FPGA Design and Implementation of Digital PID Controller based on floating point arithmetic

Pourya Alinezhad ¹, Arash Ahmadi ²

¹- M.Sc. student of Electrical and communication engineering
Shahid Bahonar University, Iran
palinezhad@eng.uk.ac.ir

²- B.Sc. of Electrical and Electronics Engineering
Karaj Islamic Azad University, Iran
ahmadi.eng87@gmail.com

Abstract

This paper proposes a method for the design of digital PID controller on Field Programmable Gate Array with floating point arithmetic. Here we use Matlab HDL coder toolbox for implementing PID controller. Also an 8bit microprocessor called Picoblaze is programmed on FPGA chip to control peripherals like A/D and D/A converters, input output bottoms, character LCD etc. Required logical and mathematical operators like comparison and summation are overloaded to accept floating point numbers as input so calculations would have higher precision than software based PID controllers. Simulink model is used to plot the time response of PID controller. VHDL code generation and programming is performed in ISE9 software from Xilinx Company. We used Coregen for implementing floating point blocks. The proposed method was implemented practically on Xilinx Spartan-3E FPGA Board.

Key words: FPGA, Picoblaze, PID controller, HDL, Simulink

1. Introduction

A control system consists of two subsystems, a plant and a controller. The plant is an entity controlled by the controller [1]. The controller can be either analog or digital. Generally, an implementation of digital PID controller includes the use of microprocessors or microcontrollers. The memory holds the application program while the processor fetches, decodes, and executes the program instructions. One of disadvantages of this method is the speed of operations because the operations depend on software which has a sequence of instructions and commands which needs many machine cycles to execute.

Therefore, FPGA-based digital PID controller is proposed because the operations on FPGA are hardware compatible operations. Better speed performance could be achieved because all operations are done in parallel. Proposed FPGA-based digital PID controller uses floating point math operands for computations. Here we overloaded these computations for floating point arithmetic to achieve better accuracy.

Modern FPGAs and their distinguishable capabilities have been advertised extensively by FPGA vendors. Moreover, some refereed articles addressed the advantages of utilizing these powerful chips [2]. Recently, Spartan II and III FPGA families from Xilinx have been successfully utilized in a variety of applications which include inverters, communications, imbedded processors, and image processing.
Here we use Spartan-3E toolkit from Xilinx Company to implement our designed PID controller.

So many works have been down to implement conventional PID controllers but a few studies were made for implementing PID controllers on FPGA’s. Gupta and Khare [1] proposed a method to design the PID controller in Simulink but no practical works were made. XU and SHUANG [3] designed a Fixed-point Digital PID Controller and showed that high reliability and fast response speed with wide dynamic range could be achieved. Sonoli and Nagabhushan [4] used FPGA based PID Controller for controlling DC Motor system. Behnam and Mansouryar [5] designed a digital PID Controller and Encoder Using Xilinx System Generator library in Simulink. Xilinx System Generator is a Simulink library block set provided by Xilinx Company to generate FPGA compatible models.

In [6] a PID controller designed to achieve a balance between the speed and the exhausted FPGA resources. [7] Presents a novel technique for implementation of an efficient FPGA based digital PID controller for the motion control of a permanent magnet DC motor. In [8] Patel & singh designed FPGA-based all Digital PID Controller on Spartan3e XC3S100E FPGA.

In this paper, we develop a new PID controller model based on floating point arithmetic. In section 2 discrete PID model is reviewed and a Simulink model is designed and simulations are performed to verify model. Also some practical issues like Untiwindup implementation are considered. Section 3 illustrates floating point numbers and HDL code generation. Section 4 shows practical results of designed PID controller and also describes some practically used tools. Finally, section 5 concludes the paper.

2. Discrete PID model

At first we need a complete model of the digital controller. We start our work by continues model and then the discrete model could be achieved with some manipulations. Consider a PID and the Planet model as shown in Fig 1.

![Fig 1. A typical Planet model](image)

As we know PID controller consists of proportional, derivative and integral coefficients. The controller could be shown in time domain by Eq.1.

\[
u(t) = k_p [e(t) + \frac{1}{T_i} \int_0^t e(\tau)d\tau + \frac{T_d}{T_i} \frac{de(t)}{dt}]
\]

\[k_p\] is the proportional coefficient and \(e(t)\) shows controller input signal. \(T_i\) and \(T_d\) are integral and derivative coefficients respectively. In Laplace domain we can write Eq.1 as follows:

\[
H(s) = K_p [1 + \frac{1}{s \times T_i} + s \times T_d]
\]

\[2\]
Now in order to have discrete form we must transfer Eq.2 to z domain. We can
determine a digital implementation of this controller by using a discrete
approximation for the derivative and integration [9]. Impulse invariance or bilinear
methods [10] could be used to achieve discrete form. Here we use bilinear
transform. Bilinear transform is made by Taylor series expansion of function and
using time shift property of signal with is equal to a delay in z domain (Eq.3).

\[ Z\{x[k - n]\} = z^{-n}X[z], \text{ where} \]
\[ Z\{x[k]\} = X[z] \]

\( Z \) is transform operator and \( n \) illustrates time shifting. Bilinear transformation
equation is performed as follows:

\[ y(k) = y(k - 1) + \frac{f(k) + f(k - 1)}{2} T_s \] (4)

\[ Y[z](1 - z^{-1}) = \frac{T_s}{2} F[z](1 + z^{-1}) \] (5)

\[ \frac{Y[z]}{F[z]} = \frac{T_s z + 1}{2 z - 1} \] (6)

Eq.4 shows Taylor expansion and Eq.5 shows transformed series to Z domain.
Finally, Eq.6 is rearranged form of Eq.5 and shows the bilinear formula. \( T_s \) is
sampling Time.

Applying bilinear transform on PID controller by Eq.2 and Eq.6 results:

\[ \frac{U[z]}{E[z]} = K_p + K_i \frac{T_s}{2} \left( \frac{z + 1}{z - 1} \right) + K_d \frac{z - 1}{T_s} \] (7)

\[ \frac{U[z]}{E[z]} = \frac{K_p (z^2 - z) + K_i \frac{T_s}{2} (z^2 + z) + K_d \frac{T_s}{2} (z^2 - 2z + 1)}{z^2 - z} \] (8)

\[ \frac{U[z]}{E[z]} = \frac{\left( K_p + K_i \frac{T_s}{2} + K_d \frac{T_s}{T_s} \right) z^2 + \left( -K_p + K_i \frac{T_s}{2} - 2K_d \frac{T_s}{T_s} \right) z + K_d \frac{T_s}{T_s}}{z^2 - z} \] (9)

After multiplying numerator and denominator of Eq.8 by \( z^{-2} \) we have:

\[ \frac{U[z]}{E[z]} = \frac{\left( K_p + K_i \frac{T_s}{2} + K_d \frac{T_s}{T_s} \right) z^{-1} + \left( -K_p + K_i \frac{T_s}{2} - 2K_d \frac{T_s}{T_s} \right) z^{-2}}{1 - z^{-1}} \] (10)

And after some straight forward simplifications we reach the following formula:

\[ U[z] = z^{-1}U[z] + a E[z] + b z^{-1}E[z] + c z^{-2}E[z] \] (11)

\[ U[k] = u[k - 1] + a e[k] + b e[k - 1] + c e[k - 2] \] (12)

Eq.12 shows the time domain discrete PID controller.
Now we progress our work by simulating the discrete PID controller base on Eq.11. Fig.2 shows the designed PID controller in Matlab’s Simulink environment. The coefficients are tuned practically.

This model still has some problems. We have to use an Untiwindup to prevent integral term from saturation. There are many ways to protect against windup where the integrator part of PID controller saturates due to accumulation nature of integration. Tracking is a simple method which is illustrated in block diagram in Fig.3.a.

Fig.2. Disceret PID controller model

Fig.3.b shows the Simulink model of Untiwindup. In Fig.3.a the system has an extra feedback path around the integrator. The signal $e_s$ is the difference between the nominal controller output $v$ and the saturated control output $u$. The signal $e_s$ is fed to the input of integrator through gain $1/T_c$. The signal $e_s$ is zero when there is no saturation. Under these circumstances it will not have any effect on the integrator. When the actuator saturates, the signal $e_s$ is different from zero and it will try to drive the integrator output to a value such that the signal $v$ is close to the saturation limit.

Fig.3.a. Untiwindup presentation

Fig.3.b. Untiwindup simulation model

The designed PID controller is simulated in a planet with $H(s) = \frac{2}{s^2 + 4s + 100}$ as system’s transfer function in a closed loop and a periodic pulse with $f = 1/20Hz$ is applied to input. Moreover predefined Simulink PID model is included for comparing results. Fig.4 shows the input signal, designed discrete PID controller without Untiwindup, designed discrete PID controller with Untiwindup and predefined Simulink Model...
respective from top to bottom. It’s obvious that the PID without Untiwindup couldn’t answer to input step pulse applied in 10sec.

So far we have defined the essential concepts for discrete PID controller. In next section a brief review on floating point representation is explained.

3. Floating point blocks

Floating-point numbers are the favorites of software people, and the least favorite of hardware people. The reason for this is because floating point takes up almost 3X the hardware of fixed-point math. The advantage of floating point is that precision is always maintained with a wide dynamic range, where fixed point numbers lose precision.

Floating-point numbers are well defined by IEEE-754 (32 and 64 bit) and IEEE-854 (variable width) specifications. Floating point has been used in processors and IP for years and is a well-understood format. This is a sign magnitude system, where the sign is processed differently from the magnitude.

There are many concepts in floating point that make it different from our common signed and unsigned number notations. These come from the definition of a floating-point number. Below is a 32-bit floating-point number format:

\[
S \ EEEEEEEE \ FFFFFFFFFFFFFFFFFFFFFF
\]

\[
31 \ 30 \ 25 \ 24 \ 0
\]/
exp . Fraction

Basically, a floating-point number comprises a sign bit (+ or –), a normalized exponent, and a fraction. To convert this number back into an integer, the following equation can be used:

\[
S \times 2^{(\text{exponent} - \text{exponent\_base})} \times (1.0 + \text{Fraction/\_fraction\_base})
\]

(14)

Where the “exponent\_base” is \(2^{((\text{maximum\_exponent}/2)-1)}\), and “Fraction\_base” is the maximum possible fraction (unsigned) plus one. Thus, for a 32-bit floating-point an example would be:

\[
0 \ 10000001 \ 10100000000000000000000000000000
\] = +1 \times 2^{(129 - 127)} \times (1.0 + 10485760/16777216) = +1 * 4.0 * 1.625 = 6.5

(15)
There are also “denormal numbers”, which are numbers smaller than can be represented with this structure. The tag for a denormal number is that the exponent is “0”. This forces you to invoke another formula where you drop the “1.0 +”:

\[
1 \ 00000000 \ 100000000000000000000000 \\
= -1 * 2** -126 * (8388608/16777216) = -1 * 2**-126 * 0.5 = -2**-127
\]  \hspace{1cm} (16)

Next, the “constants” that exist in the floating-point context are:

\[
\begin{align*}
0 \ 00000000 \ 000000000000000000000000 &= 0 \\
1 \ 00000000 \ 000000000000000000000000 &= -0 \text{ (which = 0)} \\
0 \ 11111111 \ 000000000000000000000000 &= \text{positive infinity} \\
1 \ 11111111 \ 000000000000000000000000 &= \text{negative infinity}
\end{align*}
\]  \hspace{1cm} (17)

In this project we define and use a smaller floating point with just 11 fraction bits and 7 exponent bits. So counting the sign bit, our floating point number has 20 bits. Floating point blocks are implemented using “ISE9 Corgen” which easily produces embedded codes based on user defined adjustments.

4. Results

HDL code of the designed PID Model in section 2 is generated using Matlab HDL Coder Toolbox. On the other hand floating point arithmetic operands are programmed using Corgen. Merging these two codes results an overloaded PID controller with floating point operations. The whole code is simulated in ISE9 environment and the results are shown in Fig.4.a, Fig.4.b and Fig.5.

Fig.4.a. Full PID technological schematic

Fig.4.b. Part of the technological schematic (Zoomed)
Device utilization summary could be seen in Fig.6. The target device is xc3s500e FPGA from Xilinx Family.

Now as could be seen in Fig.4 we implemented a full digital PID controller which has 9 pins. `pid_z` and `pid_u` are input and output ports respectively. `pid_kp`, `pid_ki` and `pid_kd` are proportional, derivative and integral coefficients respectively. These coefficients are adjusted by available on board devices like pushbuttons and their values could be seen on an LCD. **Linear Tech LTC2624 Quad DAC** and **LTC1407A-1 Dual A/D** are used to convert input analog signal to digital and vice versa. For additional information about the peripherals on Spartan-3E board see [11]. All UCF location constraints for FPGA pin assignment are adopted from embedded Kit.

At last an 8 bit microprocessor called Picoblaze [12] is programmed on FPGA and hundreds line of embedded assembly code for this microprocessor was written and assembled using “KCPSM3” Assembler in order to control peripheral devices like A/D and D/A converter, pushbuttons, LCD etc. The Picoblaze and Assembler source codes are available at Xilinx official website for free. Simulated signals of these peripherals are shown in Fig.7.
In this paper we designed and implemented a digital PID controller with floating point arithmetic. The advantage of the proposed method is its capability to do high precision arithmetic. The designed controller also benefits from other intrinsic performances of FPGA’s like its high speed and low power. Obviously these floating point operands could be used to implement other digital filters. Matlab HDL coder toolbox and Picoblaze played a main role in PID implementation on xc3s500e FPGA. Future researches could be down to use the designed PID to control a system like servomotor. Another topic would be implementing floating point with higher precision (32 or 64 bit).

Acknowledgements
The authors gratefully acknowledge the support and encouragements of Sir AmirReza Siminfar without whom this project couldn’t be completed.

References