ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Metodología de diseño de sistemas de control para dispositivos SoC FPGA

Autor: Gabriel Santana Quintana
Tutor(es): Dr. Pedro Pérez Carballo
Dr. Carlos Betancor Martín
Fecha: Marzo 2022

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Metodología de diseño de sistemas de control para dispositivos SoC FPGA

HOJA DE FIRMAS

Alumno:	Gabriel Santana Quintana	Fdo.:	SANTANA QUINTANA GABRIEL - 45397933G Firmado digitalmente por SANTANA QUINTANA GABRIEL - 45397933G Fecha: 2022.04.14 01:34:12 +01'00'
Tutor:	Dr. Pedro Pérez Carballo	Fdo.:	PEREZ CARBALLO PEDRO FRANCISCO - 42163642L Firmado digitalmente por PEREZ CARBALLO PEDRO FRANCISCO - 42163642L Fecha: 2022.04.11 23:51:42 +01'00'
Tutor:	Dr. Carlos Betancor Martín	Fdo.:	BETANCOR MARTIN CARLOS SALVADOR - 43623320W Firmado digitalmente por BETANCOR MARTIN CARLOS SALVADOR - 43623320W Fecha: 2022.04.12 11:23:29 +01'00'
Fecha:	Marzo 2022		

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Metodología de diseño de sistemas de control para dispositivos SoC FPGA

HOJA DE EVALUACIÓN

Calificación: .....

Presidente	Dr. Sebastián López Suárez	Fdo.:
Secretario	Dr. Sunil Lalchand Khemchandani	Fdo.:
Vocal	Dr. Alfonso Medina Escuela	Fdo.:

Fecha: Marzo 2022

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Agradecimientos

A mi familia,
por apoyarme y darme fuerzas
durante el proceso de realización de este trabajo

Resumen

En este Trabajo Fin de Máster se propone una metodología de alto nivel para el diseño de sistemas de control a ser implementados en dispositivos SoC FPGA. Este desarrollo se aborda en dos fases. En la primera se presenta la metodología propuesta y en la segunda se realiza un ejemplo de implementación para su validación.

Para la metodología de diseño propuesta se utilizan las herramientas MATLAB/Simulink y Vivado Design Suite. Esta metodología consta de tres pasos claves, siendo estos, las simulaciones *Model in the Loop* (MIL), *Software in the Loop* (SIL) y *SoC in the Loop*. Con la simulación MIL se verifica el funcionamiento de los algoritmos de control desarrollados en conjunto con el modelo del sistema a controlar. La simulación SIL se realiza con el fin de comprobar que MATLAB/Simulink es capaz de generar código de los algoritmos diseñados y que con este se obtienen resultados análogos a los de la simulación MIL. Finalmente, la simulación *SoC in the Loop* se lleva a cabo para verificar el funcionamiento de la implementación de los algoritmos desarrollados sobre un dispositivo SoC FPGA, simulando su funcionamiento en conjunto con el modelo del sistema a controlar.

Como ejemplo de implementación se escoge el algoritmo MMTES (*Maximun-Minimum Tier Equalization Swapping*) en conjunto con un controlador MPPT (*Maximum Power Point Tracker*) para maximizar la producción de potencia de un sistema fotovoltaico en condiciones de sombras parciales. Se realiza una implementación *hardware* del controlador MPPT y una implementación *hardware/software* del algoritmo MMTES. El modelado del sistema a controlar se realiza en Simulink y los algoritmos de control se desarrollan utilizando MATLAB/Simulink. La placa de desarrollo sobre la que se implementan estos algoritmos es la ZedBoard, que incluye un dispositivo Zynq 7000.

La verificación del sistema implementado sobre la ZedBoard se realiza en conjunto con el modelo del sistema en Simulink. El sistema implementado tiene una utilización de recursos y un consumo de potencia relativamente bajo. La producción de potencia obtenida al realizar la simulación *SoC in the Loop* es equiparable a los resultados de las simulaciones MIL y SIL. Por lo tanto, se valida el correcto funcionamiento del sistema implementado en la ZedBoard y con ello, la funcionalidad de la metodología de diseño.

Abstract

This Master Thesis proposes a high-level methodology for the design of control systems to be implemented in SoC FPGA devices. This development is approached in two phases. In the first one, the proposed methodology is presented and in the second one, an example of implementation is carried out for its validation.

MATLAB/Simulink and Vivado Design Suite tools are used for the proposed design methodology. This methodology consists of three key steps: Model in the Loop (MIL), Software in the Loop (SIL) and SoC in the Loop simulations. The MIL simulation verifies the operation of the control algorithms developed in conjunction with the model of the system to be controlled. The SIL simulation is carried out to verify that MATLAB/Simulink is capable of generating code for the designed algorithms and that the results obtained are similar to those of the MIL simulation. Finally, the SoC in the Loop simulation is carried out to verify the performance of the implementation of the algorithms developed on a SoC FPGA device, simulating its operation in conjunction with the model of the system to be controlled.

As an example of implementation, the MMTES (Maximun-Minimum Tier Equalization Swapping) algorithm is chosen in conjunction with an MPPT (Maximum Power Point Tracker) controller to maximize the power production of a photovoltaic system under partial shading conditions. A hardware implementation of the MPPT controller and a hardware/software implementation of the MMTES algorithm are carried out. The modelling of the system to be controlled is done in Simulink and the control algorithms are developed using MATLAB/Simulink. The development board on which these algorithms are implemented is the ZedBoard, which includes a Zynq 7000 device.

The verification of the system implemented on the ZedBoard is carried out in conjunction with the system model in Simulink. The implemented system has a relatively low resource utilisation and power consumption. The power output obtained by performing the SoC in the Loop simulation is comparable to the results of the MIL and SIL simulations. Therefore, the correct functioning of the system implemented on the ZedBoard is validated and with it, the functionality of the design methodology.

Tabla de Contenidos

TABLA DE CONTENIDOS ..... **I**

ÍNDICE DE FIGURAS ..... **V**

ÍNDICE DE TABLAS ..... **IX**

ÍNDICE DE CÓDIGOS ..... **XI**

ACRÓNIMOS ..... **XIII**

CAPÍTULO 1. INTRODUCCIÓN **1**

1.1. ANTECEDENTES 1
1.2. OBJETIVOS 6
1.3. PETICIONARIO 7
1.4. ESTRUCTURA DEL DOCUMENTO 7

CAPÍTULO 2. ESTADO DEL ARTE **9**

2.1. INTRODUCCIÓN 9
2.2. USO DE FPGA PARA SISTEMAS DE CONTROL EN ROBÓTICA 9
2.3. CONTROLADORES EMPOTRADOS PARA APLICACIONES INDUSTRIALES BASADOS EN FPGA 11
2.4. CONTROL BASADO EN FPGA PARA ELECTRÓNICA DE POTENCIA Y SISTEMAS DE GENERACIÓN DE ENERGÍA ELÉCTRICA . 13
2.5. METODOLOGÍAS DE DISEÑO 14
2.6. CONCLUSIONES 15

CAPÍTULO 3. CASO DE IMPLEMENTACIÓN **17**

3.1. INTRODUCCIÓN 17
3.2. ÁREA DE APLICACIÓN 17
3.3. CASO DE ESTUDIO 18
3.4. CONCLUSIONES 18

CAPÍTULO 4. METODOLOGÍA DE DISEÑO **19**

4.1. INTRODUCCIÓN 19
4.2. MATLAB/SIMULINK 19
4.2.1. Simscape Electrical *20*
4.2.2. Fixed-Point Designer *20*
4.2.3. MATLAB Coder *20*
4.2.4. Simulink Coder *21*
4.2.5. Embedded Coder *21*
4.2.6. HDL Coder *21*
4.2.7. HDL Coder Support Package for Xilinx Zynq Platform *21*
4.2.8. Embedded Coder Support Package for Xilinx Zynq Platform *21*
4.3. HERRAMIENTAS DE SÍNTESIS E IMPLEMENTACIÓN PARA FPGA 22
4.3.1. Xilinx Vivado Design Suite *22*
4.4. METODOLOGÍA DE DISEÑO 23

4.5. CONCLUSIONES 27

CAPÍTULO 5. ALGORITMO DE RECONFIGURACIÓN DINÁMICA DE PANELES FOTOVOLTAICOS 29

5.1. INTRODUCCIÓN 29
5.2. ALGORITMO MMTES 31
5.2.1. Características principales 31
5.2.2. Funcionamiento 32
5.2.3. Matriz de interruptores 35
5.3. IMPLEMENTACIÓN DEL ALGORITMO MMTES EN MATLAB 36
5.3.1. Función MMTESSort 36
5.3.2. Función PairForSwapping 37
5.3.3. Función SwapPair 38
5.4. MEJORAS PROPUESTAS PARA UNA IMPLEMENTACIÓN HW/SW 38
5.4.1. Bitonic Sort 41
5.4.1.1. AdaptInputData 43
5.4.1.2. BitonicSort 44
5.4.1.3. AdaptOutputData 46
5.4.2. PairForSwapping paralelo 46
5.4.2.1. PfsPllInData 47
5.4.2.2. PairForSwappingPll 48
5.4.2.3. PfsPllOutData 49
5.5. CONCLUSIONES 50

CAPÍTULO 6. CONTROLADOR MPPT 51

6.1. INTRODUCCIÓN 51
6.2. CONTROLADOR MPPT 51
6.2.1. Perturbación y Observación (P&O) 52
6.2.2. Conductancia incremental 53
6.2.3. Tensión a circuito abierto fraccional 53
6.3. IMPLEMENTACIÓN DEL CONTROLADOR 54
6.4. CONCLUSIONES 56

CAPÍTULO 7. MODELADO DEL SISTEMA 57

7.1. INTRODUCCIÓN 57
7.2. SISTEMA A MODELAR 57
7.3. MODELADO DEL *HARDWARE* 58
7.3.1. Paneles fotovoltaicos 58
7.3.2. Switch matrix 60
7.3.3. Buck Converter 62
7.4. MODELADO DE LOS ALGORITMOS A IMPLEMENTAR EN EL SOC FPGA 65
7.4.1. Función auxCtrl 65
7.4.2. Generador de PWM 66
7.5. SISTEMA FINAL 67
7.6. SIMULACIÓN *MODEL-IN-THE-LOOP* (MIL) 69
7.7. SIMULACIÓN *SOFTWARE-IN-THE-LOOP* (SIL) 84
7.8. CONCLUSIONES 85

CAPÍTULO 8. DESARROLLO Y VERIFICACIÓN DE LOS BLOQUES IP 87

8.1. INTRODUCCIÓN 87
8.2. PLACA DE DESARROLLO 87
8.3. CODISEÑO HARDWARE/SOFTWARE 88
8.4. MPPT 90
8.4.1. Conversión a punto fijo 90
8.4.2. Verificación del bloque IP generado 91
8.5. GENERADOR DE PWM 92
8.5.1. Verificación del bloque IP generado 92
8.6. MMTES 93
8.6.1. MMTESSort 94

8.6.1.1. Conversión a punto fijo 94
8.6.1.2. Verificación del bloque IP generado 94
8.6.2. PairForSwapping 95
8.6.2.1. Conversión a punto fijo 95
8.6.2.2. Verificación del bloque IP generado 95
8.6.3. Bitonic Sort 95
8.6.3.1. Verificación del bloque IP generado 96
8.6.4. PairForSwapping paralelo 96
8.6.4.1. Verificación del bloque IP generado 96
8.6.5. Comparación de resultados 97
8.7. CONCLUSIONES 99

CAPÍTULO 9. IMPLEMENTACIÓN Y VERIFICACIÓN 101

9.1. INTRODUCCIÓN 101
9.2. IMPLEMENTACIÓN DEL SISTEMA FINAL 101
9.3. VERIFICACIÓN 108
9.4. CONCLUSIONES 117

CAPÍTULO 10. CONCLUSIONES Y TRABAJOS FUTUROS 119

10.1. CONCLUSIONES 119
10.2. TRABAJOS FUTUROS 121

REFERENCIAS 123

ANEXO 131

Índice de Figuras

FIGURA 1. REPRESENTACIÓN DE LA COMPLEJIDAD ALGORÍTMICA Y LA CAPACIDAD DE PROCESAMIENTO PARALELO PARA LAS DISTINTAS TECNOLOGÍAS DE IMPLEMENTACIÓN DE SISTEMAS DE CONTROL INDUSTRIAL [3]. 2

FIGURA 2. MERCADO GLOBAL DE FPGA POR REGIÓN 2020-2026 [8]. 4

FIGURA 3. DIVISIÓN DEL MERCADO DE FPGA POR SECTORES [9]. 5

FIGURA 4. MODOS DE SIMULACIÓN DE LA METODOLOGÍA DE DISEÑO. 6

FIGURA 5. ROBOT SCARA [29]. 11

FIGURA 6. PUMA 560 [30]. 11

FIGURA 7. DIAGRAMA DE BLOQUES DEL MODELO DEL SISTEMA CONTROLAR. 18

FIGURA 8. CODISEÑO HARDWARE/SOFTWARE PARA PLATAFORMAS SOC [67]. 22

FIGURA 9. SIMULACIÓN MIL. 23

FIGURA 10. SIMULACIÓN SIL. 24

FIGURA 11. SIMULACIÓN *SOC IN THE LOOP*. 25

FIGURA 12. FLUJO DE DISEÑO. 26

FIGURA 13. DISTRIBUCIÓN DE LOS PASOS DEL FLUJO DE DISEÑO SEGÚN LAS HERRAMIENTAS NECESARIAS. 26

FIGURA 14. EJEMPLO DE SOMBRAS PARCIALES SOBRE UNA INSTALACIÓN FOTOVOLTAICA [72]. 29

FIGURA 15. CONFIGURACIÓN TCT 3X3. 31

FIGURA 16. DIAGRAMA DE FLUJO DEL ALGORITMO MMTES [56]. 33

FIGURA 17. EJEMPLO DE MATRIZ DE INTERRUPTORES PARA UN ARRAY 3 X 3 [56]. 35

FIGURA 18. REPRESENTACIÓN EN PORCENTAJES DEL TIEMPO DE EJECUCIÓN DEL ALGORITMO MMTES. . 40

FIGURA 19. BITONIC SORTING NETWORK PARA UN ARRAY DE ENTRADA DE 16 ELEMENTOS [78]. 42

FIGURA 20.ESQUEMÁTICO DEL BITONIC SORT PARA EL CASO 1, 2 Y 4. 42

FIGURA 21. PAQUETE DE DATOS A TRANSMITIR AL BITONIC SORT. 44

FIGURA 22. BITONICSORT. 45

FIGURA 23.BLOQUE DE ORDENAMIENTO CRECIENTE. 46

FIGURA 24. ARRAY DE SALIDA DE LA FUNCIÓN PFSPLLINDATA. 47

FIGURA 25. ARRAYS AUXILIARES EN LA FUNCIÓN PAIRFORSWAPPINGPLL. 48

FIGURA 26. ARRAYS POSMAX Y POSMIN. 48

FIGURA 27. EMPAQUETADO DE LOS DATOS DE SALIDA DE PAIRFORSWAPPINGPLL. 49
FIGURA 28. FUNCIÓN PFSPLLOUTDATA. 49
FIGURA 29. DIAGRAMA P-V DE UNA INSTALACIÓN FOTOVOLTAICA. 52
FIGURA 30. DIAGRAMA DE FLUJO DEL ALGORITMO MPPT DE CONDUCTANCIA INCREMENTAL [82]. 54
FIGURA 31. DIAGRAMA DE BLOQUES DEL SISTEMA A MODELAR. 58
FIGURA 32. MODELO DE MÓDULO FOTOVOLTAICO IMPLEMENTADO POR EL BLOQUE PV ARRAY [85]. 59
FIGURA 33. COMPARATIVA DE CARACTERÍSTICAS I-V A TEMPERATURA CONSTANTE PARA EL MODELO CS6A-170PE. 60
FIGURA 34. SUBSYSTEM SUBSWITCHMATRIX. 61
FIGURA 35. EJEMPLO DE CONEXIONADO DE LOS SUBSWITCHMATRIX. 61
FIGURA 36. IMPLEMENTACIÓN DEL BUCK CONVERTER EN SIMULINK. 64
FIGURA 37. CORRIENTE DE RIZADO DEL INDUCTOR Y RIZADO DE LA TENSIÓN DE SALIDA. 64
FIGURA 38. GENERACIÓN DE LA SEÑAL PWM. 66
FIGURA 39. IMPLEMENTACIÓN DEL GENERADOR DE PWM. 67
FIGURA 40. MODELADO FINAL DEL SISTEMA EN SIMULINK. 68
FIGURA 41. CURVAS P-V CASO 1. 70
FIGURA 42. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R1 EN EL CASO 1. 71
FIGURA 43.COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_2 EN EL CASO 1. 71
FIGURA 44.COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_3 EN EL CASO 1. 72
FIGURA 45. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_1 EN EL CASO 1. 73
FIGURA 46. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_2 EN EL CASO 1. 73
FIGURA 47. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_3 EN EL CASO 1. 74
FIGURA 48. CURVAS P-V CASO 2. 75
FIGURA 49.COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_1 EN EL CASO 2. 75
FIGURA 50. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_2 EN EL CASO 2. 76
FIGURA 51. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_3 EN EL CASO 2. 76
FIGURA 52. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_1 EN EL CASO 2. 77
FIGURA 53. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_2 EN EL CASO 2. 78
FIGURA 54. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_3 EN EL CASO 2. 78
FIGURA 55.CURVAS P-V CASO 4. 79
FIGURA 56. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R1 EN EL CASO 4. 80
FIGURA 57. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_2 EN EL CASO 4. 80
FIGURA 58. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA R_3 EN EL CASO 4. 81
FIGURA 59. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL1 EN EL CASO 4. 82
FIGURA 60. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_2 EN EL CASO 4. 82
FIGURA 61. COMPARATIVA DE LA POTENCIA DE SALIDA CON CARGA RL_3 EN EL CASO 4. 83
FIGURA 62. OVERLAY VIEW DE LA ZEDBOARD [98]. 88
FIGURA 63. COMUNICACIÓN ENTRE EL BLOQUE IP Y EL PROCESADOR EMPOTRADO [100]. 89
FIGURA 64. MODO DE FUNCIONAMIENTO BLOQUEANTE [101]. 89

FIGURA 65. MODO DE FUNCIONAMIENTO NO BLOQUEANTE [101]. 89
FIGURA 66. SIMULACIÓN EN MODO EXTERNO DESDE SIMULINK [99]. 90
FIGURA 67. COMPARATIVA DEL CICLO DE TRABAJO CALCULADO POR EL BLOQUE IP Y LA SIMULACIÓN EN SIMULINK. 92
FIGURA 68. VERIFICACIÓN DEL IP PWM_GENERATOR. 93
FIGURA 69. COMPARATIVA MMTESSORT Y BITONICSORT. 98
FIGURA 70. COMPARATIVA PAIRFORSWAPPING Y PAIRFORSWAPPINGPLL. 99
FIGURA 71. DIAGRAMA DE BLOQUES DE LA PLATAFORMA HARDWARE GENERADA. 102
FIGURA 72. UTILIZACIÓN DE RECURSOS DEL SISTEMA. 103
FIGURA 73. RESULTADOS DE TEMPORIZACIÓN DEL SISTEMA. 104
FIGURA 74. CONSUMICIÓN DE POTENCIA DEL SISTEMA DISEÑADO. 104
FIGURA 75. LAYOUT DEL SISTEMA. 105
FIGURA 76. MODELO ZEDBOARDMODEL. 106
FIGURA 77. MODELO EN EL QUE SE BASA EL MODELO SIMSCAPEMODEL. 109
FIGURA 78. MODELO SIMSCAPEMODEL. 110
FIGURA 79. FLUJO DE DATOS ENTRE SIMSCAPEMODEL Y LA ZEDBOARD. 111
FIGURA 80. SIMULACIÓN DE SIMSCAPEMODEL EN CONJUNTO CON ZEDBOARDMODEL PARA EL CASO 1 CON CARGA R_2. 111
FIGURA 81. SIMULACIÓN DE SIMSCAPEMODEL EN CONJUNTO CON ZEDBOARDMODEL PARA EL CASO 2 CON CARGA R_2. 112
FIGURA 82. SIMULACIÓN DE SIMSCAPEMODEL EN CONJUNTO CON ZEDBOARDMODEL PARA EL CASO 4 CON CARGA R_2. 112
FIGURA 83. SIMULACIÓN DE SIMSCAPEMODEL EN CONJUNTO CON ZEDBOARDMODEL PARA EL CASO 1 CON CARGA RL_2. 113
FIGURA 84. SIMULACIÓN DE SIMSCAPEMODEL EN CONJUNTO CON ZEDBOARDMODEL PARA EL CASO 2 CON CARGA RL_2. 114
FIGURA 85. SIMULACIÓN DE SIMSCAPEMODEL EN CONJUNTO CON ZEDBOARDMODEL PARA EL CASO 4 CON CARGA RL_2. 114
FIGURA 86. COMPARATIVA DE LOS TIEMPOS DE EJECUCIÓN DE SORT SW Y BITONICSORT HW/SW. 116
FIGURA 87. CONEXIONADO DE LA ZEDBOARD CON LA ESTACIÓN DE TRABAJO. 131
FIGURA 88. CONFIGURACIÓN DE LOS JUMPERS DE LA ZEDBOARD. 132
FIGURA 89. VENTANA MANAGE ADD-ONS. 132
FIGURA 90. HARDWARE SETUP: ELECCIÓN DE LA PLACA DE DESARROLLO. 133
FIGURA 91. HARDWARE SETUP: CONFIGURACIÓN DE RED DEL SISTEMA OPERATIVO. 133
FIGURA 92. HARDWARE SETUP: CARGA DE LA IMAGEN EN LA TARJETA SD. 134
FIGURA 93. TESTBENCH DEL CONTROLADOR MPPT. 135
FIGURA 94. LANZAMIENTO DE LA HERRAMIENTA HDL WORKFLOW ADVISOR. 135
FIGURA 95. ELECCIÓN DE LA PLACA DE DESARROLLO Y NOMBRE DEL PROYECTO. 136
FIGURA 96. SELECCIÓN DEL DISEÑO DE REFERENCIA. 136
FIGURA 97. SELECCIÓN DEL MODO DE FUNCIONAMIENTO E INTERFAZ DE ENTRADA/SALIDA DEL BLOQUE IP. 137

FIGURA 98. SELECCIÓN DE LA FRECUENCIA DE FUNCIONAMIENTO. 137
FIGURA 99. EJECUCIÓN ÍTEM 3.2 DEL HDL WORKFLOW ADVISOR. 138
FIGURA 100. MODELO SOFTWARE INTERFACE GENERADO. 138

Índice de Tablas

TABLA 1. VENTAJAS Y DESVENTAJAS DE LAS DIFERENTES TECNOLOGÍAS DIGITALES (ADAPTADO DE [3]). .. 3

TABLA 2. ESTADOS POSIBLES DE LOS INTERRUPTORES DE LA MATRIZ DE INTERRUPTORES PARA UN *ARRAY* 3 X 3. 35

TABLA 3. CASOS DE ESTUDIO DEL ALGORITMO MMTES. 39

TABLA 4. TIEMPO DE EJECUCIÓN DEL ALGORITMO MMTES EN MATLAB. 39

TABLA 5. VALORES DE IRRADIANCIA UTILIZADOS EN LA SIMULACIÓN MIL Y SIL. 69

TABLA 6. CARGAS UTILIZADAS DURANTE LA SIMULACIÓN MIL Y SIL. 69

TABLA 8. COMPARATIVA DE RESULTADOS DE LA SIMULACIÓN MIL CON CARGA INDUCTIVA. 84

TABLA 9. COMPARATIVA DE LAS POTENCIAS DE SALIDA OBTENIDAS EN LA SIMULACIÓN MIL Y SIL CON CARGA RESISTIVA. 84

TABLA 10. COMPARATIVA DE LAS POTENCIAS DE SALIDA OBTENIDAS EN LA SIMULACIÓN MIL Y SIL CON CARGA INDUCTIVA. 84

TABLA 11. RESULTADOS DE CONVERSIÓN A PUNTO FIJO DE LA FUNCIÓN MPPT_INC. 91

TABLA 12. CASO DE ESTUDIO 1. 91

TABLA 13. TIEMPOS DE EJECUCIÓN IP MMTESSORT. 95

TABLA 14. TIEMPO DE EJECUCIÓN IP PAIRFORSWAPPING. 95

TABLA 15. TIEMPO DE EJECUCIÓN IP BITONICSORT. 96

TABLA 16. TIEMPO DE EJECUCIÓN IP PAIRFORSWAPPINGPLL. 97

TABLA 17. COMPARACIÓN TIEMPOS DE EJECUCIÓN MMTESSORT Y BITONICSORT. 97

TABLA 18. COMPARACIÓN TIEMPOS DE EJECUCIÓN PAIRFORSWAPPING Y PAIRFORSWAPPINGPLL. 98

TABLA 19. DIRECCIONES DE MEMORIA DE LOS BLOQUES IP DESARROLLADOS. 103

TABLA 20. UTILIZACIÓN DE RECURSOS POR BLOQUE IP. 103

TABLA 21. CONSUMICIÓN DE POTENCIA DE LOS BLOQUES IP. 104

TABLA 22. COMPARATIVA DE LOS TIEMPOS DE EJECUCIÓN PARA LA IMPLEMENTACIÓN SOFTWARE Y HARDWARE/SOFTWARE. 115

TABLA 23. COMPARATIVA DE IMPLEMENTACIONES DE ALGORITMOS DPVAR SOBRE FPGA. 120

TABLA 24. COMPARATIVA DE IMPLEMENTACIONES DE CONTROLADORES MPPT SOBRE FPGA. 121

Índice de Códigos

CÓDIGO FUENTE 1. FUNCIÓN MMTESSORT 36
CÓDIGO FUENTE 2. FUNCIÓN PAIRFORSWAPPING 37
CÓDIGO FUENTE 3. FUNCIÓN SWAPPAIR 38
CÓDIGO FUENTE 4. FUNCIÓN MPPT_INC 55
CÓDIGO FUENTE 5. FUNCIÓN AUXCTRL 65

Acrónimos

ASIC	*Application-Specific Integrated Circuit*
AXI	*Advanced eXtensible Interface*
CHB-MLI	*Cascade H-Bridge MultiLevel Invertir*
CORDIC	*COordinate Rotation DIgital Computer*
CPU	*Central Processing Unit*
DPVAR	*Dynamic Photovoltaic Array Reconfiguration*
DSP	*Digital Signal Processor*
EKF	*Extended Kalman Filter*
FF	*Flip-Flop*
FPGA	*Field Programmable Gate Array*
FS-MPC	*Finite-State Model Predictive Control*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
HC	*Honey Comb*
HDL	*Hardware Description Language*
ILA	*Integrated Logic Analyzer*

IP	*Intellectual Property*
IUMA	*Instituto Universitario de Microelectrónica Aplicada*
LUT	*Look Up Table*
MIL	*Model-In-the-Loop*
MMTES	*Maximum-Minimun Tier Equalisation Swapping*
MPC	*Model Predictive Control*
MPP	*Maximum Power Point*
MPPT	*Maximum Power Point Tracker*
MRR	*Modular Reconfigurable Robot*
NREL	*National Renewable Energy Laboratory*
P&O	*Perturb & Observe*
PI	*Proportional Integral*
PL	*Programmable Logic*
PMSM	*Permanent Magnet Synchronous Motor*
PS	*Processing System*
PUMA	*Programmable Universal Manipulation Arm*
PWM	*Pulse Width Modulation*
RTL	*Register-Transfer Level*
RTOS	*Real-Time Operating System*
SCARA	*Selective Compliant Articulated Robot Arm*
SCC	*Specialized Computational Cores*
SICAD	*Sistemas Industriales y CAD*
SIL	*Software-In-the-Loop*

SMPS	*Switched Mode Power Supplies*
SoC	*System on Chip*
SPST	*Single Pole, Single Throw*
SPWM	*Sinusoidal Pulse Width Modulation*
SRAM	*Static Random-Access Memory*
SVPWM	*Space Vector PWM*
TCT	*Total Cross Tied*
TFM	*Trabajo Fin de Máster*
UDP	*User Datagram Protocol*
ULPGC	*Universidad de las Palmas de Gran Canaria*
USD	*United States Dollar*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*

Capítulo 1. Introducción

1.1. Antecedentes

Debido a las propuestas de los nuevos paradigmas en la industria, tales como la industria 5.0 [1], se exige a los nuevos sistemas de control industrial un mejor rendimiento, mayor flexibilidad y un mayor grado de fiabilidad. Además, para mantener la competitividad en el mercado actual, el coste y el consumo de potencia de los controladores son puntos claves a tener en cuenta, introduciendo el ahorro energético como una exigencia adicional a las ya citadas.

A la hora de implementar sistemas de control industrial en tiempo real eficientes, los diseñadores cuentan tradicionalmente con dos tecnologías principales de dispositivos digitales [2]. Por una parte, siguiendo una aproximación *software,* se utilizan microcontroladores y procesadores de señales digitales (DSP, *Digital Signal Processors*) y, por otra parte, una aproximación *hardware,* usando dispositivos FPGA (*Field Programmable Logic Array*). Existe una tercera vía que representa una solución *hardware/software* usando dispositivos SoC (*System-on-Chip*), que incluyen capacidad de procesamiento *software* y soporte para FPGA. En la Figura 1 se presentan las tecnologías citadas, comparando la complejidad algorítmica y capacidad de procesamiento en paralelo que son capaces de abordar.

Los microcontroladores y DSP son dispositivos digitales para los que se desarrolla una solución en el dominio *software*, estando formados principalmente por un microprocesador y varios periféricos, así como unidades de procesamiento especiales para

el caso de los DSP. Estos son los componentes con los que se realiza el control del sistema objetivo, así como los encargados de llevar a cabo las comunicaciones. Las ventajas que ofrece esta tecnología en las aplicaciones de control industrial son su flexibilidad y bajo coste. Entre sus desventajas podemos citar su alto consumo de potencia y la dificultad para aprovechar el paralelismo de los algoritmos de control que se implementan debido a su naturaleza secuencial.

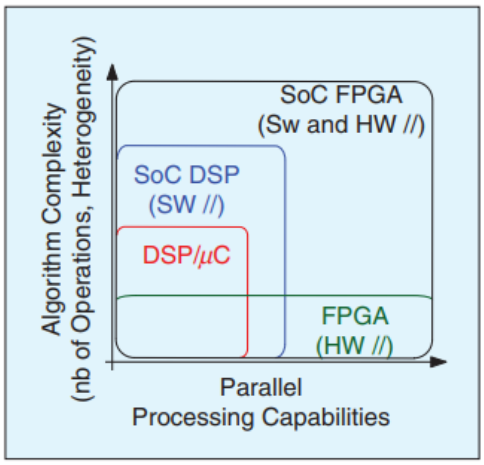


Figura 1. Representación de la complejidad algorítmica y la capacidad de procesamiento paralelo para las distintas tecnologías de implementación de sistemas de control industrial [3].

Las FPGA, al contrario que los microcontroladores y DSP, son dispositivos para los que se desarrolla una solución principalmente en el dominio *hardware*. Este tipo de dispositivos proporcionan ventajas para aplicaciones en tiempo real exigentes en respuesta temporal (*hard real time*) frente a soluciones basadas en microprocesadores, como pueden ser su capacidad de procesamiento paralelo y el bajo consumo de potencia. Al implementar controladores utilizando esta tecnología, se consiguen tiempos de ejecución bajos gracias a que se aprovecha el paralelismo de los algoritmos de control implementándolos en *hardware*, consiguiendo además un tamaño y costo reducido.

La velocidad y el determinismo que se consiguen al realizar la implementación de los controladores en FPGA presenta ventajas importantes al trabajar con sistemas de control en red frente a las implementaciones basadas en microcontroladores. En los sistemas de control en red, el *software* de comunicación juega un papel importante, siendo frecuente sacrificar el rendimiento del controlador para lograr una implementación *software* conjunta del controlador y las comunicaciones. En caso de utilizar FPGA no existe esta

limitación, ya que se desarrolla una implementación *hardware* del controlador, lo que permite conseguir una gran mejora en el rendimiento del sistema diseñado.

Como se observa en la Figura 1, existen variantes *System on Chip* (SoC), de las tecnologías ya mencionadas, que presentan mejoras en la complejidad algorítmica y la capacidad de procesamiento en paralelo. Se aprecia sobre todo una mejoría en los SoC FPGA que mantienen la capacidad de procesamiento en paralelo, ganando mucha complejidad algorítmica y ofreciendo la posibilidad de desarrollar una implementación *hardware/software* de los algoritmos de control. Un ejemplo de SoC FPGA es la familia Zynq-7000 [4] de Xilinx que combina un doble núcleo ARM Cortex-A9 [5] con una FPGA de la serie 7 [6] de Xilinx junto con diversos periféricos. En la Tabla 1 se resumen las ventajas y desventajas de las diferentes tecnologías digitales comentadas con anterioridad.

Tabla 1. Ventajas y desventajas de las diferentes tecnologías digitales (adaptado de [3]).

Criterio	**DSP/µC**	**FPGA**	**SoC DSP**	**SoC FPGA**
Perspectiva algorítmica	**media**	**media**	**buena**	**muy buena**
Gestión de la complejidad algorítmica	media	media	buena	muy buena
Rapidez, posibilidad de paralelismo	media	buena (Paralelismo HW: concurrencia y pipeline)	buena (Paralelismo SW con múltiples núcleos)	muy buena (paralelismo SW y HW)
Precisión (capacidad de punto flotante)	muy buena	buena	muy buena	SW: muy buena HW: buena
Conectividad	**media**	**media**	**buena**	**buena**
Interfaz analógica (ADC, DAC)	buena	media	buena	media
Interfaz digital (número de E/S)	buena	muy buena	buena	muy buena
Periféricos integrados	buena	buena	muy buena	muy buena
Sistema operativo integrado (acceso a Internet, ...)	media	media	media	muy buena
Flexibilidad de uso	**buena**	**buena**	**buena**	**buena**
Facilidad de código	muy buena	buena	muy buena	buena
Adaptación de la microarquitectura	mala	muy buena	media	muy buena
Curva de aprendizaje	muy buena	buena	buena	media
Coste	**medio**	**alto**	**medio**	**alto**

Entre las desventajas que presenta las soluciones basadas en FPGA frente a los DSP y los microcontroladores podemos citar su coste. Hoy en día, el coste de las FPGA es superior al de los DSP y microcontroladores, incluido el coste de diseño *hardware* implícito en la solución y de integración en los casos en que se utilicen soluciones *hardware/software*.

Hay que mencionar que se prevé un crecimiento en el mercado de las FPGA a medio plazo. En el año 2020, el mercado de las FPGA superó los 6 mil millones USD y se estima un crecimiento del 15% en el período de 2021 a 2027 [7]. En la Figura 2 se puede observar cómo las regiones de Norteamérica, Europa y Asia-Pacífico tienen un impacto mucho mayor en el mercado que el resto. De entre estas 3 regiones, se espera que el mercado de Asia-Pacífico sea el principal contribuidor al mercado mundial de las FPGA [8], seguido de cerca por el mercado norteamericano y el europeo.

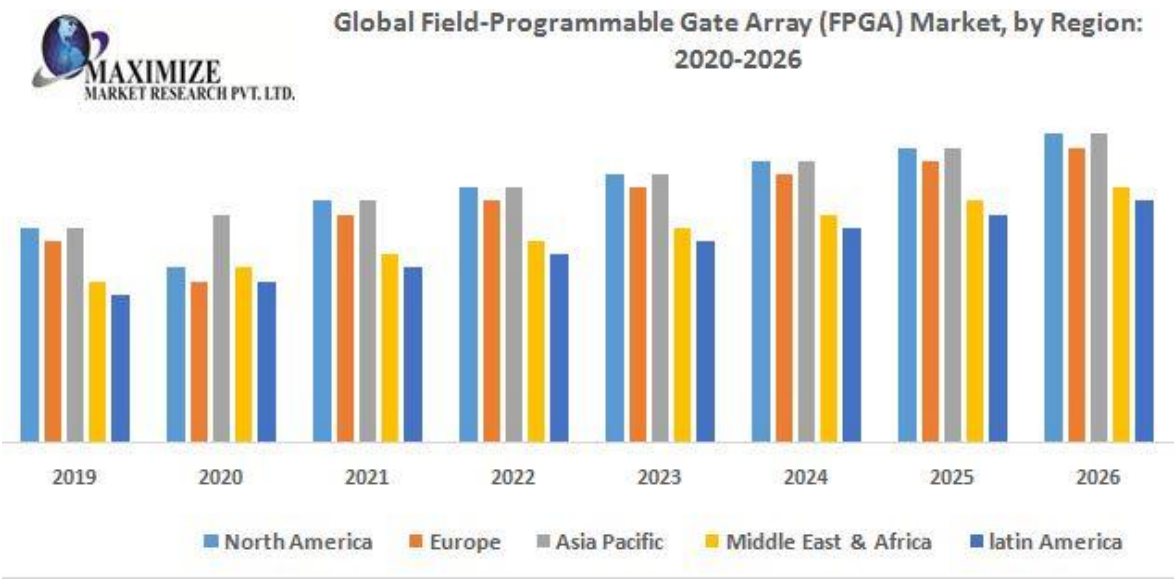


Figura 2. Mercado global de FPGA por región 2020-2026 [8].

En la Figura 3 se muestra la división del mercado mundial por sectores (comunicaciones, consumidor, procesamiento de datos, automovilístico, industrial y militar/aeroespacial) en el período de 2016 a 2021. El sector predominante es el de las comunicaciones con mucha diferencia frente al resto. Le siguen en segunda posición los sectores militar/aeroespacial, industrial y consumidor y algo más rezagados el sector de procesamiento de datos y automovilístico.

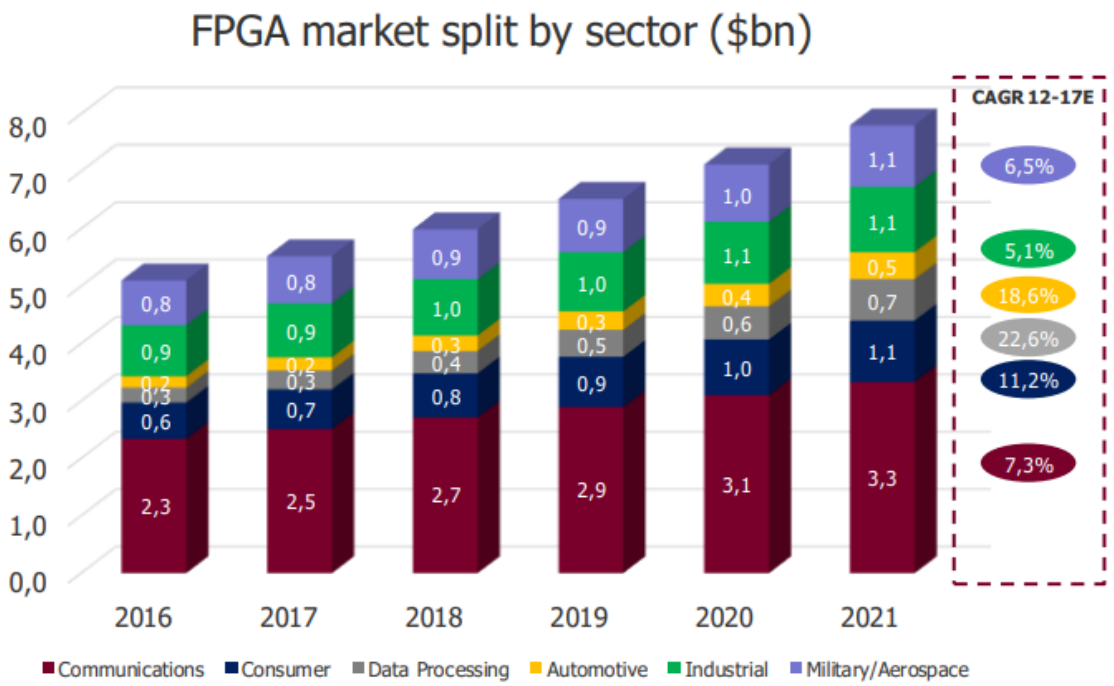


Figura 3. División del mercado de FPGA por sectores [9].

El uso tanto de FPGA como SoC FPGA en aplicaciones de control es objeto de gran interés, ya que es una tecnología flexible que permite implementar algoritmos de control para diferentes procesos industriales. Entre los campos de aplicación más destacados se encuentran:

a. Robótica ([10]–[12]).
b. Controladores empotrados para aplicaciones industriales ([13]–[15]).
c. Controladores para electrónica de potencia y sistemas de generación de energía eléctrica ([16]–[18]).

El campo de aplicación escogido para este TFM es el de controladores para electrónica de potencia y sistemas de generación de energía eléctrica. En concreto, se desarrolla un controlador para maximizar la producción de potencia de una instalación fotovoltaica que se encuentra afectada por sombras parciales. La implementación de este controlador se lleva a cabo siguiendo la metodología de diseño que se propone en este trabajo.

Dicha metodología consta de 3 puntos principales: la simulación MIL, la simulación SIL y la simulación *SoC in the Loop*. El funcionamiento de estas simulaciones se presenta en

detalle en el apartado 4.4 del Capítulo 4. Como se muestra en la Figura 4, todas estas simulaciones se llevan a cabo en conjunto con un modelo del sistema a controlar.

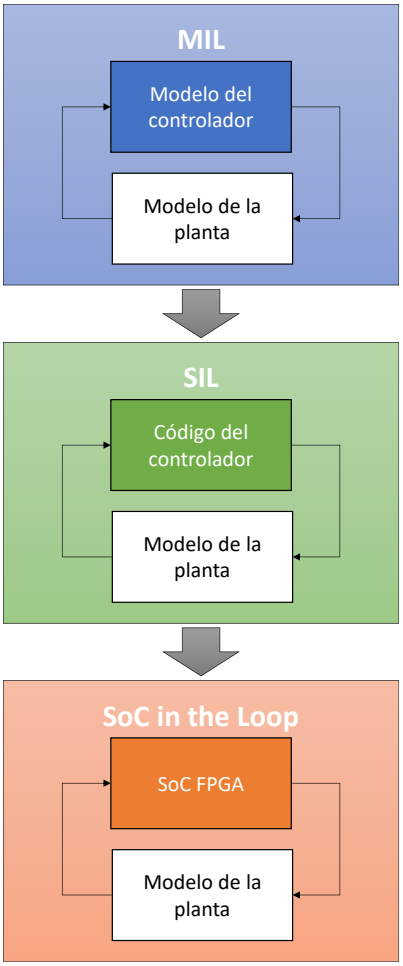


Figura 4. Modos de simulación de la metodología de diseño.

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Máster (TFM) consiste en validar la funcionalidad de la metodología de diseño de sistemas de control para dispositivos SoC FPGA que se presenta. Para ello se desarrolla un sistema de control para la producción de energía solar fotovoltaica desde modelos de alto nivel y se realiza su implementación sobre una placa de desarrollo usando dispositivos SoC FPGA Zynq.

Los objetivos operativos que se plantean son los siguientes:

O1. Estudio del estado del arte para los campos de aplicación y otras metodología de diseño existentes.

O2. Definir la arquitectura del sistema de control y realizar su modelo de alto nivel en MATLAB.

O3. Desarrollar los bloques de la arquitectura *hardware/software* para una implementación SoC FPGA.

O4. Realizar la integración del sistema sobre un dispositivo SoC FPGA.

O5. Validar, evaluar y documentar el trabajo realizado.

1.3. Peticionario

Actúa como peticionario de este TFM la División de Sistemas Industriales y CAD (SICAD) del Instituto Universitario de Microelectrónica Aplicada (IUMA) de la Universidad de Las Palmas de Gran Canaria (ULPGC).

Igualmente, este TFM permite satisfacer los requisitos de la asignatura Trabajo Fin de Máster, dentro del plan de estudios del Máster en Electrónica y Telecomunicación Aplicadas impartido por el IUMA.

1.4. Estructura del documento

En este primer capítulo se introduce al lector a los antecedentes y los objetivos planteados para este TFM. En el Capítulo 2 se realiza un análisis del estado del arte sobre el uso de FPGA en aplicaciones de control, haciendo hincapié en los campos de aplicación mencionados anteriormente, y sobre diferentes metodologías de diseño existentes. Tras realizar el análisis, en el Capítulo 3 se presenta el caso de implementación escogido para la verificación de la metodología de diseño.

En el Capítulo 4 se presenta la metodología de diseño propuesta, cuya funcionalidad se pretende validar, así como los recursos *software* y *hardware* que se precisan para esta tarea.

En el Capítulo 5 se introducen los algoritmos de reconfiguración dinámica de paneles fotovoltaicos (DPVAR, *Dynamic Photovoltaic Array Reconfiguration)* y se explica el funcionamiento y como se realiza la implementación del algoritmo escogido, así como su partición *hardware/software*.

En el Capítulo 6 se presentan los algoritmos de *Maximum Power Point Tracker* (MPPT) y se explica la implementación de la variante seleccionada.

En el Capítulo 7 se muestra el modelado del sistema a controlar en Simulink. Una vez se tiene este modelo se realiza la simulación *Model in the Loop* (MIL) y *Software in the Loop* (SIL) del algoritmo DPVAR y el controlador MPPT. Los resultados de las simulaciones se comparan para verificar el correcto funcionamiento del sistema.

En el Capítulo 8 se generan y se verifica el funcionamiento de los bloques IP correspondientes a la implementación en *hardware* del controlador MPPT y el algoritmo DPVAR.

En el Capítulo 9 se integran los bloques IP del controlador MPPT y del algoritmo DPVAR en una plataforma *hardware.* Dicha plataforma se vuelca sobre la placa de desarrollo y junto con la partición *software* del algoritmo DPVAR permite realizar una simulación *SoC in the Loop* y verificar el funcionamiento en conjunto del sistema sobre la placa de desarrollo.

Finalmente, se presentan los resultados y conclusiones obtenidas de la realización del proyecto y líneas de trabajo futuro, así como la bibliografía referenciada en el documento.

Capítulo 2. Estado del arte

2.1. Introducción

En este capítulo se hace un análisis del estado del arte en el uso de FPGA para la implementación de controladores en las áreas de robótica, controladores empotrados para aplicaciones industriales y electrónica de potencia y sistemas de generación de energía eléctrica y.

También, se presentan diferentes metodologías de diseño para realizar la implementación de algoritmos sobre tecnología FPGA.

2.2. Uso de FPGA para sistemas de control en robótica

El campo de la robótica se puede beneficiar en gran medida del uso de FPGA, ya que usualmente los sistemas robóticos integran una gran variedad y cantidad de sensores que generan un alto número de datos a procesar en tiempo real.

En caso de tratarse de sistemas con recursos energéticos limitados, como pueden ser drones o vehículos autónomos, el consumo de energía del controlador es un aspecto clave. El uso de CPU y GPU en robótica está muy extendido. El consumo de potencia de estas plataformas de cómputo oscila entre los 10 W y los 100 W, lo cual es relativamente elevado para sistemas robóticos con recursos limitados [10].

El uso de FPGA en estas situaciones permite procesar una gran cantidad de datos en tiempo real, aprovechando las posibilidades de paralelización que ofrecen y consiguiendo un consumo de potencia inferior a otras plataformas de cómputo. Ejemplos de uso de FPGA

en sistemas robóticos con recursos limitados se encuentran en [11], donde Divya et al. presentan un entorno inteligente para el parking de vehículos robotizados de uso en interiores utilizando dispositivos SoC FPGA Zynq [12]. En dicho trabajo se revisan diferentes aplicaciones para vehículos autónomos basados en la implementación de inteligencia artificial sobre FPGA. En esta misma línea, Yakun et al. [19] realizan la implementación de un algoritmo DS-SLAM de alta eficiencia energética sobre una plataforma heterogénea basada en FPGA.

Recientemente se han utilizado también FPGA, además de otras tecnologías, para implementar el cálculo del gradiente de la dinámica de cuerpos rígidos. Esta es una operación clave en la planificación del estado del arte y en los algoritmos de control para robótica [20].

La robótica modular, donde la dinámica de cada módulo se calcula por un controlador empotrado local, se beneficia del uso de FPGA. Esto se puede ver en [21] donde se consiguen frecuencias de muestreo mayores que utilizando un sistema centralizado.

Otros ejemplos de uso de FPGA en robótica modular aparecen en [22] donde Vinzenz Bargsten et al. proponen un método para el cálculo distribuido de la dinámica de movimiento de sistemas robóticos utilizando FPGA. Romanov et al. [23] proponen un nuevo enfoque de organización de un sistema de computación distribuida, basado en núcleos de computación especializados (SCCs) sobre FPGA para robots modulares reconfigurables (MRR). Además, la flexibilidad que ofrecen los SoC FPGA junto con la filosofía de la robótica modular, permite modificar fácilmente estos sistemas modulares [24].

En el ámbito de la robótica industrial también se pueden encontrar distintos casos de uso de FPGA. En [25] se presenta el desarrollo de un bloque IP para el cálculo de la cinemática directa e inversa de un robot SCARA (Figura 5). De igual forma, en [26] se mejora el movimiento mediante el desarrollo de un controlador basado en FPGA para robots industriales de arquitectura abierta. En [27] se describe el desarrollo de un controlador de arquitectura abierta para el robot PUMA 560 (Figura 6). Por último se presenta el desarrollo de un controlador basado en FPGA para robots colaborativos [28].

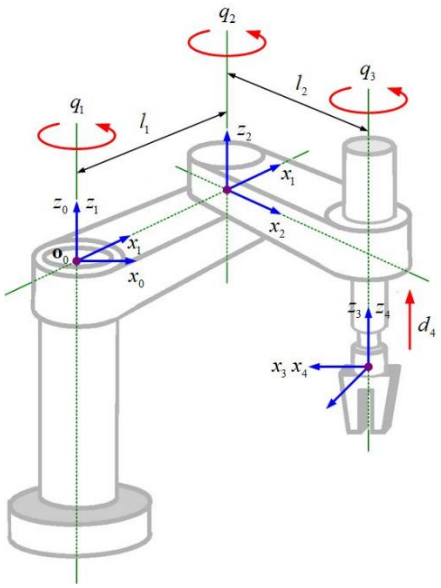


Figura 5. Robot SCARA [29].

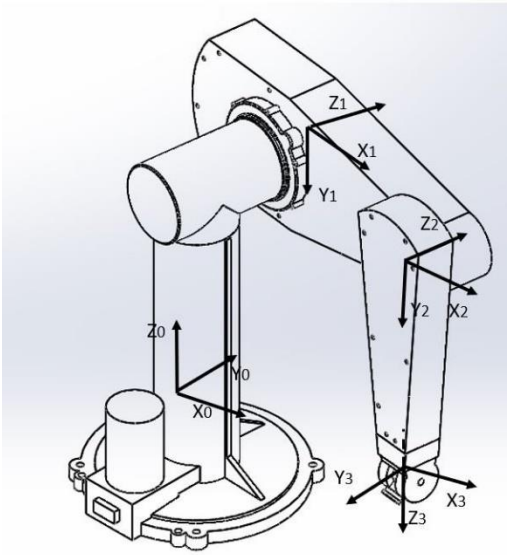


Figura 6. PUMA 560 [30].

2.3. Controladores empotrados para aplicaciones industriales basados en FPGA

Algunas de las aplicaciones más interesantes para sistemas empotrados son las relacionadas con la industria de la automoción y la industria aeroespacial. Los controladores que se desarrollan para estas aplicaciones deben cumplir en muchos casos especificaciones de tiempo real y para ello es muy frecuente que precisen de una alta capacidad computacional. Es por esto, que el uso de FPGA se ha incrementado notablemente en estos campos de aplicación [13].

Tanto para la industria automovilística como para la aeroespacial, el uso de controladores predictivos basados en modelo (*Model predictive Control*, MPC) ha cobrado gran relevancia, como se puede observar en las distintas publicaciones existentes, entre las que destacan [14], [15], [31], [32]. Estos son controladores capaces de trabajar con sistemas de múltiples variables y cuyo fin es resolver un problema de optimización para determinar la acción de control. En estos campos de aplicación el controlador MPC debe ser capaz de resolver el problema de optimización en tiempo real, generalmente en condiciones *hard-real time*. Al implementar estos controladores sobre FPGA, se puede aprovechar la capacidad de cómputo en paralelo que ofrecen estos dispositivos para lograr alcanzar los altos requerimientos [33].

Algunos casos de aplicación dentro de la industria aeroespacial se presentan a continuación. En [34] Hartley et al. implementan un MPC sobre FPGA para el control de un Boeing 747. Los mismos autores muestran en [35] la implementación de un MPC sobre FPGA y demuestran su funcionamiento con una simulación *FPGA-in-the-loop* en la que se controla una simulación no lineal de un avión de pasajeros de gran escala.

En la industria de la automoción también se pueden encontrar ejemplos de aplicación de MPC sobre FPGA. En [36], Reda et al. describen la implementación para realizar el control de la dirección automática en vehículos. Guo et al. [37] presentan el control de la estabilidad lateral del chasis activo de vehículos inteligentes. Dentro de esta misma línea, en [38] Petrelli presenta la implementación una plataforma para la realización del control necesario para evadir obstáculos y seguir una trayectoria a partir de la lectura de sensores, respetando las restricciones especificadas. Como se puede apreciar todas las aplicaciones se utilizan para sistemas críticos del automóvil.

En estos sistemas empotrados destinados a aplicaciones en el ámbito de la automoción o la aviación, la seguridad es un punto clave. En [39], Falk Salewski y Stefan Kowalewski llevan a cabo una comparativa entre microcontroladores y FPGA con respecto a sus propiedades de seguridad y fiabilidad en el ámbito de la automoción. Como resultado se muestra que las FPGA presentan ventajas en cuanto al encapsulamiento de funciones en tiempo real. Para reforzar el aspecto de la seguridad, en [40] Philippa Conmy e Iain Bate muestran cómo puede ser analizado un diseño modular empotrado sobre FPGA con el fin

de derivar las propiedades de fallo y seguridad para dar la evidencia necesaria para un caso de seguridad.

2.4. Control basado en FPGA para electrónica de potencia y sistemas de generación de energía eléctrica

El uso de FPGA en los sistemas electrónicos de potencia es de especial interés cuando se requiere un alto grado de paralelismo en la aplicación. Un ejemplo de aplicación donde se aprovecha esta característica se muestra en [16], donde se realiza una implementación *hardware* de un algoritmo de modulación por ancho de pulso del vector espacial (*multilevel multiphase space vector PWM,* SVPWM) para un convertidor de voltaje multinivel polifásico o en [17] donde se implementa la técnica de modulación por ancho de pulso sinusoidal (*sinusoidal pulse width modulation*, SPWM) para controlar un inversor multinivel de puentes H de 5 niveles (*H-Bridge multilevel invertir*, CHB-MLI). A la hora de trabajar con frecuencias de muestreo muy elevadas, el uso de FPGA también gana importancia. Esto ocurre al trabajar con fuentes conmutadas (SMPS) de baja tensión, tal como se presenta en [18].

Los dispositivos FPGA también se utilizan en electrónica de potencia para realizar el control de motores. Por ejemplo, para motores síncronos de imanes permanentes (PMSM) existen varias publicaciones sobre diferentes controladores. En [41] se propone un controlador compensador basado en filtros Kalman extendidos (*Extended Kalman Filters,* EKF) para mejorar las prestaciones del controlador proporcional integral (PI) ya implementado.

En [42] se desarrolla un controlador de corriente basado en control predictivo de estados finitos (*Finite-State Model Predictive Control*, FS-MPC). Se decide implementar sobre FPGA para explotar el paralelismo del algoritmo de control y conseguir una implementación que cumpla con especificaciones de tiempo real.

De igual manera, en [43] se propone otra aproximación para desarrollar un controlador de corriente basado en control predictivo por modelo (MPC), que se decide implementar en FPGA debido al alto coste computacional.

Finalmente en [44] se utiliza un sistema digital para la rotación de coordenadas (C*oordinate Rotation Digital Computer,* CORDIC) implementada sobre una FPGA para hacer la estimación de la posición y velocidad de un motor PMSM.

Los dispositivos FPGA también se utilizan para desarrollar controladores para sistemas de generación de energía eléctrica. Por ejemplo en, [3] se presentan 2 casos diferentes de control de sistemas eléctricos. El primero es el control, estimación y prognosis de un sistema de células de combustible (*Fuel Cells,* FC) híbrido y el segundo es la reconfiguración dinámica de módulos de energía solar fotovoltaica afectados por sombras parciales. En [45] se sigue en la línea del segundo caso y se implementa también un algoritmo de este tipo.

En FPGA también se pueden implementar los controladores MPPT que se utilizan en sistemas de generación eléctrica basado en turbinas eólicas o paneles fotovoltaicos. Estos controladores maximizan la extracción de potencia independientemente de las condiciones ambientales. Ejemplos de implementación de diferentes tipos de MPPTs sobre FPGA se pueden observar en las publicaciones [46]–[49].

2.5. Metodologías de diseño

Existen varías metodologías de diseño diferentes para llevar a cabo la implementación de algoritmos sobre tecnología SoC FPGA. Por ejemplo, en [50] Monmasson et al. presentan una metodología de diseño basada en tres aspectos principales: refinamiento del algoritmo, modularidad y optimización de la implementación.

Con el refinamiento del algoritmo se busca su adaptación a una implementación sobre FPGA cambiando, por ejemplo, la aritmética en coma flotante por punto fijo. Por otro lado, la modularidad disminuye el ciclo de diseño en sistemas complejos ya que maximiza la reutilización de los módulos diseñados. Finalmente, se propone utilizar la metodología A^3 (*Algorithmic Architecture Adequation)* [51] para encontrar la arquitectura *hardware* óptima del algoritmo que se quiere implementar para que cumpla con las restricciones de tiempo y área establecidas.

Otro ejemplo de metodología se presenta en [52], donde se describen los pasos necesarios para, a partir de la implementación HDL del algoritmo deseado, obtener el *bitstream* de programación de la FPGA. Los pasos a seguir en esta metodología para generar el *bitstream* son: síntesis, implementación, mapeado, colocado de bloques e interconexionado. Esta es una metodología similar a la que se sigue en Vivado Design Suite para generar el *bitstream* de programación de la FPGA.

También existen metodologías diseñadas para herramientas específicas como puede ser MATLAB/Simulink. Por ejemplo, en [53] se presenta una metodología de diseño en la que se implementa el algoritmo deseado en Simulink y se genera código VHDL a partir de este. Para verificar el funcionamiento del código generado, se realiza su co-simulación con ayuda de un simulador HDL. Por otro lado, en [54] se propone una metodología que combina la síntesis de alto nivel mediante Vivado HLS con MATLAB/Simulink. En esta metodología se implementa el algoritmo deseado en C/C++/SystemC y se utiliza Vivado HLS para generar el bloque IP correspondiente al algoritmo. Tras esto, se verifica el funcionamiento del bloque IP en Simulink mediante una co-simulación con un simulador HDL de manera similar a [53].

2.6. Conclusiones

En este capítulo se ha realizado el estudio de distintos trabajos presentes en la literatura en relación con los sistemas de control basados en dispositivos FPGA, mostrando sus ventajas y desventajas para aplicaciones en el ámbito industrial, incluida la generación de energías renovables y el transporte inteligente. También se han presentado diversas variantes de metodologías de diseño para dispositivos SoC FPGA.

Capítulo 3. Caso de implementación

3.1. Introducción

En este capítulo se presenta el área de aplicación y el caso de implementación escogidos. El caso de implementación estudiado en este capítulo se utiliza como caso práctico para verificar la funcionalidad de la metodología de diseño que se propone en el Capítulo 4.

3.2. Área de aplicación

En el área de aplicación de controladores para electrónica de potencia y generación de energía eléctrica se ha optado por implementar un algoritmo DPVAR, junto con un controlador MPPT.

Existen varios ejemplos de implementación de diversas variantes de controladores MPPT sobre tecnología FPGA ([46]–[49]), mostrando la viabilidad de la tecnología FPGA para la implementación de este tipo de controladores. Por otro lado, existen también algunos casos de implementación de algoritmos DPVAR sobre SoC FPGA ([55]).

Los dispositivos SoC FPGA representan un buen candidato para la implementación de este tipo de algoritmos debido a la flexibilidad que ofrecen. Generalmente los algoritmos DPVAR son complejos y conllevan un alto coste computacional. Poder realizar una implementación *hardware/software* de estos algoritmos permite, por ejemplo, implementar en C/C++ sobre el procesador integrado las partes más secuenciales del algoritmo y en *hardware* las partes de mayor coste computacional. En la partición *hardware*

del algoritmo se explota el paralelismo existente para conseguir reducir el tiempo de ejecución total frente a una implementación puramente *software*.

3.3. Caso de estudio

El algoritmo DPVAR que se escoge implementar es el que se presenta en [56]. Este es un algoritmo relativamente nuevo y que no ha sido implementado aún sobre tecnología SoC FPGA. Se ha elegido este algoritmo porque se han identificado varias partes que se pueden implementar de forma paralela en *hardware* y mejorar así el tiempo de ejecución total. En conjunto con el algoritmo DPVAR se implementa un controlador MPPT de conductancia incremental.

El funcionamiento del algoritmo DPVAR y el controlador MPPT se simula en conjunto con el modelo del sistema a controlar (Figura 7). Está formado por un *array* de nueve paneles fotovoltaicos, una matriz de interruptores, un convertidor reductor (*Buck Converter*) y una carga. El modelo del sistema a controlar se presenta con mayor detalle en el Capítulo 7.

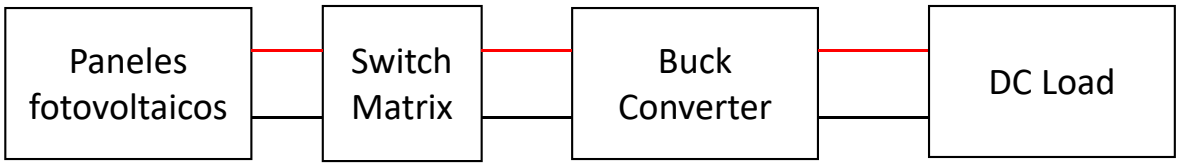


Figura 7. Diagrama de bloques del modelo del sistema controlar.

3.4. Conclusiones

En este capítulo se ha presentado el área de aplicación escogida, así como el caso de implementación. Se justifica la implementación del algoritmo DPVAR elegido sobre SoC FPGA debido a la capacidad de explotar el paralelismo que ofrece el mismo al utilizar esta tecnología.

Capítulo 4. Metodología de diseño

4.1. Introducción

En este capítulo se presenta la metodología de diseño que se propone junto con las herramientas necesarias para implementarla. En primer lugar, se introduce MATLAB/Simulink, entorno que se utiliza para realizar la implementación del algoritmo DPVAR y el controlador MPPT y verificar su correcto funcionamiento. A continuación, se explica brevemente la herramienta de implementación del diseño, Xilinx Vivado Design Suite. Por último, se presenta paso a paso la metodología de diseño propuesta en este TFM para sistemas de control sobre dispositivos SoC FPGA.

4.2. MATLAB/Simulink

MATLAB (MATrix LABoratory) [57] es un entorno de programación y cálculo numérico. Entre sus muchas aplicaciones se encuentran el desarrollo de algoritmos, análisis de datos o generación automática de código. El lenguaje de programación de MATLAB está basado en matrices y puede ser convertido automáticamente en código C/C++ y HDL.

Simulink [58] provee un entorno gráfico basado en bloques en el que se pueden realizar simulaciones multidominio y diseño basado en modelos. Entre las capacidades con las que cuenta esta herramienta están la generación automática de código, la realización de simulaciones y la comprobación y verificación de sistemas integrados.

MATLAB y Simulink son herramientas diseñadas para trabajar en conjunto. Se puede hacer uso de los algoritmos diseñados en MATLAB en los modelos creados en Simulink y a

su vez, se pueden exportar los datos generados en las simulaciones realizadas en Simulink a MATLAB para su análisis.

MATLAB y Simulink cuentan con una gran variedad de *Add-Ons* denominados *toolbox*. Estos *toolbox* permiten añadir funcionalidades al entorno como, por ejemplo, la posibilidad de generar bloques IP para dispositivos FPGA a partir de algoritmos desarrollados en MATLAB/Simulink.

A continuación, se describe la funcionalidad de los diferentes *toolbox* que son necesarios para realizar la implementación del algoritmo DPVAR y el controlador MPPT.

4.2.1. Simscape Electrical

Simscape Electrical [59] es un *toolbox* de Simulink que proporciona diversas librerías de componentes que permiten modelar y simular sistemas electrónicos, mecatrónicos y de potencia. En estas librerías se encuentran modelos de semiconductores, generadores de energías renovables o motores entre otros.

En este TFM se utilizan diversos componentes de estas librerías para modelar el sistema fotovoltaico a controlar.

4.2.2. Fixed-Point Designer

Fixed-Point Designer [60] es un *toolbox* que proporciona herramientas para generar código en punto fijo o punto flotante a partir de algoritmos generados en MATLAB/Simulink. También permite optimizar e implementar código para *hardware* empotrado.

En este TFM se utiliza la aplicación Fixed-Point Converter proporcionada por el *toolbox* Fixed-Point Designer para generar una implementación en punto fijo de la partición *hardware* del algoritmo DPVAR y el controlador MPPT que se desarrollan en MATLAB/Simulink.

4.2.3. MATLAB Coder

MATLAB Coder [61] es un *toolbox* que permite generar código C/C++ a partir de código en lenguaje MATLAB. Admite la mayoría de las expresiones del lenguaje MATLAB y

una gran variedad de *toolboxes*. El código que se genera es portable y se puede combinar con código C/C++ ya existente.

4.2.4. Simulink Coder

Simulink Coder [62] es un *toolbox* que ejecuta y genera código C/C++ de modelos de Simulink, gráficos de Stateflow y funciones de MATLAB. El código generado se puede usar, por ejemplo, en aplicaciones con o sin especificaciones de tiempo real o comprobaciones *Hardware-in-the-Loop*.

4.2.5. Embedded Coder

Embedded Coder [63] es un *toolbox* que complementa a MATLAB Coder y Simulink Coder y permite generar código C/C++ leíble, compacto y rápido para procesadores empotrados. También añade optimizaciones para mejorar la eficiencia del código y la facilidad de su integración con código previamente existente.

4.2.6. HDL Coder

HDL Coder [64] es un toolbox que permite generar código Verilog y VHDL portable y sintetizable a partir de funciones de MATLAB, modelos de Simulink y gráficos Stateflow. El código generado se puede utilizar para programar FPGA o prototipar en ASICs. HDL Coder provee un flujo de trabajo que automatiza y guía en la tarea de programar FPGA de Xilinx, Microsemi e Intel. El código generado por HDL Coder tiene trazabilidad con el modelo de Simulink facilitando la verificación de este.

4.2.7. HDL Coder Support Package for Xilinx Zynq Platform

HDL Coder Support Package for Xilinx Zynq Platform [65] es un paquete adicional del toolbox HDL Coder. Permite generar bloques IP que se pueden exportar a Xilinx Vivado o Vivado ISE y realizar diseños para dispositivos de la plataforma Zynq de Xilinx.

4.2.8. Embedded Coder Support Package for Xilinx Zynq Platform

Embedded Coder Support Package for Xilinx Zynq Platform [66] es un paquete adicional del toolbox Embedded Coder. Permite generar código ANSI C a partir de funciones de MATLAB, gráficos Stateflow y modelos de Simulink para los núcleos ARM de los dispositivos SoC Xilinx Zynq.

Si se usa en combinación con HDL Coder Support Package for Xilinx Zynq Platform, se puede programar directamente el dispositivo Xilinx Zynq con el código C y HDL generado siguiendo el codiseño *hardware/software* (Figura 8). Esto permite realizar simulaciones, verificar el funcionamiento del código y los bloques IP y realizar la implementación final.

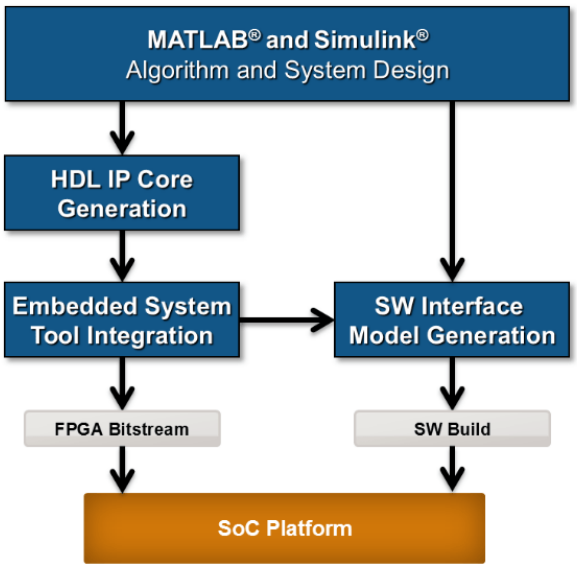


Figura 8. Codiseño hardware/software para plataformas SoC [67].

4.3. Herramientas de síntesis e implementación para FPGA

En la síntesis lógica se obtiene la implementación del diseño a nivel lógico a partir de las especificaciones de funcionamiento RTL, es decir, se generan las puertas lógicas a partir del RTL. A la fase de síntesis le sigue una fase de implementación y de generación del *bitstream*. Todo este proceso es dependiente de la tecnología y por tanto será necesario utilizar los entornos de los fabricantes de los dispositivos FPGA. Para este TFM se utilizan dispositivos Zynq de Xilinx, por lo que se utilizará Vivado Design Suite.

4.3.1. Xilinx Vivado Design Suite

Xilinx Vivado Design Suite [68] tiene una *Graphical User Interface* (GUI) que permite realizar diferentes tareas de diseño y crear la plataforma *hardware* donde se implementan los diferentes bloques IP desarrollados. Igualmente, dispone de la posibilidad de trabajar en modo de línea de comandos.

La implementación de la plataforma *hardware* se realiza en varias etapas. Estas etapas son el análisis, la síntesis lógica y la implementación. En todo momento, durante la ejecución de estas etapas, se pueden realizar mejoras en el consumo de recursos, consumo de potencia y prestaciones temporales. Tras cumplir con estas tres etapas, el siguiente paso es la generación del *bitstream*. Mediante este *bitstream* se programa la FPGA con la plataforma *hardware* diseñada.

Para la implementación final hay que tener en cuenta que la programación de las FPGA de Xilinx es volátil debido a que utilizan tecnología basada en SRAM. Por lo tanto, para poder asegurar el funcionamiento tras una pérdida de la alimentación del sistema, hay que descargar el *bitstream* en una tarjeta SD o memoria Flash para que el sistema puede arrancar correctamente al reiniciarlo.

4.4. Metodología de diseño

La metodología propuesta comienza con el diseño en MATLAB/Simulink de los algoritmos que se quieren implementar en el SoC FPGA. A continuación, se procede a realizar el modelado de la planta a controlar en Simulink, para lo cual se utilizan bloques pertenecientes al *toolbox* Simscape Electrical. Con el modelo de la planta se realiza la simulación *Model in the Loop* (MIL) (Figura 9), que consiste en simular conjuntamente los algoritmos diseñados en MATLAB/Simulink con el modelado de la planta creado [69]. Con ello se verifica el correcto funcionamiento de los algoritmos. Si el resultado de la simulación MIL es satisfactorio, se puede pasar al siguiente paso.

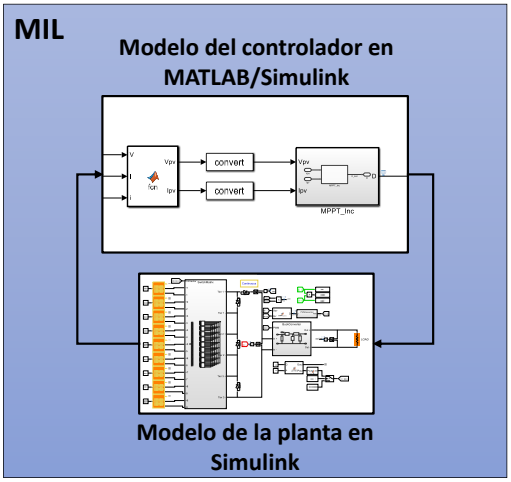


Figura 9. Simulación MIL.

Tras la simulación MIL, el siguiente paso es realizar la simulación *Software in the Loop* (SIL) (Figura 10). Consiste en generar código C/C++ a partir de los algoritmos diseñados en MATLAB/Simulink mediante el *toolbox* Simulink Coder y simular su funcionamiento en conjunto con el modelado del sistema [70]. El objetivo de la simulación SIL es verificar que MATLAB es capaz de generar código C/C++ a partir de los algoritmos diseñados, que este funciona de manera análoga a los algoritmos originales y que es implementable en *hardware*. En caso de que los resultados de simulación no sean satisfactorios, se revisa el diseño de los algoritmos para asegurar que se pueda generar código y que sean implementables en *hardware*.

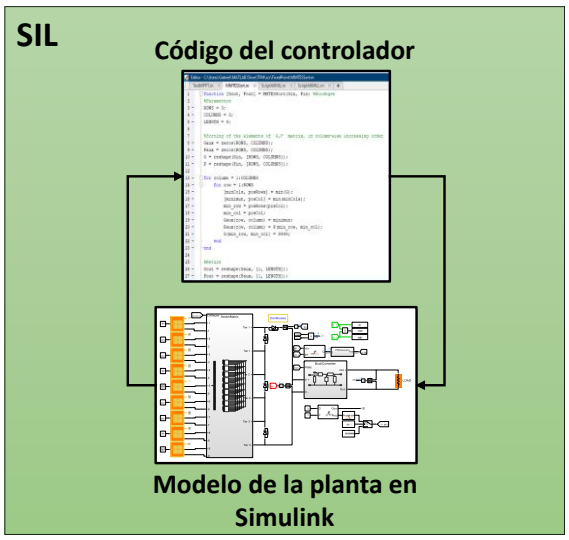


Figura 10. Simulación SIL.

Finalizada la simulación SIL, se realiza la generación de bloques IP. En este punto se genera código HDL, VHDL o Verilog, y se crean los bloques IP para aquellas partes de los algoritmos diseñados que se quieran implementar en la lógica programable de la FPGA. Para ello, primero se genera una implementación en punto fijo de las partes destinadas a implementarse en *hardware* mediante el *toolbox* Fixed-Point Designer y tras esto se generan los bloques IP mediante el *toolbox* HDL Coder. Para verificar el funcionamiento de los bloques IP generados, se exportan a Vivado Design Suite donde se genera el *bitstream* con el que programar la FPGA. Una vez programada la FPGA y gracias a los *Support Package* del *toolbox* HDL Coder, se puede realizar la simulación *SoC in the Loop* (Figura 11). El fin de esta simulación es verificar el correcto funcionamiento de los bloques IP diseñados, que

han sido programados en la FPGA, simulando su funcionamiento en conjunto con el modelo del sistema en Simulink.

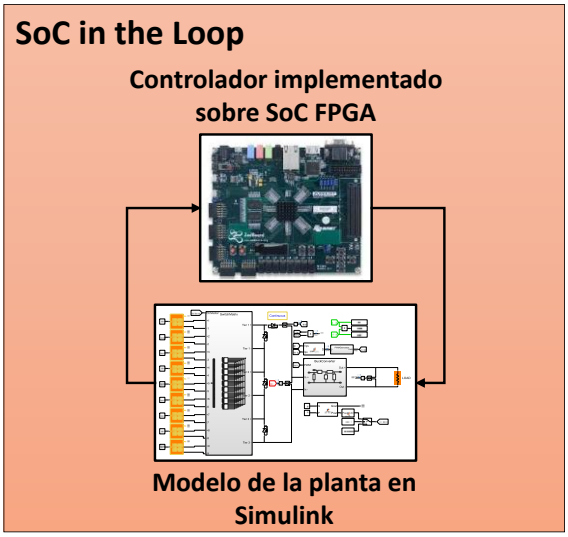


Figura 11. Simulación *SoC in the Loop*.

Por último, se realiza una simulación *SoC in the Loop* del sistema al completo con el modelo de la planta creado en Simulink. Para ello se crea en Vivado Design Suite una plataforma *hardware* que integre todos los bloques IP desarrollados, se genera el *bitstream* y se programa la FPGA. A continuación, desde Simulink se genera el código ANSI C de la parte *software* de los algoritmos diseñados mediante el *toolbox* Embedded Coder y se carga en el procesador de la plataforma SoC FPGA. Hecho esto, se simula el funcionamiento del sistema en conjunto con el modelo de la planta. Si la simulación es satisfactoria, el flujo de diseño concluye.

En la Figura 12 se ilustra la metodología presentada mediante un diagrama de flujo. En la Figura 13 se relaciona de forma gráfica los pasos que constituyen el flujo de diseño propuesto, con las herramientas presentadas en los apartados 4.2 y 4.3 necesarias para llevarlos a cabo.

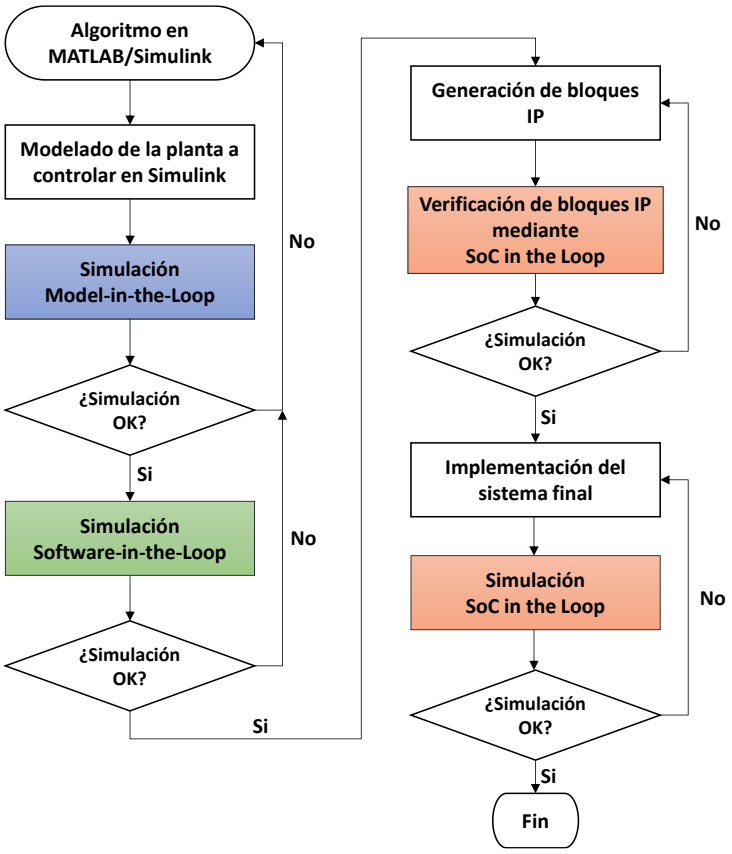


Figura 12. Flujo de diseño.

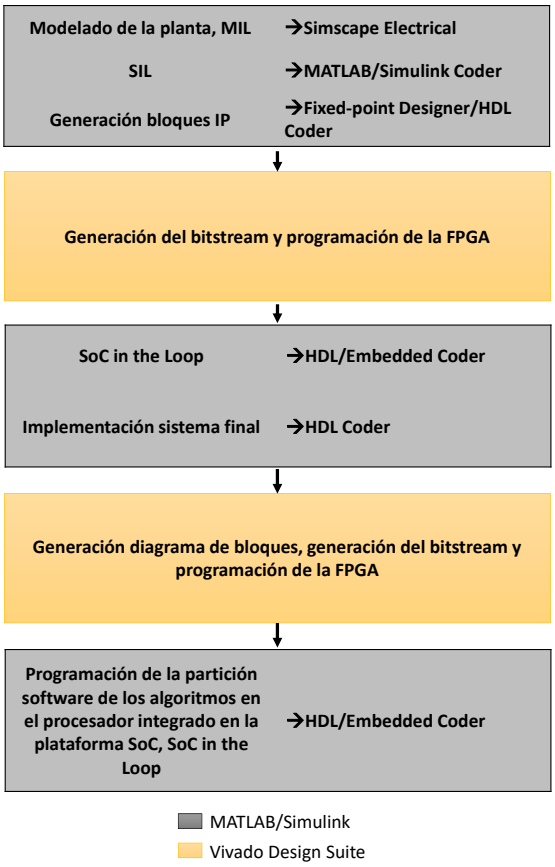


Figura 13. Distribución de los pasos del flujo de diseño según las herramientas necesarias.

4.5. Conclusiones

En este capítulo se han introducido las diferentes herramientas necesarias y se ha descrito el flujo de diseño para poder realizar la implementación de un controlador MPPT y un algoritmo DPVAR sobre un dispositivo SoC FPGA siguiendo la metodología de diseño propuesta.

Por otro lado, se ha expuesto la metodología de diseño a seguir y se ha explicado, con la ayuda de un diagrama de flujo, cada uno de los pasos que hay que realizar para llevarla a cabo.

Capítulo 5. Algoritmo de reconfiguración dinámica de paneles fotovoltaicos

5.1. Introducción

El rendimiento de una instalación fotovoltaica se puede ver afectado por diversos motivos, como por ejemplo la aparición de sombras que afecten a un determinado número de paneles fotovoltaicos de la instalación. Un ejemplo se puede ver en la Figura 14, donde los paneles afectados por sombras están remarcados en gris. Esto se denomina sombras parciales y pueden ser provocadas por nubes, árboles, antenas de telefonía móvil, etc. Las sombras parciales afectan sobre todo a instalaciones fotovoltaicas de gran extensión haciendo que la curva característica de ésta sea más compleja, lo que dificulta optimizar la producción de potencia [71].

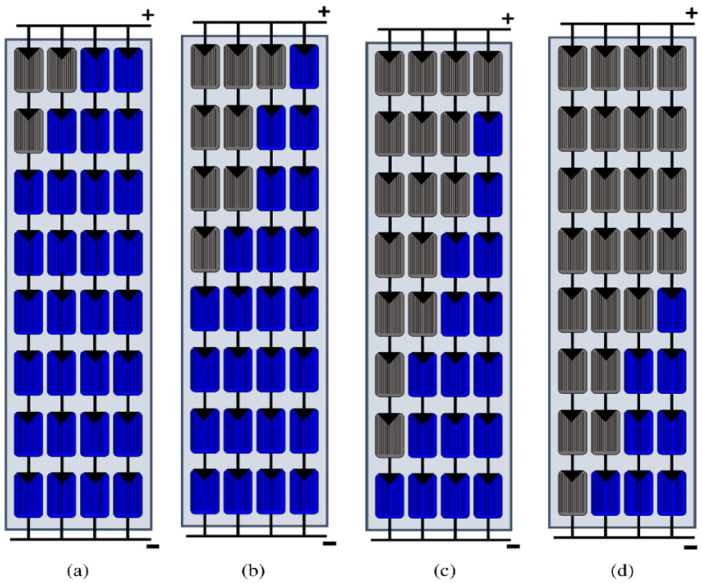


Figura 14. Ejemplo de sombras parciales sobre una instalación fotovoltaica [72].

Para intentar mejorar la producción de potencia de una instalación fotovoltaica que sufra de sombras parciales existen varios métodos. Uno de los métodos más frecuentes es el uso de distintas configuraciones del *array* fotovoltaico. Entre los diferentes tipos de configuraciones que existen, las configuraciones denominadas *Total Cross Tied* (TCT) y *Honey Comb* (HC) son las que mejor resultados ofrecen. La configuración TCT es la mejor cuando el *array* fotovoltaico es simétrico y la configuración HC cuando es asimétrico [73].

Otra de las opciones existentes es el uso de un algoritmo de reconfiguración dinámica de paneles fotovoltaicos (*Dynamic Photovoltaic Array Reconfiguration,* DPVAR). Estos son algoritmos que con la ayuda de interruptores alteran el conexionado de los paneles fotovoltaicos en la instalación para intentar maximizar la producción de potencia y paliar los efectos de las sombras parciales. Hay una gran variedad de algoritmos DPVAR que utilizan diferentes métodos para lograr mejorar la producción de energía. Entre estos métodos están la resolución de problemas de optimización o la ecualización de irradiancias. Un ejemplo de algoritmo DPVAR del tipo resolución de problemas de optimización está en [74], donde Kazerani et al. proponen una formulación matemática novedosa para la reconfiguración óptima de paneles fotovoltaicos con el fin de minimizar las pérdidas producidas por las sombras parciales. En esta publicación se formula el problema como un problema de programación cuadrática entera-mixta y se obtiene la solución óptima mediante un algoritmo de ramificación y poda. Por otro lado, los algoritmos de ecualización de irradiancias se aplican en *arrays* fotovoltaicos con una configuración TCT. El fin es conseguir que el total de irradiancia solar que recibe cada una de las filas de la configuración TCT sea equivalente. Esto se puede lograr de diferentes maneras como se muestra en [75] donde se presentan dos formas posibles de hacerlo: una basada en un algoritmo de *sorting* y otra basada en control predictivo por modelo.

En este capítulo se presenta el algoritmo DPVAR escogido para implementar sobre tecnología SoC FPGA, se explica su funcionamiento y se implementa en MATLAB. Tras esto se proponen mejoras a la implementación en MATLAB para adaptarlo a una implementación *hardware/software* que pueda aprovechar las características que ofrece la tecnología SoC FPGA.

5.2. Algoritmo MMTES

El algoritmo DPVAR escogido es el *Maximum-Minimun Tier Equalisation Swapping* (MMTES) [56]. Se escoge implementar este algoritmo porque es un algoritmo novedoso para ser implementado sobre SoC FPGA y porque se han identificado varios pasos de este que se pueden modificar para adaptarlos a una implementación sobre esta tecnología y aprovechar la capacidad de cómputo en paralelo que ofrece.

En los siguientes apartados se introducen las características principales de este algoritmo, su funcionamiento, requisitos y su implementación en MATLAB. También se proponen una serie de modificaciones para adaptarlo a una implementación sobre tecnología SoC FPGA.

5.2.1. Características principales

El algoritmo MMTES busca mejorar la potencia de salida de una instalación fotovoltaica que se encuentra bajo los efectos de sombras parciales. Para esto, dado que la corriente generada por los paneles solares es proporcional a la irradiancia que incide sobre ellos, se propone igualar la irradiancia de cada uno de los *tiers* de los que está formada la instalación, o lo que es lo mismo, igualar la corriente de los diferentes *tiers*. En la Figura 15 se ilustra un ejemplo de instalación formada por 9 paneles fotovoltaicos dispuestos según una configuración TCT 3x3 [76]. En este ejemplo hay un total de 3 *tiers* y cada uno de ellos coincide con una de las filas de la matriz 3x3. El objetivo del algoritmo MMTES es que la suma de irradiancias por filas sea equivalente.

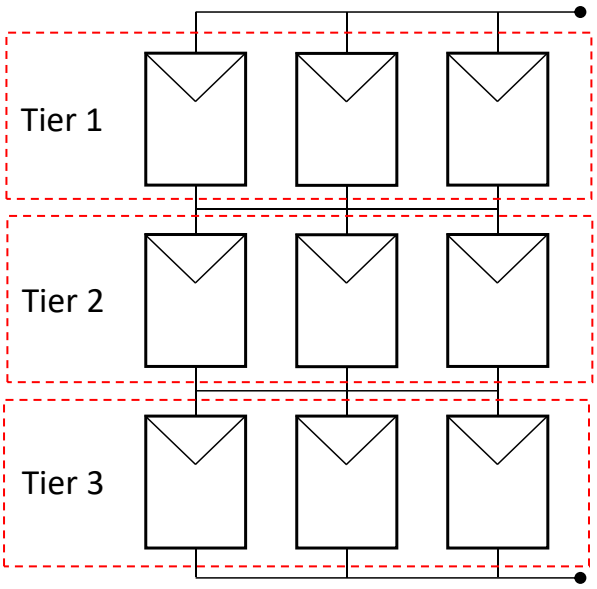


Figura 15. Configuración TCT 3x3.

El algoritmo MMTES se puede aplicar para cualquier tamaño de *array* fotovoltaico, ya sea simétrico, como por ejemplo un *array* 3x3, o asimétrico, como puede ser un *array* 3x4. Para poder modificar el conexionado de los paneles fotovoltaicos se necesita de una matriz de interruptores (*switch matrix)*. Esta matriz de interruptores es una red de interruptores que permite variar virtualmente la posición de los módulos fotovoltaicos en la instalación, sin tener que mover físicamente los paneles solares, cambiando las posiciones de los módulos hasta alcanzar la igualdad de irradiancias entre *tiers*.

5.2.2. Funcionamiento

El funcionamiento del algoritmo MMTES se ilustra en la Figura 16 mediante un diagrama de flujo. El algoritmo se divide en un total de 7 pasos:

1. Primero se inicializan las matrices G y P de tamaño (m x n). La matriz G contiene los valores de la irradiancia que incide sobre cada uno de los paneles fotovoltaicos y la matriz P su posición en el *array* (m x n). También se inicializa la matriz Z de tamaño (m x 1) que contiene los valores del sumatorio por filas de la matriz G.
2. El segundo paso es el cálculo de $set\ value$ (5.1). El objetivo principal del algoritmo es hacer que el valor de todos los componentes de Z sea lo más cercano posible al valor de $set\ value$.

$$set\ value = \frac{\sum_{i=1}^{i=m}\sum_{j=1}^{j=n} G_{ij}}{m} \tag{5.1}$$

3. El siguiente paso en el algoritmo es la ordenación de los elementos de la matriz G en orden creciente por columnas. Los cambios de posición de los elementos de la matriz G se aplican de forma homóloga a los elementos de la matriz P. Tras esta ordenación se realiza el cálculo de los elementos de la matriz Z. Si cada uno de los elementos de Z es igual al valor de $set\ value$ calculado en el segundo paso, se determina que las irradiancias de cada uno de los *tiers* son iguales y por lo tanto el algoritmo termina. En caso contrario el algoritmo prosigue con el siguiente paso.

4. En el cuarto paso se determinan Min_row y Max_row siendo estos el índice del menor y el mayor elemento en Z respectivamente.

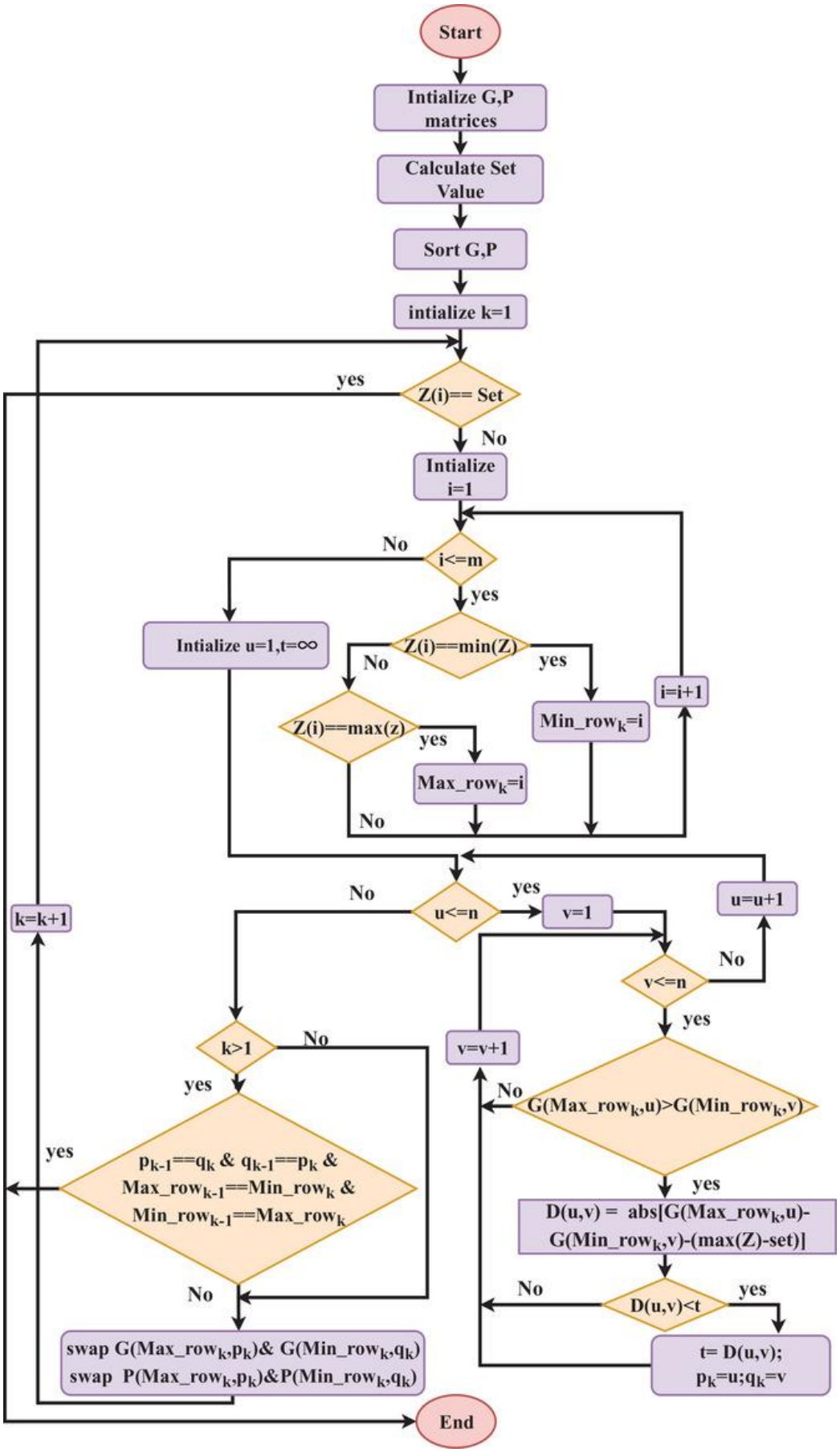


Figura 16. Diagrama de flujo del algoritmo MMTES [56].

Para conseguir igualar las irradiancias de todos los *tiers* de G, es necesario intercambiar algunos elementos entre sus filas con índices Min_row y Max_row. De esta manera es posible acercar los valores de Z al valor de $set\ value$ calculado.

5. En el paso número 5 se determinan los posibles pares de cambio existentes entre estas 2 filas. Un par de valores es un posible par de cambio si cumple con la ecuación (*5.2*).

$$G(Max\_row, u) > G(Min\_row, v) \quad \forall \quad 1 \leq u \leq n; 1 \leq v \leq n \tag{5.2}$$

6. En el paso número 6 se determina el par de cambio entre los posibles pares de cambio obtenidos en el paso anterior. Para cada uno de estos pares, se calcula D según la ecuación (*5.3*). El par de elementos que corresponda con (*5.4*) se determina como la mejor opción para realizar el intercambio. La posición de estos elementos se guarda en pk para el elemento perteneciente a la fila con índice Max_row y qk para el elemento perteneciente a la fila con índice Min_row.

$$D(u, v) = abs[G(Max\_row, u) - G(Min\_row, v) - (\max(Z) - set\ value)] \quad \forall \quad 1 \leq u \leq n; 1 \leq v \leq n \tag{5.3}$$

$$\min(D(u, v)) \tag{5.4}$$

7. En el último paso se intercambia el elemento pk correspondiente a la fila Max_row de G con el elemento qk correspondiente a la fila Min_row de G. Este intercambio de elementos también se realiza en la matriz de posiciones P. Si tras el intercambio los valores de Z siguen siendo diferentes del valor de $set\ value$, se vuelve al paso 4 y se repite el proceso. Se continuará iterando entre el paso 4 y el 7 hasta que los valores de Z sean iguales al valor de $set\ value$ o los elementos intercambiados en la iteración anterior sean los mismos que se van a intercambiar en la presente iteración. Si en dos iteraciones consecutivas se elige el mismo par de elementos para cambio, el algoritmo termina.

5.2.3. Matriz de interruptores

En los algoritmos DPVAR, la matriz de interruptores es un elemento muy importante ya que permite al algoritmo variar la topología de la instalación fotovoltaica. Para este algoritmo se propone una red de interruptores unipolares de un solo corte (*Single Pole Single Throw,* SPST). El número total de interruptores S_T para un *array* fotovoltaico de tamaño (m x n) se calcula mediante la ecuación (*5.5*).

$$S_T = S_P * S$$
$$donde\ S_P = 2 * m; S = m * n \tag{5.5}$$

En la Figura 17 se ilustra el conexionado de un panel fotovoltaico a una matriz de interruptores para un *array* (3 x 3).

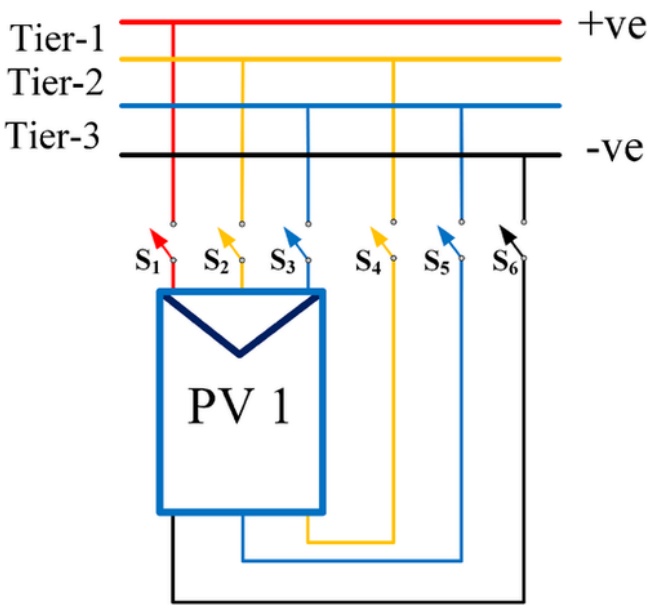


Figura 17. Ejemplo de matriz de interruptores para un array 3 x 3 [56].

La Tabla 2 muestra el *tier* al que estaría conectado el panel fotovoltaico de la Figura 17 dependiendo del estado de los interruptores.

Tabla 2. Estados posibles de los interruptores de la matriz de interruptores para un *array* 3 x 3.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**
Tier 1	ON	OFF	OFF	ON	OFF	OFF
Tier 2	OFF	ON	OFF	OFF	ON	OFF
Tier 3	OFF	OFF	ON	OFF	OFF	ON

5.3. Implementación del algoritmo MMTES en MATLAB

Una vez se ha estudiado el funcionamiento del algoritmo MMTES, se realiza su implementación en MATLAB. Para ello se ha divido el algoritmo en 3 funciones principales:

1. La función *MMTESSort*
2. La función *PairForSwapping*
3. La función *SwapPair*

5.3.1. Función MMTESSort

La Función *MMTESSort* ordena los valores de irradiancia de la matriz G en orden creciente por columnas. Cada cambio de posición que se realice sobre los elementos de la matriz G se debe aplicar también sobre la matriz de posiciones P para preservar la correlación de irradiancia-posición que existe entre ambas.

La función *MMTESSort* comienza con la inicialización de las matrices $Gaux$ y $Paux$ con todos sus valores a 0 y el mismo tamaño que G y P. A continuación, se entra en un bucle que itera a través de todas las posiciones de la matriz $Gaux$ y $Paux$ por columnas. En cada iteración de este bucle se calcula el valor mínimo de G y su posición en la matriz. Después de esto, se copia dicho valor a la matriz $Gaux$ en la posición correspondiente a la presente iteración, y a su vez se copia en la matriz $Paux$ el elemento de la matriz P cuya posición coincida con la del valor mínimo de la matriz G. Tras esto se cambia el valor del elemento mínimo de la matriz G por uno que sea muy superior al máximo valor presente en esta matriz. El código fuente de esta función se muestra en el Código fuente 1 .

Código fuente 1. Función MMTESSort

```
function [Gout, Pout] = MMTESSort(G, P) %#codegen
%Parameters
ROWS = 3;
COLUMNS = 3;

%Sorting of the elements of G & P matrix, in column-wise increasing order
Gaux = zeros(ROWS, COLUMNS);
Paux = zeros(ROWS, COLUMNS);

%Iterates through all values of matrix G
for column = 1:COLUMNS
    for row = 1:ROWS
%Calculation of the minimum value of G and its position
```

Código fuente 1 (cont.). Función MMTESSort

```
        [minCols, posRows] = min(G);
        [minimum, posCol] = min(minCols);
        min_row = posRows(posCol);
        min_col = posCol;

%Copy of the minimum value of G in Gaux in column-wise increasing order
        Gaux(row, column) = minimum;
%Copy the value of the matrix P at position (min_row, min_col) to the
matrix paux
        Paux(row, column) = P(min_row, min_col);
        G(min_row, min_col) = 9999;
    end
end

%Return
Gout = Gaux;
Pout = Paux;
end
```

5.3.2. Función PairForSwapping

La Función *PairForSwapping* determina el índice de los elementos que serán intercambiados para tratar de aproximar el valor de Z lo más posible al valor de referencia $set\ value$.

Esta función comienza con la inicialización de las variables pk y qk a 0 y la variable t a un valor muy elevado. A continuación, se iteran los elementos de las filas indicadas por los índices $\max\_rowk$ y $\min\_rowk$. Cada elemento de la fila de índice $\max\_rowk$ es comparado con todos los elementos de la fila de índice $\min\_rowk$ según la ecuación (*5.2*). Si la comparación es satisfactoria, se realiza el cálculo de D según la ecuación (*5.3*). Si el valor que se obtiene de D es menor al valor actual de t, se iguala el valor de t a D y se guarda el valor de la posición del elemento de la fila de índice $\max\_rowk$ en la variable pk y la posición del elemento de la fila de índice $\min\_rowk$ en la variable qk. El código fuente de la función *PairForSwapping* se presenta a continuación.

Código fuente 2. Función PairForSwapping

```
function [pk, qk] = PairForSwapping(G, max_rowk, min_rowk, maxZ,
set_value)%#codegen
%% Parameters
ROWS = 3;
COLUMNS = 3;

%% Initialize
%Set t as "infinite"
t = 9999;
pk = 0;
```

Código fuente 2 (cont.). Función PairForSwapping

```
qk = 0;

%% Loop
for i = 1:COLUMNS
    for j = 1:COLUMNS
        if G(max_rowk, i) > G(min_rowk, j)
            D = abs(G(max_rowk, i) - G(min_rowk, j) - (maxZ -
set_value));
            if D < t
                t = D;
                pk = i;
                qk = j;
            end
        end
    end
end
end
```

5.3.3. Función SwapPair

La Función *SwapPair* intercambia los elementos designados por la función *PairForSwapping*. Por lo tanto, a esta función hay que indicarle los índices de fila $\max\_rowk$ y $\min\_rowk$, así como los valores de pk y qk calculados mediante la función *PairForSwapping*. Es importante recalcar que los cambios de posición que se realicen en la matriz G se deben realizar de forma análoga en la matriz de posiciones P. El código fuente de la función *SwapPair* se lista a continuación.

Código fuente 3. Función SwapPair

```
function [Gout,Pout] = SwapPair(G,P, max_rowk, min_rowk, pk, qk)%#codegen
%Swap the chosen pair
%G
aux1 = G(max_rowk, pk);
G(max_rowk, pk) = G(min_rowk, qk);
G(min_rowk, qk) = aux1;
%P
aux2 = P(max_rowk, pk);
P(max_rowk, pk) = P(min_rowk, qk);
P(min_rowk, qk) = aux2;

%Return
Gout = G;
Pout = P;
end
```

5.4. Mejoras propuestas para una implementación HW/SW

Dado que se pretende implementar el algoritmo MMTES sobre tecnología SoC FPGA, se analizan las prestaciones de la implementación realizada en MATLAB para escoger una

partición *hardware/software* adecuado. Para ello se utilizan los 4 casos de estudio propuestos en [56] que se resumen en la Tabla 3.

Tabla 3. Casos de estudio del algoritmo MMTES.

	$\boldsymbol{G}\ (\frac{\boldsymbol{W}}{\boldsymbol{m^2}})$	$\boldsymbol{P}$
Caso 1	$\begin{bmatrix}400 & 500 & 600\\ 500 & 600 & 700\\ 600 & 700 & 800\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{bmatrix}$
Caso 2	$\begin{bmatrix}700 & 700 & 700\\ 700 & 700 & 600\\ 600 & 500 & 300\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{bmatrix}$
Caso 3	$\begin{bmatrix}700 & 700 & 900 & 900\\ 600 & 600 & 800 & 800\\ 500 & 500 & 700 & 700\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13 & 14\\ 21 & 22 & 23 & 24\\ 31 & 32 & 33 & 34\end{bmatrix}$
Caso 4	$\begin{bmatrix}900 & 900 & 900\\ 600 & 600 & 600\\ 250 & 200 & 300\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{bmatrix}$

Los resultados obtenidos con respecto al tiempo de ejecución de la implementación del algoritmo MMTES en MATLAB, para los 4 casos de estudio planteados, se reflejan en la Tabla 4. En esta tabla se puede observar el tiempo de ejecución en milisegundos del algoritmo MMTES y de las funciones previamente explicadas *MMTESSort*, *PairForSwapping* y *SwapPair*. Además, también se da el dato de *self time*, el cual refleja el tiempo transcurrido dentro de la función *MMTES* sin estar ejecutando ninguna de las 3 funciones anteriormente indicadas.

Tabla 4. Tiempo de ejecución del algoritmo MMTES en MATLAB.

	MMTES (ms)	**MMTESSort (ms)**	**PairForSwapping (ms)**	**SwapPair (ms)**	**self time (ms)**
Caso 1	26	7	6	1	12
Caso 2	37	6	5	4	22
Caso 3	33	11	7	5	10
Caso 4	36	9	5	1	20

En la Figura 18 se presenta el análisis temporal del algoritmo MMTES para cada uno de los 4 casos. Se muestra el porcentaje del tiempo total de ejecución requerido por cada una de las funciones desarrolladas. La función *MMTESSort* es la que requiere de un mayor tiempo de ejecución en todos los casos, llegando en el caso 3 a un 33.33% del tiempo total

de ejecución del algoritmo. En segundo lugar, se encuentra la función *PairForSwapping* llegando a consumir un 23.08% del tiempo total en el caso 1. La función *SwapPair* es la que menos tiempo consume llegando a un máximo del 15.15% del tiempo total en el caso 3, pero con un tiempo de ejecución mucho menor en el resto de los casos.

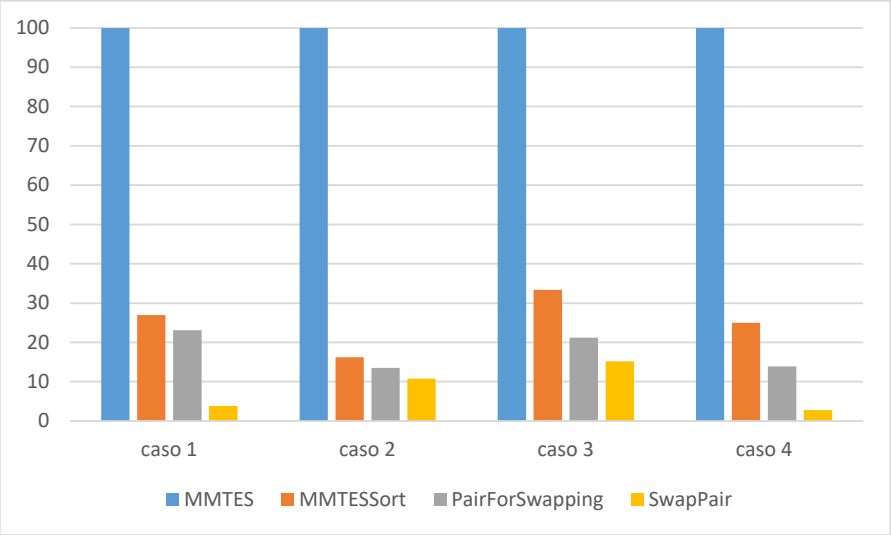


Figura 18. Representación en porcentajes del tiempo de ejecución del algoritmo MMTES.

A partir de estos datos, se procede a realizar una implementación *hardware* para las funciones *MMTESSort* y *PairForSwapping* y una implementación *software* para la función *SwapPair* y el resto del algoritmo MMTES. Se decide realizar esta partición *hardware/software* ya que las funciones *MMTESSort* y *PairForSwapping* son las que en porcentaje consumen una mayor cantidad de tiempo. En conjunto, estas dos funciones, representan entre el 40 y 50% del tiempo de ejecución del algoritmo en la mayoría de los casos. Realizar una implementación *hardware* puede mejorar en gran medida el tiempo total de ejecución del algoritmo.

Para crear una implementación hardware de la función *MMTESSort*, se ha decidido implementar el algoritmo Bitonic Sort, algoritmo de ordenación orientado a la implementación *hardware.* Para la implementación *hardware* de la función *PairForSwapping*, se ha modificado la función original de tal manera que se pueda ejecutar de forma paralela.

5.4.1. Bitonic Sort

Bitonic Sort implementa una red de *sorting* que se utiliza frecuentemente en casos en los que se necesita una velocidad de clasificación alta. Este es un algoritmo paralelo que realiza comparaciones entre los elementos a ordenar según una secuencia predefinida. Este algoritmo de *sorting* solo se puede aplicar si el número de elementos a ordenar es potencia de 2.

El funcionamiento de este algoritmo se basa en las secuencias bitónicas. Una secuencia bitónica es aquella que primero se incrementa y luego se decrementa. Por ejemplo, un *array* de n valores es una secuencia bitónica si existe un índice hasta el cual todos los valores se irán incrementando y a partir del cual todos los valores se irán decrementando (*5.6*).

También hay que tener en cuenta, que una secuencia ordenada de forma ascendente se considera una secuencia bitónica con la parte descendente vacía y una secuencia ordenada de forma descendente se considera una secuencia bitónica con la parte ascendente vacía [77].

$$\begin{gathered} x_0 \leq x_1 \leq \cdots \leq x_i \\ x_i \geq x_i + 1 \geq \cdots \geq x_n - 1 \\ 0 \leq i \leq n - 1 \end{gathered} \tag{5.6}$$

Para poder aplicar el algoritmo, el primer paso es transformar el *array* de datos de entrada en una secuencia bitónica. Para esto se empiezan formando secuencias bitónicas de 4 elementos, estando los 2 primeros elementos el orden creciente y los 2 últimos en orden decreciente.

Tras esto, se concatenan 2 secuencias bitónicas de 4 elementos y se ordena la primera de ellas en orden creciente y la segunda en orden decreciente. Se continúa de esta manera hasta que la primera mitad del *array* de datos de entrada esté ordenado de forma creciente y la segunda mitad de forma decreciente.

Una vez se ha obtenido una secuencia bitónica, se compara el primer elemento de la primera mitad con el primer elemento de la segunda mitad. Los elementos se intercambian

si el elemento de la segunda mitad es más pequeño que el de la primera mitad. La misma comparación se realiza entre el segundo elemento de la primera mitad y el segundo elemento de la segunda mitad. Se sigue de esta manera hasta haber comparado todos los elementos.

Tras realizar todas las comparaciones se obtienen 2 secuencias bitónicas en el mismo *array.* Se realizan nuevamente las mismas comparaciones que en el paso anterior para cada una de las 2 secuencias bitónicas existentes en el *array*. Este paso se repetirá hasta que el tamaño de las secuencias bitónicas sea de 1. En este punto ya se tendrá el *array* de datos ordenado [77].

En la Figura 19 se muestra un ejemplo del funcionamiento del algoritmo para un *array* de entrada de 16 elementos. Los bloques de color azul indican una ordenación creciente y los bloques verdes una ordenación decreciente.

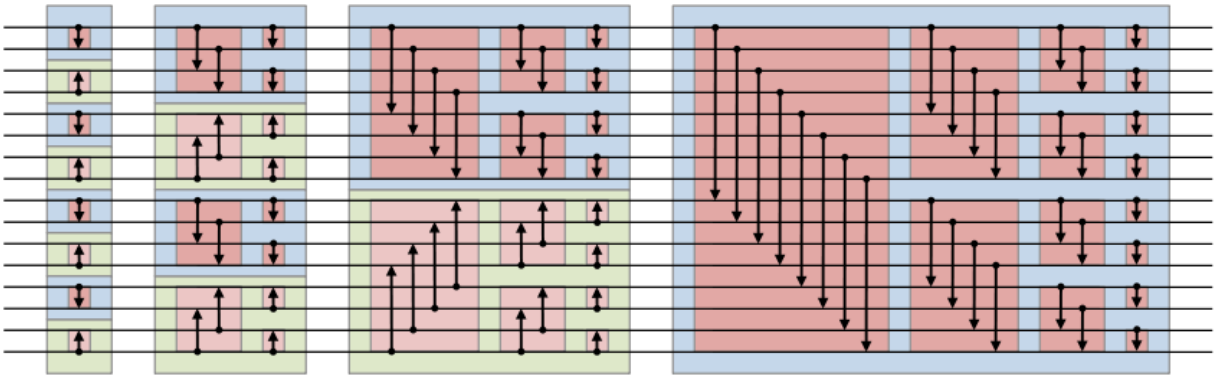


Figura 19. Bitonic Sorting Network para un array de entrada de 16 elementos [78].

En este caso, para poder implementar este algoritmo de *sorting,* primero habrá que adaptar los datos de entrada ya que la cantidad de datos no es potencia de 2 en ninguno de los casos presentados en la Tabla 3. Para ello se crean dos funciones auxiliares: una para adaptar los datos de entrada y generar un *array* cuya longitud sea potencia de 2 y otra para adaptar los datos de salida y devolver el *array* con la longitud original (Figura 20).

Figura 20.Esquemático del Bitonic Sort para el caso 1, 2 y 4.

5.4.1.1. AdaptInputData

La función *AdaptInputData* es la encargada de generar un *array* de entrada para el Bitonic Sort con una longitud que sea potencia de 2. Además de esto, también se encarga de empaquetar los datos de entrada de tal manera que se optimice su transferencia al *hardware* donde estará implementado el Bitonic Sort.

En los casos 1, 2 y 4 de la Tabla 3, el número de datos a ordenar es de 9. Dado que 9 no es potencia de 2, hay que adaptar la longitud del *array* de entrada a la potencia de 2 más cercana. En este caso, este valor es 16. Por lo tanto, a partir de un *array* de 9 valores hay que generar uno de 16 valores.

Dado que la ordenación que se va a llevar a cabo por el Bitonic Sort es en orden creciente, se concatena al *array* de entrada de 9 valores un *array* de 7 valores iguales muy superiores al mayor valor del *array* de entrada. De esta manera, una vez se haya realizado la ordenación, los valores introducidos por esta función quedarán en las 7 últimas posiciones y será fácil extraer los valores de entrada originales ordenados.

Tras adaptar la longitud del *array* de entrada, el siguiente paso es el empaquetamiento de los datos para optimizar su transferencia a *hardware*. El total de datos que se tienen que transmitir al Bitonic Sort son 32: 16 datos corresponden a la matriz de irradiancias G y los otros 16 a la matriz de posición P. Si se asume la constante solar de $1361\ W/m^2$ [79] como máximo valor teórico de los valores de irradiancia de la matriz G, se puede determinar que serán necesarios 11 bits para guardar cada uno de esos valores.

En cuanto a los valores de la matriz P, su valor varía entre 11 y 33. Esto haría necesario disponer de 6 bits para cada valor. Sin embargo, si a la hora de generar el *array* de entrada de longitud 16 para el Bitonic Sort, se le resta 10 a todos los valores de la matriz P, los valores variarían entre 1 y 23. Esto permite guardar cada uno de los valores con 5 bits. De esta manera se pueden empaquetar 2 valores de irradiancia junto con sus 2 valores correspondientes de posición en 32 bits (Figura 21).

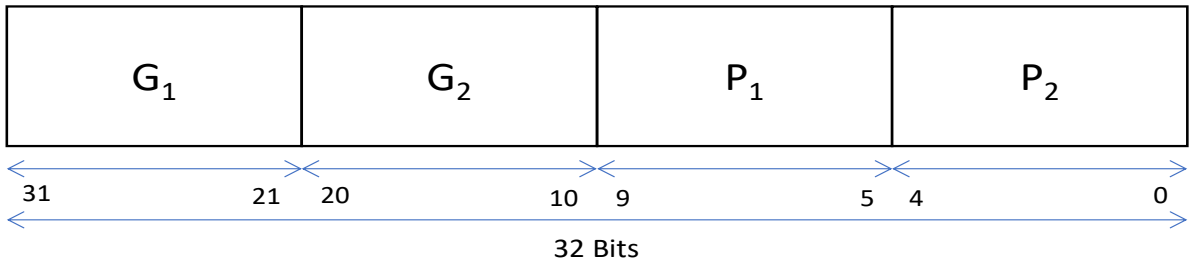


Figura 21. Paquete de datos a transmitir al Bitonic Sort.

Al empaquetar los datos de esta manera, se consigue reducir el número de transmisiones de 32 a 8, optimizando así el *Data Movement*.

La implementación de la función *AdaptInputData* se ha realizado en Simulink debido a la facilidad que ofrece para trabajar con los datos a nivel de bits.

5.4.1.2. BitonicSort

El algoritmo de ordenación Bitonic Sort se ha implementado en Simulink debido a la facilidad que ofrece esta herramienta para diseñar arquitecturas paralelas destinadas a una implementación *hardware*. El diseño de *BitonicSort* (Figura 22) se basa en la red mostrada en la Figura 19.

El código de colores se mantiene igual que el de la Figura 19. Los bloques de color azul ordenan en orden creciente, mientras que los bloques de color verde lo hacen en orden decreciente. La estructura de uno de los bloques azules se muestra en la Figura 23.

Este bloque tiene 4 entradas. Las entradas 1 y 2 son valores de irradiancia y las entradas 3 y 4 los valores de posición correspondientes a esos valores de irradiancia. Los valores de irradiancia se comparan y en caso de que la comparación sea positiva, intercambian posición. El mismo cambio de posición es aplicado también a los respectivos valores de posición. Los bloques verdes presentan la misma estructura, pero con una comparación diferente.

A la entrada de la *sorting network* el primer paso es desempaquetar los datos para poder comenzar el proceso de *sorting*. Una vez que este ha terminado, los datos se vuelven a empaquetar según se muestra en la Figura 21 y se transmiten de vuelta a la función principal del algoritmo MMTES implementado en *software*.

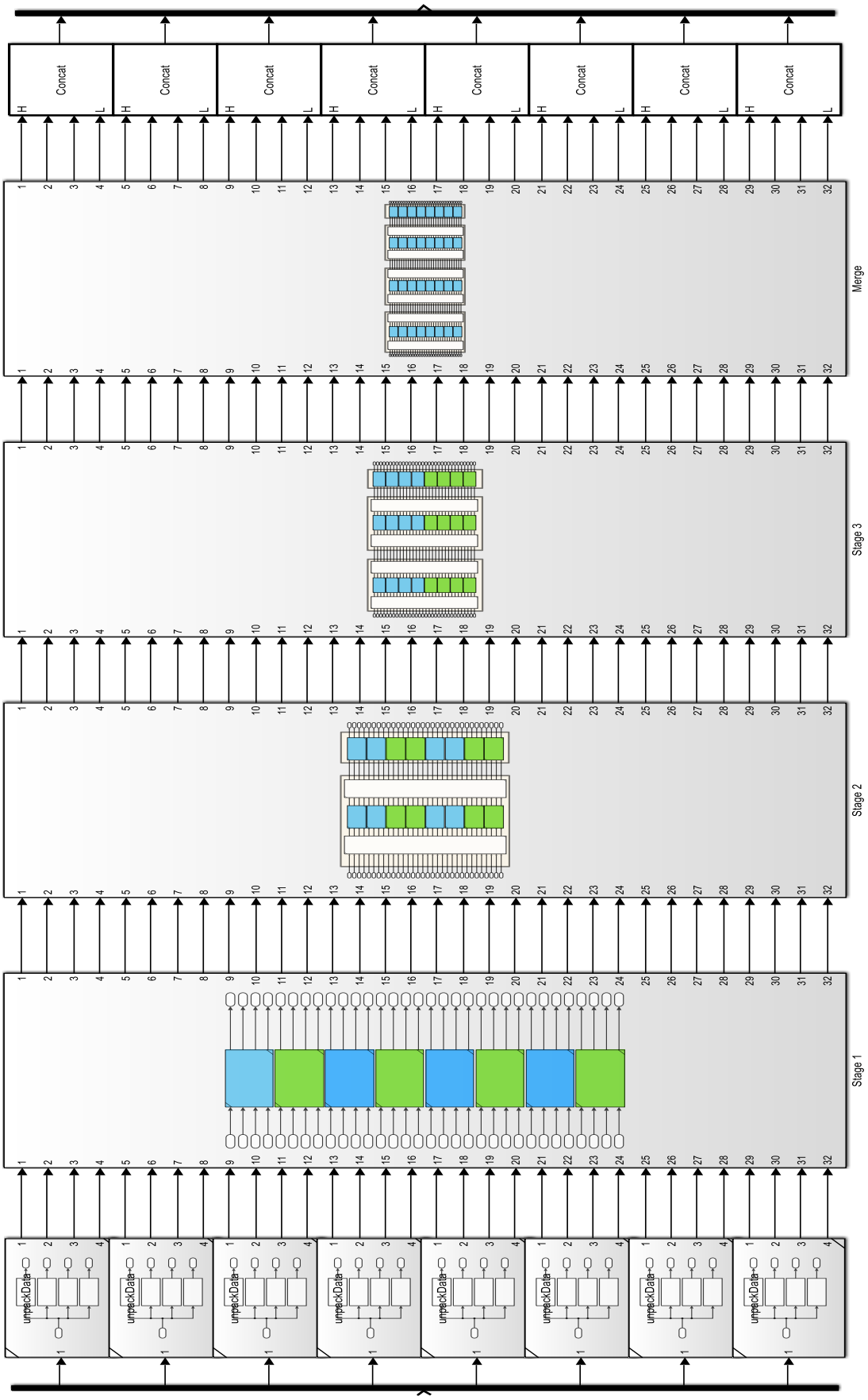


Figura 22. BitonicSort.

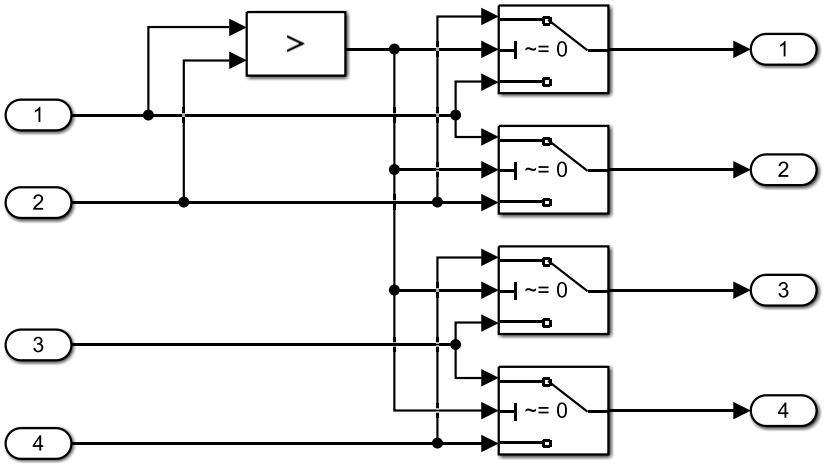


Figura 23.Bloque de ordenamiento creciente.

5.4.1.3. AdaptOutputData

La función *AdaptOutputData* es la encargada de desempaquetar los datos transmitidos por el *BitonicSort* y reconstruir la matriz de irradiancias G y la matriz de posición P. Para ello, tras desempaquetar los datos se generan los *arrays* de longitud 16 correspondientes a G y P. El *array* G habrá sido ordenado de mayor a menor por el *BitonicSort* y el *array* P habrá sufrido los mismos cambios de posición.

Una vez se tienen los 2 *arrays* de longitud 16, se extraen los 9 primeros valores de ambos *arrays* y los 7 restantes se desechan. Con estos 9 valores se genera de nuevo la matriz G y P. Para recuperar los datos originales, a todos los valores de la matriz P hay que sumarles 10, porque en el *AdaptInputData* se les restó esta cantidad para disminuir el número de bits a transmitir al *BitonicSort*.

Al igual que la función *AdaptInputData* y la función *BitonicSort*, la implementación de *AdaptOutputData* se ha realizado en Simulink debido a la facilidad que ofrece para trabajar con los datos a nivel de bits.

5.4.2. PairForSwapping paralelo

Al igual que en el caso del *BitonicSort*, para implementar la función *PairForSwappingPll* también se necesitan 2 funciones auxiliares: *PfsPllInData* y *PfsPllOutData*. Estas funciones se encargan respectivamente de empaquetar los datos a transmitir a la implementación *hardware* de la función *PairForSwappingPll* y desempaquetar los datos transmitidos por esta función.

5.4.2.1. PfsPllInData

La función *PairForSwappingPll* tiene un total de 4 parámetros de entrada. Estos parámetros son $maxRow$, $minRow$, $maxZ$ y $setValue$. En los casos 1, 2 y 4 de la Tabla 3, los parámetros $maxRow$ y $minRow$ son *arrays* de 3 elementos que contienen las filas de índice $\max\_rowk$ y $\min\_rowk$ de la matriz G respectivamente.

Asumiendo nuevamente la constante solar de $1361\ W/m^2$ como valor máximo teórico para los elementos de la matriz G, serán necesarios un total de 33 bits, 11 bits por valor, para guardar los 3 valores del *array* $maxRow$ y otros 33 bits para el *array* $minRow$.

Por otro lado, en los casos 1, 2 y 4 de la Tabla 3, el valor máximo teórico de Z, asumiendo la constante solar como valor de irradiancia máximo posible, es de $3 * 1361 = 4083$. Por lo tanto, serán necesario 12 bits para representar el valor de $maxZ$. Finalmente, el valor máximo de $setValue$, teniendo como máximo de irradiancia teórico la constante solar, se puede calcular a partir de (*5.1*) y es igual al valor de la constante solar. Por lo tanto, para representar $setValue$ serán necesarios un total de 11 bits.

Los datos se empaquetarán de 2 en 2, generando un total de 4 paquetes. Los 3 primeros paquetes serán de 22 bits y el último de 23 bits. A la hora de realizar la transmisión a la implementación *hardware* de la función *PairForSwappingPll*, cada paquete de datos será transmitido como un valor de 32 bits. Por lo tanto, a la salida de la función *PfsPllInData* habrá un *array* de 4 elementos, cada uno de ellos de 32 bits (Figura 24).

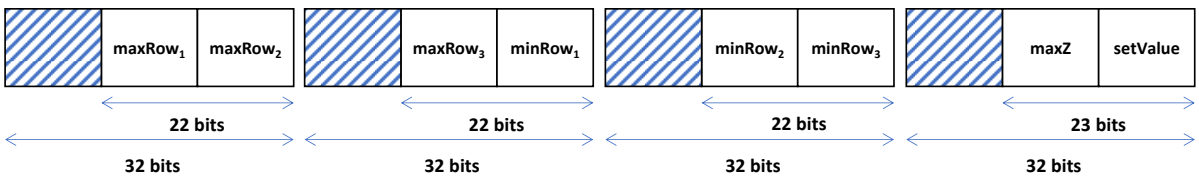


Figura 24. Array de salida de la función PfsPllInData.

Al igual que las funciones auxiliares para el *BitonicSort*, la función *PfsPllInData* ha sido implementada en Simulink debido a la facilidad que ofrece esta herramienta para trabajar a nivel de bits.

5.4.2.2. PairForSwappingPll

La función *PairForSwappingPll* se ha implementado en Simulink por los mismos motivos que el *BitonicSort*. Esta función es una adaptación para un funcionamiento paralelo de la Función *PairForSwapping*. El dato de entrada de la función *PairForSwappingPll* es el *array* de salida generado por la función *PfsPllInData*, por lo tanto, el primer paso a realizar en esta función es desempaquetar los datos de entrada. Una vez se han desempaquetado los datos, comienza la implementación paralela de la Función *PairForSwapping*.

Primero se generan 2 *arrays* auxiliares de 9 elementos. El primer *array* está formado por la concatenación de 3 copias del *array* $maxRow$ de entrada. El segundo *array* está formado por la concatenación de 3 copias del *array* $minRow$ de entrada, pero los elementos de cada copia están desplazados una posición hacia la derecha con respecto a la copia que se encuentre a su izquierda. Un ejemplo de ambos *arrays* auxiliares se ilustra en la Figura 25.

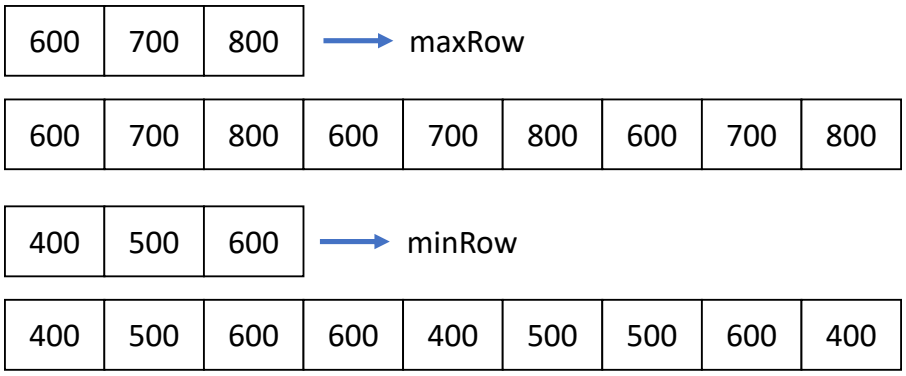


Figura 25. Arrays auxiliares en la función PairForSwappingPll.

Siguiendo este mismo método se crean otros 2 *arrays* de 9 elementos denominados $posMax$ y $posMin$. Estos *arrays* guardan la posición de los valores de irradiancia con respecto a los *arrays* de entrada $maxRow$ y $minRow$ (Figura 26).

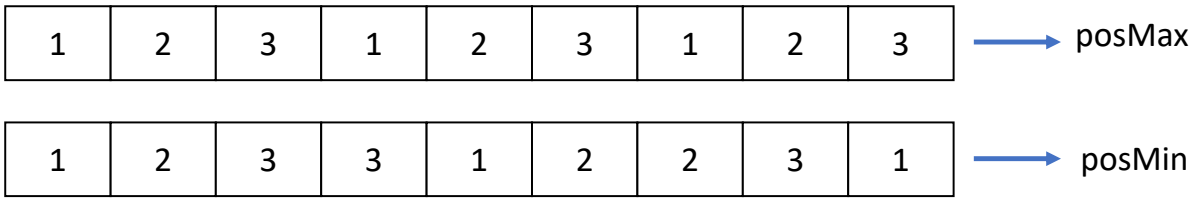


Figura 26. Arrays posMax y posMin.

Gracias a estos *arrays* auxiliares, se puede aplicar la ecuación (*5.3*) de forma paralela para todos los posibles pares de valores que se pueden generar a partir de los *arrays* de

entrada $maxRow$ y $minRow$. Los resultados de aplicar esta ecuación se guardan en un nuevo *array*.

El siguiente paso es identificar la posición del menor elemento en el *array* de resultados. Una vez se tiene la posición de este elemento, se extrae del *array* $posMax$ y $posMin$ los valores que coincidan con esa posición. El valor extraído del *array* $posMax$ es el índice pk y el extraído de $posMin$ es qk. Ambos son valores que necesitan de 2 bits para poder ser representados.

Una vez se tienen pk y qk, se empaquetan según se muestra en la Figura 27 y se transmiten de vuelta a la función principal del algoritmo MMTES implementada en *software*.

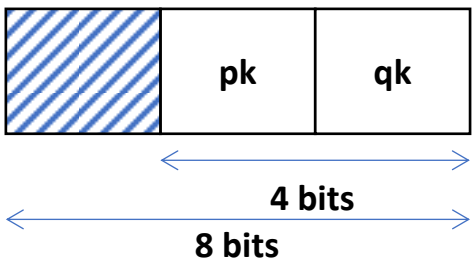


Figura 27. Empaquetado de los datos de salida de PairForSwappingPll.

5.4.2.3. PfsPllOutData

Esta función también está implementada en Simulink y es la encargada de desempaquetar los datos de salida transmitidos por la función *PairForSwappingPll*. El diseño que se implementa en Simulink para extraer los datos transmitidos se ilustra en la Figura 28.

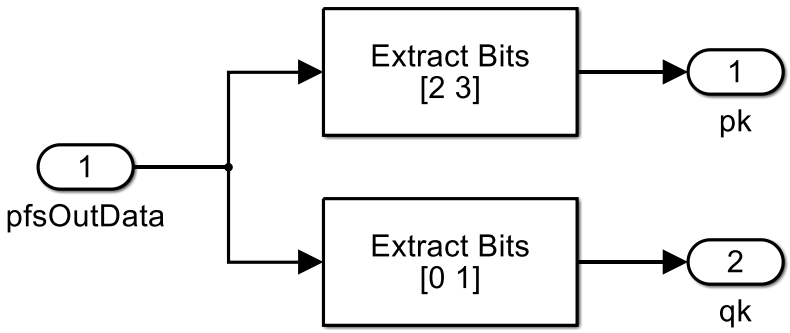


Figura 28. Función PfsPllOutData.

5.5. Conclusiones

En este capítulo se ha presentado el algoritmo DPVAR MMTES. En primer lugar, se han introducido las características principales del algoritmo, así como su funcionamiento. Se ha presentado una primera implementación del algoritmo en MATLAB que ha permitido hacer un estudio sobre el tiempo de ejecución del algoritmo para los 4 casos de implementación propuestos en el *paper* del algoritmo MMTES. En base a los datos obtenidos de estas pruebas, se ha propuesta una partición *hardware/software* del algoritmo para adaptarlo a la tecnología SoC FPGA e intentar mejorar los datos de tiempo de ejecución obtenidos en las pruebas realizadas en MATLAB. Las partes del algoritmo que se han decidido implementar en *hardware* son el algoritmo de *sorting* y la función *PairForSwapping*. Para la implementación *hardware* del algoritmo de ordenación se ha escogido el algoritmo Bitonic Sort. Este es un algoritmo *hardware friendly* que permite explotar el paralelismo que ofrece utilizar FPGA. Para la función *PairForSwapping* se ha desarrollado una modificación de esta que permite su funcionamiento en paralelo. Tanto el *BitonicSort* como la función *PairForSwappingPll* han sido desarrolladas en Simulink por la facilidad que ofrece esta herramienta para diseñar arquitecturas paralelas con el fin de ser implementadas en *hardware*.

Capítulo 6. Controlador MPPT

6.1. Introducción

En este capítulo se presenta el funcionamiento de los controladores MPPT (*Maximum Power Point Tracker*), así como un total de tres variantes diferentes, incluida una orientada a su implementación sobre tecnología SoC FPGA. El funcionamiento de la variante de controlador escogida se explica en profundidad y se expone como se realiza su implementación en MATLAB para su posterior implementación en *hardware*.

6.2. Controlador MPPT

Los controladores MPPT implementan un algoritmo denominado de seguimiento del máximo punto de potencia. Estos son controladores que se suelen implementar en sistemas de generación de energía eléctrica basados en fuentes de potencia variable como, por ejemplo, la energía solar fotovoltaica o la energía eólica.

La implementación de este tipo de algoritmos de control está muy extendida en los inversores utilizados en sistemas fotovoltaicos. Los controladores MPPT ajustan de manera continua la impedancia observada por la instalación fotovoltaica con el fin de hacerla trabajar en el punto de máxima potencia (MPP, de sus siglas en inglés), o lo más cerca a este posible [80]. El objetivo de estos controladores es maximizar la potencia producida independientemente de condiciones cambiantes como pueden ser la temperatura, la irradiancia solar o la carga.

El MPP se puede observar en la curva P-V de una instalación fotovoltaica (Figura 29). Los algoritmos MPPT controlan la tensión de funcionamiento para así poder trabajar los más cerca posible del MPP. Para conseguir esto, existen diferentes variantes de controladores MPPT. Las 3 variantes más comúnmente utilizadas se presentan en los siguientes apartados.

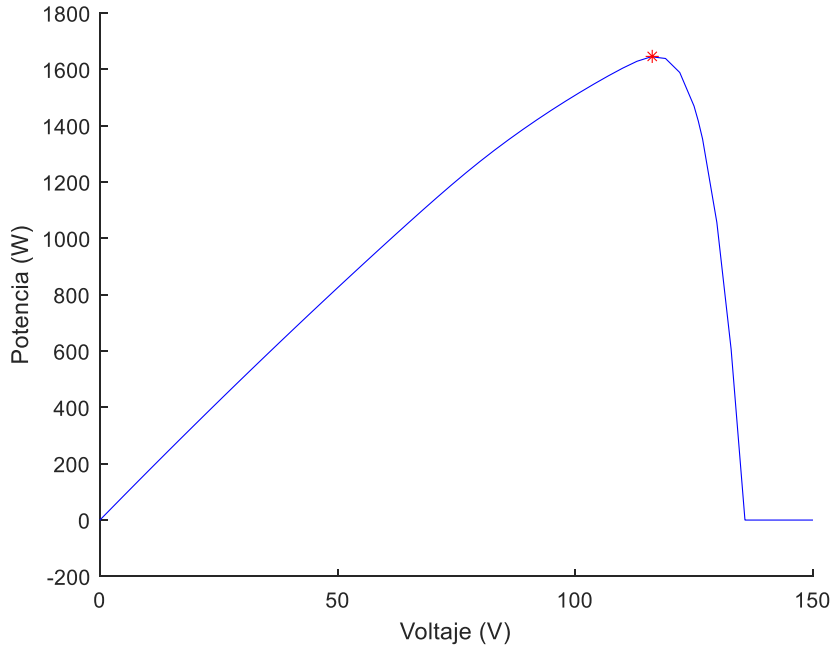


Figura 29. Diagrama P-V de una instalación fotovoltaica.

6.2.1. Perturbación y Observación (P&O)

Esta es una variante de MPPT basada, como su nombre indica, en modificar el voltaje de funcionamiento y observar la potencia [81]. Tras hacer una modificación en el voltaje, se mide la potencia y se compara con una medición anterior. Si la potencia actual es mayor que la medida con anterioridad se sigue modificando el voltaje en esa dirección. Por el contrario, si la potencia es menor que la anterior, se modifica el voltaje en la dirección contraria.

El MPPT P&O es la variante más común debido a la sencillez de su implementación. Sin embargo, este método de control puede causar oscilaciones en la potencia de salida debido a las continuas perturbaciones en el voltaje de funcionamiento.

6.2.2. Conductancia incremental

El método de conductancia incremental está basado en comparar la conductancia incremental $\frac{dI}{dV}$ con la conductancia instantánea $\frac{I}{V}$ y modificar el ciclo de trabajo de salida en función del resultado obtenido. Cuando el sistema fotovoltaico está trabajando en el MPP, se cumple la ecuación (6.1). Sin embargo, cuando se cumple la ecuación (6.2) o (6.3) significa que el sistema fotovoltaico está trabajando a la izquierda o derecha del MPP respectivamente.

$$\frac{dI}{dV} = -\frac{I}{V} \tag{6.1}$$

$$\frac{dI}{dV} > -\frac{I}{V} \tag{6.2}$$

$$\frac{dI}{dV} < -\frac{I}{V} \tag{6.3}$$

Si el sistema fotovoltaico está trabajando en el MPP, el ciclo de trabajo de salida permanece constante. Sin embargo, cuando el punto de trabajo está a la izquierda o derecha del MPP, se incrementa o decrementa el valor del ciclo de trabajo respectivamente [82].

En comparación con la variante P&O, la conductancia incremental requiere un mayor tiempo de cómputo por parte del controlador, pero es capaz de seguir con mayor rapidez cambios en el MPP y no produce oscilaciones en la potencia de salida.

6.2.3. Tensión a circuito abierto fraccional

Esta variante de controlador MPPT [83] está basada en la relación cuasi lineal existente entre la tensión a circuito abierto y la tensión del MPP (*6.4*). La constante K es el denominado factor de voltaje, cuyo valor se encuentra en el *datasheet* de los paneles fotovoltaicos y suele oscilar entre 0.7 y 0.9.

$$V_{MPP} = K * V_{OC} \tag{6.4}$$

Para medir la tensión a circuito abierto, se desconecta periódicamente la carga, lo que ocasiona pérdidas en la potencia generada.

6.3. Implementación del controlador

La variante de controlador MPPT que se escoge para implementar sobre tecnología SoC FPGA es la de conductancia incremental. Se ha decidido escoger este controlador, porque frente a la variante P&O, si bien requiere de un mayor tiempo de cómputo, consigue una mejor estabilidad de la potencia a la salida.

En comparación con la variante de tensión a circuito abierto fraccional, el MPPT de conductancia incremental es más general ya que no depende de las características de los paneles fotovoltaicos presentes en la instalación a controlar.

La implementación del controlador MPPT de conductancia incremental se realiza en MATLAB en base al diagrama de flujo que se ilustra en la Figura 30.

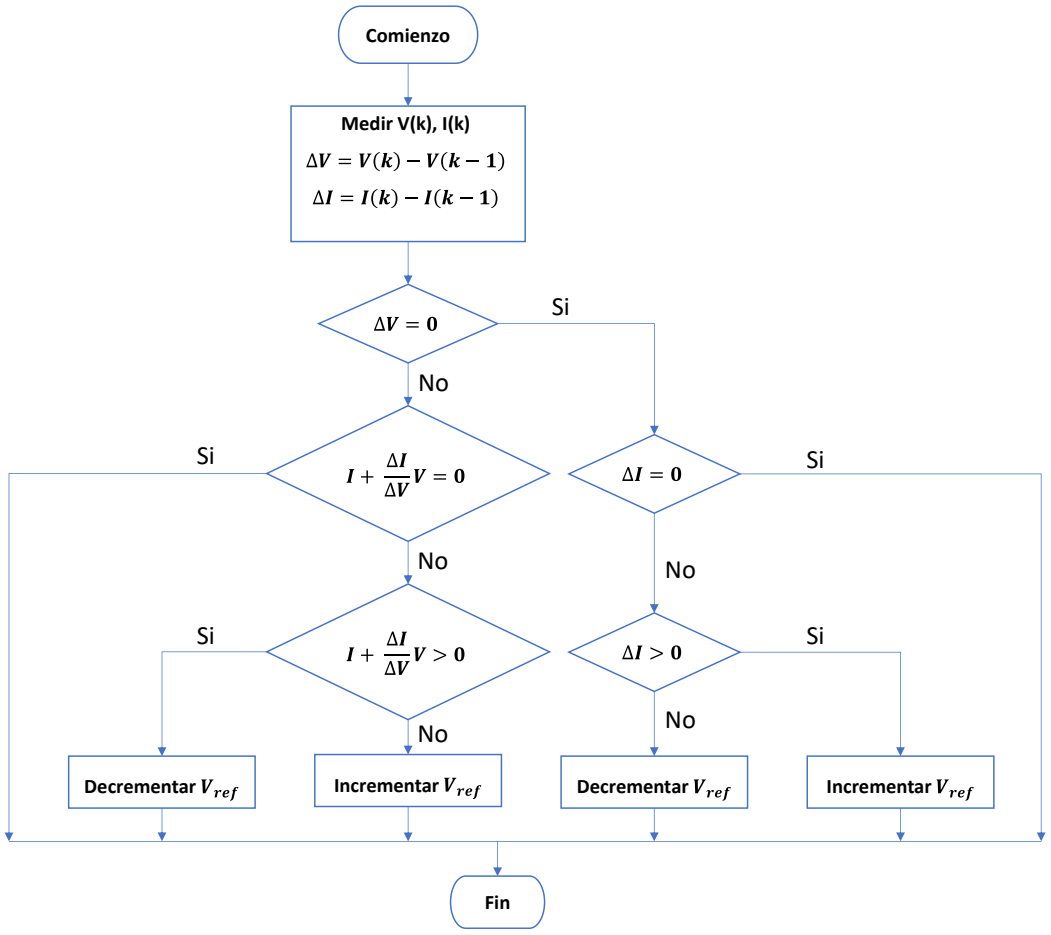


Figura 30. Diagrama de flujo del algoritmo MPPT de conductancia incremental [82].

La función *MPPT_Inc* se inicia con la declaración de las variables $prev\_Vpv$, $prev\_Ipv$ y $prev\_D$ como estáticas. Estas variables corresponden a los valores previos de tensión,

corriente y ciclo de trabajo respectivamente. El siguiente paso es inicializar el valor de $delta\_D$ a 0.0001 y calcular el valor de $delta\_Vpv$ y $delta\_Ipv$ según (6.5).

$$delta\_Vpv = Vpv - prev\_Vpv$$
$$delta\_Ipv = Ipv - prev\_Ipv \tag{6.5}$$

Tras esto, se determina el valor de D siguiendo las diferentes condiciones estipuladas por el diagrama de flujo de la Figura 30. Una vez se tiene el valor de D, se comprueba que esté dentro del rango permitido de valores. Para finalizar se actualizan los valores de $prev\_Vpv$, $prev\_Ipv$ y $prev\_D$ con los valores actuales. El código fuente de esta función se lista a continuación.

Código fuente 4. Función MPPT_Inc

```
function D = MPPT_Inc(Vpv,Ipv)
persistent prev_Vpv prev_Ipv prev_D
if isempty(prev_D)
    prev_D = 0.5; prev_Ipv = 0; prev_Vpv = 0;
end
delta_D = 0.0001;
delta_Vpv = Vpv - prev_Vpv;
delta_Ipv = Ipv - prev_Ipv;
if delta_Vpv == 0
    if delta_Ipv == 0
        D = prev_D;
    else
        if delta_Ipv > 0
            D = prev_D + delta_D;
        else
            D = prev_D - delta_D;
        end
    end
else
    if (Ipv + (delta_Ipv / delta_Vpv) * Vpv ) == 0
        D = prev_D;
    else
        if (Ipv + (delta_Ipv / delta_Vpv) * Vpv ) > 0
            D = prev_D - delta_D;
        else
            D = prev_D + delta_D;
        end
    end
end
if D > 1
    D = 1;
elseif D < 0
    D = 0;
end
prev_Vpv = Vpv; prev_Ipv = Ipv; prev_D = D;
end
```

Existen varios trabajos previos en los que se ha implementado esta variante de controlador MPPT. Por ejemplo, en [84] se realiza la implementación de diferentes variantes de controladores MPPT, entre las que se encuentra la de conductancia incremental, en conjunto con un sistema fotovoltaico similar al que se implementa en este trabajo. En ese caso, el controlador MPPT de conductancia incremental consigue un tiempo de respuesta de 2.97 s y una eficiencia en el seguimiento del MPP de un 97%.

A partir de este código creado en MATLAB, se desarrolla la implementación *hardware* del controlador MPPT de conductancia incremental en capítulos posteriores.

6.4. Conclusiones

En este capítulo se han presentado los algoritmos MPPT y las diferentes variantes de implementación existentes más utilizadas. La variante escogida finalmente para realizar una implementación sobre tecnología SoC FPGA ha sido el MPPT de conductancia incremental. Se ha elegido esta variante porque su rendimiento es independiente de las características de los paneles fotovoltaicos y consigue un valor de potencia de salida estable.

Se ha realizado una implementación en MATLAB del controlador MPPT a partir de la cual se realizará la implementación en *hardware* de este más adelante.

Capítulo 7. Modelado del sistema

7.1. Introducción

En este capítulo se presenta el sistema a modelar en Simulink. Para ello se introducen todos los modelos necesarios que forman el sistema final. Una vez el modelado del sistema final está completo, se realiza una simulación MIL del algoritmo MMTES y el controlador MPPT en conjunto con el modelado del sistema. Mediante esta simulación se verifica el correcto funcionamiento tanto del algoritmo MMTES, como del controlador MPPT.

Tras la simulación MIL se realiza una simulación SIL del algoritmo MMTES y el controlador MPPT en conjunto con el sistema modelado. El fin de esta simulación es verificar que Simulink es capaz de generar código a partir del algoritmo MMTES y el controlador MPPT.

7.2. Sistema a modelar

El modelado del sistema se realiza en Simulink. El sistema completo consta de un total de ocho bloques diferenciados. En la Figura 31 se ilustra el sistema a modelar mediante un diagrama de bloques.

El sistema está dividido en dos partes principales: el modelado del *hardware* y los algoritmos a implementar sobre SoC FPGA (Figura 31). La parte *hardware* incluye el *array* fotovoltaico afectado por sombras parciales, la matriz de interruptores que permite alterar el conexionado del *array* fotovoltaico, un *Buck Converter* que permite al controlador MPPT modificar el punto de trabajo del sistema fotovoltaico y una carga DC.

En el SoC FPGA se implementan el algoritmo MMTES, el controlador MPPT de conductancia incremental, la función *auxCtrl* y el generador de PWM.

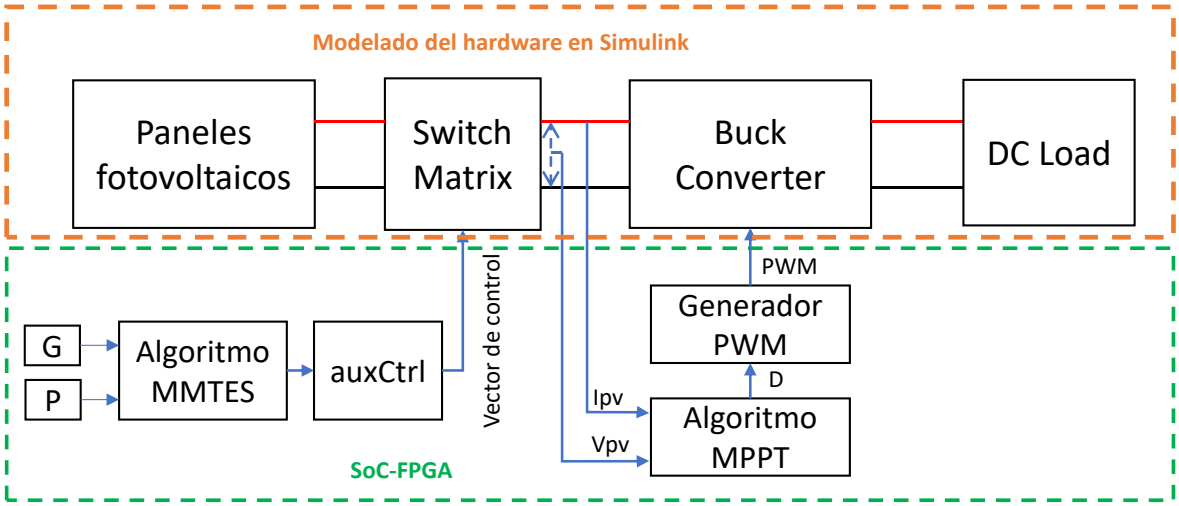


Figura 31. Diagrama de bloques del sistema a modelar.

7.3. Modelado del *hardware*

En este apartado se presenta el modelo en Simulink de los diferentes componentes que forman parte del modelado del *hardware* del sistema.

7.3.1. Paneles fotovoltaicos

Para modelar los paneles fotovoltaicos, se utiliza el bloque PV Array del *toolbox Simscape Electrical* de Simulink [85]. Este bloque implementa un circuito equivalente de un diodo (Figura 32) para modelar un panel fotovoltaico. Estos circuitos equivalentes están descritos por la ecuación (7.1) [86]. Los parámetros son la corriente I_L que depende de la irradiancia solar, la corriente de saturación inversa I_0 que depende de la temperatura del silicio, una resistencia en serie R_S, una resistencia de derivación R_{sh} y el factor de calidad del diodo n [87]. Sin embargo, dado que el bloque PV Array es capaz de modelar el conexionado de múltiples paneles fotovoltaicos a la vez, la ecuación (7.2) que implementan estos bloques es ligeramente diferente a la ecuación (*7.1*). En esta ecuación se sustituye n por (7.3) donde q es el valor de la carga de un electrón, $Ncell$ el número de células conectadas en serie en un módulo, k es la constante de Boltzman y nl el factor de idealidad del diodo.

$$I = I_L - I_0 * \left(e^{\frac{V+I*R_s}{n*T}} - 1\right) - \frac{V + I * R_s}{R_{sh}} \quad (7.1)$$

$$I = I_L - I_0 * \left(e^{\frac{q*(V+I*R_s)}{k*nl*Ncell*T}} - 1\right) - \frac{V + I * R_s}{R_{sh}} \quad (7.2)$$

$$n = \frac{k}{q} * nl * Ncell \quad (7.3)$$

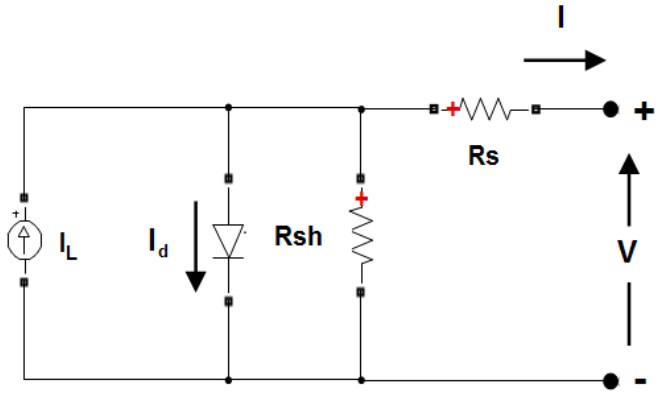
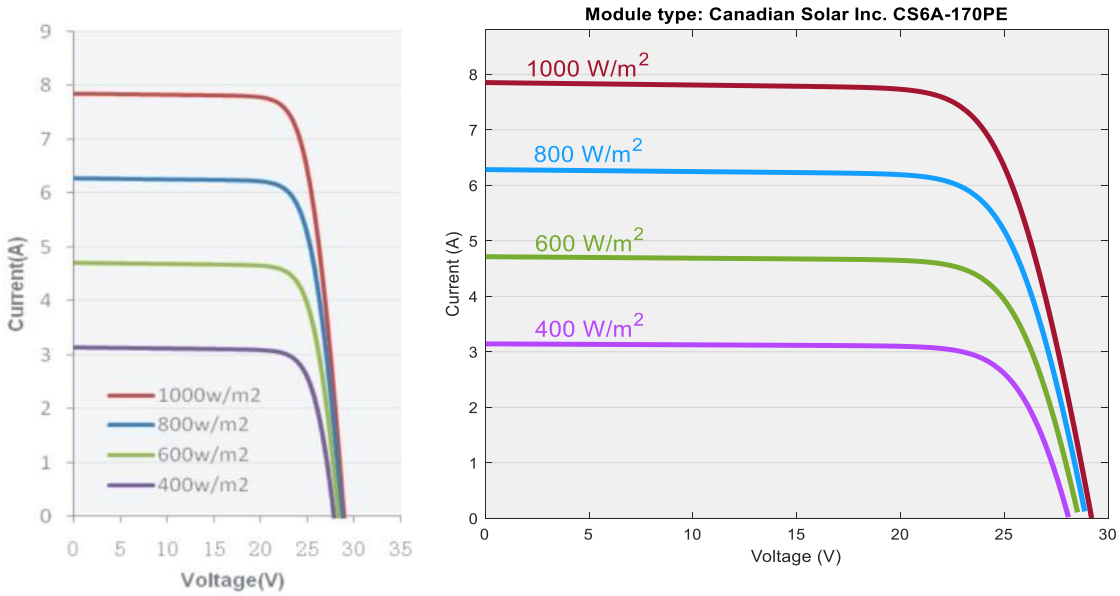


Figura 32. Modelo de módulo fotovoltaico implementado por el bloque PV Array [85].

El bloque PV Array cuenta con una amplia lista de valores precargados de diferentes módulos fotovoltaicos proporcionados por el *National Renewable Energy Laboratory* (NREL) [88] y, además, permite también definir módulos fotovoltaicos propios.

El modelo de panel fotovoltaico escogido es el CS6A-170PE de la empresa Canadian Solar Inc. [89]. Este es un panel fotovoltaico con una potencia máxima nominal de 170 W, una tensión del MPP de 23.2 V y una corriente del MPP de 7.33 A. Se ha escogido este modelo de panel fotovoltaico porque es uno de los que están precargados en el bloque PV Array y, además, el fabricante proporciona la curva I-V del mismo, lo que permite compararla con la obtenida del bloque PV Array en Simulink y así verificar el correcto funcionamiento del modelo.

En la Figura 33 se pueden comparar las curvas I-V facilitadas por el fabricante para varios valores de irradiancia a una temperatura de 25 °C, con las obtenidas en Simulink al implementar el bloque PV Array con los datos precargados de este módulo. Se puede observar que las curvas obtenidas en Simulink son idénticas a las proporcionadas por el fabricante, por lo tanto, se verifica que el bloque PV Array modela las curvas I-V del módulo fotovoltaico escogido correctamente.



Curva IV facilitada por el fabricante

Curva IV obtenida en Simulink

Figura 33. Comparativa de características I-V a temperatura constante para el modelo CS6A-170PE.

7.3.2. Switch matrix

Como se indica en el Capítulo 5, la matriz de interruptores necesaria para el algoritmo MMTES es una red de interruptores SPST que permite variar la interconexión de los diferentes paneles fotovoltaicos para conseguir igualar la irradiancia de los diferentes *tiers*. La topología de la matriz de interruptores viene definida por el tamaño del *array* fotovoltaico.

De los cuatro casos de estudio planteados en [56], tres corresponden a un *array* fotovoltaico de dimensiones (3 x 3) mientras que el caso restante corresponde a un *array* (3 x 4). Es por esto, que se ha decidido implementar en Simulink el modelado de una matriz de interruptores para los casos de estudio con un *array* fotovoltaico (3 x 3), es decir, para los casos 1, 2 y 4.

El conexionado de los interruptores para un módulo fotovoltaico en el caso de un *array* (3 x 3) se ilustra en la Figura 17. Este conexionado se modela en Simulink mediante el *subsystem* SubSwitchMatrix (Figura 34). Este *subsystem* tiene como entradas el vector de control que contiene los estados de los interruptores y el terminal positivo y negativo del bloque PV Array correspondiente. Las salidas son un total de 6 y corresponden a los terminales positivo y negativo de cada *tier* del *array* fotovoltaico. Los interruptores SPST se

modelan mediante el bloque *Ideal Switch* de Simulink tomando como referencia el interruptor comercial G5CA-1A-E-DC5 [90] con un poder de corte máximo de 10 A a 30 VDC.

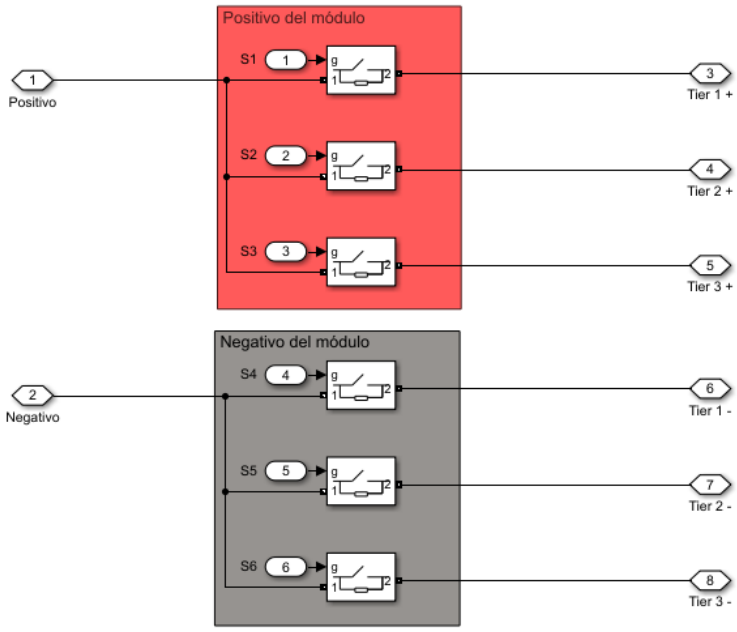


Figura 34. Subsystem SubSwitchMatrix.

Para crear el modelo final de la matriz de interruptores se ha tomado como referencia la implementación realizada en [91]. La matriz de interruptores se forma al interconectar cada uno de los SubSwitchMatrix correspondientes a cada panel fotovoltaico del *array*. Dado que el *array* fotovoltaico es de dimensiones (3 x 3), cuenta con un total de 9 paneles diferentes, cada uno de ellos con su SubSwitchMatrix correspondiente. La conexión de los 2 primeros SubSwitchMatrix se muestra en la Figura 35. El conexionado del resto se hace de manera similar.

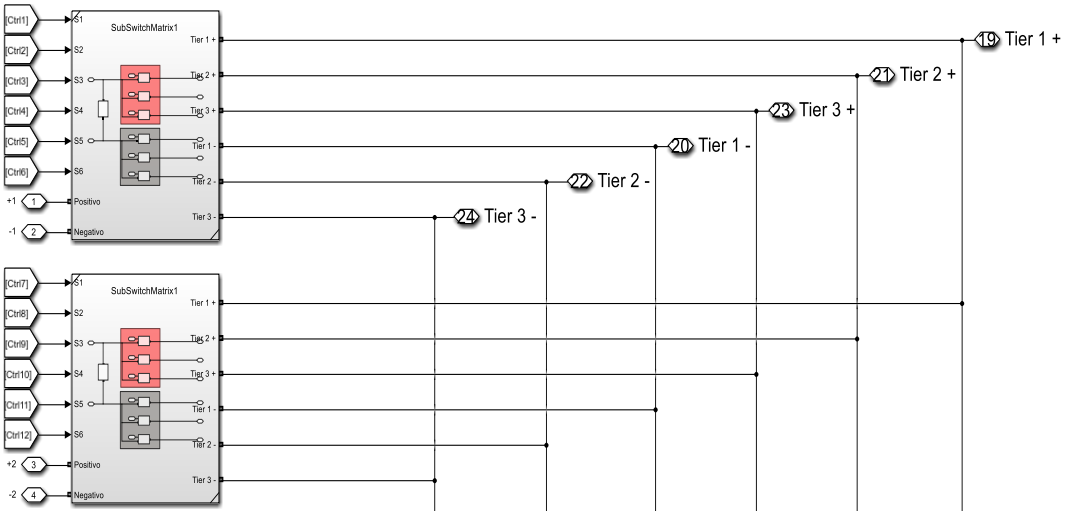


Figura 35. Ejemplo de conexionado de los SubSwitchMatrix.

La configuración del *array* fotovoltaico es siempre TCT, como la que se muestra en la Figura 15, y la matriz de interruptores permite indicar el panel fotovoltaico que está conectado a cada *tier*. La posición de los paneles, tras la reconfiguración se indica en la matriz $Pout$ a la salida del algoritmo MMTES.

Para realizar el cambio de posición de los paneles fotovoltaicos es necesario generar un vector de control a partir de la matriz $Pout$. Este vector le indica a la matriz de interruptores los estados de todos los interruptores. La función encargada de generar dicho vector de control es auxCtrl.

7.3.3. Buck Converter

El *Buck Converter* [92] es un convertidor de potencia DC-DC cuyo propósito es obtener una tensión a la salida menor a la tensión de entrada. El *Buck Converter* tiene 2 modos de funcionamiento diferentes: modo continuo (CCM) y modo discontinuo (DCM). Con el modo de funcionamiento CCM se consigue una conversión de potencia eficiente mientras que el modo DCM se utiliza para aplicaciones de menor potencia [93]. En esta aplicación, el modo de funcionamiento del *Buck Converter* es CCM.

El dimensionado de los componentes del *Buck Converter* se realiza mediante las ecuaciones presentadas en [94]. Para realizar el cálculo de los componentes hace falta el valor típico del voltaje de entrada (V_i), el voltaje de salida nominal (V_o), la máxima corriente de salida ($Iout_{max}$), la corriente máxima de rizado del inductor deseada (ΔI_L), el rizado de la tensión de salida deseado ($V_{o_{ripple}}$) y la frecuencia de conmutación (f). Para calcular V_i se parte del voltaje en el MPP del módulo fotovoltaico escogido. Este valor es de 23.2 V a 25 °C y una irradiancia de 1000 W/m^2.

Dado que en el sistema modelado hay 3 *strings* conectados en serie, cada uno de ellos formado por 3 módulos fotovoltaicos conectados en paralelo, el valor de $V_i = 23.2\,V * 3 = 69.6\,V$. El voltaje de salida del conversor V_o se establece a 48 V, porque este nivel de tensión ofrece un rendimiento óptimo en un sistema DC de baja potencia en términos de eficiencia y seguridad [95]. El valor de $Iout_{max}$ se calcula a partir de la potencia nominal de los módulos fotovoltaicos y la tensión de salida del conversor (7.4). Se escoge que el rizado de la corriente del inductor sea de un 20%, como se aconseja en [94], lo que

haría que $\Delta I_L = 6.375\ A$ (*7.5*). El rizado de la tensión de salida se escoge que sea de un 5%, por lo tanto, $V_{o_{ripple}} = 2.4\ V$ (*7.6*).

Finalmente, se elige que $f = 10\ KHz$.

$$Iout_{max} = \frac{170\ W * 9}{48\ V} = 31.875\ A \tag{7.4}$$

$$\Delta I_L = 0.2 * \ Iout_{max} = 0.2 * \ 31.875\ A = 6.375\ A \tag{7.5}$$

$$V_{o_{ripple}} = 0.05 * V_o = 0.05 * 48\ V = 2.4\ V \tag{7.6}$$

El valor mínimo del inductor L de la Figura 36 se calcula según (7.7).

$$L = \frac{V_o * (V_i - V_o)}{\Delta I_L * f * V_i} = \frac{48 * (69.6 - 48)}{6.375 * 10000 * 69.6} = 234\ \mu H \tag{7.7}$$

El valor mínimo del condensador C_{out} de la Figura 36 se calcula según (7.8).

$$C_{out} \ \geq \frac{\Delta I_L}{8 * f * V_{o_{ripple}}} \geq \frac{6.375}{8 * 10000 * 2.4} \ \geq 33.2\ \mu F \tag{7.8}$$

El valor mínimo del condensador C_{in} de la Figura 36 se calcula de manera similar al condensador C_{out} (7.9). En este caso se asume I_{min} como el 10% de $Iout_{max}$.

$$C_{in} \geq \frac{Iout_{max} + I_{min}}{8 * f * V_{in_{ripple}}} \geq \frac{31.875 + 3.19}{8 * 10000 * 69.6 * 0.05} \geq 125.96\ \mu F \tag{7.9}$$

Para el caso de los condensadores C_{in} y C_{out}, el valor final escogido es el valor estándar más cercano a los valores mínimos calculados. Es por esto por lo que finalmente el valor de C_{in} es de 127 µF y el de C_{out} de 35 µF.

Para seleccionar el interruptor S, se necesita conocer la corriente máxima de conmutación. Esta corriente se calcula mediante (7.10) y su valor es de 35.0625 A. El interruptor que se escoge y que cumple con esta especificación es el MOSFET IRF5210PbF [96].

$$I_{sw_{max}} = \ Iout_{max} + \frac{\Delta I_L}{2} = \ 31.875 + \frac{6.375}{2} = 35.0625\ A \tag{7.10}$$

En cuanto al diodo D, siguiendo la recomendación de [94], se escoge un diodo Schottky cuya I_F tenga un valor superior al calculado en (7.11). El diodo elegido es el DSSS 30-01 AR [97].

$$I_F = Iout_{max} * (1 - D) = 31.875 * \left(1 - \frac{48}{69.6}\right) = 9.89\ A \tag{7.11}$$

La implementación del *Buck Converter* en Simulink se ilustra en la Figura 36.

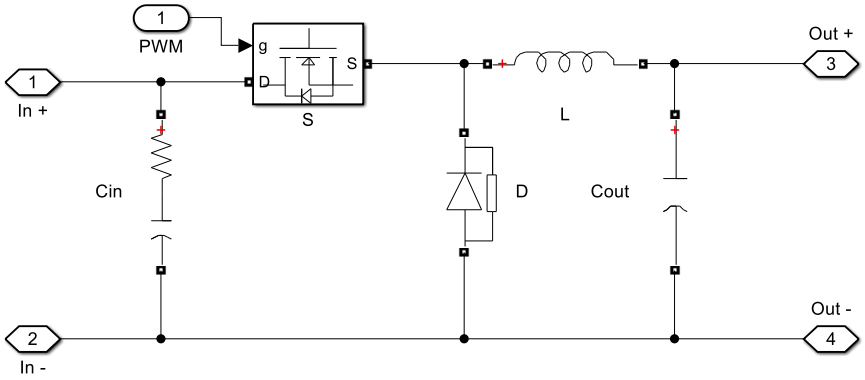


Figura 36. Implementación del Buck Converter en Simulink.

Para modelar los componentes se utilizan bloques del *toolbox* Simscape Electrical dimensionados según los valores obtenidos anteriormente.

Al simular su funcionamiento en Simulink se puede comprobar como el convertidor diseñado cumple con las especificaciones de rizado en la corriente del inductor, en torno a un 18.46 %, y con las especificaciones de rizado de la tensión de salida, en torno a un 4.23% (Figura 37).

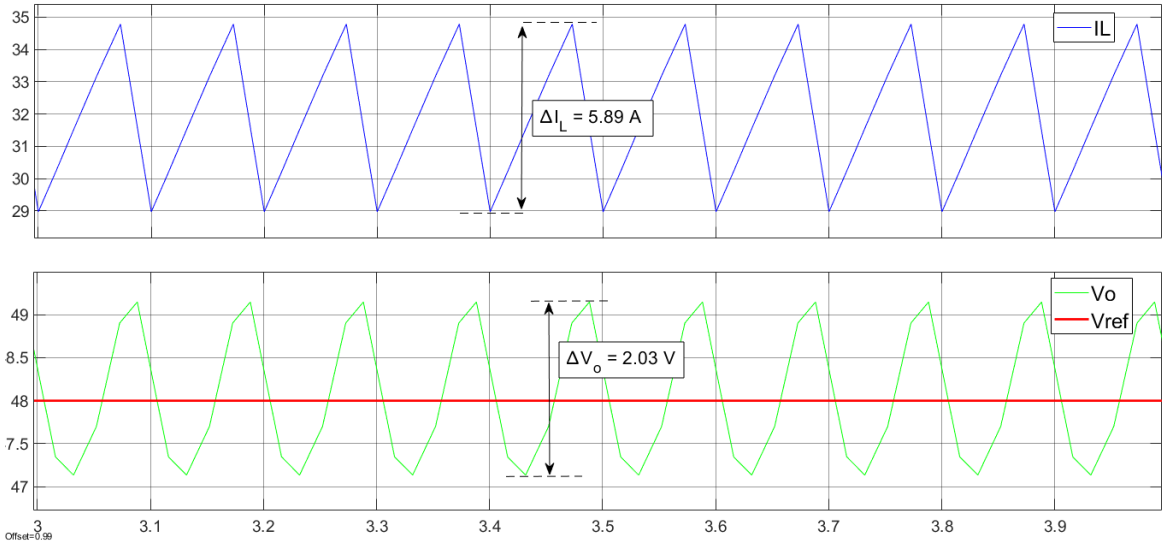


Figura 37. Corriente de rizado del inductor y rizado de la tensión de salida.

7.4. Modelado de los algoritmos a implementar en el SoC FPGA

En este apartado se presenta el modelo en Simulink de los diferentes algoritmos a implementar sobre el SoC FPGA.

7.4.1. Función auxCtrl

La función *auxCtrl* genera un vector de valores 1 y 0 a partir de la matriz $Pout$, que permite controlar el funcionamiento de la matriz de interruptores y modificar el conexionado de los paneles fotovoltaicos.

Para el caso de un *array* fotovoltaico (3 x 3), el tamaño del vector de control es de 54 valores. Está dividido en 9 campos de 6 valores cada una. Cada uno de los campos del vector de control corresponde a un SubSwitchMatrix.

Los seis valores de cada campo corresponden a los estados de los interruptores de cada SubSwitchMatrix. Si el valor es 1, indica que el interruptor correspondiente está cerrado, y si es 0 está abierto. El código de la función *auxCtrl* se lista a continuación.

Código fuente 5. Función auxCtrl

```
%This function only works for input matrices of a maximun size of 9x9
function ctrl = auxCtrl(Pout)
%% Parameters
FISRT_POS = 11;
NUM_SWITCHES = 6;   %6 switches per solar module

%% Initialization
[rows, columns] = size(Pout);
elem = rows * columns;
out = zeros(1, elem * NUM_SWITCHES);

%% Loop
for row = 1:rows
    for col = 1:columns
        firstDigit = floor(Pout(row, col) / 10);
        if firstDigit == 1
            index = (Pout(row, col) - FISRT_POS) * NUM_SWITCHES;
        else
            index = ((firstDigit * columns) - (columns - mod(Pout(row,
col), 10)) - 1) * NUM_SWITCHES;
        end
        %row indicates the tier number
        out(index + row) = 1;
        out(index + 3 + row) = 1;
    end
end
%% Return
ctrl = out;
end
```

7.4.2. Generador de PWM

El generador PWM es el encargado crear una señal PWM a partir del ciclo de trabajo calculado por el controlador MPPT, que permite controlar el funcionamiento del *Buck Converter*. Para modelarlo se utilizan bloques del *toolbox* HDL Coder, lo que permite crear una implementación *hardware* de este en capítulos futuros.

El generador diseñado crea una señal PWM de 10 KHz de frecuencia, que es la frecuencia que se ha utilizado para dimensionar el *Buck Converter* del apartado 7.3.3. Para lograr esto, primero se pasa el valor del ciclo de trabajo a un valor entero que varía entre 0 y 10000 a intervalos de 1. Una vez se ha pasado el ciclo de trabajo a un valor entero, se compara su valor con el de una señal diente de sierra de 10 KHz de frecuencia generada por un contador. Cuando el valor del ciclo de trabajo es mayor o igual que el de la señal diente de sierra, la salida del generador vale 1. Por el contrario, cuando el valor del ciclo de trabajo es menor, la salida es 0 (Figura 38).

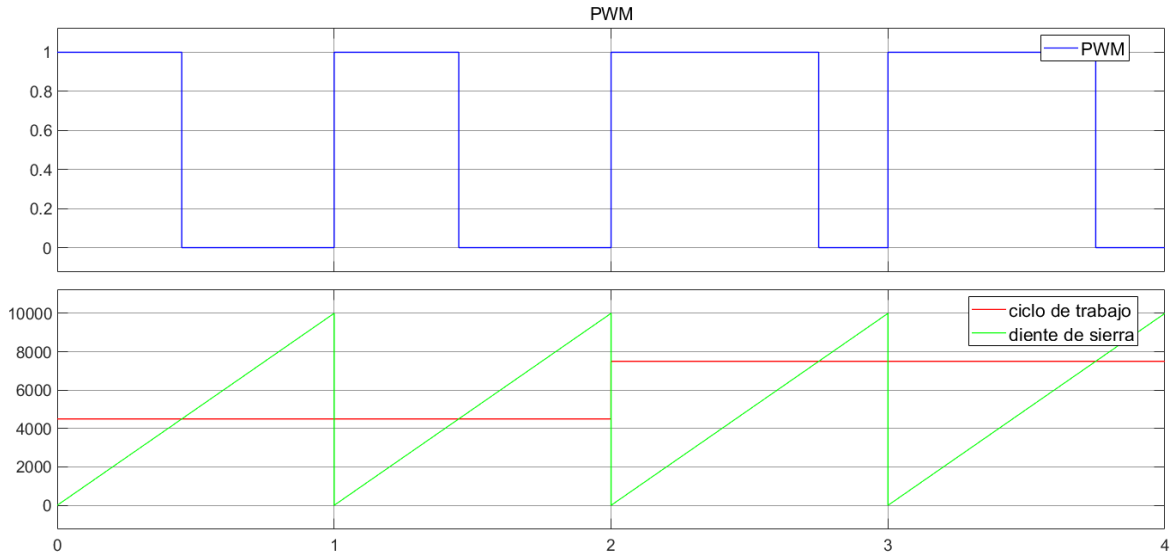


Figura 38. Generación de la señal PWM.

De esta manera se consigue generar una señal PWM de 10 KHz de frecuencia con el ciclo de trabajo determinado por el MPPT o el controlador de la batería. La implementación del generador PWM en Simulink se muestra en Figura 39.

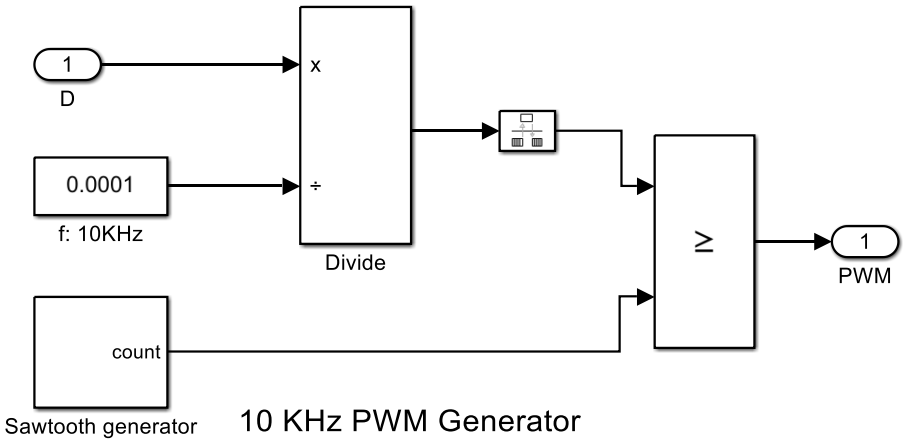


Figura 39. Implementación del generador de PWM.

7.5. Sistema final

Una vez se han modelado todos los bloques necesarios, se interconectan para montar el sistema final que se ilustra en la Figura 40. La planta a controlar está formada por nueve bloques PV Array, la matriz de interruptores, el *Buck Converter* y la carga. La tensión y corriente a la salida del *array* fotovoltaico se muestrea con una frecuencia de 16.6 KHz utilizando los bloques *voltage measurement* y *current measurement* del *toolbox Simscape Electrical*. Los valores medidos a la salida del *array* fotovoltaico se pasan como entradas al MPPT para que realice el cálculo del ciclo de trabajo del PWM del *Buck Converter*.

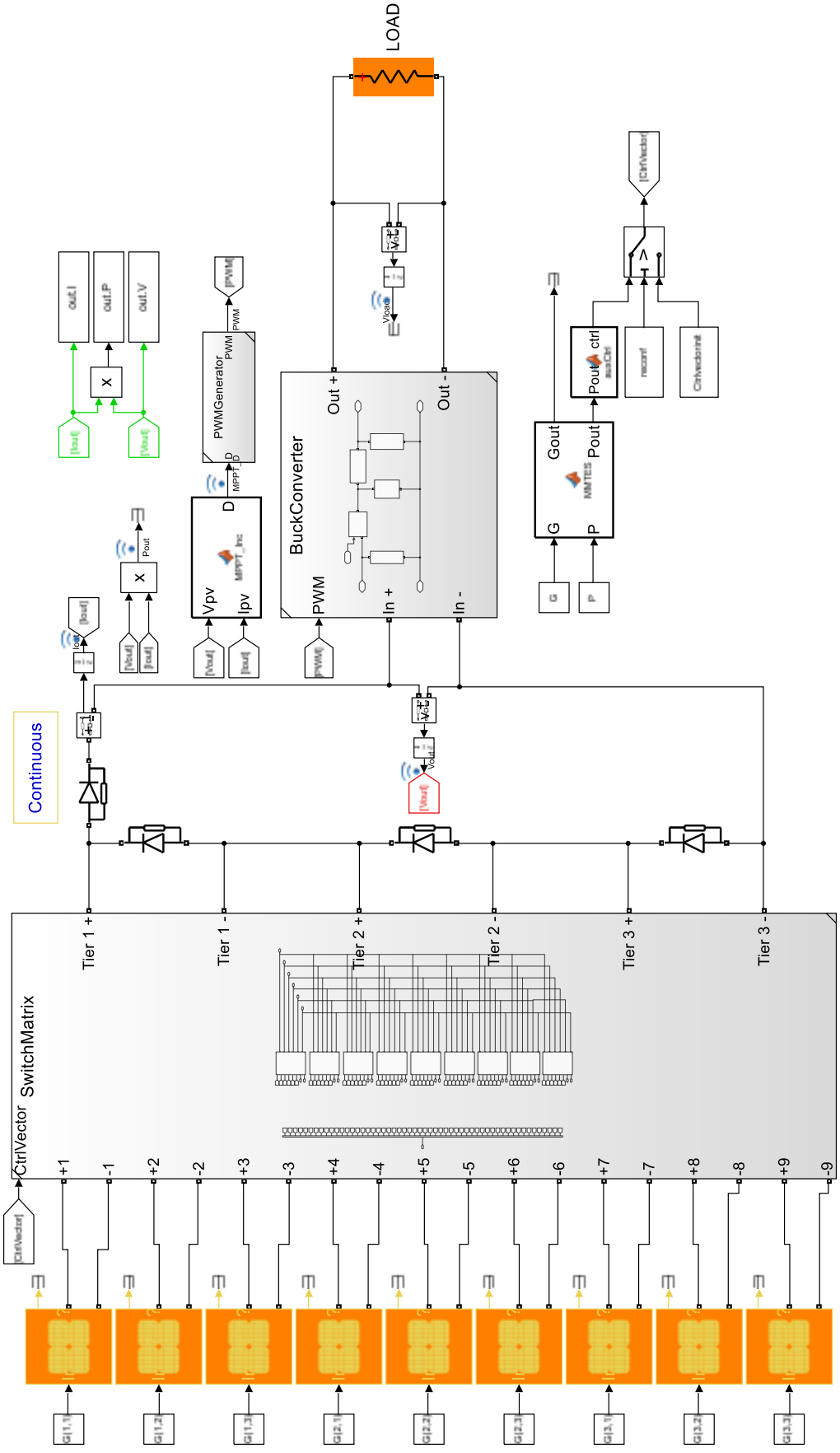


Figura 40. Modelado final del sistema en Simulink.

7.6. Simulación *Model-in-the-Loop* (MIL)

Para realizar la simulación MIL del sistema de la Figura 40 se configura el *solver* de Simulink como *fixed-step* con un *step-size* de 10 ns. Los datos de irradiancia solar que se utilizan para realizar la simulación son los que se presentan en la Tabla 5. Estos valores de irradiancia son de 3 de los 4 casos de estudio que se presentaron con anterioridad en el Capítulo 5. Se escogen los 3 casos cuya matriz de irradiancia es tamaño (3x3).

Tabla 5. Valores de irradiancia utilizados en la simulación MIL y SIL.

	$\boldsymbol{G}\ (\frac{\boldsymbol{W}}{\boldsymbol{m^2}})$	$\boldsymbol{P}$
Caso 1	$\begin{bmatrix}400 & 500 & 600\\ 500 & 600 & 700\\ 600 & 700 & 800\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{bmatrix}$
Caso 2	$\begin{bmatrix}700 & 700 & 700\\ 700 & 700 & 600\\ 600 & 500 & 300\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{bmatrix}$
Caso 4	$\begin{bmatrix}900 & 900 & 900\\ 600 & 600 & 600\\ 250 & 200 & 300\end{bmatrix}$	$\begin{bmatrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{bmatrix}$

En las simulaciones se utilizan 2 tipos de carga diferentes: resistivas R e inductivas L. Para cada tipo de carga se emplean varios valores diferentes. Los valores de las cargas utilizadas durante las simulaciones se presentan en la Tabla 6.

Tabla 6. Cargas utilizadas durante la simulación MIL y SIL.

	$\boldsymbol{R}\ (\boldsymbol{\Omega})$	$\boldsymbol{L}\ (\boldsymbol{H})$
$\boldsymbol{R_1}$	0.5	0.0
$\boldsymbol{R_2}$	1.0	0.0
$\boldsymbol{R_3}$	1.5	0.0
$\boldsymbol{RL_1}$	1.0	0.5
$\boldsymbol{RL_2}$	1.0	1.0
$\boldsymbol{RL_3}$	1.0	1.5

En la Figura 41 se comparan las curvas P-V para el caso 1. El punto de funcionamiento determinado por el MPPT se marca en rojo cuando se está aplicando el algoritmo MMTES y en verde cuando no. Se puede observar como el MPP conseguido al aplicar el algoritmo

MMTES es mayor que el conseguido sin aplicarlo. Para este caso, el valor de la potencia en el MPP al aplicar el algoritmo MMTES es un 9% superior. Además, la curva P-V presenta un solo máximo global. Esto facilita el buen funcionamiento del MPPT ya que no existen máximos locales que puedan confundir al controlador.

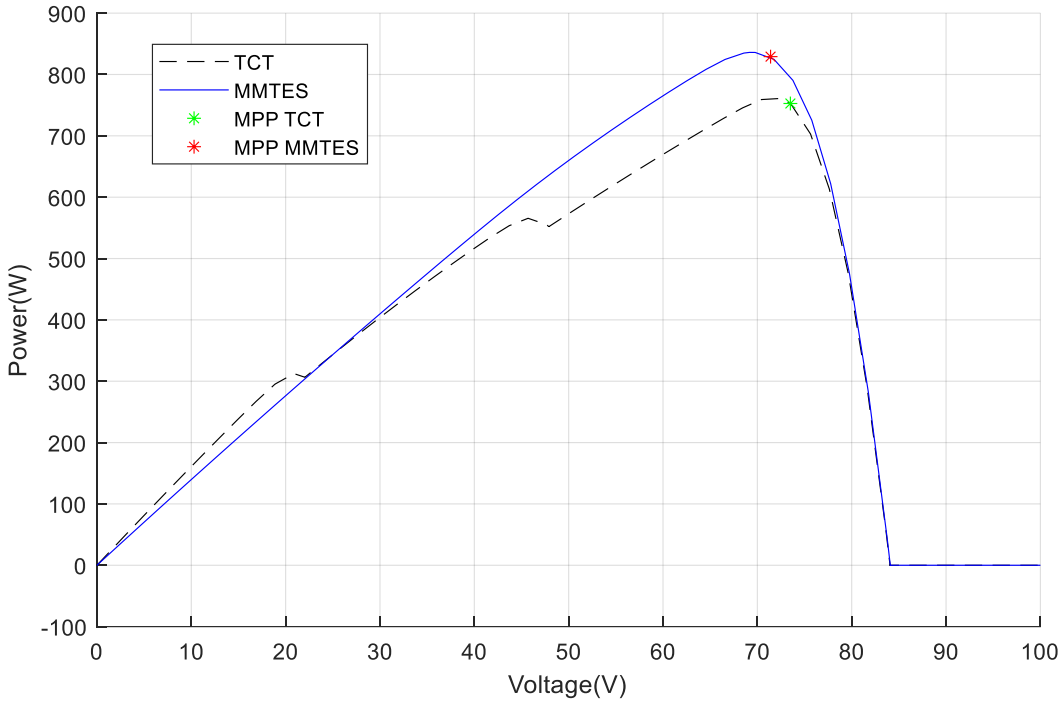


Figura 41. Curvas P-V caso 1.

En la Figura 42, la Figura 43 y la Figura 44 se comparan las potencias de salida obtenidas aplicando y no aplicando el algoritmo MMTES para el caso 1 con las cargas R_1, R_2 y R_3. Al aplicar el algoritmo MMTES se consigue una mejora en la potencia de salida de en torno a un 10% en todos los casos.

En la Figura 42 el rizado de la potencia de salida es de un 1.53% y 2.35% aplicando el algoritmo MMTES y no aplicándolo respectivamente. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 46 ms frente a los 67 ms al no aplicar reconfiguración.

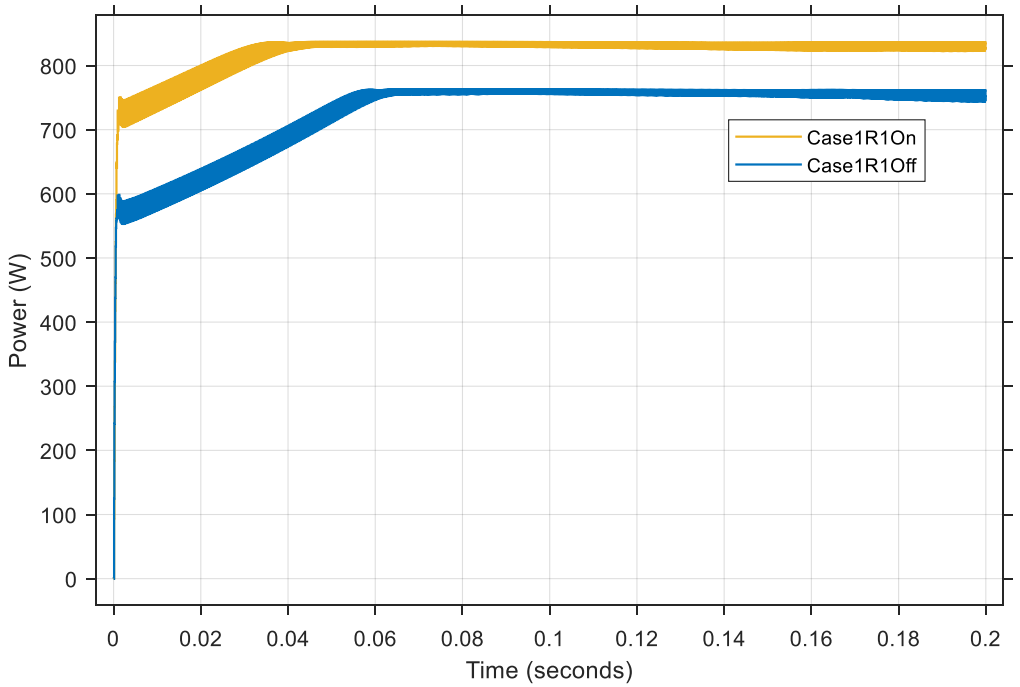


Figura 42. Comparativa de la potencia de salida con carga R1 en el caso 1.

En la Figura 43, el rizado de la potencia de salida al aplicar el algoritmo MMTES es de un 0.93% frente a un 1.9% no aplicándolo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 19 ms frente a los 14 ms al no aplicar reconfiguración.

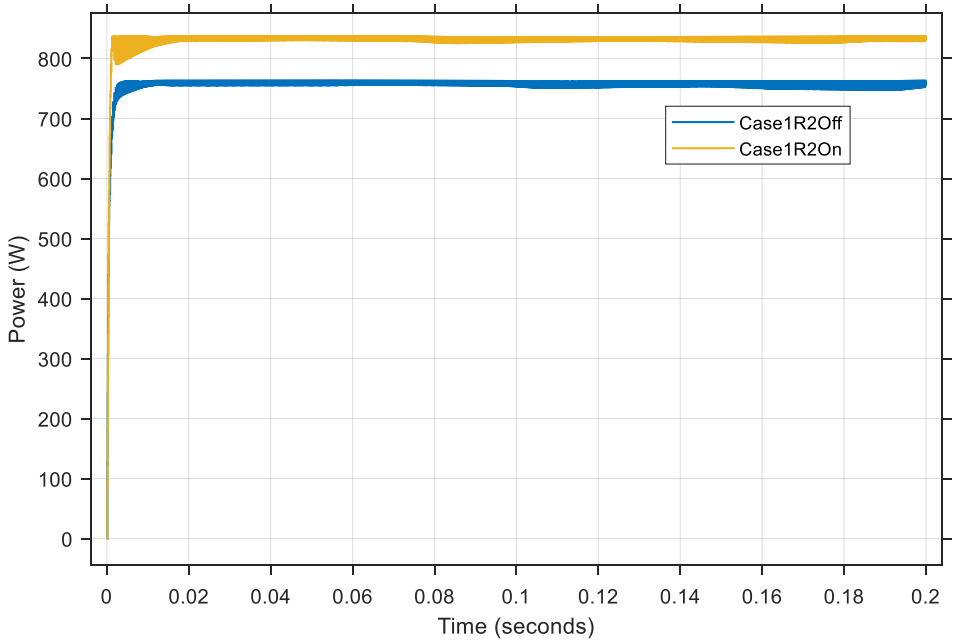


Figura 43.Comparativa de la potencia de salida con carga R_2 en el caso 1.

En la Figura 44 el rizado de la potencia de salida es de un 0.98% y 1% aplicando el algoritmo MMTES y no aplicándolo respectivamente. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 67 ms frente a los 50 ms al no aplicar reconfiguración.

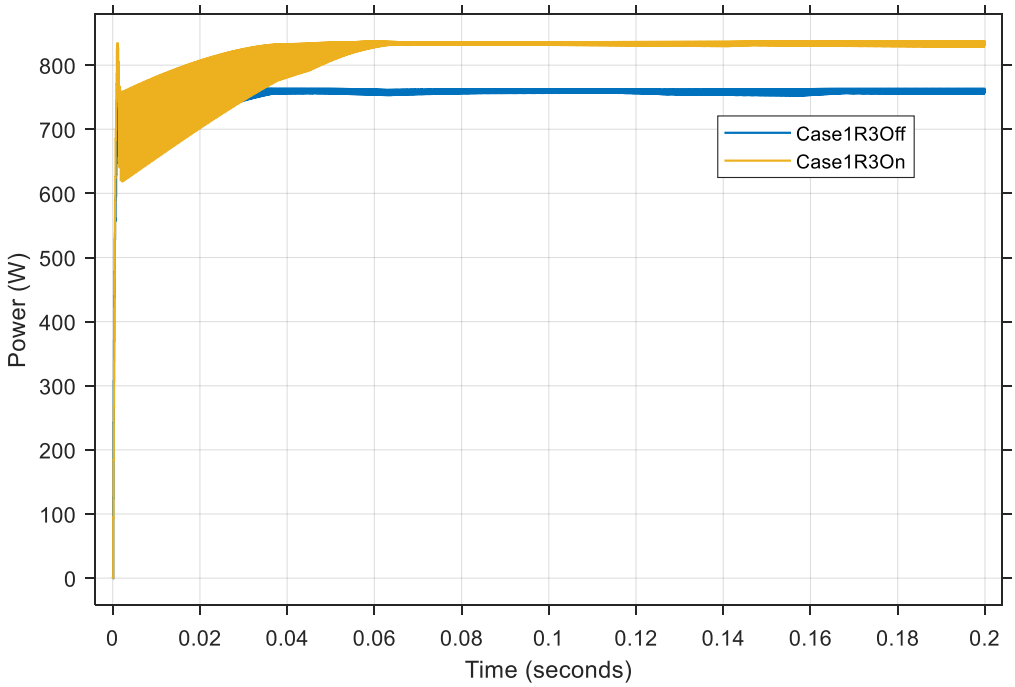


Figura 44.Comparativa de la potencia de salida con carga R_3 en el caso 1.

En la Figura 45, la Figura 46 y la Figura 47 se comparan las potencias de salida obtenidas aplicando y no aplicando el algoritmo MMTES para el caso 1 con las cargas RL_1, RL_2 y RL_3. Al aplicar el algoritmo MMTES se consigue una mejora en la potencia de salida de en torno a un 9.7% en todos los casos.

En la Figura 45 el rizado de la potencia de salida es de un 0.96% y 0.97% aplicando el algoritmo MMTES y no aplicándolo respectivamente. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 310 ms frente a los 290 ms al no aplicar reconfiguración.

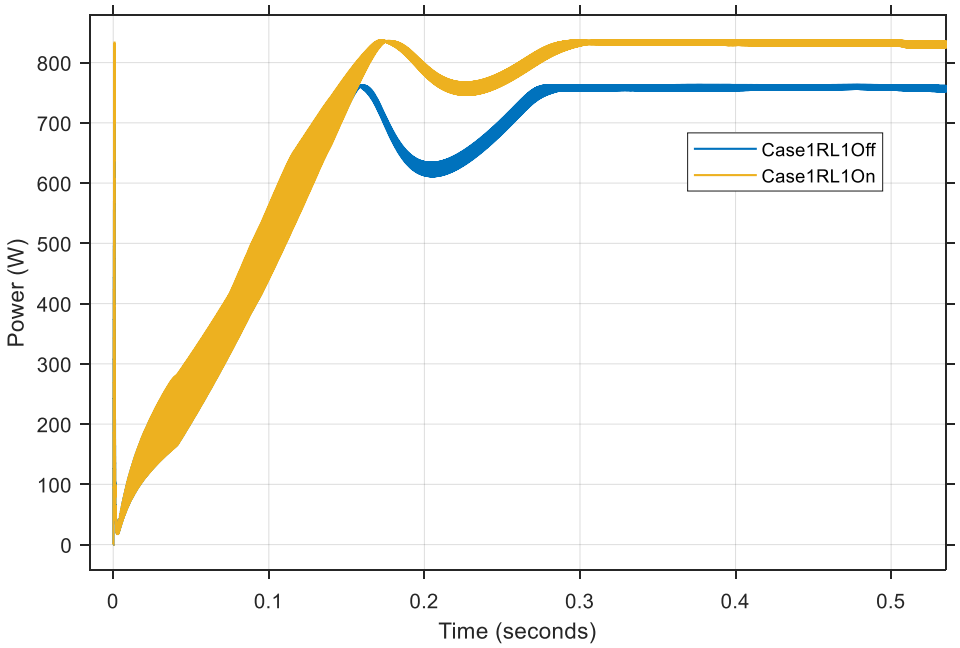


Figura 45. Comparativa de la potencia de salida con carga RL_1 en el caso 1.

En la Figura 46 el rizado de la potencia de salida al aplicar el algoritmo MMTES es de un 1.1%. El rizado obtenido es el mismo al no aplicarlo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES se consigue en 410 ms frente a los 380 ms al no aplicar reconfiguración.

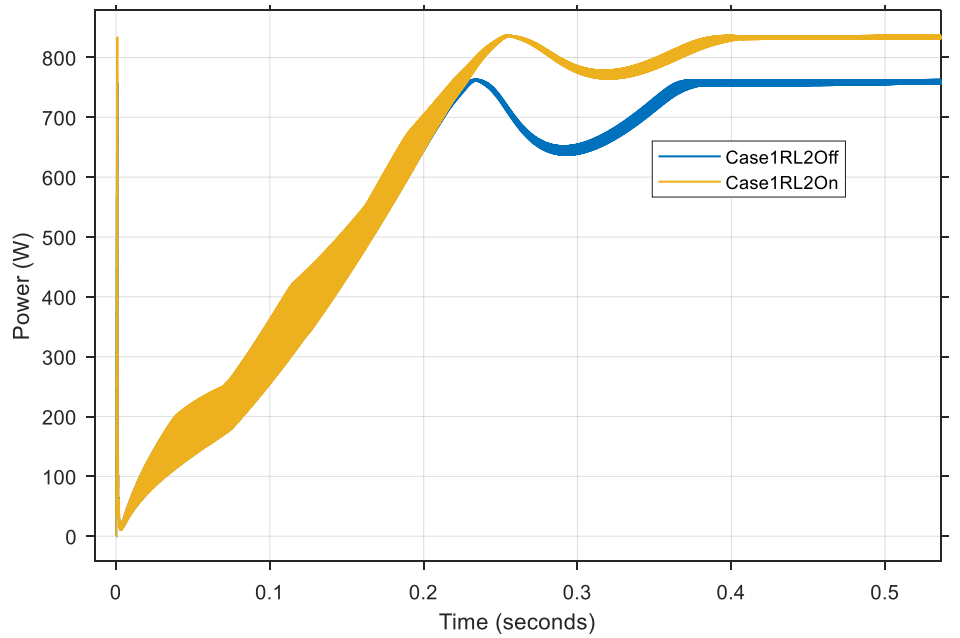


Figura 46. Comparativa de la potencia de salida con carga RL_2 en el caso 1.

En la Figura 47 el rizado de la potencia de salida es de un 0.86% y 0.98% aplicando el algoritmo MMTES y no aplicándolo respectivamente. En cuanto a la estabilización de la

potencia de salida, al aplicar el MMTES esta se consigue en 480 ms frente a los 455 ms al no aplicar reconfiguración.

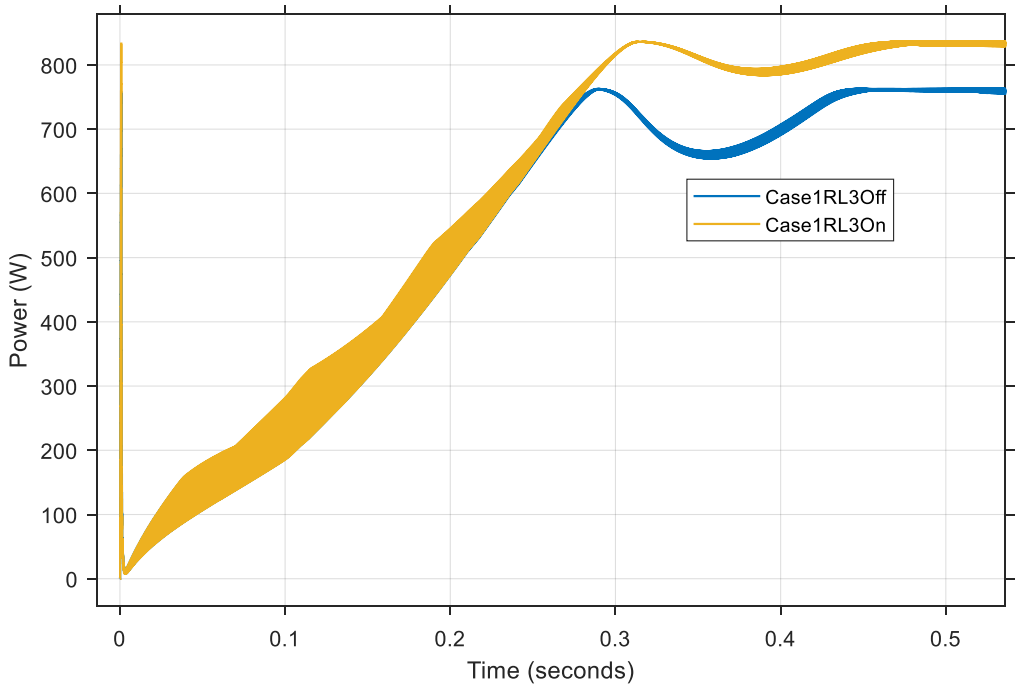


Figura 47. Comparativa de la potencia de salida con carga RL_3 en el caso 1.

En la Figura 48 se comparan las curvas P-V para el caso 2. La potencia en el MPP conseguida al aplicar el algoritmo MMTES es mayor que la conseguida sin aplicarlo. El aumento de la potencia es de un 13.34%. Además, si bien la curva P-V conseguida al aplicar el algoritmo MMTES sigue teniendo 2 máximos diferenciados, el primero de ellos es mucho menos pronunciado que en el caso en el que no se aplica reconfiguración. Esto facilita la labor del MPPT de encontrar el MPP.

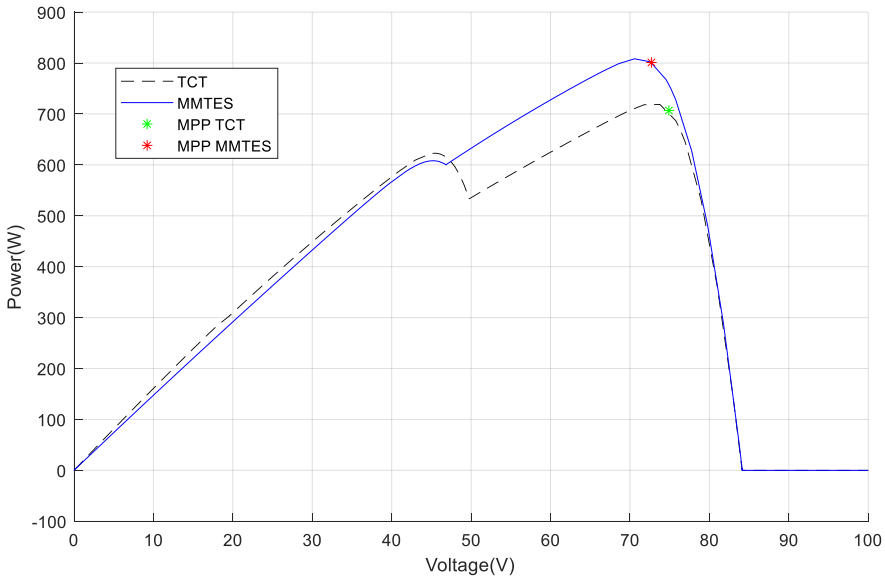


Figura 48. Curvas P-V caso 2.

En la Figura 49, la Figura 50 y la Figura 51 se comparan las potencias de salida obtenidas aplicando y no aplicando el algoritmo MMTES para el caso 2 con las cargas R_1, R_2 y R_3. Al aplicar el algoritmo MMTES se consigue una mejora en la potencia de salida de en torno a un 15% en los tres casos.

En la Figura 49 el rizado de la potencia de salida es de un 2.28% al aplicar el algoritmo MMTES y un 2.86% sin aplicarlo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 55 ms frente a los 65 ms al no aplicar reconfiguración.

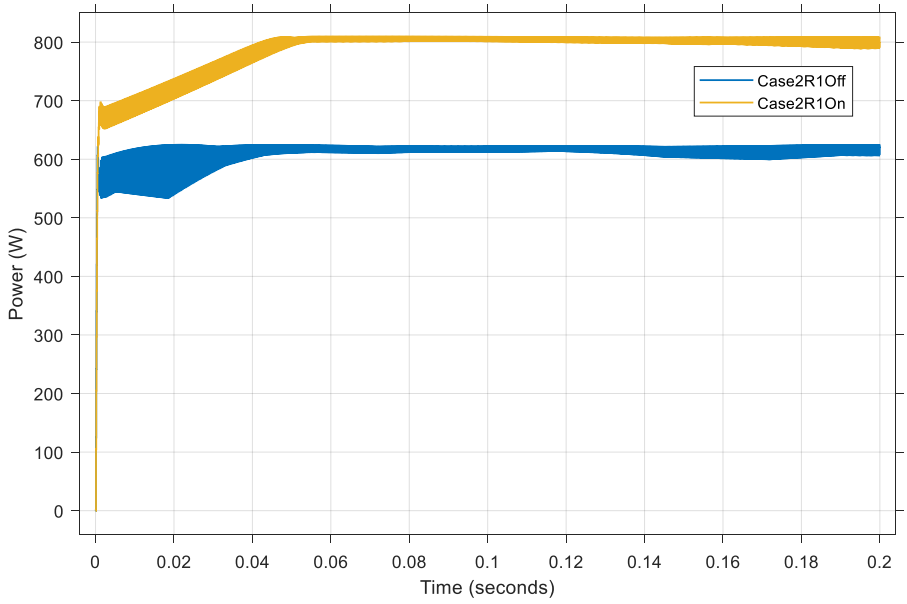


Figura 49.Comparativa de la potencia de salida con carga R_1 en el caso 2.

En la Figura 50 el rizado de la potencia de salida es de un 1% al aplicar reconfiguración frente a un 1.96% no aplicando el algoritmo MMTES. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 14 ms frente a los 25 ms al no aplicar reconfiguración.

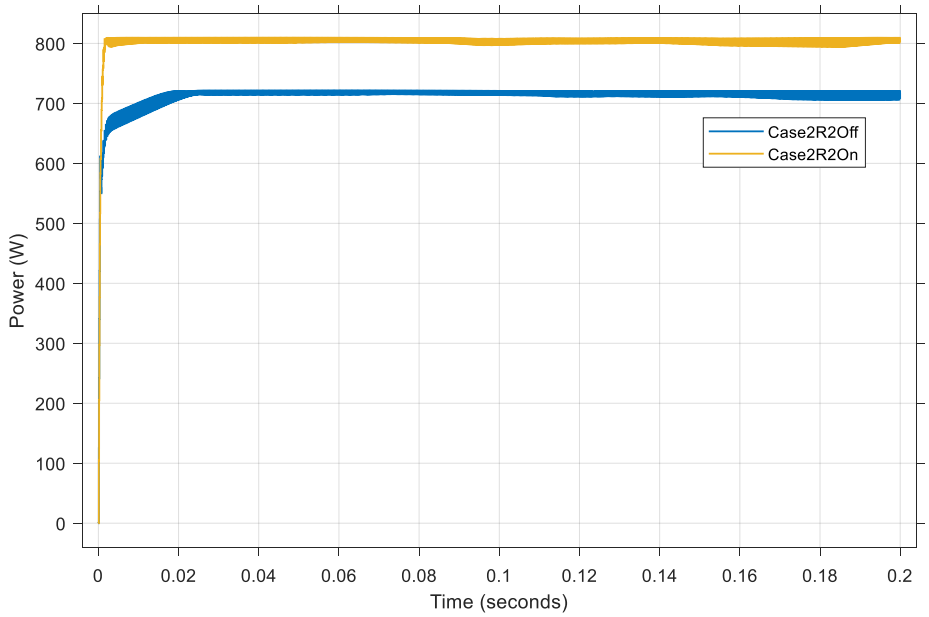


Figura 50. Comparativa de la potencia de salida con carga R_2 en el caso 2.

En la Figura 51 el rizado de la potencia de salida es de un 1% aplicando el algoritmo MMTES y un 0.95% no aplicándolo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 53 ms frente a los 22 ms al no aplicar reconfiguración.

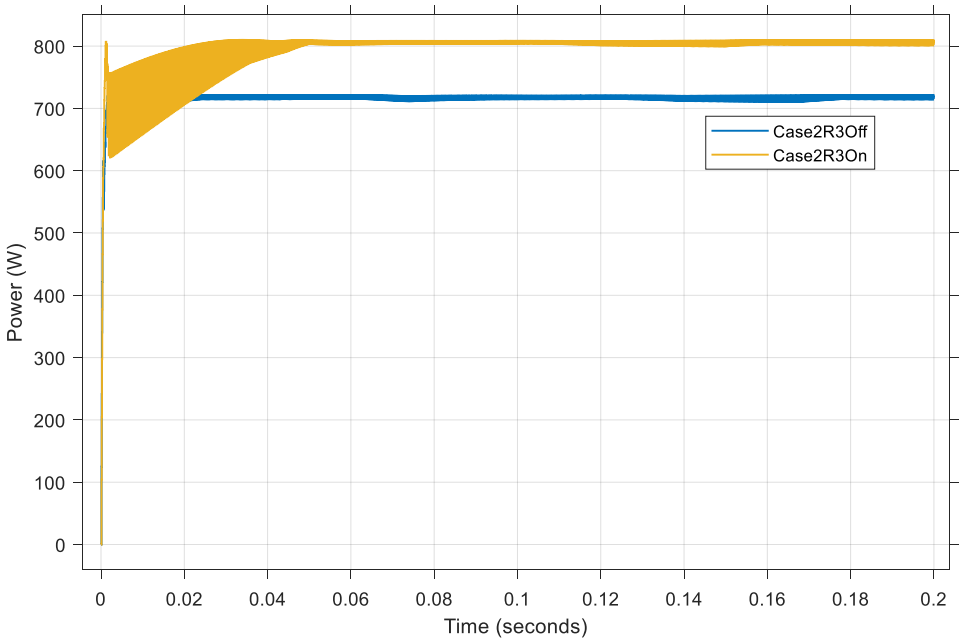


Figura 51. Comparativa de la potencia de salida con carga R_3 en el caso 2.

En la Figura 52, Figura 53 y Figura 54 se comparan las potencias de salida obtenidas aplicando y no aplicando el algoritmo MMTES para el caso 2 con las cargas RL_1, RL_2 y RL_3. Al aplicar el algoritmo MMTES se consigue una mejora en la potencia de salida de en torno a un 12.3% en todos los casos.

En la Figura 52 el rizado de la potencia de salida es de un 1.4% al aplicar el algoritmo MMTES y un 1.28% sin aplicarlo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 300 ms frente a los 280 ms al no aplicar reconfiguración.

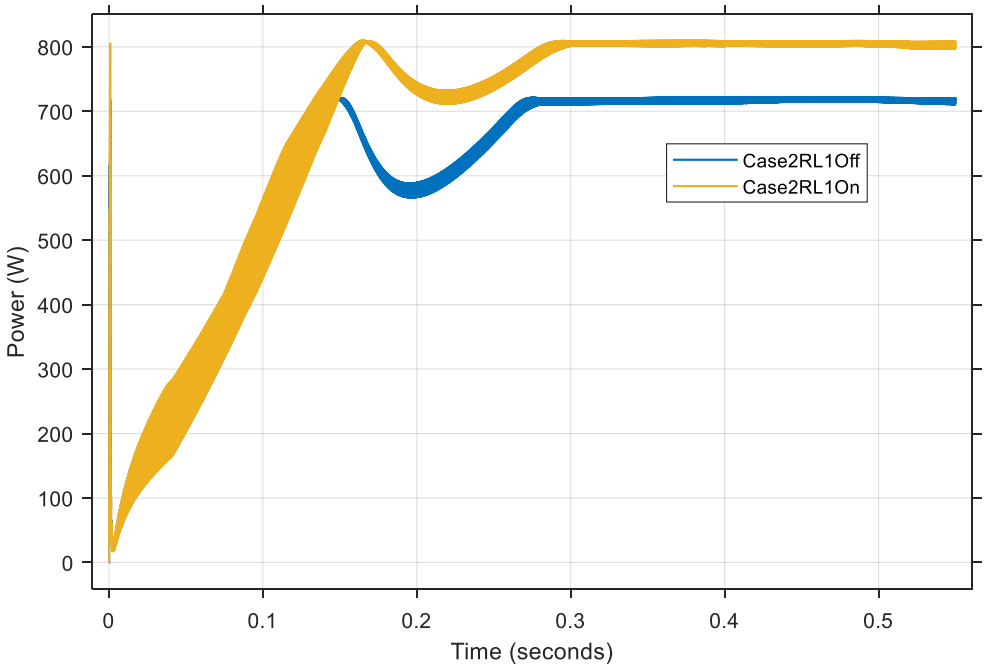


Figura 52. Comparativa de la potencia de salida con carga RL_1 en el caso 2.

En la Figura 53 el rizado de la potencia de salida es de un 0.77% al aplicar reconfiguración frente a un 1% no aplicando el algoritmo MMTES. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 420 ms frente a los 380 ms al no aplicar reconfiguración.

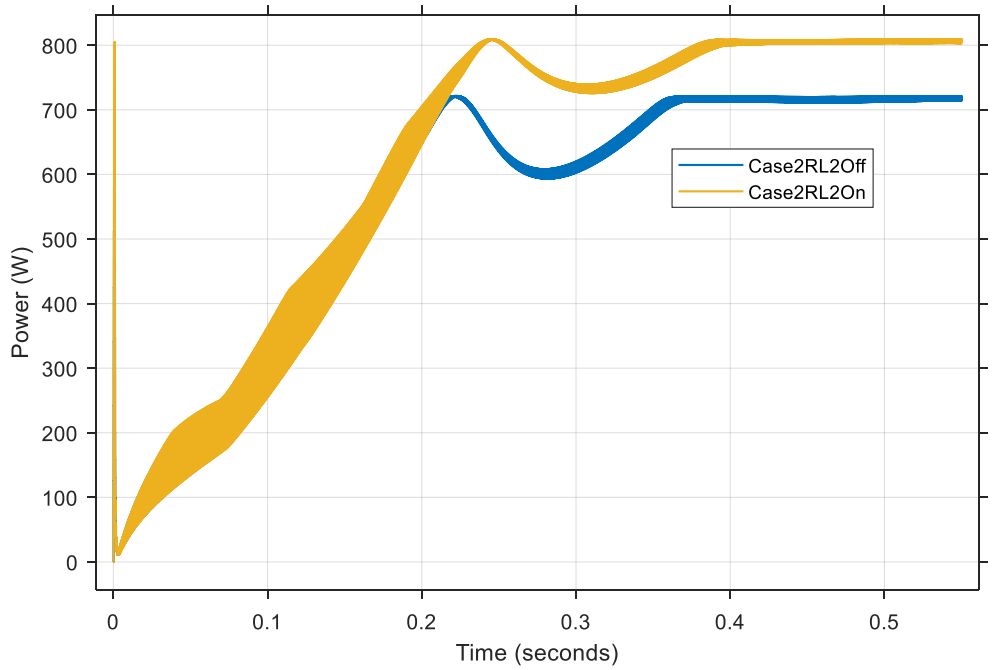


Figura 53. Comparativa de la potencia de salida con carga RL_2 en el caso 2.

En la Figura 54 el rizado de la potencia de salida es de un 1% aplicando el algoritmo MMTES y un 0.82% no aplicándolo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 475 ms frente a los 440 ms al no aplicar reconfiguración.

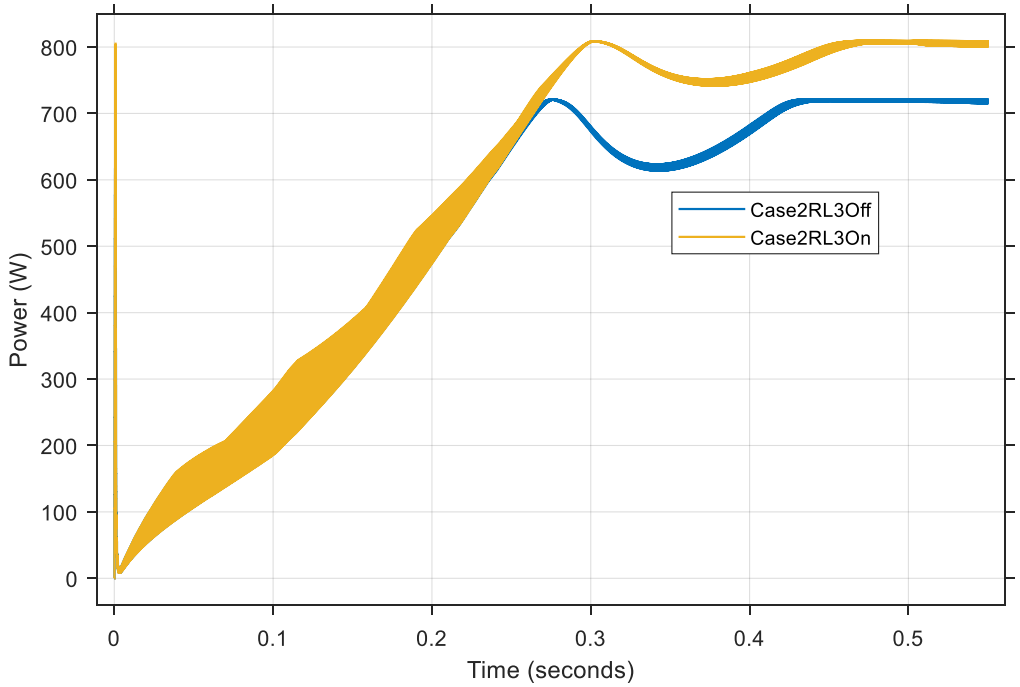


Figura 54. Comparativa de la potencia de salida con carga RL_3 en el caso 2.

En la Figura 55 se comparan las curvas P-V para el caso 4. Se puede observar como el MPP conseguido al aplicar el algoritmo MMTES es mayor que el conseguido sin aplicarlo. Para este caso, el valor de la potencia en el MPP al aplicar el algoritmo MMTES es un 40.7% superior. Además, la curva P-V presenta un solo máximo global frente a los 3 máximos que presenta al no aplicar reconfiguración.

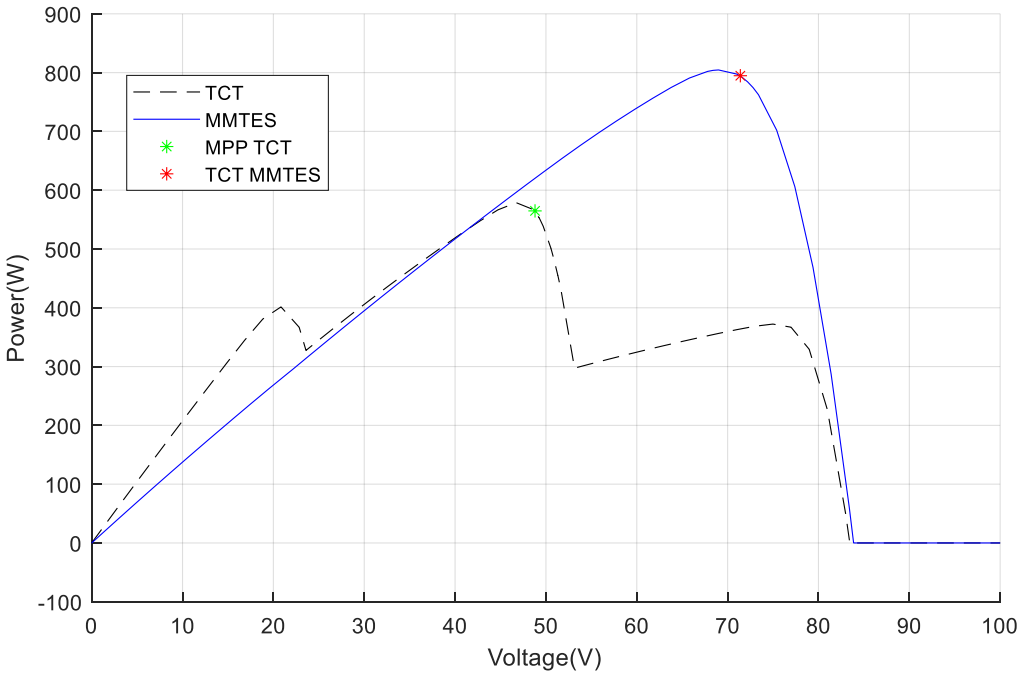


Figura 55.Curvas P-V caso 4.

En la Figura 49, Figura 50 y Figura 51 se comparan las potencias de salida obtenidas aplicando y no aplicando el algoritmo MMTES para el caso 2 con las cargas R_1, R_2 y R_3. Al aplicar el algoritmo MMTES se consigue una mejora en la potencia de salida de en torno a un 41% en los 3 casos.

En la Figura 56 el rizado de la potencia de salida es de un 1.37% al aplicar el algoritmo MMTES y un 3.67% sin aplicarlo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 0.05 s frente a los 0.03 s al no aplicar reconfiguración.

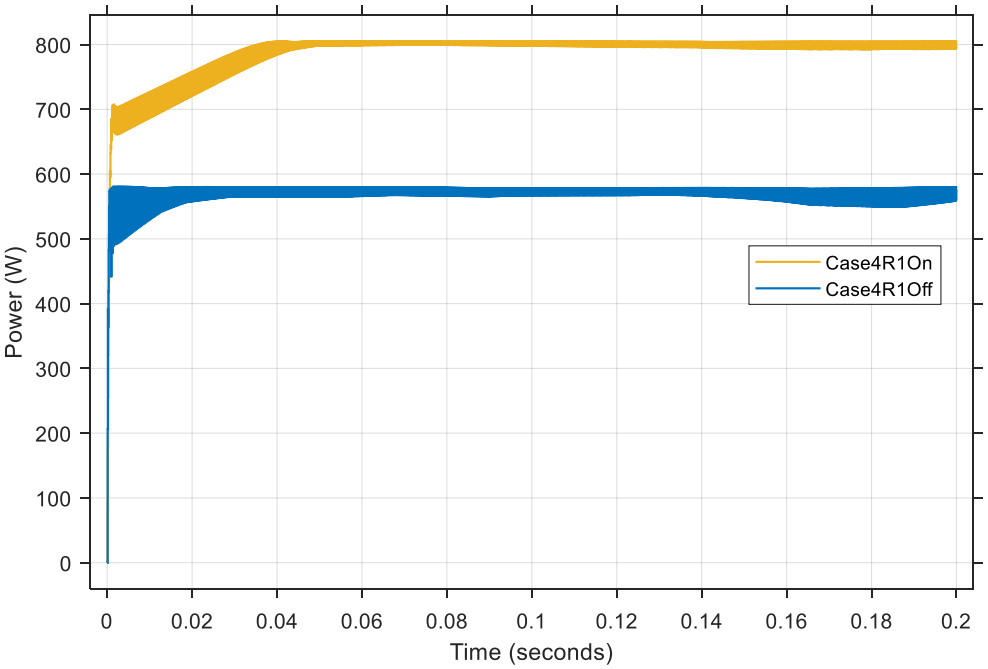


Figura 56. Comparativa de la potencia de salida con carga R1 en el caso 4.

En la Figura 57 el rizado de la potencia de salida es de un 1.27% al aplicar reconfiguración frente a un 2.62% no aplicando el algoritmo MMTES. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 25 ms frente a los 53 ms al no aplicar reconfiguración.

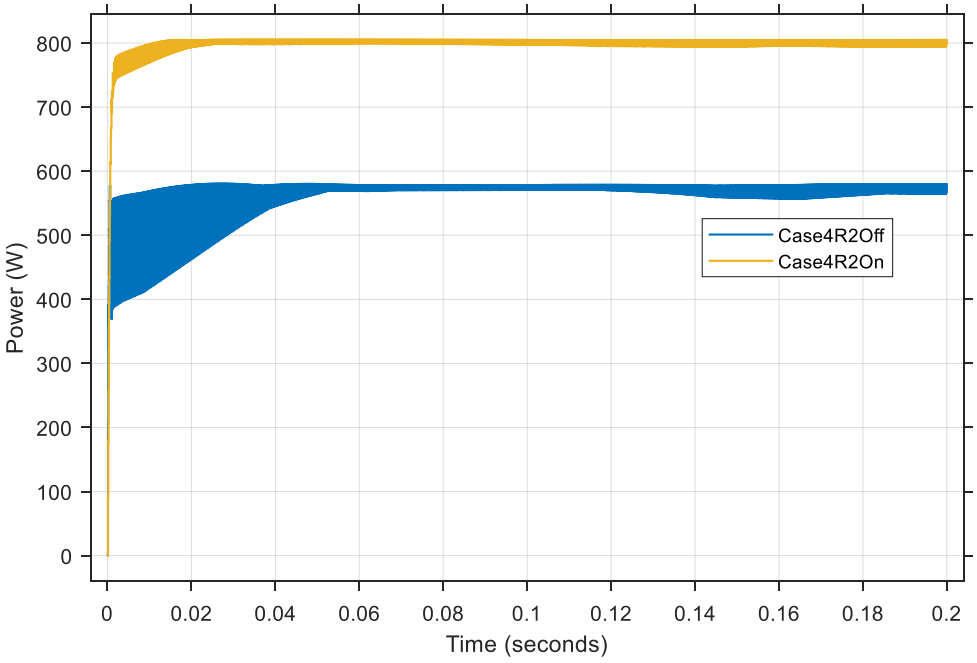


Figura 57. Comparativa de la potencia de salida con carga R_2 en el caso 4.

En la Figura 58 el rizado de la potencia de salida es de un 1.05% aplicando el algoritmo MMTES y un 2.67% no aplicándolo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 22 ms frente a los 65 ms al no aplicar reconfiguración.

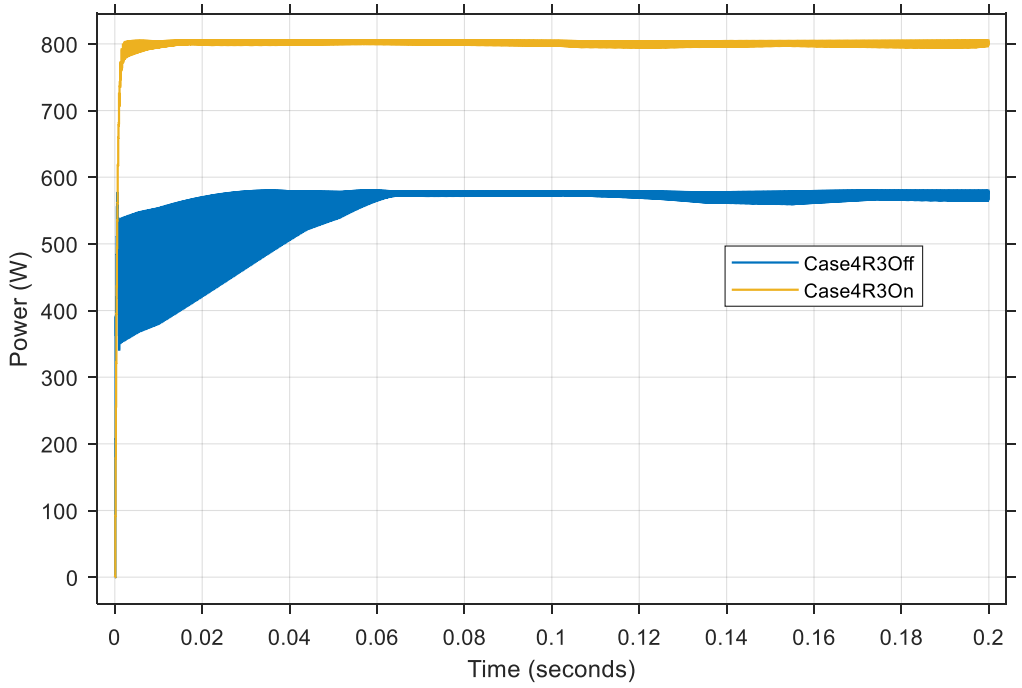


Figura 58. Comparativa de la potencia de salida con carga R_3 en el caso 4.

En la Figura 59, Figura 60 y Figura 61 se comparan las potencias de salida obtenidas aplicando y no aplicando el algoritmo MMTES para el caso 4 con las cargas RL_1, RL_2 y RL_3. Al aplicar el algoritmo MMTES se consigue una mejora en la potencia de salida de hasta un 116.5%.

En la Figura 59 el rizado de la potencia de salida es de un 1.25% al aplicar el algoritmo MMTES y un 2.48% sin aplicarlo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 300 ms frente a los 280 ms al no aplicar reconfiguración.

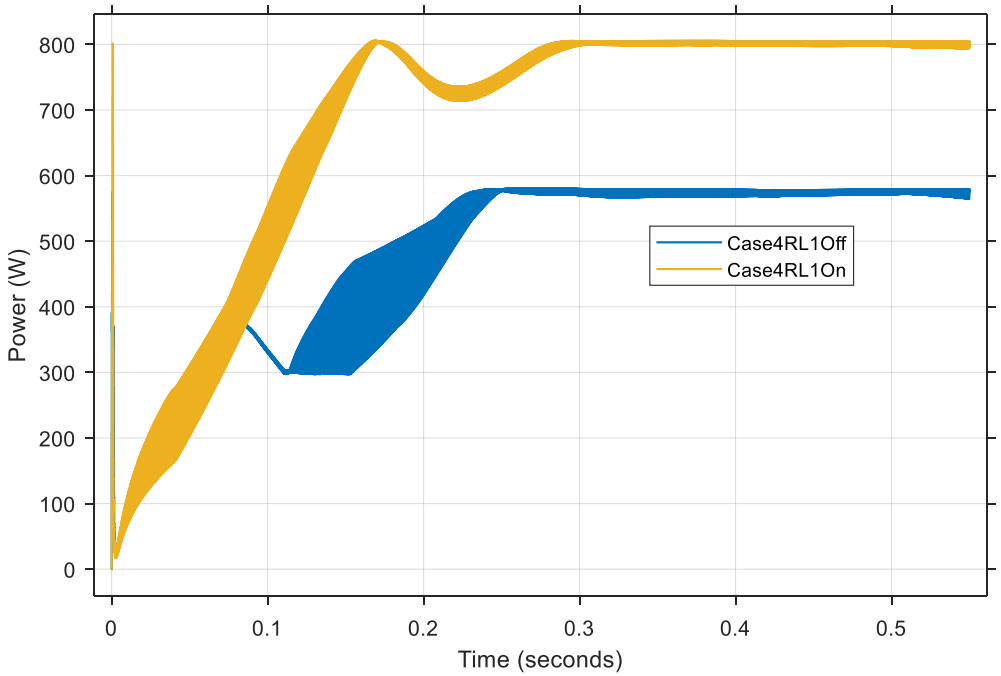


Figura 59. Comparativa de la potencia de salida con carga RL1 en el caso 4.

En la Figura 60 el rizado de la potencia de salida es de un 0.63% al aplicar reconfiguración frente a un 0.3% no aplicando el algoritmo MMTES. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 400 ms frente a los 315 ms al no aplicar reconfiguración.

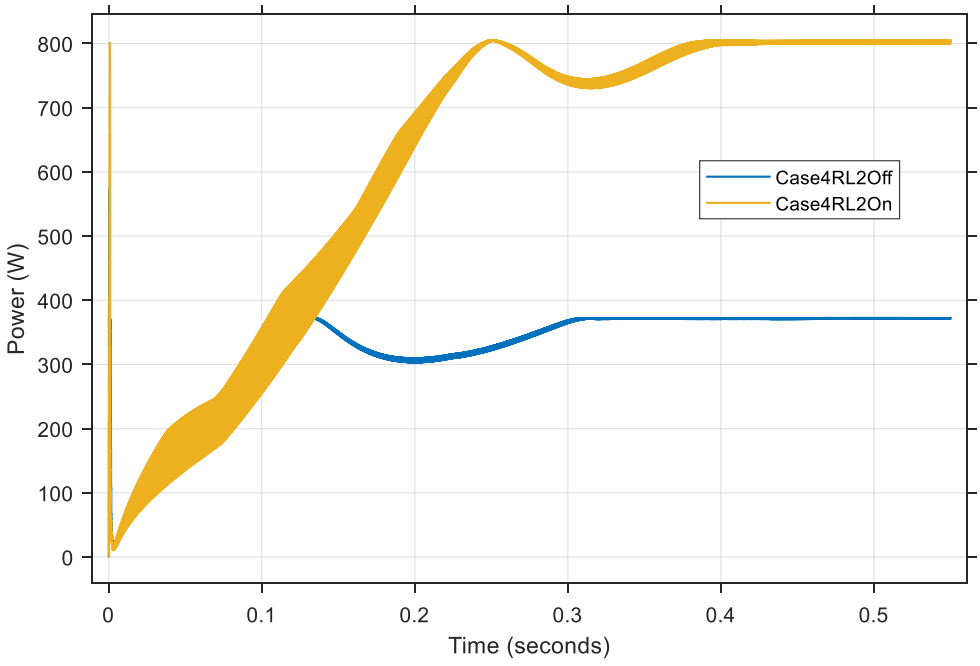


Figura 60. Comparativa de la potencia de salida con carga RL_2 en el caso 4.

En la Figura 61 el rizado de la potencia de salida es de un 0.91% aplicando el algoritmo MMTES y un 0.35% no aplicándolo. En cuanto a la estabilización de la potencia de salida, al aplicar el MMTES esta se consigue en 480 ms frente a los 345 ms al no aplicar reconfiguración.

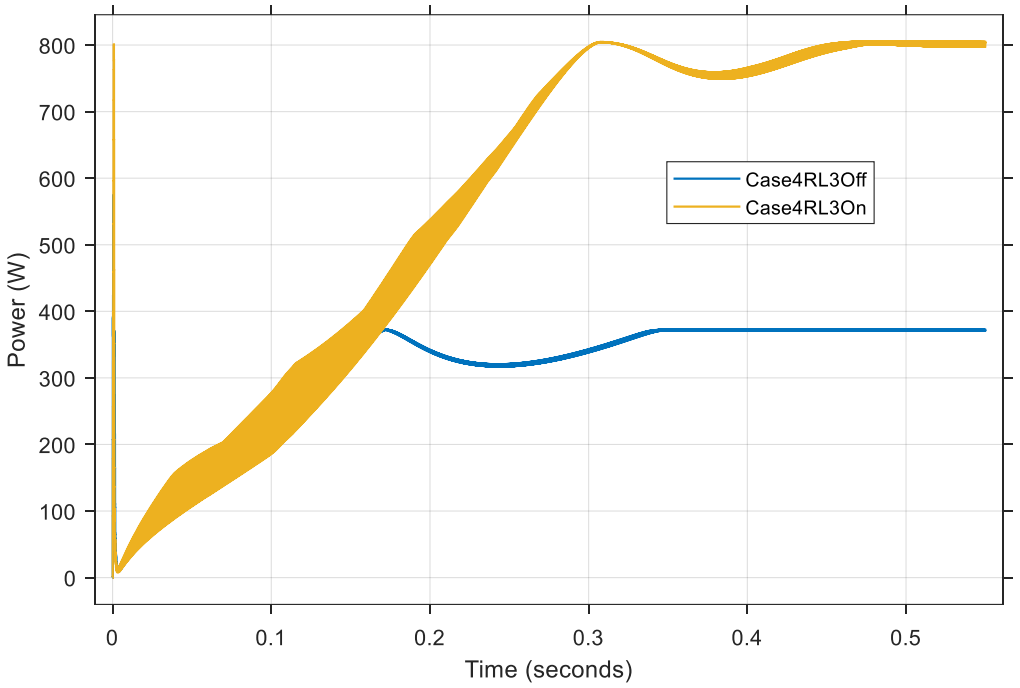


Figura 61. Comparativa de la potencia de salida con carga RL_3 en el caso 4.

En la Tabla 7 y Tabla 8 se resumen los resultados obtenidos durante la simulación MIL con cargas resistivas e inductivas.

Tabla 7. Comparativa de resultados de la simulación MIL con carga resistiva.

	Pout (W)					
	MMTES			**TCT**		
	R_1	**R_2**	**R_3**	**R_1**	**R_2**	**R_3**
caso 1	823.5	829.1	828.9	743.7	752.7	755.3
caso 2	789.7	801.1	801.3	606.6	706.8	714.5
caso 4	793.6	794.7	796.8	559.0	564.8	564.6

Tabla 8. Comparativa de resultados de la simulación MIL con carga inductiva.

	Pout (W)					
	MMTES			TCT		
	RL_1	RL_2	RL_3	RL_1	RL_2	RL_3
caso 1	825.6	836.8	834.3	751.1	755.8	762.5
caso 2	796.8	803.7	806.4	710.9	713.8	717.6
caso 4	794.3	801.0	802.7	574.9	371.6	370.8

7.7. Simulación *Software-in-the-Loop* (SIL)

Para realizar la simulación SIL hay que transformar los bloques que contienen el algoritmo MMTES, el controlador MPPT, la función *auxCtrl* y el generador de PWM a modelos. Una vez hecho esto, se configuran los modelos en modo SIL para que Simulink genere código de ellos al iniciar la simulación y utilice este código durante la misma.

Los valores de potencia de salida para los tres casos de estudio aplicando el algoritmo MMTES con carga resistiva e inductiva se presentan en la Tabla 9 y Tabla 10 respectivamente. En estas tablas también se comparan estos valores con los obtenidos en la simulación MIL.

Tabla 9. Comparativa de las potencias de salida obtenidas en la simulación MIL y SIL con carga resistiva.

	Pout (W)					
	MIL			SIL		
	R_1	R_2	R_3	R_1	R_2	R_3
caso 1	823.5	829.1	828.9	823.5	829.1	829.9
caso 2	789.7	801.1	801.3	789.7	801.1	801.3
caso 4	793.6	794.7	796.8	793.6	794.7	796.8

Tabla 10. Comparativa de las potencias de salida obtenidas en la simulación MIL y SIL con carga inductiva.

	Pout (W)					
	MIL			SIL		
	R_1	R_2	R_3	R_1	R_2	R_3
caso 1	825.6	836.8	834.3	825.6	836.8	834.3
caso 2	796.8	803.7	806.4	796.8	803.7	806.4
caso 4	794.3	801.0	802.7	794.3	801.0	802.7

Como se puede apreciar en ambas tablas, los valores de potencia de salida alcanzados en todos los casos en las simulaciones MIL y SIL son idénticos. Esto significa que Simulink es capaz de generar código para el algoritmo MMTES, el controlador MPPT, la función *auxCtrl* y el generador de PWM de manera satisfactoria. Gracias a esta comparativa se comprueba que se puede generar código a partir de las implementaciones desarrolladas en MATLAB/Simulink y se puede seguir la metodología de diseño sin problemas.

7.8. Conclusiones

En este capítulo se ha realizado el modelado del sistema a controlar en Simulink. Con el modelado del sistema se han llevado a cabo simulaciones MIL y SIL del funcionamiento del algoritmo MMTES y el controlador MPPT implementados en el Capítulo 5 y Capítulo 6. Los resultados obtenidos han sido satisfactorios tanto para la simulación MIL como la SIL. Se ha comprobado que el algoritmo MMTES mejora la producción de potencia en los 3 casos de estudio, llegando en el caso 4 a conseguir en torno a un 116% de mejora en la producción de potencia con respecto a una configuración TCT con carga inductiva. También se ha verificado que el controlador MPPT es capaz de hacer trabajar al sistema en el MPP.

Capítulo 8. Desarrollo y verificación de los bloques IP

8.1. Introducción

En este capítulo se presenta la placa de desarrollo que se va a utilizar para realizar la verificación de los bloques IP (*Intellectual Property*), así como sus características principales. También se introduce la metodología de codiseño *hardware/software* que ofrece MATLAB/Simulink para la generación y verificación de bloques IP.

Para cada uno de los bloques IP generados, se indica la interfaz de entrada/salida escogida y la frecuencia de funcionamiento. Además, se exponen los resultados de realizar una conversión a punto fijo del diseño, en caso de haber sido necesario, y se presentan los resultados de verificación. Para el caso del algoritmo MMTES, se realiza una comparación de los tiempos de ejecución conseguidos con diferentes variantes de bloques IP.

8.2. Placa de desarrollo

La placa de desarrollo escogida en este caso es la ZedBoard (Figura 62) [98]. Esta es una placa de desarrollo que integra un dispositivo SoC FPGA Zynq-7020 de Xilinx. La ZedBoard es una placa que permite desarrollar diseños basados en Linux, Android, Windows o RTOS. Cuenta con diferentes tipos de conectores que facilitan el acceso del usuario al sistema de procesamiento y la lógica programable. Es una placa de desarrollo pensada para ofrecer un prototipado rápido y facilitar la realización de pruebas de concepto. Algunas de las aplicaciones para las que está orientada esta placa son procesamiento de vídeo, control de motores, aceleración de *software* o desarrollo en Linux/Android/RTOS.

ZedBoard está soportada por MATLAB/Simulink para el codiseño *hardware*/*software*. Para preparar la placa para este flujo de trabajo, se sigue un ejemplo desarrollado por MathWorks para el kit de evaluación ZC702 de Xilinx [99]. Una guía paso a paso de la configuración del sistema operativo necesario y de la ZedBoard en base a este ejemplo se presenta en el apartado 1 del anexo de este TFM.

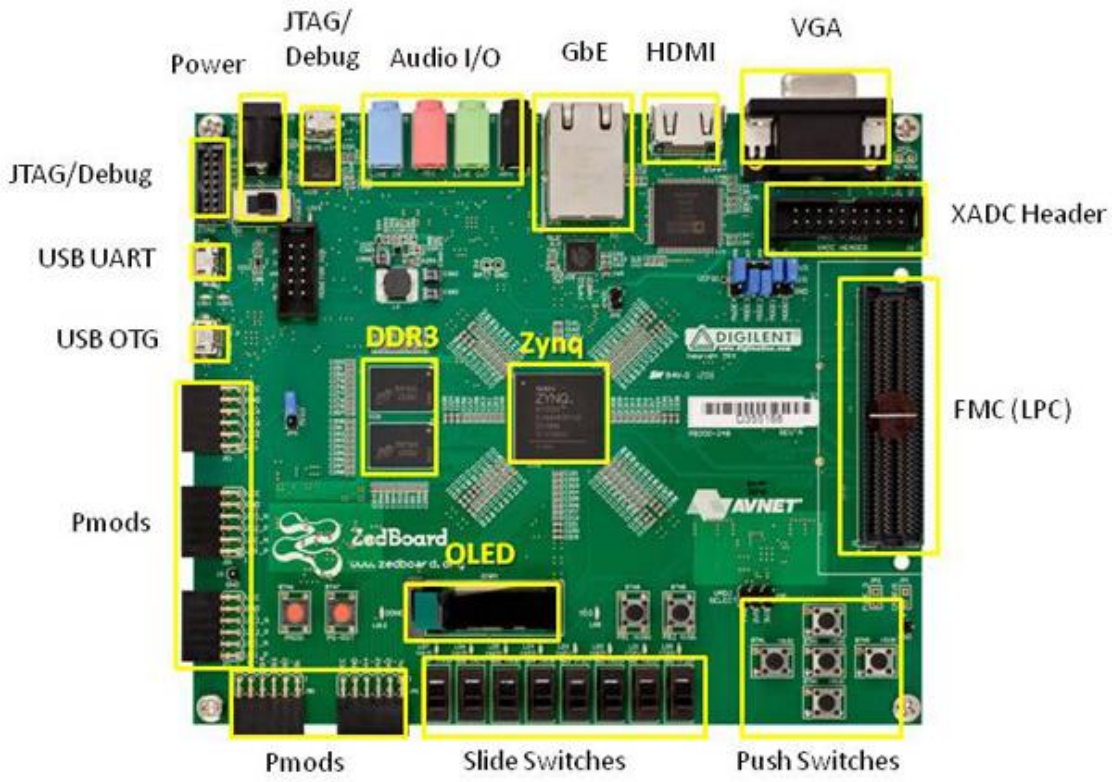


Figura 62. Overlay view de la ZedBoard [98].

8.3. Codiseño hardware/software

En este flujo de diseño se parte de un modelo en Simulink en el que está implementado el algoritmo para el que se crea un bloque IP. Mediante la herramienta HDL *Workflow Advisor* se inicia el proceso de creación del bloque IP. En esta herramienta se selecciona la placa de desarrollo, la interfaz de entrada y salida que se desea usar, el modo de funcionamiento del IP y la frecuencia de reloj. Estos bloques IP están diseñados para conectarse con el procesador empotrado a través de una interfaz AXI (Figura 63). En esta comunicación el procesador cumple el rol de maestro y el bloque IP el rol de esclavo.

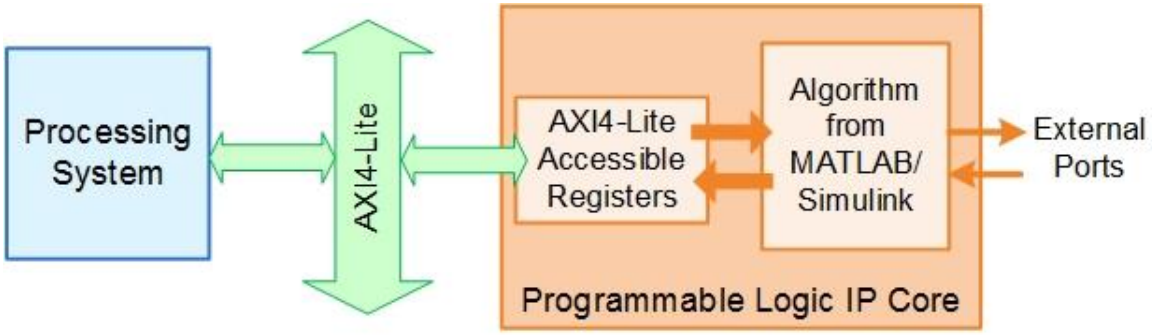


Figura 63. Comunicación entre el bloque IP y el procesador empotrado [100].

A la hora de generar el bloque IP, este se puede configurar con dos modos de funcionamiento diferenciados: el modo bloqueante (Figura 64) y el modo no bloqueante (Figura 65). En el modo bloqueante el procesador manda datos a la FPGA e interrumpe su funcionamiento hasta que esta termine de procesar los datos recibidos. Por otro lado, en el modo no bloqueante, tras transmitir datos hacia la FPGA, el procesador no permanece a la espera de que esta termine de procesarlos. En este caso, todos los bloques IP se configuran en modo bloqueante para su posterior verificación en Simulink.

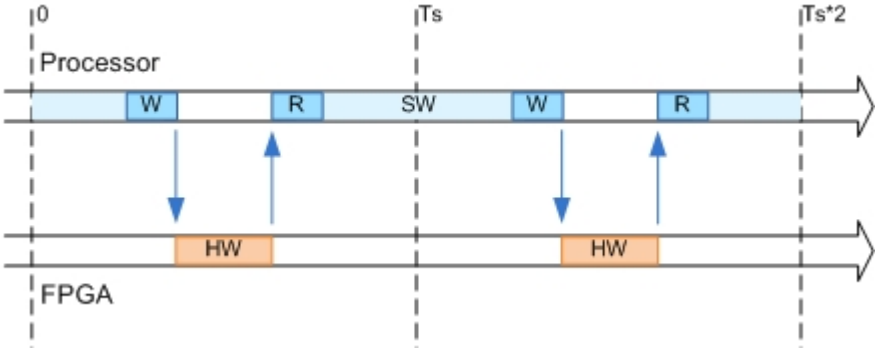


Figura 64. Modo de funcionamiento bloqueante [101].

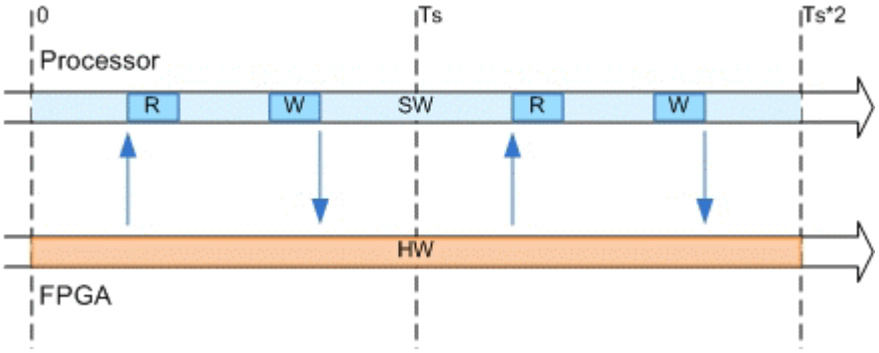


Figura 65. Modo de funcionamiento no bloqueante [101].

Una vez que se crea el bloque IP, se genera de forma automática un nuevo modelo en Simulink llamado *software interface* en el que se cambia el algoritmo, para el cual se ha generado el bloque IP, por unos bloques de escritura y lectura de interfaces AXI. También

se genera un proyecto para Vivado Design Suite que contiene la plataforma *hardware* con el IP generado. A partir de este proyecto se genera un *bitstream* y se programa la FPGA.

Para verificar el funcionamiento de los bloques IP, desde Simulink se genera código C a partir del modelo *software interface.* Este código contiene *drivers* generados a partir de los bloques de escritura y lectura de interfaces AXI que permiten controlar, comunicar y recibir datos de los bloques IP creados. El código generado se compila, se crea un ejecutable preparado para correr sobre Linux en el procesador ARM de la ZedBoard y se carga este en la placa. Tras esto, se realiza una simulación en modo externo desde Simulink (Figura 66). Esta simulación permite conectar el ordenador con la ZedBoard a través del cable Ethernet, lanzar el ejecutable cargado con anterioridad y monitorizar su funcionamiento. De esta manera se puede observar los resultados obtenidos de los bloques IP y verificarlos.

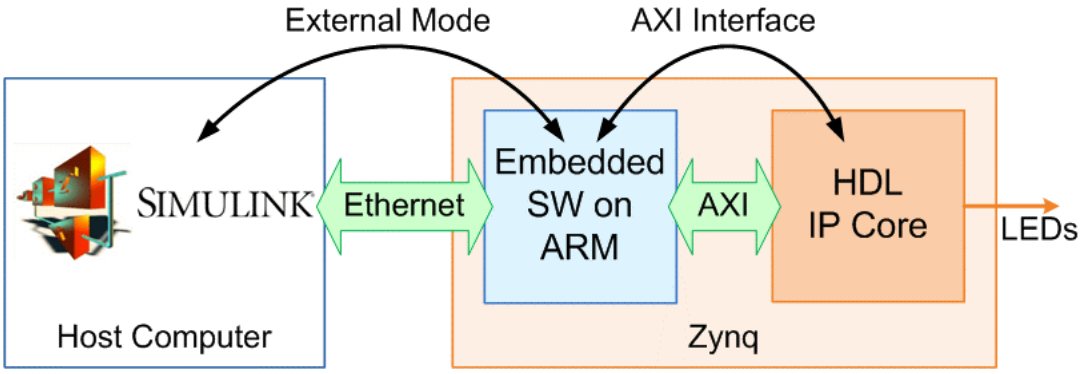


Figura 66. Simulación en modo externo desde Simulink [99].

En el apartado 2 del anexo se utiliza como ejemplo la generación del bloque IP para el controlador MPPT como guía paso a paso de la generación y verificación de bloques IP mediante el codiseño *hardware/software* desde Simulink.

8.4. MPPT

Para el bloque IP que contiene al algoritmo MPPT de conductancia incremental se escoge como interfaz de entrada y salida una interfaz AXI-Lite. La señal de reloj escogida para el bloque es de 100 MHz.

8.4.1. Conversión a punto fijo

Para poder generar un bloque IP a partir de la *Función MPPT_Inc* hay que convertir el código a punto fijo. Para ello se utiliza la aplicación *Fixed-Point Converter* proporcionada

por el *toolbox Fixed-Point Designer*. Esta es una aplicación creada para facilitar la conversión de código de MATLAB en punto flotante a código de MATLAB en punto fijo. Para utilizar esta aplicación, se crea un *testbench* en el que se llama a la función que se desea convertir a punto fijo. En base a esta llamada, la aplicación es capaz de determinar el rango de valores de entrada y salida, así como el rango de valores de las variables internas de la función y propone tipos de datos en consecuencia. Los tipos de datos propuestos pueden ser modificados por el usuario en caso de que no satisfagan sus necesidades. Una vez todos los tipos de datos estén correctos, se genera el código en punto fijo y se comprueba su error con respecto a la función en punto flotante. Para el caso de la *Función MPPT_Inc*, se resumen los resultados de la conversión en la Tabla 11.

Tabla 11. Resultados de conversión a punto fijo de la función MPPT_Inc.

Entrada/salida	Tipo de dato propuesto	Orden del error
Vpv	25 bits con 7 bits de parte entera sin signo	10^{-6}
Ipv	25 bits con 5 bits de parte entera sin signo	10^{-6}
D	32 bits con 1 bit de parte entera sin signo	10^{-8}

8.4.2. Verificación del bloque IP generado

Para verificar el funcionamiento del bloque IP, se compara el valor del ciclo de trabajo (*duty cycle*) calculado por este, con el ciclo de trabajo calculado por el algoritmo MPPT al simularlo en Simulink. Como valores de entrada para el cálculo del ciclo de trabajo en ambos casos se utilizan valores de tensión y corriente del *array* fotovoltaico obtenidos durante la simulación MIL del caso 1 (Tabla 12) de estudio con carga resistiva.

Tabla 12. Caso de estudio 1.

	$\boldsymbol{G}\ (\boldsymbol{W/m^2})$	$\boldsymbol{P}$
Caso 1	$\begin{bmatrix} 400 & 500 & 600 \\ 500 & 600 & 700 \\ 600 & 700 & 800 \end{bmatrix}$	$\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$

Una comparativa de ambos ciclo de trabajo se ilustra en la Figura 67. En la parte superior se muestra el ciclo de trabajo calculado por el bloque IP y por el MPPT simulado en Simulink. En la parte inferior, en rojo, se ilustra la diferencia entre el ciclo de trabajo calculado por el MPPT simulado en Simulink y el calculado por el bloque IP. Como se

observa, la diferencia es nula, por lo tanto, se verifica el correcto funcionamiento del bloque IP.

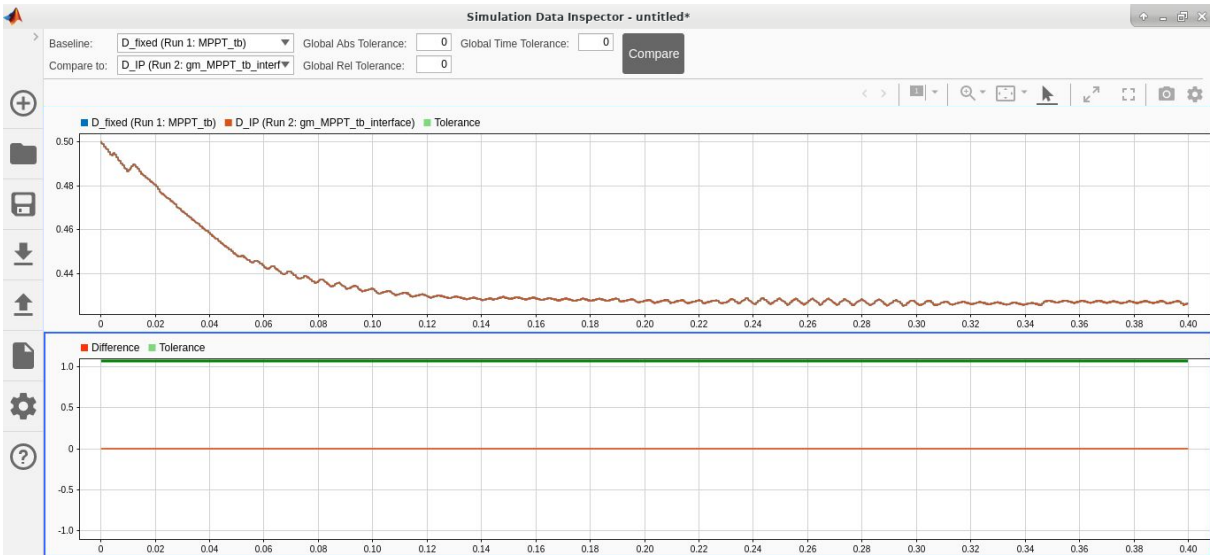


Figura 67. Comparativa del ciclo de trabajo calculado por el bloque IP y la simulación en Simulink.

8.5. Generador de PWM

Para el bloque IP del generador de PWM se utilizan interfaces de entrada/salida AXI-Lite. El reloj del bloque IP se establece a 100 MHz.

8.5.1. Verificación del bloque IP generado

Para verificar el funcionamiento del bloque IP se utilizan diferentes valores de entrada para el ciclo de trabajo que varían entre 0 y 0.9 a intervalos de 0.1. Cada valor de ciclo de trabajo se mantiene durante 100 µs. Esto debería generar un PWM de 10 KHz que en cada período aumenta su ciclo de trabajo en 0.1. En la Figura 68 se ilustra el PWM de salida generado por el bloque IP. Se puede observar cómo tiene la forma anteriormente descrita, por lo tanto, se verifica el funcionamiento del bloque.

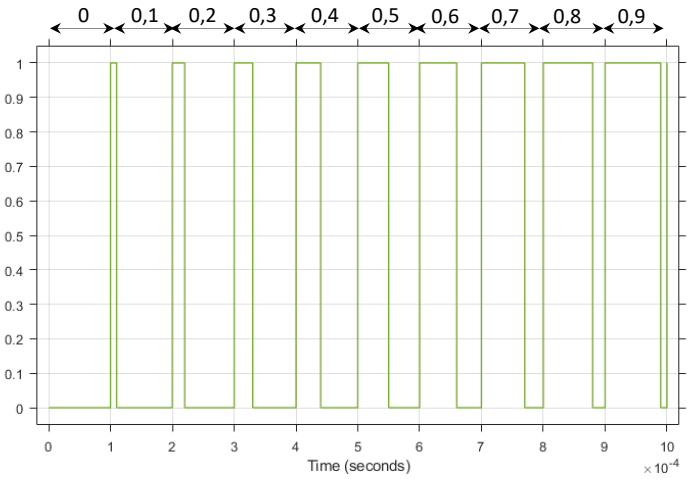


Figura 68. Verificación del IP PWM_Generator.

8.6. MMTES

Anteriormente se ha optado por realizar una implementación *hardware/software* del algoritmo MMTES. Para la implementación de la parte *hardware* se desarrollan 2 bloques IP, uno para la función *MMTESSort* y otro para implementar la función *PairForSwapping*. Dado que las funciones *MMTESSort* y *PairForSwapping* no han sido pensadas inicialmente con la idea de ser implementadas en *hardware*, se desarrollan dos variantes para su implementación.

La función *MMTESSort* se sustituye por la función *BitonicSort* y *PairForSwapping* por *PairForSwappingPll*. Estas son las funciones a partir de las cuales se crean los dos bloques IP de la implementación *hardware* del algoritmo MMTES. No obstante, con el fin de verificar que dichas funciones presentan una mejora frente a las funciones originales en una implementación *hardware*, se generan también los bloques IP correspondientes a las funciones originales.

El rendimiento de los bloques IP generados se mide mediante el IP System ILA [102] (*Integrated Logic Analyzer*) proporcionado en Vivado Design Suite. Este bloque IP es un analizador lógico con el que se pueden monitorizar las señales internas e interfaces de un diseño. Con ayuda del System ILA se puede medir el tiempo de ejecución de los bloques IP y el tiempo de ejecución, teniendo en cuenta la comunicación de los datos de entrada y salida del bloque. Con estas mediciones se puede determinar si las variantes desarrolladas de las funciones originales presentan una mejora o no.

8.6.1. MMTESSort

Para el bloque IP de la función MMTESSort se utilizan interfaces de entrada/salida AXI4 y una frecuencia de funcionamiento de 100 MHz.

8.6.1.1. Conversión a punto fijo

Al igual que en el caso del MPPT, para generar un bloque IP a partir de la función *MMTESSort*, hay que convertir su código en MATLAB de punto flotante a punto fijo. Para esto se utiliza nuevamente la aplicación *Fixed Point Converter*.

La función *MMTESSort* trabaja con valores de irradiancia enteros, por lo que, a la hora de realizar la conversión a punto fijo, el error con respecto a la implementación en punto flotante es nulo. Al realizar la conversión a punto fijo se consigue disminuir la cantidad de bits necesarios para representar los valores de las variables internas del algoritmo, así como los valores de entrada y salida.

De la misma manera, se consigue disminuir el número de bits necesarios para realizar operaciones. Toda esta reducción permite conseguir un consumo de recursos menor de la implementación *hardware* del algoritmo.

8.6.1.2. Verificación del bloque IP generado

Como valores de entrada del bloque IP se utilizan las matrices de irradiancia de los 3 casos de estudio y su matriz de posición correspondiente. Para verificar que la ordenación de las matrices realizada por el bloque IP es la correcta, se comparan con las matrices obtenidas al ejecutar la función *MMTESSort* en Simulink. Esta comparación se realiza restándole a las matrices obtenidas del bloque IP las matrices obtenidas al ejecutar la función en Simulink. Si el resultado de la resta es una matriz nula, las matrices eran idénticas y por lo tanto se verifica el funcionamiento del bloque IP. En este caso al realizar la resta se obtienen matrices nulas, por lo tanto, se verifica el correcto funcionamiento del bloque.

Los resultados del tiempo de ejecución del bloque IP para el algoritmo MMTESSort se resumen en la Tabla 13.

Tabla 13. Tiempos de ejecución IP MMTESSort.

	MMTESSort
Tiempo de ejecución	680 ns
Tiempo con comunicaciones	19240 ns

8.6.2. PairForSwapping

El bloque IP de la función *PairForSwapping* se genera con interfaces de entrada/salida AXI-Lite. Al igual que los bloques IP previos, el reloj se establece a 100 MHz.

8.6.2.1. Conversión a punto fijo

La conversión a punto fijo se realiza de la misma forma que el IP de la función *MMTESSort*. Al igual que en el caso anterior, la función *PairForSwapping* trabaja únicamente con valores enteros, por lo que, el error con respecto a la implementación en punto flotante es nulo.

8.6.2.2. Verificación del bloque IP generado

La verificación del bloque IP se realiza nuevamente comparando los valores de salida obtenidos del bloque IP ejecutándose en la ZedBoard con los valores obtenidos al ejecutar la función *PairForSwapping* en Simulink. En este caso los valores son iguales y por lo tanto se verifica el funcionamiento del bloque.

Los resultados del tiempo de ejecución del bloque IP para la función *PairForSwapping* se resumen en la Tabla 14.

Tabla 14. Tiempo de ejecución IP PairForSwapping.

	PairForSwapping
Tiempo de ejecución	660 ns
Tiempo con comunicaciones	10100 ns

8.6.3. Bitonic Sort

Se escoge una interfaz AXI-Lite para la entrada/salida de datos del bloque IP. La frecuencia del reloj del bloque se establece a 100 MHz. Dado que la función *BitonicSort* se ha desarrollado en Simulink utilizando bloques del *toolbox* HDL Coder, no es necesario realizar la conversión a punto fijo antes de generar el bloque IP.

8.6.3.1. Verificación del bloque IP generado

Para verificar este bloque IP se comparan los resultados obtenidos de la ejecución del bloque IP en la ZedBoard con la ejecución de la función *BitonicSort* en Simulink. Para generar datos de entrada compatibles con la función *BitonicSort* hay que hacer uso de la función *AdaptInputData*. De igual manera, para interpretar correctamente los datos de salida de la función *BitonicSort* hay que utilizar la función *AdaptOutputData*.

Los datos obtenidos del bloque IP y de la ejecución de la función *BitonicSort* en Simulink son idénticos. Los tiempos de ejecución del bloque IP en la ZedBoard durante el proceso de verificación se presentan en la Tabla 15.

Tabla 15. Tiempo de ejecución IP BitonicSort.

	BitonicSort
Tiempo de ejecución	580 ns
Tiempo con comunicaciones	8900 ns

8.6.4. PairForSwapping paralelo

Como en el bloque IP anterior, se escoge una interfaz AXI-Lite de entrada/salida y una frecuencia de reloj de 100 MHz. La función *PairForSwappingPll* también está implementada en Simulink utilizando bloques del *toolbox* HDL Coder, por lo tanto, no es necesario realizar su conversión a punto fijo.

8.6.4.1. Verificación del bloque IP generado

La verificación se realiza de forma análoga a la del resto de bloques IP. Se compara el resultado obtenido de ejecutar el bloque IP en la ZedBoard con el resultado obtenido de ejecutar la función *PairForSwappingPll* en Simulink. Para este bloque IP hay que tener en cuenta que es necesario utilizar la función *PfsPllInData* para generar datos compatibles con la entrada del bloque IP y *PfsPllOutData* para interpretar los datos de salida.

Los resultados obtenidos de la ejecución del bloque IP y la función en Simulink son iguales, por lo tanto, se verifica el funcionamiento del bloque IP. Las mediciones de tiempo de ejecución durante la verificación del bloque se exponen en la Tabla 16.

Tabla 16. Tiempo de ejecución IP PairForSwappingPll.

	PairForSwappingPll
Tiempo de ejecución	560 ns
Tiempo con comunicaciones	3300 ns

8.6.5. Comparación de resultados

Para el caso de los bloques *MMTESSort* y *BitonicSort*, una comparación de los tiempos de ejecución se presenta en la Tabla 17. Como se puede observar, el tiempo de ejecución en *hardware* de la función *BitonicSort* es 100 ns más rápido que el de la función *MMTESSort*. Además, hay que tener en cuenta que la función *BitonicSort* está ordenando un total de 16 valores mientras que la función *MMTESSort* ordena únicamente 9 valores.

Por lo tanto, el tiempo de ejecución en *hardware* del bloque *BitonicSort* es menor que el del bloque *MMTESSort*, aun clasificando un mayor número de datos. Si se tiene en cuenta el *data movement*, es decir, la comunicación de los datos de entrada y salida de los bloques IP, la diferencia se hace más notoria. El IP *BitonicSort* es 10.34 µs más rápido que el IP *MMTESSort*. Esta mejora se debe al empaquetado y desempaquetado de los datos de entrada y salida realizado por las funciones *AdaptInputData* y *AdaptOutputData*. Así se consigue optimizar la transmisión de datos disminuyendo notablemente el número de transmisiones necesarias. Dado que se hacen menos transmisiones, se consume menos tiempo.

Tabla 17. Comparación tiempos de ejecución MMTESSort y BitonicSort.

	MMTESSort	BitonicSort	Diferencia
Tiempo de ejecución	680 ns	580 ns	100 ns
Tiempo con comunicaciones	19240 ns	8900 ns	10340 ns

En la Figura 69 se presentan los tiempos de ejecución medidos en porcentaje. El tiempo de ejecución del *BitonicSort* es un 14.71% menor que el del *MMTESSort* sin tener en cuenta el *data movement.* Si se tiene en cuenta, la reducción es de un 53.74%.

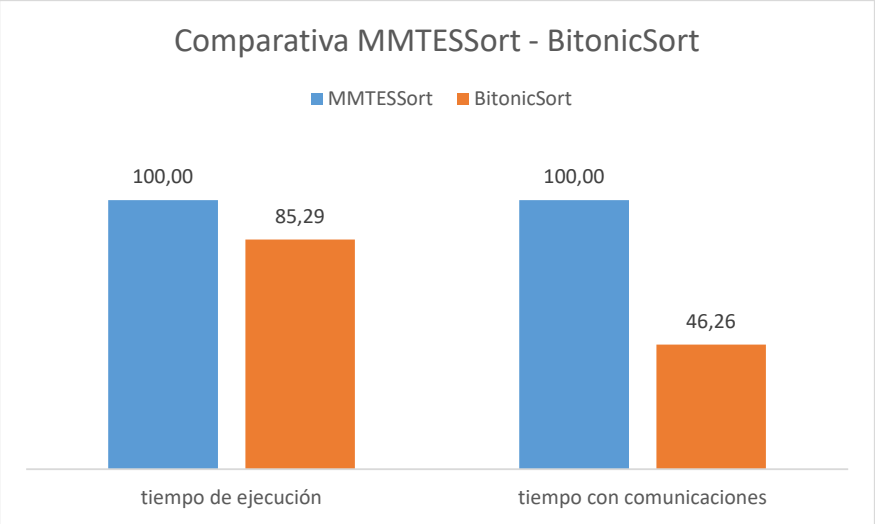


Figura 69. Comparativa MMTESSort y BitonicSort.

En el caso de los bloques *PairForSwapping* y *PairForSwappingPll*, la comparación de los tiempos de ejecución se presenta en la Tabla 18. Como se puede observar, el bloque *PairForSwappingPll* es 100 ns más rápido que el *PairForSwapping*. Por lo tanto, se puede decir que la modificación para mejorar el tiempo de ejecución en *hardware* de la función *PairForSwapping* ha surtido efecto. Al igual que en el caso anterior, la diferencia se incrementa al tener en cuenta el *data movement*. La optimización de las comunicaciones en el caso del *PairForSwappingPll* se consigue utilizando las funciones *PfsPllInData* y *PfsPllOutData*. Gracias a esta optimización de las comunicaciones se consigue rebajar en 6.8 µs el tiempo de ejecución con respecto al IP *PairForSwapping*.

Tabla 18. Comparación tiempos de ejecución PairForSwapping y PairForSwappingPll.

	PairForSwapping	**PairForSwappingPll**	**Diferencia**
Tiempo de ejecución	660 ns	560 ns	100 ns
Tiempo con comunicaciones	10100 ns	3300 ns	6800 ns

En la Figura 70 se presenta una gráfica que ilustra los tiempos de ejecución de los bloques IP en porcentaje. El tiempo de ejecución del *PairForSwappingPll* es un 15.15% menor que el del *PairForSwapping* sin tener en cuenta el *data movement.* Si se tiene en cuenta, la reducción es de un 63.33%.

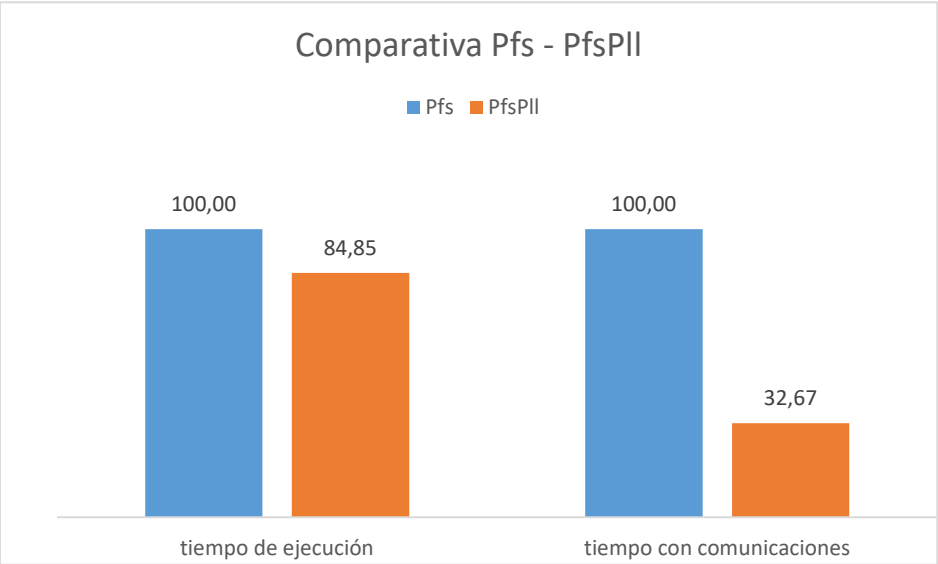


Figura 70. Comparativa PairForSwapping y PairForSwappingPll.

8.7. Conclusiones

El codiseño *hardware/software* ofrecido por MATLAB/Simulink permite generar y verificar bloques IP para su implementación en FPGA. El flujo de diseño es guiado en todo momento por la aplicación *HDL Workflow Advisor* lo que facilita y acelera en gran medida el proceso de generación y verificación de los bloques. La verificación de todos los bloques IP generados ha sido satisfactoria. Además, se ha comprobado que las mejoras propuestas en el apartado 5.4 para la implementación *hardware* de las funciones *MMTESSort* y *PairForSwapping* suponen, en efecto, una mejora que llega a disminuir el tiempo de ejecución hasta un 63.33 % en el caso de la función *PairForSwapping*.

Capítulo 9. Implementación y verificación

9.1. Introducción

Tras la generación y verificación realizada de los bloques IP, el siguiente paso es la creación de un sistema que integre todos estos bloques para su implementación en la placa de desarrollo. En este capítulo se muestran los resultados obtenidos al realizar una implementación para la tarjeta de prototipado ZedBoard.

Para validar el sistema creado se utiliza MATLAB/Simulink. Este programa permite simular el funcionamiento del sistema generado en la ZedBoard en conjunto con el modelo del sistema a controlar en Simulink.

9.2. Implementación del sistema final

El primer paso para crear un sistema que integre todos los bloques IP desarrollados es generar un nuevo proyecto en Vivado Design Suite. En este nuevo proyecto se crea un diseño de bloques con el *IP Integrator* en el que se integran los bloques IP generados junto con el PS (*Processing System*) del Zynq montado en la ZedBoard y demás bloques necesarios para la interconexión. Para poder hacer esto, hay que añadir al *IP Repository* del proyecto los bloques IP desarrollados. Esto permite acceder a ellos desde el proyecto y generar el diagrama de bloques.

En la Figura 71 se ilustra el diagrama de bloques generado. En azul está remarcado el PS del Zynq encargado de ejercer como maestro en las transmisiones AXI y ejecutar la partición *software* del diseño, en naranja está remarcado el bloque IP del algoritmo MPPT

y en verde están remarcados los bloques IP *BitonicSort* y *PairForSwappingPll* que conforman la partición *hardware* del algoritmo MMTES. Cabe destacar que aun habiendo sido verificado el funcionamiento del IP *PWM_Generator*, este no se ha incluido en el sistema final.

La razón por la que no se ha incluido es que la finalidad del IP *PWM_Generator* es generar una PWM en uno de los pines de salida de la ZedBoard para controlar el *Buck Converter*. Sin embargo, el funcionamiento del *Buck Converter* se simula en Simulink, por lo que no es posible controlarlo mediante la PWM que generaría el bloque IP en el pin de salida de la ZedBoard. Es por esto por lo que se decide simular el funcionamiento del *PWM_Generator* directamente en Simulink.

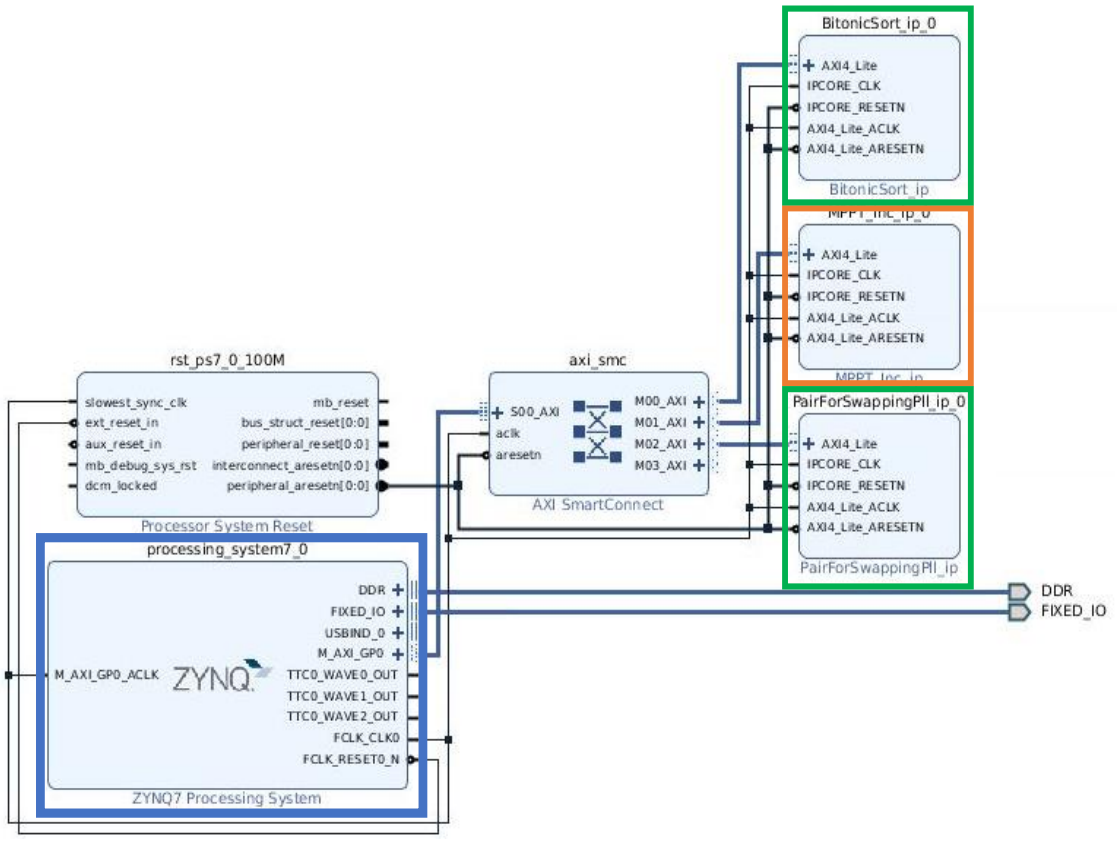


Figura 71. Diagrama de bloques de la plataforma hardware generada.

Una vez se crea el diagrama de bloques, hay que asignar las direcciones de memoria para las interfaces AXI correspondientes a cada uno de los bloques IP desarrollados. Esto es necesario hacerlo para poder acceder a los diferentes bloques IP desde Linux. Las direcciones se asignan desde la pestaña *Address Editor* del *IP Integrator* quedando como se muestran en la Tabla 19.

Tabla 19. Direcciones de memoria de los bloques IP desarrollados.

IP	*Offset Address*	*Range*
BitonicSort	0x400D0000	4K
MPPT_Inc	0x400E0000	4K
PairForSwappingPll	0x400F0000	4K

Tras la asignación de direcciones se valida el diseño y se genera el HDL *Wrapper*. La generación del HDL *Wrapper* es necesaria para poder iniciar el proceso de síntesis e implementación del diseño. Tanto la síntesis como la implementación se realizan para un período de reloj de 10 ns (100 MHz).

En la Figura 72 se presenta la utilización de recursos del sistema diseñado. Como se puede apreciar, la utilización de recursos es relativamente baja para este sistema. El recurso más utilizado son los LUTs llegando a utilizar el 11% del total disponible. En la Tabla 20 se presenta un desglose de la utilización de recursos de los bloques IP desarrollados.

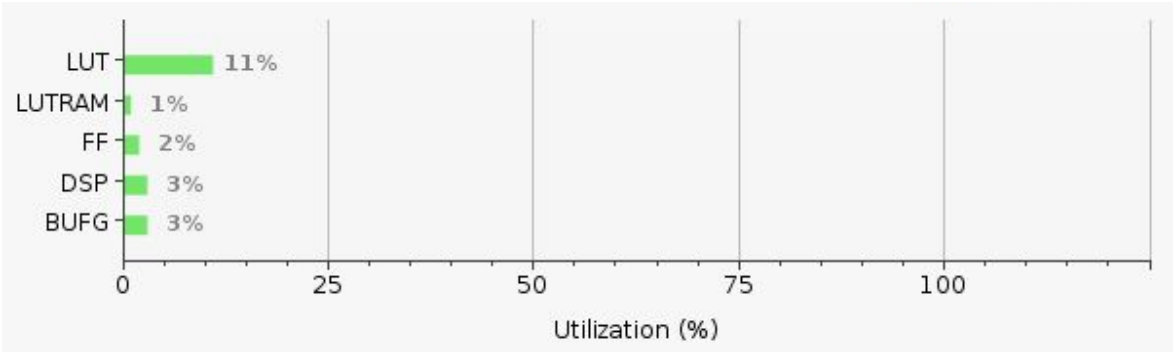


Figura 72. Utilización de recursos del sistema.

Tabla 20. Utilización de recursos por bloque IP.

	LUT	FF	DSP	LUTRAM	BUFG
BitonicSort	2975 (5.59%)	606 (0.57%)	0	0	0
MPPT_Inc	1743 (3.31%)	244 (0.25%)	6 (2.73%)	0	0
PairForSwappingPll	733 (1.38%)	496 (0.47%)	0	0	0

En la Figura 73 se presentan los resultados del análisis temporal del diseño. Se puede observar cómo este tiene un *slack* positivo de 0.203 ns a una frecuencia de funcionamiento de 100 MHz.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.203 ns	Worst Hold Slack (WHS):	0.037 ns	Worst Pulse Width Slack (WPWS):	3.000 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:	7641	Total Number of Endpoints:	7601	Total Number of Endpoints:	2961

All user specified timing constraints are met.

Figura 73. Resultados de temporización del sistema.

En cuanto a la potencia consumida, en la Figura 74 se presenta el consumo total de potencia de la plataforma. Como se puede observar, la potencia total consumida es de 1.716 W. El 89.1% de la potencia total es consumida por el PS del Zynq.

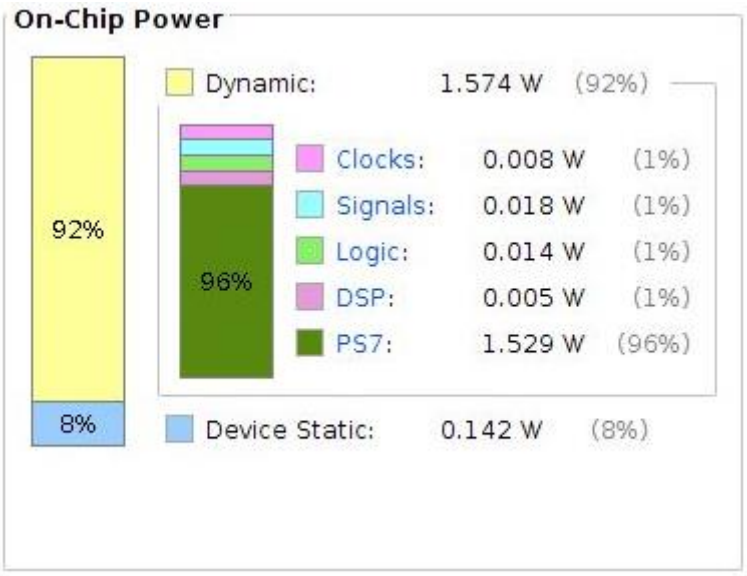


Figura 74. Consumición de potencia del sistema diseñado.

El consumo de potencia de los bloques IP desarrollados se puede considerar casi despreciable en comparación con el consumo del PS. En la Tabla 21 se presenta el consumo de potencia de los diferentes bloques IP.

Tabla 21. Consumición de potencia de los bloques IP.

	Potencia
BitonicSort	2 mW (< 1 %)
MPPT_Inc	34 mW (2 %)
PairForSwappingPll	2 mW (< 1 %)

Por último, en la Figura 75 se puede observar el *layout* de la plataforma con los recursos utilizados. En esta figura están remarcados en verde los recursos utilizados por el IP BitonicSort, en rojo los recursos usados por el MPPT y en azul los recursos utilizados por el *PairForSwappingPll*.

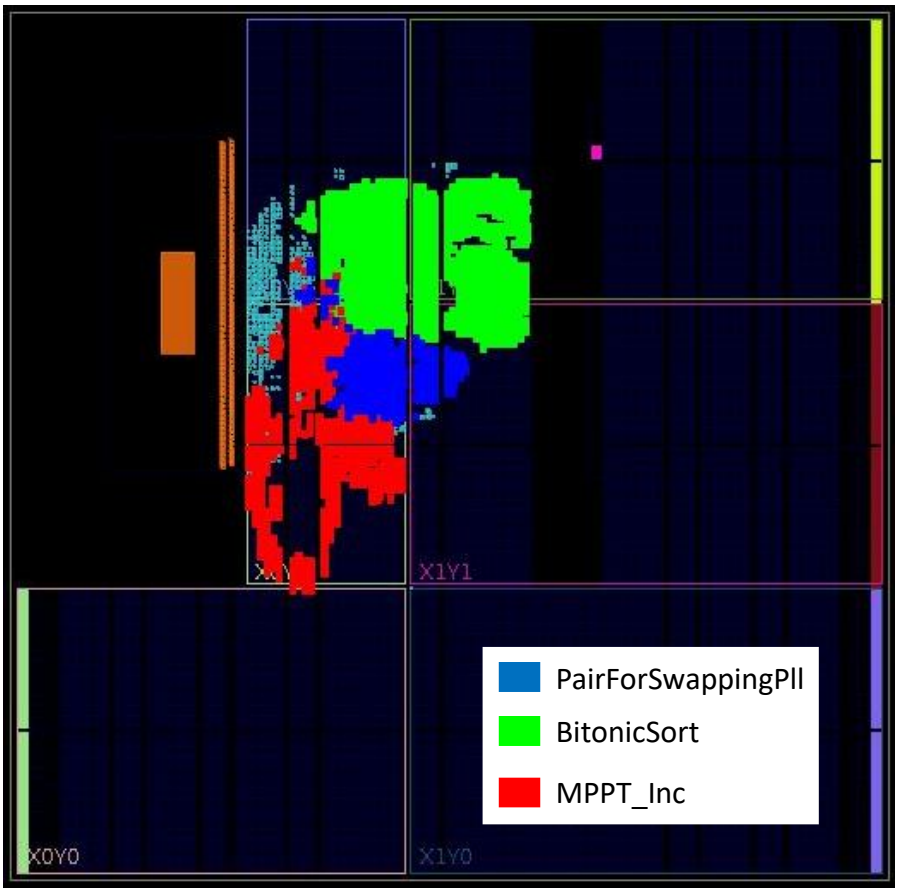


Figura 75. Layout del sistema.

Una vez se ha terminado la síntesis e implementación y se ha comprobado que los resultados son satisfactorios, el siguiente paso es la generación del *bitstream.* El *bitstream* permite programar el PL (*Programmable Logic*) del Zynq con el sistema diseñado. Cargada la plataforma *hardware* diseñada en la placa, solo resta generar y programar el *software* en el PS para tener el sistema completo funcionando en la ZedBoard.

El *software* para el sistema diseñado se genera a partir de un modelo en Simulink con ayuda del *toolbox Embedded Coder*. A este modelo se le denomina **ZedBoardModel**. Se toma como referencia para diseñar ZedBoardModel el modelo *software interface* que genera la aplicación *HDL Workflow Advisor* cuando crea un bloque IP.

En la Figura 76 se ilustra el modelo ZedBoardModel a partir del cual se genera el *software* del sistema diseñado. En este modelo se encuentran implementados el controlador MPPT y el algoritmo MMTES. Todos los algoritmos implementados están divididos en dos partes diferenciadas: la parte *software* y la parte *hardware*. La parte

software implementa la partición de los algoritmos que se ejecutan en el PS del Zynq, es decir, en el procesador ARM.

La parte *hardware* implementa la partición de los algoritmos que se implementan en el PL del Zynq, es decir, en la FPGA. Para el caso del controlador MPPT, se ha diseñado una implementación puramente *hardware* por medio del IP *MPPT_Inc*. Sin embargo, la transmisión de los datos de entrada al IP y la lectura de los datos de salida es gestionada por el procesador ARM.

El algoritmo MMTES se implementa siguiendo una estrategia *hardware/software*. La parte *hardware* consta de los IPs *BitonicSort* y *PairForSwappingPll* y la parte *software* del resto del algoritmo MMTES, la función *auxCtrl* y las funciones de adaptación de los datos de entrada y salida para los bloques IP.

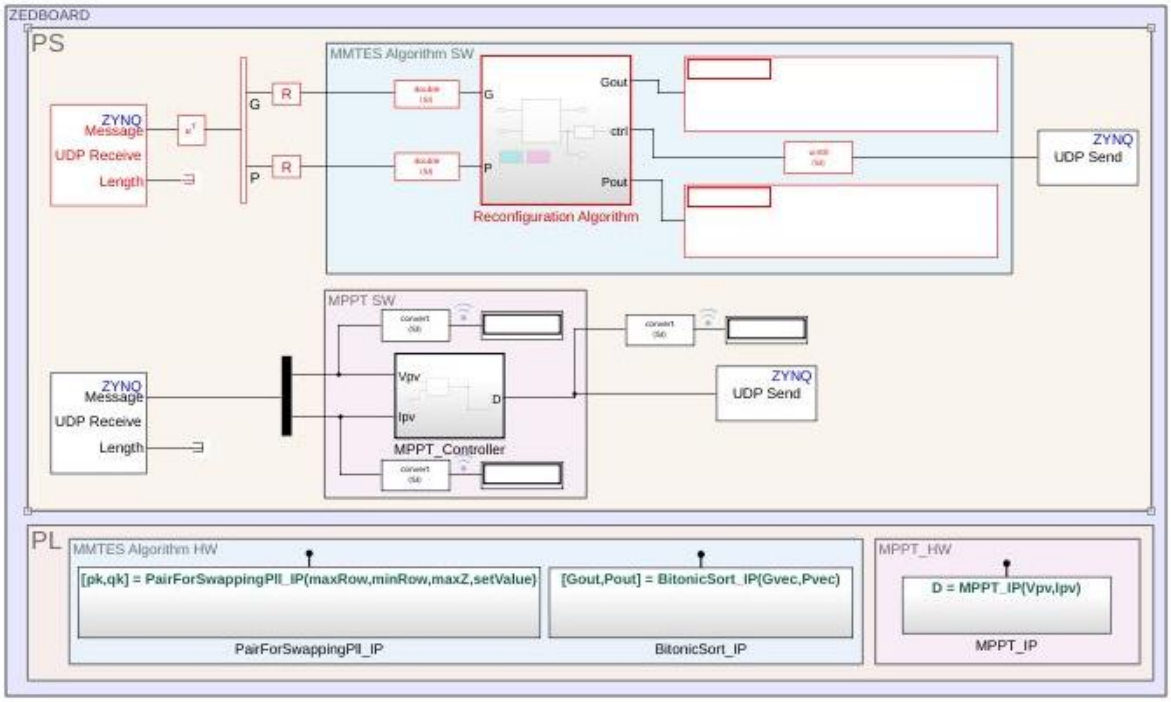


Figura 76. Modelo ZedBoardModel.

Para realizar la implementación de la parte *hardware* del controlador MPPT y el algoritmo MMTES en el modelo de Simulink, se utilizan los bloques **AXI4-Interface Write** y **AXI4-Interface Read**. Estos bloques escriben o leen datos de grupos de registros mapeados en memoria. De esta manera, a través del *central interconnect* del PS, se consigue una comunicación mapeada en memoria con los bloques IP presentes en el PL.

A estos bloques de lectura y escritura AXI hay que indicarles el *driver* que van a utilizar y el *offset* del registro*,* con respecto a la dirección base del *driver,* al que van a escribir o que van a leer. El *driver* es lo que le permite al sistema operativo poder interactuar con los bloques IP del PL. Este *driver* viene incluido en la imagen de Linux para la ZedBoard que se proporciona en el Embedded Coder Support Package for Xilinx Zynq Platform. El *driver* se denomina *mwipcore*, y por defecto viene creada una instancia de este, denominada *mwipcore0*, con dirección base **0x400D0000**.

Dado que en este caso se cuenta con más de un bloque IP en el PL con el que el procesador tiene que comunicarse, hace falta crear más instancias de este *driver*. Los pasos para crear una instancia de este *driver* se explican en el apartado 3 del anexo. Se crean 2 instancias más con nombres *mwipcore1* y *mwipcore2* correspondientes al IP *MPPT_Inc* y *PairForSwappingPll* respectivamente. Las direcciones base de estos *drivers* se asignan según las direcciones base de los bloques IP mostradas en la Tabla 19. La instancia *mwipcore0* se asigna al IP *BitonicSort*. Con respecto a los *offset* de los diferentes registros de los bloques IP, estos se indican en los informes generados por la aplicación **HDL Workflow Advisor** al crear los diferentes bloques IP.

La implementación de la parte *software* consta de la partición *software* del algoritmo MMTES, la función *auxCtrl*, las funciones de empaquetamiento y desempaquetamiento de datos para los bloques IP y las comunicaciones.

La partición *software* del algoritmo MMTES y la función *auxCtrl* están desarrolladas en MATLAB y se utilizan bloques MATLAB Function para implementarlas en Simulink. Por otro lado, las funciones de empaquetamiento y desempaquetamiento de datos para los bloques IP están desarrolladas en Simulink, por lo que su implementación es directa en el modelo.

Finalmente, para las comunicaciones se utilizan los bloques UDP Receive y UDP Send que proporciona el Embedded Coder Support Package for Xilinx Zynq Platform. Estos bloques permiten recibir y enviar paquetes UDP a través de la conexión ethernet de la ZedBoard. La comunicación UDP es necesaria para enviar y recibir datos para la verificación del sistema. En el siguiente apartado se explica más en detalle el proceso de verificación.

Una vez está completo el modelo, se crea el programa ejecutable. Es importante que antes de generar el ejecutable, se active la opción *allow tasks to execute concurrently on target* en las opciones del *solver* en Simulink. Esto permite que las diferentes tareas que crea Simulink al generar el código puedan ejecutarse concurrentemente en el procesador. Tras activar esta opción, se genera código C, se compila y se crea el ejecutable para Linux. Este se carga en la ZedBoard y se lanza. Hecho esto, la implementación del sistema está completa.

9.3. Verificación

Para realizar la verificación del sistema implementado en la ZedBoard, se crea en Simulink un nuevo modelo basado en el que se ilustra en la Figura 77. A este modelo se le denomina SimscapeModel y contiene el modelado del sistema a controlar por la ZedBoard. En SimscapeModel se reemplaza el algoritmo MMTES, la función *auxCtrl,* y el controlador MPPT del modelo de la Figura 77, por bloques de envío y recepción de paquetes UDP (Figura 78). El color de los bloques UDP de la Figura 78, corresponde con el color de los bloques de la Figura 77 sustituidos.

Estos bloques de envío y recepción de paquetes UDP se utilizan para establecer un canal de comunicación entre SimscapeModel y la ZedBoard. Durante la simulación de SimscapeModel, se envían valores de tensión y corriente del *array* fotovoltaico, la matriz de irradiancias y la matriz de posición a la ZedBoard. En la ZedBoard estos datos se utilizan como valores de entrada para el controlador MPPT y el algoritmo MMTES. El ciclo de trabajo calculado por el MPPT y el vector de control calculado por el algoritmo MMTES se transmiten desde la ZedBoard nuevamente hacia el modelo SimscapeModel. De esta manera, al realizar la cosimulación del modelo SimscapeModel, se está llevando a cabo la verificación del sistema implementado en la ZedBoard. En la Figura 79 se presenta un diagrama de bloques que ilustra el flujo de datos a través de UDP entre SimscapeModel y la ZedBoard a la hora de realizar la verificación.

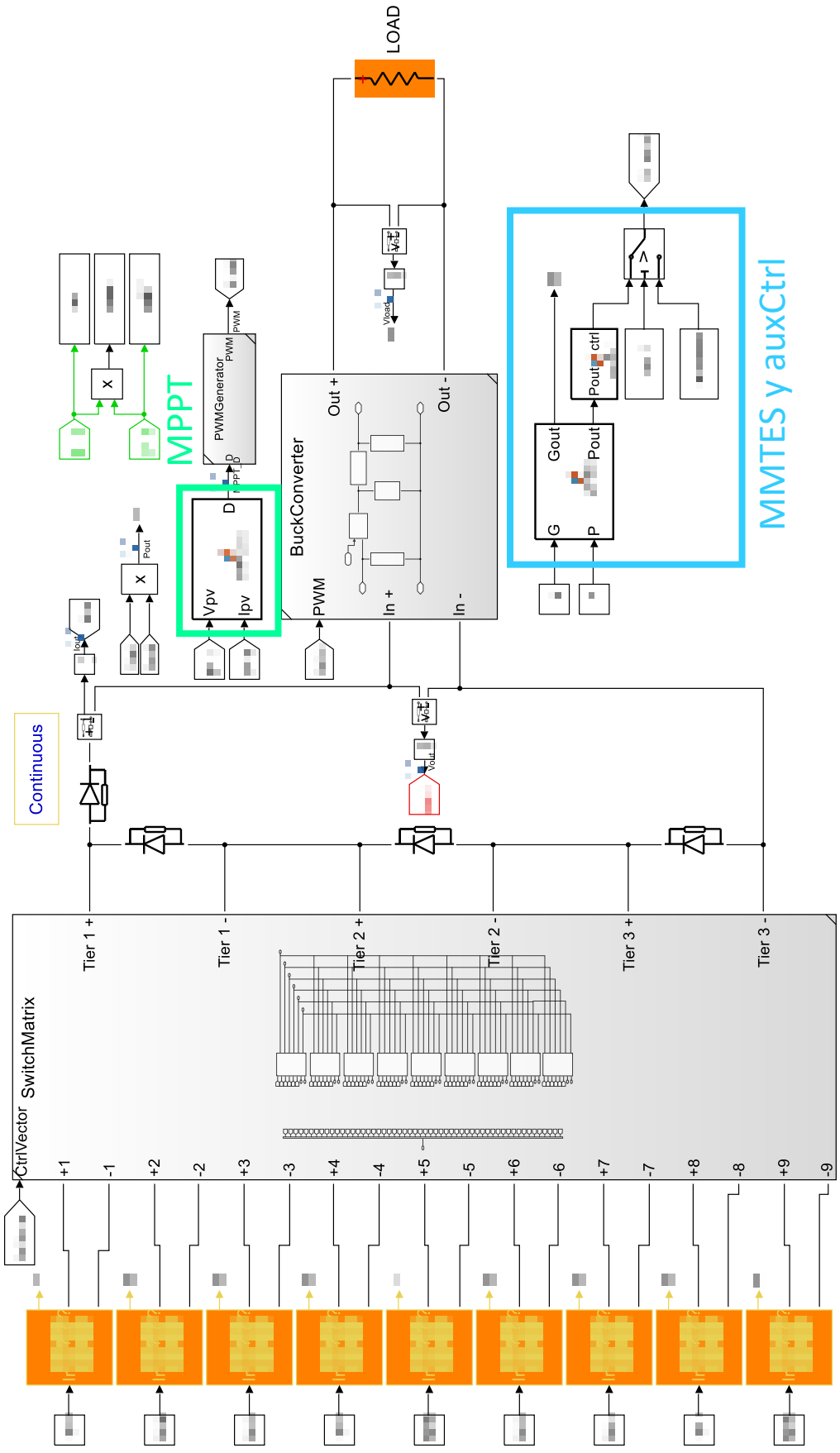


Figura 77. Modelo en el que se basa el modelo SimscapeModel.

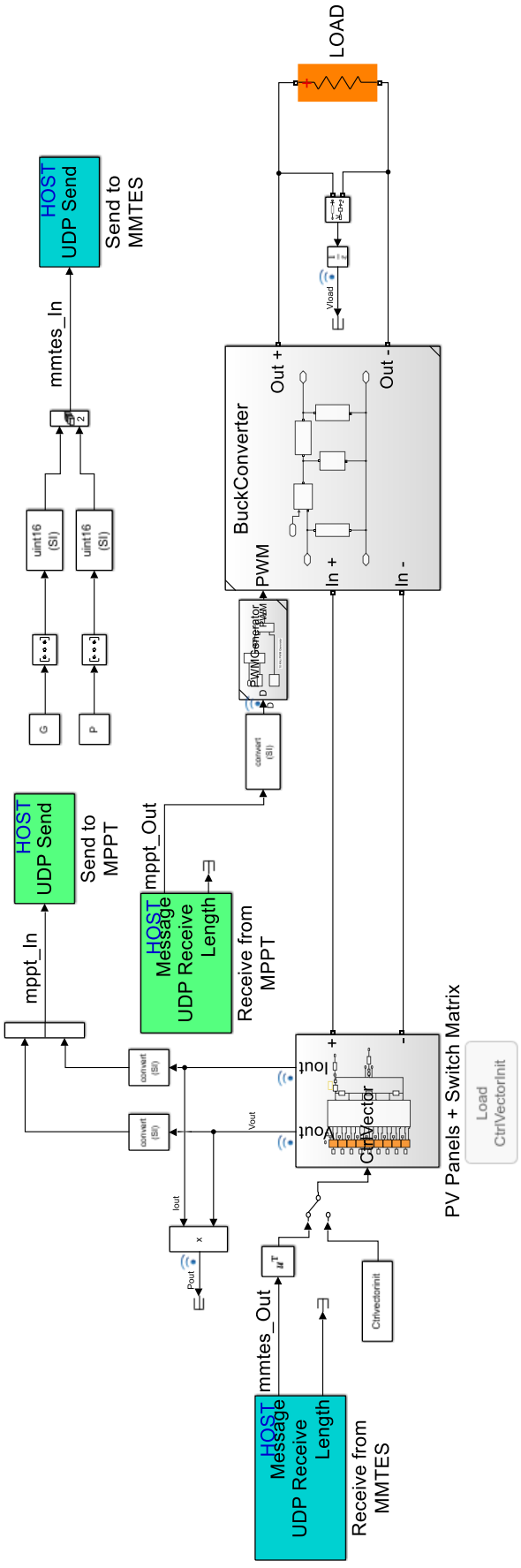


Figura 78. Modelo SimscapeModel.

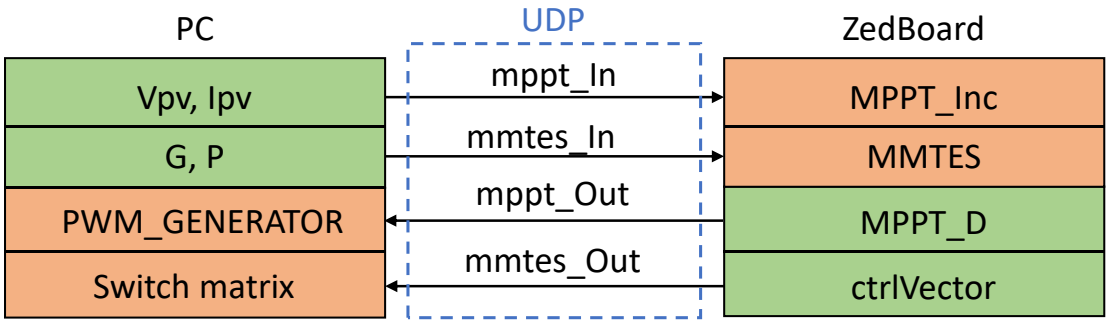


Figura 79. Flujo de datos entre SimscapeModel y la ZedBoard.

En la Figura 80 se observa el resultado de realizar la simulación de SimscapeModel en conjunto con el sistema implementado en la ZedBoard. Esta simulación corresponde al caso de estudio 1 con carga R_2. La Figura 80 está dividida en dos gráficas diferenciadas. En la gráfica superior se compara la potencia de salida obtenida durante la simulación *SoC in the Loop* con la obtenida en las simulaciones MIL y SIL. En la inferior se muestra el valor de la diferencia entre las potencias de salida de las simulaciones *SoC in the Loop* y MIL/SIL. La potencia de salida obtenida con la simulación *SoC in the Loop* es prácticamente idéntica a la obtenida en las simulaciones MIL y SIL. La diferencia entre las potencias de salida varía entre 0 y 10^{-4}.

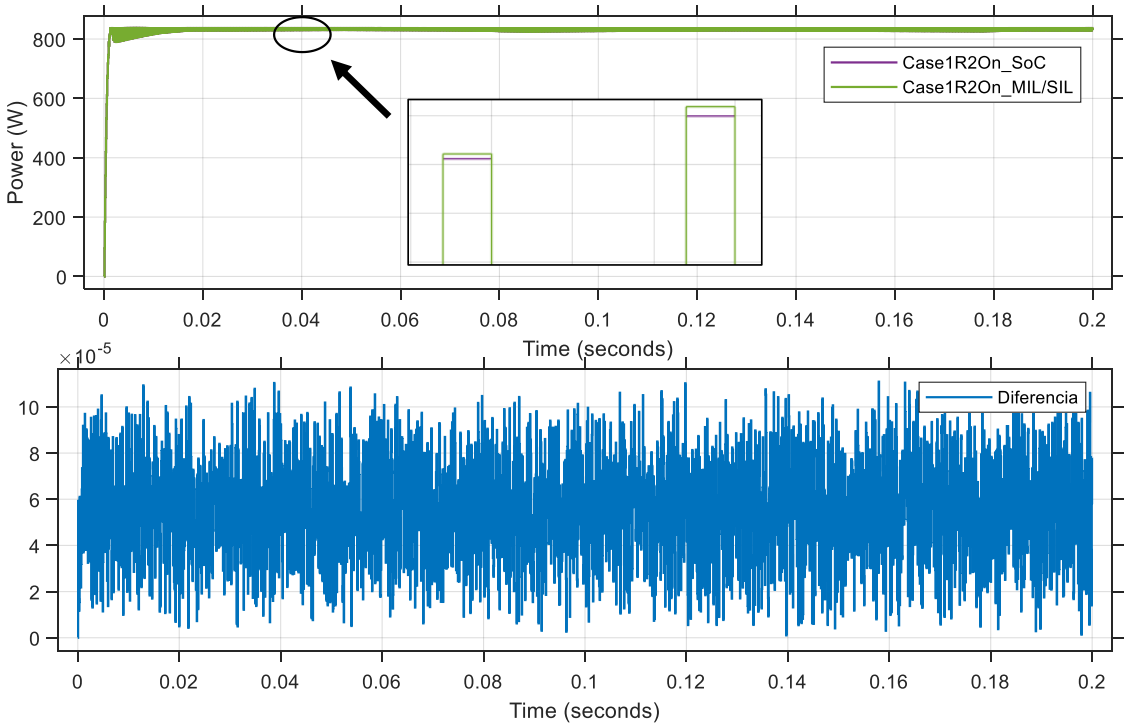


Figura 80. Simulación de SimscapeModel en conjunto con ZedBoardModel para el caso 1 con carga R_2.

De igual manera, en la Figura 81 y Figura 82 se presentan los resultados de realizar la simulación *SoC in the Loop* con la carga R_2 para los casos 2 y 4. En estos casos también la diferencia entre los resultados de la simulación *SoC in the Loop* y MIL/SIL varía entre 0 y 10^{-4}.

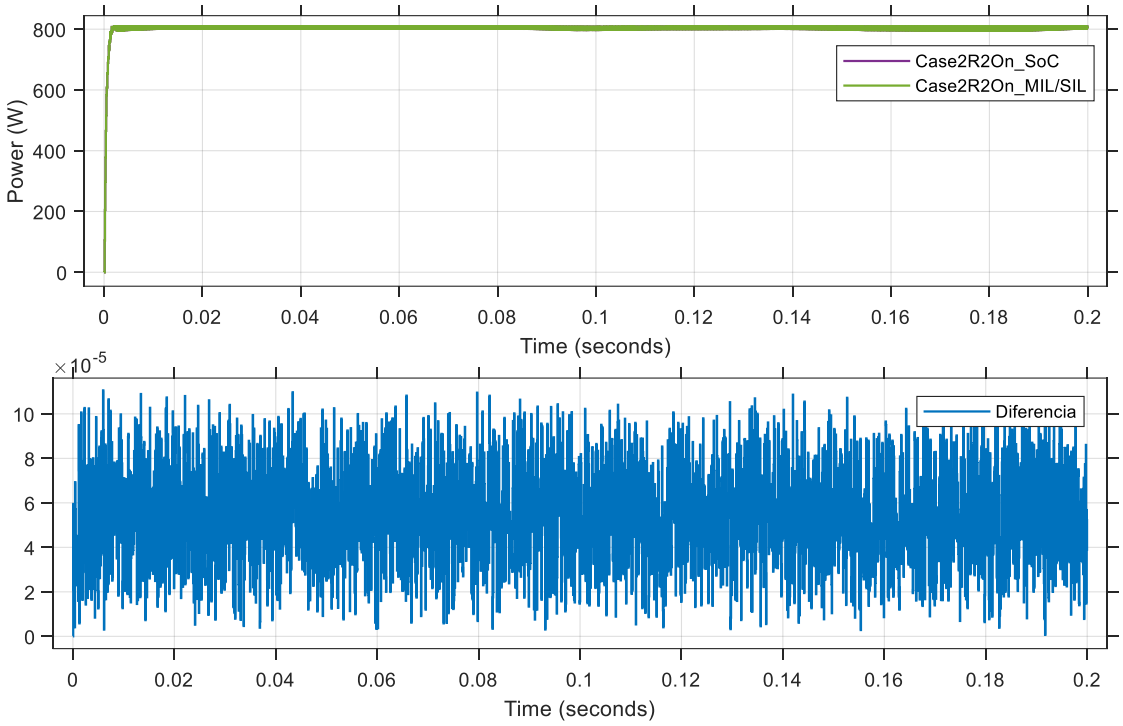


Figura 81. Simulación de SimscapeModel en conjunto con ZedBoardModel para el caso 2 con carga R_2.

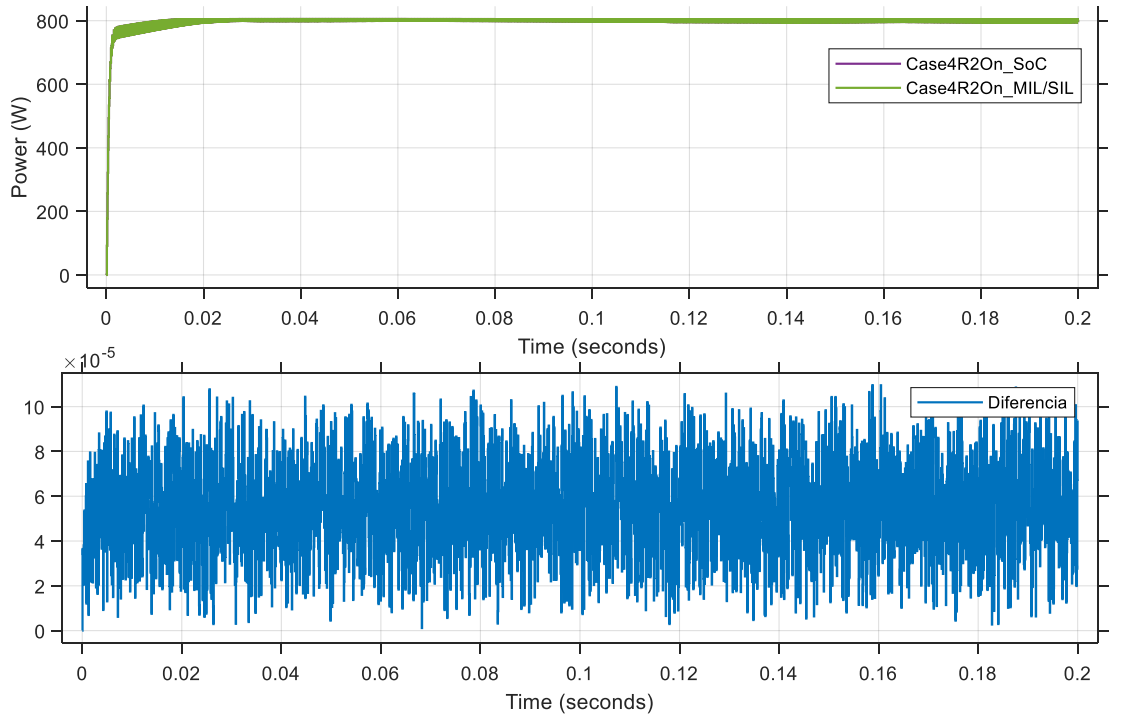


Figura 82. Simulación de SimscapeModel en conjunto con ZedBoardModel para el caso 4 con carga R_2.

También se realiza la simulación de SimscapeModel en conjunto con la implementación de la ZedBoard para una carga inductiva. En la Figura 83 se presentan los resultados de simular el caso 1 con la carga RL_2. Al igual que las simulaciones anteriores con carga resistiva, la diferencia en la potencia de salida entre la simulación *SoC in the Loop* y MIL/SIL varía entre 0 y 10^{-4}.

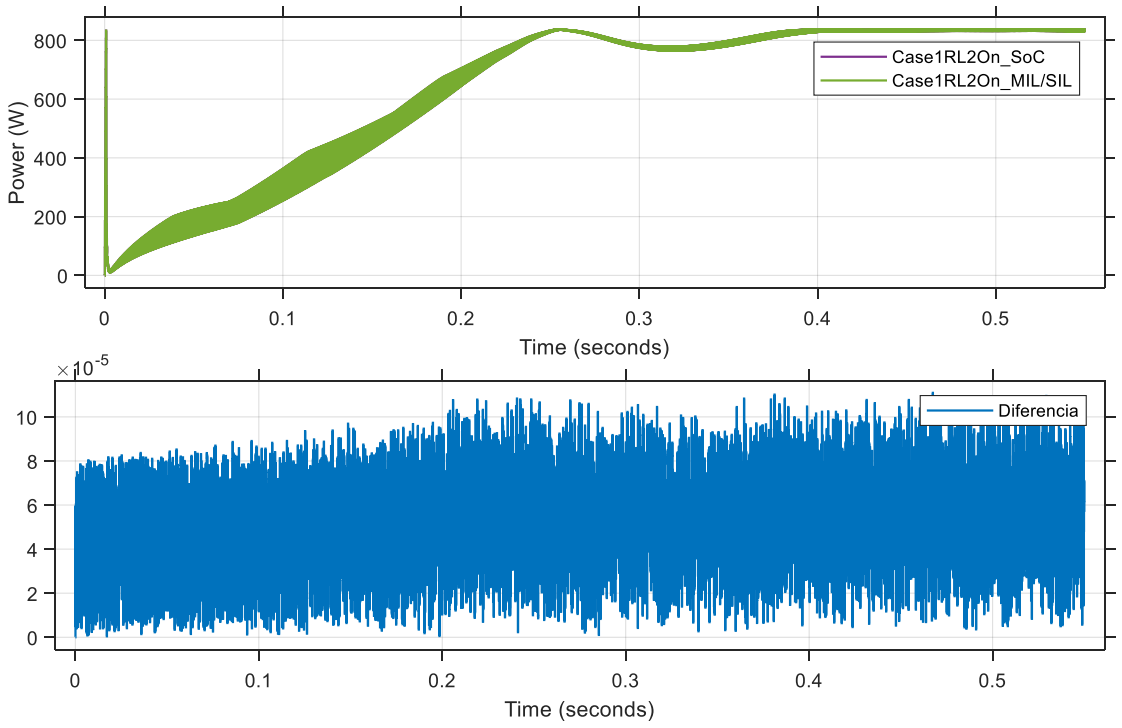


Figura 83. Simulación de SimscapeModel en conjunto con ZedBoardModel para el caso 1 con carga RL_2.

Para los casos 2 y 4, Figura 84 y Figura 85 respectivamente, con la carga RL_2, la diferencia entre la simulación *SoC in the Loop* y MIL/SIL permanece igualmente entre 0 y 10^{-4}.

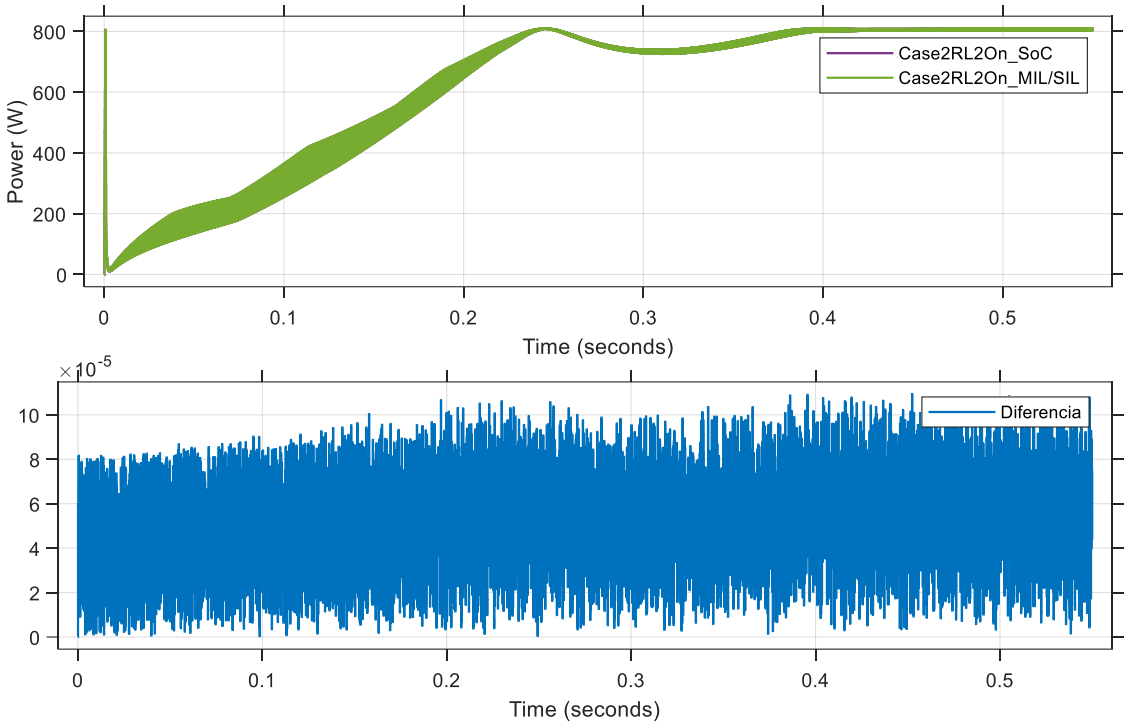


Figura 84. Simulación de SimscapeModel en conjunto con ZedBoardModel para el caso 2 con carga RL_2.

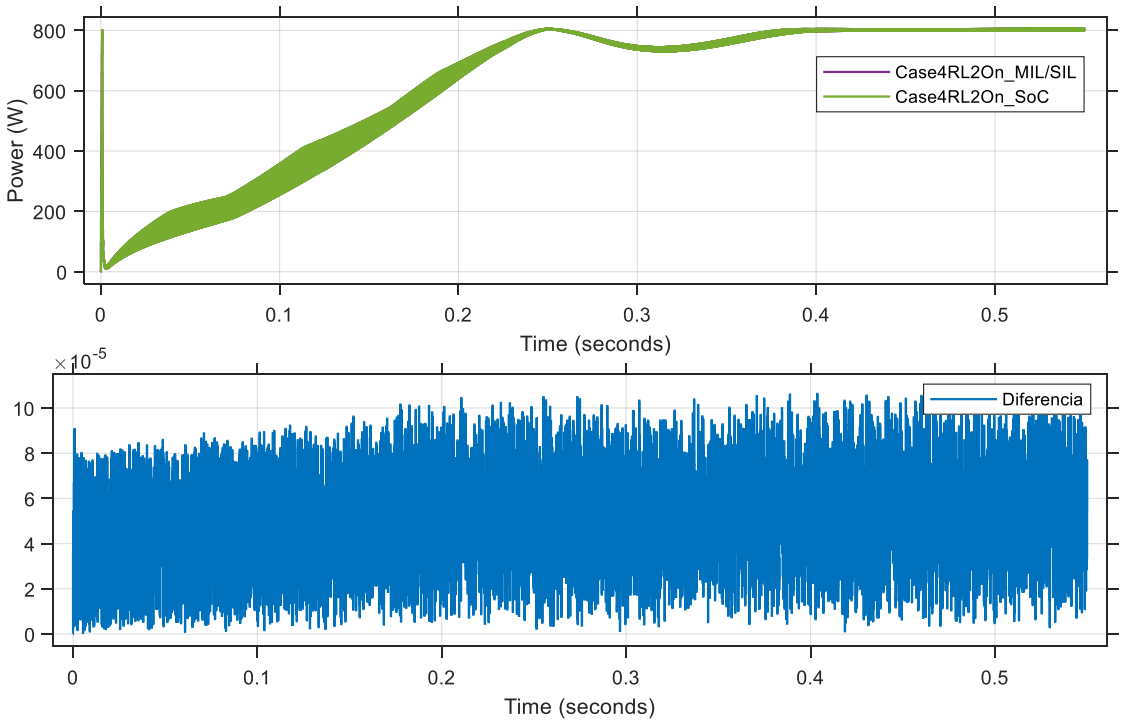


Figura 85. Simulación de SimscapeModel en conjunto con ZedBoardModel para el caso 4 con carga RL_2.

La diferencia que se obtiene en la potencia de salida en la simulación *SoC in the Loop* frente a la MIL/SIL se debe a que la implementación del MPPT en la ZedBoard es en punto fijo frente a la implementación en coma flotante en Simulink. Esto hace que el ciclo de trabajo calculado por la ZedBoard sea ligeramente diferente al calculado por Simulink. A

causa de esto, el punto de trabajo de la instalación fotovoltaica varía ligeramente provocando la diferencia en las potencias de salida.

Durante el proceso de verificación se toman medidas del tiempo de ejecución de la implementación *hardware/software* del algoritmo MMTES en la ZedBoard. También se mide el tiempo de ejecución del *BitonicSort* y del *PairForSwappingPll* teniendo en cuenta el tiempo consumido por las funciones de empaquetado y desempaquetado de los datos para los bloques IP. Con el fin de realizar una comparativa de los tiempos de ejecución, se crea un nuevo modelo en Simulink basado en el modelo ZedBoardModel, se genera un ejecutable para Linux y se carga en la placa. Este nuevo modelo cambia la implementación *hardware/software* del algoritmo MMTES por uno puramente *software*. Esto quiere decir que, el algoritmo MMTES se ejecuta de forma íntegra en el PS del Zynq sin utilizar los bloques IP implementados en el PL. Al simular SimscapeModel con este nuevo modelo cargado en la ZedBoard, se puede medir el tiempo de ejecución de la implementación puramente *software* del algoritmo MMTES funcionando en el procesador ARM de la ZedBoard. Una comparativa de los tiempos de ejecución medidos se presenta en la Tabla 22.

Tabla 22. Comparativa de los tiempos de ejecución para la implementación software y hardware/software.

	***Software* (µs)**	***Hardware Software* (µs)**	**Diferencia**
PairForSwapping	3.59	7.60	+ 4.01
Sort	20.95	21.22	+ 0.27
Resto de funciones	29.30	9.49	-19.81
Total	53.84	38.31	-15.53

Como se puede observar, el tiempo de ejecución medio para la implementación *hardware/software* es considerablemente menor que el de la implementación puramente *software*, en torno a un 29%. Sin embargo, el tiempo de ejecución en *software* de las funciones *PairForSwapping* y *Sort* es menor al de las implementaciones *hardware/software* de *PairForSwappingPll* y *BitonicSort*. Esto es debido mayoritariamente a que en el tiempo de ejecución de las funciones *PairForSwappingPll* y *BitonicSort* se está teniendo en cuenta el tiempo consumido por las funciones de empaquetado y desempaquetado de datos, así como el *Data Movement*.

Dado que la ejecución en *software* es de tipo secuencial, el aumento en la cantidad de datos afecta más al tiempo de ejecución que en el caso de la implementación *hardware/software* donde se implementan algoritmos paralelos en el *hardware*. Por ejemplo, la función *Sort* en *software* está ordenando un total de nueve valores (matriz de irradiancia y posición 3 x 3) mientras que el IP *BitonicSort*, para los mismos datos de entrada, ordena 16 valores.

Si se comparan los tiempos de ejecución de estas dos implementaciones para un número equivalente de valores, es decir, con una matriz de irradiancias y posición 4 x 4, las mediciones son muy diferentes. Mientras que el tiempo de ejecución de la implementación *hardware/software* del *BitonicSort* permanece en 21.22 µs, el de la implementación *software* de *Sort* asciende hasta los 53.39 µs. Esto representa un aumento del 155% en el tiempo de ejecución con respecto a una matriz de entrada 3 x 3 mientras que en el caso del *BitonicSort* el tiempo de ejecución permanece sin cambios.

En la Figura 86 se muestra una gráfica con los tiempos de ejecución estimados de ambas implementaciones para diferentes cantidades de paneles fotovoltaicos. Como se puede ver, el aumento en el tiempo de ejecución en función de la cantidad de paneles fotovoltaicos a controlar en la implementación *software* es cada vez mayor. Por otro lado, en el caso de la implementación *hardware/software*, el aumento en el tiempo de ejecución conforme aumentan los paneles fotovoltaicos es mucho menor. Para el caso de la función ***PairForSwapping*** es similar.

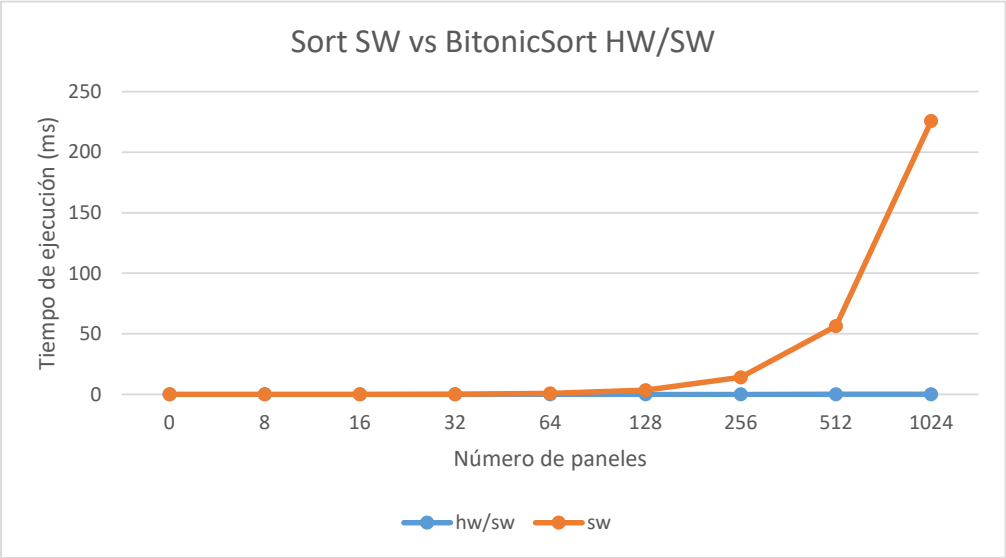


Figura 86. Comparativa de los tiempos de ejecución de Sort SW y BitonicSort HW/SW.

La razón por la cual el tiempo medio de ejecución de la implementación *hardware/software* es menor al de la implementación *software* aun siendo las funciones ***PairForSwappingPll*** y ***BitonicSort*** más lentas en los casos de estudio, es que los tiempos de ejecución son más estables. El tiempo mínimo de ejecución medido para la implementación *software* es menor que el menor tiempo de ejecución de la implementación *hardware/software*. Por otro lado, el mayor tiempo de ejecución de la implementación *software* es mucho mayor que el de la implementación *hardware/software*. El valor del tiempo de ejecución de la implementación *software* varía mucho, al contrario que en la implementación *hardware/software* donde se mantiene estable.

El bajo aumento del tiempo de ejecución en función del número de paneles fotovoltaicos a controlar le confiere a la implementación *hardware/software* una ventaja frente a la implementación *software* en términos de escalabilidad del sistema. La implementación *hardware/software* ofrece la posibilidad de aumentar el número de paneles fotovoltaicos a controlar manteniendo el tiempo de ejecución total del algoritmo MMTES mucho menor que en una implementación puramente *software* para el mismo número de paneles fotovoltaicos. Además, la estabilidad del tiempo de ejecución obtenida en la implementación *hardware/software* le confiere a esta la posibilidad de integrarse en un sistema en tiempo real en caso de ser requerido, algo que sería más difícil de lograr con la implementación *software* debido a la gran variabilidad en los tiempos de ejecución que presenta.

9.4. Conclusiones

En primer lugar, hay que remarcar las facilidades que ofrece MATLAB/Simulink para implementar un sistema que combine una partición *hardware* con una partición *software* sobre un dispositivo SoC FPGA. Facilita especialmente la creación de la partición *software*, ya que el toolbox Embedded Coder permite generar código de forma automática para ser ejecutado en el procesador del SoC FPGA. Además, este toolbox ofrece las herramientas necesarias para poder comunicar la placa de desarrollo que contiene el SoC FPGA con MATLAB/Simulink y así, poder llevar a cabo la verificación del sistema implementado mediante una simulación *SoC in the Loop*.

Por otro lado, los resultados de la simulación *SoC in the Loop* realizada con satisfactorios. Los valores de potencia de salida obtenidos para todos los casos en la simulación *SoC in the Loop* varían muy poco con respecto a los valores obtenidos durante las simulaciones MIL y SIL. El valor de la diferencia oscila entre 0 y 10^{-4}. Esta diferencia de debe a la implementación hardware del MPPT realizada. Esta, a diferencia de la implementación en coma flotante utilizada en Simulink durante las simulaciones MIL y SIL, utiliza aritmética en punto fijo. Esto introduce un pequeño error, en torno a 10^{-8} en el cálculo del ciclo de trabajo, variando ligeramente el punto de trabajo de la instalación y causando la diferencia en la potencia de salida.

Los tiempos de ejecución de la implementación *hardware/software* del algoritmo MMTES presentan una clara mejora frente a los de la implementación puramente *software*. El tiempo de ejecución de la implementación *hardware/software* es de media un 29% menor. Además, esta implementación también presenta grandes ventajas en cuanto a la escalabilidad del sistema. Gracias a la implementación paralela en *hardware* de las funciones ***BitonicSort*** y ***PairForSwappingPll,*** la implementación *hardware/software* consigue tiempos de ejecución un 30% menores que la implementación *software* al aumentar el tamaño del sistema fotovoltaico a controlar.

Capítulo 10. Conclusiones y trabajos futuros

10.1. Conclusiones

Respecto a los resultados obtenidos al realizar la implementación del algoritmo MMTES y el controlador MPPT, estos son satisfactorios. La funcionalidad de la implementación desarrollada para estos dos algoritmos ha sido verificada mediante una simulación *SoC in the Loop* con una placa de desarrollo. Además de verificar la funcionalidad, se ha comprobado que la implementación *hardware/software* realizada del algoritmo MMTES presenta mejoras en cuanto a tiempo de ejecución y estabilidad temporal frente a una implementación puramente *software*. En concreto, para los casos de estudio implementados, se ha conseguido reducir en un 29% el tiempo medio de ejecución del algoritmo MMTES.

En cuanto a la metodología de diseño propuesta, su funcionalidad queda demostrada mediante la implementación realizada. La metodología de diseño que se propone presenta varias mejoras frente a metodologías utilizadas en ocasiones anteriores [54]. Entre las mejoras que ofrece se encuentra la facilidad para realizar la partición *hardware/software* de un diseño desde alto nivel y, partiendo de esta, generar de forma automática código para el *software* empotrado de la partición *software* y código HDL para la partición *hardware*. Además, poder realizar la mayoría de los pasos del flujo de diseño incluidos en la metodología de codiseño presentada desde una única herramienta, en este caso MATLAB/Simulink, facilita y agiliza el proceso de diseño, especialmente para aquellos casos en los que se requiera un tiempo de desarrollo corto o la evaluación de distintas alternativas del algoritmo. Por último, la posibilidad que ofrece esta metodología de utilizar

modelados creados en Simulink durante el proceso de verificación es una característica muy atractiva, sobre todo, en el ámbito industrial donde puede llegar a ser muy complicado y costoso realizar la verificación de una implementación con el sistema real.

En [53] se estudia una metodología, que al igual que la presentada en este TFM, está basada en MATLAB/Simulink. Es una metodología similar que también facilita y agiliza el diseño de controladores para FPGA, pero que no es compatible con tecnología SoC FPGA. La metodología presentada en este TFM ofrece las mismas prestaciones que la presentada en [53] en cuanto al diseño de controladores sobre FPGA y, además, permite el diseño e implementación de sistemas más complejos que requieran de una aproximación *hardware/software* sobre tecnología SoC FPGA.

En [103] se realiza un estudio en el campo del prototipado virtual dentro de las aplicaciones para automoción. Se estudia la co-simulación de sistema *hardware/software* que incluyen *firmware*, protocolos de comunicación y sistemas físico/mecánicos en el ámbito del desarrollo ágil basado en modelos. Se compara MATLAB/Simulink con SystemC en cuanto al rendimiento de las simulaciones, las capacidades de modelado que ofrecen y la aplicabilidad en diferentes fases del proceso de desarrollo. Los resultados de simulación con ambas aproximaciones son similares. Se llega a la conclusión de que MATLAB/Simulink es interesante como método de desarrollo interactivo mientras que SystemC, en conjunto con un simulador de juego de instrucciones (ISS), es eficiente en la creación de modelos de procesamiento.

Existen diferentes implementaciones de algoritmos DPVAR sobre tecnología FPGA y SoC FPGA en el estado del arte. Si bien estos algoritmos no son comparables entre sí debido a que tienen principios de funcionamiento diferentes, en la Tabla 23 se presentan algunos resultados.

Tabla 23. Comparativa de implementaciones de algoritmos DPVAR sobre FPGA.

Algoritmo	Placa	t (s)	FF	LUT	BRAM	DSP	I/O Pins	Comb. func	Total registers
MMTES	ZedBoard	$38.31*10^{-6}$	1.0%	7.0%	0.0%	0.0%	-	-	-
GA [55]	Zybo	2.37	30.0%	81.0%	50.0%	33.8%	-	-	-
irradiance equalization [45]	Altera DE2-70	$0.54*10^{-6}$	-	-	-	-	220	1333	442

De igual manera, en la Tabla 24 se presentan resultados de implementación, con respecto a utilización de recursos, de diferentes tipos de controladores MPPT sobre tecnología FPGA.

Tabla 24. Comparativa de implementaciones de controladores MPPT sobre FPGA.

MMPT	Placa	FF	LUT	DSP
Inc	ZedBoard	244	1743	6
Kalman [47]	Zybo	780	567	18
P&O [104]	XC6SLX45	1416	1062	1

10.2.Trabajos futuros

El algoritmo MMTES ha sido implementado en este trabajo para un *array* fotovoltaico de pequeñas dimensiones. Se ha implementado de esta manera para reducir los tiempos de simulación del sistema a controlar en Simulink y utilizar los casos de estudio propuestos. Sin embargo, los *arrays* fotovoltaicos industriales de grandes dimensiones son los que se pueden beneficiar en mayor medida de este tipo de algoritmos. Cuanto mayor sea la extensión del *array* fotovoltaico, más fácil es que sea afectado por sombras parciales producidas por las nubes.

En base a los puntos previamente comentados, se proponen las siguientes líneas de trabajos futuros:

1. Realizar la validación del sistema implementado en la ZedBoard en conjunto con una planta real.
2. Realizar la implementación del algoritmo MMTES para un *array* fotovoltaico de grandes dimensiones.
3. Modificar el modelado del sistema para implementar un *array* de mayor dimensión.
4. Añadir al modelo del sistema a controlar una etapa inversora trifásica/alterna.

Referencias

[1] M. Breque, L. De Nul, and A. Petridis, “Industry 5.0 - Publications Office of the EU,” 04-Jan-2021. [Online]. Available: https://op.europa.eu/en/publication-detail/-/publication/468a892a-5097-11eb-b59f-01aa75ed71a1/. [Accessed: 27-Sep-2021]

[2] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, “FPGAs in Industrial Control Applications,” *IEEE Trans. Ind. Informatics*, vol. 7, no. 2, pp. 224–243, May 2011, doi: 10.1109/TII.2011.2123908.

[3] E. Monmasson, M. Hilairet, G. Spagnuolo, and M. Cirstea, “System-on-Chip FPGA Devices for Complex Electrical Energy Systems Control,” *IEEE Ind. Electron. Mag.*, p. 0, 2021, doi: 10.1109/MIE.2021.3052179.

[4] X. Inc, “Zynq-7000 SoC Data Sheet: Overview,” *DS190 (v1.11.1) July 2*, 2018. .

[5] ARM, “Cortex-A9. Technical Reference Manual,” 2012. .

[6] I. Xilinx, “7 Series FPGAs Data Sheet: Overview,” 2020. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[7] P. Wadhwani and S. Yadav, “Field Programmable Gate Array (FPGA) Market Size & Share | Global Forecasts 2027,” Jan-2021. [Online]. Available: https://www.gminsights.com/industry-analysis/field-programmable-gate-array-fpga-market-size. [Accessed: 27-Sep-2021]

[8] “Global Field-Programmable Gate Array (FPGA) Market -Industry Analysis and Forecast (2019-2026).” [Online]. Available: https://www.maximizemarketresearch.com/market-report/global-field-programmable-gate-array-fpga-market/22895/. [Accessed: 29-Dec-2021]

[9] “Validation of European high capacity rad-hard FPGA and software tools,” pp. 1–12, 2021.

[10] Z. Wan *et al.*, “A Survey of FPGA-Based Robotic Computing,” *IEEE Circuits Syst. Mag.*, vol. 21, no. 2, pp. 48–74, 2021, doi: 10.1109/MCAS.2021.3071609.

[11] G. Divya Vani, K. S. Rao, and M. C. Chinnaiah, “Self-Automated Parking with FPGA-Based Robot,” in *Micro and Nanoelectronics Devices, Circuits and Systems: Select Proceedings of MNDCS 2021*, T. R. Lenka, D. Misra, and A. Biswas, Eds. Singapore: Springer Singapore, 2022, pp. 459–470 [Online]. Available: https://doi.org/10.1007/978-981-16-3767-4_45

[12] A. Kasem, A. Bouzid, A. Reda, and J. Vásárhelyi, “A Survey about Intelligent Solutions for Autonomous Vehicles based on FPGA,” *Carpathian Journal of Electronic and Computer Engineering 13(2):9-13*, Dec-2020. [Online]. Available: https://www.researchgate.net/publication/347947301_A_Survey_about_Intelligent_Solutions_for_Autonomous_Vehicles_based_on_FPGA/citations. [Accessed: 03-Oct-2021]

[13] M. Vyas, “Trends of FPGA use in Automotive Engineering,” in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, 2018, pp. 580–591, doi: 10.1109/RTEICT42901.2018.9012495.

[14] S. Di Cairano and I. V Kolmanovsky, “Real-time optimization and model predictive control for aerospace and automotive applications,” in *2018 Annual American Control Conference (ACC)*, 2018, pp. 2392–2409, doi: 10.23919/ACC.2018.8431585.

[15] S. Di Cairano and I. V Kolmanovsky, “Automotive applications of model predictive control,” in *Handbook of Model Predictive Control*, Springer, 2019, pp. 493–527.

[16] Ó. Lopez, J. Alvarez, J. Doval-Gandoy, and F. D. Freijedo, “Multilevel Multiphase Space Vector PWM Algorithm,” *IEEE Trans. Ind. Electron.*, vol. 55, no. 5, pp. 1933–1942, May 2008, doi: 10.1109/TIE.2008.918466.

[17] H. Hatas, N. Genc, and A. Mamizadeh, “FPGA Implementation of SPWM for Cascaded Multilevel Inverter by Using XSG,” in *2019 4th International Conference on Power Electronics and their Applications (ICPEA)*, 2019, pp. 1–6, doi: 10.1109/ICPEA1.2019.8911189.

[18] B. J. Patella, A. Prodic, A. Zirger, and D. Maksimovic, “High-frequency digital controller IC for DC/DC converters,” in *APEC. Seventeenth Annual IEEE Applied Power Electronics Conference and Exposition (Cat. No.02CH37335)*, 2002, vol. 1, pp. 374–380 vol.1, doi: 10.1109/APEC.2002.989273.

[19] Y. Wu *et al.*, “An FPGA Based Energy Efficient DS-SLAM Accelerator for Mobile Robots in Dynamic Environment,” *Appl. Sci.*, vol. 11, no. 4, 2021, doi: 10.3390/app11041828. [Online]. Available: https://www.mdpi.com/2076-3417/11/4/1828

[20] B. Plancher, S. M. Neuman, T. Bourgeat, S. Kuindersma, S. Devadas, and V. Janapa Reddi, “Accelerating Robot Dynamics Gradients on a CPU, GPU, and FPGA,” *IEEE Robot. Autom. Lett.*, p. 1, 2021, doi: 10.1109/LRA.2021.3057845.

[21] W. Zhu, T. Lamarche, E. Dupuis, D. Jameux, P. Barnard, and G. Liu, “Precision Control of Modular Robot Manipulators: The VDC Approach With Embedded FPGA,” *IEEE Trans. Robot.*, vol. 29, no. 5, pp. 1162–1179, Oct. 2013, doi: 10.1109/TRO.2013.2265631.

[22] V. Bargsten and J. de Gea Fernández, “Distributed computation and control of robot motion dynamics on FPGAs,” *SN Appl. Sci.*, vol. 2, no. 7, p. 1239, 2020, doi: 10.1007/s42452-020-2898-6. [Online]. Available: https://doi.org/10.1007/s42452-020-2898-6

[23] A. M. Romanov *et al.*, “Modular Reconfigurable Robot Distributed Computing System for Tracking Multiple Objects,” *IEEE Syst. J.*, vol. 15, no. 1, pp. 802–813, 2021, doi: 10.1109/JSYST.2020.2990921.

[24] B. Pierce and G. Cheng, “Versatile modular electronics for rapid design and development of humanoid robotic subsystems,” in *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2014, pp. 735–741, doi: 10.1109/AIM.2014.6878166.

[25] W.-C. Chen, C.-S. Chen, F.-C. Lee, and Y.-S. Kung, “FPGA-realization of the kinematics IP for SCARA robot,” *Microsyst. Technol.*, vol. 27, no. 4, pp. 1075–1090, 2021, doi: 10.1007/s00542-018-4061-5. [Online]. Available: https://doi.org/10.1007/s00542-018-4061-5

[26] M.-A. Martínez-Prado, J. Rodríguez-Reséndiz, R.-A. Gómez-Loenzo, G. Herrera-Ruiz, and L.-A. Franco-Gasca, “An FPGA-Based Open Architecture Industrial Robot Controller,” *IEEE Access*, vol. 6, pp. 13407–13417, 2018, doi: 10.1109/ACCESS.2018.2797803.

[27] D. Jokić, S. Lubura, V. Rajs, M. Bodić, and H. Šiljak, “Two Open Solutions for Industrial Robot Control: The Case of PUMA 560,” *Electronics*, vol. 9, no. 6, 2020, doi: 10.3390/electronics9060972. [Online]. Available: https://www.mdpi.com/2079-9292/9/6/972

[28] B. P. Jeppesen, N. Roy, L. Moro, and F. Baronti, “An FPGA-based controller for collaborative robotics,” in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017, pp. 1067–1072, doi: 10.1109/ISIE.2017.8001394.

[29] J. Lavin, S. Chavez-Vázquez, J. F. Gómez-Aguilar, G. Delgado-Reyes, and M. Ruiz-Jaimes, “Fractional-order passivity-based adaptive controller for a robot manipulator type scara,” *Fractals*, vol. 28, 2020, doi: 10.1142/S0218348X20400083.

[30] M. A. Magaña Méndez, E. Hernandez Rios, J. Ben´ıtez-Morales, O. Domínguez-Ramírez, and J. Fernández, “Implementación en un laboratorio virtual de un sistema teleoperado en configuración maestro esclavo con comunicación basada en el protocolo TCP/IP,” *Pädi Boletín Científico Ciencias Básicas e Ing. del ICBI*, vol. 7, pp. 50–59, 2019, doi: 10.29057/icbi.v7iEspecial.4289.

[31] U. Eren, A. Prach, B. B. Koçer, S. V. Rakovic, E. Kayacan, and B. Açikmese, “Model predictive control in aerospace systems: Current state and opportunities,” *J. Guid. Control. Dyn.*, vol. 40, no. 7, pp. 1541–1566, 2017, doi: 10.2514/1.G002507.

[32] P. W. Gibbens and E. D. B. Medagoda, “Efficient Model Predictive Control Algorithm for Aircraft,” *Artic. J. Guid. Control Dyn.*, 2011, doi: 10.2514/1.52162. [Online]. Available: https://www.researchgate.net/publication/270872533. [Accessed: 04-Oct-2021]

[33] S. Lucia, D. Navarro, Ó. Lucía, P. Zometa, and R. Findeisen, “Optimized FPGA Implementation of Model Predictive Control for Embedded Systems Using High-Level Synthesis Tool,” *IEEE Trans. Ind. Informatics*, vol. 14, no. 1, pp. 137–145, 2018, doi: 10.1109/TII.2017.2719940.

[34] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, “Predictive Control of a Boeing 747 Aircraft using an FPGA,” *IFAC Proc. Vol.*, vol. 45, no. 17, pp. 80–85, Jan. 2012, doi: 10.3182/20120823-5-NL-3013.00016.

[35] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, “Predictive Control Using an FPGA With Application to Aircraft Control,” *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 3, pp. 1006–1017, 2014, doi: 10.1109/TCST.2013.2271791.

[36] A. Reda, A. Bouzid, and J. Vásárhelyi, “Model Predictive Control for Automated Vehicle Steering,” *Acta Polytech. Hungarica*, vol. 17, no. 7, pp. 163–182, 2020.

[37] H. Guo, F. Liu, F. Xu, H. Chen, D. Cao, and Y. Ji, “Nonlinear Model Predictive Lateral Stability Control of Active Chassis for Intelligent Vehicles and Its FPGA Implementation,” *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 49, no. 1, pp. 2–13, 2019, doi: 10.1109/TSMC.2017.2749337.

[38] M. PETRELLI, “Analysis and Design of embedded digital platforms for high performance Model Predictive Control in Automotive Applications,” 2020.

[39] F. Salewski and S. Kowalewski, “Hardware/Software Design Considerations for Automotive Embedded Systems,” *IEEE Trans. Ind. Informatics*, vol. 4, no. 3, pp. 156–163, 2008, doi: 10.1109/TII.2008.2002919.

[40] P. Conmy and I. Bate, “Component-Based Safety Analysis of FPGAs,” *IEEE Trans. Ind. Informatics*, vol. 6, no. 2, pp. 195–205, May 2010, doi: 10.1109/TII.2009.2039938.

[41] H. Yang, R. Yang, W. Hu, and Z. Huang, “FPGA-Based Sensorless Speed Control of PMSM Using Enhanced Performance Controller Based on the Reduced-Order EKF,” *IEEE J. Emerg. Sel. Top. Power Electron.*, vol. 9, no. 1, pp. 289–301, 2021, doi: 10.1109/JESTPE.2019.2962697.

[42] I. Mishra, R. N. Tripathi, V. K. Singh, and T. Hanamoto, “Step-by-Step Development and Implementation of FS-MPC for a FPGA-Based PMSM Drive System,” *Electronics*, vol. 10, no. 4, 2021, doi: 10.3390/electronics10040395. [Online]. Available: https://www.mdpi.com/2079-9292/10/4/395

[43] L. Rovere, A. Formentini, and P. Zanchetta, “FPGA Implementation of a Novel Oversampling Deadbeat Controller for PMSM Drives,” *IEEE Trans. Ind. Electron.*, vol. 66, no. 5, pp. 3731–3741, 2019, doi: 10.1109/TIE.2018.2851994.

[44] Z. Ma and X. Zhang, “FPGA Implementation of Sensorless Sliding Mode Observer With a Novel Rotation Direction Detection for PMSM Drives,” *IEEE Access*, vol. 6, pp. 55528–55536, 2018, doi: 10.1109/ACCESS.2018.2871730.

[45] L. Bouselham, B. Hajji, A. Mellit, A. Rabhi, and K. Kassmi, “Hardware implementation of new irradiance equalization algorithm for reconfigurable PV architecture on a FPGA platform,” in *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, 2019, pp. 1–8, doi: 10.1109/WITS.2019.8723746.

[46] A. A. Aldair, A. A. Obed, and A. F. Halihal, “Design and implementation of ANFIS-reference model controller based MPPT using FPGA for photovoltaic system,” *Renew. Sustain. Energy Rev.*, vol. 82, pp. 2202–2217, Feb. 2018, doi: 10.1016/J.RSER.2017.08.071.

[47] G. Becerra-Nuñez *et al.*, “An FPGA Kalman-MPPT Implementation Adapted in SST-Based Dual Active Bridge Converters for DC Microgrids Systems,” *IEEE Access*, vol. 8, pp. 202946–202957, 2020, doi: 10.1109/ACCESS.2020.3033718.

[48] A. Youssef, M. El Telbany, and A. Zekry, “Reconfigurable generic FPGA implementation of fuzzy logic controller for MPPT of PV systems,” *Renew. Sustain. Energy Rev.*, vol. 82, pp. 1313–1319, 2018, doi: https://doi.org/10.1016/j.rser.2017.09.093. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S136403211731345X

[49] A. Senthilvel, K. N. Vijeyakumar, and B. Vinothkumar, “FPGA based implementation of MPPT algorithms for photovoltaic system under partial shading conditions,” *Microprocess. Microsyst.*, vol. 77, p. 103011, 2020, doi: https://doi.org/10.1016/j.micpro.2020.103011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933119307148

[50] E. Monmasson and M. N. Cirstea, “FPGA Design Methodology for Industrial Control Systems—A Review,” *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, 2007, doi: 10.1109/TIE.2007.898281.

[51] Y. Sorel, “Massively parallel computing systems with real time constraints: the ‘Algorithm Architecture Adequation’ methodology,” in *Proceedings of the First International Conference on Massively Parallel Computing Systems (MPCS) The Challenges of General-Purpose and Special-Purpose Computing*, 1994, pp. 44–53, doi: 10.1109/MPCS.1994.367018.

[52] G. Udgirkar, “Survey of FPGA Methodology Considering High Performance Design Implementation,” in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2021, pp. 667–672, doi: 10.1109/ICICCS51141.2021.9432310.

[53] Y. P. Siwakoti and G. E. Town, “Design of FPGA-controlled power electronics and drives using MATLAB Simulink,” in *2013 IEEE ECCE Asia Downunder*, 2013, pp. 571–577, doi: 10.1109/ECCE-Asia.2013.6579155.

[54] G. Santana Quintana, “Controlador Pid Para Motores Eléctricos Implementado Sobre Fpga.” 2021 [Online]. Available: http://hdl.handle.net/10553/105670

[55] G. Petrone, F. Serra, G. Spagnuolo, and E. Monmasson, “SoC implementation of a photovoltaic reconfiguration algorithm by exploiting a HLS-based architecture,” *Math. Comput. Simul.*, vol. 158, pp. 520–537, 2019, doi: https://doi.org/10.1016/j.matcom.2018.12.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378475418303331

[56] G. H. K. Varma, V. R. Barry, R. K. Jain, and D. Kumar, “An MMTES algorithm for dynamic photovoltaic array reconfiguration to enhance power output under partial shading conditions,” *IET Renew. Power Gener.*, vol. 15, no. 4, pp. 809–820, Mar. 2021, doi: 10.1049/RPG2.12070. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1049/rpg2.12070. [Accessed: 05-Oct-2021]

[57] “MATLAB Documentation - MathWorks.” [Online]. Available: https://es.mathworks.com/help/matlab/index.html?lang=en. [Accessed: 05-Oct-2021]

[58] MathWorks, “Simulink Documentation - MathWorks United Kingdom.” [Online]. Available: https://uk.mathworks.com/help/simulink/index.html

[59] MathWorks, “Simscape Electrical Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/physmod/sps/index.html?searchHighlight=simscape electrical&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[60] MathWorks, “Fixed-Point Designer Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/fixedpoint/index.html?searchHighlight=fixed point designer&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[61] MathWorks, “MATLAB Coder Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/coder/index.html?searchHighlight=matlab coder&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[62] MathWorks, “Simulink Coder Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/rtw/index.html?searchHighlight=Simulink Coder&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[63] MathWorks, “Embedded Coder Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/ecoder/index.html?searchHighlight=Embedded coder&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[64] MathWorks, “HDL Coder Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/hdlcoder/index.html?searchHighlight=hdl coder&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[65] MathWorks, “HDL Coder Support Package for Xilinx Zynq Platform Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/supportpkg/xilinxzynq7000/index.html?searchHighlight=HDL Coder Support Package for Xilinx Zynq Platform&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[66] MathWorks, “Embedded Coder Support Package for Xilinx Zynq Platform Documentation - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/supportpkg/xilinxzynq7000ec/index.html?searchHighlight=Embedded Coder Support Package for Xilinx Zynq Platform&s_tid=srchtitle. [Accessed: 06-Oct-2021]

[67] MathWorks, “Hardware-Software Co-Design Workflow for SoC Platforms - MATLAB & Simulink - MathWorks España.” [Online]. Available: https://es.mathworks.com/help//hdlcoder/ug/hardware-software-codesign-workflow-for-soc-platforms.html. [Accessed: 30-Mar-2022]

[68] Xilinx Inc, “Vivado Design Suite User Guide.” [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug893-vivado-ide.pdf. [Accessed: 06-Oct-2021]

[69] I. D. D. M. de Pierrepont Franzetti, D. Carminati, M. Scanavino, and E. Capello, “Model-In-the-Loop Testing of Control Systems and Path Planner Algorithms for QuadRotor UAVs,” in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 1809–1818, doi: 10.1109/ICUAS48674.2020.9213885.

[70] M. A. Taut, G. Chindris, and A. C. Taut, “Software-in-the-Loop System for Motor Control Algorithms,” in *2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2019, pp. 419–426, doi: 10.1109/SIITME47687.2019.8990711.

[71] H. Patel and V. Agarwal, “MATLAB-Based Modeling to Study the Effects of Partial Shading on PV Array Characteristics,” *IEEE Trans. Energy Convers.*, vol. 23, no. 1, pp. 302–310, 2008, doi: 10.1109/TEC.2007.914308.

[72] R. Mahto, D. Sharma, D. Xavier, and R. Raghavan, “Improving performance of photovoltaic panel by reconfigurability in partial shading condition,” *J. Photonics Energy*, vol. 10, p. 42004, 2020, doi: 10.1117/1.JPE.10.042004.

[73] R. Ramaprabha and B. L. Mathur, “A Comprehensive Review and Analysis of Solar Photovoltaic Array Configurations under Partial Shaded Conditions,” *Int. J. Photoenergy*, vol. 2012, p. 120214, 2012, doi: 10.1155/2012/120214. [Online]. Available: https://doi.org/10.1155/2012/120214

[74] M. Z. Shams El-Dein, M. Kazerani, and M. M. A. Salama, “Optimal Photovoltaic Array Reconfiguration to Reduce Partial Shading Losses,” *IEEE Trans. Sustain. Energy*, vol. 4, no. 1, pp. 145–153, 2013, doi: 10.1109/TSTE.2012.2208128.

[75] D. Nguyen and B. Lehman, “A reconfigurable solar photovoltaic array under shadow conditions,” in *2008 Twenty-Third Annual IEEE Applied Power Electronics Conference and Exposition*, 2008, pp. 980–986, doi: 10.1109/APEC.2008.4522840.

[76] S. Pareek and R. Dahiya, “Output Power Maximization of Partially Shaded 4*4 PV Field by Altering its Topology,” *Energy Procedia*, vol. 54, pp. 116–126, Jan. 2014, doi: 10.1016/J.EGYPRO.2014.07.254.

[77] “Bitonic Sort - GeeksforGeeks.” [Online]. Available: https://www.geeksforgeeks.org/bitonic-sort/. [Accessed: 26-Oct-2021]

[78] “Bitonic sorter - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/Bitonic_sorter. [Accessed: 26-Oct-2021]

[79] “Constante solar - Wikipedia, la enciclopedia libre.” [Online]. Available: https://es.wikipedia.org/wiki/Constante_solar. [Accessed: 26-Oct-2021]

[80] T. Esram and P. L. Chapman, “Comparison of Photovoltaic Array Maximum Power Point Tracking Techniques,” *IEEE Trans. Energy Convers.*, vol. 22, no. 2, pp. 439–449, 2007, doi: 10.1109/TEC.2006.874230.

[81] N. Femia, G. Petrone, G. Spagnuolo, and M. Vitelli, “Optimization of perturb and observe maximum power point tracking method,” *IEEE Trans. Power Electron.*, vol. 20, no. 4, pp. 963–973, 2005, doi: 10.1109/TPEL.2005.850975.

[82] H. Deopare and A. Deshpande, “Modeling and simulation of Incremental conductance Maximum Power Point tracking,” in *2015 International Conference on Energy Systems and Applications*, 2015, pp. 501–505, doi: 10.1109/ICESA.2015.7503400.

[83] D. Baimel, S. Tapuchi, Y. Levron, and J. Belikov, “Improved Fractional Open Circuit Voltage MPPT Methods for PV Systems,” *Electronics*, vol. 8, no. 3, 2019, doi: 10.3390/electronics8030321. [Online]. Available: https://www.mdpi.com/2079-9292/8/3/321

[84] B. Bendib, H. Belmili, and F. Krim, “A survey of the most used MPPT methods: Conventional and advanced algorithms applied for photovoltaic systems,” *Renew. Sustain. Energy Rev*, vol. 45, pp. 637–648, 2015, doi: 10.1016/j.rser.2015.02.009.

[85] MathWorks, “Implement PV array modules - Simulink - MathWorks España.” [Online]. Available: https://es.mathworks.com/help/physmod/sps/powersys/ref/pvarray.html.

[Accessed: 30-Oct-2021]

[86] S. Lineykin, M. Averbukh, and A. Kuperman, "Five-parameter model of photovoltaic cell based on STC data and dimensionless," in *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, 2012, pp. 1–5, doi: 10.1109/EEEI.2012.6377079.

[87] A. Orioli and A. Di Gangi, "A procedure to calculate the five-parameter model of crystalline silicon photovoltaic modules on the basis of the tabular performance data," *Appl. Energy*, vol. 102, pp. 1160–1177, Feb. 2013, doi: 10.1016/J.APENERGY.2012.06.036.

[88] "National Renewable Energy Laboratory (NREL) Home Page | NREL." [Online]. Available: https://www.nrel.gov/. [Accessed: 29-Jan-2022]

[89] "CS6A-170 PE DataSheet." 2009 [Online]. Available: www.canadian-solar.com. [Accessed: 30-Jan-2022]

[90] "G5CA PCB Power Relay Flat Relays that Switch 10A/15A Loads Power" [Online]. Available: https://www.mouser.es/ProductDetail/Omron-Electronics/G5CA-1A-E-DC5?qs=nNiD76Ca0%252BtQDYYKdX7WhQ%3D%3D. [Accessed: 30-Jan-2022]

[91] N. N. Thanh and N. P. Quang, "SIMULATION OF RECONFIGURATION SYSTEM USING MATLAB-SIMULINK ENVIRONMENT," *J. Comput. Sci. Cybern.*, vol. 34, no. 2, pp. 127–143, Oct. 2018, doi: 10.15625/1813-9663/34/2/9194.

[92] S. Masri, N. Mohamad, and M. H. M. Hariri, "Design and development of DC-DC buck converter for photovoltaic application," in *2012 International Conference on Power Engineering and Renewable Energy (ICPERE)*, 2012, pp. 1–5, doi: 10.1109/ICPERE.2012.6287236.

[93] S. Khader, "Design and simulation of a chopper circuits energized by Photovoltaic modules," in *2011 IEEE GCC Conference and Exhibition (GCC)*, 2011, pp. 73–76, doi: 10.1109/IEEEGCC.2011.5752631.

[94] B. Hauke, "Basic Calculation of a Buck Converter's Power Stage," 2011 [Online]. Available: https://www.ti.com/lit/an/slva477b/slva477b.pdf?ts=1643555553319&ref_url=https%253A%252F%252Fwww.google.com%252F. [Accessed: 31-Jan-2022]

[95] S. Anand and B. G. Fernandes, "Optimal voltage level for DC microgrids," in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, 2010, pp. 3034–3039, doi: 10.1109/IECON.2010.5674947.

[96] "IRF5210PbF HEXFET ® Power MOSFET" [Online]. Available: https://www.mouser.es/datasheet/2/196/Infineon_IRF5210_DataSheet_v01_01_EN-1228304.pdf. [Accessed: 07-Feb-2022]

[97] I. Semiconductor GmbH, "DSSS 30-01 AR," 2005 [Online]. Available: https://www.mouser.com/datasheet/2/240/L630-1547898.pdf. [Accessed: 07-Feb-2022]

[98] "ZedBoard | Avnet Boards." [Online]. Available: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/. [Accessed: 02-Nov-2021]

[99] MathWorks, "Getting Started with Targeting Xilinx Zynq Platform - MATLAB & Simulink - MathWorks España." [Online]. Available: https://es.mathworks.com/help/hdlcoder/ug/getting-started-with-hardware-software-codesign-workflow-for-xilinx-zynq-platform.html. [Accessed: 02-Nov-2021]

[100] MathWorks, "Custom IP Core Generation - MATLAB & Simulink - MathWorks España." [Online]. Available: https://es.mathworks.com/help/hdlcoder/ug/custom-ip-core-generation.html. [Accessed: 02-Nov-2021]

[101] MathWorks, "Processor and FPGA Synchronization - MATLAB & Simulink - MathWorks España." [Online]. Available: https://es.mathworks.com/help/hdlcoder/ug/processor-and-fpga-synchronization.html. [Accessed: 02-Nov-2021]

[102] Xilinx Inc, "System Integrated Logic Analyzer v1.1 LogiCORE IP Product Guide" [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/system_ila/v1_1/pg261-system-ila.pdf. [Accessed: 14-Nov-2021]

[103] X. Pan, C. Zivkovic, and C. Grimm, "Virtual prototyping of heterogeneous automotive applications: matlab, SystemC, or both?," 2019, pp. 544–549, doi: 10.1145/3287624.3287629.

[104] M. Ricco, P. Manganiello, G. Petrone, E. Monmasson, and G. Spagnuolo, "FPGA-based implementation of an adaptive P&O MPPT controller for PV applications," in *2014 IEEE 23rd*

International Symposium on Industrial Electronics (ISIE), 2014, pp. 1876–1881, doi: 10.1109/ISIE.2014.6864901.

Anexo

1. Configuración de la ZedBoard

En este apartado se explica paso a paso como configurar la ZedBoard y el sistema operativo Linux para utilizar el codiseño *hardware/software* ofrecido por MATLAB/Simulink. Es importante tener en cuenta que, para poder llevar a cabo este flujo de diseño en la red del IUMA, MATLAB se tiene que ejecutar de forma local en la estación de trabajo. Además, la estación de trabajo tiene que contar con al menos 2 tarjetas de red.

El conexionado de la ZedBoard con la estación de trabajo se ilustra en la Figura 87. Al realizar el conexionado, el interruptor de encendido de la ZedBoard tiene que estar en OFF.

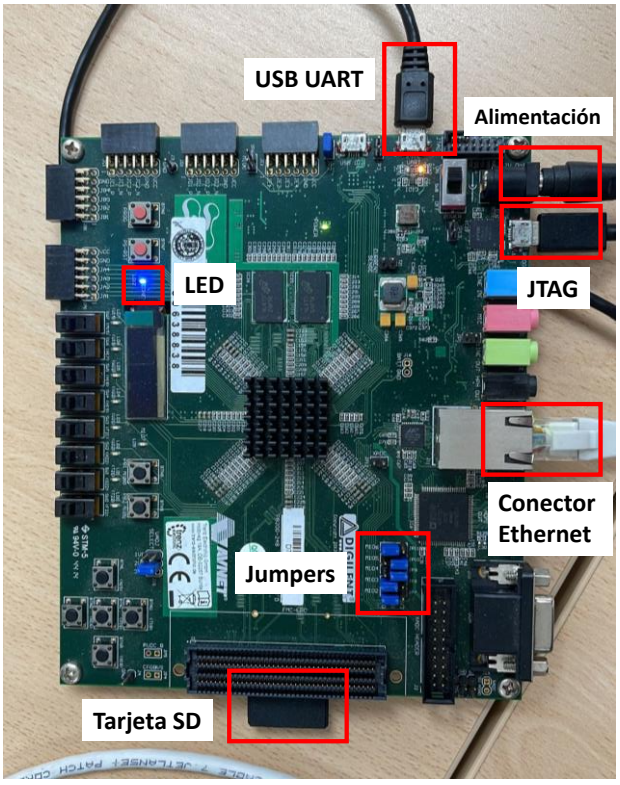


Figura 87. Conexionado de la ZedBoard con la estación de trabajo.

Los *jumpers* de la ZedBoard se tienen que configurar como se ilustra en la Figura 88. Esta configuración permite arrancar el sistema operativo Linux desde la tarjeta SD.

Figura 88. Configuración de los jumpers de la ZedBoard.

Una vez la ZedBoard está conectada a la estación de trabajo, hay que cargar en la tarjeta SD el sistema operativo. Para ello hay que seguir los siguientes pasos:

1) En la pestaña de *Add-Ons* de MATLAB hay seleccionar la opción *Manage Add-Ons.*
2) En la ventana que se despliega, seleccionar el botón resaltado en la Figura 89.

Figura 89. Ventana Manage Add-Ons.

3) En el *Hardware Setup* se selecciona la placa de desarrollo a utilizar (Figura 90) y se pulsa el botón next.

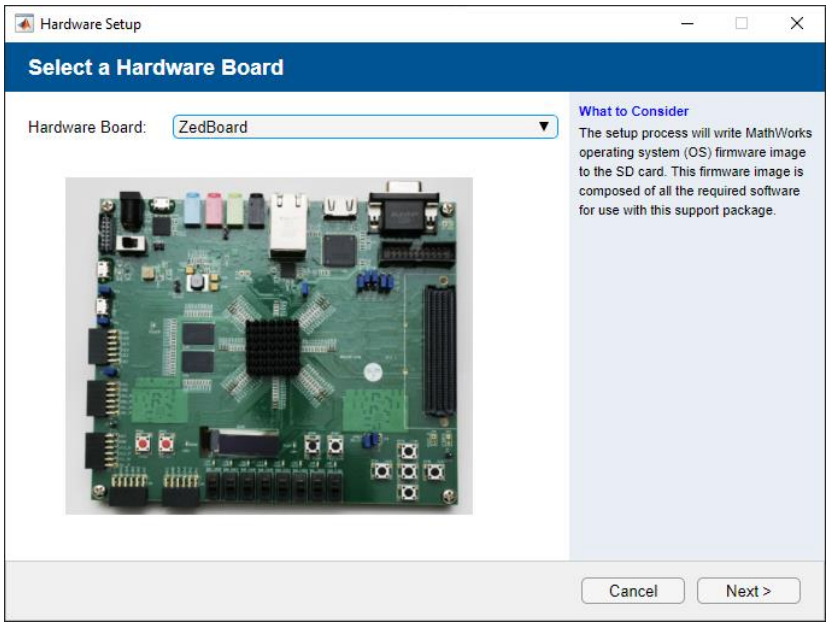


Figura 90. Hardware Setup: elección de la placa de desarrollo.

4) En la siguiente ventana se selecciona la configuración de red del sistema operativo. La *IP Address* es la dirección IP que se le asigna a la placa y el *default gateway* la dirección IP de la tarjeta de red a la que está conectada (Figura 91). Una vez se rellenan todos los campos, se pulsa el botón next.

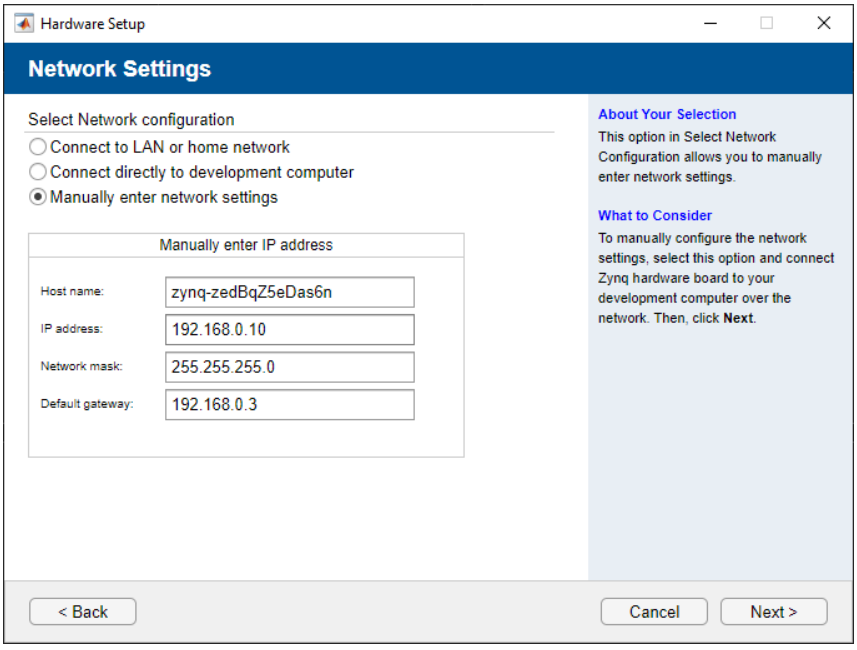


Figura 91. Hardware Setup: configuración de red del sistema operativo.

5) Se carga la imagen del sistema operativo con la configuración definida en la tarjeta SD (Figura 92).

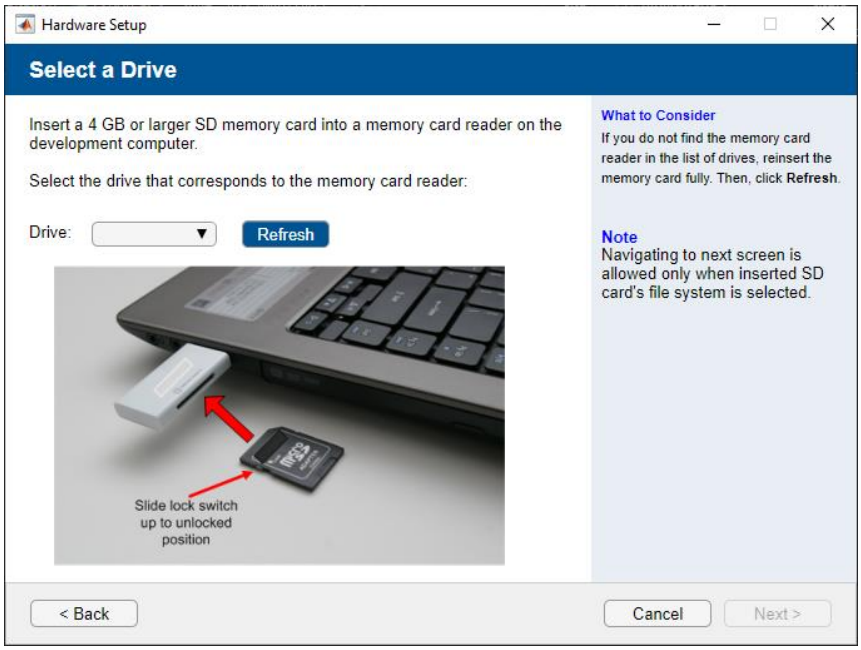


Figura 92. Hardware Setup: carga de la imagen en la tarjeta SD.

Una vez se ha cargado el sistema operativo en la tarjeta SD, se introduce en la ranura correspondiente en la ZedBoard y se enciende la placa. Si al encender la placa el LED resaltado en la Figura 87 no está encendido de color azul, significará que no se ha arrancado correctamente el sistema operativo. En este caso, mediante un programa de comunicación serie como por ejemplo minicom, se accede a la placa y se ejecuta el comando *boot*. Tras esto, el LED debería estar encendido de color azul.

El último paso para comprobar que la configuración de la ZedBoard ha sido correcta, es ejecutar el comando zynq desde el terminal de MATLAB. Para el caso de la ZedBoard con la configuración de red mostrada en la Figura 91, el comando quedaría de la siguiente forma:

$$z = zynq('linux','192.168.0.10','root','root')$$

Si el objeto z se crea correctamente, la configuración de la ZedBoard ha sido un éxito.

2. Generación y verificación de bloques IP

En este apartado se presenta el proceso de generación y verificación de bloques IP utilizando la herramienta HDL Workflow Advisor. Se utiliza como ejemplo el proceso de generación del IP MPPT_Inc.

1) Crear un modelo en Simulink que funcione como *testbench* en el cual, el algoritmo para el que se quiere generar un bloque IP, tiene que estar contenido en un *atomic subsystem* (Figura 93).

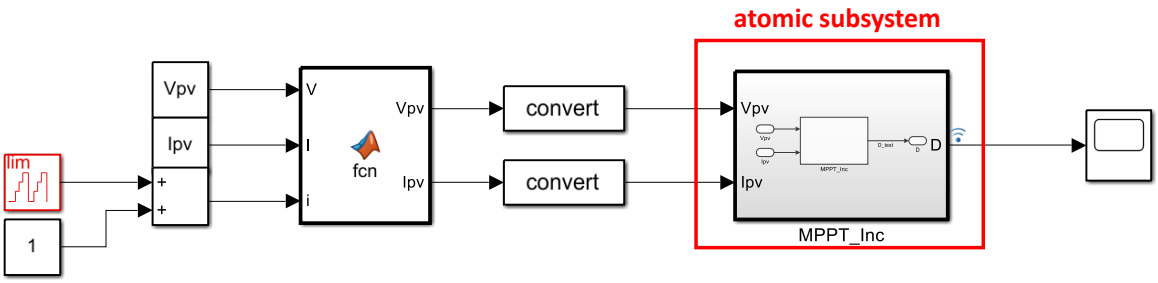


Figura 93. Testbench del controlador MPPT.

2) Click derecho en el *subsystem* que contiene al algoritmo y en la opción HDL Coder se selecciona la opción HDL Workflow Advisor (Figura 94).

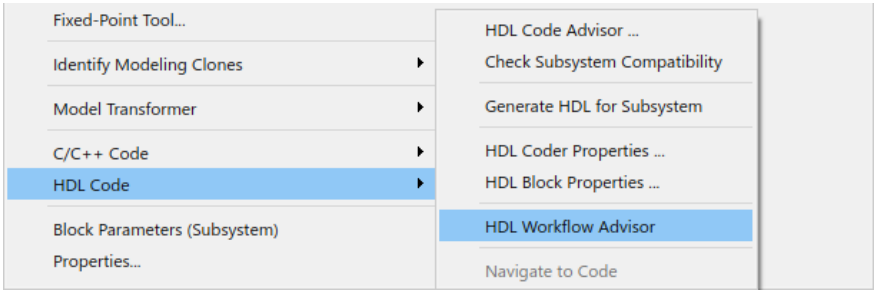


Figura 94. Lanzamiento de la herramienta HDL Workflow Advisor.

3) Seleccionar la placa de desarrollo y el nombre del proyecto (Figura 95).

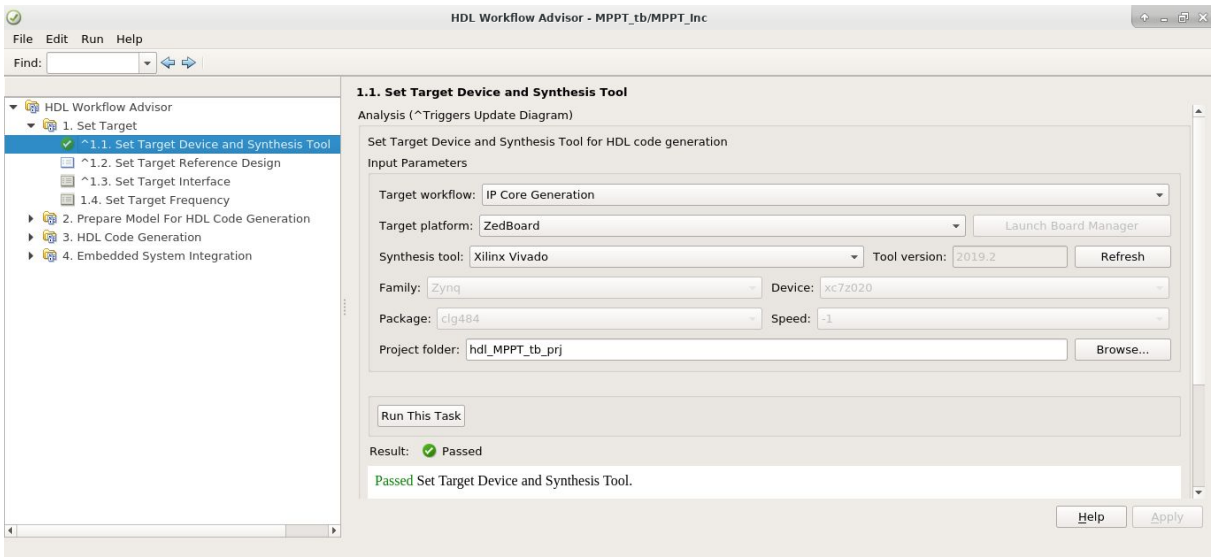


Figura 95. Elección de la placa de desarrollo y nombre del proyecto.

4) Seleccionar el diseño de referencia (Figura 96).

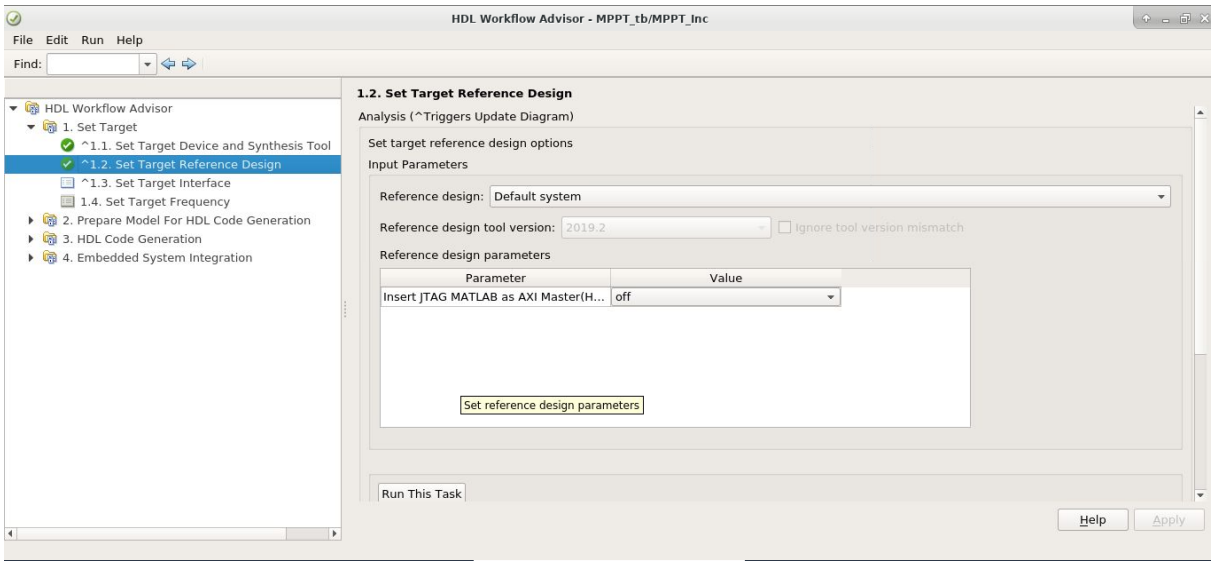


Figura 96. Selección del diseño de referencia.

5) Definir el modo de funcionamiento del bloque IP y las interfaces de entrada y salida de datos (Figura 97).

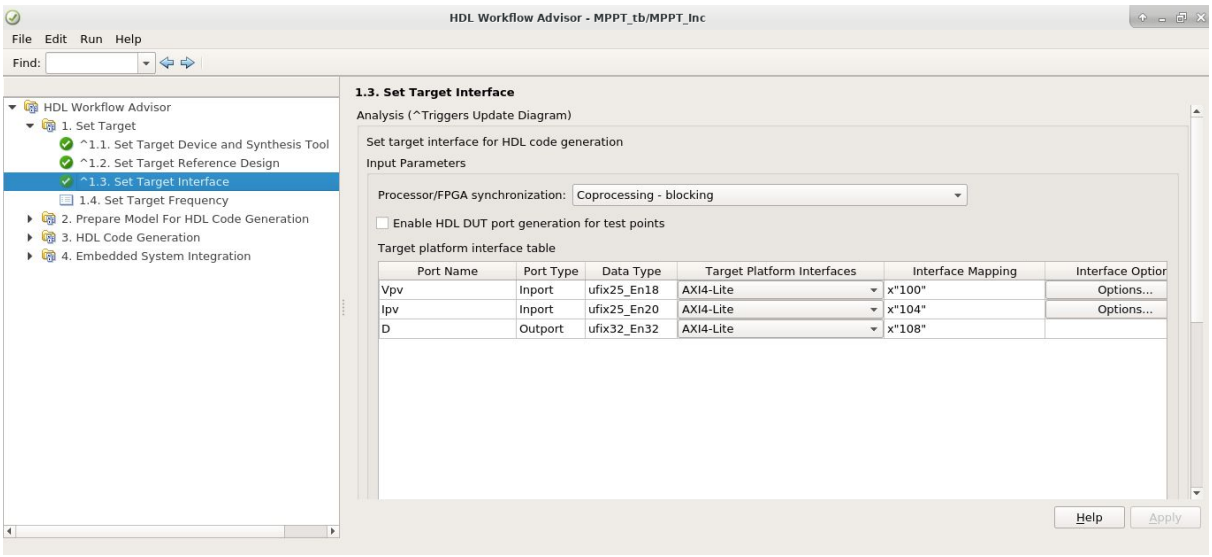


Figura 97. Selección del modo de funcionamiento e interfaz de entrada/salida del bloque IP.

6) Seleccionar la frecuencia de funcionamiento del bloque IP (Figura 98).

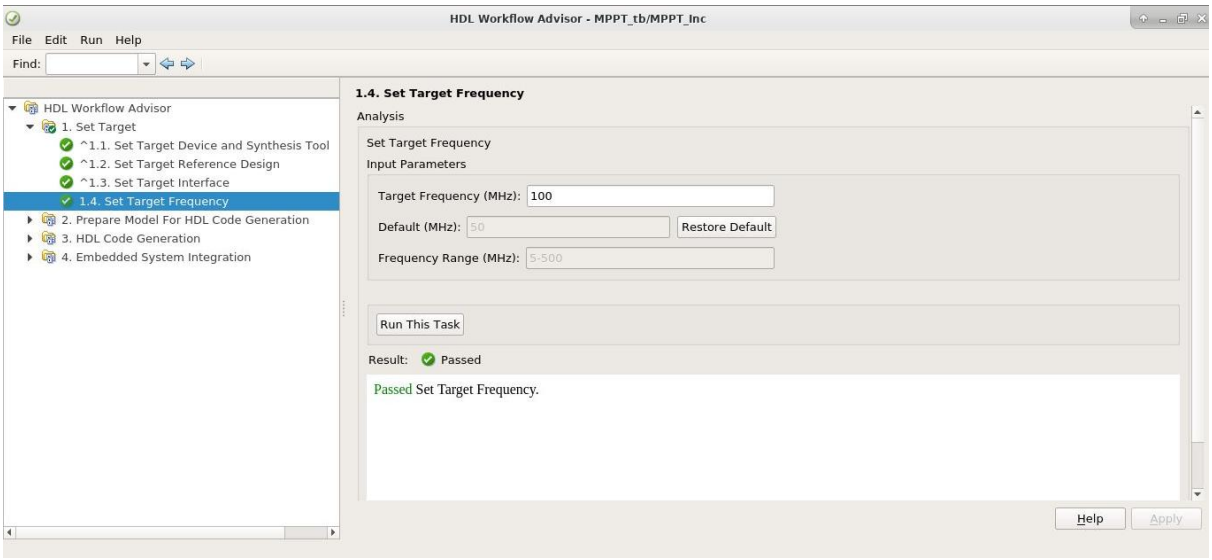


Figura 98. Selección de la frecuencia de funcionamiento.

7) Click derecho en 3.2. del apartado 3. *HDL Code Generation* y seleccionar *Run to Selected Task* (Figura 99). Cuando se ejecuten todas las tareas aparecerá un informe del bloque IP generado incluyendo el código HDL, consumo de recursos, direcciones de las interfaces, etc.

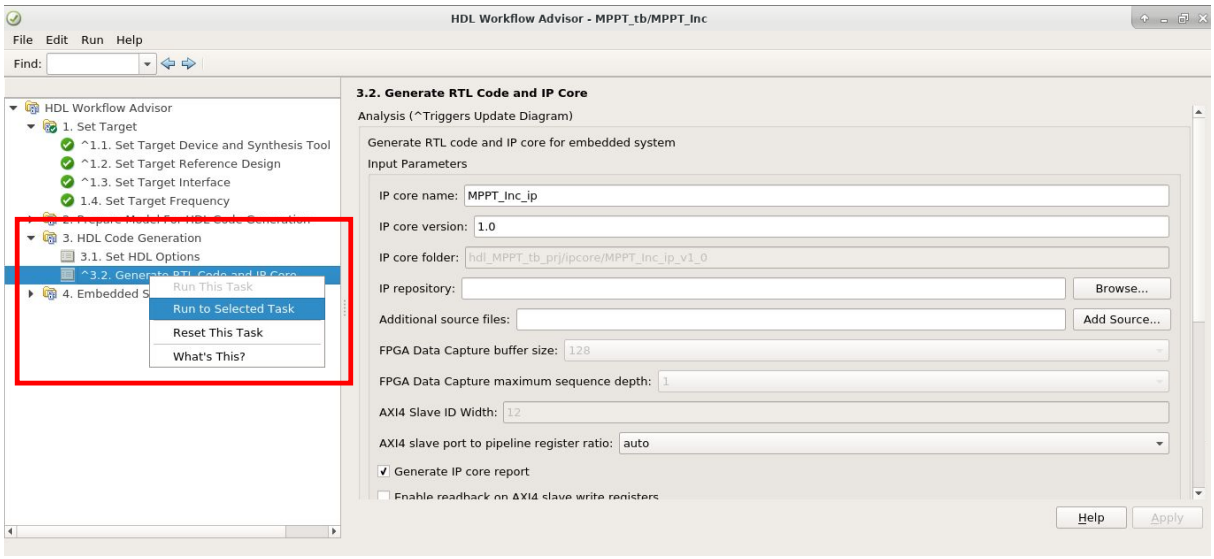


Figura 99. Ejecución ítem 3.2 del HDL Workflow Advisor.

8) Ejecutar 4.1. Create Project para generar un proyecto de Vivado Design Suite que contenga un diagrama de bloques con el IP generado para IP Integrator.

9) Generar el modelo *software interface* para Simulink ejecutando el ítem 4.2. En este modelo se sustituye el *atomic subsystem* por bloques de escritura y lectura de interfaces AXI (Figura 100).

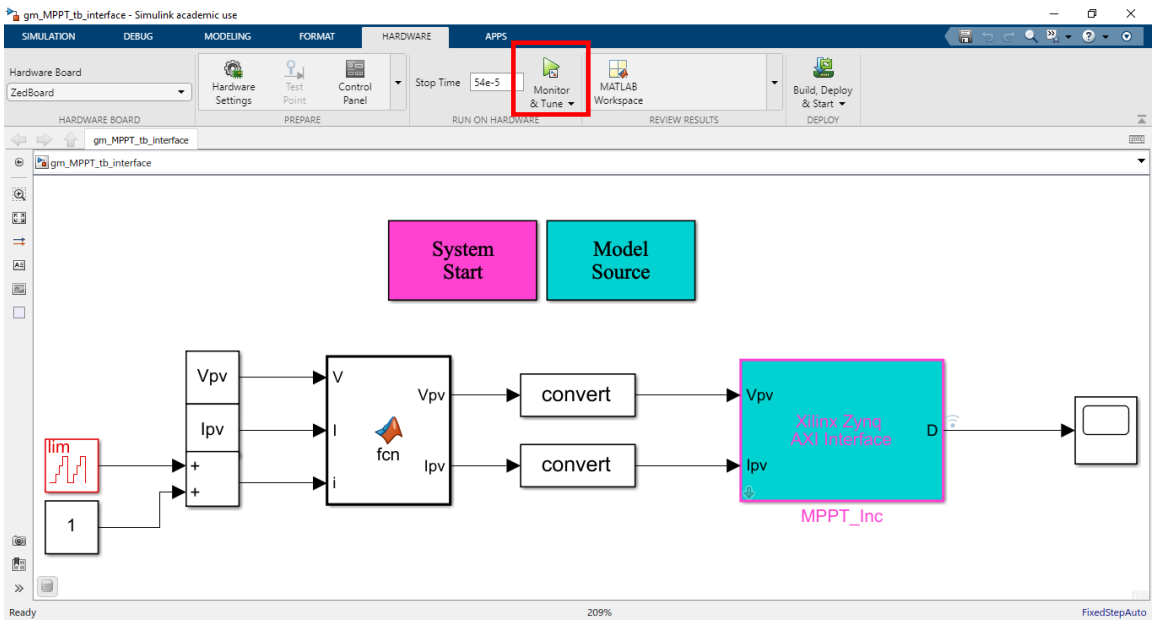


Figura 100. Modelo software interface generado.

10) Abrir el proyecto de Vivado Design Suite creado, generar el bitstream y programarlo en la placa.

11) Lanzar la simulación del modelo *software interface* con la opción *Monitor & Tune*. Esta es la opción remarcada en rojo en la Figura 100.
12) Observar los datos provenientes del bloque IP mediante un bloque *scope* o el *Data Inspector* de Simulink y verificar el correcto funcionamiento del bloque IP generado.

3. Creación de driver en Linux

Para crear nuevas instancias del driver *mwipcore* en el Linux de la ZedBoard, hay que seguir los siguientes pasos:

1) Acceder mediante un programa de comunicación serie, como por ejemplo minicom, al Linux de la ZedBoard.
2) Localizar el directorio en el que se encuentra el archivo devicetree.dtb utilizando por ejemplo el comando:

$$find - name\ "*dtb".$$

3) Crear el archivo editable devicetree.dts usando el comando:

$$\mathrm{dtc}\ -\mathrm{I}\ \mathrm{dtb}\ -\mathrm{O}\ \mathrm{dts}\ -\mathrm{f}\ \mathrm{mnt/devicetree.dtb}\ -\mathrm{o}\ \mathrm{devicetree.dts}$$

4) Modificar el archivo devicetree.dts, utilizando por ejemplo el editor vi, creando una nueva instancia del *driver mwipcore* a partir de la instancia que viene por defecto (*mwipcore0*).
5) Añadir en el apartado __symbols__ del archivo devicetree.dts las referencias a las nuevas instancias creadas.
6) Regenerar el archivo devicetree.dtb utilizando el comando:

$$\mathrm{dtc}\ -\mathrm{I}\ \mathrm{dts}\ -\mathrm{O}\ \mathrm{dtb}\ -\mathrm{f}\ \mathrm{devicetree.dts}\ -\mathrm{o}\ \mathrm{mnt/devicetree.dtb}$$

7) Reiniciar el sistema operativo con el comando *reboot*.
8) Acceder al directorio dev/ y comprobar si están las nuevas instancias del *driver* creadas.