

PID controller IP Core user manual

Author: Zhu Xu

Email: m99a1@yahoo.cn

Introduction

The PID controller IP core performs digital proportional–integral–derivative controller (PID controller) algorithm. The algorithm first calculates the error between a measured value (PV) and its ideal value (SP), then use the error as an argument to calculate the manipulate value(MV). The MV will adjust the process to minimize the error. It can be used to calculate duty cycle for PWM (Pulse Width Modulation).

Features

- 16-bit signed coefficient and data input: Kp, Ki, Kd, SP and PV.
- 32-bit signed $u(n)$ output.
- Containing one high speed 32-bit prefix-2 Han-Carlson adder and one high speed pipelined 16x16-bit multiplier.
- Latency from input of PV to finished calculation and update of $u(n)$ is 17 clock cycles.
- Ki, Kp, Kd, SP, PV can be updated anytime after reset.
- After every update of Kp or Kd, register Kpd which stores $Kp+Kd$ will be calculated and updated.
- After every update of PV, calculation and update of $e(n)$, $e(n-1)$, sigma and $u(n)$ will be triggered in sequence.
- Overflow register records overflow signals when calculating Kpd, $e(n)$, $e(n-1)$, $u(n)$ and sigma.
- Using 2278 of 4608 (49%) Core Cells in Actel A2F200M3F FPGA and running at 100MHz clock frequency.
- Wishbone B4 compliant interface. Support 16-bit, 32-bit and 64-bit bus width.

PID controller description

Below describe the algorithm of PID controller.

Analog

A PID is made of three basic blocks whose outputs are:

- Proportional to the input
- The integral of the input

- The derivative of the input

Figure 1 shows the block diagram of a generic system controlled by PID controller. The goal of the PID block is to generate an output $u(t)$ that drives the system at hand (the "PLANT") so that its output $[y(t)$ or PV, Process Value] matches a reference signal $[x(t)$ or SP, Set Point]. The input to the PID is the error between the reference signal (ideal or desired behavior of the PLANT) and the real output behavior. Obviously, the target is to get the error as close to zero as possible.

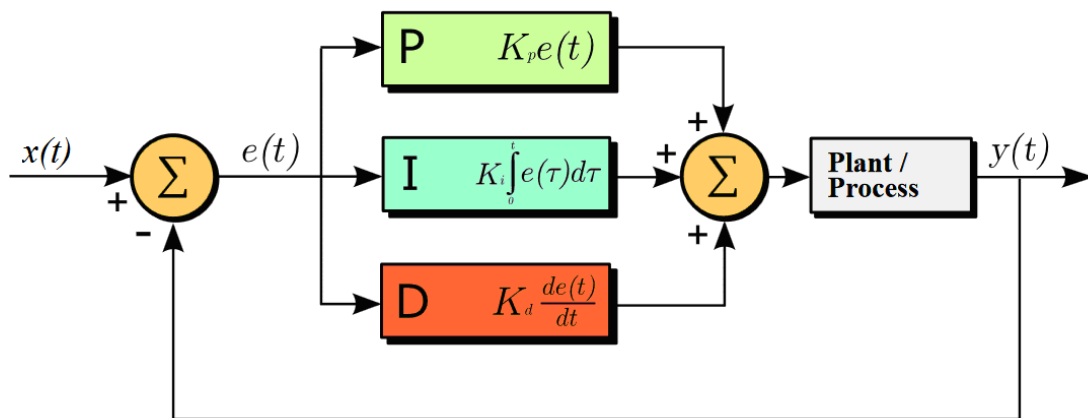


Figure 1: a generic system controlled by PID

The equation that describes the PID controller behavior in continuous time domain is shown in equation 1.

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Equation 1

Digital

Transform equation 1 into discrete time domain we get equation 2.

$$u(n) = (K_p + K_d) e(n) + K_i \sum_{j=1}^n e(j) - K_d e(n-1)$$

Equation 2

From equation 2 we can construct the algorithm's block diagram shown in figure 2.

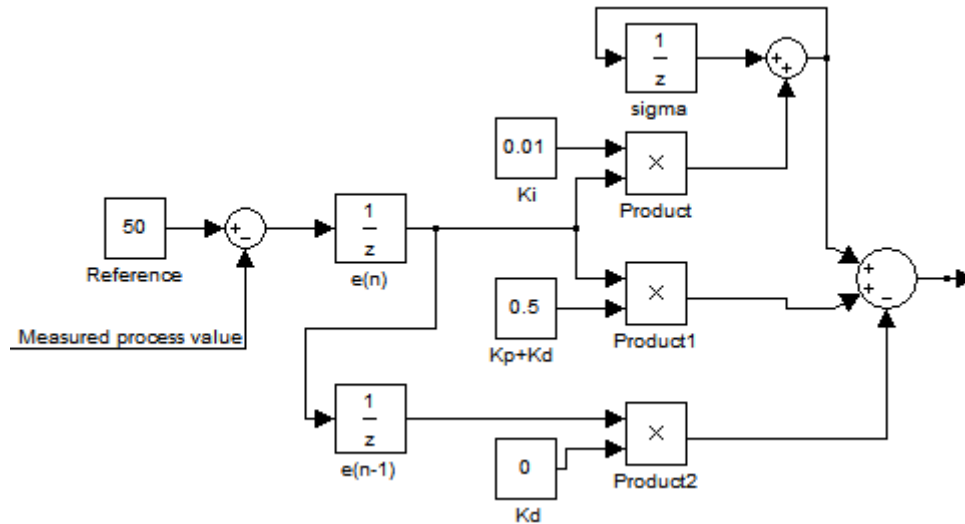


Figure 2: digital PID controller algorithm

Operation

Coefficients and Data Update

Coefficients (K_p , K_i , K_d , PV) and measured process value (PV) are stored in different registers that be read and written any time after reset by a host through Wishbone slave interface. Normally, K_p , K_i , K_d , PV are updated right after reset before continuously update of PV . You also can update coefficients randomly for dynamic tuning. Writing action to specific registers mentioned above won't be responded until finished calculation of the last $u(n)$.

Calculation of $u(n)$

Calculation of $u(n)$ will be triggered every time PV is updated. The calculation procedure is demonstrated in figure 3.

Update of overflow register

Overflow register records any overflow signal in previous calculations of K_p , $e(n)$, $e(n-1)$, $u(n)$ and σ . It will be updated after every addition operation. If any of the 5 overflow register bits is set, then the final $u(n)$ is incorrect.

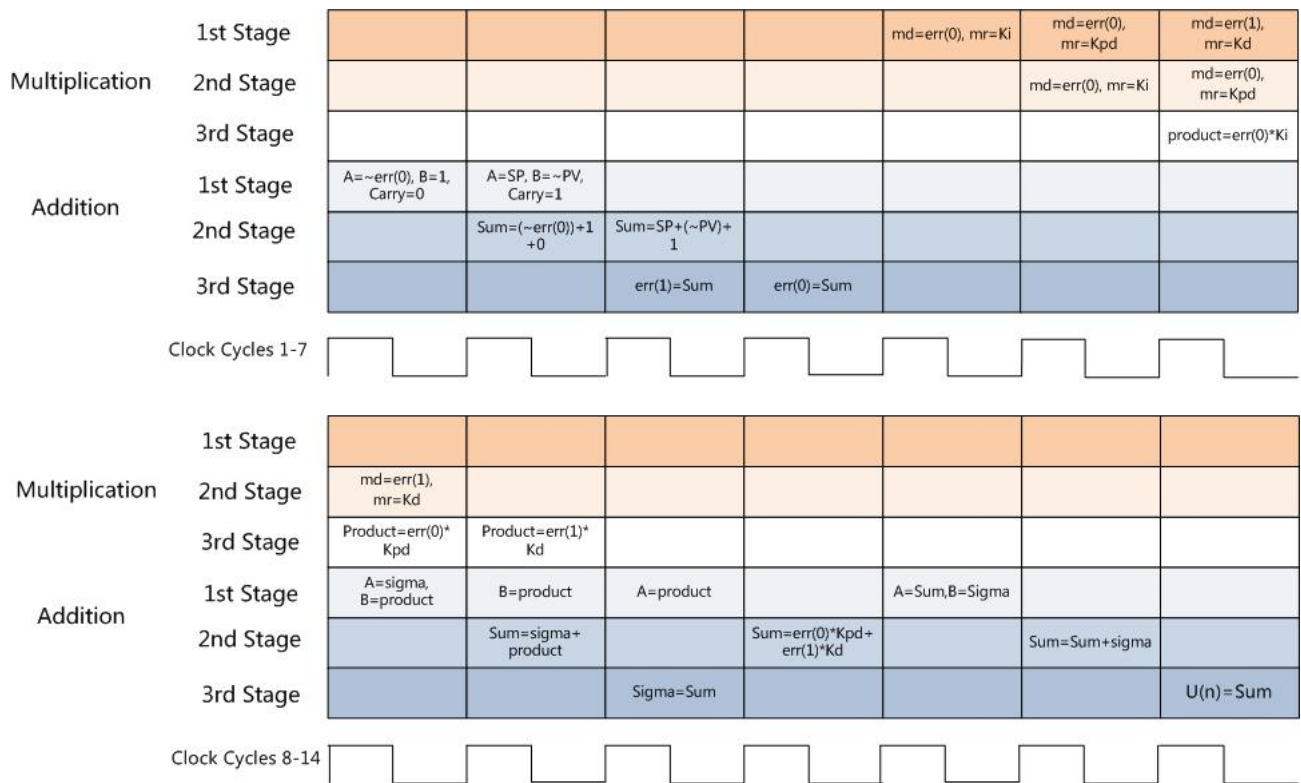


Figure 3: calculation procedure of u(n)

Registers

| Name | Address for 16-bit/32-bit/64-bit bus width | Width | Access | Description |
|--------|--|-------|--------|---|
| Kp | Base+0x0/0/0 | 16 | R/W | Stores coefficient Kp |
| Ki | Base+0x2/4/8 | 16 | R/W | Stores coefficient Ki |
| Kd | Base+0x4/8/10 | 16 | R/W | Stores coefficient Kd |
| SP | Base+0x6/C/18 | 16 | R/W | Stores reference SP |
| PV | Base+0x8/10/20 | 16 | R/W | Stores PV |
| Kpd | Base+0xA/14/28 | 16 | R | Stores coefficient Kp+Kd |
| err[0] | Base+0xC/18/30 | 16 | R | Stores e(n) |
| err[1] | Base+0xE/1C/38 | 16 | R | Stores e(n-1) |
| un | Base+0x10/20/40 | 32 | R | Stores u(n) |
| sigma | Base+0x12/24/48 | 32 | R | Stores $K_i \sum_{j=1}^n e(j)$ |
| of | Base+0x14/28/50 | 5 | R | of[0]=1 if Kpd overflows, of[1]=1 if err[0] overflows, of[2]=1 if err[1] overflows, of[3]=1 if un overflows, of[4]=1 if sigma overflows |

I/O ports

The IP core has a Wishbone Slave Interface and another interface for direct 32-bit u(n) output.

Wishbone Slave Interface

It's Wishbone B4 compliant.

| Name | Name in Wishbone B4 | Size | Direction | Description |
|-----------|---------------------|----------|-----------|--|
| i_clk | CLK_I | 1 | input | Clock input |
| i_rst | RST_I | 1 | input | Reset input |
| i_wb_cyc | CYC_I | 1 | input | Indicates valid bus cycle (core select) |
| i_wb_stb | STB_I | 1 | input | Indicates valid data transfer cycle |
| i_wb_we | WE_I | 1 | input | Write transaction when asserted high |
| i_wb_adr | ADR_I | 16 | input | Address input |
| i_wb_data | DAT_I | 16/32/64 | input | Data input |
| o_wb_ack | ACK_O | 1 | output | Acknowledgment output (indicates normal transaction termination) |
| o_wb_data | DATA_O | 16/32/64 | output | Data output |

Direct 32-bit u(n) output

| Name | Size | Direction | Description |
|---------|------|-----------|----------------------|
| o_un | 32 | output | u(n) output |
| o_valid | 1 | output | Indicates valid o_un |