Implementation of FPGA Based Incremental PID Controller Using Conventional Method and Distributed Arithmetic Algorithm

Mariamma John[1], *B. Thilagavathi*[2], *Anitha Mary. X*[3]

Abstract

In this paper, two efficient design schemes for implementation of the Incremental Proportional-Integral-Derivative (PID) controller using Field Programmable Gate Array (FPGA) technology are presented. Conventional implementation and also Distributed arithmetic based implementation of Incremental PID controller is done. Conventional implementation contains a large number of multipliers and adders. It's not focused on optimal use of hard ware resources and do not efficiently utilize the memory-rich characteristics of FPGAs. An FPGA chip consists of a lot of memory blocks, referred to as Look-Up Tables (LUT), which can be utilized to implement efficient designs. In the Distributed Arithmetic (DA) scheme is an efficient LUT design method, and is very promising in the FPGA implementation of PID controller. Distributed arithmetic replaces multiplication by addition and shifting. The values are precomputed and placed in LUT. Based on the LUT scheme, the proposed PID controller reduces the cost of the FPGA design by enabling the chip to accommodate more logic and arithmetic functions while requiring less power consumption.

Keywords : PID Controller, Incremental Form, Conventional Method, Distributed Arithmetic, FPGA.

1. Introduction

Proportional Integral Derivative (PID) controller is the most common type of controller used in dynamic systems. An important feature of this controller is that it does not need a precise analytical model of the system that is being controlled. For this reason, PID controllers have been widely used in process control, manufacturing, robotics, automation, transportation, and interestingly in real-time scheduling of concurrent tasks in multi-tasking applications.

PID controllers are often combined with logic, sequential functions, and other blocks to implement complicated systems. Implementation of PID controllers has gone through several stages of evolution, from the early mechanical and pneumatic designs to the microprocessor-based systems. Recently, Field Programmable Gate Arrays (FPGA) have become an alternative solution for the realization of digital control systems, previously dominated by the general purpose microprocessor and application specific integrated circuits (ASIC).The FPGA-based controllers offer advantages such as high-speed computation, complex functionality, real – time processing capabilities, and low power consumption .These are attractive features from the embedded systems design point of view.

[1]PG Scholar, Embedded Systems, Dept. of Electronics and Instrumentation Engineering, Karunya University, Coimbatore. Email: mariammajohn@karunya.edu.in

[2]Lecturer, Dept. of Electronics and Instrumentation Engineering, Karunya University, Coimbatore. Email:deepanasri@rediffmail.com

[3]Assistant Professor, Dept. of Electronics and Instrumentation Engineering, Karunya University, Coimbatore. Email:anithajohnson2003@gmail.com

Conventional implementation of FPGA based controllers have not focused on optimal use of hardware resources. These designs usually require a large number of multipliers and adders and do not efficiently utilize the memory-rich characteristics of FPGAs. An FPGA chip consists of a lot of memory blocks, referred to as Look-Up Tables (LUT), which can be utilized to implement efficient designs. In this work, we utilize the Distributed Arithmetic (DA) scheme , which is an efficient LUT design method, and is very promising in the FPGA implementation of PID controller. Incremental form of PID controller is obtained from the basic controller equation.

The organization of this paper is as follows. In section 2, PID controller is discussed. In Section 3, incremental form of PID is discussed. Section 4 throw some light on multiplier based conventional method. Section 5 and 6 tells about DA and its fusion on incremental equation. Simulation results are discussed in section 7.

2. PID CONTROLLER

The PID control algorithm is one of the most commonly used control algorithms in industry. A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems. A PID controller attempts to correct the error between a measured process variable and a desired set point by calculating and then outputting a corrective action that can adjust the process accordingly and rapidly, to keep the error minimal as shown in Fig1.

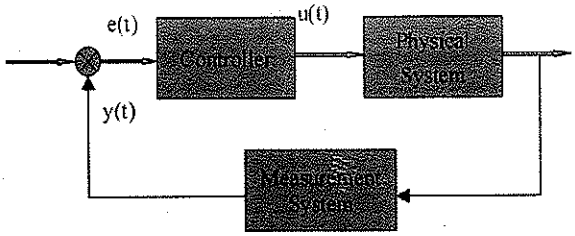

Figure 1 : Block Diagram of a Control System

In Fig.1 variable (e(t)) represents the tracking error, the difference between the desired input value (set point) and the actual output (output). This error signal (e(t)) will be sent to the PID controller, and the controller computes both the derivative and the integral of this error signal. The signal (u(t)) just past the controller is now equal to the proportional gain (k_p) times the magnitude of the error plus the integral gain (k_i) times the integral of the error plus the derivative gain (k_d) times the derivative of the error. T_i is the reset time and T_d is the derivative time.

$$u(t) = k_p \left[e(t) + \frac{1}{T_i}\int_0^T e(t)dt + T_d \frac{de(t)}{dt} \right] \qquad (1)$$

The PID controller calculation (algorithm) involves three separate parameters; the proportional, the integral and derivative values. The *proportional* value determines the reaction to the current error, the *integral* value determines the reaction based on the sum of recent errors, and the *derivative* value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element.

3. INCREMENTAL FORM

Analog Control refers to the design and implementation of controllers in the continuous domain. There are a number of ways by which this common and versatile controller can be implemented in discretised form. The two forms used are incremental (velocity) form and positional (full valve) form. The equation (1) is discretised and the following equation is obtained :

$$u[n] = k_p\, e[n] + k_i \sum_{j=0}^{n} e[j] + k_d \left[e[n] - e[n-1] \right] \qquad (2)$$

Where $k_i = k_p \dfrac{T}{T_i}$ is the integral coefficient and $k_d = k_p \dfrac{T_d}{T}$ is the derivative coefficient. This form is called

the position form of the PID algorithm. An alternative would be to compute u[n] based on past output u[n-1]and correction term Δu(n). This approach is often called as the velocity form of the PID algorithm. The first step in this regard would be to calculate u[n-1] based on equation(2).

$$u[n-1] = k_p\, e[n-1] + k_i \sum_{j=0}^{n-1} e[j] + k_d\, [e[n-1] - e[n-2]] \quad (3)$$

Then calculate correction term as :

$$\Delta u[n] = u[n] - u[n-1]$$
$$= e[n] + k_1\, e[n-1] + k_2\, e[n-2] \quad (4)$$

Where

$$k_0 = k_p + k_i + kd$$
$$k_1 = -k_p - 2k_d$$
$$k_2 = k_d$$

The current control output is calculated as:

$$u[n] = u[n-1] + \Delta u[n] \quad (5)$$

$$= u[n-1] + k_0 e[n] + k_1\, e[n-1] + k_2\, e[n-2] \quad (6)$$

The above equation (6) represents the incremental form of PID controller.

4. Conventional Method

Incremental form of PID is decomposed into its basic operations. Here p and p_d refers to the controlled variable and its desired value (setpoint) respectively. p0,p2,p2,s1,s2 are temporary variables.

e[n] = P + (- Pd)

P0 = k_0 * e[n]

P1 = k_1 * e[n-1]

P2 = k_2 * e[n-2]

S1 = P0 + P1

S2 = P2 + u[n-1]

u[n] = S1 + S2

Input represents the current position P. The negation of P, P neg, is generated by bit-wise complementing and adding 1. At the rising edge of control, signal e[n] of the last cycle is latched at register REG1, thus becomes e[n–1] of this cycle. In the same manner, e[n–2] and u[n – 1] are recorded at REG3 and REG4 by latching e[n – 1] and u[n] respectively. The registers can be set to initial values of 0 by asserting the reset signal, reset. As long as the desired position Pd is also initialized to 0 when the system is reset, the control output is 0, which can keep the controlled device (i.e. the motor, in this system) static. The computed control output u[n] may exceed the range that the controlled device can bear. Bounder is a value limitation logic that keeps the output in the user defined range of UpBound and LowBound.

The VHDL coding for conventional method were written and simulation result was obtained.

5. Distributed Arithmetic

Distributed Arithmetic (DA), along with Modulo Arithmetic, are computation algorithms that perform multiplication with look-up table based schemes. Both stirred some interest over two decades ago but have languished ever since. Indeed, DA specifically targets the sum of products (sometimes referred to as the vector dot product) computation that covers many of the important DSP filtering and frequency transforming functions. Ironically, many DSP designers have never heard of this algorithm. Inspired by the potential of the Xilinx FPGA look-up table architecture, the DA algorithm was resurrected in the early 90's and shown to produce very efficient filter designs.

The derivation of the DA algorithm is extremely simple but its applications are extremely broad. The mathematics includes a mix of Boolean and ordinary algebra and requires no prior preparation - even for the logic designer.

The arithmetic sum of products that defines the response of linear, time-invariant networks can be expressed as:

$$y(n) = \sum_{k=1}^{K} A_k x_k(n) \qquad (7)$$

where :

y(n) = response of network at time n.

$x_k(n)$ = kth input variable at time n.

A_k= weighting factor of kth input variable that is constant for all n, and so it remains time-invariant.

In filtering applications the constants, A_k, are the filter coefficients and the variables, x_k , are the prior samples of a single data source (for example, an analog to digital converter). In frequency transforming whether the discrete Fourier or the fast Fourier transform - the constants are the sine/cosine basis functions and the variables are a block of samples from a single data source. Examples of multiple data sources maybe found in image processing. The multiply-intensive nature of equ 7 can be appreciated by observing that a single output response requires the accumulation of K product terms. In DA the task of summing product terms is replaced by table look-up procedures that are easily implemented in the Xilinx configurable logic block (CLB) look-up table architecture.

We start by defining the number format of the variable to be 2's complement, fractional - a standard practice for fixed-point microprocessors in order to bound number growth under multiplication. The constant factors, A_k, need not be so restricted, nor are they required to match the data word length, as is the case for the microprocessor. The constants may have a mixed integer and fractional format; they may be written in the fractional format as shown in equ. 8

$$x_k = -x_{k0} + \sum_{b=1}^{B-1} x_{kb} 2^{-b}$$

where x_{kb} is a binary variable and can assume only values of 0 and 1. A sign bit of value -1 is indicated by x_{k0}. Note that the time index, n, has been dropped since it is not needed to continue the derivation. The final result is obtained by first substituting equ.8 into equ.7

$$y = \sum_{k=1}^{K} A_k \left[-x_{k0} + \sum_{b=1}^{B-1} x_{kb} 2^{-b} \right] = \sum_{k=1}^{K} x_{k0} * A_k + \sum_{k=1}^{K} \sum_{b=1}^{B-1} x_{kb} * A_k 2^{-b}$$

and then explicitly expressing all the product terms under the summation symbols equation 9:

$$\begin{aligned} y = &-[x_{10} * A_1 + x_{20} * A_2 + x_{30} * A_3 + \ldots + x_{K0} * A_K] \\ &+ [x_{11} * A_1 + x_{21} * A_2 + x_{31} * A_3 + \ldots + x_{K1} * A_K] 2^{-1} \\ &+ [x_{12} * A_1 + x_{22} * A_2 + x_{32} * A_3 + \ldots + x_{K2} * A_K] 2^{-2} \\ &\vdots \\ &+ [x_{1(B-2)} * A_1 + x_{2(B-2)} * A_2 + x_{3(B-2)} * A_3 + \ldots + x_{K(B-2)} * A_K] 2^{-(B-2)} \\ &+ [x_{1(B-1)} * A_1 + x_{2(B-1)} * A_2 + x_{3(B-1)} * A_3 + \ldots + x_{K(B-1)} * A_K] 2^{-(B-1)} \end{aligned}$$

Each term within the brackets denotes a binary AND operation involving a bit of the input variable and all the bits of the constant. The plus signs denote arithmetic sum operations. The exponential factors denote the scaled contributions of the bracketed pairs to the total sum. You can now construct a look-up table that can be addressed by the same scaled bit of all the input variables and can access the sum of the terms within each pair of brackets.

The arithmetic operations have now been reduced to addition, subtraction, and binary scaling. With scaling by negative powers of 2, the actual implementation entails the shifting of binary coded data words toward the least significant bit and the use of sign extension bits to maintain the sign at its normal bit position. The hardware implementation of a binary full adder (as is done in the CLBs) entails two operands, the addend and the augend to produce sum and carry output bits. The ultimate in

gate efficiency would be a single DALUT, a single parallel adder, and, of course, fewer flip-flops for the input data source. Again with our B=16 examples, a rephrasing of equation 9 yields the desired result :

$$y = \{\cdots 14 \cdots \{[sum15]2^{-1} + [sum14]\}2^{-1} + [sum13]\}2^{-1} + [sum12]\}2^{-1} + [sum11]\}2^{-1} + [sum10]\}2^{-1} + [sum9]\}2^{-1} + [sum8]\}2^{-1} + [sum7]\}2^{-1} + [sum6]\}2^{-1} + [sum5]\}2^{-1} + [sum4]\}2^{-1} + [sum3]\}2^{-1} + [sum2]\}2^{-1} + [sum1]\}2^{-1} - [sum0]$$

Starting from the least significant end, i.e. addressing the DALUT with the least significant bit of all K-1 and then added to the DALUT input variables the DALUT contents, [sum15], are stored, scaled by 2^{-1} contents, [sum14] when the address changes to the next-to-the-least-significant bits. The process repeats until the most significant bit addresses the DALUT, [sum0]. If this is a sign bit a subtraction occurs. Now a vision of the hardware emerges. A serial shift register, B bits long, for each of the K variables addresses the DALUT least significant bit first. At each shift the output is applied to a parallel adder whose output is stored in an accumulator register. The accumulator output scaled by 2^{-1} henceforth, the adder, register and scalar shall be referred to as a scaling accumulator. The functional blocks are shown in fig. 2. All can be readily mapped into the Xilinx 4000 CLBs. There is a performance price to be paid for this gate efficiency - the computation takes at least B clocks.

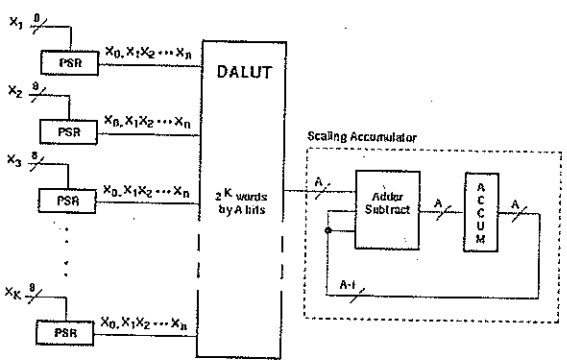

Figure 2 : DA Representation

6. DA On Incremental Form

Fig 3 shows PID based control system; here the output from ADC will be 8 bit if we are using an 8 bit ADC. This should be multiplied with step size and then only we get P. The difference between P and P_d gives error e[n] which forms the input to controller and it produces a desired controller output u[n].

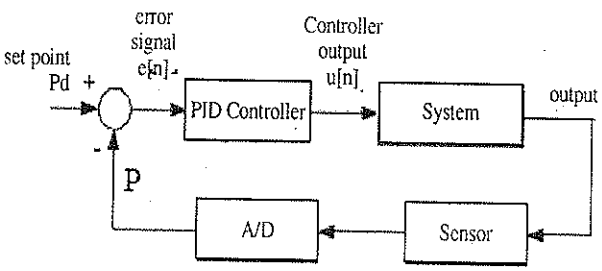

Figure 3 : PID Based Control System

We need to create 2 LUT for implementing PID with distributed arithmetic. A LUT can be of 2^k x 1 size, where k is the number of inputs to LUT.

LUT_1 for the ADC Side

Here we have to find P by the following equation

$$P = a[n] * s \qquad (10)$$

Where, a[n] is the n bit output from ADC s is the step size

so for the LUT we have 1 input a[n], so we form 2^1 x 1 LUT. The contents of LUT_1 table are as shown in Table 1.

Table 1 : Contents of LUT_1

a[n] input	P
0	0
1	s

Fig 4 shows the block diagram of ADC side to implement. The a[n] output from ADC acts as the input to LUT_1. It will be placed in a parallel serial register and will be giving out 1 lsb at 1 clock this become the input to LUT. In LUT value corresponding to the input bits '0' and '1'

is pre- computed and stored so as the input bit come corresponding output will be given out from LUT which will be given to adder and then to accumulator and shifted by 2^{-1}. After shifting its fed back to adder/subtractor and gets added with next output of LUT. Thus after n clock cycles the result of multiplication (actually done by shifting and addition) is obtained in P.

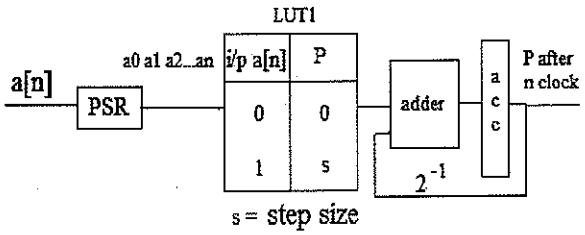

Figure 4 : Block Diagram of ADC Side

LUT_2 for error containg part of incremental form

Equation (6) can be written as shown below

$$u[n] = u[n-1] + E \qquad (4.14)$$

Where, u[n] is the controller output

u[n-1] is the past controller output

$$E = k_0 e[n] + k_1 e[n-1] + k_2 e[n-2] \qquad (4.15)$$

Here e[n], e[n-1], e[n-2] are the inputs to LUT. Size of LUT will be 2^3 x 1. The contents of LUT_2 table are as shown in Table 2. As in Fig 5 the input P (which comes as the output of LUT_1 after n clock cycles) is subtracted from P_d and it form the error e[n] which will put in parallel serial register and will output 1 lsb at 1 clock.

Table 2 : Contents of LUT_2

e[n]	e[n-1]	e[n-2]	E
0	0	0	0
0	0	1	k_2
0	1	0	k_1
0	1	1	$k_1 + k_2$
1	0	0	k_0
1	0	1	$k_0 + k_2$
1	1	0	$k_0 + k_1$
1	1	1	$k_0 + k_1 + k_2$

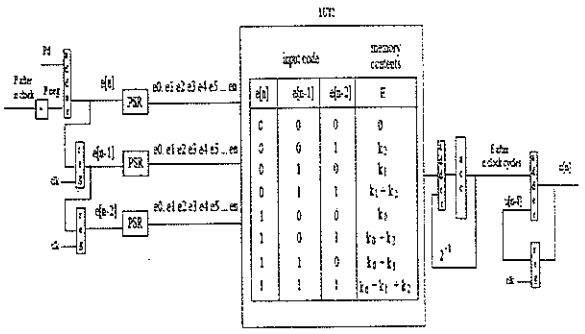

Figure 5 : Block Daigram of Incremental Equation Side

At the rising edge of control, signal e(n) of the last cycle is latched at register reg1, thus becomes e(n-1) of this cycle. In the same manner, e(n-2) and u(n -1) are recorded at reg2 and reg3 by latching e(n -1) and u(n) respectively. At first instant e[n-1] and e[n-2] are zero. e[n],e[n-1],e[n-2] forms the input of LUT_2 and as said before the output gets added and shifted by 2^{-1}. And after n cycles we get the u[n] which is added with previous u[n-1] to give the output of controller.

7. Simulation Results

In the conventional method's coding output from 8 bit ADC is multiplied with the step size. This output of this multiplication forms P input which will be stored as 'Pf' containing fractional part and 'Pi' containing integer part.

Its twos compliment is taken an added with the desired output 'Pd' the fractional part of 'P' and 'Pd' is first added and if it produces a carry it will be added with their integer portions.When reset zero initial values of e[n], e[n-1], e[n-2], u[n-1] are made zero. The output value will be obtained only when clock is given.

The error obtained is stored as separate integer and fraction portion. The value of e[n] will be latched to e [n-1] and e[n-2] as explained previously. Calculations are done taking fractional and integer portion separately. The fraction part of output is obtained in 'outputf' and the integer portion in 'outputi'.

The RTL schematic of code is as shown in Fig 6, the multipliers and adders are clearly seen on the diagram. The simulation wave form is shown in Fig.7. Conventional implementation of FPGA based controllers have not focused on optimal use of hardware resources. These designs usually require a large number of multipliers and adders and do not efficiently utilize the memory-rich characteristics of FPGAs. So we now go for Distributed Arithmetic scheme which uses LUTs of FPGA.

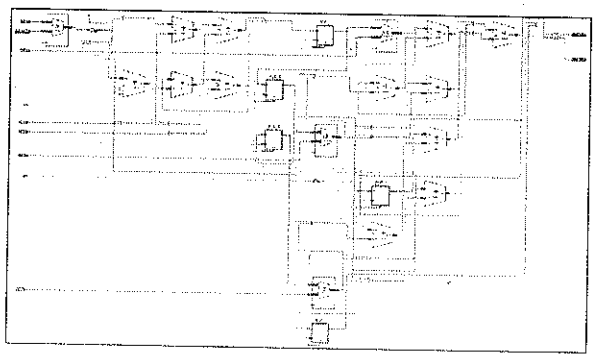

Figure 6 : RTL Schematic

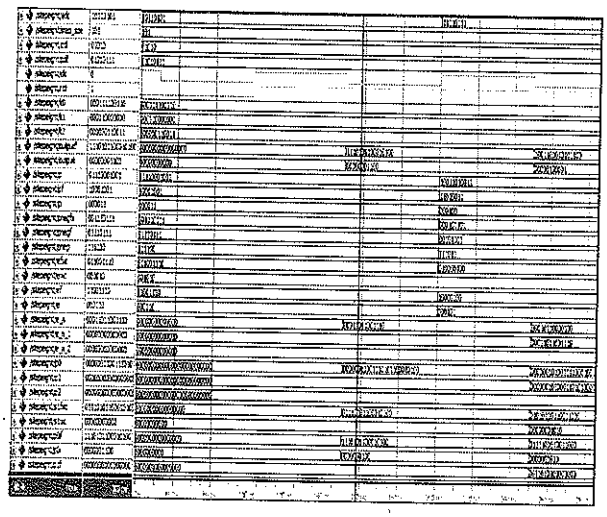

Figure 7 : Simulation Waveform

8. Conclusion

In this paper conventional method of incremental pid controller has been simulated in model sim, now going to fuse distributed arithmetic algorithm onto this incremental equation. By proposed DA-based LUT scheme, the memory inside FPGA has been utilized to provide efficient design for PID controllers.

Future works include writing code for a DA scheme based on the block diagram designed. Implementation of algorithms on FPGA SPARTAN 2E and hardware interface like a temperature control system.

References

[1] Yuen Fong Chan, M. Moallem, Member IEEE and Wei Wang, MemberIEEE, *"Design and Implementation of Modular FPGA-Based PID Controllers"*, IEEE Transactions on Industrial Electronics, Vol.54, No.4, August 2007.

[2] Yuen Fong Chan, M. Moallem and Wei Wang, *"Efficient Implementation of PID Control Algorithm using FPGA Technology"*, 43rd IEEE Conference on Decision and Control , PP.14-17, December 2004.

[3] Wei Zhao, Byung Hwa Kim, Amy.C. Larson and Richard M. Voyles, *"FPGA Implementation of Closed-Loop Control System for Small-Scale Robot"*, Seagate Technology, Shakopee, MN, advanced robotics, 2005. ICAR '05. Proceedings, 12th international conference Publication Date: 18-20 July 2005.

[4] A.White, *"Application of distributed arithmetic to digital signal processing: A tutorial review"*, IEEE Trans. Acoust Speech Signal Process, Mag, Vol.6,No.3,PP. 4 to 19, Jul 1989.

[5] *" The role of distributed arithmetic in FPGA based signal processing"*

Author's Biography

Mr. K. Rajsekaran, received B.E. ECE in PSG tech, Coimbatore in the year 1976 and M.E (Instrumentation) from IIT Kharagphur in 1978. Currently pursuing Ph.D in Anna University Coimbatore. He is working as HOD for the past 7 years.

Mariamma John, P.G. scholar from Karunya university, Coimbatore.

Anitha Mary. X, received Bachelor of Engineering (electronics and instrumentation engineering) from Karunya Institute of Technology in the year 2001 and Master of Engineering (VLSI Design) from Anna University, Coimbatore in the year 2009. Currently pursuing Ph.D in Control System area. She attended 3 National and 4 International Conference. She guided many U.G. and P.G. projects. Currently working as Assistant Professor in the EIE department.

Thilagavathi completed U.G. from SACS MAVMM Engineering college Madurai in the year 2002 and P.G from Arulmigu Kalasalingam college of engineering in the year 2008. She attended 2 national and 4 international conferences.