Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2010, Article ID 593264, 11 pages
doi:10.1155/2010/593264

Research Article

Word Length Selection Method for Controller Implementation on FPGAs Using the VHDL-2008 Fixed-Point and Floating-Point Packages

I. Urriza, L. A. Barragán, D. Navarro, J. I. Artigas, and O. Lucia

Departamento de Ingeniería Electrónica y Comunicaciones, Universidad de Zaragoza, María de Luna 1, 50018 Zaragoza, Spain

Correspondence should be addressed to I. Urriza, urriza@unizar.es

Received 23 December 2009; Revised 9 April 2010; Accepted 29 July 2010

Academic Editor: Miriam Leeser

This paper presents a word length selection method for the implementation of digital controllers in both fixed-point and floating-point hardware on FPGAs. This method uses the new types defined in the VHDL-2008 fixed-point and floating-point packages. These packages allow customizing the word length of fixed and floating point representations and shorten the design cycle simplifying the design of arithmetic operations. The method performs bit-true simulations in order to determine the word length to represent the constant coefficients and the internal signals of the digital controller while maintaining the control system specifications. A mixed-signal simulation tool is used to simulate the closed loop system as a whole in order to analyze the impact of the quantization effects and loop delays on the control system performance. The method is applied to implement a digital controller for a switching power converter. The digital circuit is implemented on an FPGA, and the simulations are experimentally verified.

1. Introduction

The advances in FPGAs technology [1, 2] have had a big impact on the implementation of digital control algorithms [3]. With the introduction of advanced FPGA architectures, it has become affordable to implement fixed-point [4–8], and floating-point [9–13] data formats and operations in these FPGAs. In addition to implementing these algorithms directly in hardware, FPGAs also allow an HW/SW realization of these algorithms using soft-core or hard-core processors. This work will focus on the first option.

The best representation, fixed-point or floating-point, depends on the application. So, the designer must analyze which is the suitable representation for his application. On one hand, floating-point implementations will consume more FPGA hardware resources and achieve lower speed. On the other hand, digital controllers are designed considering real number coefficients and signals, and floating-point to fixed-point conversion may become a difficult and time-consuming process. In fact, this process has been identified in a recent survey [14] as the most difficult aspect of implementing a fixed-point algorithm on an FPGA.

VHDL is a standard Hardware Description Language that allows high flexibility and technology independence. However, VHDL hardware description is considered as a low abstraction level tool and, in order to reduce design time and effort, some fixed-point implementations use tools such as Altera DSP Builder [4] Xilinx System Generator [6] that directly generates VHDL from Matlab/Simulink, or Handle-C [7].

This work shows how to implement digital controllers in fixed-point using the VHDL-2008 fixed-point package *fixed_pkg* [15], and in floating point using the VHDL-2008 floating-point package *float_pkg* [16]. These packages are included in the IEEE Standard VHDL LRM 1076–2008. The utilization of these packages simplifies the design of arithmetic operations, shortens the design cycle, and makes VHDL competitive with other alternatives. These packages also allow customizing the word length of fixed and floating point representations. This method can also be applied

to the implementation of any infinite impulse response filter.

Several techniques based on bit-true simulations or analytical analysis have been proposed to determine the word length in fixed-point implementations of digital signal processing applications [17–23] and digital controllers [24–27]. There are fewer methods [28] that offer a uniform treatment for bit-width selection of both fixed-point and floating-point designs. However, in [28] analytical analysis is applied to data flows without feedback, while in this work, the whole system is simulated in closed loop.

We perform bit-true simulations using the new types defined in the VHDL-2008 fixed-point and floating-point packages, and a word length that maintains an acceptable closed-loop performance in both cases is determined using an approach very closely related to the method presented in [17, 22]. Thus, our method covers both fixed-point and floating-point word length selection. Except on [25, 27], Matlab/Simulink or C/C++ is used to perform bit-true simulations, and then VHDL is automatically generated by the tool. Using the types defined in the packages has the advantage that the simulated code is also the synthesized one. This paper further develops the work described in [12, 27], developing a framework that offers and unifies treatment for fixed-point and floating-point designs.

The remainder of this paper is organized as follows. Section 2 provides a brief description of fixed-point and floating-point numerical representations. Section 3 shows the traditional approach to implement arithmetic operations in VHDL using the *numeric_std* package, and how the new packages simplify the design of fixed and floating-point operations. Section 4 presents the proposed word length selection procedure that is applied to an application example in Section 5. Section 6 shows the experimental results obtained using the proposed method and VHDL-2008 packages, and finally in Section 7 some conclusions are drawn.

2. Numerical Representations

Figure 1 shows the data format of a fixed-point two's complement representation. The values are coded in w bits with nq bits after the point (fractional word length).

A fixed-point data format is expressed as $\langle w, nq\rangle$. The value of a w-bit binary number in this representation is $x \cdot 2^{-nq}$, where x is the integer equivalent of the two's complement w bit binary number. The MSB position with respect to the binary point is $ni = w - nq - 1$. The resolution of the fixed point number is 2^{-nq}, and the range is

$$\left[-2^{ni},\ 2^{ni} - 2^{-nq}\right]. \tag{1}$$

FPGAs are not constrained to work with a specific number of bits w to represent data. However, in fixed-point DSPs, w is kept constant because it is constrained by the DSP architecture.

Floating point numbers are represented using an exponent and a mantissa following the format shown in Figure 2, where s is the sign bit (0 for positive and 1 for negative numbers), e is the exponent, and f is the unsigned fractional part of the mantissa. The floating-point is "normalized" when it uses a hidden bit, so that the mantissa value is $1 + f$. The exponent is biased with bias $= 2^{ne-1} - 1$, where n_e is the number of bits in the exponent field. Then, the value of the normalized number is

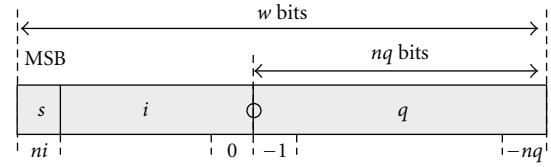

Figure 1: Fixed-point format.

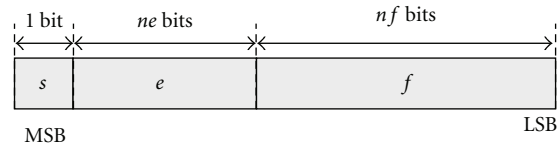

Figure 2: Floating-point format.

$$(-1)^s \cdot (1.f) \cdot 2^{e-\text{bias}}. \tag{2}$$

The range of normalized numbers is composed of the intervals $[-\max, -\min]$ and $[\min, \max]$ where

$$\begin{aligned} \min &= 2^{e_{\min}-\text{bias}} = 2^{1-\text{bias}}, \\ \max &= \left(2 - 2^{-nf}\right) \cdot 2^{e_{\max}-\text{bias}} = \left(2 - 2^{-nf}\right) \cdot 2^{\text{bias}}. \end{aligned} \tag{3}$$

A single precision floating point number in the standard IEEE-754 [29] is encoded in 32 bits with $ne = 8$, bias = 127, and $nf = 23$. A double precision floating point number in this standard is encoded in 64 bits with $ne = 11$, bias = 1023, and $nf = 52$. Again, FPGAs, unlike DSPs, are not constrained to work with these standards to represent data.

Certain values of e and f are reserved for representing special numbers, such as 0, NaNs (Not-a-Number) that indicates exceptions, $\pm\infty$ (Infinity), and denormalized numbers. These special values are as follows.

(i) If $e = 0$ and $f = 0$, it represents a 0.

(ii) If $e = 11\ldots1$ and $f \neq 0$, it represents an NaN.

(iii) If $e = 11\ldots1$ and $f = 0$, it represents $\pm\infty$ depending on sign.

(iv) If $e = 0$ and $f \neq 0$, it is a denormal number with value:

$$(-1)^s \cdot (0.f) \cdot 2^{-\text{bias}+1}. \tag{4}$$

3. Fixed and Floating-Point Number Representations in VHDL

The traditional approach to make arithmetic operations between vectors of *std_logic* elements is to use *numeric_std* or *std_logic_arith* packages. Both packages define the signed

vector type and arithmetic and comparison operations between objects of this type. From now on, we will use the first package.

In order to sum two operands in fixed-point format, the VHDL designer has to align the binary points, and take into account that the add operator produces a result vector whose length is equal to the length of the longest operand. The following code shows how the signal with the smallest number of fractional bits is left shifted to align the binary point. Besides, sign extension has to be explicitly described in VHDL to manage the carry.

```
signal A: signed(8 downto 0);
                -- <w=9,nq=8> format
signal B: signed(7 downto 0); --<8,5>
signal Y: signed(11 downto 0); --<12,8 >
--
Y <= A + (B(7)&B& "000");
```

The multiplication of two signed has the length of the sum of the lengths of the operands. Thus, the designer is in charge of managing the bit growth, and uses slicing and indexing of the result to extract the required part of the result. To perform rounding, before cutting off the bits below the specified LSB, LSB/2 is added first.

```
signal C: signed(7 downto 0); -- <8,5>
signal D: signed(7 downto 0); -- <8,5>
signal Y: signed(10 downto 0); --<11,7>
signal Y_tmp: signed(15 downto 0);
constant LSB2: signed(3 downto 0):=
                                "0100";
--
Y_tmp <= (C * D) + LSB2;
Y <= Y_tmp(13 downto 3);
```

In the above VHDL code, an overflow will occur if the multiplication result cannot be represented in the destination operand format <11,7>. This event must be explicitly managed by the designer using saturation arithmetic for instance.

Figure 3 shows a block diagram with the steps required to implement the addition of two floating-point numbers n_A and n_B [30]. It is assumed that the operands and the result n_R are normalized single precision floating point numbers. Basically, the exponents of the two numbers are compared and the mantissa corresponding to the smaller exponent is right shifted by the difference of the exponents so that the resulting two mantissas are aligned and can be added. The result is rounded to the appropriate number of bits and normalized. Using the *numeric_std* package, the designer has to describe in VHDL all these operations [11] in order to implement an addition.

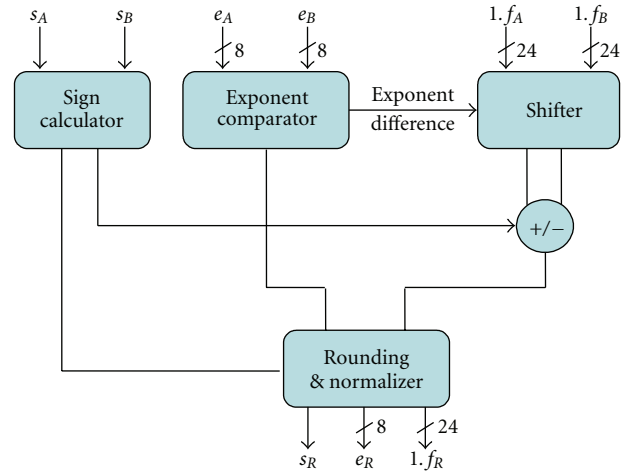

Figure 3: Floating point adder block diagram.

3.1. Fixed-Point Package. The VHDL *fixed_pkg* package supports the design of fixed-point digital systems from a higher abstraction level. This fully synthesizable package is part of the VHDL-2008 standard and is intended to work with fixed-point arithmetic. A compatibility version with VHDL-1993 is available to allow its use with no up-to-date tools.

This package defines two new types to represent unsigned and signed fixed-point numbers, respectively *ufixed* and *sfixed*. They are arrays of *std_logic* objects with integer range, and then a signal with format $\langle w, nq \rangle$ is declared as follows:

```
signal A: sfixed (ni downto -nq);
```

The package defines arithmetic, and comparison operations between objects of these types, following the propagation rules shown in Table 1 for addition and multiplication.

Then, the designer is relieved of aligning binary points and performing sign extension in sum operations.

```
signal A: sfixed(0 downto -8); -- <9,8>
signal B: sfixed(2 downto -5); -- <8,5>
signal Y: sfixed(3 downto -8); --<12,8>
--
Y <= A + B;
```

The package also defines a resize function to convert from one fixed-point type to another. This function allows specifying the quantization mode (truncation or rounding) and the overflow mode (wrap-around or saturation), and is very useful for managing the bit growth in multiplication operations.

```
signal C: sfixed(2 downto -5); -- <8,5>
signal D: sfixed(2 downto -5); -- <8,5 >
signal Y: sfixed(3 downto -7); --<11,7>
--
Y <= resize(C*D,Y,fixed_saturate,
                            fixed_round);
```

Table 1: *Fixed_pkg* propagation rules.

Operation	Result range
A ± B	MAX(A'left,B'left)+1 downto MIN(A'right,B'right)
A ∗ B	A'left+B'left+1 downto A'right+B'right

The arguments are the data to be resized, the size of the result, the overflow style, and the rounding method.

3.2. Floating-Point Package. The VHDL *float_pkg* package provides resources to design floating-point digital systems from a higher abstraction level. This fully synthesizable package is part of the VHDL-2008 standard; a compatibility version with VHDL-1993 is available to allow its use with no up-to-date tools.

The *float_pkg* package is VHDL-93 compatible and defines the type *float* to represent floating-point numbers as an array of *std_logic* objects with integer range:

```
signal A: float (ne downto -nf);
```

The top bit is the sign bit ($A(ne)$), the next bits are the exponent ($A(ne\text{-}1$ *downto* $0)$), and the negative bits are the fraction ($A(-1$ *downto-*$nf)$). Operators for all of the standard arithmetic and compare operations are defined in this package. Floating point result will have the maximum exponent and maximum mantissa of its input arguments. There is also a resize function to perform floating point conversions. Using this package, all the burden of exponent alignment and normalization described in Figure 3 is transparent to the designer as the following example shows.

```
signal A, B, Y: float(8 downto -23);
--
Y <= A + B;
```

For most designs, full IEEE support may be not necessary. The package allows to turn off denormal numbers (turn on by default), and turn off NaNs and overflow checks (turn on by default), select truncate (round to nearest by default), turn off guard bits (3 extra bits are used to maintain precision by default). This way, the hardware resource utilization can be reduced and the design clock frequency increased. However, this package does not allow pipelining the operations to increase speed.

4. Word Length Selection Procedure

The proposed methodology defines the necessary steps to obtain a synthesizable VHDL description of the digital controller from its transfer function $C(z)$ and architecture specification. The synthesized controller must fulfill the controller specifications. In the time domain, the specifications usually are the settling time, the percent overshoot, and the steady-state error, and in the frequency domain, the bandwidth and the gain crossover frequency.

It is assumed that the transfer function $C(z)$ of the digital controller can be written as

$$C(z) = \frac{\sum_{j=0}^{N} b_j \cdot z^{-j}}{1 + \sum_{j=1}^{M} a_j \cdot z^{-j}}, \tag{5}$$

where numerator b_j and denominator a_j coefficients are constant. The controller can be implemented using different structures or architectures. The more common architectures are: direct forms I/II, transposed forms, parallel/cascade of second order sections, and multicycle [31]. These architectures have equivalent input-output behavior but they have different internal signal range and quantization sensitivity. Then, the architecture is an input of this procedure.

This proposed word length selection procedure is a refinement process that selects the finite word length characteristics of each signal in the digital controller. For each coefficient and internal signal in the selected architecture, it is necessary to determine its finite word length format: *ni* and *nq* for fixed-point signals, and *ne* and *nf* for float-point signals, and for each operator or format change, the overflow and rounding strategies. To measure the finite word length effects, some performance parameters are chosen. These parameters are used to guide the word length selection procedure to meet the required controller specifications.

We use a hybrid approach inspired by [17, 22] that makes use of the VHDL-2008 libraries to perform bit-true simulations. The design flow is shown in Figure 4 and involves the following steps.

Step A [Reference Test Bench Generation]. A test bench to simulate in closed loop the plant, and the digital controller is generated to verify the functionality in representative working conditions. If the controlled plant is an analog system, then the complete system (plant + controller) is modeled using a mixed-signal simulation tool that can simulate any combination of Spice-like models and mixed signal hardware description languages such as VHDL-AMS. The electrical/electronic systems will be modeled in Spice, nonelectrical systems such as mechanical, thermal, optical, and chemical [32] will be modeled in VHDL-AMS, and the digital controller is modeled in VHDL.

The coefficients and internal signals are modeled in VHDL using real type objects. The VHDL predefined floating-point type real corresponds to the IEEE 64-bit double precision representation. This type can be simulated but it is not synthesizable. We will assume that signals of type real are represented with infinite precision, that is, do not have quantization errors. This will be the *reference model* for the next step.

Step B [Reference Model Simulation]. If the digital controller is implemented in fixed point, the model is simulated to obtain the maximum $s_i^{\max}$ values for each signal stored in registers. From these values and using (1), the correspondent parameters *ni* are obtained.

If the digital controller is implemented in floating point, the model is simulated to obtain the maximum $s_i^{\max}$ and minimum $s_i^{\min}$ values for each signal stored in registers, since

only normalized numbers will be used. From these values and using (3), the correspondent parameters *ne* are obtained taking into account that bias $= 2^{ne-1} - 1$.

Once these parameters are chosen, range propagation is performed for multipliers and adders output. Then, parameter *ni* or *ne* for each internal node is obtained to avoid overflow.

Step C [Synthesizable Digital Controller Design]. A fully synthesizable model of the controller is designed using the VHDL fixed-point or floating-point packages [15, 16].

Initially, the quantized coefficients $\hat{a}j$ and $\hat{b}j$ are obtained to meet the controller specifications in the frequency domain. If the controller includes an integrator in order to have zero steady-state error, the quantized coefficients of the denominator must fulfill the following condition:

$$\sum_{j=1}^{M} \hat{a}_j = -1. \tag{6}$$

If the digital controller is implemented in fixed point, coefficients are declared as constants of type *sfixed* with the parameters *ni* and *nq* necessary to meet the controller specifications. Internal nodes are declared as signals of type *sfixed* with the parameter *ni* obtained in Step B to avoid overflow. The fractional word lengths are initialized.

In the same way, if the digital controller is implemented in floating point, coefficients are declared as constants of type *float* with the parameters *ne* and *nf* needed to meet the controller specifications. Internal nodes are declared as signals of type *float* with the parameter *ne* obtained in step B to avoid overflow. The mantissa word lengths are initialized.

On an FPGA, the designer can use a different parameter (*nq* or *nf* depending on the numerical representation) for each internal signal in the controller. In order to reduce the design space, some signals with similar finite word length characteristics are grouped together, and the same tentative fractional or mantissa word length is assigned to each group.

Step D [Bit-True Simulation]. The testbench with the synthesizable digital controller is simulated and the results are compared with the ones of the reference model. The precision of the internal signals (parameter *nq* or *nf*) is increased until the performance parameter is achieved.

In order to set this word length in a systematic way, we apply two different procedures depending on the created number of groups. If only one group has been created, the procedure begins setting the precision of the internal signals to the output precision, and it is iteratively incremented and the system is resimulated until the specifications are met. If multiple groups have been created, the min *+1* minimization procedure presented in [22] is used to tune the word lengths.

5. Application Example

In this section, the design of a digital compensator is presented to control a buck converter using the proposed word length selection procedure. We present two implementations, one using fixed point arithmetic and the other with floating point arithmetic.

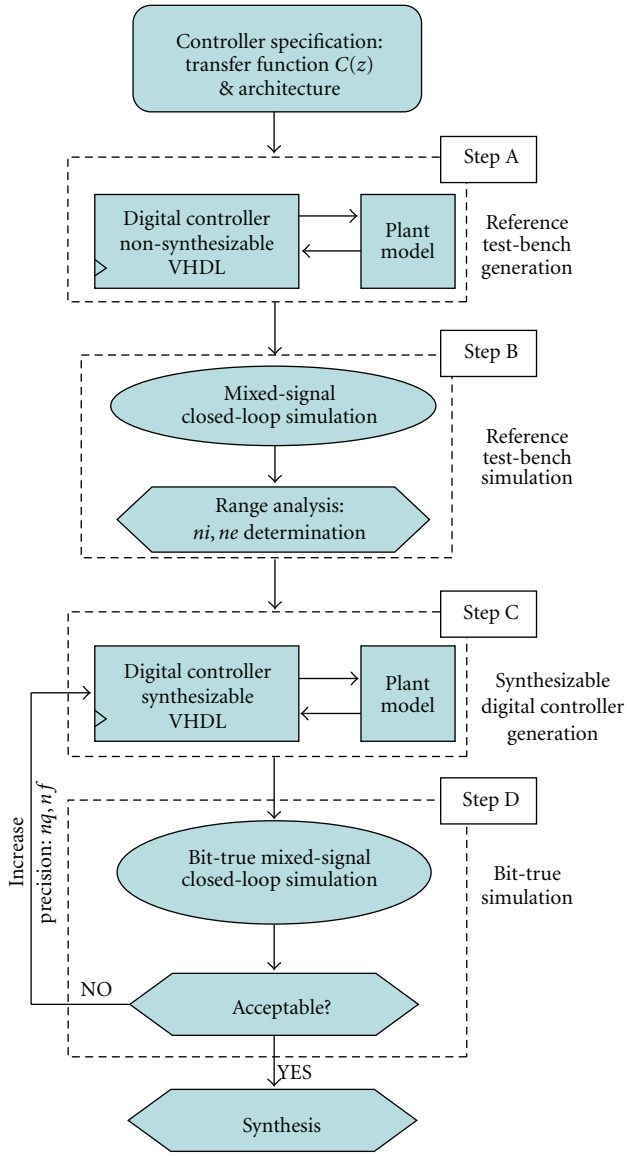

Figure 4: Design Flow.

Figure 5 shows the reference example. The power stage is a buck converter with the following parameters: input voltage $Vg = 12\,\text{V}$, $L = 47\,\mu\text{H}$, $C = 200\,\mu\text{F}$, capacitor equivalent series resistance $R_C = 25\,\text{m}\Omega$, inductor DC resistance $R_L = 30\,\text{m}\Omega$, load resistance $R_O = 2.7\,\Omega$, and switching frequency $f_{SW} = 100\,\text{kHz}$. The output voltage v_o has to be regulated at $V_{o\_\text{ref}} = 5\,\text{V}$.

The output voltage is measured using a resistor divisor with $R_S = 100\,\text{k}\Omega$ which scales the output voltage to the ADC input voltage range [0, 3.3 V]. The digital controller $C(z)$ determines the value of the power switch duty ratio. The digital pulse width modulator (DPWM) generates the driving signal *qc* that controls the switch *SW* of the converter. The ADC output ($v_f(k)$) resolution is 7 bits with format

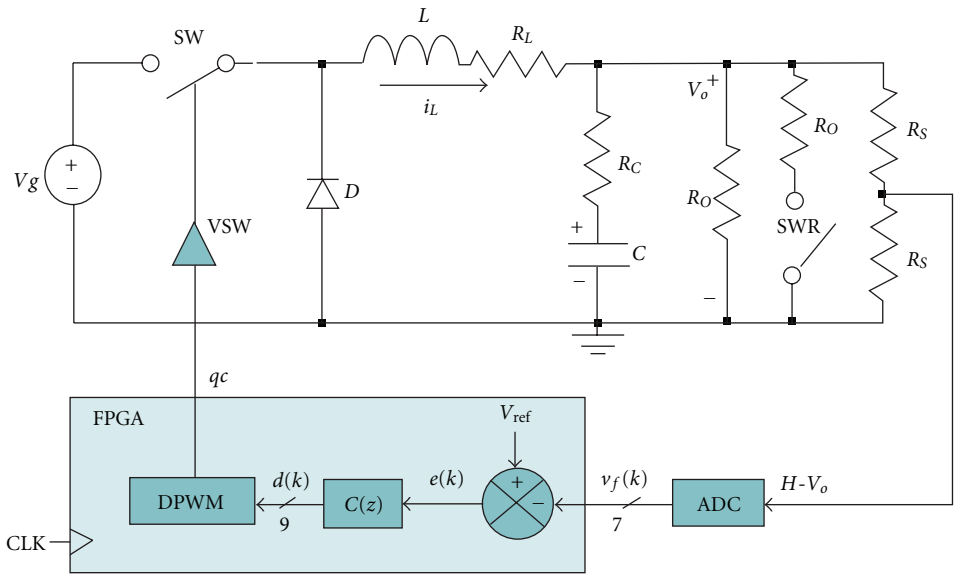

Figure 5: Reference example: Buck converter digitally controlled.

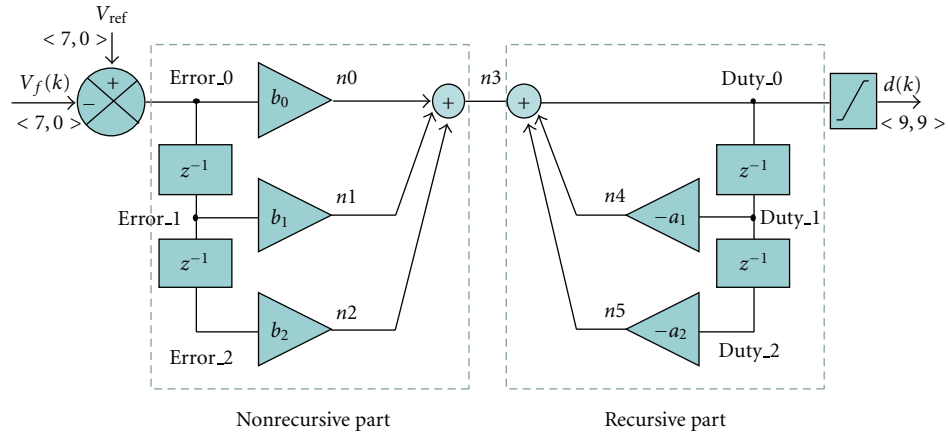

Figure 6: Direct form I implementation.

unsigned <7,0>, the FPGA clock frequency is 50 MHz, and the DPWM input ($d(k)$) resolution is 9 bits with format unsigned <9,9> and clamped between 0.1 and 0.9. Thus, assuming a sampling period of $T = 1/f_{SW}$, the digital controller $C(z)$ to implement is

$$C(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{0.1222 - 0.2236\, z^{-1} + 0.1019\, z^{-2}}{1 - 0.9\, z^{-1} - 0.1\, z^{-2}}. \quad (7)$$

The system bandwidth (BW) is 5 kHz with a phase margin (PM) of 61.4°. The compensator will be implemented in direct form I (Figure 6).

Step A [Reference Test-Bench Generation]. The whole system shown in Figure 5 has been simulated in closed loop using the mixed-signal simulation tool ADVance MS from Mentor Graphics. The buck converter has been modeled in Eldo. The conversion of the digital signal *qc* to the voltage *VSW* that controls switch *SW* is generated in VHDL-AMS. Switch *SWR* allows simulating step load changes. The ideal transfer characteristic and the serial interface of the ADC are modeled in VHDL. The compensator shown in Figure 6 is modeled in nonsynthesizable VHDL.

The stabilization time is taken as performance parameter, we define this parameter as the time between a change in the load and the instant when the error becomes 0 and the controller output $d(k)$ is stable.

Step B [Reference Test-Bench Simulation]. Figure 7 shows a transient response simulation of the closed loop from start-up. Once steady-state is reached after start-up, the load resistor is stepped from 2.7 Ω to 1.35 Ω at 12.8 ms. From top to bottom, the buck output voltage (*V_out*), the digital error (*error*), the 9-bits duty ratio command of the DPWM

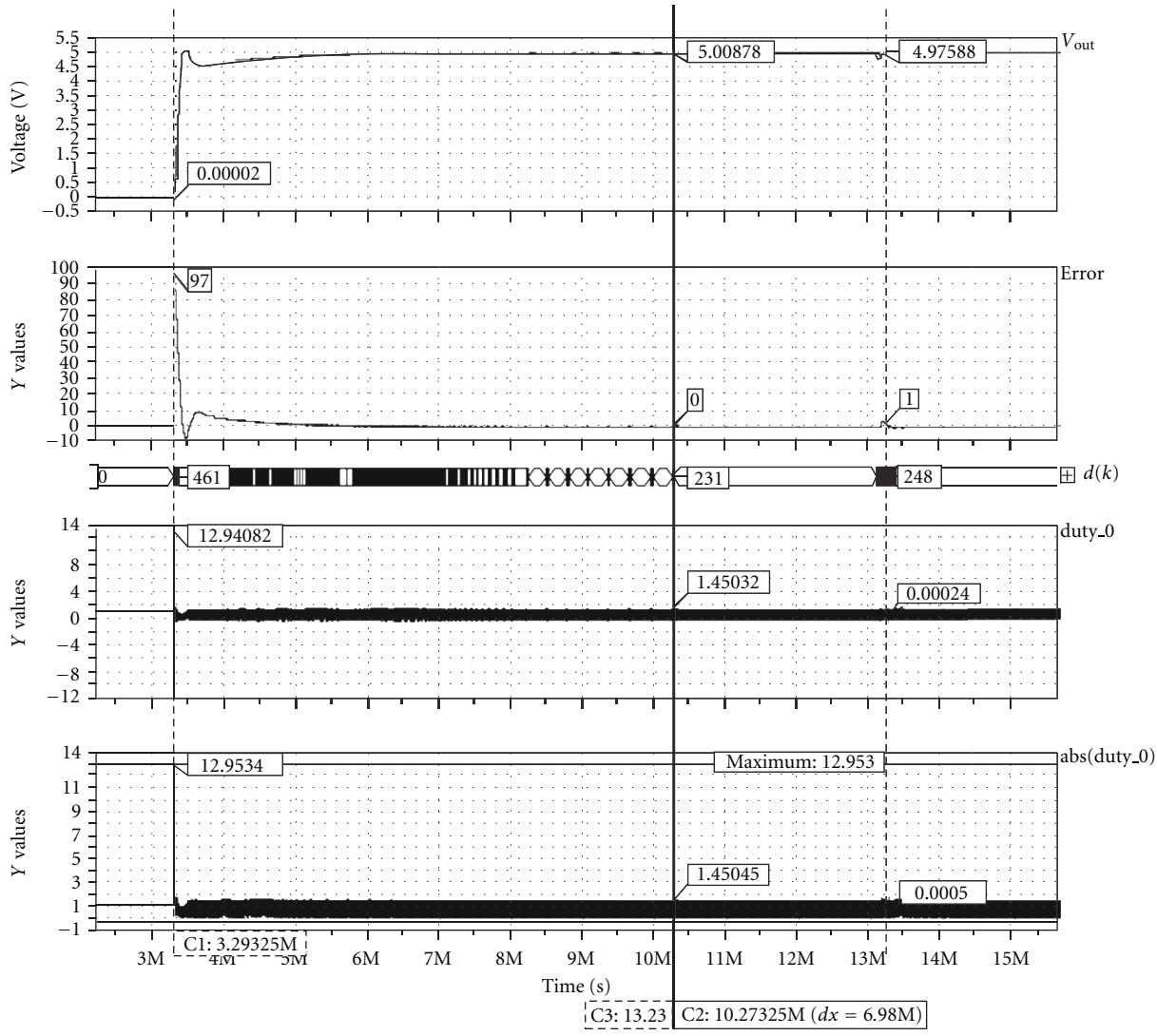

Figure 7: Reference model simulation results.

($d(n)$), the value of the digital signal *duty_0,* and *abs*(*duty_0*) are shown.

We have chosen the settling time as the performance parameter and evaluated it for the reference model. It can be seen in Figure 7 that there are not limit cycles and oscillations stop inside the zero error bin from start-up in 6.98 ms, and from the first load change in 1.7 ms. Then our design objective is to model a finite word length controller with approximately the same values for this parameter and without limit cycles.

From this simulation, we also obtain the range of the signals that are stored in flip-flops. Figure 6 shows the architecture of the implemented controller, in which, the delay units are modeled using flip-flops.

The controller input signal is obtained subtracting v_f to V_{ref}, since both operands have the format unsigned <7,0>, *error_0* has the format signed <8,0>. The nonrecursive part flip-flops have the same format and range as *error_0*, whereas the recursive part flip-flops have both the same format and range. From the reference model simulation, we can obtain the range of the signal $duty\_0$, in this case (Figure 7 bottom wave):

$$0.5 \times 10^{-3} \leq \text{abs}(duty\_0) < 13. \tag{1}$$

Step C [Synthesizable Digital Controller Design (Fixed-Point)]. In this implementation, each coefficient and node in the system is modeled using a custom fixed point format.

The compensator coefficients must be quantized. The quantization process is performed using high level design tools like Matlab, such that controller specifications are maintained in the frequency domain. Table 2 shows the quantized coefficients. With these coefficients, the new values for bandwidth and phase margin are BW = 5.1 kHz, PM = 62°.

Table 2: Quantized coefficients.

	coeff	*quantized*	*error*	*sfixed range*	*float range*
$b0$	0.1222	500×2^{-12}	−1.297e−4	(−3 downto −12)	(4 downto −8)
$b1$	−0.2236	-916×2^{-12}	−3,281e−5	(−2 downto −12)	(4 downto −9)
$b2$	0.1019	417×2^{-12}	9,336e−5	(−3 downto −12)	(4 downto −8)
$-a1$	0.9	29×2^{-5}	6,25e−3	(0 downto −5)	(3 downto −4)
$-a2$	0.1	3×2^{-5}	6,25e−3	(−3 downto −5)	(4 downto −1)

Table 3: Fixed-point node formats.

Node	sfixed range
*error*_0, *error*_1, *error*_2	(7 downto 0)
n3	(7 downto −12)
*duty*_0	(8 downto MIN(−12,−(fbits + 5)))
*duty*_1, *duty*_2	(4 downto-*fbits*)

The quantized coefficients (Table 2) are declared as constants in VHDL using the function *to_sfixed* from the *fixed_pkg*:

```
constant A2:sfixed(-3 downto -5):=
          to_sfixed(-0.09375,-3,-5);
...
```

The compensator in Figure 6 can be seen as two subsystems in cascade. The first is the nonrecursive part and has the structure of an FIR filter, the second is the recursive part. Then the signals in each block are grouped together.

In the nonrecursive part, the format of nodes *error*_1 and *error*_2 is the same as the input of the controller *error*_0, that is signed with format <8,0>. The formats of the nodes *n*0 to *n3* are obtained applying the propagation rules in Table 1.

In the recursive part, parameter *ni* for nodes *duty*_1 and *duty*_2 is obtained from the range analysis performed in previous step (1). This value is set to 4. Parameter *nq* for nodes *duy*_1 and *duty*_2 is the unknown *fbits*. The formats of the nodes *duty*_0, *n*4, and *n5* are obtained applying the propagation rules in Table 1. It is necessary to resize the signal *duty*_0 to obtain duty_1; it is performed by the *resize*( ) function using wrap-around and rounding to the nearest.

The results are summarized in Table 3.

Step D [Bit-True Simulations (Fixed-Point)]. In this step, we have to set the value of *fbits*, and we proceed as follows. Starting from a given minimum value of *fbits* = 9 (duty resolution), it is iteratively incremented and the system is resimulated until the specifications are met. Table 4 shows the values obtained for the start-up settling time parameter in this iterative process, we have used the start-up settling time because it is the most restrictive in the simulations. It is shown that quantization causes limit cycles (l.c.) in recursive structures if resolution is not enough. The chosen value is *fbits* = 13.

Step E [Synthesizable Digital Controller Design (Floating-Point)]. In this implementation, each node in the system

Table 4: fbits parameter versus start-up settling time.

fbits	9	10	11	12	**13**
Settlling time	l.c.	l.c.	l.c.	l.c.	**6.74 ms**

is modeled using a custom floating point format in which denormals, NaNs, and $\pm\infty$ are disabled. To avoid format changes that require additional hardware, we use only two different floating-point formats (subtypes) with the same *ne* to model the system, one to model nodes and coefficients of nonrecursive part *(err_type*), and other to model the nodes and coefficients *of recursive part* (*duty_type*). These subtypes are defined in VHDL as follows:

```
subtype err_type is float(eebits downto
                                -embits);
subtype duty_type is float(debits downto
                                -dmbits);
```

The quantized coefficients (Table 2) are declared as constants in VHDL using the function *to_float* from the *float_pkg*:

```
constant A2:duty_type:=
    to_float(-0.09375,debits,dmbits);
...
```

The compensator input signal (*error*_0) is obtained subtracting v_f to V_{ref}. These signals are *std_logic_vector* objects and the result of the subtraction must be converted to float type. The *float_pkg* provides the function *to_float* to perform this conversion.

The digital compensator in Figure 6 is modeled using the resources provided by *float_pkg* and the defined floating point subtypes. Table 5 shows the selected floating-point formats.

We set the floating-point format for subtype *err_type* from the known range and precision of the input and the quantized coefficients that allows to determinate the range and precision of node *n*3.

The range analysis performed in *Step B* is used to set the value of *debits* in *duty_type* from the range of signal *duty*_0 (1). The number of bits used to represent the mantissa in *duty_type* (*dmbits*) will be selected by simulation in the next step. Table 5 summarizes the used formats.

Step F [Bit-True Simulations (Floating-Point)]. In this step, we use simulations and an iterative process to set the value of

Table 5: Floating-point node formats.

node	subtype	float range
error_0, *error_1*, *error_2*	*err_type*	(5 downto − 18)
*n*1, *n*2, *n*3	*err_type*	(5 downto − 18)
*b*0, *b*1, *b*2	*err_type*	(5 downto − 18)
duty_0, *duty_1*, *duty_2*	*duty_type*	(5 downto −*dmbits*)
*n*4, *n*5	*duty_type*	(5 downto −*dmbits*)
*a*1, *a*2	*duty_type*	(5 downto −*dmbits*)

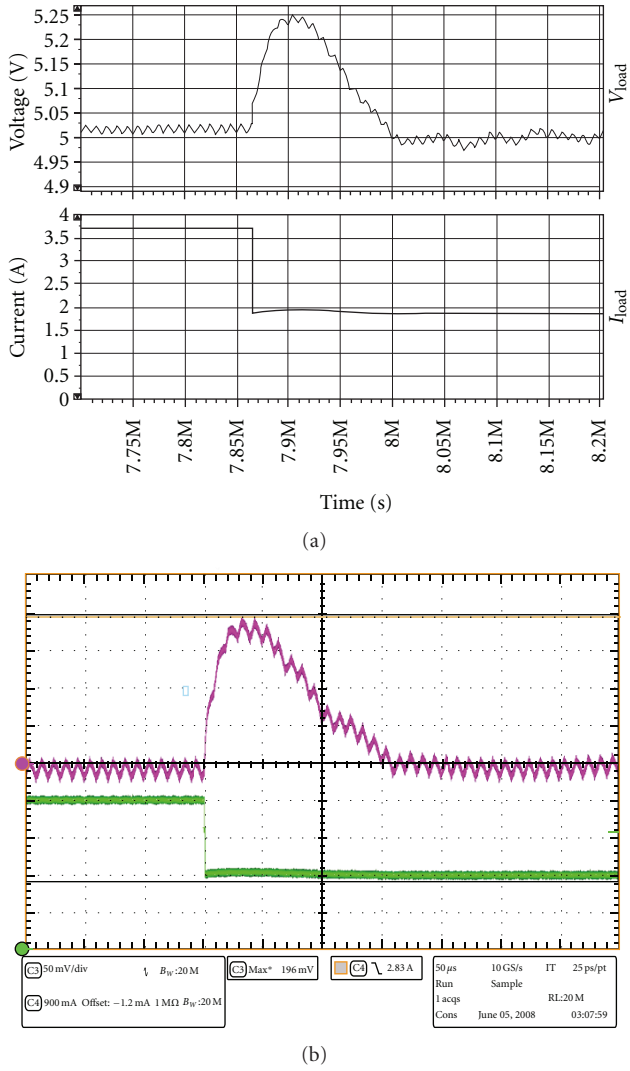

Figure 8: Load transient response. (a) Simulation (b) Measured, voltage: 50 mV/div, current: 0.9 A/div, time: 50 μs/div.

dmbits. Starting from a given minimum value of *dmbits* = 9 (duty resolution), it is iteratively incremented and simulated until the specifications are met in bit-true simulations. Table 6 shows the obtained start-up values obtained in this process. The selected value is *dmbits* = 14.

A conversion is needed to obtain the DPWM input from the compensator output. To simplify this function and following [33], the system has been readjusted to provide a duty ratio command between 1.1 and 1.9. Then this block confines *duty_0* to values between 1.1 and 1.9 saturating if it is needed. So that the 9 most significant bits of the fractional part of this clamped value is the DPWM input $d(k)$.

Table 6: *dmbits* parameter versus start-up settling time.

dmbits	9	10	11	12	13	**14**
Settling time	l.c.	l.c.	l.c.	l.c.	l.c.	**7.1 ms**

6. Experimental Results

A buck converter prototype has been designed for testing purposes [27]; the proposed controller has been implemented in a Xilinx FPGA XC3S400 Spartan-3 embedded in a Digilent Spartan-E System Board. The controller has been synthesized using Precision RTL Synthesis 2007a.18 from Mentor Graphics and implemented using Xilinx ISE 9.2.

The A/D converter is the ADCS7476 from National Semiconductor. It has 12 bits and can operate at 1 MSPS. As analyzed in [34], a necessary condition to avoid limit cycle oscillations is that the DPWM will be able to find an output voltage that lies within the ADC zero error bin. Thus, the resolution of the DPWM must be greater than the resolution of the ADC. In order to fulfill this condition, only the 7 most significant bits of the ADC are used.

We have used this prototype to verify the fixed and float point implementations. Figure 8 shows the simulated (a) and measured (b) response of the converter output voltage to a step-load change, where the load has been stepped from 1.35 Ω to 2.7 Ω. This simulation corresponds to the custom floating point format without denormals and NaNs; it can be seen that simulations closely match experimental results.

7. Conclusions

We present in this paper a word selection method valid for both fixed and floating point implementations of digital controllers on FPGAs. This method is based on mixed-signal bit-true simulations of the whole system. The test bench is modeled in VHDL-AMS and Spice. The controller is modeled in VHDL using the VHDL-2008 *fixed_pkg* and *float_pkg* packages that are included in the Draft Standard VHDL LRM P1076/D4.2. These packages allow customizing the word length of fixed and floating point representations which can result in important resource savings and computation speed

up. Besides, they simplify the design of arithmetic operations shortening the design cycle.

We have used this method in the design flow of a digital controller for a switching power converter that is implemented on an FPGA. Two implementations of the digital controller have been presented using fixed point and floating point arithmetic. Experimental results of the second implementation are presented. The results show that the designed system meets its specifications and the simulations match closely experimental results.

Acknowledgments

This research was supported in part by the Spanish MEC under Project TEC2007-64188 and Project CSD2009-00046, by DGA under Project PI008/08, and by Bosch and Siemens Home Appliances Group. The authors are with the Department of Electronic Engineering and Communications, and the Aragon Institute for Engineering Research (I3A), University of Zaragoza, 50018 Zaragoza, Spain.

References

[1] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of FPGAs," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[2] E. Monmasson and M. N. Cirstea, "FPGA design methodology for industrial control systems—a review," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1824–1842, 2007.

[3] C. Paiz and M. Porrmann, "The utilization of reconfigurable hardware to implement digital controllers: a review," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '07)*, pp. 2380–2385, June 2007.

[4] G. G. Parma and V. Dinavahi, "Real-time digital hardware simulation of power electronics and drives," *IEEE Transactions on Power Delivery*, vol. 22, no. 2, pp. 1235–1246, 2007.

[5] L. Corradini, P. Mattavelli, E. Tedeschi, and D. Trevisan, "High-bandwidth multisampled digitally controlled DC-DC converters using ripple compensation," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 4, pp. 1501–1508, 2008.

[6] Ó. López, J. Álvarez, J. Doval-Gandoy et al., "Comparison of the FPGA implementation of two multilevel space vector PWM algorithms," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 4, pp. 1537–1547, 2008.

[7] M. Zamora and M. P. Henry, "An FPGA implementation of a digital Coriolis mass flow metering drive system," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 7, pp. 2820–2831, 2008.

[8] Z. Shu, Y. Guo, and J. Lian, "Steady-state and dynamic study of active power filter with efficient FPGA-based control algorithm," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 4, pp. 1527–1536, 2008.

[9] Y. Lee, Y. Choi, M. Lee, and S.-B. Ko, "Performance analysis of bit-width reduced floating-point arithmetic units in FPGAs: a case study of neural network-based face detector," *EURASIP Journal on Applied Signal Processing*, vol. 2009, Article ID 258921, 11 pages, 2009.

[10] Z. Salcic, J. Cao, and S. K. Nguang, "A floating-point FPGA-based self-tuning regulator," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 2, pp. 693–704, 2006.

[11] H. Hu, T. Jin, X. Zhang, Z. Lu, and Z. Qian, "A floating-point coprocessor configured by a FPGA in a digital platform based on fixed-point DSP for power electronics," in *Proceedings of the Power Electronics and Motion Control Conference (IPEMC '06)*, vol. 2, pp. 1–5, August 2006.

[12] I. Urriza, L. A. Barragán, J. I. Artigas, D. Navarro, and O. Lucía, "FPGA implementation of a digital controller for a dc-dc converter using floating-point arithmetic," in *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*, pp. 2913–2918, 2009.

[13] T. Li and Y. Fujimoto, "Control system with high-speed and real-time communication links," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 4, pp. 1548–1557, 2008.

[14] T. Hill, "AccelDSP synthesis tool floating-point to fixed-point conversion of MATLAB algorithms targeting FPGAs," Xilinx WP239, 2006.

[15] D. Bishop, "Fixed point package user's guide," Packages and bodies for the IEEE 1076–2008 LRM, http://www.vhdl.org/fphdl/Fixed_ug.pdf.

[16] D. Bishop, "Floating point package user's guide," Packages and bodies for the IEEE 1076–2008 LRM, http://www.vhdl.org/fphdl/Float_ug.pdf.

[17] W. Sung and K. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Transactions on Signal Processing*, vol. 43, no. 12, pp. 3087–3090, 1995.

[18] D. Menard, D. Chillet, and O. Sentieys, "Floating-to-fixed-point conversion for digital signal processors," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 96421, 9 pages, 2006.

[19] D. Menard, R. Serizel, R. Rocher, and O. Sentieys, "Accuracy constraint determination in fixed-point system design," *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, Article ID 242584, 2008.

[20] M. Coors, H. Keding, O. Lüthje, and H. Meyr, "Design and DSP implementation of fixed-point systems," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 9, pp. 908–925, 2002.

[21] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement," in *Proceedings of the Design Automation and Test in Europe (DATE '99)*, pp. 271–276, 1999.

[22] M.-A. Cantin, Y. Savaria, D. Prodanos, and P. Lavoie, "An automatic word length determination method," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '01)*, pp. 53–56, May 2001.

[23] S. Roy and P. Banerjee, "An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design," *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 886–896, 2005.

[24] J. Lima, R. Menotti, J. M. P. Cardoso, and E. Marques, "A methodology to design FPGA-based PID controllers," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 2577–2583, October 2006.

[25] S. Jovanovic and P. Poure, "Design of power electronic digital controller based on FPGA/SOC using VHDL-AMS language," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '07)*, pp. 2301–2306, June 2007.

[26] J. Carletta, R. Veillette, F. Krach, and Z. Fang, "Determining appropriate precisions for signals in fixed-point IIR filters," in *Proceedings of the 40th Design Automation Conference*, pp. 656–661, June 2003.

[27] I. Urriza, L. A. Barragán, J. I. Artigas et al., "Word length selection method based on mixed simulation for digital PID controllers implemented in FPGA," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '08)*, pp. 1965–1970, July 2008.

[28] A. A. Gaffar, O. Mencer, W. Luk, and P. Y. K. Cheung, "Unifying bit-width optimisation for fixed-point and floating-point designs," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 79–88, April 2004.

[29] "IEEE Standard for Binary Floating-Point Arithmetic," ANSI/IEEE Std., IEEE 754–1985, 1985.

[30] J. Liang, R. Tessier, and O. Mencer, "Floating point unit generation and evaluation for FPGAs," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '03)*, pp. 185–194, 2003.

[31] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1999.

[32] F. Pêcheux, C. Lallement, and A. Vachoux, "VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–224, 2005.

[33] J. W. Gray, "PID routines for MC68HC11K4 and MC68HC11N4 Microcontroller," Freescale Semiconductor Application Note AN1215/D, 2004.

[34] A. Prodic, D. Maksimovic, and R. W. Erickson, "Design and implementation of a digital PWM controller for a high-frequency switching dc-dc power converter," in *Proceeding of the 27th Annual Conference of the IEEE Industrial Electronics Society (IECON '01)*, pp. 893–898, December 2001.