Universidad Zaragoza
1542

Trabajo Fin de Máster

Alternativas de implementación de un controlador digital en FPGA para un convertidor Buck en prácticas de laboratorio

Alternatives for implementation of a digital controller in FPGA for a Buck converter in laboratory exercises

Autor:

Adrián Marco Artigas

Director:

José Ignacio Artigas Maestre

ESCUELA DE INGENIERÍA Y ARQUITECTURA

Zaragoza, Septiembre 2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Adrián Marco Artigas,

con nº de DNI 76971691Y en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Máster, (Título del Trabajo) Alternativas de implementación de un controlador digital en FPGA para un convertidor Buck en prácticas de laboratorio,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 20 de septiembre de 2016.

Fdo: Adrián Marco Artigas.

Alternativas de implementación de un controlador digital en FPGA para un convertidor Buck en prácticas de laboratorio

RESUMEN

El conocimiento de nuevas técnicas por parte de los estudiantes y futuros ingenieros es fundamental para que puedan desenvolverse de una manera más cómoda y eficiente en su futuro como diseñadores hardware. La presentación de las nuevas técnicas y recursos durante la fase de estudios son clave para abordar este planteamiento.

Este trabajo se centra en la exploración de alternativas actuales de implementación para un controlador en VHDL. Se trata de explorar nuevas opciones y recursos actuales que las nuevas herramientas de desarrollo incorporan para facilitar el trabajo al diseñador. Como base se han tenido en cuenta varios artículos de investigación, elaborados por profesores de la escuela, en los que estas alternativas habían sido planteadas.

Para la realización del trabajo se ha utilizado la implementación del control de un convertidor electrónico de potencia Buck. Este controlador, que será implementado sobre una FPGA, ha sido diseñado de varias maneras. Primero, se ha utilizado la técnica tradicional con el planteamiento de las operaciones coma fija. Con ayuda de librerías de coma fija y coma flotante se han desarrollado versiones adicionales. Gracias a los bloques IP de coma flotante se ha realizado una alternativa a la librería float. Por último, se ha implementado el controlador en un microcontrolador que puede ser embebido dentro de la FPGA, haciendo posible la programación de las operaciones utilizando código C.

Índice

Índice de figuras

Índice de tablas

Índice de ecuaciones

Lista de acrónimos

ADC: Analog to Digital Converter.

ASIC: Application-Specific Integrated Circuit.

BRAM: Block RAM.

BUFG: Global clock Buffer.

DSP: Digital Signal Processor.

FF: Flip-Flop.

FPGA: Field Programmable Gate Array.

FPU: Floating Point Unit.

HDL: Hardware Description Language.

IDE: Integrated Development Environment.

IEEE: Institute of Electrical and Electronics Engineers.

IO: Input Output resource.

IP: Intellectual Property.

LUT: LookUp Table.

MAC: Multiply–Accumulate.

MCS: Micro-Controller System.

PLA: Programmable Logic Array.

PWM: Pulse Width Modulator.

RAM: Random Access Memory.

RTL: Register-Transfer Level.

SDK: Software Development Kit.

TFM: Trabajo Final de Máster.

WNS: Worst Negative Slack.

1 Introducción

El continuo avance de la tecnología obliga a los ingenieros a conocer y adquirir las destrezas necesarias acorde a las herramientas hardware y software más actuales. El conocimiento de nuevas herramientas y posibilidades es esencial para reducir los costes y el tiempo de puesta en marcha de las aplicaciones. Presentar algunas de ellas a los futuros ingenieros durante su formación facilita este planteamiento.

Dentro del marco de las FPGA (Field Programmable Gate Array) y la electrónica de potencia, este TFM estudia algunas de las alternativas actuales que ofrecen los fabricantes para acelerar los tiempos de diseño. Se aplicarán diferentes alternativas en la implementación de un controlador para un convertidor Buck.

1.1 Estado de la técnica

Los dispositivos programables, tales como los PLA (Programmable Logic Array), han estado presentes desde la década de los 70. Sin embargo, su uso estaba muy limitado por razones tecnológicas. Años después, la utilización de grandes matrices de puertas lógicas configurables daba lugar al concepto de FPGA (Field Programmable Gate Array).

El elevado precio de los circuitos integrados de aplicación específica ASIC (Application-Specific Integrated Circuit), combinado con los avances en fabricación de semiconductores, hicieron de la FPGA un dispositivo idóneo para un creciente número de nuevas aplicaciones. Se convertían entonces en una alternativa a los ASIC, así como una solución para rápido prototipado de sistemas. Sin embargo, no proporcionaban la cantidad de recursos deseables, y las herramientas software no habían alcanzado todavía el suficiente desarrollo como para optimizar los diseños.

Fue en la década de los 90 cuando las FPGAs alcanzaron el nivel de madurez que las convertirían en idóneas para su aplicación en numerosos ámbitos. A finales de esta década, la industria de las FPGA traía el potencial suficiente de recursos que permitían su incorporación en campos como las comunicaciones o el procesado de señal.

Aun así, estos dispositivos seguían teniendo numerosas limitaciones, una de ellas estaba relacionada con la reducida cantidad de recursos necesarios para implementar operaciones en coma flotante. Como consecuencia, los diseñadores tenían que readaptar su trabajo a coma fija. Las FPGAs actuales incluyen bloques sumadores y multiplicadores capaces de operar a la misma velocidad que los operadores en coma fija. También incluyen otras utilidades a destacar, como bloques analógicos, procesadores ARM integrados, memorias RAM, o interfaces de comunicación entre dispositivos. Otro avance de las nuevas herramientas de desarrollo es la capacidad de embeber un procesador en el diseño, que es implementado en la FPGA utilizando su memoria RAM y bloques lógicos.

En 2013, Xilinx lanzaba al mercado una nueva herramienta de diseño para FPGA, Vivado Design Suite. Vivado surgió para ser utilizada con las nuevas FPGA de la serie 7. Al igual que su predecesora, ISE Design Suite, permite a los desarrolladores sintetizar los diseños, ejecutar análisis de propagación de señal, examinar los diagramas RTL, simular los diseños y programar

los dispositivos. La herramienta introduce una nueva metodología de diseño enfocada a incrementar la productividad del desarrollador. Está fuertemente enfocada a diseños basados en bloques IP y también permite la creación de nuevos bloques a partir de código C/C++.

Puede decirse que, en la actualidad las FPGA cuentan con una gran variedad de recursos y los tiempos de desarrollo se han acelerado notablemente, por lo que se convierten en una excelente alternativa para su inclusión en gran cantidad de aplicaciones industriales; como las comunicaciones, control de movimiento, electrónica de potencia, aeroespacial, etc. Extraído de [1], [2] y [3].

1.2 Objetivo

El objetivo de este trabajo final de máster consiste en explorar algunas alternativas de implementación que ofrecen los nuevos entornos de desarrollo (IDE) y que se proponen en [4]. Estas son las librerías de coma fija y coma flotante, bloques IP y soft-core. No se han considerado soluciones de generación automática de código, como las basadas en herramientas de Matlab (HDL Coder, XSG) o las herramientas de síntesis de alto nivel (de C a HDL).

1.3 Metodología

Se desarrollarán varias alternativas para implementar un mismo controlador. Las diferentes alternativas se validarán mediante simulación. Esta simulación permitirá comprobar el funcionamiento del sistema completo, planta + control en lazo cerrado. Tanto el modelo matemático del convertidor electrónico de potencia (planta), como su control, se han implementado sobre la plataforma Vivado Design Suite.

Las diferentes alternativas también se probarán en un sistema real con el fin de validar que las implementaciones sobre FPGA corresponden con los resultados obtenidos en simulación. Para ello se ha utilizado una placa de desarrollo de Digilent y un prototipo de convertidor Buck. La placa elegida es la Basys3, que incluye una FPGA de la serie 7 de Xilinx.

1.4 Herramientas

Para la realización de este trabajo final de máster se han empleado dos tipos de herramientas. Una herramienta software, para diseño y simulación y herramientas hardware, para su comprobación en un sistema real.

La herramienta software es Vivado Design Suite 2015.3, un IDE (Integrated Development Environment) de última generación. Ofrece un nuevo enfoque para aumentar la productividad ofreciendo ventajas basadas en tres aspectos, aceleración de la fase de diseño, entorno de simulación y optimización de la implementación. La aceleración del diseño es posible gracias a una nueva metodología basada en la generación de bloques IP y la integración de bloques IP predefinidos. En cuanto a simulación, Vivado incluye un simulador de lenguaje mixto y un entorno de programación y depurado. Por último, Vivado es capaz de optimizar la implementación del diseño hasta un 20% y mejorar del rendimiento y consumo en la FPGA.

Las herramientas hardware son el convertidor electrónico de potencia Buck y la placa de control con FPGA. El convertidor Buck es un prototipo utilizado en prácticas de laboratorio,

cuyo modelo matemático se ha descrito en VHDL para su simulación. El prototipo también monta un conversor ADC, que medirá la tensión de salida del convertidor.

La placa de control es la Basys3 de Digilent. Se trata de una tarjeta diseñada exclusivamente para Vivado Design Suite que incorpora una FPGA de arquitectura Artix®7 de Xilinx. La Basys3 incluye periféricos, una gran colección de dispositivos de E/S, todos los circuitos de soporte FPGA requeridos, y una versión libre de las herramientas de desarrollo. Las características de esta placa pueden consultarse en los anexos de este documento.

Fig. 1. Placa de evaluación Basys3.

1.5 Organización del documento

El documento se desarrolla de la siguiente manera. En el punto 2 se expone como punto de partida el circuito electrónico del convertidor electrónico de potencia junto al controlador digital utilizado y se da a conocer la base sobre la cual se ha trabajado en este TFM.

En el punto 3 se desarrolla el grueso del trabajo. En él se explican todas las alternativas de implementación estudiadas:

1. Implementación en coma fija.
2. Implementación en coma fija con fixed_pkg.
3. Implementación en coma flotante con float_pkg.
4. Implementación en coma flotante con float_pkg, sencilla.
5. Implementación con bloques IP float.
6. Implementación con MicroBlaze.

El punto 4 recoge los resultados de simulación y validación experimental obtenidos en cada alternativa.

Por último, en el punto **¡Error! No se encuentra el origen de la referencia.**, se exponen las conclusiones y las líneas de trabajo futuras.

Los diseños completos pueden consultarse en los anexos.

2 El convertidor Buck en lazo cerrado

En este bloque se presenta el modelo matemático del convertidor junto con el controlador utilizado. El circuito del convertidor y su controlador han sido recopilados de la asignatura "Control Digital con FPGA de Etapas de Potencia" y están explicados en [4]. El modelo del convertidor fue obtenido por el autor de este TFM como parte del trabajo de dicha asignatura. El tipo de controlador y sus coeficientes fueron proporcionados como punto de partida para su posterior implementación en una FPGA.

2.1 Circuito

El esquema simplificado del convertidor Buck + control se muestra en la Fig. 2.

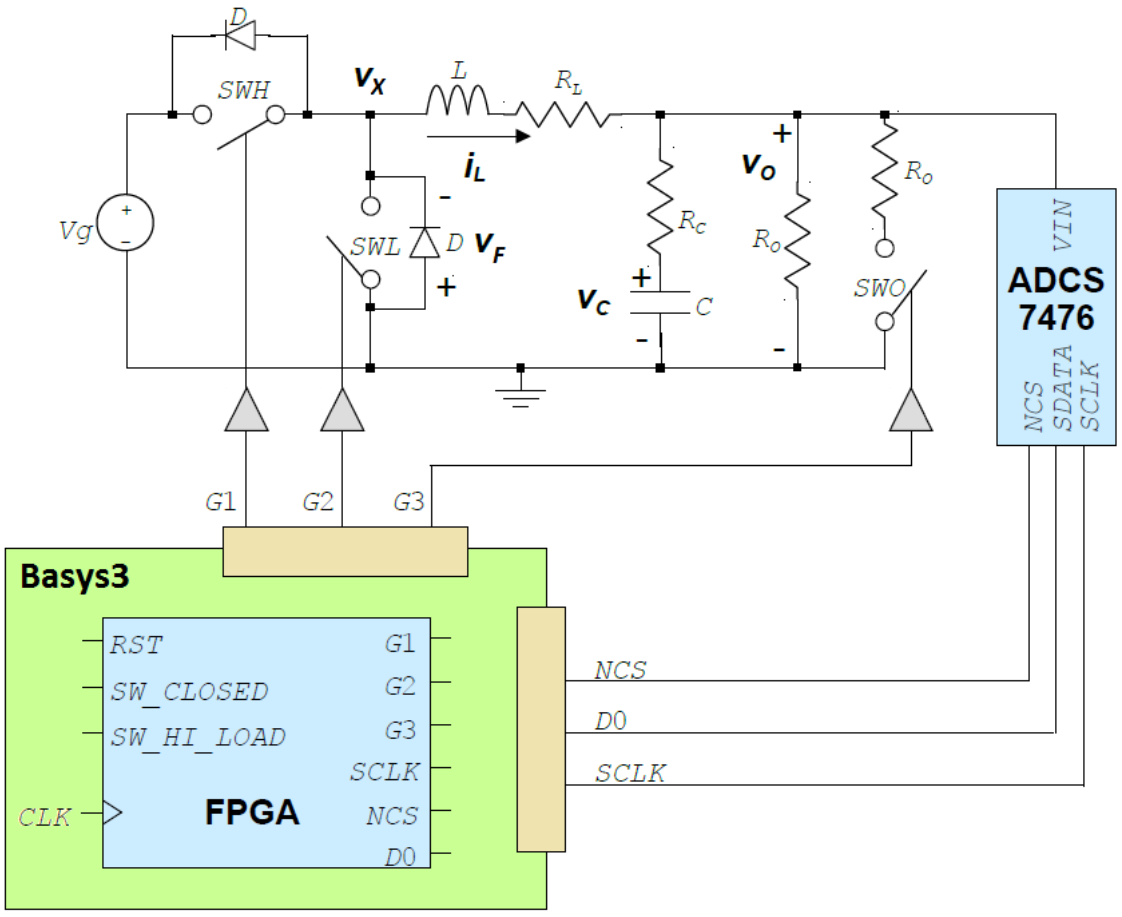


Fig. 2. Esquema completo, convertidor + ADC + placa de desarrollo

Los parámetros de este convertidor son los siguientes:

- Tensión de entrada, V_g = 5 V (rango de 3 a 6 V).
- Tensión de salida, V_o = 2.5 V.
- L = 68 µH, R_L = 98 mΩ.
- C = 220 µF, R_C = 80 mΩ.
- R_o = 5 Ω.
- Frecuencia de conmutación, f_{sw} = 100 kHz.
- Frecuencia de muestreo, f_{ADC} = 100 kHz.

La tensión de salida V_o, se conecta a través de un filtro antialiasing de ganancia 1 al ADC, cuyo rango va de 0 a 3.3 V.

La resolución del PWM es de 9 bits y la del ADC de 8 bits. Con una resolución PWM mayor que la del ADC se asegura que no haya ciclos límite.

El interruptor SW_HI_LOAD se utiliza para provocar cambios en la carga del convertidor, mientras que el interruptor SW_CLOSED se utiliza para habilitar el controlador o utilizar una modulación fija. G2 no se utiliza y SWL permanece siempre abierto. El botón RST resetea todo el sistema implementado en la FPGA.

La frecuencia de reloj de la FPGA es de 100 MHz, el diseño se ha planteado para funcionar a 50 MHz. El esquema completo del convertidor Buck puede consultarse en los anexos.

2.2 Modelo del convertidor para simulación

Para poder llevar a cabo una simulación, es necesario implementar un modelo matemático del convertidor y de los componentes del lazo de control. Estos modelos se han desarrollado en un test bench utilizando Vivado.

El modelo matemático del convertidor se ha obtenido a través de sus ecuaciones de estado, que son:

- q1: SWH ON, D en corte.
- q2: SWH OFF, D conduce.
- q3: SWH OFF, D en corte.

Se muestran en la Ecuación 1 donde V_X es la tensión V_G o $-V_F$ (tensión directa del diodo), dependiendo del estado de los interruptores.

$$\frac{d}{dt}\begin{pmatrix} i_L \\ v_c \end{pmatrix} = \begin{pmatrix} -\frac{1}{L}\left(\frac{R_L R_O + R_L R_C + R_C R_O}{R_O + R_L}\right) & -\frac{1}{L}\frac{R_O}{R_C + R_O} \\ \frac{1}{C}\frac{R_O}{R_C + R_O} & -\frac{1}{C}\frac{R_O}{R_C + R_O} \end{pmatrix}\begin{pmatrix} i_L \\ v_C \end{pmatrix} + \begin{pmatrix} \frac{1}{L} \\ 0 \end{pmatrix} v_x;$$

$$v_O = \begin{pmatrix} \frac{R_C R_O}{R_C + R_O} & \frac{R_O}{R_C + R_O} \end{pmatrix}\begin{pmatrix} i_L \\ v_C \end{pmatrix};$$

Ecuación 1. Ecuaciones de estado del convertidor en tiempo continuo.

La matriz es discretizada mediante el método de "Euler hacia delante" con un tiempo de muestreo T_S = 20 ns, resultando en la matriz expresada en la Ecuación 2.

$$\frac{d}{dt}\begin{pmatrix} i_L(k+1) \\ v_c(K+1) \end{pmatrix} = \begin{pmatrix} 1-\frac{T_S}{L}\left(\frac{R_L R_O + R_L R_C + R_C R_O}{R_O + R_L}\right) & -\frac{T_S}{L}\frac{R_O}{R_C + R_O} \\ \frac{T_S}{C}\frac{R_O}{R_C + R_O} & 1-\frac{T_S}{C}\frac{R_O}{R_C + R_O} \end{pmatrix}\begin{pmatrix} i_L(k) \\ v_C(k) \end{pmatrix} + \begin{pmatrix} \frac{T_S}{L} \\ 0 \end{pmatrix} v_x;$$

$$v_O = \begin{pmatrix} \frac{R_C R_O}{R_C + R_O} & \frac{R_O}{R_C + R_O} \end{pmatrix}\begin{pmatrix} i_L \\ v_C \end{pmatrix};$$

Ecuación 2. Ecuaciones de estado del convertidor en tiempo discreto.

Finalmente, es posible plantear estas ecuaciones en código VHDL para simular el convertidor. El fragmento de código de este modelo, extraído del test bench, se muestra en la Fig. 3. El test bench completo puede consultarse en los anexos de este documento.

```
175     ------------------------------------------------------------------
176     --      MODELO DEL BUCK ECUACIONES EN DIFERENCIAS IL VO VC
177     ------------------------------------------------------------------
178     process
179         variable IL_aux : real;
180     begin
181         wait for Ts;
182         if (G1 = '1') then -- q1
183             IL_aux := IL*(1.0-(dT*(RL*RO+RL*RC+RC*RO)/(L*(RO+RC)))) -
184             VC*((dT*RO)/( (L*(RO+RC)))) + (VG*dT/L);
185         else --q2
186             IL_aux := IL*(1.0-(dT*(Rl*RO+Rl*RC+RC*RO)/(L*(RO+RC)))) -
187             VC*((dT*RO)/( (L*(RO+RC)))) - (VF*dT/L);
188         end if;
189         if (IL_aux < 0.0) then --q3
190             IL_aux := 0.0;
191         end if;
192         IL <= IL_aux;
193         VC <= (((dT/C)*(Ro/(Ro+Rc)))*iL) + vC*(1.0-((dT/C)*(1.0/(RO+RC))));
194         VO <= (((RC*RO)/(RO+RC))*iL) + ((Ro/(RO+RC))*VC);
195         HVO <= VO;
196     end process;
```

Fig. 3. Modelo del convertidor Buck en VHDL.

2.3 Controlador

Se desea obtener un ancho de banda f_c= 5 kHz y un margen de fase de 60°. Para ello se utiliza el controlador PI+ PAF, proporcionado en la asignatura "Control Digital con FPGA de Etapas de Potencia", que se muestra en la Ecuación 3.

$$C(s) = 5{,}05\frac{(s+3142)(s+15550)}{s(s+63470)}$$

Ecuación 3. Función de transferencia del controlador en tiempo continuo.

La función de transferencia del controlador en tiempo discreto se muestra en la Ecuación 4.

$$C(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Ecuación 4. Función de transferencia del controlador en tiempo discreto.

Donde (b_0, b_1, b_2) = (3,480, -7,658, 4,197) y (a_1, a_2) = (0,5182, -1,5182). Suponiendo sendas ganancias del PWM y del conversor analógico-digital iguales a 1.

El diagrama de bloques del control en lazo cerrado se muestra en la Fig. 4, donde C(z) es el controlador, PWM es el bloque modulador, Gvd es la función de transferencia del convertidor y ADC es el convertidor analógico-digital. Ref representa la consigna y Vo la tensión de salida del convertidor.

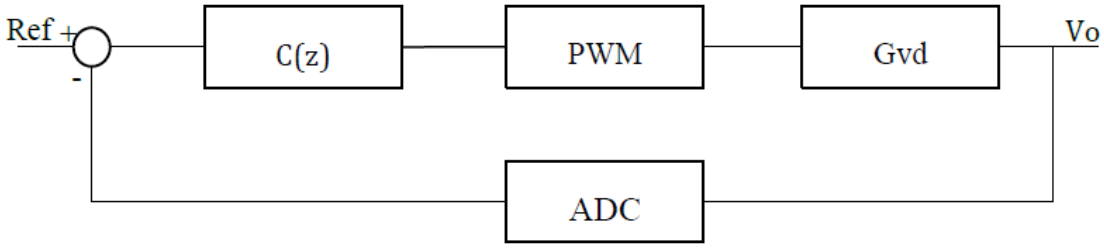


Fig. 4. Diagrama de bloques del lazo de control.

Las ganancias de los bloques PWM y ADC se reflejan en la Ecuación 5.

$$K_{PWM} = \frac{1}{500};\ K_{ADC} = \frac{2^8}{3{,}3};$$

Ecuación 5. Ganancias del lazo de control.

Para compensar estas ganancias el controlador C(z) queda multiplicado por el factor obtenido en la Ecuación 6. Lo que da lugar al controlador de la Ecuación 4 multiplicado por este factor.

$$\frac{1}{K_{PWM}}\frac{1}{K_{ADC}} = 6{,}6;$$

Ecuación 6. Ganancia extra del controlador.

3 Alternativas de implementación

Tras el punto de partida descrito en el apartado anterior, se presenta a continuación el trabajo realizado en este TFM. Aquí se exponen las diferentes alternativas estudiadas para implementar el controlador sobre una FPGA.

En la Fig. 5 se muestra el circuito digital diseñado para la FPGA, que equivale al recuadro azul de la Fig. 2. Este diagrama es el que se ha seguido para implementar todas las alternativas salvo la versión con microcontrolador.

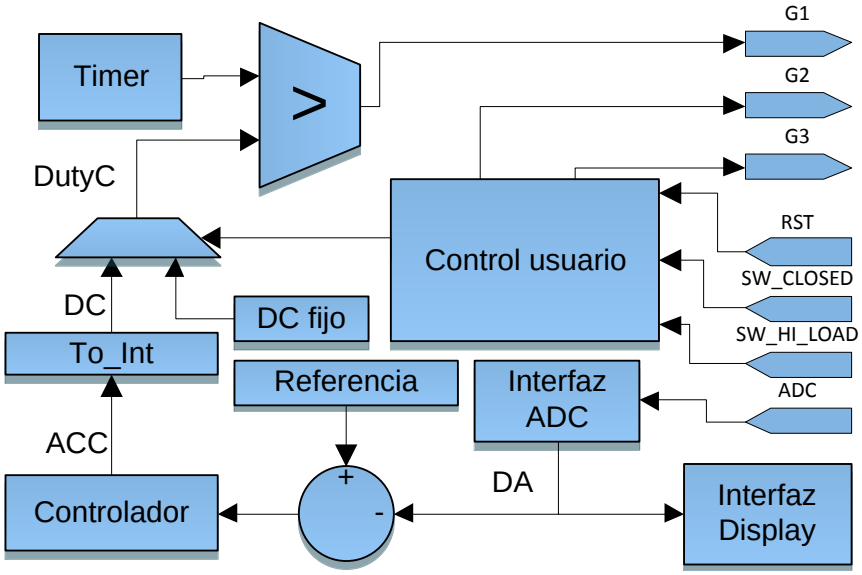


Fig. 5. Diagrama de bloques principal.

La organización de las tareas que desempeñan los diferentes bloques se realiza a través del contador llamado "Timer". De esta manera, todas ellas se repetirán en los mismos puntos cíclicamente. Como se ve en la Fig. 6, en cierto punto N1 se inicia la conversión del ADC y se espera un tiempo para obtener la respuesta. En otro punto N2 actúa el controlador y se espera de nuevo para que se efectúen los cálculos y obtener así el duty cycle. Este duty actuará a partir del ciclo siguiente.

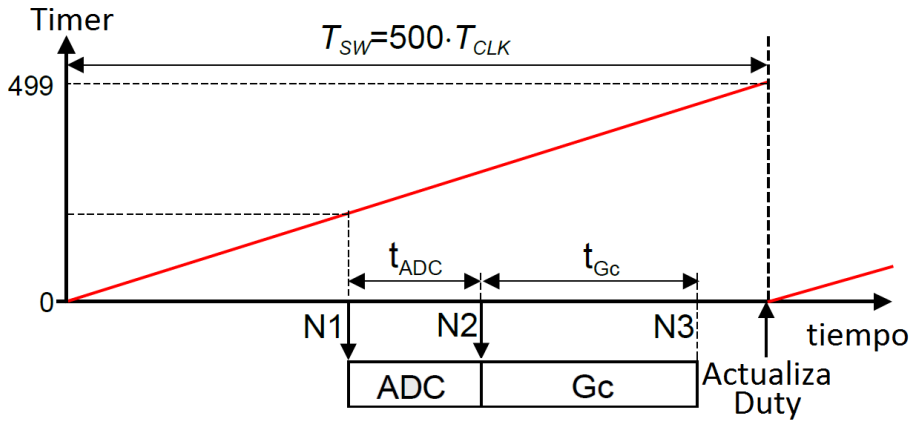


Fig. 6. Planificación temporal.

El controlador de la Ecuación 4 puede ser representado en forma de diagrama de bloques, Fig. 7, con el fin de visualizar las distintas operaciones de una manera segmentada y más intuitiva a la hora de describir el diseño en código VHDL.

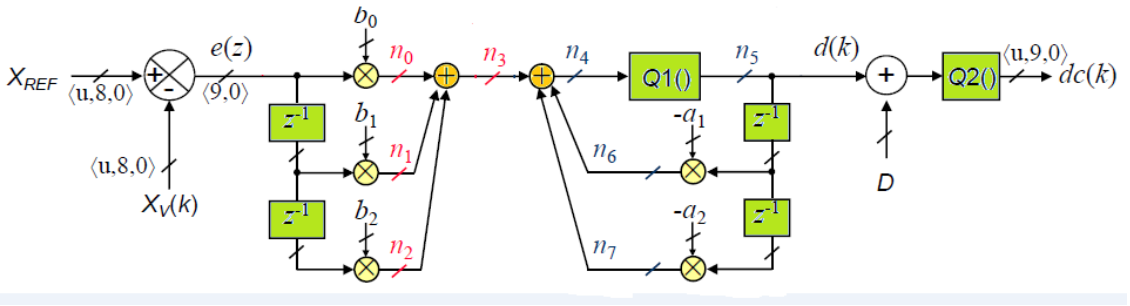


Fig. 7. Diagrama de operaciones del controlador en tiempo discreto.

Para implementar esta estructura se ha seguido un método común en todas las alternativas, la utilización de una unidad de multiplicación y acumulación (MAC). Este método consiste en distribuir las operaciones del controlador en varios ciclos de reloj, de forma que en cada ciclo se realiza una operación de multiplicación y acumulación. De este modo todas las operaciones del controlador se realizarán con el mismo hardware. En la Tabla 1 se muestran las operaciones que se seguirán con este método. La sucesión de las operaciones se ha implementado a través de una máquina de estados.

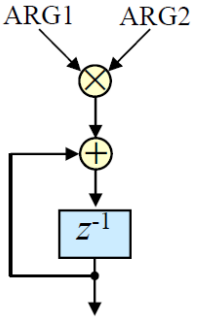
	# ciclo	ARG1	ARG2	Operación
	1	b_0	e(z)	ACC <= X*Y
	2	b_1	$e(z^{-1})$	ACC <= ACC + X*Y
	3	b_2	$e(z^{-2})$	ACC <= ACC + X*Y
	4	$-a_1$	$d(z^{-1})$	ACC <= ACC + X*Y
	5	$-a_2$	$d(z^{-2})$	ACC <= ACC + X*Y
	6			$e(z^{-1})$ <= e(z) $e(z^{-2})$ <= $e(z^{-1})$ $d(z^{-1})$ <= d(z) $d(z^{-2})$ <= $d(z^{-1})$

Tabla 1. Distribución de operaciones en la unidad MAC.

3.1 Implementación en coma fija

La representación en coma fija complemento a 2 consiste en destinar una cantidad fija de bits para la parte entera y otra para la parte fraccionaria, el primer bit queda reservado para indicar el signo. La precisión de esta representación dependerá del número de bits que se reserven a cada parte. La notación utilizada para designar el formato de un determinado número con esta representación es **<w,f>**, donde **w** es el número total de bits y **f** es el número de bits después del punto decimal. La relación matemática entre estos números es I **= w-1-f**, donde I es el número de bits que representan la parte entera y el 1 es el bit reservado para el signo.

La primera alternativa estudiada consiste en desarrollar las operaciones de números reales utilizando esta representación. Para ello, los números han de ser dimensionados manualmente. Esta implementación es muy costosa para el diseñador, requiere de un estudio meticuloso de todos los números y operaciones que intervienen en el controlador.

Las constantes se pasan a codificar en coma fija utilizando el formato <18,f>, utilizando el mismo formato para los coeficientes del numerador y el mismo formato para los coeficientes del denominador.

- b2 = 3,480*6,6 = 22,968 $I > \log_2(22{,}968) = 4{,}52$ → I=5
- b1 = -7,658*6,6 = -50,5428 $I > \log_2(50{,}5428) = 5{,}66$ → I=6
- b0 = 4,197*6,6 = 27,7002 $I > \log_2(27{,}7002) = 4{,}79$ → I=5

Por tanto, I = 6 y el formato para los tres es <18,11>.

- a2 = 0,5182 $I > \log_2(0{,}5182) = -0{,}95$ → I=0
- a1 = -1,5182 $I > \log_2(1{,}5182) = 0{,}60$ → I=1

Por tanto, I =1 y el formato para los dos es <18,16>.

Finalmente, los coeficientes codificados en sus respectivos formatos son:

- b2 = round $(22{,}968 * 2^{11}) = 47.038$
- b1 = round $(-50{,}5428 * 2^{11}) = -103.512$
- b0 = round $(27{,}7002 * 2^{11}) = 56.730$
- a2 = round $(0{,}5182 * 2^{16}) = 33.961$
- a1 = round $(-1{,}5182 * 2^{16}) = -99.497$

El nodo n5, es el único nodo que necesita ser estimado por simulación, obteniendo el formato <s,22,8>. El formato del resto de nodos se obtiene operando con los formatos de los operadores, como se refleja en la Tabla 2.

Nodo	Operación	Formatos	Formato resultado
0	b0*error0	<18,11> * <9,0>	<27,11>
1	b1*error1	<18,11> * <9,0>	<27,11>
2	b2*error2	<18,11> * <9,0>	<27,11>
3	n0+n1+n2	<27,11> + <27,11> + <27,11>	<30,11>
5	Cuantización	<42,24>	<22,8>
6	-a1*duty1	<18,16> * <22,8>	<40,24>
7	-a2*duty2	<18,16> * <22,8>	<40,24>
4	n3+n6+n7	<30,11> +<40,24> + <40,24>	<42,24>

Tabla 2. Formato de los nodos en coma fija.

En la Fig. 8 se muestra la declaración de todas las señales que intervienen en la implementación del controlador. Se utilizan los tipos **signed** o **unsigned** dependiendo del formato con signo o sin signo de las señales.

```
60      -- SEÑALES DE E/S DEL CONTROLADOR
61      signal REFERENCE: unsigned (8 downto 0):= to_unsigned(194,9); --Consigna
62      signal nDC, DC : unsigned (8 downto 0);
63      signal nDutyC, DutyC: unsigned (8 downto 0);
64      signal nREFERENCE: unsigned (12 downto 0);
65
66      -- SEÑALES PARA EL CONTROLADOR EN COMA FIJA
67      signal nACC, ACC: signed(41 downto 0) := (others => '0'); --<42,24>
68      signal nERROR0, nERROR1, nERROR2, ERROR0, ERROR1, ERROR2 : signed(8 downto 0); --<9,0>
69      signal nMUL1, MUL1,nDUTY1, nDUTY2, DUTY1, DUTY2 : signed(21 downto 0); -- <22,8>
70
71      -- CONSTANTES DEL CONTROLADOR EN COMA FIJA
72      signal nMUL2, MUL2: signed(17 downto 0);
73      constant b2 : signed(17 downto 0):= to_signed(47038,   18);     --  3.480 * 6.6 <18,11>
74      constant b1 : signed(17 downto 0):= to_signed(-103512, 18);     -- -7.658 * 6.6 <18,11>
75      constant b0 : signed(17 downto 0):= to_signed(56730,   18);     --  4.197 * 6.6 <18,11>
76      constant a2 : signed(17 downto 0):= to_signed(33961,   18);     --  0.5182      <18,16>
77      constant a1 : signed(17 downto 0):= to_signed(-99497,  18);     -- -1.5185      <18,16>
```

Fig. 8. Declaración de señales en coma fija.

Las operaciones del controlador se realizan con la máquina de estados de la Fig. 9. En cada operación de multiplicación y acumulación es necesario adaptar el formato de los diferentes resultados concatenando ceros o con la función **Resize**.

```
180         case ESTADO is
181             when REPOSO =>
182                 if (START_GC = '1') then
183                     nERROR0 <= signed(REFERENCE)- signed('0'& DA);
184                     nESTADO <= E0;
185                 end if;
186             when E0 =>
187                 nMUL1 <= ERROR0 & "0000000000000";
188                 nMUL2 <= b0;
189                 nESTADO <= E1;
190             when E1 =>
191                 nACC <= Resize(MUL1 * MUL2, 42);
192                 nMUL1 <= ERROR1 & "0000000000000";
193                 nMUL2 <= b1;
194                 nESTADO <= E2;
195             when E2 =>
196                 nACC <= Resize(MUL1 * MUL2 + ACC, 42);
197                 nMUL1 <= ERROR2 & "0000000000000";
198                 nMUL2 <= b2;
199                 nESTADO <= E3;
200             when E3 =>
201                 nACC <= Resize(MUL1 * MUL2 + ACC, 42);
202                 nMUL1 <= DUTY1;
203                 nMUL2 <= -a1;
204                 nESTADO <= E4;
205             when E4 =>
206                 nACC <= Resize(MUL1 * MUL2 + ACC, 42);
207                 nMUL1 <= DUTY2;
208                 nMUL2 <= -a2;
209                 nESTADO <= E5;
210             when E5 =>
211                 nACC <= Resize(MUL1 * MUL2 + ACC, 42);
212                 nESTADO <= E6;
213             when E6 =>
214                 --Actualizar valores
215                 nERROR1 <= ERROR0;
216                 nERROR2 <= ERROR1;
217                 nDUTY1 <= ACC(37 downto 16);
218                 nDUTY2 <= DUTY1;
219                 --Cuantizador, coge solo la parte entera
220                 if (ACC(41 downto 24) > 450) then
221                     nDC <= to_unsigned(450, 9);
222                 elsif (ACC(41 downto 24) < 50) then
223                     nDC <= to_unsigned(50, 9);
224                 else
225                     nDC <= unsigned(ACC(32 downto 24));
226                 end if;
227                 nESTADO <= REPOSO;
228         end case;
```

Fig. 9. Máquina de estados en coma fija.

3.2 Implementación en coma fija con fixed_pkg

En este apartado se estudia de nuevo la implementación del control en coma fija, esta vez con ayuda de la librería **fixed_pkg**. Para su implementación ha sido de gran ayuda lo explicado en [5].

La utilización de esta librería permite que el dimensionamiento de los coeficientes resulte mucho más legible. Las señales quedan declaradas indicando, por un lado, la cantidad de bits a la izquierda del punto y por otro, la cantidad de bits de la parte decimal. En cuanto a las operaciones, ya no es necesario tener en mente donde está la coma. Gracias al formato que presenta esta librería los operandos son alineados de forma correcta. Por último, habrá que adaptar el formato del resultado al formato de la señal destino. Esto puede hacerse cómodamente con la función **Resize**, sin necesidad de concatenar ni recortar bits, simplemente indicando el formato de la señal destino.

En la Fig. 10 se muestra la declaración de las señales del controlador. El número decimal puede introducirse directamente con la función **to_sfixed**. El formato se indica dentro de la propia función, que ha de encajar con el especificado en la declaración con el tipo **sfixed**. El primer número (en positivo) indica el número de bits reservado para la parte entera teniendo en cuenta que el primer bit a la izquierda de la coma es el bit 0. El segundo número (en negativo) indica el número de bits de la parte decimal. Un formato “6 downto -11” indicaría que hay 7 bits a la izquierda del punto y 11 bits a la derecha.

```
63      -- SEÑALES DE E/S DEL CONTROLADOR
64      signal REFERENCE: unsigned(8 downto 0):= to_unsigned(194,9); --Consigna
65      signal nDC, DC : unsigned(8 DOWNTO 0); -- Duty cuantizado
66      signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final
67      signal nREFERENCE : unsigned (12 downto 0);
68
69      -- SEÑALES PARA EL CONTROLADOR EN COMA FIJA USANDO FIXED_PKG
70      signal nACC, ACC: sfixed(17 downto -24);                                    -- 18_24
71      signal nERROR0, nERROR1, nERROR2, ERROR0, ERROR1, ERROR2: sfixed(8 downto 0);   --  9_0
72      signal nDUTY1, nDUTY2, DUTY1, DUTY2 : sfixed(13 downto -8);                 -- 14_8
73      signal nMUL1, MUL1 : sfixed(13 downto -8);
74      signal nMUL2, MUL2 : sfixed(6 downto -16);
75
76      -- CONSTANTES EN COMA FIJA USANDO FIXED_PKG
77      constant b2 : sfixed (6 downto -11) := to_sfixed (3.480   *6.6,   6,-11);   --7_11
78      constant b1 : sfixed (6 downto -11) := to_sfixed (-7.658  *6.6,   6,-11);   --7_11
79      constant b0 : sfixed (6 downto -11) := to_sfixed (4.197   *6.6,   6,-11);   --7_11
80      constant a2 : sfixed (1 downto -16) := to_sfixed (0.5182,         1,-16);   --2_16
81      constant a1 : sfixed (1 downto -16) := to_sfixed (-1.5182,        1,-16);   --2_16
```

Fig. 10. Declaración de señales en coma fija con fixed_pkg.

En la Fig. 11 se muestran las operaciones de la máquina de estados.

```
155         case ESTADO is
156             when REPOSO =>
157                 if (START_GC = '1') then
158                     nERROR0 <= to_sfixed(signed(REFERENCE) - signed('0'&DA), 8, 0);
159                     nESTADO <= E0;
160                 end if;
161             when E0 =>
162                 nMUL1 <= Resize(ERROR0, 13, -8);
163                 nMUL2 <= Resize(b0, 6, -16);
164                 nESTADO <= E1;
165             when E1 =>
166                 nACC <= Resize(MUL1 * MUL2, 17, -24);
167                 nMUL1 <= Resize(ERROR1, 13, -8);
168                 nMUL2 <= Resize(b1, 6, -16);
169                 nESTADO <= E2;
170             when E2 =>
171                 nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
172                 nMUL1 <= Resize(ERROR2, 13, -8);
173                 nMUL2 <= Resize(b2, 6, -16);
174                 nESTADO <= E3;
175             when E3 =>
176                 nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
177                 nMUL1 <= Resize(DUTY1, 13,-8);
178                 nMUL2 <= Resize(-a1, 6, -16);
179                 nESTADO <= E4;
180             when E4 =>
181                 nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
182                 nMUL1 <= Resize(DUTY2, 13,-8);
183                 nMUL2 <= Resize(-a2, 6, -16);
184                 nESTADO <= E5;
185             when E5 =>
186                 nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
187                 nESTADO <= E6;
188             when E6 =>
189                 --Actualizar valores
190                 nERROR1 <= ERROR0;
191                 nERROR2 <= ERROR1;
192                 nDUTY1 <= ACC(13 downto -8); -- 14_8 <= 18_24
193                 nDUTY2 <= DUTY1;
194                 --Cuantizador, coge solo la parte entera
195                 if (ACC(17 downto 0) > 450) then -- 18_0 <= 18_24
196                     nDC <= to_unsigned(450,9);
197                 elsif (ACC(17 downto 0) < 50) then -- 18_0 <= 18_24
198                     nDC <= to_unsigned(50,9);
199                 else
200                     nDC <= unsigned(ACC(8 downto 0)); -- 9_0 <= 18_24
201                 end if;
202                 nESTADO <= REPOSO;
203         end case;
```

Fig. 11. Máquina de estados en coma fija con fixed_pkg.

3.3 Implementación en coma flotante con float_pkg

En este apartado se estudia la implementación en coma flotante utilizando la librería **float_pkg**. Para su implementación ha sido de gran ayuda lo explicado en [6] y [7].

El formato en coma flotante está formado por un primer bit de signo, una cierta cantidad de bits que representan el exponente y los bits restantes para la mantisa. Habitualmente se utilizan dos formatos dependiendo de la precisión deseada y están estandarizados por el IEEE, se muestran en la Fig. 12. También es posible utilizar un tamaño personalizado.

	signo	Exponente	Mantisa
Precisión simple	1 bit	8 bits	23 bits
Precisión doble	1 bit	11 bits	52 bits

Fig. 12. Estándares de representación en coma fija según IEEE 754.

El diseño se ha basado en la utilización del formato estándar de 32 bits. Todas las señales tienen el mismo formato, por lo que su sintaxis y comprensión resultan muy sencillas.

En la Fig. 13 se muestra la declaración de las señales en formato **float** de 32 bits y las señales de entrada y salida del bloque de control, que conservan el formato **unsigned**. La inicialización de una señal ó constante de tipo **float32** necesita la función **to_float**.

```
63      -- SEÑALES DE E/S DEL CONTROLADOR
64      signal REFERENCE: unsigned(8 downto 0):= to_unsigned(194,9); --Consigna
65      signal nDC, DC : unsigned(8 downto 0); -- Duty cuantizado
66      signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final
67
68      -- SEÑALES PARA EL CONTROLADOR EN COMA FLOTANTE USANDO FLOAT_PKG
69      signal nACC, ACC: float32;
70      signal nERROR0, nERROR1, nERROR2, ERROR0, ERROR1, ERROR2 : float32;
71      signal nDUTY1, nDUTY2, DUTY1, DUTY2 : float32;
72      signal nMUL1, MUL1 : float32;
73      signal nMUL2, MUL2 : float32;
74      signal nPRODUCTO, PRODUCTO : float32;
75      signal nACCnormalizado, ACCnormalizado: float32;
76
77      -- CONSTANTES EN COMA FLOTANTE USANDO FLOAT_PKG
78      constant b2 : float32 := to_float (3.480    *6.6);
79      constant b1 : float32 := to_float (-7.658   *6.6);
80      constant b0 : float32 := to_float (4.197    *6.6);
81      constant a2 : float32 := to_float (0.5182);
82      constant a1 : float32 := to_float (-1.5182);
```

Fig. 13. Declaración de señales en coma flotante.

La máquina de estados de la Fig. 14 desarrolla la sucesión de operaciones de multiplicación, suma y acumulación. Todas las señales tienen el formato **float32**, por tanto, no es necesario efectuar ninguna adaptación entre ellas y el resultado. El código queda limpio y perfectamente legible.

La conversión entre señales de tipo **float32** a **unsigned** se efectúa con la función **to_unsigned**. En cambio, para convertir una señal de tipo **unsigned** a **float32** es necesario primero convertirla a **signed** y después utilizar la función **to_float.**

```
167         case ESTADO is
168             when REPOSO =>
169                 if (START_GC = '1') then
170                     nERROR0 <= to_float(signed(REFERENCE) - (signed('0'&DA)));
171                     nESTADO <= E0;
172                 end if;
173             when E0 =>
174                 nMUL1 <= ERROR0;
175                 nMUL2 <= b0;
176                 nESTADO <= E1;
177             when E1 =>
178                 nACC <= MUL1 * MUL2;
179                 nMUL1 <= ERROR1;
180                 nMUL2 <= b1;
181                 nESTADO <= E2;
182             when E2 =>
183                 nPRODUCTO <= MUL1 * MUL2;
184                 nMUL1 <= ERROR2;
185                 nMUL2 <= b2;
186                 nESTADO <= E3;
187             when E3 =>
188                 nACC <= PRODUCTO + ACC;
189                 nPRODUCTO <= MUL1 * MUL2;
190                 nMUL1 <= DUTY1;
191                 nMUL2 <= -a1;
192                 nESTADO <= E4;
193             when E4 =>
194                 nACC <= PRODUCTO + ACC;
195                 nPRODUCTO <= MUL1 * MUL2;
196                 nMUL1 <= DUTY2;
197                 nMUL2 <= -a2;
198                 nESTADO <= E5;
199             when E5 =>
200                 nACC <= PRODUCTO + ACC;
201                 nPRODUCTO <= MUL1 * MUL2;
202                 nESTADO <= E6;
203             when E6 =>
204                 nACC <= PRODUCTO + ACC;
205                 nESTADO <= E7;
206             when E7 =>
207                 --Actualizar valores
208                 nERROR1 <= ERROR0;
209                 nERROR2 <= ERROR1;
210                 nDUTY1 <= ACC;
211                 nDUTY2 <= DUTY1;
212                 nESTADO <= E8;
213             when E8 =>
214                 if (ACC > to_float(450.0)) then
215                     nDC <= to_unsigned(450,9);
216                 elsif (ACC < to_float(50.0)) then
217                     nDC <= to_unsigned(50,9);
218                 else
219                     nDC <= to_unsigned (ACC,9);
220                 end if;
221                 nESTADO <= REPOSO;
222         end case;
```

Fig. 14. Máquina de estados en coma flotante.

Esta versión en coma flotante ha sido problemática de implementar. Pese a funcionar perfectamente en simulación, la utilización de la librería de coma flotante de Vivado da lugar a una síntesis errónea. Varias alertas durante la síntesis sobre construcciones no sintetizables en la propia librería advierten de ello y se comprueba en la modulación que produce la FPGA, que es completamente aleatoria. Para corroborar que efectivamente es problema de la síntesis, se compara el perfil de modulación y la evolución de la variable a controlar del código VHDL inicial, Fig. 15, y del archivo generado tras la síntesis, Fig. 16.

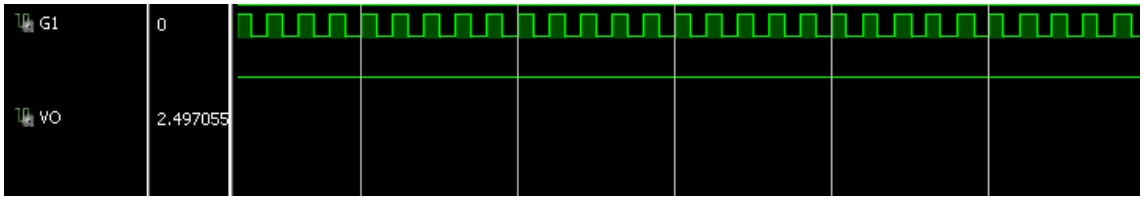


Fig. 15. Perfil de modulación y tensión de salida en simulación pre-síntesis.

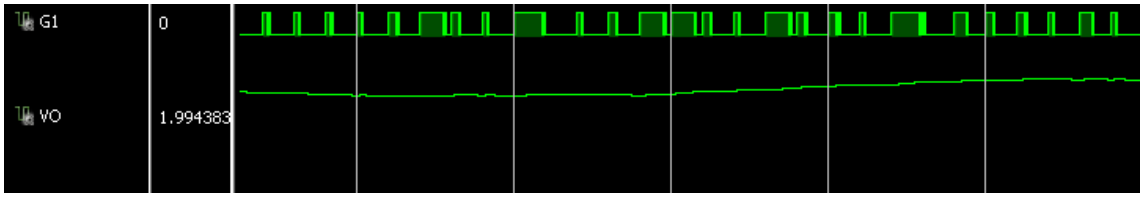


Fig. 16. Perfil de modulación y tensión de salida en simulación post-síntesis.

Para solucionar el problema, se ha utilizado una herramienta ajena a Vivado para sintetizar el diseño, ha sido “Mentor Graphics Precision RTL Synthesis”.

Con Precision Synthesis se obtiene un nuevo archivo post-síntesis completamente distinto al que proporciona Vivado. Para comprobar que la síntesis es correcta se repite la operación de simulación. Esta vez, la simulación post-síntesis coincide con la simulación inicial de la Fig. 15.

Al sintetizar el diseño con esta herramienta aparece un nuevo problema, el timing report advierte de los importantes retrasos que hay en las operaciones en coma flotante, hasta 32 ns de retraso entre lógica combinacional y propagación de señal debido a las operaciones en coma flotante.

La solución se ha realizado separando las operaciones de multiplicación y suma en bloques distintos y dividiendo la frecuencia de los biestables que intervienen en las operaciones. De esta manera, dichos biestables funcionarán a 25 MHz (40 ns) que darán el tiempo suficiente para obtener los resultados esperados en las operaciones en coma fija.

Finalmente, el controlador en coma flotante ya es funcional en la FPGA. El código completo puede consultarse en el punto correspondiente de los anexos.

3.4 Implementación en coma flotante con float_pkg, sencilla

La preparación del diseño anterior puede ser costosa pese a utilizar la librería float. Por ello, se presenta a continuación una versión más sencilla de implementar en la que todos los cálculos del controlador se plantean en una única expresión, Ecuación 7, sin utilizar la estructura MAC.

$$duty0 = -a1 * duty1 - a2 * duty2 + b0 * error0 + b1 * error1 + b2 * error2;$$

Ecuación 7. Ecuación del controlador.

Los problemas que plantea esta expresión son un enorme retraso combinacional y la utilización de más recursos de la FPGA. Para solucionar el primer problema se plantea mantener la expresión anterior un tiempo suficiente para que toda la lógica combinacional actúe y pueda dar un resultado correcto. La máquina de estados resultante se muestra en la Fig. 17. El estado E1 mantiene la operación hasta que el temporizador alcance el valor N3. El tiempo que dure este estado deberá ser superior al retraso combinacional que genere la operación que, según Precision, es de unos 120ns.

```
154         case ESTADO is
155             when REPOSO =>
156                 if (START_GC = '1') then
157                     n_error <= to_float(signed(REFERENCE) - (signed('0'&DA)));
158                     nESTADO <= E0;
159                 end if;
160             when E0 =>
161                 --Actualizar valores
162                 n_error0 <= error;
163                 n_error1 <= error0;
164                 n_error2 <= error1;
165                 n_duty1 <= duty0;
166                 n_duty2 <= duty1;
167                 nESTADO <= E1;
168             when E1 =>
169             n_duty0 <= error0*b0 + error1*b1 + error2*b2 - duty1*a1 - duty2*a2;
170                 if (TIMER = N3) then
171                     nESTADO <= E2;
172                 end if;
173             when E2 =>
174                 if (duty0 > to_float(450.0)) then
175                     nDC <= to_unsigned(450,9);
176                 elsif (duty0 < to_float(50.0)) then
177                     nDC <= to_unsigned(50,9);
178                 else
179                     nDC <= to_unsigned (duty0,9);
180                 end if;
181                 nESTADO <= REPOSO;
182         end case;
```

Fig. 17. Máquina de estados en coma flotante.

La conversión de flotante a tipo entero sin signo se ha hecho esta vez a través de la operación **to_unsigned**. Esta opción es mucho más fácil de implementar y desde luego más legible que la planteada en la Fig. 14 del punto anterior.

3.5 Implementación con bloques IP float

Vivado Design Suite está muy enfocado al diseño apoyado en bloques IP (Intellectual Property). Estos bloques son componentes parametrizables que pueden ser instanciados desde el diseño VHDL.

En esta nueva versión del control se utiliza un bloque IP capaz de hacer operaciones en coma flotante. Su parametrización permite configurar el bloque para que efectúe un producto entre sus dos entradas A y B y al resultado le sume la entrada C. Este bloque es muy similar a la estructura MAC planteada en la Tabla 1, se muestra en la Fig. 18.

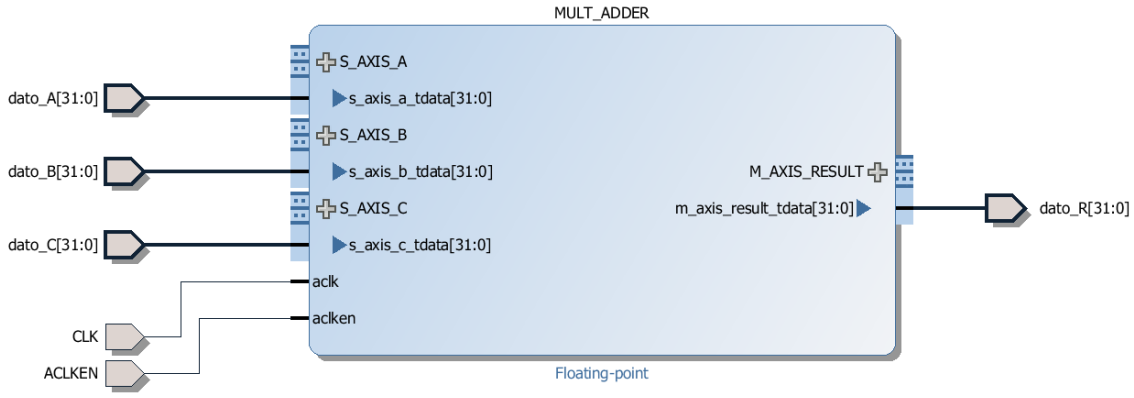


Fig. 18. Bloque IP multiplicador-sumador en coma flotante.

El bloque opera con señales de tipo **float** en formato de 32 bits. Para generar las constantes sin utilizar la librería float (que daba problemas en la versión anterior) es posible utilizar varios bloques IP de coma flotante, configurados como conversores de coma fija a coma flotante. Lo mismo se ha hecho para convertir el formato del error y el resultado acumulado. Para este último se ha configurado el bloque como conversor de coma flotante a entero. La nueva declaración de los coeficientes del controlador se muestra en la Fig. 19.

```
100     -- CONSTANTES EN COMA FIJA USANDO FIXED_PKG
101     signal b2: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (3.480*6.6,   6,-11));
102     signal b1: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (-7.658*6.6, 6,-11));
103     signal b0: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (4.197*6.6,   6,-11));
104     signal a2: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (-0.5182,     1,-16));
105     signal a1: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (1.5182,      1,-16));
```

Fig. 19. Declaración de constantes para los bloques IP float.

Para que los ficheros VHDL puedan interactuar con el bloque o, en otras palabras, que el bloque pueda ser instanciado desde el diseño principal, es necesario crear puertos de E/S. Estos puertos tratan al diseño IP completo como un único componente. Los puertos de un diseño IP son de tipo **std_logic_vector**. Esto obliga a que las señales del diseño VHDL sean definidas en este formato, aunque luego sean tratadas como coma flotante, coma fija, o entero. El diseño IP completo se muestra en la Fig. 20.

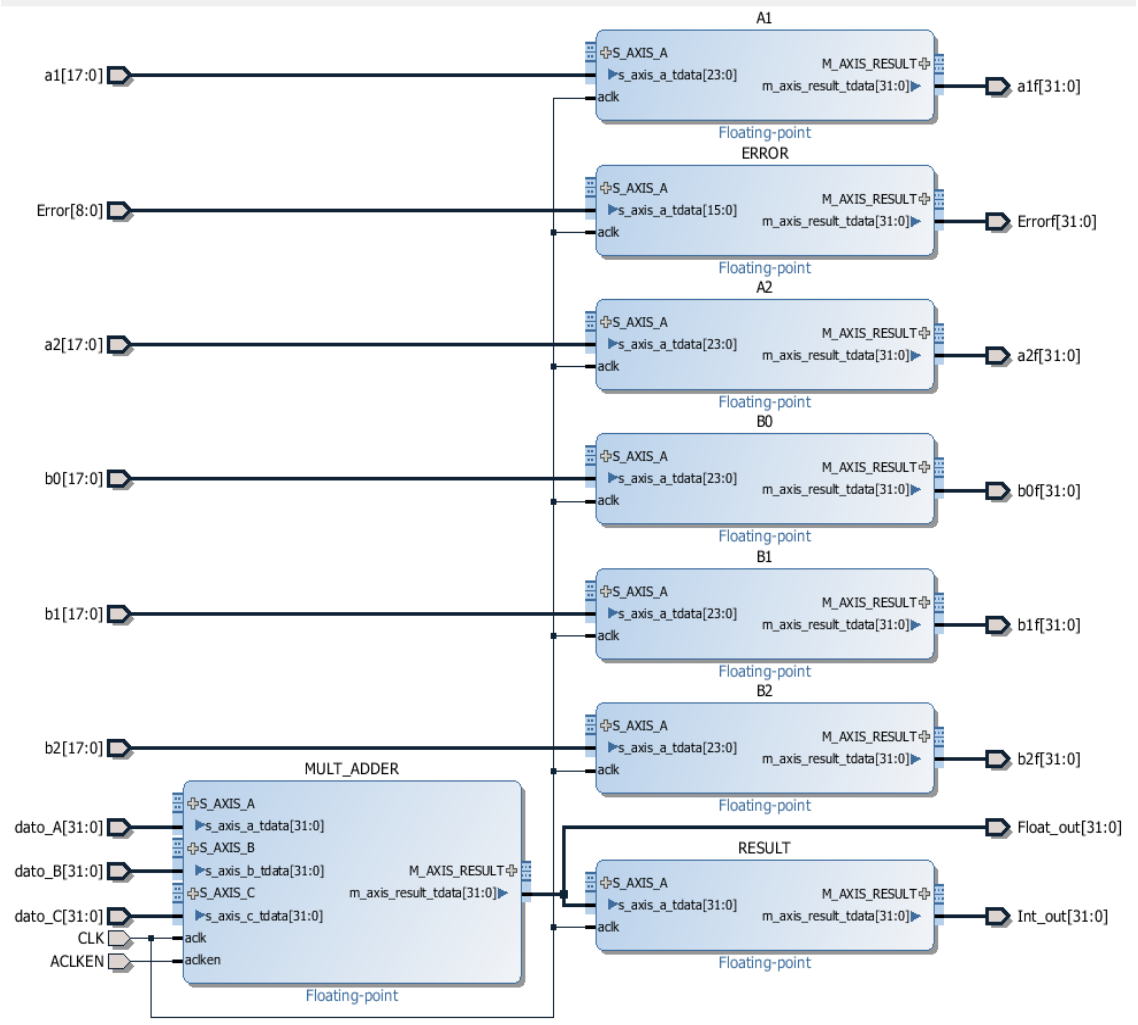


Fig. 20. Diseño de bloques IP de coma flotante.

El bloque IP "MULT_ADDER" es el encargado de realizar las operaciones de multiplicación, suma y acumulación. Por lo tanto, la máquina de estados hará la función de proporcionar ordenadamente los operadores a este bloque y de recoger su resultado una vez haya acabado el proceso de control. El resto de bloques IP tienen la función de convertir formatos.

En la Fig. 21 se muestra la máquina de estados, que va proporcionando valores al diseño IP de forma ordenada para finalmente recoger el resultado.

```
211         case ESTADO is
212             when REPOSO =>
213                 if (START_GC = '1') then
214                     nESTADO <= E0;
215                 end if;
216             when E0 => --ESTADO NECESARIO PARA RESETEAR EL BLOQUE IP
217                 nError <= std_logic_vector(to_sfixed((signed(REFERENCE) - signed('0'&DA)),8,0));
218                 ndato_A <= (others => '0');
219                 ndato_B <= (others => '0');
220                 nESTADO <= E1;
221             when E1 =>
222                 nERROR0 <= Errorf;
223                 ndato_A <= (others => '0');
224                 ndato_B <= (others => '0');
225                 nESTADO <= E2;
226             when E2 =>
227                 ndato_A <= ERROR0;
228                 ndato_B <= b0f;
229                 nESTADO <= E3;
230             when E3 =>
231                 ndato_A <= ERROR1;
232                 ndato_B <= b1f;
233                 nESTADO <= E4;
234             when E4 =>
235                 ndato_A <= ERROR2;
236                 ndato_B <= b2f;
237                 nESTADO <= E5;
238             when E5 =>
239                 ndato_A <= DUTY1;
240                 ndato_B <= a1f;
241                 nESTADO <= E6;
242             when E6 =>
243                 ndato_A <= DUTY2;
244                 ndato_B <= a2f;
245                 nESTADO <= E7;
246             when E7 => --ESTADO NECESARIO PARA QUE EL BLOQUE IP ACTUALICE SU SALIDA
247                 nESTADO <= E8;
248             when E8 =>
249                 --Actualizar valores
250                 nERROR1 <= ERROR0;
251                 nERROR2 <= ERROR1;
252                 nDUTY1 <= Float_out;
253                 nDUTY2 <= DUTY1;
254                 nESTADO <= E9;
255             when E9 =>
256                 nESTADO <= E10;
257             when E10 =>
258                 --Cuantizador, coge solo la parte entera
259                 if (signed(Int_out) > 450) then
260                     nDC <= to_unsigned(450,9);
261                 elsif (signed(Int_out) < 50) then
262                     nDC <= to_unsigned(50,9);
263                 else
264                     nDC <= unsigned(Int_out(8 downto 0));
265                 end if;
266                 nESTADO <= REPOSO;
267         end case;
```

Fig. 21. Máquina de estados con el operador IP de coma flotante.

3.6 Implementación con MicroBlaze

En este último planteamiento se estudia la utilización en el diseño de un microcontrolador soft-core. Se trata de implementar un sistema microcontrolador junto a periféricos hardware en el mismo chip. El soft-core utilizado es el mismo que el descrito en [8], MicroBlaze.

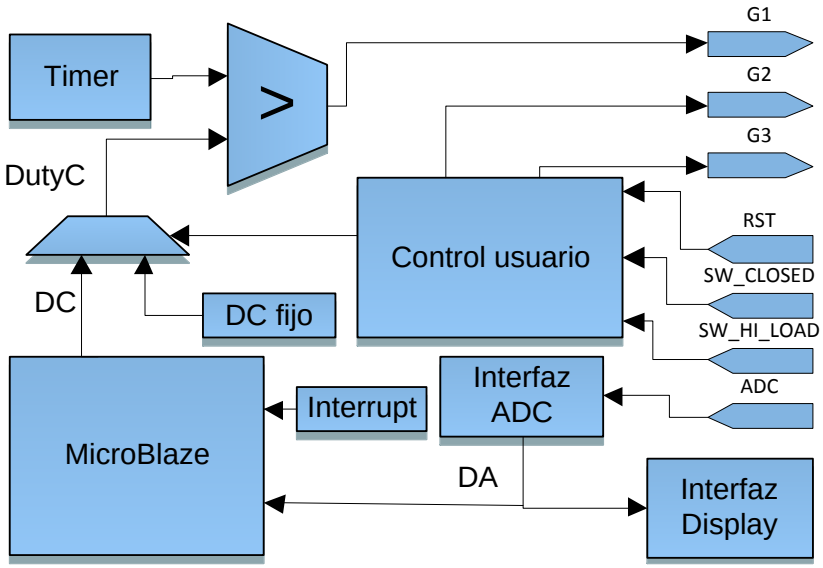


Fig. 22. Diagrama de bloques con MicroBlaze.

La nueva organización del circuito a nivel de bloques se muestra en la Fig. 22. Se mantiene la planificación temporal planteada en las otras versiones. El nuevo diseño elimina el bucle de control, que es absorbido por el microcontrolador "MicroBlaze". El contador genera una interrupción en MicroBlaze, éste lee el dato actual de tensión de salida V_o proporcionado por el conversor ADC y calcula el nuevo duty "DC" a aplicar sobre el Buck.

En Vivado existen dos opciones a la hora de incorporar el soft-core al diseño, MicroBlaze MCS y MicroBlaze. En la primera se utiliza un único bloque, en el que está incluido el procesador MicroBlaze junto a sus periféricos formando un microcontrolador, de ahí las siglas MCS (MicroController System). El problema es que se trata de un microcontrolador sencillo donde se pueden configurar pocas opciones. En concreto, el problema sobre la aplicación planteada está en no poder habilitar la FPU (Floating Point Unit), que es fundamental para realizar operaciones de coma flotante en tiempos aceptables para el control del Buck.

La segunda opción, la elegida, consiste en utilizar como punto de partida el bloque MicroBlaze, que requerirá de bloques IP adicionales para hacer funcional el sistema. Por parte del usuario, solo es necesario añadir bloques GPIO para poder conectar puertos de entrada y salida al esquema y ejecutar la opción de conexión automática. Vivado añade bloques de memoria, bloques de control, gestión de reset del sistema y un bloque de depurado. El diseño resultante se muestra en la Fig. 23.

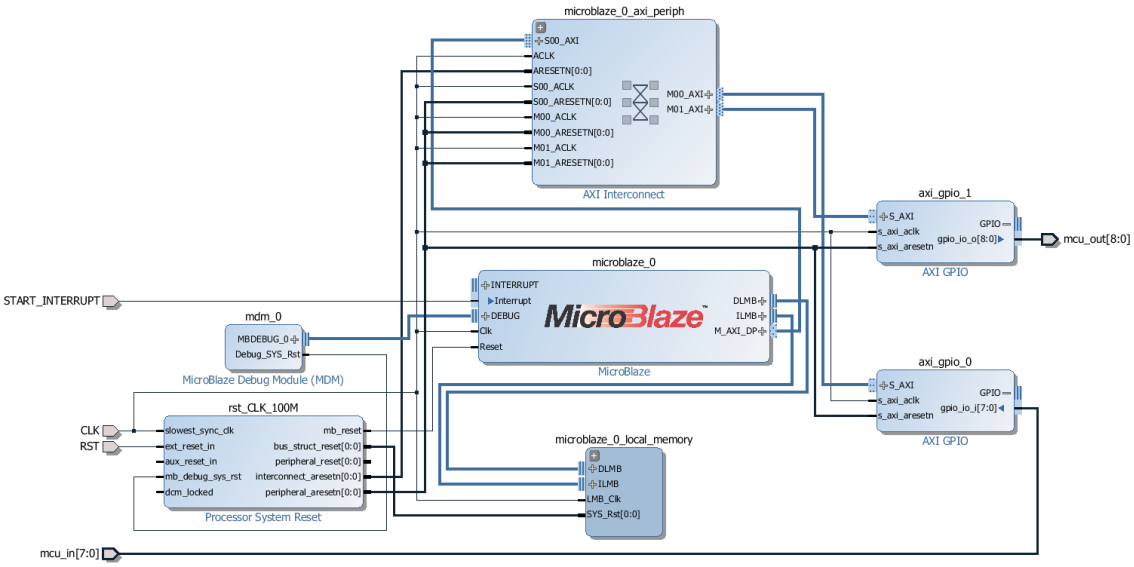


Fig. 23. Diseño IP de MicroBlaze.

Una vez obtenido el diseño hardware (soft-core + periféricos VHDL), es necesario elaborar el programa que ejecutará el microcontrolador. La implementación del código se realiza en una herramienta externa a Vivado llamada SDK, Software Development Kit. En ella es posible escribir el código en C/C++. Antes de empezar a trabajar con el SDK es necesario exportar el diseño hardware desde Vivado, lo que permitirá al SDK obtener la información necesaria para generar las librerías y soporte necesarios para la aplicación a desarrollar.

El planteamiento del código C es el siguiente. En primer lugar, se declaran todas los coeficientes y variables, véase Fig. 24. En el bucle principal se configura la interrupción que será llamada por el contador descrito en hardware y entrará en un bucle while infinito a espera de recibir la interrupción. Ajena al bucle principal se encuentra la función ISR en la que se desarrolla el código C equivalente al descrito por una máquina de estados en las versiones anteriores. Todas las operaciones y conversiones de formato se realizan aquí, de este modo se evita utilizar la librería de flotantes que daba problemas en Vivado. La rutina de interrupción puede verse en la Fig. 25.

```
15  XGpio GPInput, GPOutput; /* The Instance of the GPIO Driver */
16  uint32_t ADC, DC;
17
18  float ADC_f;
19  float error0 = 0.0;
20  float error1 = 0.0;
21  float error2 = 0.0;
22  float duty0  = 0.0;
23  float duty1  = 0.0;
24  float duty2  = 0.0;
25
26  const float Ref = 194.0;
27  const float b2 = 3.480    *6.6; //22.968
28  const float b1 = -7.658   *6.6; //-50.5428
29  const float b0 = 4.197    *6.6; //27.7002
30  const float a2 = 0.5182;
31  const float a1 = -1.5182;
32  const float duty_max = 450.0;
33  const float duty_min = 50.0;
```

Fig. 24. Declaración de variables en C.

```
46  void mi_ISR (void)
47  {
48      //Leer ADC
49      ADC = XGpio_DiscreteRead(&GPInput, 1);
50      ADC_f = (float)ADC;
51
52      //Calcular error
53      error0 = (Ref - ADC_f);
54
55      //Ecuacion del controlador
56      duty0 = -a1*duty1 - a2*duty2 + b0*error0 + b1*error1 + b2*error2;
57
58      //Actualizacion de variables
59      duty2 = duty1;
60      duty1 = duty0;
61      error2 = error1;
62      error1 = error0;
63
64      //Cuantizador para el ciclo de servicio
65      if (duty0 > duty_max)
66          duty0 = duty_max;
67      else if (duty0 < duty_min)
68          duty0 = duty_min;
69
70      //Salida del controlador
71      DC = (int) duty0;
72      XGpio_DiscreteWrite(&GPOutput, 1, DC);
73  }
```

Fig. 25. Rutina de control en C.

Es muy destacable que el MicroBlaze recibe la lectura del ADC y envía la respuesta DC en formato **unsigned** desde el hardware en VHDL. Es decir, el procesador tendrá que convertir el dato de entrada a formato **float** para calcular el resultado y convertirlo de nuevo a formato **unsigned** antes de escribirlo en la salida. Este conjunto de operaciones da lugar a un tiempo de cálculo del controlador mucho mayor que en las versiones anteriores. Se explica con detalle en el punto 4 con su simulación.

4 Resultados de simulación y validación experimental

Se explican a continuación las diferentes simulaciones realizadas en los diferentes diseños. Se resumen en un único apartado por ser resultados prácticamente idénticos en todas ellas, a excepción de la versión con MicroBlaze, como se detalla más adelante.

En la Fig. 26 se muestra una simulación genérica de las tareas que realiza la FPGA. En ella se ve como el "TIMER" se encarga de su coordinación. En primer lugar, se lanza la conversión ADC con la señal "START_ADC", pasado un tiempo se obtiene el dato leído "DA". De aquí en adelante pueden comenzar los cálculos del controlador con la señal "START_GC". La máquina de estados entra en funcionamiento y al final arroja un resultado en "DC". Este Duty Cycle se hace efectivo en el ciclo de "TIMER" siguiente con la señal "DutyC".

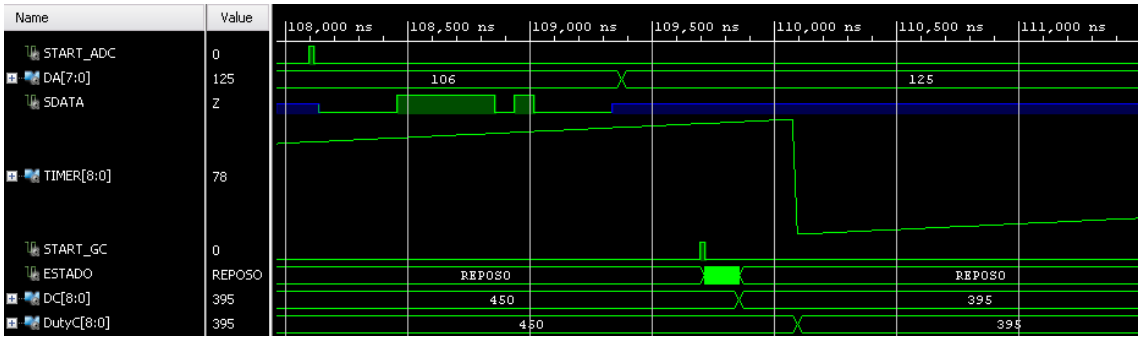


Fig. 26. Simulación genérica de procesos dentro de la FPGA.

Para resaltar el tiempo de cálculo de la máquina de estados se efectúa la simulación mostrada en la Fig. 27. Los marcadores azules y la regla temporal amarilla resaltan que el nuevo resultado se obtiene en 0,52 µs. Este resultado es similar en todas las versiones (varía en función del número de estados para llegar al resultado), salvo la implementada con MicroBlaze.

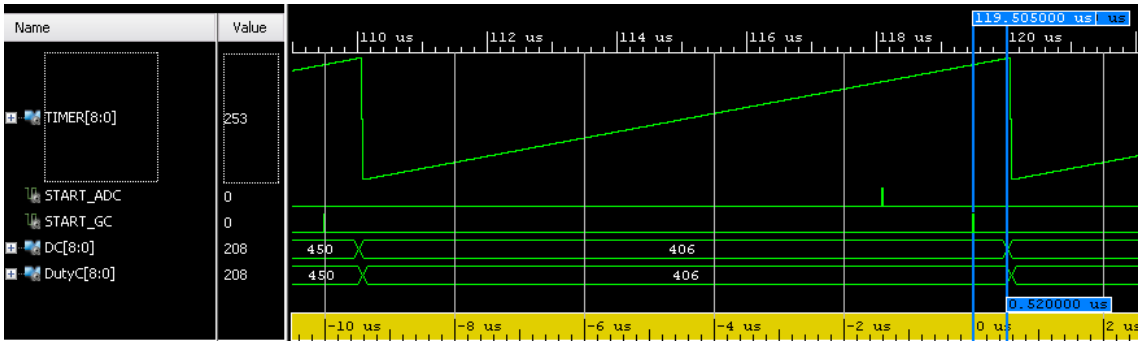


Fig. 27. Simulación de tiempo de cálculo por hardware con la librería float.

Puede verse en la Fig. 28 que el tiempo transcurrido desde que el MicroBlaze recibe el dato de entrada (mcu_in) hasta que da un resultado (mcu_out) es de hasta 7,33 µs. Además, hay que tener en cuenta que el microcontrolador está funcionando a 100 MHz y no a 50 como el resto del hardware.

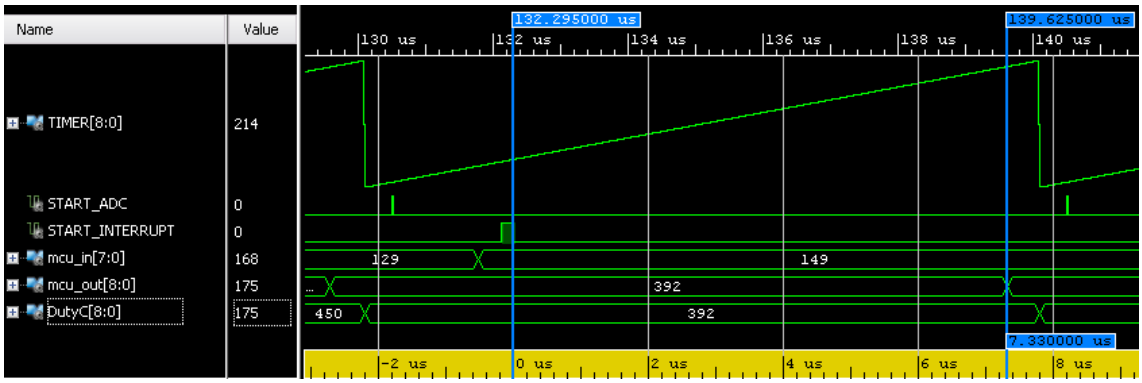


Fig. 28. Simulación de tiempo de cálculo con MicroBlaze.

Este tiempo de cálculo tan grande tiene implicaciones, como peor respuesta transitoria del controlador y que el error no se estabilice en cero. Además, tomar la medida con el ADC cerca de las conmutaciones puede producir ruido en la medida. Algunos de estos problemas se manifiestan comparando las simulaciones a gran escala de las diferentes soluciones.

En las Fig. 29 y Fig. 30 se muestran las simulaciones para la versión en coma fija y MicroBlaze respectivamente. En ellas se observan la variable a controlar V_o (tensión de salida del convertidor), el duty cycle "DC" que genera el controlador ("mcu_out" en caso de MicroBlaze), la señal de Error (no accesible con MicroBlaze) y la carga R_o. Puede verse que, en ambos, la respuesta del sistema produce una sobreoscilación en el arranque, pero los picos iniciales son distintos. A los 4 ms se produce un cambio de carga de 5 a 2,5 Ω y se produce una pequeña perturbación en V_o. Se observa en la versión con MicroBlaze que el error y, por tanto, "DC" (mcu_out), no se estabilizan.

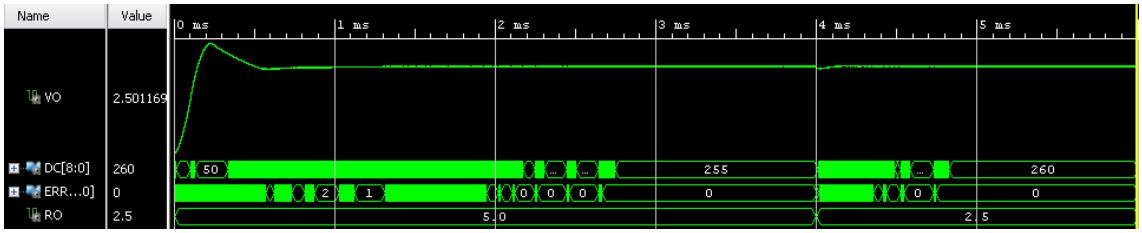


Fig. 29. Simulación a gran escala del controlador en coma fija.

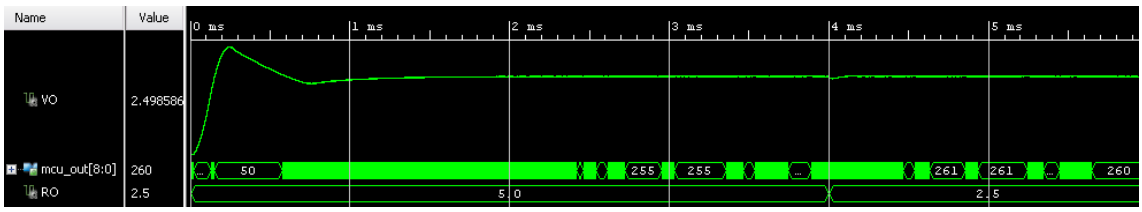


Fig. 30. Simulación a gran escala del controlador en MicroBlaze.

En la Tabla 3 se recogen los parámetros transitorios de todas las alternativas. Donde V_P es el pico inicial de arranque con R_O = 5 Ω, T_{RP} es el tiempo en alcanzar el régimen permanente (error 0) con dicha carga, T_C es el tiempo de cálculo del controlador, F_{CLK} es la frecuencia de los biestables que intervienen en el cálculo, o del procesador en caso de MicroBlaze y WNS (Worst Negative Slack) es el peor tiempo que presenta la lógica combinacional en algún punto.

Las dos primeras soluciones conducen a resultados idénticos. La opción con la librería en coma flotante tiene peores tiempos y peor respuesta transitoria al haber separado las operaciones de multiplicación y suma (mayor T_C) y operar a mitad de frecuencia debido al WNS. La alternativa con MicroBlaze tiene los peores resultados de control debido a un T_C excesivo, el régimen permanente no llega a alcanzarse, aparecen continuas oscilaciones en el error.

	Coma fija	Fixed	Float	Float senci.	Float IP	MicroBlaze
V_P (V)	3,15	3,15	3,15	3,15	3,15	3,44
T_{RP} (ms)	2,74	2,74	2,98	2,72	2,5	-
T_C (ns)	140	140	360	160	220	7330
F_{CLK} (MHz)	50	50	25	50	50	100
WNS (ns)	13,22	6,16	21,5	113,6	12,75	1,67

Tabla 3. Resumen de variables transitorias según alternativa de implementación.

Una vez obtenidos los resultados de simulación en cada una de las soluciones, se han comprobado en el sistema real, con el montaje de la Fig. 31.

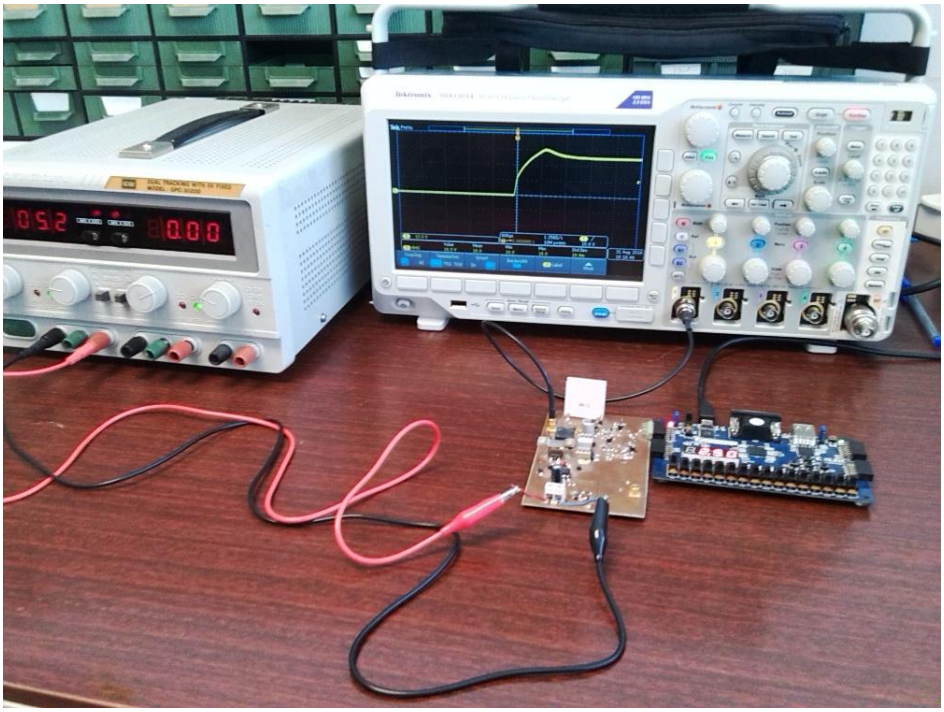


Fig. 31. Montaje para verificación en laboratorio.

En el osciloscopio se capturan los mismos instantes que los vistos en simulación con la tensión de salida del convertidor, el arranque con carga de 5 Ω, Fig. 32, y el cambio de carga a 2,5 Ω, Fig. 33.

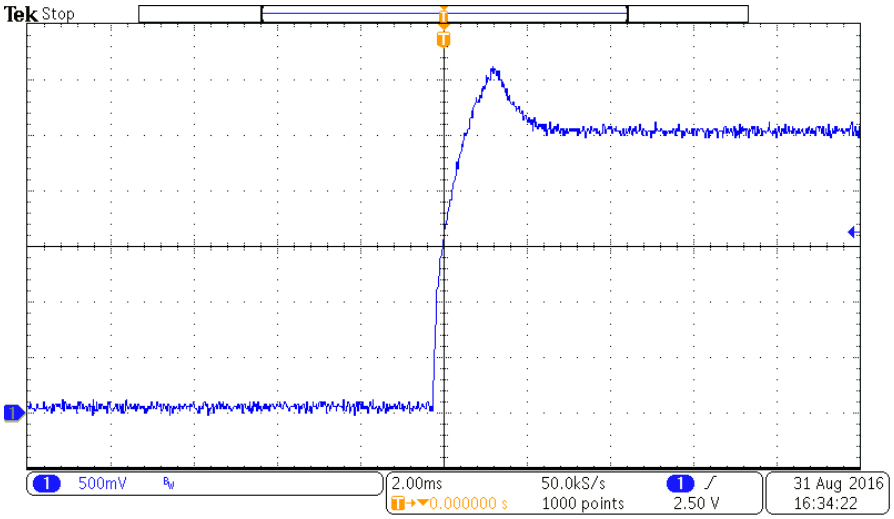


Fig. 32. Osciloscopio, arranque del controlador.

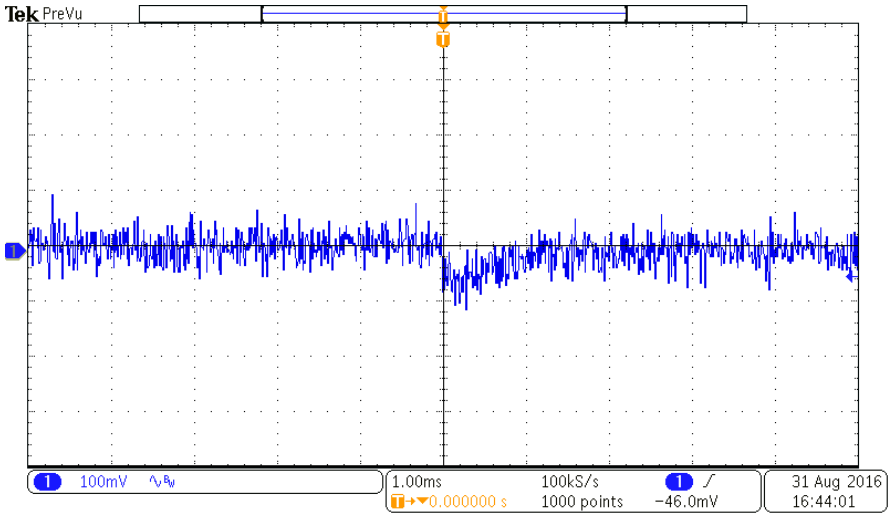


Fig. 33. Osciloscopio, cambio de carga.

Con todas estas pruebas se da por concluido el trabajo. Se ha llegado a versiones funcionales en todas las alternativas planteadas.

Por último, se presentan los resultados obtenidos a nivel de recursos en cada una de las versiones. Estos resultados se muestran en la Tabla 4, con valores concretos y en la Tabla 5, con valores porcentuales.

Recurso	Coma fija	Fixed	Float	Float senci.	Float IP	MicroBlaze
LUT	225	449	2667	6674	1711	2103
LUTRAM	-	-	-	-	15	131
FF	229	230	303	714	1169	1799
BRAM	-	-	0.5	0.5	-	16
DSP	1	1	2	7	2	5
IO	22	22	22	22	22	22
BUFG	1	1	1	1	2	3

Tabla 4. Utilización de recursos.

Recurso	Coma fija	Fixed	Float	Float senci.	Float IP	MicroBlaze
LUT	1,08	2,16	12,82	32,09	8,23	10,11
LUTRAM	0,00	0,00	0,00	0,00	0,16	1,36
FF	0,55	0,55	0,73	1,72	2,81	4,32
BRAM	0,00	0,00	1,00	1,00	0,00	32,00
DSP	1,11	1,11	2,22	7,78	2,22	5,56
IO	20,75	20,75	20,75	20,75	20,75	20,75
BUFG	3,13	3,13	3,13	3,13	6,25	9,38

Tabla 5. Utilización de recursos en valor porcentual.

Se observa que, en las versiones iniciales, la cantidad y variedad de recursos que utiliza Vivado para implementar el diseño es muy reducida. Conforme se va avanzando en el desarrollo de nuevas alternativas, la cantidad de recursos necesarios aumenta. En particular, en las alternativas más sencillas o de más alto nivel, la cantidad de recursos es mayor. Estas son la versión con la librería float_pkg (versión sencilla) y la versión con MicroBlaze.

5 Conclusiones y líneas futuras

En este último apartado se recogen las aportaciones de este Trabajo Fin de Máster y las conclusiones extraídas. Además, se proponen algunas líneas futuras como continuación al trabajo realizado.

5.1 Conclusiones

Este Trabajo Fin de Master tiene como objetivo explorar una serie de alternativas para la implementación de un controlador en un convertidor electrónico de potencia. Se trata de plantear soluciones más sencillas a la tradicional en coma fija utilizada en asignaturas de este ámbito y, además, utilizar herramientas actuales, como el entorno de desarrollo Vivado y una FPGA de la serie 7 de Xilinx.

Las conclusiones inmediatas que se extraen al comparar las alternativas estudiadas son en cuanto a:

- Tiempos de diseño. Alternativas que utilizan librerías de coma fija o coma flotante, bloques IP o soft-cores, dan lugar a diseños más sencillos y rápidos de implementar.
- Coste de recursos. La utilización de estas alternativas conlleva un consumo mayor de recursos disponibles en la FPGA.

La elección de una u otra alternativa al comenzar un diseño depende del perfil del propio ingeniero. Pueden plantearse dos: uno generalista y otro especializado.

- Con un perfil generalista, como podría ser el de un ingeniero industrial, es recomendable utilizar alternativas de más alto nivel que no supongan un tiempo de desarrollo excesivo para el diseñador. Por ejemplo, la utilización de librerías de coma flotante, o la inclusión un microcontrolador en el diseño como MicroBlaze.
- Con un perfil más especializado, como el de un ingeniero electrónico, es beneficioso profundizar en diseños más eficientes en cuanto a recursos. Las mejores alternativas pueden ser las que utilizan las librerías de coma fija o coma flotante con reutilización de recursos (estructura MAC). La opción en coma fija tradicional, pese a ser la mejor en cuanto a recursos, es la más costosa para el diseñador.

Algunos problemas que han surgido en este TFM resaltan la importancia de varios factores que pueden ser problemáticos durante el diseño y que un estudiante podría no tener en cuenta.

- Problemas en cuanto a síntesis. Se ha visto que Vivado no podía sintetizar correctamente los diseños que utilizaban la librería de coma flotante de su colección “IEEE-proposed”. Como solución se ha utilizado otro compilador para este diseño, “Precision RTL Synthesis”. Se pone de manifiesto la importancia de la simulación post-síntesis.
- Problemas en cuanto a timing. La utilización de operaciones en coma flotante da lugar a retrasos combinacionales que pueden ser superiores al periodo de reloj. Es importante tener en cuenta el periodo con el que se actualizan los biestables para que el diseño funcione.

- Problemas en cuanto a un tiempo de cálculo excesivo. La utilización de un microcontrolador en el lazo de control provoca tiempos de cálculo importantes que dan lugar a una peor respuesta transitoria del sistema.

5.2 Líneas futuras

El planteamiento de nuevas alternativas está enfocado a que los estudiantes puedan conocer técnicas actuales y sencillas, que les puedan ser de ayuda en su futuro como ingenieros. Algunas líneas de trabajo serían:

- Plantear una serie de prácticas de laboratorio en las que se incluya la utilización de las alternativas aquí planteadas. Más adelante, sería necesario evaluar si estas nuevas prácticas son beneficiosas para los estudiantes.
- Explorar otras técnicas que no se han tenido en cuenta en este TFM. Soluciones de generación automática de código, como las basadas en herramientas de Matlab (HDL Coder, XSG) o las herramientas de síntesis de alto nivel (de C a HDL).

6 Referencias

[1] J. J. Rodriguez-Andina, M. J. Moure, y M. D. Valdes, «Features, Design Tools, and Application Domains of FPGAs», *IEEE Trans. Ind. Electron.*, vol. 54, n.º 4, pp. 1810-1823, ago. 2007.

[2] J. J. Rodríguez-Andina, M. D. Valdés-Peña, y M. J. Moure, «Advanced Features and Industrial Applications of FPGAs; A Review», *IEEE Trans. Ind. Inform.*, vol. 11, n.º 4, pp. 853-864, ago. 2015.

[3] «Xilinx». [En línea]. Disponible en: http://www.xilinx.com/.

[4] J. I. Artigas, L. A. Barragán, U. Isidro, D. Navarro, y Ó. Lucía, «FPGA-based digital control implementation of a power converter for teaching purposes», en *2011 5th IEEE International Conference on e-Learning in Industrial Electronics (ICELIE)*, 2011, pp. 55-60.

[5] D. Bishop, «Fixed point package user`s guide». .

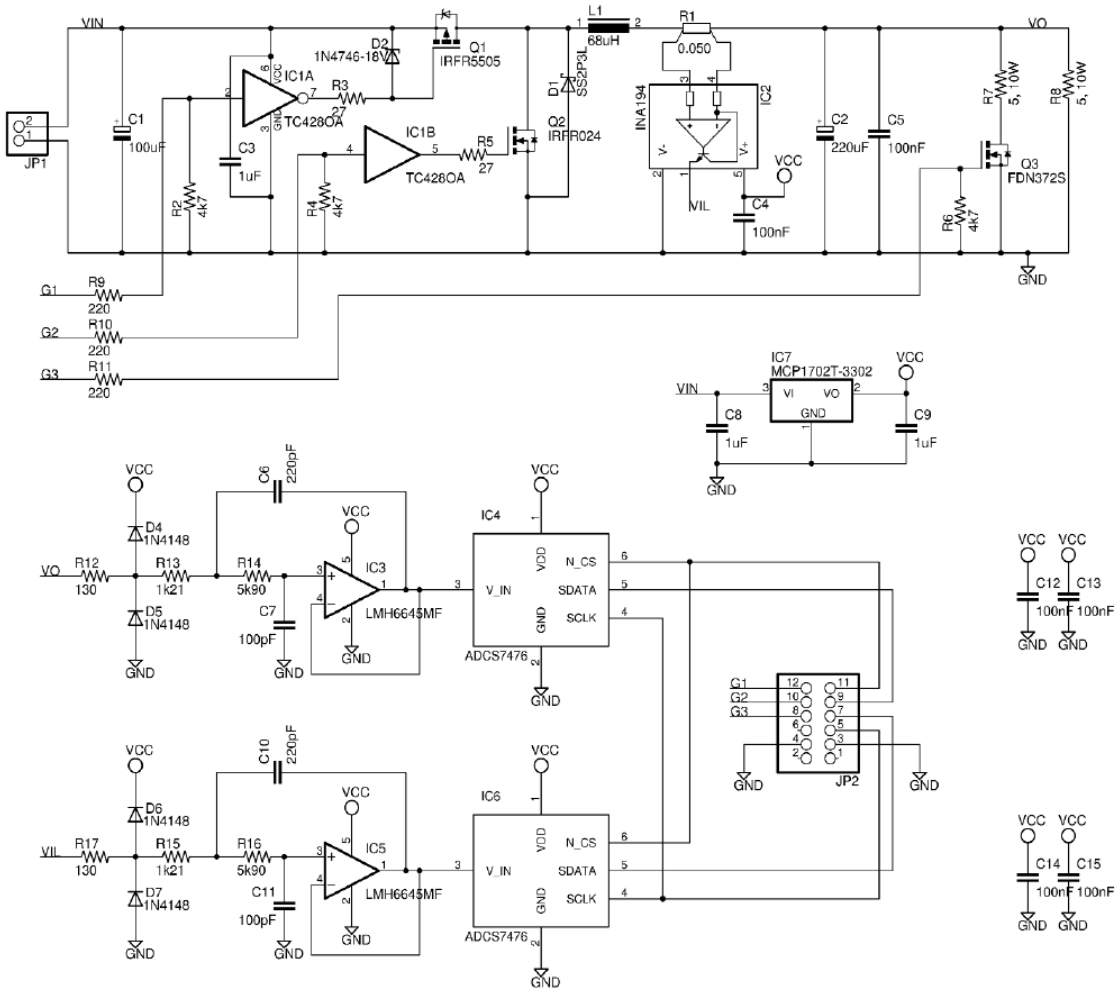
[6] D. Bishop, «Floating point package user's guide», *ResearchGate*.

[7] I. Urriza, L. A. Barragán, J. I. Artigas, D. Navarro, y O. Lucía, «FPGA implementation of a digital controller for a dc-dc converter using floating point arithmetic», en *35th Annual Conference of IEEE Industrial Electronics, 2009. IECON '09*, 2009, pp. 2913-2918.

[8] I. Urriza, L. A. Barragán, D. Navarro, J. I. Artigas, O. Lucía, y O. Jiménez, «FPGA Embedded Soft-Core Processor Implementation of a Digital Controller for a DC-DC Converter», *ResearchGate*, ene. 2009.

7 Anexos

7.1 Esquema del convertidor Buck



7.2 Características de la Basys3

- FPGA Artix de la serie 7: XC7A35T-1CPG236C
- 33.280 bloques lógicos en 5.200 slices (cada slice contiene 4 LUTs de 6 entradas y 8 flip-flops). En total 21.800 LUT y 41.600 flip-flops.
- 1.800 Kbits para RAM.
- 5 Bloques de gestión de reloj (PLL).
- 90 DSPs.
- Posibilidad de señales de reloj internas de más de 450 MHz.
- Conversor ADC integrado en la FPGA (XADC).
- Puertos USB-JTAG para programación y comunicación de la FPGA.
- Conector Micro-B USB.
- Memoria Serial Flash.
- Conversor USB-UART Bridge.
- Salida VGA de 12-bit.
- USB HID Host para ratón, teclado y pen-drives.
- 16 interruptores.

- 16 LEDs.
- 5 pulsadores.
- 4 displays de 7 segmentos.
- 4 puertos PMOD: 3 estándar de 12 pines y 1 dual para señal XADC o estándar.

7.3 Test bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tbench IS
END tbench;

ARCHITECTURE behavior OF tbench IS
-- Component Declaration for the Unit Under Test (UUT)
   COMPONENT fpga
   PORT(
      CLK : IN  std_logic;
      RST : IN  std_logic;
      SDATA : IN  std_logic;
      SW_CLOSED : IN  std_logic;
      SW_HI_LOAD : IN  std_logic;
      NCS : OUT  std_logic;
      SCLK : OUT  std_logic;
      SSEG : OUT  std_logic_vector(6 downto 0);
      DP : OUT  std_logic;
      ASEL : OUT  std_logic_vector(3 downto 0);
      G1 : OUT  std_logic;
      G2 : OUT  std_logic;
      G3 : OUT  std_logic
      );
   END COMPONENT;

   --Inputs
   signal CLK : std_logic := '0';
   signal RST : std_logic := '0';
   signal SDATA : std_logic := '0';
   signal SW_CLOSED : std_logic := '1'; -- LAZO CERRADO
   signal SW_HI_LOAD : std_logic := '0';
   signal DATA: std_logic_vector(11 downto 0);

   --Outputs
   signal NCS : std_logic;
   signal SCLK : std_logic;
   signal SSEG : std_logic_vector(6 downto 0);
   signal DP : std_logic;
   signal ASEL : std_logic_vector(3 downto 0);
   signal G1 : std_logic;
   signal G2 : std_logic;
   signal G3 : std_logic;

   -- Parámetros ecs en diferencias
   constant VG : real := 5.0;
   constant VF : real := 0.0;
   constant L  : real := 68.0e-6;
   constant C  : real := 220.0e-6;
   constant RC : real := 80.0e-3;
   constant RL : real := 98.0e-3;
   signal RO   : real := 5.0;
   constant Ts : time := 20 ns;
   constant dT : real := 2.0e-8;
   constant VFS : real := 3.3;
   constant LSB : real := VFS/2.0**8;
   constant VFSM1LSB : real:= VFS - LSB; -- LSB del conversor ADC
   -- Variables de estado
   signal IL, VC, VO, HVO : real := 0.0; -- Valor inicial

   -- Clock period definitions
   constant CLK_period : time := 10 ns;
```

```
   -- Comunicacion simulada del ADC
   procedure write_serial(signal DATA: in std_logic_vector(11 downto 0);
      signal NCS, SCLK: in std_logic;  signal SDATA: out std_logic) is
   begin
      SDATA <= 'Z';
      wait until nCS'event and nCS='0';
      for i in 2 downto 0 loop
         wait until SCLK'event and SCLK='0';
         SDATA <= '0';
      end loop;
      for i in 11 downto 0 loop
         wait until SCLK'event and SCLK='0';
         SDATA <= DATA(i);
      end loop;
      wait until SCLK'event and SCLK='0';
         SDATA <= 'Z';
   end write_serial;

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: fpga PORT MAP (
      CLK => CLK,
      RST => RST,
      SDATA => SDATA,
      SW_CLOSED => SW_CLOSED,
      SW_HI_LOAD => SW_HI_LOAD,
      NCS => NCS,
      SCLK => SCLK,
      SSEG => SSEG,
      DP => DP,
      ASEL => ASEL,
      G1 => G1,
      G2 => G2,
      G3 => G3
      );

   -- G3, PERTURBACIÓN 10ms cambio de carga
   SW_HI_LOAD <= '0', '1' after 6 ms;
   RO <= 2.5 when SW_HI_LOAD = '1' else 5.0;
   SW_CLOSED <='1', '0' after 10 ms;

   -- Clock process definitions
   CLK_process :process
   begin
      CLK <= '0';
      wait for CLK_period/2;
      CLK <= '1';
      wait for CLK_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      RST <='1';
      wait for 100 ns;
      RST <='0';
      wait for CLK_period*10;
      -- insert stimulus here
   wait;
   end process;

   ----------------------------------------------------------------
   --    MODELO DEL BUCK ECUACIONES EN DIFERENCIAS IL VO VC
   ----------------------------------------------------------------
   process
      variable IL_aux : real;
   begin
      wait for Ts;
      if (G1 = '1') then -- q1
         IL_aux := IL*(1.0-(dT*(RL*RO+RL*RC+RC*RO)/(L*(RO+RC)))) -
         VC*((dT*RO)/( (L*(RO+RC)))) + (VG*dT/L);
      else --q2
         IL_aux := IL*(1.0-(dT*(Rl*RO+Rl*RC+RC*RO)/(L*(RO+RC)))) -
```

```
            VC*((dT*RO)/( (L*(RO+RC)))) - (VF*dT/L);
         end if;
         if (IL_aux < 0.0) then --q3
            IL_aux := 0.0;
         end if;
         IL <= IL_aux;
         VC <= (((dT/C)*(Ro/(Ro+Rc)))*iL) + vC*(1.0-((dT/C)*(1.0/(RO+RC))));
         VO <= (((RC*RO)/(RO+RC))*iL) + ((Ro/(RO+RC))*VC);
         HVO <= VO;
      end process;

      ----------------------------------------------------------------
      --                ADC SIMULADO
      ----------------------------------------------------------------
      write serial(DATA,NCS,SCLK, SDATA);
      -- Conversor ADC (8 bits, 3.3 V)
      process (NCS) is
         variable RES: integer;
         variable VAUX : real;
      begin
         if (NCS' event and NCS='0') then
            if (HVO >= VFSM1LSB) then
               VAUX := VFSM1LSB;
            elsif (HVO <= 0.0) then
               VAUX := 0.0;
            else
               VAUX := HVO;
            end if;
            RES := integer(VAUX*((2.0**8) - 1.0) / VFSM1LSB);
            DATA <= conv_std_logic_vector(RES,8)&"0000";-- OJO, TIENE QUE SER DA
         end if;
      end process;

END;
```

7.4 Interfaz ADC

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

entity adc is
   Port (SDATA : in  STD_LOGIC;
      NCS : out  STD_LOGIC;
      SCLK : out STD_LOGIC;
      CLK : in STD_LOGIC;
      RST : in STD_LOGIC;
      START_ADC : in STD_LOGIC;
      DA : out  STD_LOGIC_VECTOR (7 downto 0));
end adc;

architecture Behavioral of adc is
   type STATES is (IDLE, START, CK0, CK1, CK2, CK3);
   signal STATE, nSTATE: STATES;
   signal CNT, nCNT: STD_LOGIC_VECTOR (3 downto 0);
   signal DATA, nDATA: STD_LOGIC_VECTOR (14 downto 0);
   signal nDA, pDA : std_logic_vector(7 downto 0);
   signal EN : std_logic;

begin
   -- Control FFs
   process(RST, EN, CLK)
   begin
      if (RST = '1') then
         EN <= '0';
         STATE <= IDLE;
         CNT <= (others => '0');
         DATA <= (others => '0');
         pDA <= (others => '0');
      elsif (CLK'event and CLK = '1') then
         if EN = '1' then
            STATE <= nSTATE;
            CNT <= nCNT;
            DATA <= nDATA;
            pDA <= nDA;
         end if;
         EN <= NOT EN;
      end if;
   end process;

   -- Maquina de estados
   process(STATE, START_ADC, SDATA, CNT, DATA, pDA)
   begin
      nSTATE <= STATE;
      nCNT <= CNT;
      nDATA <= DATA;
      SCLK <= '1';
      NCS <= '1';
      nDA <= pDA;

      case STATE is
         when IDLE =>
            if (START_ADC = '1') then
               nCNT <= (others => '0');
               nSTATE <= START;
            end if;
         when START =>
            NCS <= '0';
            nSTATE <= CK0;
         when CK0 =>
            SCLK <= '0';
            nSTATE <= CK1;
            NCS <= '0';
         when CK1 =>
            SCLK <= '0';
            nSTATE <= CK2;
            NCS <= '0';
            if (CNT = "1111") then
               nSTATE <= IDLE;
```

```
                nDA <= DATA (11 downto 4); -- Solo los 8 MSB
            else
                nSTATE <= CK2;
            end if;
         when CK2 =>
            SCLK <= '1';
            NCS <= '0';
            nSTATE <= CK3;
         when CK3 =>
            nDATA (14 - conv_integer(CNT)) <= SDATA;
            nCNT <= CNT + '1';
            SCLK <= '1';
            nSTATE <= CK0;
            NCS <= '0';
      end case;
   end process;
DA <= pDA;
end Behavioral;
```

7.5 Versión en coma fija

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

entity fpga is
   Port(CLK: in  STD_LOGIC;
        RST: in  STD_LOGIC;
        SDATA: in  STD_LOGIC;
        SW_CLOSED: in STD_LOGIC;
        SW_HI_LOAD: in STD_LOGIC;
        NCS: out  STD_LOGIC;
        SCLK: out  STD_LOGIC;
        SSEG: out  STD_LOGIC_VECTOR (6 downto 0);
        DP: out  STD_LOGIC;
        ASEL: out  STD_LOGIC_VECTOR (3 downto 0);
        G1: out  STD_LOGIC;
        G2: out  STD_LOGIC;
        G3: out  STD_LOGIC
        );
end fpga;

architecture Behavioral of fpga is
   component adc2bcd is
      Port (DA: in STD_LOGIC_VECTOR (7 downto 0);
            DO: out STD_LOGIC_VECTOR (11 downto 0));
   end component;

   component adc is
      Port  (  SDATA    : in  STD_LOGIC;
               NCS      : out  STD_LOGIC;
               SCLK     : out STD_LOGIC;
               CLK      : in STD_LOGIC;
               RST      : in STD_LOGIC;
               START_ADC: in STD_LOGIC;
               DA       : out STD_LOGIC_VECTOR (7 downto 0)
            );
   end component;

   signal DA: STD_LOGIC_VECTOR (7 downto 0);
   signal DO: STD_LOGIC_VECTOR (11 downto 0);

   -- Counter to refresh data and multiplex digits
   signal CN_DISP, nCN_DISP: unsigned (19 downto 0);
   signal START_ADC, BCD_EN: STD_LOGIC;

   -- Multiplexed Display
   signal BCD: unsigned (3 downto 0);
   signal BCD_CTL, nBCD_CTL: STD_LOGIC_VECTOR(2 downto 0);

   -- MAQUINA DE ESTADOS
   type estados is (REPOSO,E0,E1,E2,E3,E4,E5,E6);
   signal ESTADO, nESTADO: estados;

   -- SEÑALES PARA LA PLANIFICACION DEL CONTROL
   signal START_GC : std_logic; -- Señales de inicio
   signal TIMER, nTIMER : unsigned(8 DOWNTO 0); -- PWM
   constant N2 : unsigned(8 downto 0) := to_unsigned(480,9); -- Para iniciar GC
   constant N1 : unsigned(8 downto 0) := to_unsigned(400,9); -- Para iniciar ADConv

   -- SEÑALES DE E/S DEL CONTROLADOR
   signal REFERENCE: unsigned (8 downto 0):= to_unsigned(194,9); --Consigna
   signal nDC, DC : unsigned (8 downto 0);
   signal nDutyC, DutyC: unsigned (8 downto 0);
   signal nREFERENCE: unsigned (12 downto 0);

   -- SEÑALES PARA EL CONTROLADOR EN COMA FIJA
   signal nACC, ACC: signed(41 downto 0) := (others => '0'); --<42,24>
   signal nERROR0, nERROR1, nERROR2, ERROR0, ERROR1, ERROR2: signed(8 downto 0); --<9,0>
   signal nMUL1, MUL1,nDUTY1, nDUTY2, DUTY1, DUTY2 : signed(21 downto 0); -- <22,8>

   -- CONSTANTES DEL CONTROLADOR EN COMA FIJA
   signal nMUL2, MUL2: signed(17 downto 0);
   constant b2 : signed(17 downto 0):= to_signed(47038,   18); --  3.480 * 6.6 <18,11>
```

```
   constant b1 : signed(17 downto 0):= to_signed(-103512, 18); -- -7.658 * 6.6 <18,11>
   constant b0 : signed(17 downto 0):= to_signed(56730,   18); --  4.197 * 6.6 <18,11>
   constant a2 : signed(17 downto 0):= to_signed(33961,   18); --  0.5182      <18,16>
   constant a1 : signed(17 downto 0):= to_signed(-99497,  18); -- -1.5185      <18,16>

   -- SEÑAL DE ACTIVACION DE LOS BIESTABLES
   signal EN : std_logic;

begin
   U1: adc2bcd PORT MAP (DA => DA, DO => DO);
   U2: adc PORT MAP (SDATA => SDATA, NCS => NCS, SCLK => SCLK,
                     CLK => CLK, RST => RST, START_ADC => START_ADC, DA => DA);
   ----------------------------
   -- PROCESO DE BIESTABLES --
   ----------------------------
   process (CLK, EN, RST)
   begin
      if (RST = '1') then
          EN <= '0';
         CN_DISP <= (others => '0');
         BCD_CTL <= "110";
         -- CONTROLADOR
         TIMER <= (others => '0');
         ESTADO <= REPOSO;
         ACC <= (others => '0');
         ERROR2<= (others => '0');
         ERROR1<= (others => '0');
         ERROR0<= (others => '0');
         DUTY2<= (others => '0');
         DUTY1<= (others => '0');
         DC <= (others => '0');
         DutyC <= (others => '0');
         MUL1 <= (others => '0');
         MUL2 <= (others => '0');
      elsif (CLK'event and CLK = '1') then
         if EN = '1' then
            CN_DISP <= nCN_DISP;
            BCD_CTL <= nBCD_CTL;
            -- CONTROLADOR
            TIMER <= nTIMER;
            ESTADO <= nESTADO;
            ACC  <= nACC;
            ERROR2<= nERROR2;
            ERROR1 <= nERROR1;
            ERROR0 <= nERROR0;
            DUTY2 <= nDUTY2;
            DUTY1 <= nDUTY1;
            DC <= nDC;
            DutyC <= nDutyC;
            MUL1 <= nMUL1;
            MUL2 <= nMUL2;
         end if;
         EN <= NOT EN;
      end if;
   end process;

   ------------------------------------------------------
   --                   CONTROLADOR                    --
   ------------------------------------------------------

   --SEÑAL INICIO ADC inicio
   START_ADC <= '1' when (TIMER = N1) else '0';

   -- SEÑAL DE ACTIVACIÓN DEL CONTROL
   START_GC <= '1' when (TIMER = N2) else '0';

   -- CONTROLADOR
   process (START_GC, ESTADO, REFERENCE, DA, DC, ERROR0, ERROR1, ERROR2, ACC, DUTY1,
DUTY2, MUL1, MUL2)
   begin
         nESTADO <= ESTADO;
         nACC <= ACC;
         nERROR0 <= ERROR0;
         nERROR1 <= ERROR1;
         nERROR2 <= ERROR2;
```

```
        nDUTY1 <= DUTY1;
        nDUTY2 <= DUTY2;
        nDC <= DC;
        nMUL1 <= MUL1;
        nMUL2 <= MUL2;
    case ESTADO is
        when REPOSO =>
            if (START_GC = '1') then
                nERROR0 <= signed(REFERENCE)- signed('0'& DA);
                nESTADO <= E0;
            end if;
        when E0 =>
            nMUL1 <= ERROR0 & "0000000000000";
            nMUL2 <= b0;
            nESTADO <= E1;
        when E1 =>
            nACC <= Resize(MUL1 * MUL2, 42);
            nMUL1 <= ERROR1 & "0000000000000";
            nMUL2 <= b1;
            nESTADO <= E2;
        when E2 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 42);
            nMUL1 <= ERROR2 & "0000000000000";
            nMUL2 <= b2;
            nESTADO <= E3;
        when E3 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 42);
            nMUL1 <= DUTY1;
            nMUL2 <= -a1;
            nESTADO <= E4;
        when E4 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 42);
            nMUL1 <= DUTY2;
            nMUL2 <= -a2;
            nESTADO <= E5;
        when E5 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 42);
            nESTADO <= E6;
        when E6 =>
            --Actualizar valores
            nERROR1 <= ERROR0;
            nERROR2 <= ERROR1;
            nDUTY1 <= ACC(37 downto 16);
            nDUTY2 <= DUTY1;
            --Cuantizador, coge solo la parte entera
            if (ACC(41 downto 24) > 450) then
                nDC <= to_unsigned(450, 9);
            elsif (ACC(41 downto 24) < 50) then
                nDC <= to_unsigned(50, 9);
            else
                nDC <= unsigned(ACC(32 downto 24));
            end if;
                nESTADO <= REPOSO;
    end case;

end process;

-- TIMER: Generación de DPWM
nTIMER <= (TIMER+1) when (TIMER < 499) else
          (others => '0');

-- Selección del Duty en función del SW
nDutyC <= DC when (TIMER = 499 and SW_CLOSED = '1') else --Lazo cerrado
          to_unsigned(200,9) when (SW_CLOSED = '0') else --Lazo abierto
          DutyC; --Resto del tiempo, no hacer nada.

-- SEÑALES DE CONTROL DEL CONVERTIDOR:

G1 <= '1' when (DutyC > TIMER) else '0'; -- G1
G2 <= '0';  -- G2 siempre abierto
G3 <= '1' when (SW_HI_LOAD = '1') else '0';
```

7.6 Versión en coma fija con fixed_pkg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library ieee_proposed;
use ieee_proposed.fixed_pkg.all;

entity fpga is
   Port (CLK : in STD_LOGIC;
      RST : in STD_LOGIC;
      SDATA : in  STD_LOGIC;
      SW_CLOSED : in STD_LOGIC;
      SW_HI_LOAD : in STD_LOGIC;
      NCS : out  STD_LOGIC;
      SCLK : out  STD_LOGIC;
      SSEG : out  STD_LOGIC_VECTOR (6 downto 0);
      DP : out  STD_LOGIC;
      ASEL : out  STD_LOGIC_VECTOR (3 downto 0);
      G1 : out  STD_LOGIC;
      G2 : out  STD_LOGIC;
      G3 : out  STD_LOGIC
      );
end fpga;

architecture Behavioral of fpga is
   component adc2bcd is
      Port (DA: in STD_LOGIC_VECTOR (7 downto 0);
            DO: out STD_LOGIC_VECTOR (11 downto 0));
   end component;

   component adc is
      Port  (  SDATA : in  STD_LOGIC;
               NCS : out  STD_LOGIC;
               SCLK : out STD_LOGIC;
               CLK : in STD_LOGIC;
               RST : in STD_LOGIC;
               START_ADC : in STD_LOGIC;
               DA : out STD_LOGIC_VECTOR (7 downto 0)
            );
   end component;

   signal DA: STD_LOGIC_VECTOR (7 downto 0);
   signal DO: STD_LOGIC_VECTOR (11 downto 0);

   -- Counter to refresh data and multiplex digits
   signal CN_DISP, nCN_DISP: unsigned (19 downto 0);
   signal START_ADC, BCD_EN: STD_LOGIC;

   -- Multiplexed Display
   signal BCD: unsigned (3 downto 0);
   signal BCD_CTL, nBCD_CTL: STD_LOGIC_VECTOR(2 downto 0);

   -- MAQUINA DE ESTADOS PARA EL CONTROLADOR
   type estados is (REPOSO,E0,E1,E2,E3,E4,E5,E6);
   signal ESTADO, nESTADO: estados;

   -- SEÑALES PARA LA PLANIFICACION DEL CONTROL
   signal START_GC : std_logic; -- Señales de inicio
   signal TIMER, nTIMER : unsigned(8 DOWNTO 0); -- PWM
   constant N2 : unsigned(8 downto 0) := to_unsigned(480,9); -- Para iniciar GC
   constant N1 : unsigned(8 downto 0) := to_unsigned(400,9); -- Para iniciar ADConv

   -- SEÑALES DE E/S DEL CONTROLADOR
   signal REFERENCE: unsigned(8 downto 0):= to_unsigned(194,9); --Consigna
   signal nDC, DC : unsigned(8 DOWNTO 0); -- Duty cuantizado
   signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final
   signal nREFERENCE : unsigned (12 downto 0);

   -- SEÑALES PARA EL CONTROLADOR EN COMA FIJA USANDO FIXED_PKG
   signal nACC, ACC: sfixed(17 downto -24); -- 18_24
   signal nERROR0, nERROR1, nERROR2, ERROR0, ERROR1, ERROR2: sfixed(8 downto 0); --  9_0
   signal nDUTY1, nDUTY2, DUTY1, DUTY2 : sfixed(13 downto -8); -- 14_8
   signal nMUL1, MUL1 : sfixed(13 downto -8);
   signal nMUL2, MUL2 : sfixed(6 downto -16);
```

```
   -- CONSTANTES EN COMA FIJA USANDO FIXED PKG
   constant b2 : sfixed (6 downto -11) := to_sfixed (3.480   *6.6,   6,-11);   --7_11
   constant b1 : sfixed (6 downto -11) := to_sfixed (-7.658  *6.6,   6,-11);   --7_11
   constant b0 : sfixed (6 downto -11) := to_sfixed (4.197   *6.6,   6,-11);   --7_11
   constant a2 : sfixed (1 downto -16) := to_sfixed (0.5182,         1,-16);   --2_16
   constant a1 : sfixed (1 downto -16) := to_sfixed (-1.5182,        1,-16);   --2_16

   -- SEÑAL DE ACTIVACION DE LOS BIESTABLES
   signal EN : STD_LOGIC;
begin
   U1: adc2bcd PORT MAP (DA => DA, DO => DO);
   U2: adc PORT MAP (SDATA => SDATA, NCS => NCS, SCLK => SCLK,
                     CLK => CLK, RST => RST, START_ADC => START_ADC, DA => DA);
   ----------------------------
   -- PROCESO DE BIESTABLES --
   ----------------------------
   process (CLK, EN, RST)
   begin
      if (RST = '1') then
          EN <= '0';
         CN_DISP <= (others => '0');
         BCD_CTL <= "110";
         -- CONTROLADOR
         TIMER <= (others => '0');
         ESTADO <= REPOSO;
         ACC <= (others => '0');
         ERROR2<=(others => '0');
         ERROR1<= (others => '0');
         ERROR0<= (others => '0');
         DUTY2<= (others => '0');
         DUTY1<= (others => '0');
         DC <=(others => '0');
         DutyC <=(others => '0');
         MUL1 <=(others => '0');
         MUL2 <=(others => '0');
      elsif (CLK'event and CLK = '1') then
         if EN = '1' then
            CN_DISP <= nCN_DISP;
            BCD_CTL <= nBCD_CTL;
            -- CONTROLADOR
            TIMER <= nTIMER;
            ESTADO <= nESTADO;
            ACC  <= nACC;
            ERROR2<=nERROR2;
            ERROR1 <= nERROR1;
            ERROR0 <= nERROR0;
            DUTY2 <= nDUTY2;
            DUTY1 <= nDUTY1;
            DC <= nDC;
            DutyC <= nDutyC;
            MUL1 <= nMUL1;
            MUL2 <= nMUL2;
         end if;
         EN <= NOT EN;
      end if;
   end process;

   ----------------------------------------------------------------------
   --                          CONTROLADOR                             --
   ----------------------------------------------------------------------

   --SEÑAL INICIO ADC inicio
   START_ADC <= '1' when(TIMER = N1) else '0';
   -- SEÑAL DE ACTIVACIÓN DEL CONTROL
   START_GC <= '1' when (TIMER = N2) else '0';

   -- CONTROLADOR
   process (START_GC, DC, ESTADO, REFERENCE, DA,  ERROR0, ERROR1, ERROR2, ACC, DUTY1,
DUTY2, MUL1, MUL2)
   begin
         nESTADO <= ESTADO;
         nACC <= ACC;
         nERROR0 <= ERROR0;
         nERROR1 <= ERROR1;
```

```
        nERROR2 <= ERROR2;
        nDUTY1 <= DUTY1;
        nDUTY2 <= DUTY2;
        nDC <= DC;
        nMUL1 <= MUL1;
        nMUL2 <= MUL2;
    case ESTADO is
        when REPOSO =>
            if (START_GC = '1') then
                nERROR0 <= to_sfixed(signed(REFERENCE) - signed('0'&DA), 8, 0); -- 9_0
                nESTADO <= E0;
            end if;
        when E0 =>
            nMUL1 <= Resize(ERROR0, 13, -8);
            nMUL2 <= Resize(b0, 6, -16);
            nESTADO <= E1;
        when E1 =>
            nACC <= Resize(MUL1 * MUL2, 17, -24);
            nMUL1 <= Resize(ERROR1, 13, -8);
            nMUL2 <= Resize(b1, 6, -16);
            nESTADO <= E2;
        when E2 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
            nMUL1 <= Resize(ERROR2, 13, -8);
            nMUL2 <= Resize(b2, 6, -16);
            nESTADO <= E3;
        when E3 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
            nMUL1 <= Resize(DUTY1, 13,-8);
            nMUL2 <= Resize(-a1, 6, -16);
            nESTADO <= E4;
        when E4 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
            nMUL1 <= Resize(DUTY2, 13,-8);
            nMUL2 <= Resize(-a2, 6, -16);
            nESTADO <= E5;
        when E5 =>
            nACC <= Resize(MUL1 * MUL2 + ACC, 17, -24);
            nESTADO <= E6;
        when E6 =>
            --Actualizar valores
            nERROR1 <= ERROR0;
            nERROR2 <= ERROR1;
            nDUTY1 <= ACC(13 downto -8); -- 14_8 <= 18_24
            nDUTY2 <= DUTY1;
            --Cuantizador, coge solo la parte entera
            if (ACC(17 downto 0) > 450) then -- 18_0 <= 18_24
                nDC <= to_unsigned(450,9);
            elsif (ACC(17 downto 0) < 50) then -- 18_0 <= 18_24
                nDC <= to_unsigned(50,9);
            else
                nDC <= unsigned(ACC(8 downto 0)); -- 9_0 <= 18_24
            end if;
            nESTADO <= REPOSO;
    end case;
end process;

-- TIMER: Generación de DPWM
nTIMER <= (TIMER+1) when (TIMER < 499) else
          (others => '0');

-- Selección del Duty en función del SW
nDutyC <= DC when (TIMER = 499 and SW_CLOSED = '1') else --Lazo cerrado
       to_unsigned(200,9) when (SW_CLOSED = '0') else --Lazo abierto
       DutyC; --Resto del tiempo, no hacer nada.

-- SEÑALES DE CONTROL DEL CONVERTIDOR:

G1 <= '1' when (DutyC > TIMER) else '0'; -- G1
G2 <= '0';  -- G2 siempre abierto
G3 <= '1' when (SW_HI_LOAD = '1') else '0';
```

7.7 Versión en coma flotante con float_pkg, segmentada

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.fixed_float_types.all;
use IEEE.fixed_pkg.all ;
use IEEE.float_pkg.all ;

entity fpga is
   Port (CLK         : in  STD_LOGIC;
      RST         : in  STD_LOGIC;
      SDATA       : in  STD_LOGIC;
      SW_CLOSED   : in STD_LOGIC;
      SW_HI_LOAD  : in STD_LOGIC;
      NCS         : out  STD_LOGIC;
      SCLK        : out  STD_LOGIC;
      SSEG        : out  STD_LOGIC_VECTOR (6 downto 0);
      DP          : out  STD_LOGIC;
      ASEL        : out  STD_LOGIC_VECTOR (3 downto 0);
      G1          : out  STD_LOGIC;
      G2          : out  STD_LOGIC;
      G3          : out  STD_LOGIC
      );
end fpga;

architecture Behavioral of fpga is
   component adc2bcd is
      Port (DA: in STD_LOGIC_VECTOR (7 downto 0);
            DO: out STD_LOGIC_VECTOR (11 downto 0));
   end component;

   component adc is
      Port  (  SDATA : in  STD_LOGIC;
               NCS : out  STD_LOGIC;
               SCLK : out STD_LOGIC;
               CLK : in STD_LOGIC;
               RST : in STD_LOGIC;
               START_ADC : in STD_LOGIC;
               DA : out STD_LOGIC_VECTOR (7 downto 0)
            );
   end component;

   signal DA: STD_LOGIC_VECTOR (7 downto 0);
   signal DO: STD_LOGIC_VECTOR (11 downto 0);

   -- Counter to refresh data and multiplex digits
   signal CN_DISP, nCN_DISP: unsigned (19 downto 0);
   signal START_ADC, BCD_EN: STD_LOGIC;

   -- Multiplexed Display
   signal BCD: unsigned (3 downto 0);
   signal BCD_CTL, nBCD_CTL: STD_LOGIC_VECTOR(2 downto 0);

   -- MAQUINA DE ESTADOS PARA EL CONTROLADOR
   type estados is (REPOSO,E0,E1,E2,E3,E4,E5,E6,E7,E8);
   signal ESTADO, nESTADO: estados;

   -- SEÑALES PARA LA PLANIFICACION DEL CONTROL
   signal START_GC : std_logic; -- Señales de inicio
   signal TIMER, nTIMER : unsigned(8 downto 0); -- PWM
   constant N2 : unsigned(8 downto 0) := to_unsigned(480,9); -- Para iniciar GC
   constant N1 : unsigned(8 downto 0) := to_unsigned(400,9); -- Para iniciar ADConv

   -- SEÑALES DE E/S DEL CONTROLADOR
   signal REFERENCE: unsigned(8 downto 0):= to_unsigned(194,9); --Consigna
   signal nDC, DC : unsigned(8 downto 0); -- Duty cuantizado
   signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final

   -- SEÑALES PARA EL CONTROLADOR EN COMA FLOTANTE USANDO FLOAT_PKG
   signal nACC, ACC: float32;
   signal nERROR0, nERROR1, nERROR2, ERROR0, ERROR1, ERROR2 : float32;
   signal nDUTY1, nDUTY2, DUTY1, DUTY2 : float32;
   signal nMUL1, MUL1 : float32;
   signal nMUL2, MUL2 : float32;
```

```
   signal nPRODUCTO, PRODUCTO : float32;

   -- CONSTANTES EN COMA FLOTANTE USANDO FLOAT PKG
   constant b2 : float32 := to_float (3.480    *6.6);
   constant b1 : float32 := to_float (-7.658   *6.6);
   constant b0 : float32 := to_float (4.197    *6.6);
   constant a2 : float32 := to_float (0.5182);
   constant a1 : float32 := to_float (-1.5182);

   signal DIVIDER, pDIVIDER : unsigned (1 downto 0);

begin
   U1: adc2bcd PORT MAP (DA => DA, DO => DO);
   U2: adc PORT MAP (SDATA => SDATA, NCS => NCS, SCLK => SCLK,
         CLK => CLK, RST => RST, START_ADC => START_ADC, DA => DA);

   -- Contador de dos bits para dividir frecuencia CLK
   DIVIDER <= pDIVIDER + 1;
   ---------------------------
   -- PROCESO DE BIESTABLES --
   ---------------------------
   process (CLK, RST)
   begin
      if (RST = '1') then
         pDIVIDER <= (others => '0');
         CN_DISP <= (others => '0');
         BCD_CTL <= "110";
         -- CONTROLADOR
         TIMER <= (others => '0');
         ESTADO <= REPOSO;
         ACC <= (others => '0');
         ERROR2<=(others => '0');
         ERROR1<= (others => '0');
         ERROR0<= (others => '0');
         DUTY2<= (others => '0');
         DUTY1<= (others => '0');
         DC <=(others => '0');
         DutyC <=(others => '0');
         MUL1 <= (others => '0');
         MUL2 <= (others => '0');
         PRODUCTO <= (others => '0');

      elsif (CLK'event and CLK = '1') then
         pDIVIDER <= DIVIDER;
         if pDIVIDER = 0 OR pDIVIDER = 1 then
            CN_DISP <= nCN_DISP;
            BCD_CTL <= nBCD_CTL;
            TIMER <= nTIMER;
            DutyC <= nDutyC;
         end if;
         if pDIVIDER = 0 then
            -- BIESTABLES PARA OPERACIONES
            ESTADO <= nESTADO;
            ACC  <= nACC;
            ERROR2<=nERROR2;
            ERROR1 <= nERROR1;
            ERROR0 <= nERROR0;
            DUTY2 <= nDUTY2;
            DUTY1 <= nDUTY1;
            DC <= nDC;
            MUL1 <= nMUL1;
            MUL2 <= nMUL2;
            PRODUCTO <= nPRODUCTO;
        end if;
      end if;
   end process;
```

```
   ----------------------------------------------------------------
   --                CONTROLADOR                                 --
   ----------------------------------------------------------------

   --SEÑAL INICIO ADC inicio
   START_ADC <= '1' when(TIMER = N1) else '0';

   -- SEÑAL DE ACTIVACIÓN DEL CONTROL
   START_GC <= '1' when (TIMER = N2) else '0';

   -- CONTROLADOR
   process (START_GC, ESTADO, REFERENCE, DA,  ERROR0, ERROR1, ERROR2, ACC, DUTY1, DUTY2,
MUL1, MUL2, DC, PRODUCTO)
   begin
         nESTADO <= ESTADO;
         nACC <= ACC;
         nERROR0 <= ERROR0;
         nERROR1 <= ERROR1;
         nERROR2 <= ERROR2;
         nDUTY1 <= DUTY1;
         nDUTY2 <= DUTY2;
         nMUL1 <= MUL1;
         nMUL2 <= MUL2;
         nDC <= DC;
         nPRODUCTO <= PRODUCTO;
      case ESTADO is
         when REPOSO =>
            if (START_GC = '1') then
               nERROR0 <= to_float(signed(REFERENCE) - (signed('0'&DA)));
               nESTADO <= E0;
            end if;
         when E0 =>
            nMUL1 <= ERROR0;
            nMUL2 <= b0;
            nESTADO <= E1;
         when E1 =>
            nACC <= MUL1 * MUL2;
            nMUL1 <= ERROR1;
            nMUL2 <= b1;
            nESTADO <= E2;
         when E2 =>
            nPRODUCTO <= MUL1 * MUL2;
            nMUL1 <= ERROR2;
            nMUL2 <= b2;
            nESTADO <= E3;
         when E3 =>
            nACC <= PRODUCTO + ACC;
            nPRODUCTO <= MUL1 * MUL2;
            nMUL1 <= DUTY1;
            nMUL2 <= -a1;
            nESTADO <= E4;
         when E4 =>
            nACC <= PRODUCTO + ACC;
            nPRODUCTO <= MUL1 * MUL2;
            nMUL1 <= DUTY2;
            nMUL2 <= -a2;
            nESTADO <= E5;
         when E5 =>
            nACC <= PRODUCTO + ACC;
            nPRODUCTO <= MUL1 * MUL2;
            nESTADO <= E6;
         when E6 =>
            nACC <= PRODUCTO + ACC;
            nESTADO <= E7;
         when E7 =>
            --Actualizar valores
            nERROR1 <= ERROR0;
            nERROR2 <= ERROR1;
            nDUTY1 <= ACC;
            nDUTY2 <= DUTY1;
            nESTADO <= E8;
         when E8 =>
            if (ACC > to_float(450.0)) then
               nDC <= to_unsigned(450,9);
            elsif (ACC < to_float(50.0)) then
```

```
            nDC <= to_unsigned(50,9);
         else
            nDC <= to_unsigned (ACC,9);
         end if;
         nESTADO <= REPOSO;
   end case;
end process;


-- TIMER: Generación de DPWM
nTIMER <= (TIMER+1) when (TIMER < 499) else
          (others => '0');

-- Selección del Duty en función del SW
nDutyC <= nDC when (TIMER = 499 and SW_CLOSED = '1') else --Lazo cerrado
      to_unsigned(200,9) when (SW_CLOSED = '0') else --Lazo abierto
      DutyC; --Resto del tiempo, no hacer nada.

-- SEÑALES DE CONTROL DEL CONVERTIDOR:
G1 <= '1' when (DutyC > TIMER) else '0'; -- G1     DutyC
G2 <= '0';  -- G2 siempre abierto
G3 <= '1' when (SW_HI_LOAD = '1') else '0';
```

7.8 Versión en coma flotante con float_pkg, sencilla

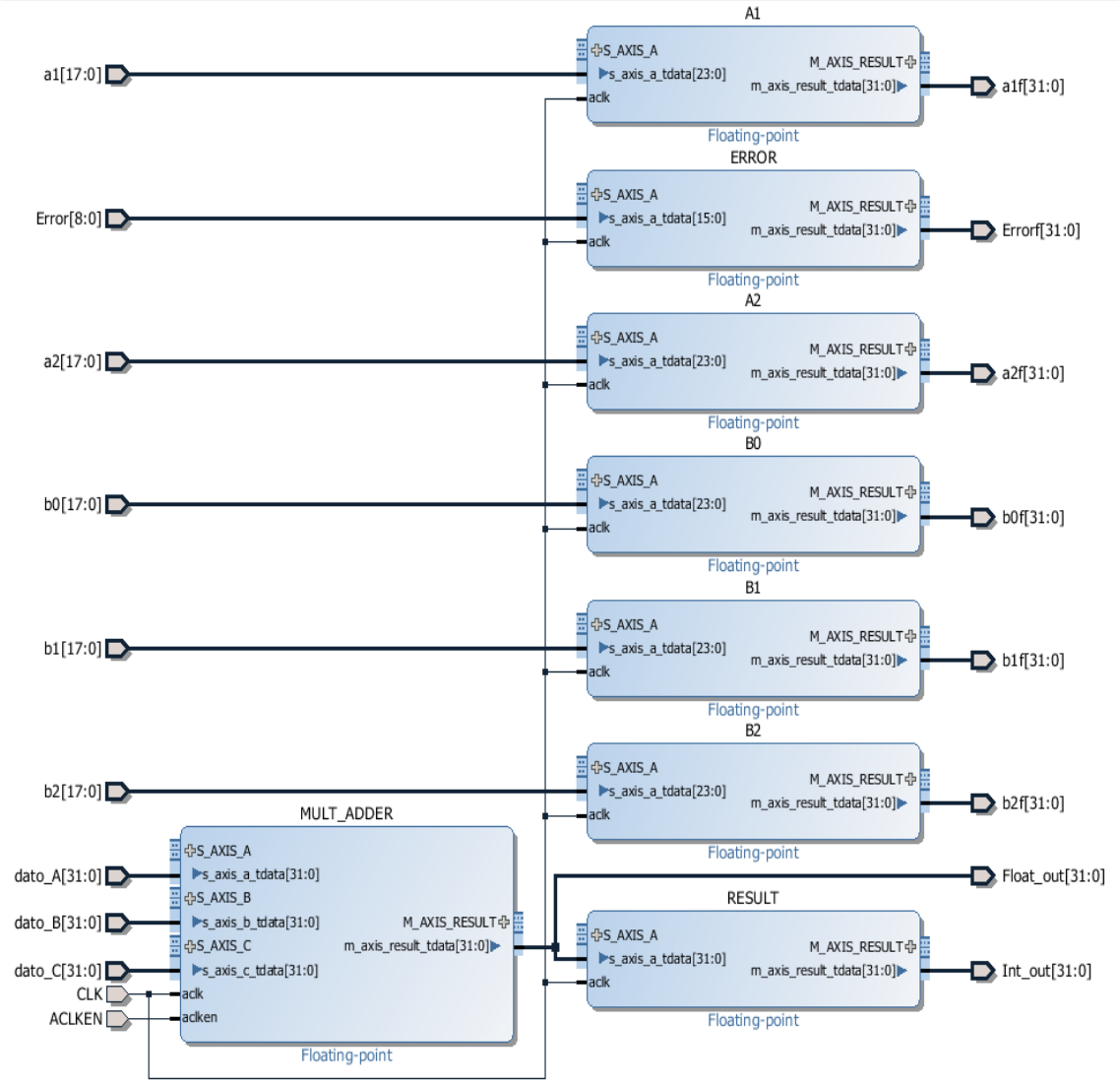
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.fixed_float_types.all;
use IEEE.fixed_pkg.all ;
use IEEE.float_pkg.all ;

entity fpga is
   Port (CLK: in  STD_LOGIC;
      RST: in  STD_LOGIC;
      SDATA: in  STD_LOGIC;
      SW_CLOSED: in STD_LOGIC;
      SW_HI_LOAD: in STD_LOGIC;
      NCS: out  STD_LOGIC;
      SCLK: out  STD_LOGIC;
      SSEG: out  STD_LOGIC_VECTOR (6 downto 0);
      DP: out  STD_LOGIC;
      ASEL: out  STD_LOGIC_VECTOR (3 downto 0);
      G1: out  STD_LOGIC;
      G2: out  STD_LOGIC;
      G3: out  STD_LOGIC
      );
end fpga;

architecture Behavioral of fpga is
   component adc2bcd is
      Port (DA: in STD_LOGIC_VECTOR (7 downto 0);
            DO: out STD_LOGIC_VECTOR (11 downto 0));
   end component;

   component adc is
      Port (SDATA : in  STD_LOGIC;
         NCS : out  STD_LOGIC;
         SCLK : out STD_LOGIC;
         CLK : in STD_LOGIC;
         RST : in STD_LOGIC;
         START_ADC : in STD_LOGIC;
         DA : out STD_LOGIC_VECTOR (7 downto 0)
         );
   end component;

   signal DA: STD_LOGIC_VECTOR (7 downto 0);
   signal DO: STD_LOGIC_VECTOR (11 downto 0);

   -- Counter to refresh data and multiplex digits
   signal CN_DISP, nCN_DISP: unsigned (19 downto 0);
   signal START_ADC, BCD_EN: STD_LOGIC;

   -- Multiplexed Display
   signal BCD: unsigned (3 downto 0);
   signal BCD_CTL, nBCD_CTL: STD_LOGIC_VECTOR(2 downto 0);

   -- MAQUINA DE ESTADOS PARA EL CONTROLADOR
   type estados is (REPOSO,E0,E1,E2);
   signal ESTADO, nESTADO: estados;

   -- SEÑALES PARA LA PLANIFICACION DEL CONTROL
   signal START_GC : std_logic; -- Señales de inicio
   signal TIMER, nTIMER : unsigned(8 downto 0); -- PWM
   constant N3 : unsigned(8 downto 0) := to_unsigned(490,9); -- Fin lógica combinacional
   constant N2 : unsigned(8 downto 0) := to_unsigned(480,9); -- Para iniciar GC
   constant N1 : unsigned(8 downto 0) := to_unsigned(400,9); -- Para iniciar ADConv

   -- SEÑALES DE E/S DEL CONTROLADOR
   signal REFERENCE: unsigned(8 downto 0):= to_unsigned(194,9); --Consigna
   signal nDC, DC : unsigned(8 downto 0); -- Duty cuantizado
   signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final

   -- SEÑALES PARA EL CONTROLADOR EN COMA FLOTANTE USANDO FLOAT PKG
   signal n_error, n_error0, n_error1, n_error2, error, error0, error1, error2: float32;
   signal n_duty0, duty0, n_duty1, n_duty2, duty1, duty2 : float32;
```

```
   -- CONSTANTES EN COMA FLOTANTE USANDO FLOAT PKG
   constant b2 : float32 := to_float (3.480    *6.6);
   constant b1 : float32 := to_float (-7.658   *6.6);
   constant b0 : float32 := to_float (4.197    *6.6);
   constant a2 : float32 := to_float (0.5182);
   constant a1 : float32 := to_float (-1.5182);
   -- SEÑAL DE ACTIVACION DE LOS BIESTABLES
   signal EN : std_logic;

begin
   U1: adc2bcd PORT MAP (DA => DA, DO => DO);
   U2: adc PORT MAP (SDATA => SDATA, NCS => NCS, SCLK => SCLK,
         CLK => CLK, RST => RST, START_ADC => START_ADC, DA => DA);

   ----------------------------
   -- PROCESO DE BIESTABLES --
   ----------------------------
   process (CLK, EN, RST)
   begin
      if (RST = '1') then
         EN <= '0';
         CN_DISP <= (others => '0');
         BCD_CTL <= "110";
         -- CONTROLADOR
         TIMER <= (others => '0');
         ESTADO <= REPOSO;
         duty0 <= (others => '0');
         duty1 <= (others => '0');
         duty2 <= (others => '0');
         error1 <= (others => '0');
         error2 <= (others => '0');
         error0 <= (others => '0');
         error <= (others => '0');
         DC <=(others => '0');
         DutyC <=(others => '0');

      elsif (CLK'event and CLK = '1') then
         if EN = '1' then
            CN_DISP <= nCN_DISP;
            BCD_CTL <= nBCD_CTL;
            TIMER <= nTIMER;
            DutyC <= nDutyC;
            -- CONTROLADOR
            ESTADO <= nESTADO;
            duty0 <= n_duty0;
            duty1 <= n_duty1;
            duty2 <= n_duty2;
            error1 <= n_error1;
            error2 <= n_error2;
            error0 <= n_error0;
            error <= n_error;
            DC <= nDC;
         end if;
         EN <= NOT EN;
      end if;
   end process;


   ---------------------------------------------------------------------
   --                CONTROLADOR                        --
   ---------------------------------------------------------------------

   --SEÑAL INICIO ADC inicio
   START_ADC <= '1' when(TIMER = N1) else '0';

   -- SEÑAL DE ACTIVACIÓN DEL CONTROL
   START_GC <= '1' when (TIMER = N2) else '0';

   -- CONTROLADOR
   process (START_GC, ESTADO, REFERENCE, DA, error, error0, error1, error2, duty0,
duty1, duty2, TIMER)
   begin
         nESTADO <= ESTADO;
         n_duty0 <= duty0;
         n_duty1 <= duty1;
```

```
        n_duty2 <= duty2;
        n_error1 <= error1;
        n_error2 <= error2;
        n_error0 <= error0;
        n_error <= error;
        nDC <= DC;

    case ESTADO is
       when REPOSO =>
          if (START_GC = '1') then
             n_error <= to_float(signed(REFERENCE) - (signed('0'&DA)));
             nESTADO <= E0;
          end if;
       when E0 =>
          --Actualizar valores
          n_error0 <= error;
          n_error1 <= error0;
          n_error2 <= error1;
          n_duty1 <= duty0;
          n_duty2 <= duty1;
          nESTADO <= E1;
       when E1 =>
       n_duty0 <= error0*b0 + error1*b1 + error2*b2 - duty1*a1 - duty2*a2;
          if (TIMER = 450) then
             nESTADO <= E2;
          end if;
       when E2 =>
          if (duty0 > to_float(450.0)) then
             nDC <= to_unsigned(450,9);
          elsif (duty0 < to_float(50.0)) then
             nDC <= to_unsigned(50,9);
          else
             nDC <= to_unsigned (duty0,9);
          end if;
          nESTADO <= REPOSO;
    end case;
end process;


-- TIMER: Generación de DPWM
nTIMER <= (TIMER+1) when (TIMER < 499) else
          (others => '0');


-- Selección del Duty en función del SW
nDutyC <= nDC when (TIMER = 499 and SW_CLOSED = '1') else --Lazo cerrado
       to_unsigned(200,9) when (SW_CLOSED = '0') else --Lazo abierto
       DutyC; --Resto del tiempo, no hacer nada.

-- SEÑALES DE CONTROL DEL CONVERTIDOR:
G1 <= '1' when (DutyC > TIMER) else '0'; -- G1     DutyC
G2 <= '0';  -- G2 siempre abierto
G3 <= '1' when (SW_HI_LOAD = '1') else '0';
```

7.9 Versión con bloques IP de coma flotante

7.9.1 Diagrama de bloques



7.9.2 VHDL

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

entity fpga is
   Port (CLK: in  STD_LOGIC;
      RST: in  STD_LOGIC;
      SDATA: in  STD_LOGIC;
      SW_CLOSED: in STD_LOGIC;
      SW_HI_LOAD: in STD_LOGIC;
      NCS: out  STD_LOGIC;
      SCLK: out  STD_LOGIC;
      SSEG: out  STD_LOGIC_VECTOR (6 downto 0);
      DP: out  STD_LOGIC;
      ASEL: out  STD_LOGIC_VECTOR (3 downto 0);
      G1: out  STD_LOGIC;
      G2: out  STD_LOGIC;
      G3: out  STD_LOGIC
      );
end fpga;
```

```
architecture Behavioral of fpga is
   component controller_wrapper is
      port (ACLKEN : in STD_LOGIC;
         CLK : in STD_LOGIC;
         Error : in STD_LOGIC_VECTOR ( 8 downto 0 );
         Errorf : out STD_LOGIC_VECTOR ( 31 downto 0 );
         Float_out : out STD_LOGIC_VECTOR ( 31 downto 0 );
         Int_out : out STD_LOGIC_VECTOR ( 31 downto 0 );
         a1 : in STD_LOGIC_VECTOR ( 17 downto 0 );
         a1f : out STD_LOGIC_VECTOR ( 31 downto 0 );
         a2 : in STD_LOGIC_VECTOR ( 17 downto 0 );
         a2f : out STD_LOGIC_VECTOR ( 31 downto 0 );
         b0 : in STD_LOGIC_VECTOR ( 17 downto 0 );
         b0f : out STD_LOGIC_VECTOR ( 31 downto 0 );
         b1 : in STD_LOGIC_VECTOR ( 17 downto 0 );
         b1f : out STD_LOGIC_VECTOR ( 31 downto 0 );
         b2 : in STD_LOGIC_VECTOR ( 17 downto 0 );
         b2f : out STD_LOGIC_VECTOR ( 31 downto 0 );
         dato_A : in STD_LOGIC_VECTOR ( 31 downto 0 );
         dato_B : in STD_LOGIC_VECTOR ( 31 downto 0 );
         dato_C : in STD_LOGIC_VECTOR ( 31 downto 0 ));
   end component controller_wrapper;

   component adc2bcd is
      Port (DA: in STD_LOGIC_VECTOR (7 downto 0);
            DO: out STD_LOGIC_VECTOR (11 downto 0));
   end component;

   component adc is
      Port( SDATA : in  STD_LOGIC;
         NCS : out  STD_LOGIC;
         SCLK : out STD_LOGIC;
         CLK : in STD_LOGIC;
         RST : in STD_LOGIC;
         START_ADC : in STD_LOGIC;
         DA : out STD_LOGIC_VECTOR (7 downto 0)
       );
   end component;

   signal DA: STD_LOGIC_VECTOR (7 downto 0);
   signal DO: STD_LOGIC_VECTOR (11 downto 0);

   -- Counter to refresh data and multiplex digits
   signal CN_DISP, nCN_DISP: unsigned (19 downto 0);
   signal START_ADC, BCD_EN: STD_LOGIC;

   -- Multiplexed Display
   signal BCD: unsigned (3 downto 0);
   signal BCD_CTL, nBCD_CTL: STD_LOGIC_VECTOR(2 downto 0);

   -- SEÑALES PARA LA PLANIFICACION DEL CONTROL
   signal START_GC : std_logic; -- Señales de inicio
   signal TIMER, nTIMER : unsigned(8 DOWNTO 0); -- PWM
   constant N2 : unsigned(8 downto 0) := to_unsigned(480,9); -- Para iniciar GC
   constant N1 : unsigned(8 downto 0) := to_unsigned(400,9); -- Para iniciar ADConv

   -- MAQUINA DE ESTADOS PARA EL CONTROLADOR
   type estados is (REPOSO,E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10);
   signal ESTADO, nESTADO: estados;

   -- SEÑALES DEL CONTROLADOR
   signal REFERENCE: unsigned(8 downto 0) := to_unsigned(194,9); --Consigna
   signal nDC, DC : unsigned(8 downto 0); -- Duty cuantizado
   signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final

   -- SEÑALES PARA EL CONTROLADOR
   signal ERROR0, ERROR1, ERROR2, DUTY1, DUTY2: std_logic_vector(31 downto 0);
   signal nERROR0, nERROR1, nERROR2, nDUTY1, nDUTY2: std_logic_vector(31 downto 0);
   signal Errorf, Int_out: std_logic_vector(31 downto 0);
   signal nError, Error: STD_LOGIC_VECTOR (8 downto 0);
   signal b0f, b1f, b2f, a1f, a2f: STD_LOGIC_VECTOR (31 downto 0);
   signal ndato_A, dato_A: STD_LOGIC_VECTOR (31 downto 0);
   signal ndato_B, dato_B: STD_LOGIC_VECTOR (31 downto 0);
   signal ndato_C, dato_C: STD_LOGIC_VECTOR (31 downto 0);
```

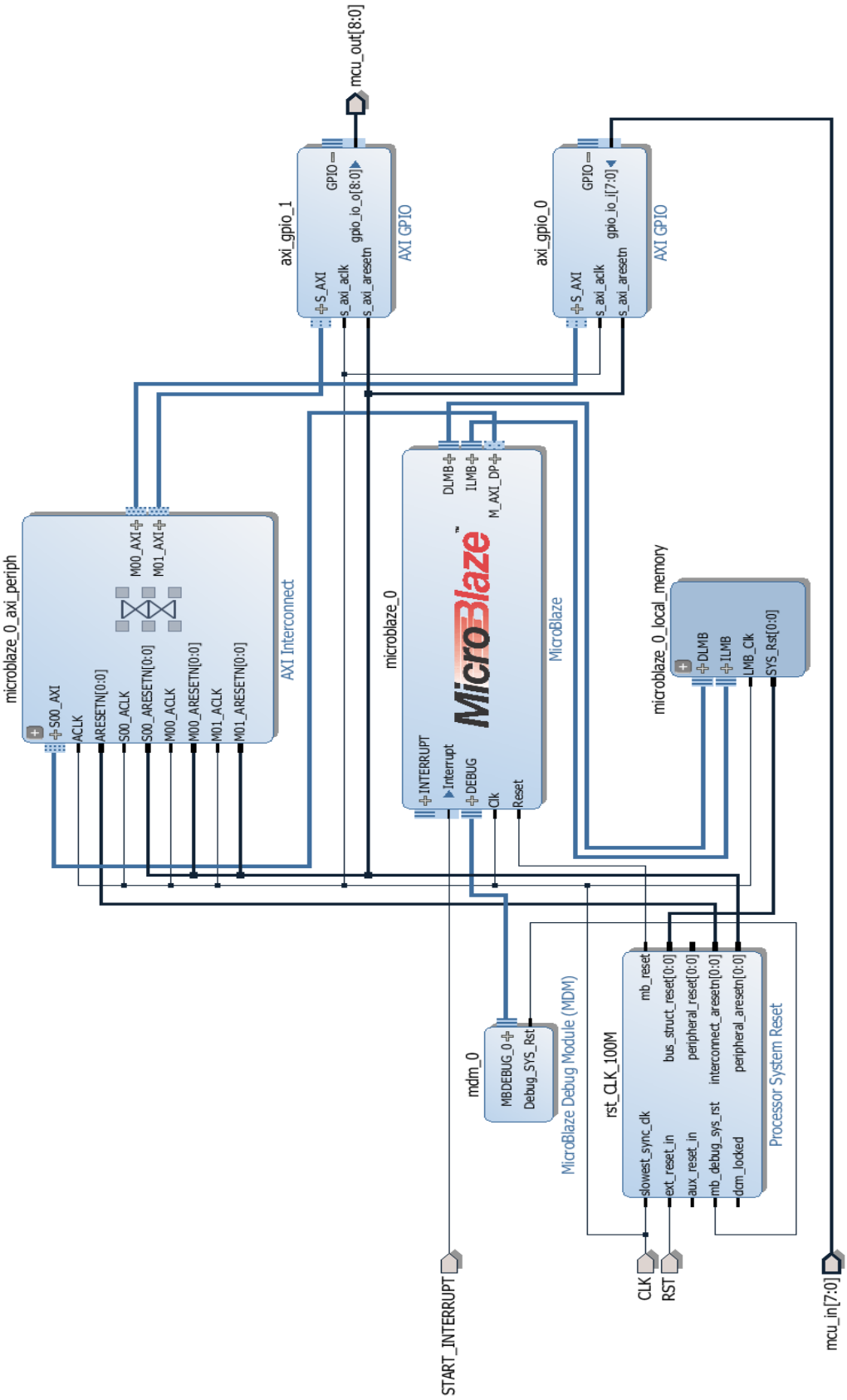
```
    signal Float_out: STD_LOGIC_VECTOR (31 downto 0);
    signal ACLKEN: STD_LOGIC;

    -- CONSTANTES EN COMA FIJA USANDO FIXED_PKG
    signal b2: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (3.480*6.6,
    6,-11));   --7_11
    signal b1: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (-7.658*6.6,
    6,-11));   --7_11
    signal b0: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (4.197*6.6,
    6,-11));   --7_11
    signal a2: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (-0.5182,
    1,-16));   --2_16
    signal a1: STD_LOGIC_VECTOR (17 downto 0):= STD_LOGIC_VECTOR(to_sfixed (1.5182,
    1,-16));   --2_16

    -- SEÑAL DE ACTIVACION DE LOS BIESTABLES
    signal EN: STD_LOGIC;

begin
    U1: adc2bcd PORT MAP (DA => DA, DO => DO);
    U2: adc PORT MAP (SDATA => SDATA, NCS => NCS, SCLK => SCLK,
                      CLK => CLK, RST => RST, START_ADC => START_ADC, DA => DA);
    U3: controller_wrapper port map (CLK => EN, ACLKEN => ACLKEN,
                  dato_A(31 downto 0) => dato_A(31 downto 0),
                  dato_B(31 downto 0) => dato_B(31 downto 0),
                  dato_C(31 downto 0) => dato_C(31 downto 0),
                  Float_out(31 downto 0) => Float_out(31 downto 0),
                  b2(17 downto 0) => b2 (17 downto 0),
                  b1(17 downto 0) => b1 (17 downto 0),
                  b0(17 downto 0) => b0 (17 downto 0),
                  a2(17 downto 0) => a2 (17 downto 0),
                  a1(17 downto 0) => a1 (17 downto 0),
                  b2f(31 downto 0) => b2f (31 downto 0),
                  b1f(31 downto 0) => b1f (31 downto 0),
                  b0f(31 downto 0) => b0f (31 downto 0),
                  a2f(31 downto 0) => a2f (31 downto 0),
                  a1f(31 downto 0) => a1f (31 downto 0),
                  Error(8 downto 0) => Error (8 downto 0),
                  Errorf(31 downto 0) => Errorf (31 downto 0),
                  Int_out(31 downto 0) => Int_out (31 downto 0)
                  );
    ----------------------------
    -- PROCESO DE BIESTABLES --
    ----------------------------
    process (CLK, EN, RST)
    begin
       if (RST = '1') then
           EN <= '0';
          CN_DISP <= (others => '0');
          BCD_CTL <= "110";
          -- CONTROLADOR
          TIMER <= (others => '0');
          ESTADO <= REPOSO;
          ERROR2<=(others => '0');
          ERROR1<= (others => '0');
          ERROR0<= (others => '0');
          DUTY2<= (others => '0');
          DUTY1<= (others => '0');
          DC <= (others => '0');
          DutyC <= (others => '0');
          Error <= (others => '0');
          dato_A <= (others => '0');
          dato_B <= (others => '0');
          dato_C <= (others => '0');
       elsif (CLK'event and CLK = '1') then
          if EN = '1' then
             CN_DISP <= nCN_DISP;
             BCD_CTL <= nBCD_CTL;
             -- CONTROLADOR
             TIMER <= nTIMER;
             ESTADO <= nESTADO;
             ERROR2<=nERROR2;
             ERROR1 <= nERROR1;
             ERROR0 <= nERROR0;
             DUTY2 <= nDUTY2;
```

```
            DUTY1 <= nDUTY1;
            DC <= nDC;
            DutyC <= nDutyC;
            Error <= nError;
            dato_A <= ndato_A;
            dato_B <= ndato_B;
            dato_C <= ndato_C;
         end if;
         EN <= NOT EN;
      end if;
   end process;

   ---------------------------------------------------------------------
   --                          CONTROLADOR                            --
   ---------------------------------------------------------------------

   --SEÑAL INICIO ADC inicio
   --START ADC <= '1' when (TIMER = N1) else '0';
   START_ADC <= '1' when(TIMER = N1) else '0';
   -- SEÑAL DE ACTIVACIÓN DEL CONTROL
   START_GC <= '1' when (TIMER = N2) else '0';

   -- Logica combinacional para el control del bloque IP float
   ndato_C <= (others => '0') when (ESTADO = E0) else Float_out;
   ACLKEN <= '1' when (ESTADO = E0 or ESTADO = E1 or ESTADO = E2 or ESTADO = E3 or
         ESTADO = E4 or ESTADO = E5 or ESTADO = E6 or ESTADO = E7) else '0';

   -- CONTROLADOR
   process (START_GC, ESTADO, REFERENCE, DA, Errorf, ERROR0, ERROR1, ERROR2, DUTY1,
            Error, dato_A, dato_B, DUTY2, DC, Float_out, Int_out, dato_A, dato_B,
            b0f, b1f, b2f, a1f, a2f)
   begin
         nESTADO <= ESTADO;
         nERROR0 <= ERROR0;
         nERROR1 <= ERROR1;
         nERROR2 <= ERROR2;
         nDUTY1 <= DUTY1;
         nDUTY2 <= DUTY2;
         nError <= Error;
         ndato_A <= dato_A;
         ndato_B <= dato_B;
         nDC <= DC;
      case ESTADO is
         when REPOSO =>
            if (START_GC = '1') then
               nESTADO <= E0;
            end if;
         when E0 => --ESTADO NECESARIO PARA RESETEAR EL BLOQUE IP
            nError <= std_logic_vector(to_sfixed((signed(REFERENCE) -
                        signed('0'&DA)),8,0));
            ndato_A <= (others => '0');
            ndato_B <= (others => '0');
            nESTADO <= E1;
         when E1 =>
            nERROR0 <= Errorf;
            ndato_A <= (others => '0');
            ndato_B <= (others => '0');
            nESTADO <= E2;
         when E2 =>
            ndato A <= ERROR0;
            ndato_B <= b0f;
            nESTADO <= E3;
         when E3 =>
            ndato_A <= ERROR1;
            ndato_B <= b1f;
            nESTADO <= E4;
         when E4 =>
            ndato_A <= ERROR2;
            ndato_B <= b2f;
            nESTADO <= E5;
         when E5 =>
            ndato_A <= DUTY1;
            ndato_B <= a1f;
            nESTADO <= E6;
         when E6 =>
```

```
            ndato_A <= DUTY2;
            ndato_B <= a2f;
            nESTADO <= E7;
         when E7 => --ESTADO NECESARIO PARA QUE EL BLOQUE IP ACTUALICE SU SALIDA
            nESTADO <= E8;
         when E8 =>
            --Actualizar valores
            nERROR1 <= ERROR0;
            nERROR2 <= ERROR1;
            nDUTY1 <= Float_out;
            nDUTY2 <= DUTY1;
            nESTADO <= E9;
         when E9 =>
            nESTADO <= E10;
         when E10 =>
            --Cuantizador, coge solo la parte entera
            if (signed(Int_out) > 450) then
               nDC <= to_unsigned(450,9);
            elsif (signed(Int_out) < 50) then
               nDC <= to_unsigned(50,9);
            else
               nDC <= unsigned(Int_out(8 downto 0));
            end if;
            nESTADO <= REPOSO;
      end case;

   end process;

   -- TIMER: Generación de DPWM
   nTIMER <= (TIMER+1) when (TIMER < 499) else
             (others => '0');

   -- Selección del Duty en función del SW
   nDutyC <= DC when (TIMER = 499 and SW_CLOSED = '1') else --Lazo cerrado
         to_unsigned(200, 9) when (SW_CLOSED = '0') else --Lazo abierto
         DutyC; --Resto del tiempo, no hacer nada.

   -- SEÑALES DE CONTROL DEL CONVERTIDOR:

   G1 <= '1' when (DutyC > TIMER) else '0'; -- G1
   G2 <= '0';  -- G2 siempre abierto
   G3 <= '1' when (SW_HI_LOAD = '1') else '0';
```

7.10 Versión con MicroBlaze

7.10.1 Diagrama de bloques



7.10.2 VHDL

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

entity fpga is
   Port (CLK: in  STD_LOGIC;
      RST: in  STD_LOGIC;
      SDATA: in  STD_LOGIC;
      SW_CLOSED: in STD_LOGIC;
      SW_HI_LOAD: in STD_LOGIC;
      NCS: out  STD_LOGIC;
      SCLK: out  STD_LOGIC;
      SSEG: out  STD_LOGIC_VECTOR (6 downto 0);
      DP: out  STD_LOGIC;
      ASEL: out  STD_LOGIC_VECTOR (3 downto 0);
      G1: out  STD_LOGIC;
      G2: out  STD_LOGIC;
      G3: out  STD_LOGIC
      );
end fpga;

architecture Behavioral of fpga is
   component microprocessor_wrapper is
      port (mcu_in : in STD_LOGIC_VECTOR (7 downto 0);
         mcu_out : out STD_LOGIC_VECTOR (8 downto 0);
         RST : in STD_LOGIC;
         CLK : in STD_LOGIC;
         START_INTERRUPT : in STD_LOGIC
      );
   end component microprocessor_wrapper;

   component adc2bcd is
      Port (DA: in STD_LOGIC_VECTOR (7 downto 0);
            DO: out STD_LOGIC_VECTOR (11 downto 0));
   end component;

   component adc is
      Port  (  SDATA : in  STD_LOGIC;
               NCS : out  STD_LOGIC;
               SCLK : out STD_LOGIC;
               CLK : in STD_LOGIC;
               RST : in STD_LOGIC;
               START_ADC : in STD_LOGIC;
               DA : out STD_LOGIC_VECTOR (7 downto 0)
            );
   end component;

   signal DA: STD_LOGIC_VECTOR (7 downto 0);
   signal DO: STD_LOGIC_VECTOR (11 downto 0);

   -- Counter to refresh data and multiplex digits
   signal CN_DISP, nCN_DISP: unsigned (19 downto 0);
   signal START_ADC, BCD_EN: STD_LOGIC;

   -- Multiplexed Display
   signal BCD: unsigned (3 downto 0);
   signal BCD_CTL, nBCD_CTL: STD_LOGIC_VECTOR(2 downto 0);

   -- SEÑALES PARA LA PLANIFICACION DEL CONTROL
   signal START_GC : std_logic; -- Señal de inicio
   signal TIMER, nTIMER : unsigned(8 downto 0); -- PWM
   constant N2 : unsigned(8 downto 0) := to_unsigned(100,9); -- Para iniciar GC
   constant N1 : unsigned(8 downto 0) := to_unsigned(20,9); -- Para iniciar ADConv

   -- SEÑALES DEL CONTROLADOR
   signal nDutyC, DutyC: unsigned (8 downto 0); -- Duty command final
   signal mcu_in: STD_LOGIC_VECTOR (7 downto 0);
   signal mcu_out: STD_LOGIC_VECTOR (8 downto 0);
   signal START_INTERRUPT: STD_LOGIC;
   signal EN : std_logic;
begin
   U1: adc2bcd PORT MAP (DA => DA, DO => DO);
   U2: adc PORT MAP (SDATA => SDATA, NCS => NCS, SCLK => SCLK,
```

```
         CLK => CLK, RST => RST, START_ADC => START_ADC, DA => DA);
U3: microprocessor_wrapper port map (CLK => CLK, RST => RST,
         START_INTERRUPT => START_INTERRUPT,
         mcu_in => mcu_in, mcu_out => mcu_out);

----------------------------
-- PROCESO DE BIESTABLES --
----------------------------
process (CLK, RST)
begin
   if (RST = '1') then
       EN <= '0';
      CN_DISP <= (others => '0');
      BCD_CTL <= "110";
      -- CONTROLADOR
      TIMER <= (others => '0');
      DutyC <=(others => '0');
   elsif (CLK'event and CLK = '1') then
      if EN = '1' then
         CN_DISP <= nCN_DISP;
         BCD_CTL <= nBCD_CTL;
         -- CONTROLADOR
         TIMER <= nTIMER;
         DutyC <= nDutyC;
      end if;
      EN <= not EN;
   end if;
end process;


-----------------------------------------------------------------------
--                          CONTROLADOR                              --
-----------------------------------------------------------------------

--SEÑAL INICIO ADC inicio
START_ADC <= '1' when(TIMER = N1) else '0';

-- Logica combinacional para el control del microcontrolador
-- Es necesario mantener el pulso para que funcione la interrupcion
START_INTERRUPT <= '1' when (TIMER > N2) and (TIMER < N2+10) and (SW_CLOSED = '1')
              else '0';
mcu_in <= DA;

-- TIMER: Generación de DPWM
nTIMER <= (TIMER + 1) when (TIMER < 499) else  (others => '0');

-- Selección del Duty en función del SW
nDutyC <= unsigned(mcu_out) when (TIMER= 499 and SW_CLOSED = '1') else --Lazo cerrado
      to_unsigned(200,9) when (SW_CLOSED = '0') else --Lazo abierto
      DutyC; --Resto del tiempo, no hacer nada.

-- SEÑALES DE CONTROL DEL CONVERTIDOR:
G1 <= '1' when (DutyC > TIMER) else '0'; -- G1
G2 <= '0';  -- G2 siempre abierto
G3 <= '1' when (SW_HI_LOAD = '1') else '0';
```

7.10.3 Código C

```
#include "xparameters.h"
#include "xgpio.h"

/************************** Function Prototypes ******************************/
void mi_ISR( void ) __attribute__ ((interrupt_handler));

/************************** Variable Definitions *****************************/
XGpio GPInput, GPOutput; /* The Instance of the GPIO Driver */
uint32_t ADC, DC;

float ADC_f;
float error0 = 0.0;
float error1 = 0.0;
float error2 = 0.0;
float duty0  = 0.0;
float duty1  = 0.0;
float duty2  = 0.0;

const float Ref = 194.0;
const float b2 = 3.480  *6.6; //22.968
const float b1 = -7.658 *6.6; //-50.5428
const float b0 = 4.197  *6.6; //27.7002
const float a2 = 0.5182;
const float a1 = -1.5182;
const float duty_max = 450.0;
const float duty_min = 50.0;

int main()
{
   //Initialize IOs
   XGpio_Initialize(&GPInput, XPAR_AXI_GPIO_0_DEVICE_ID);
   XGpio_Initialize(&GPOutput, XPAR_AXI_GPIO_1_DEVICE_ID);

   //Enable interrupts
   microblaze_enable_interrupts();

   while(1)
   {
   }
   return 0;
}

void mi_ISR (void)
{
   //Leer ADC
   ADC = XGpio_DiscreteRead(&GPInput, 1);
   ADC_f = (float)ADC;

   //Calcular error
   error0 = (Ref - ADC_f);

   //Ecuacion del controlador
   duty0 = -a1*duty1 - a2*duty2 + b0*error0 + b1*error1 + b2*error2;

   //Actualizacion de variables
   duty2 = duty1;
   duty1 = duty0;
   error2 = error1;
   error1 = error0;

   //Cuantizador para el ciclo de servicio
   if (duty0 > duty_max)
      duty0 = duty_max;
   else if (duty0 < duty_min)
      duty0 = duty_min;

   //Salida del controlador
   DC = (int) duty0;
   XGpio_DiscreteWrite(&GPOutput, 1, DC);
}
```