notas	
Tests	Verificar el diseño de bloques. Establecer y realizar comunicaciones bidireccionales con el subsistema TMR.

Identificador: RF-04

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El sistema debe permitir la inyección de errores en la memoria de configuración desde un ordenador externo.
Notas	
Tests	Verificar el diseño de bloques. Inyectar errores en la memoria de configuración.

Identificador: RF-05

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input checked="" type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El sistema debe permitir la monitorización del estado del subsistema TMR desde un ordenador externo.
Notas	
Tests	Verificar el diseño de bloques. Inyectar errores en la memoria de configuración y observar el cambio del estado del subsistema TMR.

Identificador: RF-06

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El subsistema de <i>scrubbing</i> debe corregir todos los errores simples en la memoria de configuración.
Notas	Un error simple se produce cuando una partícula radiactiva afecta a un solo bit.
Tests	Verificar el diseño de bloques. Inyectar 20 errores simples en posiciones aleatorias y comprobar que todos se corrigen.

Identificador: RF-07

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El subsistema de <i>scrubbing</i> debe corregir todos los errores dobles adyacentes en la memoria de configuración.
Notas	Un error doble se produce cuando una partícula radiactiva afecta a dos bits.
Tests	Verificar el diseño de bloques. Inyectar 20 errores dobles adyacentes en posiciones aleatorias y comprobar que todos se corrigen.

Identificador: RF-08

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El subsistema de <i>scrubbing</i> debe corregir errores de múltiples bits cuando estos se distribuyen de forma unitaria en diferentes <i>frames</i> de la en la memoria de configuración.
Notas	Un error de múltiples bits se produce cuando una partícula radiactiva afecta a más de 2 bits.
Tests	Verificar el diseño de bloques. Inyectar 20 errores múltiples de un número de bits aleatorio entre 3 y 10, con cada bit en posiciones aleatorias de <i>frames</i> aleatorios pero diferentes entre sí y comprobar que todos se corrijen.

Identificador: RF-09

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El subsistema de <i>scrubbing</i> debe corregir errores de múltiples bits cuando estos se distribuyan en parejas adyacentes en diferentes <i>frames</i> de la en la memoria de configuración.
Notas	Un error de múltiples bits se produce cuando una partícula radiactiva afecta a más de 2 bits.
Tests	Verificar el diseño de bloques. Inyectar 20 errores múltiples de un número de parejas de bits aleatorio entre 3 y 10, con cada pareja adyacente en posiciones aleatorias de <i>frames</i> aleatorios pero diferentes entre sí y comprobar que todos se corrijen.

Identificador: RF-10

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El subsistema TMR, ante un error en uno y solo uno de sus sub-bloques, debe mantener en funcionamiento los dos restantes, enmascarando las salida del sub-bloque afectado.
Notas	
Tests	Verificar el diseño de bloques. Inyectar errores en bits que configuran BELs asociados a <i>cells</i> primitivas de uno de los sub-bloques hasta desencadenar su la incoherencia y comprobar que el sistema sigue en pie y funcional. Realizar 20 pruebas para cada sub-bloque.

Identificador: RF-11

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El subsistema TMR, ante un error en más de uno de sus sub-bloques, debe detenerse.
Notas	
Tests	Verificar el diseño de bloques. Inyectar errores en bits que configuran BELs asociados a <i>cells</i> primitivas de dos de los sub-bloques hasta desencadenar su incoherencia y comprobar que el sistema se para. Realizar 10 pruebas para cada pareja ordenada de sub-bloques.

Identificador: RN-01				
Dispositivo	<input type="checkbox"/> D-ASIC	<input type="checkbox"/> A-ASIC	<input type="checkbox"/> FPGA	<input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A	<input type="checkbox"/> B	<input type="checkbox"/> C	<input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Claridad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Verificabilidad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Estabilidad	<input type="checkbox"/> Alta	<input checked="" type="checkbox"/> Media	<input type="checkbox"/> Baja	
Texto principal	El sistema debe configurarse en el SoC ZYNQ XC7Z020-1CLG400C.			
Notas				
Tests	Verificar el dispositivo utilizado y el uso de sus recursos.			

Identificador: RN-02				
Dispositivo	<input type="checkbox"/> D-ASIC	<input type="checkbox"/> A-ASIC	<input type="checkbox"/> FPGA	<input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A	<input type="checkbox"/> B	<input type="checkbox"/> C	<input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Claridad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Verificabilidad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Estabilidad	<input type="checkbox"/> Alta	<input checked="" type="checkbox"/> Media	<input type="checkbox"/> Baja	
Texto principal	El sistema se debe basar en el procesador <i>softcore</i> MicroBlaze-V.			
Notas				
Tests	Verificar el diseño de bloques.			

Identificador: RN-03				
Dispositivo	<input type="checkbox"/> D-ASIC	<input type="checkbox"/> A-ASIC	<input checked="" type="checkbox"/> FPGA	<input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A	<input type="checkbox"/> B	<input type="checkbox"/> C	<input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Claridad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Verificabilidad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Estabilidad	<input checked="" type="checkbox"/> Alta	<input type="checkbox"/> Media	<input type="checkbox"/> Baja	
Texto principal	El diseño, la síntesis y la implementación del sistema deben realizarse en Vivado.			
Notas				
Tests	No hay tests. Simplemente se utilizará Vivado para las fases de diseño, síntesis e la implementación.			

Identificador: RN-04

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El <i>debugging</i> del sistema debe realizarse en Vitis.
Notas	
Tests	No hay tests. Simplemente se utilizará Vitis para la fase de <i>debugging</i> .

Identificador: RN-05

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	La comunicación bidireccional del subsistema TMR con el ordenador externo debe implementarse mediante el protocolo UART.
Notas	
Tests	Verificar el diseño de bloques. Establecer y realizar comunicaciones bidireccionales con el subsistema TMR utilizando una interfaz USBUART.

Identificador: RN-06

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	La inyección de errores en la memoria de configuración desde un ordenador externo debe implementarse mediante el protocolo UART.
Notas	
Tests	Verificar el diseño de bloques. Inyectar errores en la memoria de configuración utilizando una interfaz USBUART.

Identificador: RN-07

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input checked="" type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	La monitorización del estado del subsistema TMR desde un ordenador externo debe implementarse mediante el protocolo UART.
Notas	
Tests	Verificar el diseño de bloques. Inyectar errores en la memoria de configuración y observar el cambio del estado del subsistema TMR mediante una interfaz USBUART.

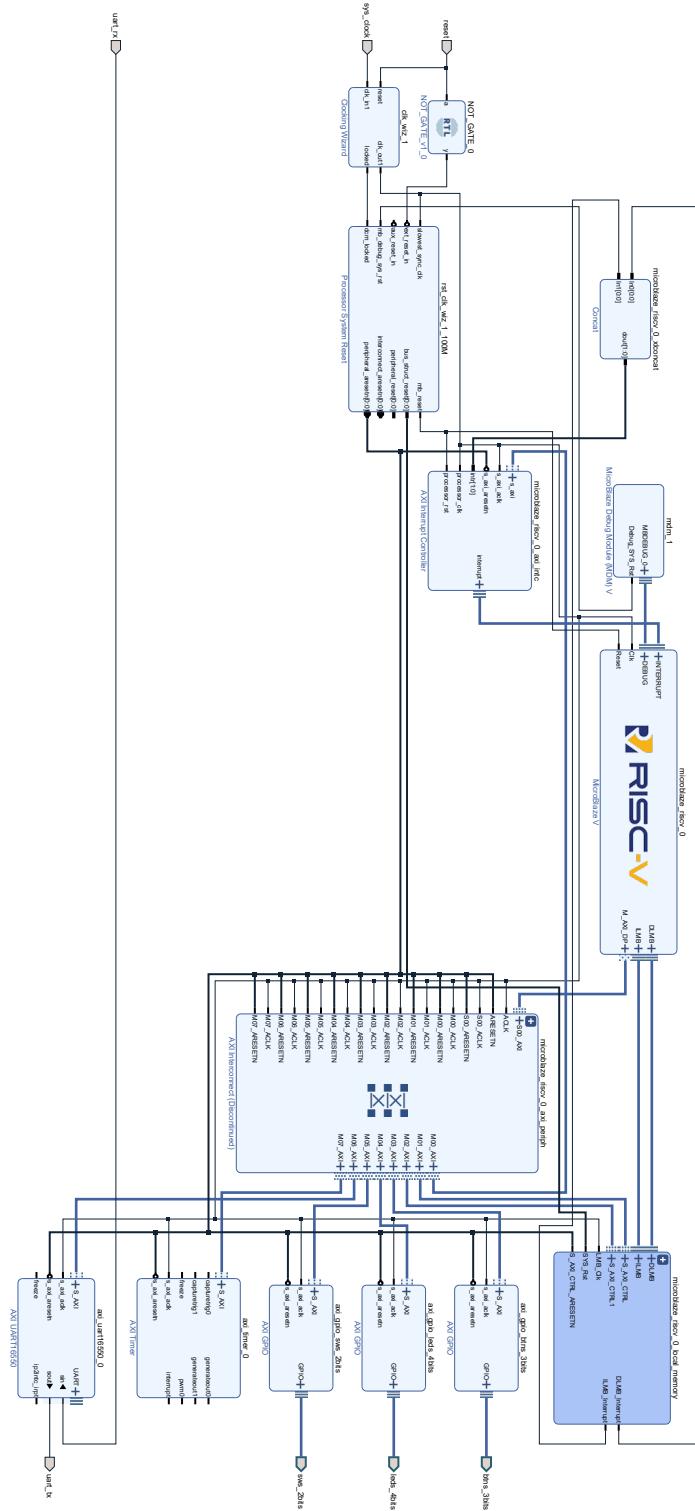
Identificador: RN-08

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El diseño debe superar los requisitos temporales de Vivado, con un Worst Negative Slack y un Total Negative Slack superiores a -1 ns.
Notas	
Tests	Generar el informe temporal y comprobar que cumple sus requisitos, con un Worst Negative Slack y un Total Negative Slack superiores a -1 ns.

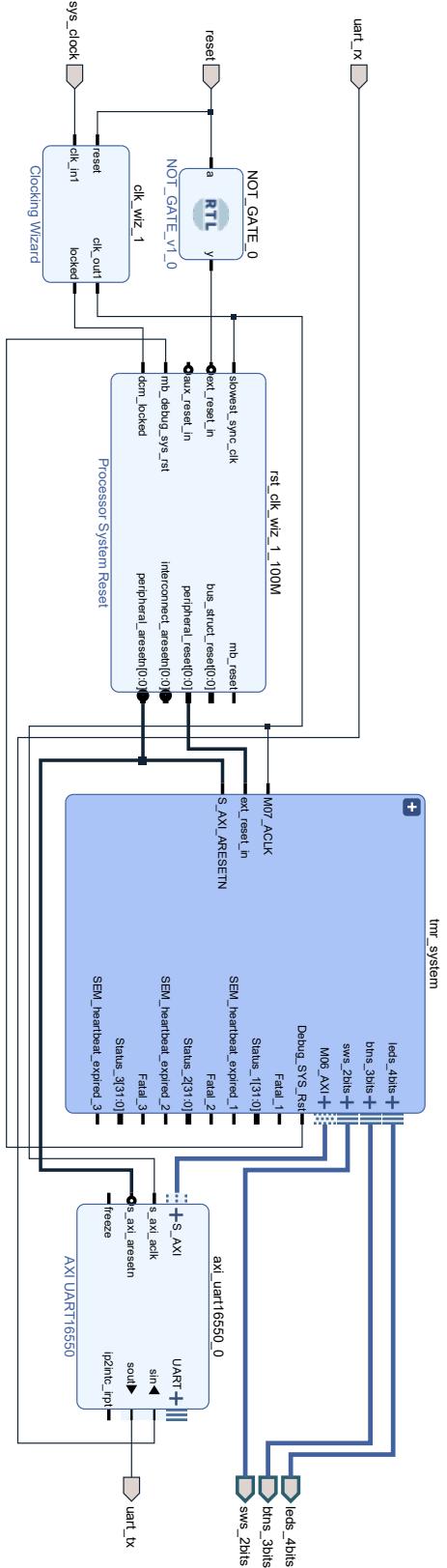
Identificador: RN-08

Dispositivo	<input type="checkbox"/> D-ASIC <input type="checkbox"/> A-ASIC <input checked="" type="checkbox"/> FPGA <input checked="" type="checkbox"/> IP
Criticidad	<input checked="" type="checkbox"/> A <input type="checkbox"/> B <input type="checkbox"/> C <input type="checkbox"/> D
Prioridad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Claridad	<input type="checkbox"/> Alta <input checked="" type="checkbox"/> Media <input type="checkbox"/> Baja
Verificabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Estabilidad	<input checked="" type="checkbox"/> Alta <input type="checkbox"/> Media <input type="checkbox"/> Baja
Texto principal	El diseño debe superar los requisitos energéticos de Vivado.
Notas	
Tests	Generar el informe energético en Vivado y comprobar que cumple sus requisitos.

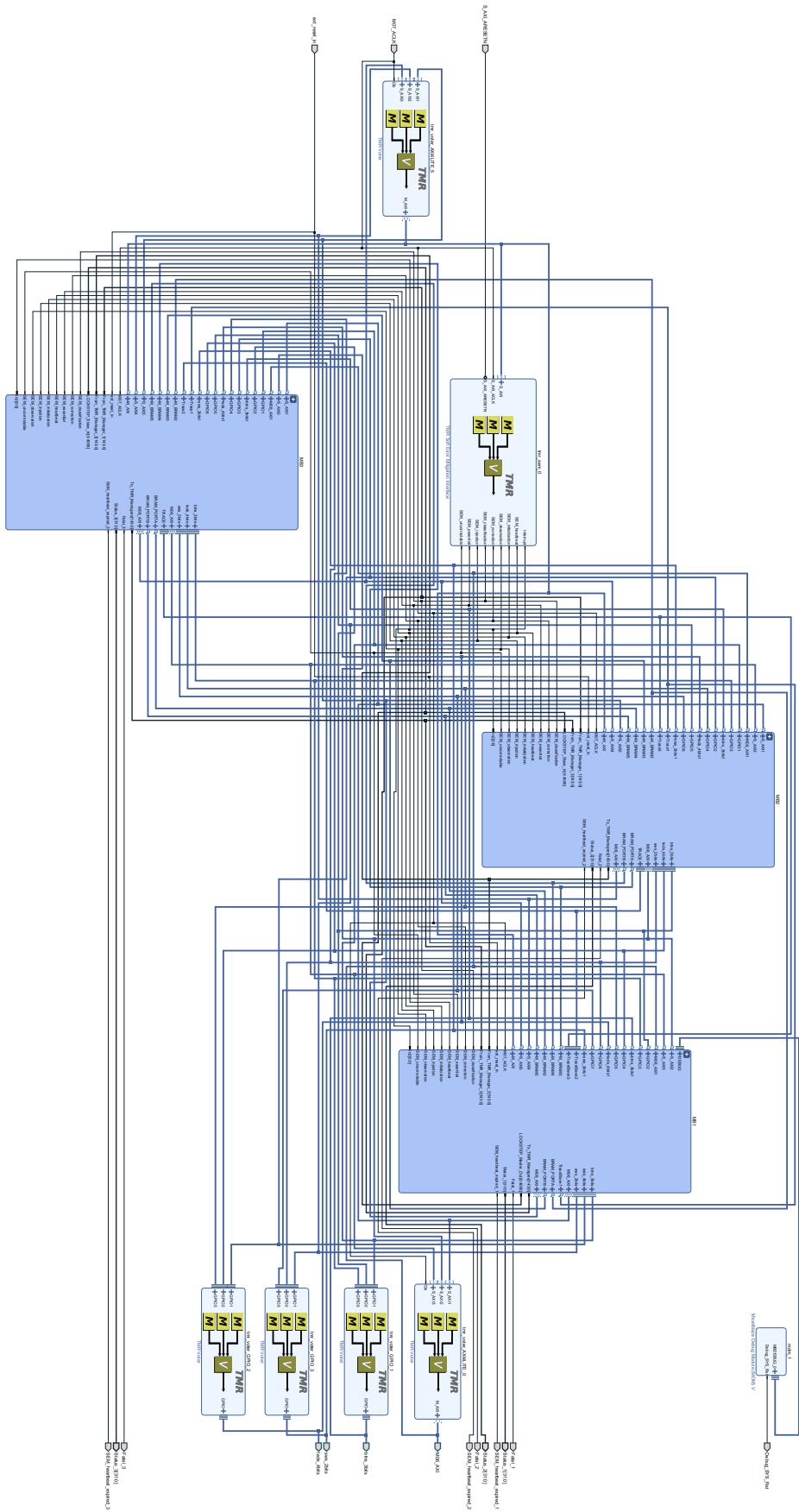
ANEXO B. DIAGRAMAS Y ESQUEMAS DEL SISTEMA



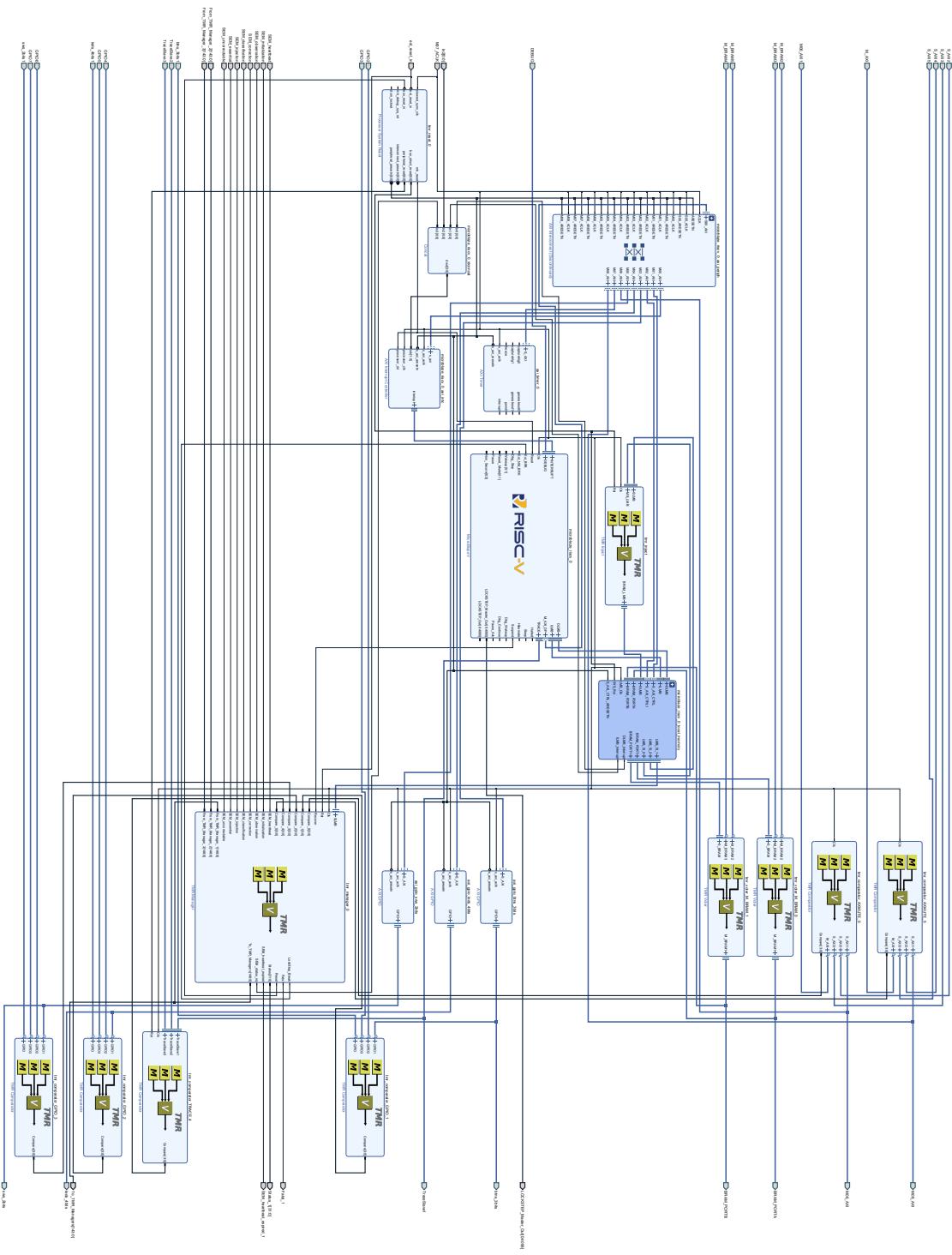
1. Diagrama de bloques del diseño no tolerante a fallos.



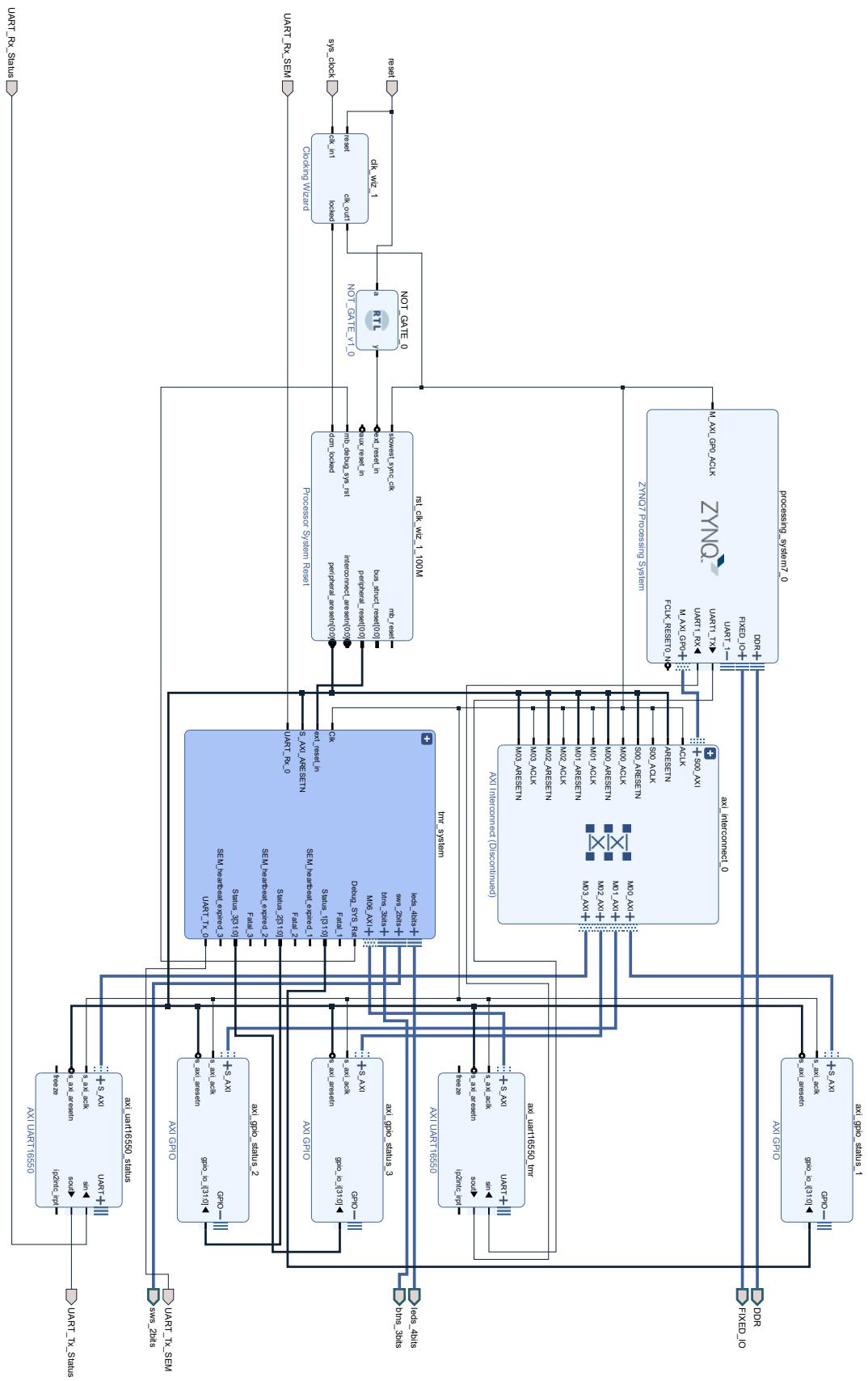
2. Diagrama de bloques del diseño tolerante a fallos preliminar.



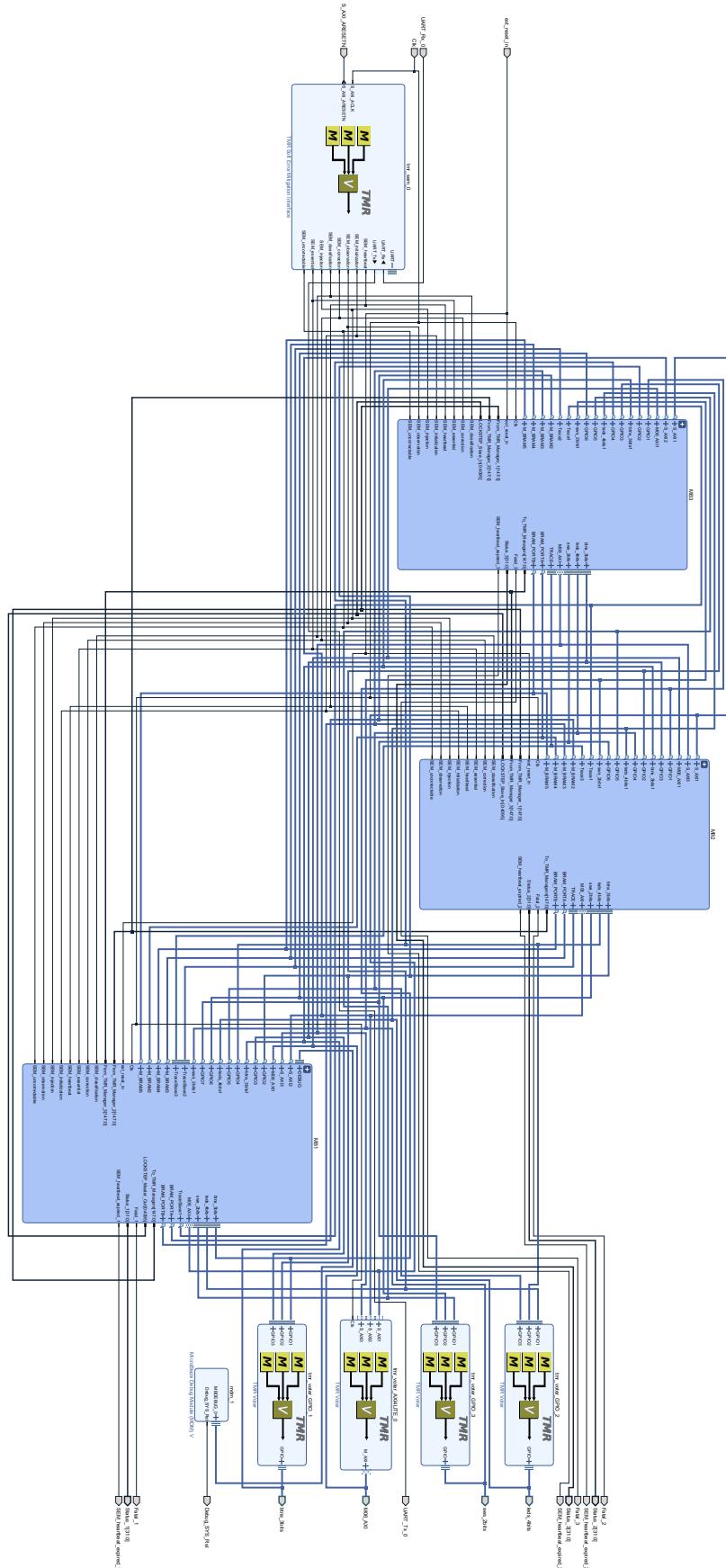
3. Diagrama de bloques del subsistema TMR preliminar.



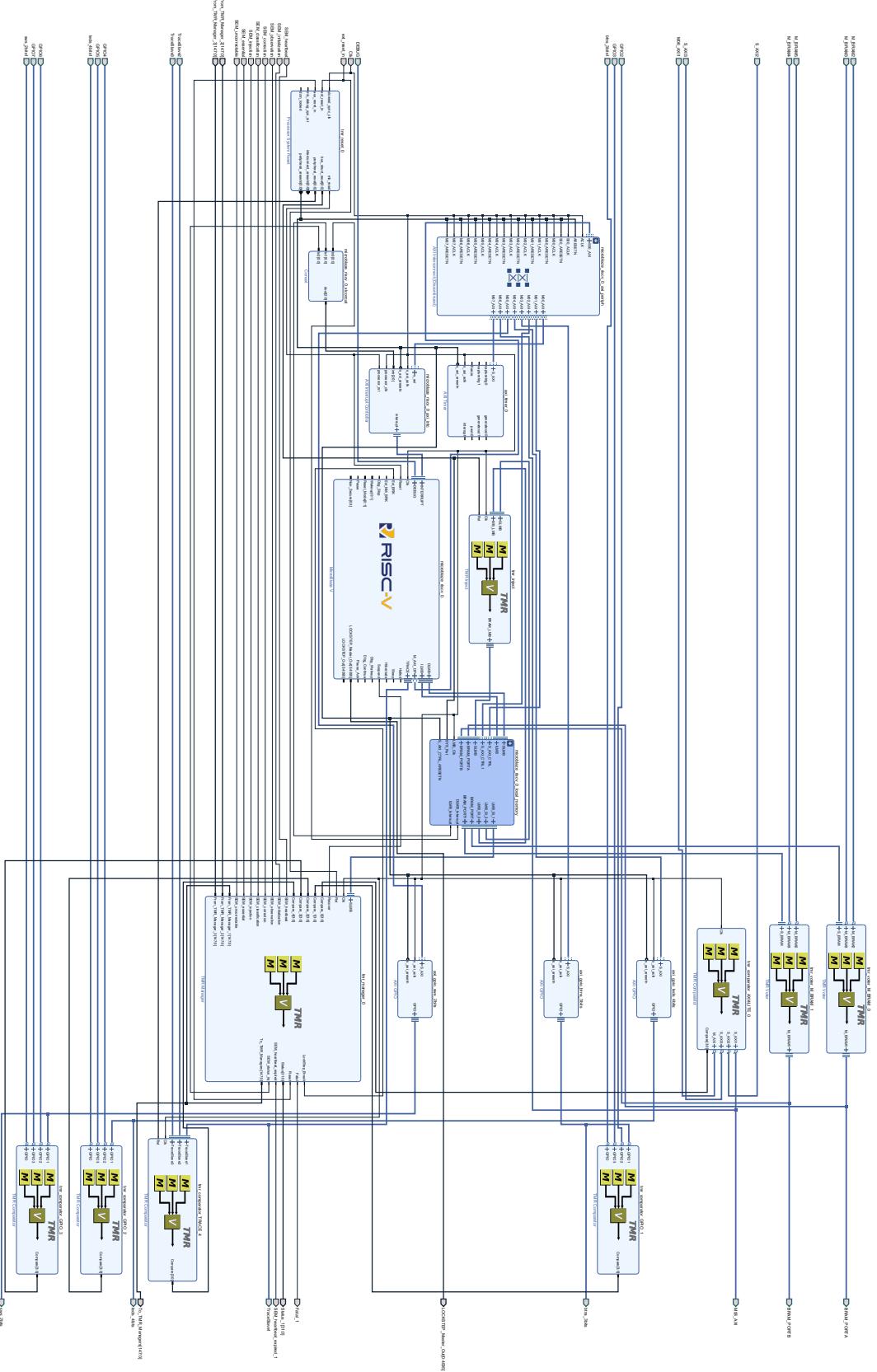
4. Diagrama de bloques del sub-bloque MB1 preliminar.



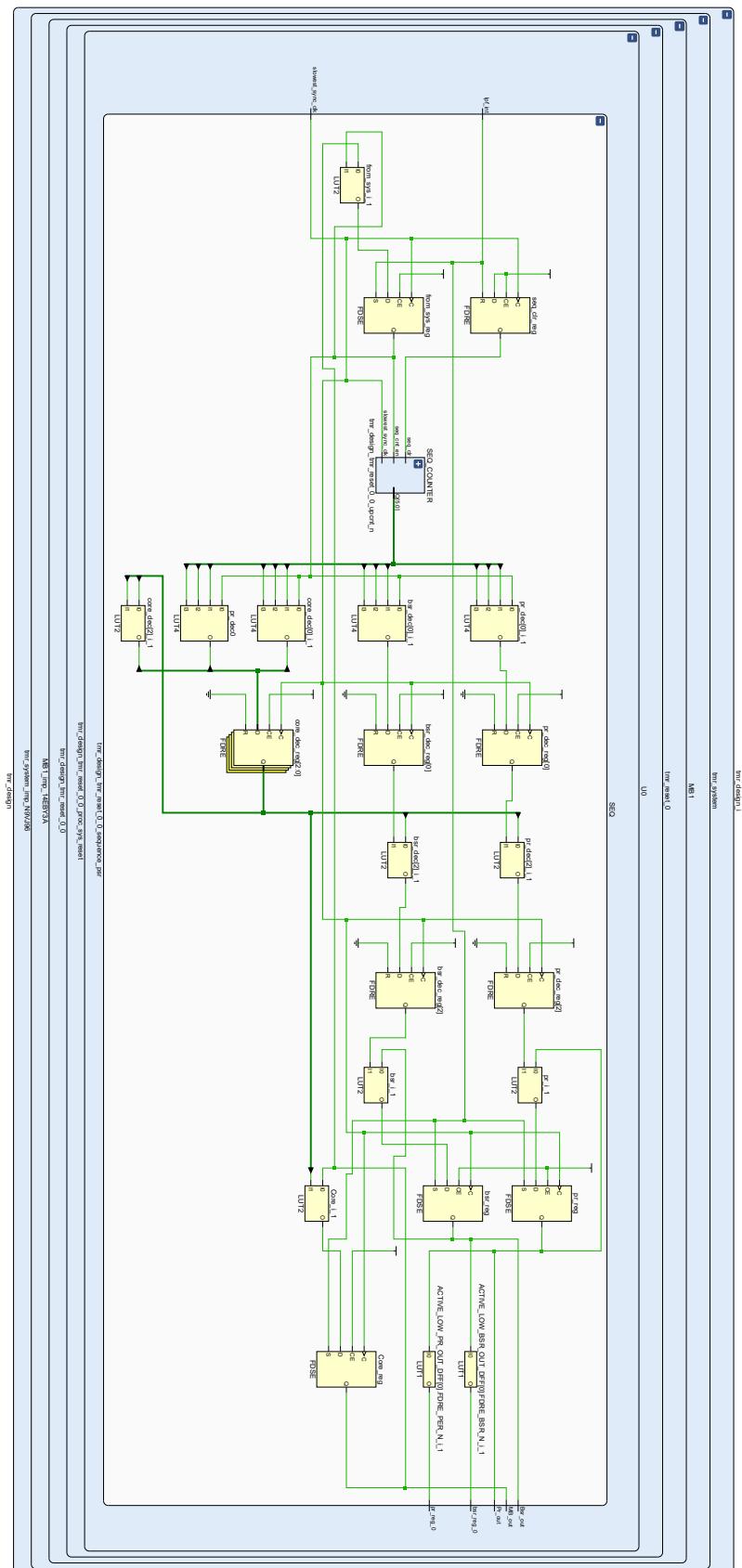
5. Diagrama de bloques del diseño tolerante a fallos definitivo.



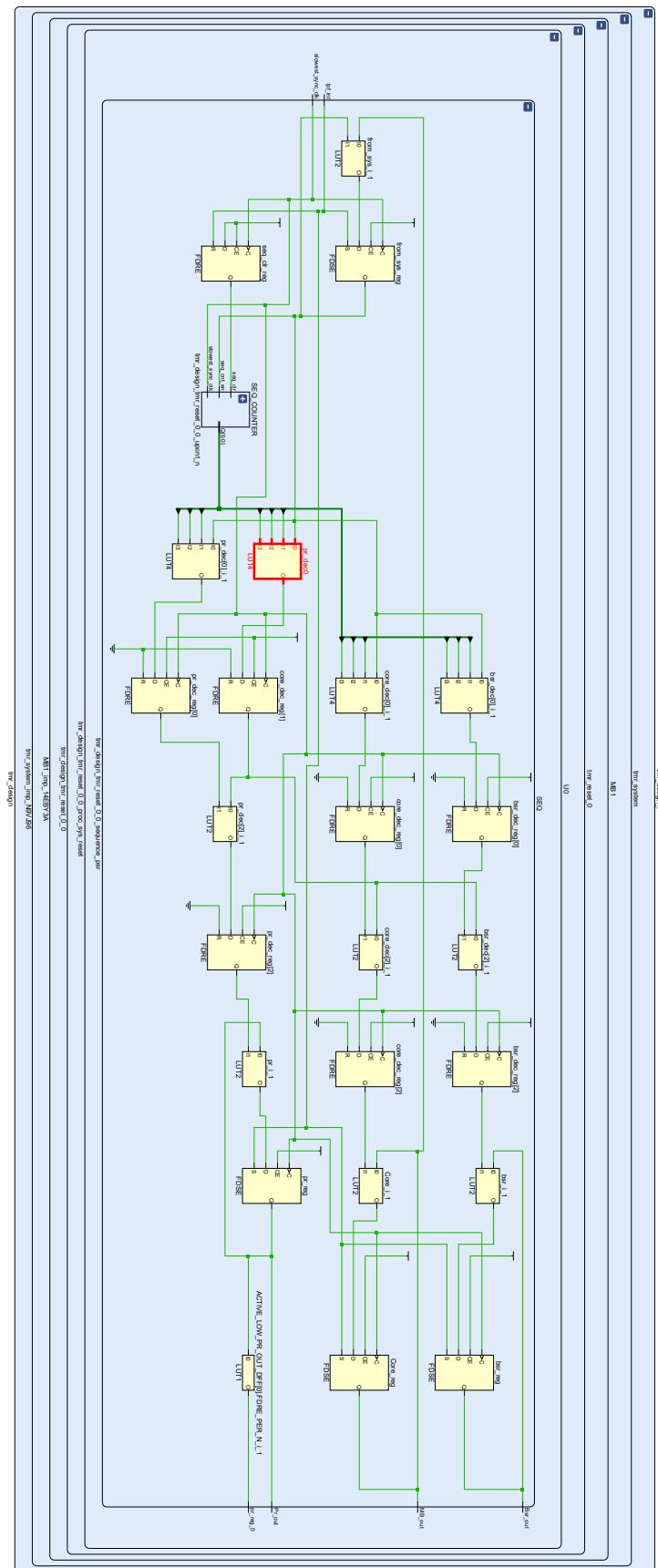
6. Diagrama de bloques del subsistema TMR definitivo.



7. Diagrama de bloques del sub-bloque MB1 definitivo.



8. Esquema lógico de la *cell jerárquica*
`tmr_design_i/tmr_system/MB1/tmr_reset_0/U0/SEQ`.



9. Esquema lógico-físico de la *cell jerárquica tmr_design_i/tmr_system/MB1/tmr_reset_0/U0/SEQ*.

ANEXO C. DIRECCIONES DE MEMORIA DEL DISEÑO TOLERANTE A FALLOS DEFINITIVO

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address	Slave Base Address	Slave High Address	Lock
Network_0								
/processing_system7_0								
/processing_system7_0/Data (32 address bits : 0x40000000 [16])								
/axi_BPIO_Status_2/S_AXI	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF	0x0	0x1FF	none
/axi_BPIO_Status_3/S_AXI	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF	0x0	0x1FF	none
/axi_Uart16550_Status/S_AXI	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF	0x0	0x1FF	none
Network_1								
microblaze_riscv_0								
/tim_System/MB3/microblaze_riscv_0/Data (32 address bits : 46)								
/tim_System/MB3/microblaze_riscv_0/local_memory/11mb_bram_if_cntlr/S_AXI_CTRL	S_AXI_CTRL	Reg	0x7601_0000	64K	0x7601_FFFF	0x0	0x1FF	none
/tim_System/MB3/axi_gpio_0_btris_3bits_S_AXI	S_AXI_CTRL	Reg	0x4000_0000	64K	0x4000_FFFF	0x0	0x1FF	none
/axi_Uart16550_tmr/S_AXI	S_AXI_CTRL	Reg	0x7600_0000	64K	0x7600_FFFF	0x0	0x1FF	none
/tim_System/MB3/axi_gpio_sws_2bits/S_AXI	S_AXI_CTRL	Reg	0x4440_0000	64K	0x4440_FFFF	0x0	0x1FF	none
/tim_System/MB3/axi_gpio_leds_4bits/S_AXI	S_AXI_CTRL	Reg	0x4002_0000	64K	0x4002_FFFF	0x0	0x1FF	none
/tim_System/MB3/microblaze_riscv_0_axi_intc/S_AXI	S_AXI_CTRL	Reg	0x4001_0000	64K	0x4001_FFFF	0x0	0x1FF	none
/tim_System/MB3/axi_timer_0/S_AXI	S_AXI_CTRL	Reg	0x4120_0000	64K	0x4120_FFFF	0x0	0x1FF	none
/tim_System/MB3/tmr_manager_0/S_LMB	S_AXI_CTRL	Reg	0x41C0_0000	64K	0x41C0_FFFF	0x0	0x1FF	none
/tim_System/MB3/microblaze_riscv_0_local_memory/11mb_bram_if_cntlr/S_AXI_CTRL	S_AXI_CTRL	Reg	0x4442_0000	64K	0x4442_FFFF	0x0	0x1FF	none
/tim_System/MB2/axi_gpio_sws_2bits/S_AXI	S_AXI_CTRL	Reg	0x4000_0000	64K	0x4000_FFFF	0x0	0x1FF	none
/tim_System/MB2/axi_gpio_leds_4bits/S_AXI	S_AXI_CTRL	Reg	0x4002_0000	64K	0x4002_FFFF	0x0	0x1FF	none
/tim_System/MB2/axi_timer_0/S_AXI	S_AXI_CTRL	Reg	0x4001_0000	64K	0x4001_FFFF	0x0	0x1FF	none
/tim_System/MB2/microblaze_riscv_0_axi_intc/S_AXI	S_AXI_CTRL	Reg	0x4120_0000	64K	0x4120_FFFF	0x0	0x1FF	none
/tim_System/MB2/tmr_manager_0/S_LMB	S_AXI_CTRL	Reg	0x4441_0000	64K	0x4441_FFFF	0x0	0x1FF	none
/tim_System/MB2/microblaze_riscv_0_axi_intc/S_AXI	S_AXI_CTRL	Reg	0x4442_0000	64K	0x4442_FFFF	0x0	0x1FF	none
/tim_System/MB2/microblaze_riscv_0_axi_intc/S_AXI	S_AXI_CTRL	Reg	0x4120_0000	64K	0x4120_FFFF	0x0	0x1FF	none
/tim_System/MB2/microblaze_riscv_0_axi_intc/S_AXI	S_AXI_CTRL	Reg	0x4001_0000	64K	0x4001_FFFF	0x0	0x1FF	none
/tim_System/MB1/axi_gpio_btris_3bits/S_AXI	S_AXI_CTRL	Reg	0x7600_0000	64K	0x7600_FFFF	0x0	0x1FF	none
/tim_System/MB1/axi_gpio_sws_2bits/S_AXI	S_AXI_CTRL	Reg	0x4120_0000	64K	0x4120_FFFF	0x0	0x1FF	none
/tim_System/MB1/axi_gpio_leds_4bits/S_AXI	S_AXI_CTRL	Reg	0x4000_0000	64K	0x4000_FFFF	0x0	0x1FF	none
/tim_System/MB1/axi_timer_0/S_AXI	S_AXI_CTRL	Reg	0x7600_0000	64K	0x7600_FFFF	0x0	0x1FF	none
/tim_System/MB1/microblaze_riscv_0_local_memory/11mb_bram_if_cntlr/S_AXI_CTRL	S_AXI_CTRL	Reg	0x41C0_0000	64K	0x41C0_FFFF	0x0	0x1FF	none
Network_2								
microblaze_riscv_0								
/tim_System/MB1/microblaze_riscv_0/Instruction (32 address bits : 46)								
/tim_System/MB1/microblaze_riscv_0_local_memory/11mb_bram_if_cntlr/S_LMB	S_LMB	Mem	0x0	32K	0x7FFF	0x0	0x7FFF	none
Network_3								
microblaze_riscv_0								
/tim_System/MB2/microblaze_riscv_0/Instruction (32 address bits : 46)								
/tim_System/MB2/microblaze_riscv_0_local_memory/11mb_bram_if_cntlr/S_LMB	S_LMB	Mem	0x0	32K	0x7FFF	0x0	0x7FFF	none
Network_4								
microblaze_riscv_0								
/tim_System/MB3/microblaze_riscv_0/Instruction (32 address bits : 46)								
/tim_System/MB3/microblaze_riscv_0_local_memory/11mb_bram_if_cntlr/S_LMB	S_LMB	Mem	0x0	32K	0x7FFF	0x0	0x7FFF	none

ANEXO D. ARCHIVO DE RESTRICCIONES XDC

```
1  ## This file is a general .xdc for the PYNQ-Z1 board Rev. C
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used ports (in each line, after get_ports) according to the top level
5  ##     signal names in the project
6
7  ## Clock signal 125 MHz
8
9  set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports sys_clock]
10 #create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { sysclk }];
11
12 ##Switches
13
14 set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {sws_2bits_tri_i[0]}]
15 set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {sws_2bits_tri_i[1]}]
16
17 ##RGB LEDs
18
19 #set_property -dict { PACKAGE_PIN L15      IOSTANDARD LVCMOS33 } [get_ports { led4_b }]; #
20     IO_L22N_T3_AD7N_35 Sch=led4_b
21 #set_property -dict { PACKAGE_PIN G17      IOSTANDARD LVCMOS33 } [get_ports { led4_g }]; #
22     IO_L16P_T2_35 Sch=led4_g
23 #set_property -dict { PACKAGE_PIN N15      IOSTANDARD LVCMOS33 } [get_ports { led4_r }]; #
24     IO_L21P_T3_DQS_AD14P_35 Sch=led4_r
25 #set_property -dict { PACKAGE_PIN G14      IOSTANDARD LVCMOS33 } [get_ports { led5_b }]; #
26     IO_0_35 Sch=led5_b
27 #set_property -dict { PACKAGE_PIN L14      IOSTANDARD LVCMOS33 } [get_ports { led5_g }]; #
28     IO_L22P_T3_AD7P_35 Sch=led5_g
29 #set_property -dict { PACKAGE_PIN M15      IOSTANDARD LVCMOS33 } [get_ports { led5_r }]; #
30     IO_L23N_T3_35 Sch=led5_r
31
32 ##LEDs
33
34 set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[0]}]
35 set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[1]}]
36 set_property -dict {PACKAGE_PIN N16 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[2]}]
37 set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports {leds_4bits_tri_o[3]}]
38
39 ##Buttons
40
41 set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports reset]
42 set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports {btms_3bits_tri_i[0]}]
43 set_property -dict {PACKAGE_PIN L20 IOSTANDARD LVCMOS33} [get_ports {btms_3bits_tri_i[1]}]
44 set_property -dict {PACKAGE_PIN L19 IOSTANDARD LVCMOS33} [get_ports {btms_3bits_tri_i[2]}]
45
46 ##Pmod Header JA
47
48 #set_property -dict { PACKAGE_PIN Y18      IOSTANDARD LVCMOS33 } [get_ports { ja[0] }]; #
49     IO_L17P_T2_34 Sch=ja_p[1]
50 set_property -dict {PACKAGE_PIN Y19 IOSTANDARD LVCMOS33} [get_ports UART_Tx_SEM]
51 set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports UART_Rx_SEM]
52 #set_property -dict { PACKAGE_PIN Y17      IOSTANDARD LVCMOS33 } [get_ports { ja[3] }]; #
53     IO_L7N_T1_34 Sch=ja_n[2]
54 #set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 } [get_ports { ja[4] }]; #
55     IO_L12P_T1_MRCC_34 Sch=ja_p[3]
56 #set_property -dict { PACKAGE_PIN U19      IOSTANDARD LVCMOS33 } [get_ports { ja[5] }]; #
57     IO_L12N_T1_MRCC_34 Sch=ja_n[3]
58 #set_property -dict { PACKAGE_PIN W18      IOSTANDARD LVCMOS33 } [get_ports { ja[6] }]; #
59     IO_L22P_T3_34 Sch=ja_p[4]
60 #set_property -dict { PACKAGE_PIN W19      IOSTANDARD LVCMOS33 } [get_ports { ja[7] }]; #
61     IO_L22N_T3_34 Sch=ja_n[4]
```

```

49
50 ##Pmod Header JB
51
52 #set_property -dict { PACKAGE_PIN W14 IOSTANDARD LVCMOS33 } [get_ports { jb[0] }]; #
53     IO_L8P_T1_34 Sch=jb_p[1]
54 set_property -dict {PACKAGE_PIN Y14 IOSTANDARD LVCMOS33} [get_ports UART_Tx_Status]
55 set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports UART_Rx_Status]
56 #set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { jb[3] }]; #
57     IO_L1N_T0_34 Sch=jb_n[2]
58 #set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { jb[4] }]; #
59     IO_L18P_T2_34 Sch=jb_p[3]
60 #set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 } [get_ports { jb[5] }]; #
61     IO_L18N_T2_34 Sch=jb_n[3]
62 #set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { jb[6] }]; #
63     IO_L4P_T0_34 Sch=jb_p[4]
64 #set_property -dict { PACKAGE_PIN W13 IOSTANDARD LVCMOS33 } [get_ports { jb[7] }]; #
65     IO_L4N_T0_34 Sch=jb_n[4]
66
67 ##Audio Out
68
69 #set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { aud_pwm }]; #
70     IO_L20N_T3_34 Sch=aud_pwm
71 #set_property -dict { PACKAGE_PIN T17 IOSTANDARD LVCMOS33 } [get_ports { aud_sd }]; #
72     IO_L20P_T3_34 Sch=aud_sd
73
74 ##Mic input
75
76 #set_property -dict { PACKAGE_PIN F17 IOSTANDARD LVCMOS33 } [get_ports { m_clk }]; #
77     IO_L6N_T0_VREF_35 Sch=m_clk
78 #set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { m_data }]; #
79     IO_L16N_T2_35 Sch=m_data
80
81 ##ChipKit Single Ended Analog Inputs
82 ##NOTE: The ck_an_p pins can be used as single ended analog inputs with voltages from
83     0-3.3V (Chipkit Analog pins A0-A5).
84 ##      These signals should only be connected to the XADC core. When using these pins as
85      digital I/O, use pins ck_io[14-19].
86
87 #set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { ck_an_n[0] }];
88     #IO_L3N_T0_DQS_AD1N_35 Sch=ck_an_n[0]
89 #set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports { ck_an_p[0] }];
90     #IO_L3P_T0_DQS_AD1P_35 Sch=ck_an_p[0]
91 #set_property -dict { PACKAGE_PIN E19 IOSTANDARD LVCMOS33 } [get_ports { ck_an_n[1] }];
92     #IO_L5N_T0_AD9N_35 Sch=ck_an_n[1]
93 #set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { ck_an_p[1] }];
94     #IO_L5P_T0_AD9P_35 Sch=ck_an_p[1]
95 #set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { ck_an_n[2] }];
96     #IO_L20N_T3_AD6N_35 Sch=ck_an_n[2]
97 #set_property -dict { PACKAGE_PIN K14 IOSTANDARD LVCMOS33 } [get_ports { ck_an_p[2] }];
98     #IO_L20P_T3_AD6P_35 Sch=ck_an_p[2]
99 #set_property -dict { PACKAGE_PIN J16 IOSTANDARD LVCMOS33 } [get_ports { ck_an_n[3] }];
100    #IO_L24N_T3_AD15N_35 Sch=ck_an_n[3]
101 #set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { ck_an_p[3] }];
102    #IO_L24P_T3_AD15P_35 Sch=ck_an_p[3]
103 #set_property -dict { PACKAGE_PIN H20 IOSTANDARD LVCMOS33 } [get_ports { ck_an_n[4] }];
104    #IO_L17N_T2_AD5N_35 Sch=ck_an_n[4]
105 #set_property -dict { PACKAGE_PIN J20 IOSTANDARD LVCMOS33 } [get_ports { ck_an_p[4] }];
106    #IO_L17P_T2_AD5P_35 Sch=ck_an_p[4]
107 #set_property -dict { PACKAGE_PIN G20 IOSTANDARD LVCMOS33 } [get_ports { ck_an_n[5] }];
108    #IO_L18N_T2_AD13N_35 Sch=ck_an_n[5]
109 #set_property -dict { PACKAGE_PIN G19 IOSTANDARD LVCMOS33 } [get_ports { ck_an_p[5] }];
110    #IO_L18P_T2_AD13P_35 Sch=ck_an_p[5]
111
112 ##ChipKit Digital I/O Low
113
114 #set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { ck_io[0] }];
115     IO_L5P_T0_34 Sch=ck_io[0]

```

```

91 #set_property -dict { PACKAGE_PIN U12    IOSTANDARD LVCMOS33 } [get_ports { ck_io[1] }]; #
92     IO_L2N_T0_34 Sch=ck_io[1]
93 #set_property -dict { PACKAGE_PIN U13    IOSTANDARD LVCMOS33 } [get_ports { ck_io[2] }]; #
94     IO_L3P_T0_DQS_PUDC_B_34 Sch=ck_io[2]
95 #set_property -dict { PACKAGE_PIN V13    IOSTANDARD LVCMOS33 } [get_ports { ck_io[3] }]; #
96     IO_L3N_T0_DQS_34 Sch=ck_io[3]
97 #set_property -dict { PACKAGE_PIN V15    IOSTANDARD LVCMOS33 } [get_ports { ck_io[4] }]; #
98     IO_L10P_T1_34 Sch=ck_io[4]
99 #set_property -dict { PACKAGE_PIN T15    IOSTANDARD LVCMOS33 } [get_ports { ck_io[5] }]; #
100    IO_L5N_T0_34 Sch=ck_io[5]
101 #set_property -dict { PACKAGE_PIN R16    IOSTANDARD LVCMOS33 } [get_ports { ck_io[6] }]; #
102    IO_L19P_T3_34 Sch=ck_io[6]
103 #set_property -dict { PACKAGE_PIN U17    IOSTANDARD LVCMOS33 } [get_ports { ck_io[7] }]; #
104    IO_L9N_T1_DQS_34 Sch=ck_io[7]
105 #set_property -dict { PACKAGE_PIN V17    IOSTANDARD LVCMOS33 } [get_ports { ck_io[8] }]; #
106    IO_L21P_T3_DQS_34 Sch=ck_io[8]
107 #set_property -dict { PACKAGE_PIN V18    IOSTANDARD LVCMOS33 } [get_ports { ck_io[9] }]; #
108    IO_L21N_T3_DQS_34 Sch=ck_io[9]
109 #set_property -dict { PACKAGE_PIN T16    IOSTANDARD LVCMOS33 } [get_ports { ck_io[10] }]; #
110    IO_L9P_T1_DQS_34 Sch=ck_io[10]
111 #set_property -dict { PACKAGE_PIN R17    IOSTANDARD LVCMOS33 } [get_ports { ck_io[11] }]; #
112    IO_L19N_T3_VREF_34 Sch=ck_io[11]
113 #set_property -dict { PACKAGE_PIN P18    IOSTANDARD LVCMOS33 } [get_ports { ck_io[12] }]; #
114    IO_L23N_T3_34 Sch=ck_io[12]
115 #set_property -dict { PACKAGE_PIN N17    IOSTANDARD LVCMOS33 } [get_ports { ck_io[13] }]; #
116    IO_L23P_T3_34 Sch=ck_io[13]

117 ##ChipKit Digital I/O On Outer Analog Header
118 ##NOTE: These pins should be used when using the analog header signals A0-A5 as digital I/
119     O (Chipkit digital pins 14-19)
120
121 #set_property -dict { PACKAGE_PIN Y11    IOSTANDARD LVCMOS33 } [get_ports { ck_io[14] }]; #
122     IO_L18N_T2_13 Sch=ck_a[0]
123 #set_property -dict { PACKAGE_PIN Y12    IOSTANDARD LVCMOS33 } [get_ports { ck_io[15] }]; #
124     IO_L20P_T3_13 Sch=ck_a[1]
125 #set_property -dict { PACKAGE_PIN W11    IOSTANDARD LVCMOS33 } [get_ports { ck_io[16] }]; #
126     IO_L18P_T2_13 Sch=ck_a[2]
127 #set_property -dict { PACKAGE_PIN V11    IOSTANDARD LVCMOS33 } [get_ports { ck_io[17] }]; #
128     IO_L21P_T3_DQS_13 Sch=ck_a[3]
129 #set_property -dict { PACKAGE_PIN T5     IOSTANDARD LVCMOS33 } [get_ports { ck_io[18] }]; #
130     IO_L19P_T3_13 Sch=ck_a[4]
131 #set_property -dict { PACKAGE_PIN U10    IOSTANDARD LVCMOS33 } [get_ports { ck_io[19] }]; #
132     IO_L12N_T1_MRCC_13 Sch=ck_a[5]

133 ##ChipKit Digital I/O On Inner Analog Header
134 ##NOTE: These pins will need to be connected to the XADC core when used as differential
135     analog inputs (Chipkit analog pins A6-A11)
136
137 #set_property -dict { PACKAGE_PIN B20    IOSTANDARD LVCMOS33 } [get_ports { ck_io[20] }]; #
138     IO_L1N_T0_AD0N_35 Sch=ad_n[0]
139 #set_property -dict { PACKAGE_PIN C20    IOSTANDARD LVCMOS33 } [get_ports { ck_io[21] }]; #
140     IO_L1P_T0_AD0P_35 Sch=ad_p[0]
141 #set_property -dict { PACKAGE_PIN F20    IOSTANDARD LVCMOS33 } [get_ports { ck_io[22] }]; #
142     IO_L15N_T2_DQS_AD12N_35 Sch=ad_n[12]
143 #set_property -dict { PACKAGE_PIN F19    IOSTANDARD LVCMOS33 } [get_ports { ck_io[23] }]; #
144     IO_L15P_T2_DQS_AD12P_35 Sch=ad_p[12]
145 #set_property -dict { PACKAGE_PIN A20    IOSTANDARD LVCMOS33 } [get_ports { ck_io[24] }]; #
146     IO_L2N_T0_AD8N_35 Sch=ad_n[8]
147 #set_property -dict { PACKAGE_PIN B19    IOSTANDARD LVCMOS33 } [get_ports { ck_io[25] }]; #
148     IO_L2P_T0_AD8P_35 Sch=ad_p[8]

149 ##ChipKit Digital I/O High
150
151 #set_property -dict { PACKAGE_PIN U5     IOSTANDARD LVCMOS33 } [get_ports { ck_io[26] }]; #
152     IO_L19N_T3_VREF_13 Sch=ck_io[26]
153 #set_property -dict { PACKAGE_PIN V5     IOSTANDARD LVCMOS33 } [get_ports { ck_io[27] }]; #
154     IO_L6N_T0_VREF_13 Sch=ck_io[27]

```

```

129 #set_property -dict { PACKAGE_PIN V6      IOSTANDARD LVCMOS33 } [get_ports { ck_io[28] }]; #
130     IO_L22P_T3_13 Sch=ck_io[28]
131 #set_property -dict { PACKAGE_PIN U7      IOSTANDARD LVCMOS33 } [get_ports { ck_io[29] }]; #
132     IO_L11P_T1_SRCC_13 Sch=ck_io[29]
133 #set_property -dict { PACKAGE_PIN V7      IOSTANDARD LVCMOS33 } [get_ports { ck_io[30] }]; #
134     IO_L11N_T1_SRCC_13 Sch=ck_io[30]
135 #set_property -dict { PACKAGE_PIN U8      IOSTANDARD LVCMOS33 } [get_ports { ck_io[31] }]; #
136     IO_L17N_T2_13 Sch=ck_io[31]
137 #set_property -dict { PACKAGE_PIN V8      IOSTANDARD LVCMOS33 } [get_ports { ck_io[32] }]; #
138     IO_L15P_T2_DQS_13 Sch=ck_io[32]
139 #set_property -dict { PACKAGE_PIN V10     IOSTANDARD LVCMOS33 } [get_ports { ck_io[33] }]; #
140     IO_L21N_T3_DQS_13 Sch=ck_io[33]
141 #set_property -dict { PACKAGE_PIN W10     IOSTANDARD LVCMOS33 } [get_ports { ck_io[34] }]; #
142     IO_L16P_T2_13 Sch=ck_io[34]
143 #set_property -dict { PACKAGE_PIN W6      IOSTANDARD LVCMOS33 } [get_ports { ck_io[35] }]; #
144     IO_L22N_T3_13 Sch=ck_io[35]
145 ## ChipKit SPI
146
147 #set_property -dict { PACKAGE_PIN W15     IOSTANDARD LVCMOS33 } [get_ports { ck_miso }]; #
148     IO_L10N_T1_34 Sch=ck_miso
149 #set_property -dict { PACKAGE_PIN T12     IOSTANDARD LVCMOS33 } [get_ports { ck_mosi }]; #
150     IO_L2P_T0_34 Sch=ck_mosi
151 #set_property -dict { PACKAGE_PIN H15     IOSTANDARD LVCMOS33 } [get_ports { ck_sck }]; #
152     IO_L19P_T3_35 Sch=ck_sck
153 #set_property -dict { PACKAGE_PIN F16     IOSTANDARD LVCMOS33 } [get_ports { ck_ss }]; #
154     IO_L6P_T0_35 Sch=ck_ss
155 ## ChipKit I2C
156
157 #set_property -dict { PACKAGE_PIN P16     IOSTANDARD LVCMOS33 } [get_ports { ck_scl }]; #
158     IO_L24N_T3_34 Sch=ck_scl
159 #set_property -dict { PACKAGE_PIN P15     IOSTANDARD LVCMOS33 } [get_ports { ck_sda }]; #
160     IO_L24P_T3_34 Sch=ck_sda
161 ##HDMI Rx
162
163 #set_property -dict { PACKAGE_PIN H17     IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_cec }]; #
164     #IO_L13N_T2_MRCC_35 Sch=hdmi_rx_cec
165 #set_property -dict { PACKAGE_PIN P19     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_clk_n }]; #
166     #IO_L13N_T2_MRCC_34 Sch=hdmi_rx_clk_n
167 #set_property -dict { PACKAGE_PIN N18     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_clk_p }]; #
168     #IO_L13P_T2_MRCC_34 Sch=hdmi_rx_clk_p
169 #set_property -dict { PACKAGE_PIN W20     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_d_n[0] }]; #
170     #IO_L16N_T2_34 Sch=hdmi_rx_d_n[0]
171 #set_property -dict { PACKAGE_PIN V20     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_d_p[0] }]; #
172     #IO_L16P_T2_34 Sch=hdmi_rx_d_p[0]
173 #set_property -dict { PACKAGE_PIN U20     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_d_n[1] }]; #
174     #IO_L15N_T2_DQS_34 Sch=hdmi_rx_d_n[1]
175 #set_property -dict { PACKAGE_PIN T20     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_d_p[1] }]; #
176     #IO_L15P_T2_DQS_34 Sch=hdmi_rx_d_p[1]
177 #set_property -dict { PACKAGE_PIN P20     IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_d_n[2] }]; #
178     #IO_L14N_T2_SRCC_34 Sch=hdmi_rx_d_n[2]

```

```

167 #set_property -dict { PACKAGE_PIN N20    IOSTANDARD TMDS_33 } [get_ports { hdmi_rx_d_p[2]
168     }]; #IO_L14P_T2_SRCC_34 Sch=hdmi_rx_d_p[2]
169 #set_property -dict { PACKAGE_PIN T19    IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_hpd }];
170     #IO_25_34 Sch=hdmi_rx_hpd
171 #set_property -dict { PACKAGE_PIN U14    IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_scl }];
172     #IO_L11P_T1_SRCC_34 Sch=hdmi_rx_scl
173 #set_property -dict { PACKAGE_PIN U15    IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_sda }];
174     #IO_L11N_T1_SRCC_34 Sch=hdmi_rx_sda
175
176 ##HDMI Tx
177
178 #set_property -dict { PACKAGE_PIN G15    IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_cec }];
179     #IO_L19N_T3_VREF_35 Sch=hdmi_tx_cec
180 #set_property -dict { PACKAGE_PIN L17    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_clk_n
181     }]; #IO_L11N_T1_SRCC_35 Sch=hdmi_tx_clk_n
182 #set_property -dict { PACKAGE_PIN L16    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_clk_p
183     }]; #IO_L11P_T1_SRCC_35 Sch=hdmi_tx_clk_p
184 #set_property -dict { PACKAGE_PIN K18    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_d_n[0]
185     }]; #IO_L12N_T1_MRCC_35 Sch=hdmi_tx_d_n[0]
186 #set_property -dict { PACKAGE_PIN K17    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_d_p[0]
187     }]; #IO_L12P_T1_MRCC_35 Sch=hdmi_tx_d_p[0]
188 #set_property -dict { PACKAGE_PIN J19    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_d_n[1]
189     }]; #IO_L10N_T1_AD11N_35 Sch=hdmi_tx_d_n[1]
190 #set_property -dict { PACKAGE_PIN K19    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_d_p[1]
191     }]; #IO_L10P_T1_AD11P_35 Sch=hdmi_tx_d_p[1]
192 #set_property -dict { PACKAGE_PIN H18    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_d_n[2]
193     }]; #IO_L14N_T2_AD4N_SRCC_35 Sch=hdmi_tx_d_n[2]
194 #set_property -dict { PACKAGE_PIN J18    IOSTANDARD TMDS_33 } [get_ports { hdmi_tx_d_p[2]
195     }]; #IO_L14P_T2_AD4P_SRCC_35 Sch=hdmi_tx_d_p[2]
196 #set_property -dict { PACKAGE_PIN R19    IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_hpdn
197     }]; #IO_0_34 Sch=hdmi_tx_hpdn
198 #set_property -dict { PACKAGE_PIN M17    IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_scl }];
199     #IO_L8P_T1_AD10P_35 Sch=hdmi_tx_scl
200 #set_property -dict { PACKAGE_PIN M18    IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_sda }];
201     #IO_L8N_T1_AD10N_35 Sch=hdmi_tx_sda
202
203 ##Crypto SDA
204
205 #set_property -dict { PACKAGE_PIN J15    IOSTANDARD LVCMOS33 } [get_ports { crypto_sda }];
206     #IO_25_35 Sch=crypto_sda
207
208 ##Pblocks
209
210 create_pblock pblock_MB1
211 add_cells_to_pblock [get_pblocks pblock_MB1] [get_cells -quiet [list tmr_design_i/
212     tmr_system/MB1]]
213 create_pblock pblock_MB2
214 add_cells_to_pblock [get_pblocks pblock_MB2] [get_cells -quiet [list tmr_design_i/
215     tmr_system/MB2]]
216 create_pblock pblock_MB3
217 add_cells_to_pblock [get_pblocks pblock_MB3] [get_cells -quiet [list tmr_design_i/
218     tmr_system/MB3]]

```

ANEXO E. ARCHIVO MAIN.C DE LA APLICACIÓN DEL CORTEX-A9

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include "xgpio.h"
4 #include "xuartps_hw.h"
5 #include "xuartps.h"
6 #include "xuartns550.h"
7 #include "xparameters.h"
8 #include "xil_io.h"
9 #include "sleep.h"
10
11
12 // Function to convert uint32_t to binary string
13 void uint32_to_binstr(uint32_t value, char *str, int bits) {
14     for (int i = bits - 1; i >= 0; i--) {
15         str[bits - 1 - i] = (value & (1U << i)) ? '1' : '0';
16     }
17     str[bits] = '\0';
18 }
19
20 int main(void) {
21     XUartPs xuartps0;
22     XUartPs xuartps1;
23     XUartNs550 uart_status;
24     char binstr1[33], binstr2[33], binstr3[33]; // 32 bits + null terminator
25     char msg[64];
26     XGpio status1, status2, status3;
27     uint32_t tmr_status1_prev = 0xFFFFFFFF, tmr_status2_prev = 0xFFFFFFFF,
28             tmr_status3_prev = 0xFFFFFFFF; // Initialize all to 0xFFFFFFFF to detect changes
29     uint32_t tmr_status1, tmr_status2, tmr_status3;
30     uint32_t reg_value;
31     int32_t length, sent, chunk;
32
33     // Transfer control of the PL configuration logic from PCAP (Programmable Logic
34     // Configuration Access Port)
35     // to ICAP (Internal Configuration Access Port)
36     // To do this, we clear PCAP_PR (bit 27) in the PS device configuration control
37     // register
38     // (DEVCFG_CTRL, address 0xF8007000)
39
40     reg_value = Xil_In32(0xF8007000); // Read the DEVCFG_CTRL register
41     reg_value &= ~(~(uint32_t)1 << 27); // Clear the PCAP_PR bit: ~(1 << 27) is all 1s
42     // except bit 27, which is 0
43     Xil_Out32(0xF8007000, reg_value); // Write back the modified value
44
45     // Initialize the GPIOs for the TMR Status
46     XGpio_Initialize(&status1, XPAR_XGPIO_0_BASEADDR);
47     XGpio_Initialize(&status2, XPAR_XGPIO_1_BASEADDR);
48     XGpio_Initialize(&status3, XPAR_XGPIO_2_BASEADDR);
49     // Set the direction of the GPIOs to input
50     XGpio_SetDataDirection(&status1, 1, 0xFFFFFFFF);
51     XGpio_SetDataDirection(&status2, 1, 0xFFFFFFFF);
52     XGpio_SetDataDirection(&status3, 1, 0xFFFFFFFF);
53
54     // Initialize the UART used to print the TMR Status
55     XUartNs550_Initialize(&uart_status, XPAR_XUARTNS550_0_BASEADDR);
56     // Set data format to 8 bits, no parity, 1 stop bit (8N1)
57     XUartNs550Format format;
58     format.BaudRate = 115200; // Set baud rate to 115200
59     format.DataBits = XUN_FORMAT_8_BITS;
```

```

56     format.Parity    = XUN_FORMAT_NO_PARITY;
57     format.StopBits = XUN_FORMAT_1_STOP_BIT;
58     XUartNs550_SetDataFormat(&uart_status, &format);
59     // Enable FIFOs, resetting both TX and RX FIFOs
60     XUartNs550_SetOptions(&uart_status, XUN_OPTION_FIFOS_ENABLE | XUN_OPTION_RESET_TX_FIFO
61                           | XUN_OPTION_RESET_RX_FIFO);
62
63     // Initialize UART 0
64     XUartPs_Config *cfg0 = XUartPs_LookupConfig(XPAR_XUARTPS_0_BASEADDR);
65     cfg0->ModemPinsConnected = 0;
66     XUartPs_CfgInitialize(&xuartps0, cfg0, cfg0->BaseAddress);
67     XUartPs_SetBaudRate(&xuartps0, 115200);
68
69     // Initialize UART 1
70     XUartPs_Config *cfg1 = XUartPs_LookupConfig(XPAR_XUARTPS_1_BASEADDR);
71     cfg1->ModemPinsConnected = 0;
72     XUartPs_CfgInitialize(&xuartps1, cfg1, cfg1->BaseAddress);
73     XUartPs_SetBaudRate(&xuartps1, 115200);
74
75     // Initialization messages
76     xil_printf("PCAP_PR bit cleared, control transferred to ICAP.\n");
77     xil_printf("UART 0 initialized at base address: 0x%08X\n", cfg0->BaseAddress);
78     xil_printf("UART 1 initialized at base address: 0x%08X\n", cfg1->BaseAddress);
79     xil_printf("UART 0 and UART 1 are now bridged.\n");
80
81     while (1) {
82         // Bridge UART0 RX to UART1 TX
83         if (XUartPs_IsReceiveData(xuartps0.Config.BaseAddress)) {
84             XUartPs_SendByte(xuartps1.Config.BaseAddress, XUartPs_RecvByte(xuartps0.Config
85                               .BaseAddress));
86         }
87
88         // Bridge UART1 RX to UART0 TX
89         if (XUartPs_IsReceiveData(xuartps1.Config.BaseAddress)) {
90             XUartPs_SendByte(xuartps0.Config.BaseAddress, XUartPs_RecvByte(xuartps1.Config
91                               .BaseAddress));
92         }
93
94         // Get the TMR status by reading the GPIOs
95         tmr_status1 = XGpio_DiscreteRead(&status1, 1);
96         tmr_status2 = XGpio_DiscreteRead(&status2, 1);
97         tmr_status3 = XGpio_DiscreteRead(&status3, 1);
98
99         // Check if the TMR status has changed
100        if (tmr_status1 != tmr_status1_prev) {
101            uint32_to_binstr(tmr_status1, binstr1, 32);
102            sprintf(msg, "TMR Status 1 changed: %b\n", binstr1);
103            length = strlen(msg);
104            sent = 0;
105            while (sent < length) { // The UART's FIFO size is 16 bytes, so we need to
106                send it in chunks
107                chunk = XUartNs550_Send(&uart_status, (u8*)&msg[sent], length - sent);
108                sent += chunk;
109                while (XUartNs550_IsSending(&uart_status));
110            }
111            tmr_status1_prev = tmr_status1;
112        }
113        if (tmr_status2 != tmr_status2_prev) {
114            uint32_to_binstr(tmr_status2, binstr2, 32);
115            sprintf(msg, "TMR Status 2 changed: %b\n", binstr2);
116            length = strlen(msg);
117            sent = 0;
118            while (sent < length) {
119                chunk = XUartNs550_Send(&uart_status, (u8*)&msg[sent], length - sent);
120                sent += chunk;
121                while (XUartNs550_IsSending(&uart_status));
122            }
123        }

```

```
119         tmr_status2_prev = tmr_status2;
120     }
121     if (tmr_status3 != tmr_status3_prev) {
122         uint32_to_binstr(tmr_status3, binstr3, 32);
123         sprintf(msg, "TMR Status 3 changed: %b\n", binstr3);
124         length = strlen(msg);
125         sent = 0;
126         while (sent < length) {
127             chunk = XUartNs550_Send(&uart_status, (u8*)&msg[sent], length - sent);
128             sent += chunk;
129             while (XUartNs550_IsSending(&uart_status));
130         }
131         tmr_status3_prev = tmr_status3;
132     }
133
134     usleep(10); // Avoid UART saturation
135
136 }
137 }
```

ANEXO F. ARCHIVO XPARAMETERS.H DEL DOMINIO DEL CORTEX-A9

```
1 #ifndef XPARAMETERS_H    /* prevent circular inclusions */
2 #define XPARAMETERS_H    /* by using protection macros */
3
4 #define XPAR_XCORESIGHTPS_DCC_NUM_INSTANCES 1
5
6 /* Definitions for peripheral CORESIGHT */
7 #define XPAR_CORESIGHT_COMPATIBLE "xlnx,ps7-coresight-comp-1.00.a"
8 #define XPAR_CORESIGHT_BASEADDR 0xf8800000
9 #define XPAR_CORESIGHT_HIGHADDR 0xf88fffff
10
11 /* Canonical definitions for peripheral CORESIGHT */
12 #define XPAR_XCORESIGHTPS_DCC_0_BASEADDR 0xf8800000
13 #define XPAR_XCORESIGHTPS_DCC_0_HIGHADDR 0xf88fffff
14 #define XPAR_XCORESIGHTPS_DCC_0_COMPATIBLE "xlnx,ps7-coresight-comp-1.00.a"
15
16 #define XPAR_XDEVCFG_NUM_INSTANCES 1
17
18 /* Definitions for peripheral DEVCFG */
19 #define XPAR_DEVCFG_COMPATIBLE "xlnx,zynq-devcfg-1.0"
20 #define XPAR_DEVCFG_BASEADDR 0xf8007000
21 #define XPAR_DEVCFG_HIGHADDR 0xf80070ff
22 #define XPAR_DEVCFG_INTERRUPTS 0x4008
23 #define XPAR_DEVCFG_INTERRUPT_PARENT 0xf8f01000
24
25 /* Canonical definitions for peripheral DEVCFG */
26 #define XPAR_XDEVCFG_0_BASEADDR 0xf8007000
27 #define XPAR_XDEVCFG_0_HIGHADDR 0xf80070ff
28 #define XPAR_XDEVCFG_0_COMPATIBLE "xlnx,zynq-devcfg-1.0"
29 #define XPAR_XDEVCFG_0_INTERRUPTS 0x4008
30 #define XPAR_XDEVCFG_0_INTERRUPT_PARENT 0xf8f01000
31
32 #define XPAR_XDMAPS_NUM_INSTANCES 1
33
34 /* Definitions for peripheral DMAC_S */
35 #define XPAR_DMAC_S_COMPATIBLE "arm,pl330"
36 #define XPAR_DMAC_S_BASEADDR 0xf8003000
37 #define XPAR_DMAC_S_HIGHADDR 0xf8003fff
38 #define XPAR_DMAC_S_INTERRUPTS 0x400d
39 #define XPAR_DMAC_S_INTERRUPTS_1 0x400e
40 #define XPAR_DMAC_S_INTERRUPTS_2 0x400f
41 #define XPAR_DMAC_S_INTERRUPTS_3 0x4010
42 #define XPAR_DMAC_S_INTERRUPTS_4 0x4011
43 #define XPAR_DMAC_S_INTERRUPTS_5 0x4028
44 #define XPAR_DMAC_S_INTERRUPTS_6 0x4029
45 #define XPAR_DMAC_S_INTERRUPTS_7 0x402a
46 #define XPAR_DMAC_S_INTERRUPTS_8 0x402b
47 #define XPAR_DMAC_S_INTERRUPT_PARENT 0xf8f01000
48
49 /* Canonical definitions for peripheral DMAC_S */
50 #define XPAR_XDMAPS_0_BASEADDR 0xf8003000
51 #define XPAR_XDMAPS_0_HIGHADDR 0xf8003fff
52 #define XPAR_XDMAPS_0_COMPATIBLE "arm,pl330"
53 #define XPAR_XDMAPS_0_INTERRUPTS 0x400d
54 #define XPAR_XDMAPS_0_INTERRUPT_PARENT 0xf8f01000
55
56 #define XPAR_XEMACPS_NUM_INSTANCES 1
57
58 /* Definitions for peripheral GEM0 */
59 #define XPAR_GEM0_COMPATIBLE "xlnx,zynq-gem"
```

```

60 #define XPAR_GEM0_BASEADDR 0xe000b000
61 #define XPAR_GEM0_HIGHADDR 0xe000bfff
62 #define XPAR_GEM0_DMA_COHERENT 0x0
63 #define XPAR_GEM0_INTERRUPTS 0x4016
64 #define XPAR_GEM0_INTERRUPT_PARENT 0xf8f01000
65 #define XPAR_GEM0_REF_CLK 0x0
66 #define XPAR_GEM0_PHY_MODE "rgmii-id"
67
68 /* Canonical definitions for peripheral GEM0 */
69 #define XPAR_XEMACPS_0_BASEADDR 0xe000b000
70 #define XPAR_XEMACPS_0_HIGHADDR 0xe000bfff
71 #define XPAR_XEMACPS_0_COMPATIBLE "xlnx,zynq-gem"
72 #define XPAR_XEMACPS_0_DMA_COHERENT 0x0
73 #define XPAR_XEMACPS_0_INTERRUPTS 0x4016
74 #define XPAR_XEMACPS_0_INTERRUPT_PARENT 0xf8f01000
75 #define XPAR_XEMACPS_0_PHY_MODE "rgmii-id"
76 #define XPAR_XEMACPS_0_REF_CLK 0x0
77
78 #define XPAR_XGPIO_NUM_INSTANCES 3
79
80 /* Definitions for peripheral AXI_GPIO_STATUS_1 */
81 #define XPAR_AXI_GPIO_STATUS_1_COMPATIBLE "xlnx,axi-gpio-2.0"
82 #define XPAR_AXI_GPIO_STATUS_1_BASEADDR 0x41200000
83 #define XPAR_AXI_GPIO_STATUS_1_HIGHADDR 0x4120ffff
84 #define XPAR_AXI_GPIO_STATUS_1_INTERRUPT_PRESENT 0x0
85 #define XPAR_AXI_GPIO_STATUS_1_IS_DUAL 0x0
86 #define XPAR_AXI_GPIO_STATUS_1_GPIO_WIDTH 0x20
87
88 /* Canonical definitions for peripheral AXI_GPIO_STATUS_1 */
89 #define XPAR_XGPIO_0_BASEADDR 0x41200000
90 #define XPAR_XGPIO_0_HIGHADDR 0x4120ffff
91 #define XPAR_XGPIO_0_COMPATIBLE "xlnx,axi-gpio-2.0"
92 #define XPAR_XGPIO_0_GPIO_WIDTH 0x20
93 #define XPAR_XGPIO_0_INTERRUPT_PRESENT 0x0
94 #define XPAR_XGPIO_0_IS_DUAL 0x0
95
96 /* Definitions for peripheral AXI_GPIO_STATUS_2 */
97 #define XPAR_AXI_GPIO_STATUS_2_COMPATIBLE "xlnx,axi-gpio-2.0"
98 #define XPAR_AXI_GPIO_STATUS_2_BASEADDR 0x41210000
99 #define XPAR_AXI_GPIO_STATUS_2_HIGHADDR 0x4121ffff
100 #define XPAR_AXI_GPIO_STATUS_2_INTERRUPT_PRESENT 0x0
101 #define XPAR_AXI_GPIO_STATUS_2_IS_DUAL 0x0
102 #define XPAR_AXI_GPIO_STATUS_2_GPIO_WIDTH 0x20
103
104 /* Canonical definitions for peripheral AXI_GPIO_STATUS_2 */
105 #define XPAR_XGPIO_1_BASEADDR 0x41210000
106 #define XPAR_XGPIO_1_HIGHADDR 0x4121ffff
107 #define XPAR_XGPIO_1_COMPATIBLE "xlnx,axi-gpio-2.0"
108 #define XPAR_XGPIO_1_GPIO_WIDTH 0x20
109 #define XPAR_XGPIO_1_INTERRUPT_PRESENT 0x0
110 #define XPAR_XGPIO_1_IS_DUAL 0x0
111
112 /* Definitions for peripheral AXI_GPIO_STATUS_3 */
113 #define XPAR_AXI_GPIO_STATUS_3_COMPATIBLE "xlnx,axi-gpio-2.0"
114 #define XPAR_AXI_GPIO_STATUS_3_BASEADDR 0x41220000
115 #define XPAR_AXI_GPIO_STATUS_3_HIGHADDR 0x4122ffff
116 #define XPAR_AXI_GPIO_STATUS_3_INTERRUPT_PRESENT 0x0
117 #define XPAR_AXI_GPIO_STATUS_3_IS_DUAL 0x0
118 #define XPAR_AXI_GPIO_STATUS_3_GPIO_WIDTH 0x20
119
120 /* Canonical definitions for peripheral AXI_GPIO_STATUS_3 */
121 #define XPAR_XGPIO_2_BASEADDR 0x41220000
122 #define XPAR_XGPIO_2_HIGHADDR 0x4122ffff
123 #define XPAR_XGPIO_2_COMPATIBLE "xlnx,axi-gpio-2.0"
124 #define XPAR_XGPIO_2_GPIO_WIDTH 0x20
125 #define XPAR_XGPIO_2_INTERRUPT_PRESENT 0x0
126 #define XPAR_XGPIO_2_IS_DUAL 0x0

```

```

127
128 #define XPAR_XGPIOPS_NUM_INSTANCES 1
129
130 /* Definitions for peripheral GPIO0 */
131 #define XPAR_GPIO0_COMPATIBLE "xlnx,zynq-gpio-1.0"
132 #define XPAR_GPIO0_BASEADDR 0xe000a000
133 #define XPAR_GPIO0_HIGHADDR 0xe000afff
134 #define XPAR_GPIO0_INTERRUPTS 0x4014
135 #define XPAR_GPIO0_INTERRUPT_PARENT 0xf8f01000
136
137 /* Canonical definitions for peripheral GPIO0 */
138 #define XPAR_XGPIOPS_0_BASEADDR 0xe000a000
139 #define XPAR_XGPIOPS_0_HIGHADDR 0xe000afff
140 #define XPAR_XGPIOPS_0_COMPATIBLE "xlnx,zynq-gpio-1.0"
141 #define XPAR_XGPIOPS_0_INTERRUPTS 0x4014
142 #define XPAR_XGPIOPS_0_INTERRUPT_PARENT 0xf8f01000
143
144 #define XPAR_XQSPIPS_NUM_INSTANCES 1
145
146 /* Definitions for peripheral QSPI */
147 #define XPAR_QSPI_COMPATIBLE "xlnx,zynq-qspi-1.0"
148 #define XPAR_QSPI_BASEADDR 0xe000d000
149 #define XPAR_QSPI_HIGHADDR 0xe000dff
150 #define XPAR_QSPI_CLOCK_FREQ 0xebec200
151 #define XPAR_QSPI_CONNECTION_MODE 0x0
152 #define XPAR_QSPI_INTERRUPTS 0x4013
153 #define XPAR_QSPI_INTERRUPT_PARENT 0xf8f01000
154 #define XPAR_QSPI_QSPI_MODE 0x0
155 #define XPAR_QSPI_QSPI_BUS_WIDTH 0x2
156
157 /* Canonical definitions for peripheral QSPI */
158 #define XPAR_XQSPIPS_0_BASEADDR 0xe000d000
159 #define XPAR_XQSPIPS_0_HIGHADDR 0xe000dff
160 #define XPAR_XQSPIPS_0_COMPATIBLE "xlnx,zynq-qspi-1.0"
161 #define XPAR_XQSPIPS_0_CLOCK_FREQ 0xebec200
162 #define XPAR_XQSPIPS_0_CONNECTION_MODE 0x0
163 #define XPAR_XQSPIPS_0_INTERRUPTS 0x4013
164 #define XPAR_XQSPIPS_0_INTERRUPT_PARENT 0xf8f01000
165 #define XPAR_XQSPIPS_0_QSPI_MODE 0x0
166 #define XPAR_XQSPIPS_0_QSPI_BUS_WIDTH 0x2
167
168 #define XPAR_XSCUGIC_NUM_INSTANCES 1
169
170 /* Definitions for peripheral INTC */
171 #define XPAR_INTC_COMPATIBLE "arm,cortex-a9-gic"
172 #define XPAR_INTC_BASEADDR 0xf8f01000
173 #define XPAR_INTC_HIGHADDR 0xf8f01fff
174 #define XPAR_INTC_BASEADDR_1 0xf8f00100
175 #define XPAR_INTC_HANDLER_TABLE 0x0
176
177 /* Canonical definitions for peripheral INTC */
178 #define XPAR_XSCUGIC_0_BASEADDR 0xf8f01000
179 #define XPAR_XSCUGIC_0_HIGHADDR 0xf8f01fff
180 #define XPAR_XSCUGIC_0_HANDLER_TABLE 0x0
181 #define XPAR_XSCUGIC_0_COMPATIBLE "arm,cortex-a9-gic"
182
183 #define XPAR_XSCUTIMER_NUM_INSTANCES 1
184
185 /* Definitions for peripheral SCUTIMER */
186 #define XPAR_SCUTIMER_COMPATIBLE "arm,cortex-a9-twd-timer"
187 #define XPAR_SCUTIMER_BASEADDR 0xf8f00600
188 #define XPAR_SCUTIMER_HIGHADDR 0xf8f0061f
189 #define XPAR_SCUTIMER_INTERRUPTS 0x13100d
190 #define XPAR_SCUTIMER_INTERRUPT_PARENT 0xf8f01000
191
192 /* Canonical definitions for peripheral SCUTIMER */
193 #define XPAR_XSCUTIMER_0_BASEADDR 0xf8f00600

```

```

194 #define XPAR_XSCUTIMER_0_HIGHADDR 0xf8f0061f
195 #define XPAR_XSCUTIMER_0_COMPATIBLE "arm,cortex-a9-twd-timer"
196 #define XPAR_XSCUTIMER_0_INTERRUPTS 0x13100d
197 #define XPAR_XSCUTIMER_0_INTERRUPT_PARENT 0xf8f01000
198
199 #define XPAR_XSCUWDT_NUM_INSTANCES 1
200
201 /* Definitions for peripheral SCUWDT */
202 #define XPAR_SCUWDT_COMPATIBLE "xlnx,ps7-scuwdt-1.00.a"
203 #define XPAR_SCUWDT_BASEADDR 0xf8f00620
204 #define XPAR_SCUWDT_HIGHADDR 0xf8f006ff
205 #define XPAR_SCUWDT_INTERRUPTS 0x10400e
206 #define XPAR_SCUWDT_INTERRUPT_PARENT 0xf8f01000
207
208 /* Canonical definitions for peripheral SCUWDT */
209 #define XPAR_XSCUWDT_0_BASEADDR 0xf8f00620
210 #define XPAR_XSCUWDT_0_HIGHADDR 0xf8f006ff
211 #define XPAR_XSCUWDT_0_COMPATIBLE "xlnx,ps7-scuwdt-1.00.a"
212 #define XPAR_XSCUWDT_0_INTERRUPTS 0x10400e
213 #define XPAR_XSCUWDT_0_INTERRUPT_PARENT 0xf8f01000
214
215 #define XPAR_XSDPS_NUM_INSTANCES 1
216
217 /* Definitions for peripheral SDHCI0 */
218 #define XPAR_SDHCI0_COMPATIBLE "arasan,sdhci-8.9a"
219 #define XPAR_SDHCI0_BASEADDR 0xe0100000
220 #define XPAR_SDHCI0_HIGHADDR 0xe0100fff
221 #define XPAR_SDHCI0_SDIO_CLK_FREQ_HZ 0x2faf080
222 #define XPAR_SDHCI0_HAS_CD 0x1
223 #define XPAR_SDHCI0_HAS_WP 0x0
224 #define XPAR_SDHCI0_BUS_WIDTH 0x0
225 #define XPAR_SDHCI0_MIO_BANK 0x0
226 #define XPAR_SDHCI0_HAS_EMIO 0x0
227 #define XPAR_SDHCI0_SLOT_TYPE 0x0
228 #define XPAR_SDHCI0_IS_CACHE_COHERENT 0x0
229 #define XPAR_SDHCI0_CLOCKS 0x15
230 #define XPAR_SDHCI0_CLK_50_SDR_ITAP_DLY 0x0
231 #define XPAR_SDHCI0_CLK_50_SDR_OTAP_DLY 0x0
232 #define XPAR_SDHCI0_CLK_50_DDR_ITAP_DLY 0x0
233 #define XPAR_SDHCI0_CLK_50_DDR_OTAP_DLY 0x0
234 #define XPAR_SDHCI0_CLK_100_SDR_OTAP_DLY 0x0
235 #define XPAR_SDHCI0_CLK_200_SDR_OTAP_DLY 0x0
236 #define XPAR_SDHCI0_CLK_200_DDR_OTAP_DLY 0x0
237
238 /* Canonical definitions for peripheral SDHCI0 */
239 #define XPAR_XSDPS_0_BASEADDR 0xe0100000
240 #define XPAR_XSDPS_0_HIGHADDR 0xe0100fff
241 #define XPAR_XSDPS_0_BUS_WIDTH 0x0
242 #define XPAR_XSDPS_0_COMPATIBLE "arasan,sdhci-8.9a"
243 #define XPAR_XSDPS_0_CLOCKS 0x15
244 #define XPAR_XSDPS_0_CLK_50_SDR_ITAP_DLY 0x0
245 #define XPAR_XSDPS_0_CLK_50_SDR_OTAP_DLY 0x0
246 #define XPAR_XSDPS_0_CLK_50_DDR_ITAP_DLY 0x0
247 #define XPAR_XSDPS_0_CLK_50_DDR_OTAP_DLY 0x0
248 #define XPAR_XSDPS_0_CLK_100_SDR_OTAP_DLY 0x0
249 #define XPAR_XSDPS_0_CLK_200_SDR_OTAP_DLY 0x0
250 #define XPAR_XSDPS_0_CLK_200_DDR_OTAP_DLY 0x0
251 #define XPAR_XSDPS_0_HAS_CD 0x1
252 #define XPAR_XSDPS_0_HAS_WP 0x0
253 #define XPAR_XSDPS_0_HAS_EMIO 0x0
254 #define XPAR_XSDPS_0_IS_CACHE_COHERENT 0x0
255 #define XPAR_XSDPS_0_MIO_BANK 0x0
256 #define XPAR_XSDPS_0_SDIO_CLK_FREQ_HZ 0x2faf080
257 #define XPAR_XSDPS_0_SLOT_TYPE 0x0
258
259 #define XPAR_XUARTNS550_NUM_INSTANCES 1
260

```

```

261 /* Definitions for peripheral AXI_UART16550_STATUS */
262 #define XPAR_AXI_UART16550_STATUS_COMPATIBLE "xlnx,axi-uart16550-2.0"
263 #define XPAR_AXI_UART16550_STATUS_BASEADDR 0x43c00000
264 #define XPAR_AXI_UART16550_STATUS_HIGHADDR 0x43c0ffff
265 #define XPAR_AXI_UART16550_STATUS_CLOCK_FREQ 0x2faf080
266 #define XPAR_AXI_UART16550_STATUS_CURRENT_SPEED 0x0
267
268 /* Canonical definitions for peripheral AXI_UART16550_STATUS */
269 #define XPAR_XUARTNS550_0_BASEADDR 0x43c00000
270 #define XPAR_XUARTNS550_0_HIGHADDR 0x43c0ffff
271 #define XPAR_XUARTNS550_0_COMPATIBLE "xlnx,axi-uart16550-2.0"
272 #define XPAR_XUARTNS550_0_CLOCK_FREQ 0x2faf080
273 #define XPAR_XUARTNS550_0_CURRENT_SPEED 0x0
274
275 #define XPAR_XUARTPS_NUM_INSTANCES 2
276
277 /* Definitions for peripheral UART0 */
278 #define XPAR_UART0_COMPATIBLE "xlnx,xuartps"
279 #define XPAR_UART0_BASEADDR 0xe0000000
280 #define XPAR_UART0_HIGHADDR 0xe0000fff
281 #define XPAR_UART0_CLOCK_FREQ 0x5f5e100
282 #define XPAR_UART0_CTS_OVERRIDE 0x1
283 #define XPAR_UART0_CLOCKS 0x17
284 #define XPAR_UART0_INTERRUPTS 0x401b
285 #define XPAR_UART0_INTERRUPT_PARENT 0xf8f01000
286
287 /* Canonical definitions for peripheral UART0 */
288 #define XPAR_XUARTPS_0_BASEADDR 0xe0000000
289 #define XPAR_XUARTPS_0_HIGHADDR 0xe0000fff
290 #define XPAR_XUARTPS_0_COMPATIBLE "xlnx,xuartps"
291 #define XPAR_XUARTPS_0_CLOCK_FREQ 0x5f5e100
292 #define XPAR_XUARTPS_0_CTS_OVERRIDE 0x1
293 #define XPAR_XUARTPS_0_CLOCKS 0x17
294 #define XPAR_XUARTPS_0_INTERRUPTS 0x401b
295 #define XPAR_XUARTPS_0_INTERRUPT_PARENT 0xf8f01000
296
297 /* Definitions for peripheral UART1 */
298 #define XPAR_UART1_COMPATIBLE "xlnx,xuartps"
299 #define XPAR_UART1_BASEADDR 0xe0001000
300 #define XPAR_UART1_HIGHADDR 0xe0001fff
301 #define XPAR_UART1_CLOCK_FREQ 0x5f5e100
302 #define XPAR_UART1_CTS_OVERRIDE 0x1
303 #define XPAR_UART1_CLOCKS 0x18
304 #define XPAR_UART1_INTERRUPTS 0x4032
305 #define XPAR_UART1_INTERRUPT_PARENT 0xf8f01000
306
307 /* Canonical definitions for peripheral UART1 */
308 #define XPAR_XUARTPS_1_BASEADDR 0xe0001000
309 #define XPAR_XUARTPS_1_HIGHADDR 0xe0001fff
310 #define XPAR_XUARTPS_1_COMPATIBLE "xlnx,xuartps"
311 #define XPAR_XUARTPS_1_CLOCK_FREQ 0x5f5e100
312 #define XPAR_XUARTPS_1_CTS_OVERRIDE 0x1
313 #define XPAR_XUARTPS_1_CLOCKS 0x18
314 #define XPAR_XUARTPS_1_INTERRUPTS 0x4032
315 #define XPAR_XUARTPS_1_INTERRUPT_PARENT 0xf8f01000
316
317 #define XPAR_XXADCPS_NUM_INSTANCES 1
318
319 /* Definitions for peripheral ADC */
320 #define XPAR_ADC_COMPATIBLE "xlnx,zynq-xadc-1.00.a"
321 #define XPAR_ADC_BASEADDR 0xf8007100
322 #define XPAR_ADC_HIGHADDR 0xf800711f
323
324 /* Canonical definitions for peripheral ADC */
325 #define XPAR_XXADCPS_0_BASEADDR 0xf8007100
326 #define XPAR_XXADCPS_0_HIGHADDR 0xf800711f
327 #define XPAR_XXADCPS_0_COMPATIBLE "xlnx,zynq-xadc-1.00.a"

```

```

328
329     /* Definitions for peripheral L2 */
330     #define XPAR_L2_BASEADDR 0xf8f02000
331     #define XPAR_L2_HIGHADDR 0xf8f02ffff
332
333     /* Canonical definitions for peripheral L2 */
334     #define XPAR_PS7_PL310_0_BASEADDR 0xf8f02000
335     #define XPAR_PS7_PL310_0_HIGHADDR 0xf8f02ffff
336
337     /* Definitions for peripheral MC */
338     #define XPAR_MC_BASEADDR 0xf8006000
339     #define XPAR_MC_HIGHADDR 0xf8006ffff
340
341     /* Canonical definitions for peripheral MC */
342     #define XPAR_PS7_DDRC_0_BASEADDR 0xf8006000
343     #define XPAR_PS7_DDRC_0_HIGHADDR 0xf8006ffff
344
345     /* Definitions for peripheral SLCR */
346     #define XPAR_SLCR_BASEADDR 0xf8000000
347     #define XPAR_SLCR_HIGHADDR 0xf8000ffff
348
349     /* Canonical definitions for peripheral SLCR */
350     #define XPAR_PS7_SLCR_0_BASEADDR 0xf8000000
351     #define XPAR_PS7_SLCR_0_HIGHADDR 0xf8000ffff
352
353     /* Definitions for peripheral GLOBAL_TIMER */
354     #define XPAR_GLOBAL_TIMER_BASEADDR 0xf8f00200
355     #define XPAR_GLOBAL_TIMER_HIGHADDR 0xf8f0021f
356
357     /* Canonical definitions for peripheral GLOBAL_TIMER */
358     #define XPAR_PS7_GLOBALTIMER_0_BASEADDR 0xf8f00200
359     #define XPAR_PS7_GLOBALTIMER_0_HIGHADDR 0xf8f0021f
360
361     /* Definitions for peripheral PS7_PMU_0 */
362     #define XPAR_PS7_PMU_0_BASEADDR 0xf8891000
363     #define XPAR_PS7_PMU_0_HIGHADDR 0xf8891ffff
364
365     /* Canonical definitions for peripheral PS7_PMU_0 */
366     #define XPAR_PS7_PMU_0_BASEADDR 0xf8891000
367     #define XPAR_PS7_PMU_0_HIGHADDR 0xf8891ffff
368
369     /* Definitions for peripheral PS7_OCMC_0 */
370     #define XPAR_PS7_OCMC_0_BASEADDR 0xf800c000
371     #define XPAR_PS7_OCMC_0_HIGHADDR 0xf800cfffc
372
373     /* Canonical definitions for peripheral PS7_OCMC_0 */
374     #define XPAR_PS7_OCMC_0_BASEADDR 0xf800c000
375     #define XPAR_PS7_OCMC_0_HIGHADDR 0xf800cfffc
376
377     /* Definitions for peripheral PS7_GPV_0 */
378     #define XPAR_PS7_GPV_0_BASEADDR 0xf8900000
379     #define XPAR_PS7_GPV_0_HIGHADDR 0xf89fffff
380
381     /* Canonical definitions for peripheral PS7_GPV_0 */
382     #define XPAR_PS7_GPV_0_BASEADDR 0xf8900000
383     #define XPAR_PS7_GPV_0_HIGHADDR 0xf89fffff
384
385     /* Definitions for peripheral PS7_SCUC_0 */
386     #define XPAR_PS7_SCUC_0_BASEADDR 0xf8f00000
387     #define XPAR_PS7_SCUC_0_HIGHADDR 0xf8f000fc
388
389     /* Canonical definitions for peripheral PS7_SCUC_0 */
390     #define XPAR_PS7_SCUC_0_BASEADDR 0xf8f00000
391     #define XPAR_PS7_SCUC_0_HIGHADDR 0xf8f000fc
392
393     /* Definitions for peripheral PS7_IOP_BUS_CONFIG_0 */
394     #define XPAR_PS7_IOP_BUS_CONFIG_0_BASEADDR 0xe0200000

```

```

395 #define XPAR_PS7_IOP_BUS_CONFIG_0_HIGHADDR 0xe0200fff
396
397 /* Canonical definitions for peripheral PS7_IOP_BUS_CONFIG_0 */
398 #define XPAR_PS7_IOP_BUS_CONFIG_0_BASEADDR 0xe0200000
399 #define XPAR_PS7_IOP_BUS_CONFIG_0_HIGHADDR 0xe0200fff
400
401 /* Definitions for peripheral PS7_DMA_NS */
402 #define XPAR_PS7_DMA_NS_BASEADDR 0xf8004000
403 #define XPAR_PS7_DMA_NS_HIGHADDR 0xf8004fff
404
405 /* Canonical definitions for peripheral PS7_DMA_NS */
406 #define XPAR_PS7_DMA_0_BASEADDR 0xf8004000
407 #define XPAR_PS7_DMA_0_HIGHADDR 0xf8004fff
408
409 /* Definitions for peripheral PS7_AFI_0 */
410 #define XPAR_PS7_AFI_0_BASEADDR 0xf8008000
411 #define XPAR_PS7_AFI_0_HIGHADDR 0xf8008fff
412
413 /* Canonical definitions for peripheral PS7_AFI_0 */
414 #define XPAR_PS7_AFI_0_BASEADDR 0xf8008000
415 #define XPAR_PS7_AFI_0_HIGHADDR 0xf8008fff
416
417 /* Definitions for peripheral PS7_AFI_1 */
418 #define XPAR_PS7_AFI_1_BASEADDR 0xf8009000
419 #define XPAR_PS7_AFI_1_HIGHADDR 0xf8009fff
420
421 /* Canonical definitions for peripheral PS7_AFI_1 */
422 #define XPAR_PS7_AFI_1_BASEADDR 0xf8009000
423 #define XPAR_PS7_AFI_1_HIGHADDR 0xf8009fff
424
425 /* Definitions for peripheral PS7_AFI_2 */
426 #define XPAR_PS7_AFI_2_BASEADDR 0xf800a000
427 #define XPAR_PS7_AFI_2_HIGHADDR 0xf800afff
428
429 /* Canonical definitions for peripheral PS7_AFI_2 */
430 #define XPAR_PS7_AFI_2_BASEADDR 0xf800a000
431 #define XPAR_PS7_AFI_2_HIGHADDR 0xf800afff
432
433 /* Definitions for peripheral PS7_AFI_3 */
434 #define XPAR_PS7_AFI_3_BASEADDR 0xf800b000
435 #define XPAR_PS7_AFI_3_HIGHADDR 0xf800bfff
436
437 /* Canonical definitions for peripheral PS7_AFI_3 */
438 #define XPAR_PS7_AFI_3_BASEADDR 0xf800b000
439 #define XPAR_PS7_AFI_3_HIGHADDR 0xf800bfff
440
441 /* BOARD definition */
442 #define XPS_BOARD_PYNQ-Z1
443
444 #define XPAR_PS7_DDR_0_BASEADDRESS 0x100000
445 #define XPAR_PS7_DDR_0_HIGHADDRESS 0x1fffffff
446 #define XPAR_PS7_QSPI_LINEAR_0_BASEADDRESS 0xfc000000
447 #define XPAR_PS7_QSPI_LINEAR_0_HIGHADDRESS 0xfcffffff
448 #define XPAR_PS7_RAM_0_BASEADDRESS 0x0
449 #define XPAR_PS7_RAM_0_HIGHADDRESS 0x2ffff
450 #define XPAR_PS7_RAM_1_BASEADDRESS 0xfffff0000
451 #define XPAR_PS7_RAM_1_HIGHADDRESS 0xfffffdff
452
453 #define XPAR_CPU_CORE_CLOCK_FREQ_HZ 650000000
454
455 /* Number of SLRs */
456 #define NUMBER_OF_SLRS 0x1
457
458 /* Device ID */
459 #define XPAR_DEVICE_ID "7z020"
460
461 #endif /* end of protection macro */

```

ANEXO G. ARCHIVO MAIN.C DE LA APLICACIÓN DEL SUBSISTEMA TMR

```
1 #include <stdio.h>
2 #include "xparameters.h"
3 #include "xgpio.h"
4 #include "sleep.h"
5 #include "xil_printf.h"
6 #include "xuartns550.h"
7 #include "xtmrctr.h"
8
9
10 int main(void) {
11
12     XGpio sws, btns, leds; // Declare XGpio objects
13     XTmrCtr timer; // Declare a timer object
14     XUartNs550 uart; // Declare a UART object
15
16     // Initialize the UART
17     XUartNs550_Initialize(&uart, XPAR_XUARTNS550_0_BASEADDR);
18     // Set data format to 8 bits, no parity, 1 stop bit (8N1)
19     XUartNs550Format format;
20     format.BaudRate = 115200; // Set baud rate to 115200
21     format.DataBits = XUN_FORMAT_8_BITS;
22     format.Parity = XUN_FORMAT_NO_PARITY;
23     format.StopBits = XUN_FORMAT_1_STOP_BIT;
24     XUartNs550_SetDataFormat(&uart, &format);
25     // Enable FIFOs, resetting both TX and RX FIFOs
26     XUartNs550_SetOptions(&uart, XUN_OPTION_FIFOS_ENABLE | XUN_OPTION_RESET_TX_FIFO |
27                           XUN_OPTION_RESET_RX_FIFO);
28
29     xil_printf("Initializing...\r\n");
30
31     char input_buffer[128];
32     unsigned int input_index = 0;
33     unsigned int mode = 0;
34     unsigned int led_seq_index = 0; // Index for led sequence in mode1
35     unsigned int now; // Variable to store the current tick of the timer
36     unsigned int last_tick = 0; // Variable to store the previously checked tick of
37                               // the timer
38
39     int sws_check, btns_check; // Declare variables to store the values of the
40                               // switches and buttons
41
42     // Initialize sws (XPAR_XGPIO_2_BASEADDR is defined in xparameters.h)
43     XGpio_Initialize(&sws, XPAR_XGPIO_2_BASEADDR);
44     // Note: DirectionMask is a 32-bit mask where each bit represents the direction of
45     //       the corresponding pin
46     // 0: output, 1: input
47     // Set all bits to input (we could also use 0x3, 11 in binary, for 2-bit input, since
48     // XPAR_AXI_GPIO_SWS_GPIO_WIDTH is 0x2)
49     XGpio_SetDataDirection(&sws, 1, 0xFFFFFFFF);
50
51     // Initialize btns
52     XGpio_Initialize(&btns, XPAR_XGPIO_0_BASEADDR);
53     XGpio_SetDataDirection(&btns, 1, 0xFFFFFFFF);
54
55     // Initialize leds
56     XGpio_Initialize(&leds, XPAR_XGPIO_1_BASEADDR);
57     XGpio_SetDataDirection(&leds, 1, 0x0); // Set all bits to output
58
59     // Initialize the timer
```

```

55 XTmrCtr_Initialize(&timer, XPAR_XTMRCTR_0_BASEADDR);
56 XTmrCtr_SetOptions(&timer, 0, XTC_AUTO_RELOAD_OPTION); // Set timer to auto-reload
57 XTmrCtr_SetOptions(&timer, 1, XTC_AUTO_RELOAD_OPTION); // Set second timer to auto-
      reload
58 XTmrCtr_SetResetValue(&timer, 0, 0); // Reset timer 0
59 XTmrCtr_SetResetValue(&timer, 1, 0); // Reset timer 1
60 XTmrCtr_Start(&timer, 0); // Start timer 0
61 XTmrCtr_Start(&timer, 1); // Start timer 1
62
63
64 xil_printf("Ready\r\n");
65 // Flash the LEDs to indicate that the program is running
66 for (int i = 0; i < 5; i++) {
67     XGpio_DiscreteWrite(&leds, 1, 0xF); // Turn on all LEDs
68     usleep(100000); // Sleep for 100ms
69     XGpio_DiscreteWrite(&leds, 1, 0x0); // Turn off all LEDs
70     usleep(100000); // Sleep for 100ms
71 }
72
73 while (1) {
74     // Non-blocking UART read
75     if (XUartNs550_IsReceiveData(XPAR_XUARTNS550_0_BASEADDR)) {
76         char c = XUartNs550_RecvByte(XPAR_XUARTNS550_0_BASEADDR);
77
78         // Check for Enter key (carriage return or newline)
79         if (c == '\r' || c == '\n') {
80             input_buffer[input_index] = '\0'; // Null-terminate the string
81             // Process the input
82             if (input_index == 0) {
83                 xil_printf("\r\nNo input received. Current mode: %d\r\n", mode);
84             } else {
85                 // Check if the input is a valid mode ("mode0" or "mode1")
86                 if (strcmp(input_buffer, "mode0") == 0) {
87                     input_index = 0; // Reset buffer for next input
88                     if (mode == 0) {
89                         xil_printf("\r\nAlready in mode 0\r\n");
90                         continue;
91                     }
92
93                     mode = 0; // Set mode to 0
94
95                 } else if (strcmp(input_buffer, "mode1") == 0) {
96                     input_index = 0; // Reset buffer for next input
97                     if (mode == 1) {
98                         xil_printf("\r\nAlready in mode 1\r\n");
99                         continue;
100
101                     }
102                     mode = 1; // Set mode to 1
103                     led_seq_index = 0; // Reset the sequence
104                 } else {
105                     xil_printf("\r\nInvalid Input. Please enter 'mode0' or 'mode1'.\r\
106                         n\t*Mode 0: Control LEDs with switches\r\n\t*Mode
107                         1: LEDs blink in sequence*\r\n");
108                     input_index = 0; // Reset buffer for next input
109                     continue;
110                 }
111             }
112             // Handle Backspace (ASCII 8 or 127)
113             // NOTE: Only works for ascii characters
114             else if ((c == 8 || c == 127) && input_index > 0) {
115                 input_index--;
116                 // Erase character on terminal (backspace, space, backspace)
117                 // The space is important to overwrite the last character
118                 // If not, only the cursor is moved one space back, but the character
119                 // remains

```

```

118         xil_printf("\b \b");
119     }
120
121     else if (c >= 32 && c < 127 && (input_index < sizeof(input_buffer) - 1)) {
122         input_buffer[input_index++] = c; // Store character in buffer
123     }
124
125     else if (input_index >= sizeof(input_buffer) - 1) {
126         // Full buffer
127         xil_printf("\r\nInput buffer full! Press Enter.\r\n");
128     }
129
130 }
131
132 if (mode == 0) {
133     // Read the values of the switches and buttons
134     sws_check = XGpio_DiscreteRead(&sws, 1);
135     btns_check = XGpio_DiscreteRead(&btns, 1);
136     // If btns_check is 0x0 (000), no buttons are pressed
137     // If btns_check is 0x1 (001), the first button is pressed
138     // If btns_check is 0x2 (010), the second button is pressed
139     // If btns_check is 0x4 (101), the third button is pressed
140     // For combinations of buttons, the value of btns_check will be the sum of the
141     // values of the pressed buttons
142
143     // If both switches are on, turn on all LEDs
144     if (sws_check == 0x3) {
145         XGpio_DiscreteWrite(&leds, 1, 0xF);
146     }
147     // If both switches are off, turn off all LEDs
148     else if (sws_check == 0x0) {
149         XGpio_DiscreteWrite(&leds, 1, 0x0);
150     }
151     // Else, turn on the LED that corresponds to the button being pressed
152     else {
153         // Shift 1 bit to the left the buttons pressed, since the LED 0 does not
154         // have a corresponding button
155         XGpio_DiscreteWrite(&leds, 1, (btns_check << 1) & 0xF); // Mask to 4 bits
156     }
157 }
158
159 else if (mode == 1) {
160     // Turns the 4 LEDs in sequence, with a delay of 500ms between each LED
161     // Non-blocking LED sequence using the timer
162     now = XTmrCtr_GetValue(&timer, 0); // Read current timer value
163     // 25,000,000 ticks = 500ms at 50MHz (frequency at which the timer is run)
164     if ((now - last_tick) > 25000000) {
165         // Shift the value 1 '(3 - led_seq_index)' times to the left to start the
166         // sequence from LED 3
167         XGpio_DiscreteWrite(&leds, 1, 1 << (3 - led_seq_index));
168         // xil_printf("\r\n%d\r\n", 3 - led_seq_index); // Optionally print the
169         // current LED
170         led_seq_index = (led_seq_index + 1) % 4;
171         last_tick = now;
172     }
173
174 else {
175     // If the mode is not 0 or 1, print an error message
176     xil_printf("\r\nError: An invalid mode was enabled. Defaulting to mode 0.\r\n"
177     );
178     mode = 0;
179 }
180
181 }
```

```
179 |     return 0;  
180 | }
```

ANEXO H. ARCHIVO XPARAMETERS.H DEL DOMINIO DEL MICROBLAZE-V

```
1 #ifndef XPARAMETERS_H      /* prevent circular inclusions */
2 #define XPARAMETERS_H      /* by using protection macros */
3
4 #define XPAR_XGPIO_NUM_INSTANCES 3
5
6 /* Definitions for peripheral TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS */
7 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS_COMPATIBLE "xlnx,axi-gpio-2.0"
8 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS_BASEADDR 0x40000000
9 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS_HIGHADDR 0x4000ffff
10 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS_INTERRUPT_PRESENT 0x0
11 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS_IS_DUAL 0x0
12 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS_GPIO_WIDTH 0x3
13
14 /* Canonical definitions for peripheral TMR_SYSTEM_MB1_AXI_GPIO_BTNS_3BITS */
15 #define XPAR_XGPIO_0_BASEADDR 0x40000000
16 #define XPAR_XGPIO_0_HIGHADDR 0x4000ffff
17 #define XPAR_XGPIO_0_COMPATIBLE "xlnx,axi-gpio-2.0"
18 #define XPAR_XGPIO_0_GPIO_WIDTH 0x3
19 #define XPAR_XGPIO_0_INTERRUPT_PRESENT 0x0
20 #define XPAR_XGPIO_0_IS_DUAL 0x0
21
22 /* Definitions for peripheral TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS */
23 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS_COMPATIBLE "xlnx,axi-gpio-2.0"
24 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS_BASEADDR 0x40010000
25 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS_HIGHADDR 0x4001ffff
26 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS_INTERRUPT_PRESENT 0x0
27 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS_IS_DUAL 0x0
28 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS_GPIO_WIDTH 0x4
29
30 /* Canonical definitions for peripheral TMR_SYSTEM_MB1_AXI_GPIO_LEDS_4BITS */
31 #define XPAR_XGPIO_1_BASEADDR 0x40010000
32 #define XPAR_XGPIO_1_HIGHADDR 0x4001ffff
33 #define XPAR_XGPIO_1_COMPATIBLE "xlnx,axi-gpio-2.0"
34 #define XPAR_XGPIO_1_GPIO_WIDTH 0x4
35 #define XPAR_XGPIO_1_INTERRUPT_PRESENT 0x0
36 #define XPAR_XGPIO_1_IS_DUAL 0x0
37
38 /* Definitions for peripheral TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS */
39 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS_COMPATIBLE "xlnx,axi-gpio-2.0"
40 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS_BASEADDR 0x40020000
41 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS_HIGHADDR 0x4002ffff
42 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS_INTERRUPT_PRESENT 0x0
43 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS_IS_DUAL 0x0
44 #define XPAR_TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS_GPIO_WIDTH 0x2
45
46 /* Canonical definitions for peripheral TMR_SYSTEM_MB1_AXI_GPIO_SWS_2BITS */
47 #define XPAR_XGPIO_2_BASEADDR 0x40020000
48 #define XPAR_XGPIO_2_HIGHADDR 0x4002ffff
49 #define XPAR_XGPIO_2_COMPATIBLE "xlnx,axi-gpio-2.0"
50 #define XPAR_XGPIO_2_GPIO_WIDTH 0x2
51 #define XPAR_XGPIO_2_INTERRUPT_PRESENT 0x0
52 #define XPAR_XGPIO_2_IS_DUAL 0x0
53
54 #define XPAR_XINTC_NUM_INSTANCES 1
55
56 /* Definitions for peripheral TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC */
57 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_COMPATIBLE "xlnx,axi-intc-4.1"
58 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_BASEADDR 0x41200000
59 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_HIGHADDR 0x4120ffff
```

```

60 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_KIND_OF_INTR 0x0
61 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_IS_FAST 0x1
62 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_IVAR_RST_VAL 0x10
63 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_NUM_INTR_INPUTS 0x3
64 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_ADDR_WIDTH 0x20
65 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_OPTIONS 0x0
66 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_INTCTYPE 0x0
67 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_HANDLER_TABLE 0x0
68 #define XPAR_TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC_NUM_SW_INTR 0x0
69
70 /* Canonical definitions for peripheral TMR_SYSTEM_MB3_MICROBLAZE_RISCV_0_AXI_INTC */
71 #define XPAR_XINTC_0_BASEADDR 0x41200000
72 #define XPAR_XINTC_0_HIGHADDR 0x4120ffff
73 #define XPAR_XINTC_0_HANDLER_TABLE 0x0
74 #define XPAR_XINTC_0_ADDR_WIDTH 0x20
75 #define XPAR_XINTC_0_COMPATIBLE "xlnx,axi-intc-4.1"
76 #define XPAR_XINTC_0_IS_FAST 0x1
77 #define XPAR_XINTC_0_IVAR_RST_VAL 0x10
78 #define XPAR_XINTC_0_INTCTYPE 0x0
79 #define XPAR_XINTC_0_KIND_OF_INTR 0x0
80 #define XPAR_XINTC_0_NUM_INTR_INPUTS 0x3
81 #define XPAR_XINTC_0_NUM_SW_INTR 0x0
82 #define XPAR_XINTC_0_OPTIONS 0x0
83
84 #define XPAR_XTMRCTR_NUM_INSTANCES 1
85
86 /* Definitions for peripheral TMR_SYSTEM_MB1_AXI_TIMER_0 */
87 #define XPAR_TMR_SYSTEM_MB1_AXI_TIMER_0_COMPATIBLE "xlnx,axi-timer-2.0"
88 #define XPAR_TMR_SYSTEM_MB1_AXI_TIMER_0_BASEADDR 0x41c00000
89 #define XPAR_TMR_SYSTEM_MB1_AXI_TIMER_0_HIGHADDR 0x41c0ffff
90 #define XPAR_TMR_SYSTEM_MB1_AXI_TIMER_0_CLOCK_FREQUENCY 0x2faf080
91
92 /* Canonical definitions for peripheral TMR_SYSTEM_MB1_AXI_TIMER_0 */
93 #define XPAR_XTMRCTR_0_BASEADDR 0x41c00000
94 #define XPAR_XTMRCTR_0_HIGHADDR 0x41c0ffff
95 #define XPAR_XTMRCTR_0_COMPATIBLE "xlnx,axi-timer-2.0"
96 #define XPAR_XTMRCTR_0_CLOCK_FREQUENCY 0x2faf080
97
98 #define XPAR_XUARTNS550_NUM_INSTANCES 1
99
100 /* Definitions for peripheral AXI_UART16550_TMR */
101 #define XPAR_AXI_UART16550_TMR_COMPATIBLE "xlnx,axi-uart16550-2.0"
102 #define XPAR_AXI_UART16550_TMR_BASEADDR 0x44a00000
103 #define XPAR_AXI_UART16550_TMR_HIGHADDR 0x44a0ffff
104 #define XPAR_AXI_UART16550_TMR_CLOCK_FREQ 0x2faf080
105 #define XPAR_AXI_UART16550_TMR_CURRENT_SPEED 0x0
106
107 /* Canonical definitions for peripheral AXI_UART16550_TMR */
108 #define XPAR_XUARTNS550_0_BASEADDR 0x44a00000
109 #define XPAR_XUARTNS550_0_HIGHADDR 0x44a0ffff
110 #define XPAR_XUARTNS550_0_COMPATIBLE "xlnx,axi-uart16550-2.0"
111 #define XPAR_XUARTNS550_0_CLOCK_FREQ 0x2faf080
112 #define XPAR_XUARTNS550_0_CURRENT_SPEED 0x0
113
114 /* BOARD definition */
115 #define XPS_BOARD_PYNQ-Z1
116
117 #define XPAR_LMB_BRAM_0_BASEADDRESS 0x0
118 #define XPAR_LMB_BRAM_0_HIGHADDRESS 0x7fff
119
120 /* CPU parameters definition */
121 #define XPAR_CPU_CORE_CLOCK_FREQ_HZ 50000000
122 #define XPAR_MICROBLAZE_RISCV_USE_DCACHE 0
123 #define XPAR_MICROBLAZE_RISCV_DCACHE_LINE_LEN 4
124 #define XPAR_MICROBLAZE_RISCV_DCACHE_BYTE_SIZE 4096
125 #define XPAR_MICROBLAZE_RISCV_USE_ICACHE 0
126 #define XPAR_MICROBLAZE_RISCV_ICACHE_LINE_LEN 4

```

```
127 #define XPAR_MICROBLAZE_RISCV_ICACHE_BYTE_SIZE 4096
128 #define XPAR_MICROBLAZE_RISCV_USE_FPU 0
129 #define XPAR_MICROBLAZE_RISCV_USE_MMU 0
130 #define XPAR_MICROBLAZE_RISCV_USE_SLEEP 4
131 #define XPAR_MICROBLAZE_RISCV_FAULT_TOLERANT 0
132 #define XPAR_MICROBLAZE_RISCV_D_LMB 1
133 #define XPAR_MICROBLAZE_RISCV_USE_BRANCH_TARGET_CACHE 0
134 #define XPAR_MICROBLAZE_RISCV_BRANCH_TARGET_CACHE_SIZE 0
135
136 /* Number of SLRs */
137 #define NUMBER_OF_SLRS 0x1
138
139 /* Device ID */
140 #define XPAR_DEVICE_ID "7z020"
141
142 #endif /* end of protection macro */
```

ANEXO I. ESTRUCTURA DE LOS FRAMES QUE CONFIGURAN LA TILE DSP_R_X25Y95

ANEXO J. ARCHIVO TILE2INJECTION.PY

```
1      '''
2      Takes a tile or site name, the part name and the configuration bus
3      and returns all the possible error injection commands for the Tile
4      using the Soft Error Mitigation Controller v4.1 LogicORE IP
5      '''
6
7      import argparse
8      import json
9      import os
10     import sys
11
12    def main():
13        parser = argparse.ArgumentParser(
14            description='Generate error injection commands to use in Soft Error Mitigation
15            Controller LogicORE IP for a given tile')
16        parser.add_argument('tile_or_site', type=str, help='Tile or site name. If a site is
17            provided, the output will be based on its parent tile')
18        parser.add_argument('part', type=str, help='Part name')
19        parser.add_argument('--config_bus', type=str, required=False,
20                            default=None, # CLB_IO_CLK, BLOCK_RAM or CFG_CLB (If None, the
21                            first in the JSON entry will be used)
22                            help='Configuration bus (also known as block type) of the tile or
23                            site'
24                            )
25        parser.add_argument('--tilegrid', type=str, required=False,
26                            default=None,
27                            help='Path to tilegrid.json file')
28        parser.add_argument('--mask', type=str, required=False,
29                            default=None,
30                            help='Tile\'s mask file')
31        parser.add_argument('--out', type=str, required=False,
32                            default=None,
33                            help='Output file to write the error injection commands. If not
34                            specified, it will print to stdout')
35        parser.add_argument('--verbose', action='store_true', # action='store_true' makes
36                            args.verbose True iff the flag is present
37                            help='Print verbose output')
38
39    args = parser.parse_args()
40    tile_or_site = args.tile_or_site
41    part = args.part
42    config_bus = args.config_bus
43    tilegrid_path = args.tilegrid
44    mask_path = args.mask
45    out_path = args.out
46    verbose = args.verbose
47
48    # print the arguments
49    if verbose:
50        print(f'Tile or site: {tile_or_site}')
51        print(f'Part: {part}')
52        print(f'Config bus: {config_bus}')
53        print(f'Tilegrid: {tilegrid_path}')
54        print(f'Mask file: {mask_path}')
55        print(f'Verbose: {verbose}')
56
57    # Check that the part name starts by any off the following prefixes:
58    part_prefixes = {'artix7': ['xc7a100t', 'xc7a200t', 'xc7a50t'],
59                     'kintex7': ['xc7k70t'],
60                     'spartan7': ['xc7s50'],
61                     'zynq7': ['xc7z010', 'xc7z020']}
```

```

56         }
57
58     for family, prefixes in part_prefixes.items():
59         for prefix in prefixes:
60             if part.startswith(prefix):
61                 part = prefix
62             if not tilegrid_path:
63                 tilegrid_path = os.path.join(
64                     os.path.dirname(__file__), '...', 'database', family, part, 'tilegrid.json')
65                 part_family = family
66             break
67         else:
68             continue
69         break
70     else:
71         raise ValueError(f'Part {part} does not match any known prefix: {part_prefixes}')
72     # Print the tilegrid path
73     if verbose:
74         print(f'Tilegrid path: {tilegrid_path}')
75     # Check if the tilegrid file exists
76     if not os.path.exists(tilegrid_path):
77         raise FileNotFoundError(f'Tilegrid file {tilegrid_path} does not exist')
78     # Load the tilegrid file
79     with open(tilegrid_path, 'r') as f:
80         tilegrid = json.load(f)
81
82     # Try to match the tile or site in the tilegrid
83     tile_entry = None
84
85     # Check if the input matches a tile name
86     if tile_or_site in tilegrid:
87         tile_entry = tilegrid[tile_or_site]
88         if verbose:
89             print(f"Matched as tile: {tile_or_site}")
90     else:
91         # Otherwise, search for a matching site name
92         for tile_name, tile_data in tilegrid.items():
93             sites = tile_data.get("sites", {})
94             if tile_or_site in sites:
95                 tile_entry = tile_data
96                 if verbose:
97                     print(f"Matched as site: {tile_or_site} in tile: {tile_name}")
98                 tile_or_site = tile_name # Overwrite tile_or_site to the tile name, which
99                             # is the useful identifier
100                break
101
102    if tile_entry is None:
103        raise ValueError(f"{tile_or_site} not found as tile or site in tilegrid.")
104
105    # Get the relevant values from the tile entry
106    bits = tile_entry.get("bits", {})
107    # Determine config bus to use
108    if not config_bus:
109        # Use the first available config bus if not specified or invalid
110        if bits:
111            config_bus = next(iter(bits))
112        else:
113            raise ValueError(f"No configuration bus found for tile {tile_or_site}")
114    elif config_bus not in bits:
115        raise ValueError(f"Configuration bus {config_bus} not found in tile {tile_or_site}")
116
117    bus_info = bits[config_bus]
118    base_addr = int(bus_info["baseaddr"], 0)
119    frames = int(bus_info["frames"])
120    offset = int(bus_info["offset"])

```

```

120     words = int(bus_info["words"])
121     tile_type = tile_entry.get("type", "")
122
123     if verbose:
124         print(f"Config bus: {config_bus}")
125         print(f"Base address: 0x{base_addr:08X}")
126         print(f"Frames: {frames}")
127         print(f"Offset: {offset}")
128         print(f"Words: {words}")
129         print(f"Tile type: {tile_type}")
130
131     # Check if the mask file is provided, otherwise use the default mask file
132     if not mask_path:
133         # If the config_bus is CLB_IO_CLK, we simply look for mask_<tile_type>.db
134         if config_bus == "CLB_IO_CLK":
135             mask_path = os.path.join(
136                 os.path.dirname(__file__), '...', 'database', part_family, f'mask_{{
137                     tile_type.lower()}.db'}
138             # Otherwise, we look for mask_<tile_type>.<config_bus>.db
139             else:
140                 mask_path = os.path.join(
141                     os.path.dirname(__file__), '...', 'database', part_family, f'mask_{{
142                         tile_type.lower()}{config_bus.lower()}.db')
143             if not os.path.exists(mask_path):
144                 raise FileNotFoundError(f"Mask file {mask_path} does not exist")
145
146             if verbose:
147                 print(f"Using mask file: {mask_path}")
148
149             # Read the mask file
150             mask_dict = {}
151             with open(mask_path, 'r') as f:
152                 for line in f:
153                     line = line.strip()
154                     if not line or not line.startswith("bit "):
155                         continue
156                     _, bitstr = line.split()
157                     left, right = bitstr.split("_")
158                     left = int(left)
159                     right = int(right)
160                     if left not in mask_dict:
161                         mask_dict[left] = []
162                         mask_dict[left].append(right)
163             if verbose:
164                 print(f"Mask file loaded with {len(mask_dict)} entries.")
165
166             # Generate the error injection commands
167             # Open or create the output file if specified, otherwise use stdout
168             if out_path:
169                 out_file = open(out_path, 'w')
170             else:
171                 out_file = sys.stdout
172             try:
173                 frame_count = 0
174                 for minor_addr, bit_positions in mask_dict.items():
175                     if frame_count >= frames:    # Not needed in principle, but just in case
176                         break
177                     frame_count += 1
178                     for bit_position in bit_positions:
179                         word_addr = offset + bit_position // 32
180                         bit_addr = bit_position % 32
181                         if bit_position // 32 > words - 1: # The word referenced by the bit position
182                             is out of range
183                             continue
184                         # Pack the fields into a 40-bit value
185                         value = (
186                             (0 << 39) |                                # bit 39: always 0

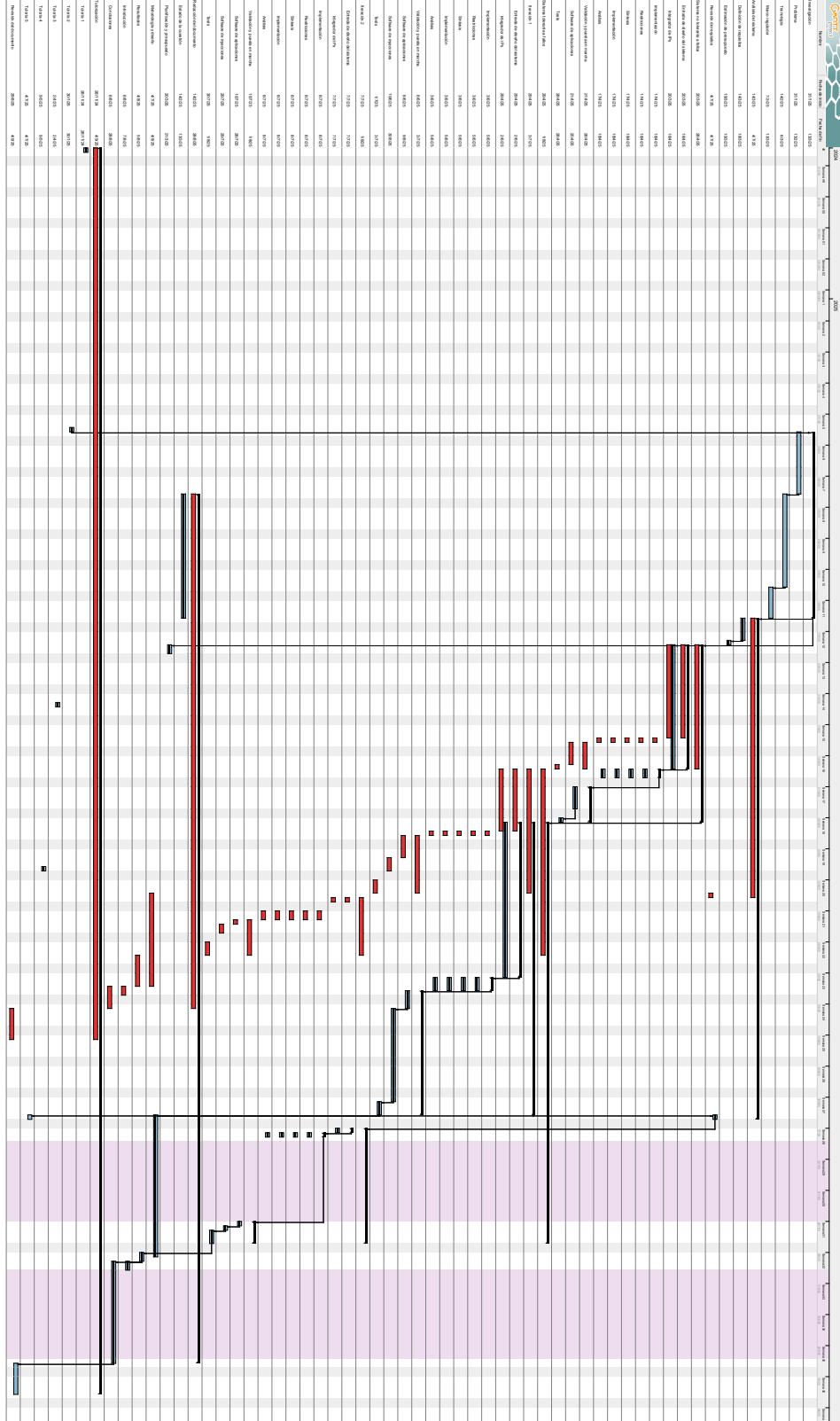
```

```
184     (0 << 37) |                               # bits 38:37: 00 for non-SSI (the
185         case for all the listed families)
186     (((base_addr >> 7) & 0x3FFF) << 19) | # bits 36:19: base_addr bits 25:7
187     ((minor_addr & 0x7F) << 12) |           # bits 18:12: minor_addr (7 bits)
188     ((word_addr & 0x7F) << 5) |             # bits 11:5: word_addr (7 bits)
189     (bit_addr & 0x1F)                         # bits 4:0: bit_addr (5 bits)
190
191     )
192     # Output as 10-digit hex (40 bits)
193     out_file.write(f"N {value:010X}\n")
194
195     if out_file is not sys.stdout:
196         out_file.close()
197
198     except Exception as e:
199         if out_file is not sys.stdout:
200             out_file.close()
201             raise e
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
```

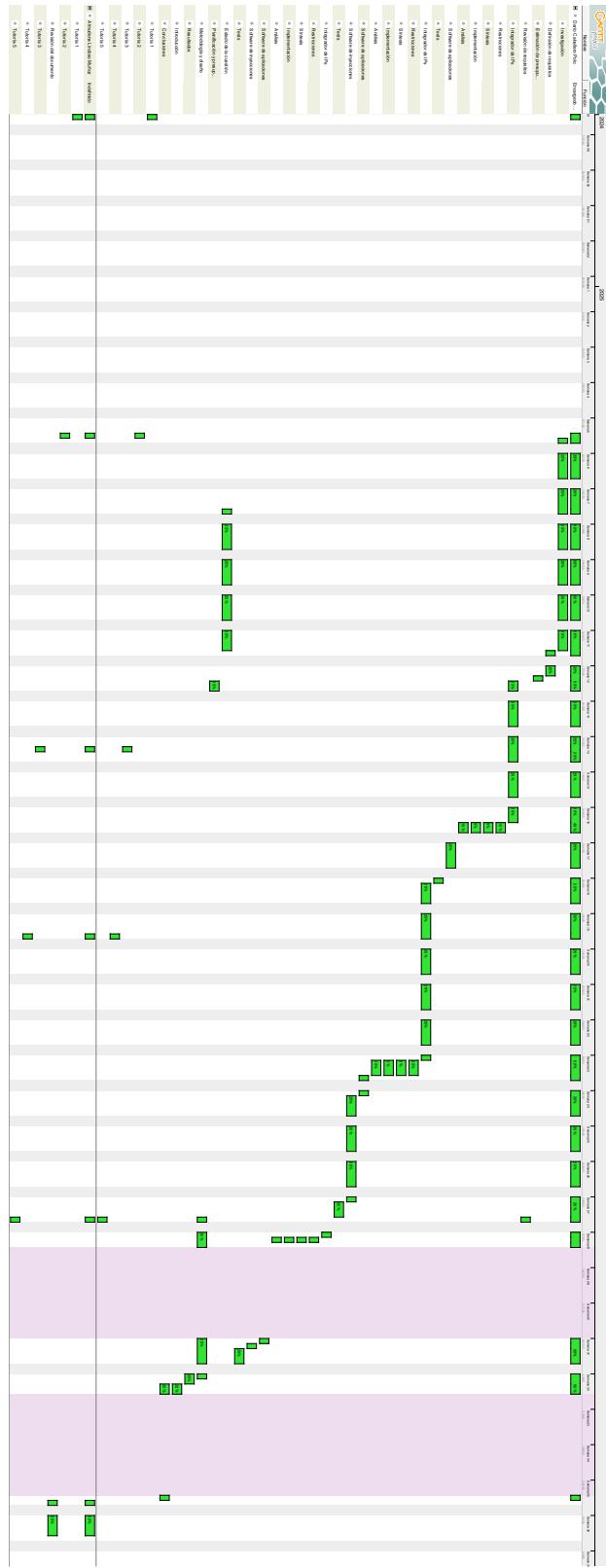
ANEXO K. TAREAS

Nombre	Fecha de inicio	Fecha de fin
Investigación	31/1/25	13/3/25
Problema	31/1/25	13/2/25
Tecnología	14/2/25	6/3/25
Marco regulador	7/3/25	13/3/25
Análisis del sistema	14/3/25	4/7/25
Definición de requisitos	14/3/25	18/3/25
Estimación de presupuesto	19/3/25	19/3/25
Revisión de requisitos	4/7/25	4/7/25
Sistema no tolerante a fallos	20/3/25	28/4/25
Entrada de diseño del sistema	20/3/25	16/4/25
Integrador de IPs	20/3/25	16/4/25
Implementación	17/4/25	18/4/25
Restricciones	17/4/25	18/4/25
Síntesis	17/4/25	18/4/25
Implementación	17/4/25	18/4/25
Análisis	17/4/25	18/4/25
Validación y puesta en marcha	21/4/25	28/4/25
Software de aplicaciones	21/4/25	25/4/25
Tests	28/4/25	28/4/25
Sistema tolerante a fallos	29/4/25	1/8/25
Iteración 1	29/4/25	3/7/25
Entrada de diseño del sistema	29/4/25	2/6/25
Integrador de IPs	29/4/25	2/6/25
Implementación	3/6/25	5/6/25
Restricciones	3/6/25	5/6/25
Síntesis	3/6/25	5/6/25
Implementación	3/6/25	5/6/25
Análisis	3/6/25	5/6/25
Validación y puesta en marcha	6/6/25	3/7/25
Software de aplicaciones	6/6/25	9/6/25
Software de inyecciones	10/6/25	30/6/25
Tests	1/7/25	3/7/25
Iteración 2	7/7/25	1/8/25
Entrada de diseño del sistema	7/7/25	7/7/25
Integrador de IPs	7/7/25	7/7/25
Implementación	8/7/25	8/7/25
Restricciones	8/7/25	8/7/25
Síntesis	8/7/25	8/7/25
Implementación	8/7/25	8/7/25
Análisis	8/7/25	8/7/25
Validación y puesta en marcha	10/7/25	1/8/25
Software de aplicaciones	10/7/25	28/7/25
Software de inyecciones	29/7/25	29/7/25
Tests	30/7/25	1/8/25
Redacción del documento	14/2/25	28/8/25
Estado de la cuestión	14/2/25	13/3/25
Planificación y presupuesto	20/3/25	21/3/25
Metodología y diseño	4/7/25	4/8/25
Resultados	4/8/25	5/8/25
Introducción	6/8/25	7/8/25
Conclusiones	6/8/25	28/8/25
Tutorización	28/11/24	4/9/25
Tutoría 1	28/11/24	28/11/24
Tutoría 2	30/1/25	30/1/25
Tutoría 3	2/4/25	2/4/25
Tutoría 4	9/5/25	9/5/25
Tutoría 5	4/7/25	4/7/25
Revisión del documento	29/8/25	4/9/25

ANEXO L. DIAGRAMA DE GANTT



ANEXO M. DIAGRAMA DE RECURSOS



ANEXO N. DECLARACIÓN DE USO DE IA GENERATIVA



DECLARACIÓN DE USO DE IA GENERATIVA EN EL TRABAJO DE FIN DE GRADO

He usado IA Generativa en este trabajo

Marca lo que corresponda:

NO

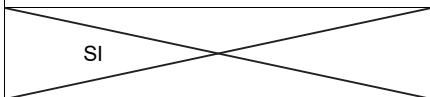
Si has marcado Sí, completa las siguientes 3 partes de este documento:

Parte 1: reflexión sobre comportamiento ético y responsable

Ten presente que el uso de IA Generativa conlleva unos riesgos y puede generar una serie de consecuencias que afecten a la integridad moral de tu actuación con ella. Por eso, te pedimos que contestes con honestidad a las siguientes preguntas (*Marca lo que corresponda*):

Pregunta		
1. En mi interacción con herramientas de IA Generativa he remitido datos de carácter sensible con la debida autorización de los interesados.		
Sí, he usado estos datos con autorización	NO, he usado estos datos sin autorización	NO, no he usado datos de carácter sensible
2. En mi interacción con herramientas de IA Generativa he remitido materiales protegidos por derechos de autor con la debida autorización de los interesados.		
Sí, he usado estos materiales con autorización	NO, he usado estos materiales sin autorización	NO, no he usado materiales protegidos
3. En mi interacción con herramientas de IA Generativa he remitido datos de carácter personal con la debida autorización de los interesados.		
Sí, he usado estos datos con autorización	NO, he usado estos datos sin autorización	NO, no he usado datos de carácter personal

4. Mi utilización de la herramienta de IA Generativa ha **respetado sus términos de uso**, así como los principios éticos esenciales, no orientándola de manera maliciosa a obtener un resultado inapropiado para el trabajo presentado, es decir, que produzca una impresión o conocimiento contrario a la realidad de los resultados obtenidos, que suplante mi propio trabajo o que pueda resultar en un perjuicio para las personas.

	NO
---	----

Si **NO** has contado con la autorización de los interesados en alguna de las preguntas 1, 2 o 3, explica brevemente el motivo (*por ejemplo, "los materiales estaban protegidos pero permitían su uso para este fin" o "los términos de uso, que se pueden encontrar en esta dirección (...), impiden el uso que he hecho, pero era imprescindible dada la naturaleza del trabajo"*).

Parte 2: declaración de uso técnico

Declaro haber hecho uso del sistema de IA Generativa DeepSeek-V3 para:

Desarrollar contenido específico

Se ha hecho uso de IA Generativa como herramienta de soporte para el desarrollo del contenido específico del TFG, incluyendo:

- *Asistencia en el desarrollo de líneas de código (programación)*

Declaro haber hecho uso del sistema de IA Generativa DeepSeek-V3 para solicitar la creación de plantillas de tablas de LaTeX que siguieran los formatos especificados por mí. Para ello, se empleó el *prompt* «Generate a LaTeX table with dummy entries that matches the format of the table in the image», junto con una imagen adjunta de la tabla deseada creada por mí en Excel. Las interacciones posteriores trataron de corregir errores en la generación de las tablas.

Parte 3: reflexión sobre utilidad

Por favor, aporta una valoración personal (formato libre) sobre las fortalezas y debilidades que has identificado en el uso de herramientas de IA Generativa en el desarrollo de tu trabajo. Menciona si te ha servido en el proceso de aprendizaje, o en el desarrollo o en la extracción de conclusiones de tu trabajo.

La IA Generativa me ha ayudado ampliamente a familiarizarme con la creación de tablas de mayor complejidad en LaTeX. Si bien sus debilidades actuales en el análisis de las imágenes y la asimilación de la información y su traducción a código son evidentes, ha sido capaz de otorgarme las bases sobre las que realizar modificaciones sencillas para llegar a las tablas deseadas. Así, la IA me ha facilitado el aprendizaje sobre las diferentes formas de creación de una tabla, las librerías que se pueden utilizar, y en general diferentes métodos para obtener el formato deseado.