AN1298
APPLICATION NOTE
PID CONTROLLER USING ST52X420 MCU

Author: N. Abbate

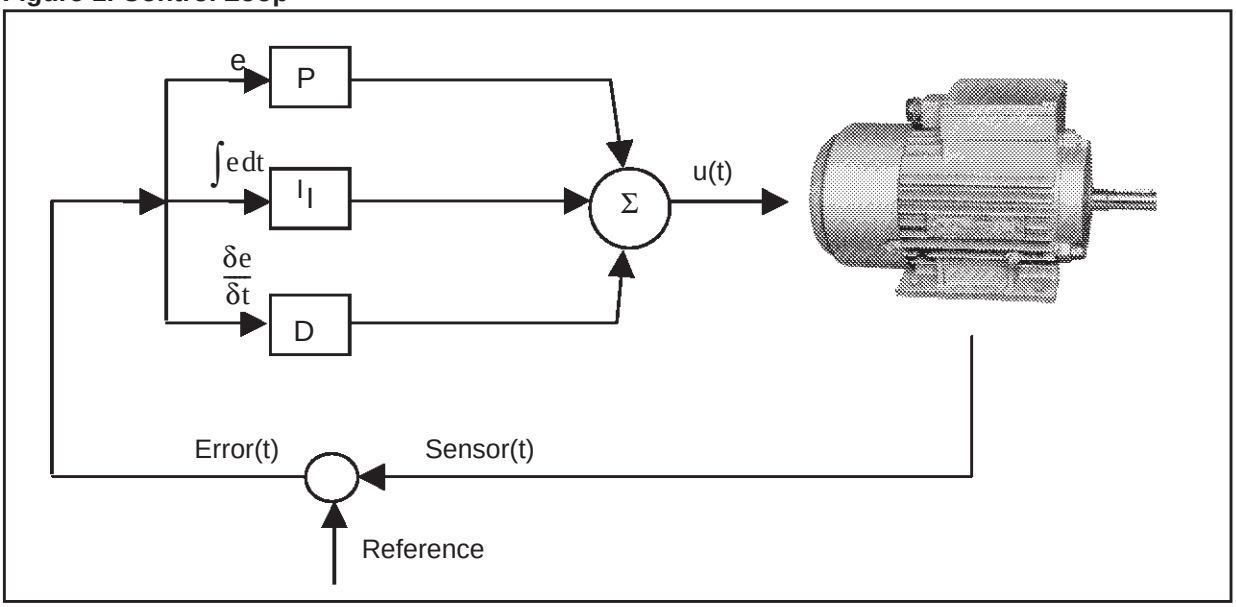
1. INTRODUCTION

Intention of this application note is to explain the implementation of a PID controller using the ST52x420 microcontroller.

PIDs have a fundamental importance in the industrial controller field, for the simplicity of approach, the robustness of control and the immediacy of calibration.

PID controller is a typology of regulator that implements proportional, integral and derivative action. In fig.1 is shown the control loop of a standard PID regulator.

Figure 1. Control Loop



In this regulator, the variable *"error"*, the derivative and the integral errors are performed to calculate the output variable *u(t)*, function of the above.

The PID algorithm is given by:

$$u(t) = k_p e(t) + k_i \int e dt + k_d \frac{\partial e}{\partial t}$$

where K_p, K_i, K_d are PID gain constants, *u(t)* is control signal and *e(t)* is error signal.

To convert to discrete form, the integral term $\int e dt$ is approximated with $\Sigma e_i T$, where *T* is the sampling interval and e_i is the value of the error signal at sample time *i*, written as :

$$\int e dt \cong \Sigma e_i T$$

If the sampling time is smaller, the differential term $\frac{\delta e}{\delta t}$ can be approximated like

$$\frac{\delta e}{\delta t} \cong \frac{e(n) - e(n-1)}{T}$$

where *e(n)* and *e(n-1)* are values of the error signal *e* at time intervals *n* and *n-1*.

The PID algorithm can now be approximated in discrete form by:

$$u(n) = k_p e(n) + k_i \Sigma e_i T + k_d \left[\frac{e(n) - e(n-1)}{T}\right]$$

In this form it is simpler to implement an algorithm. The constants **k(p,i,d)** can be designed using various methods such as Ziegler Nichols, phase margin, critical frequencies, heuristic etc.

The choice between these techniques depends on the type of application because it is extremely different to implement a PID regulator in a system where only the mathematical models is known (i.e. transfer's function), or in a non-linear system, where only input/output measures are possible.

A brief overview to some of the effects linked to the influence of proportional, integral or derivative action is shown in the following paragraph.

2. INFLUENCE OF PID PARAMETER

Three are the parameters that influence the performances of a PID controller: Proportional Integral and Derivative constants. Each one has a different impact on the system and an appropriate mix of these three parameters provides a good regulator. However, it is not necessary to use all the parameters at the same time.

Proportional actions express the control action proportional to the error. This is very efficient to reduce external noise and disturbance in our system and guarantee stability of closed loop.

Integral action is essential to make sure that the process output agrees with the set point in steady state.

Derivative action may be interpreted as if the control is made proportional to the forecasted process output, where the prediction is made by extrapolating the error by the tangent to the error curve.

Another main parameter is the "integration time", but this is already inside the PID regulator.

Extremely important is the Wind-up of integral action. When a large error is present in the system, for instance when a large step disturbance is encountered, the integrator continually builds up even though the output is saturated. This condition is called "wind up". When the system exits from this condition, the integrator "unwinds" causing excessive oscillation. This integral PID implementation avoids this problem by stopping the action of the integrator during output variable *duty* saturation.

3. PID IMPLEMENTATION

ST52x420 has on-chip hardware multiplication between two bytes. This functionality is very important for the correct flow of the algorithm, because the variable error, created from difference between encoder and reference signals in an apposite folder called "*encoder*", is manipulated by the PID regulator to generate the output control signal mainly using multiplication.

FUZZYSTUDIO™ 4 development tool allows to design the regulator in a simple way.

The main block of the controller is an ALU block called PID, where the input variable is manipulated and stored, to create the signed-byte control variable *appsbyte* that increments or decrements the value of the control variable *duty*.

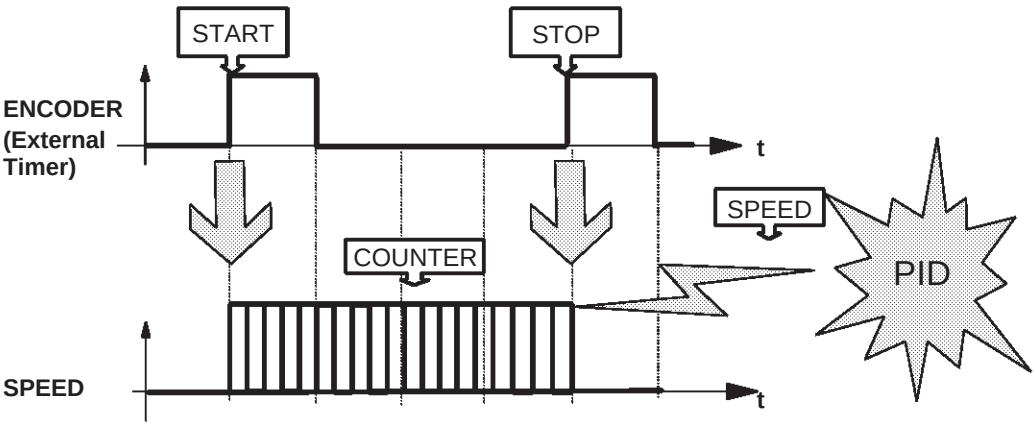
The Instruction set of ST52x420 allows to use different types of variables: word, signed-byte, word and byte. The problem consists in translating the information content in a multiplication between signed-byte and constants (obtaining signed-word variable) in a signed-byte variable.

This operation is executed using a scaling between the variable *"increment"* (signed-word), using only the high side of double byte that represents the signed-word, and reduce this quantity of a value corresponding to 128; the result is stored in a signed-byte variable.

4. FUNCTIONAL DESCRIPTION

In this application, an external encoder acquires the angular speed of an asynchronous motor. After the acquisition, and the appropriate conditioning, the signal is stored in the variable *speed*. This action has been performed with an appropriate procedure called *encoder*.

Figure 2. Speed acquisition task



Inside the interrupt routine of the PWM1 peripheral, a counter is used to modify the sampling time of regulator, to define the opportune integration time for each application.

So the real regulator's time loop vary from 50µs to 50µs*counter [byte/word] with an incremental software loop.

The control sampling time is synchronized with 20KHz PWM of Timer_1 Peripheral, employed to supply the asynchronous motor with a patented board.

Figure 3. PWM configuration

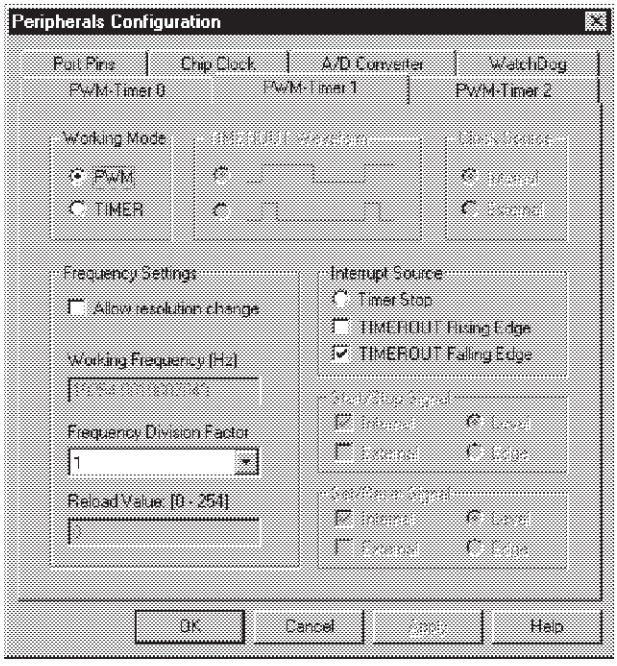
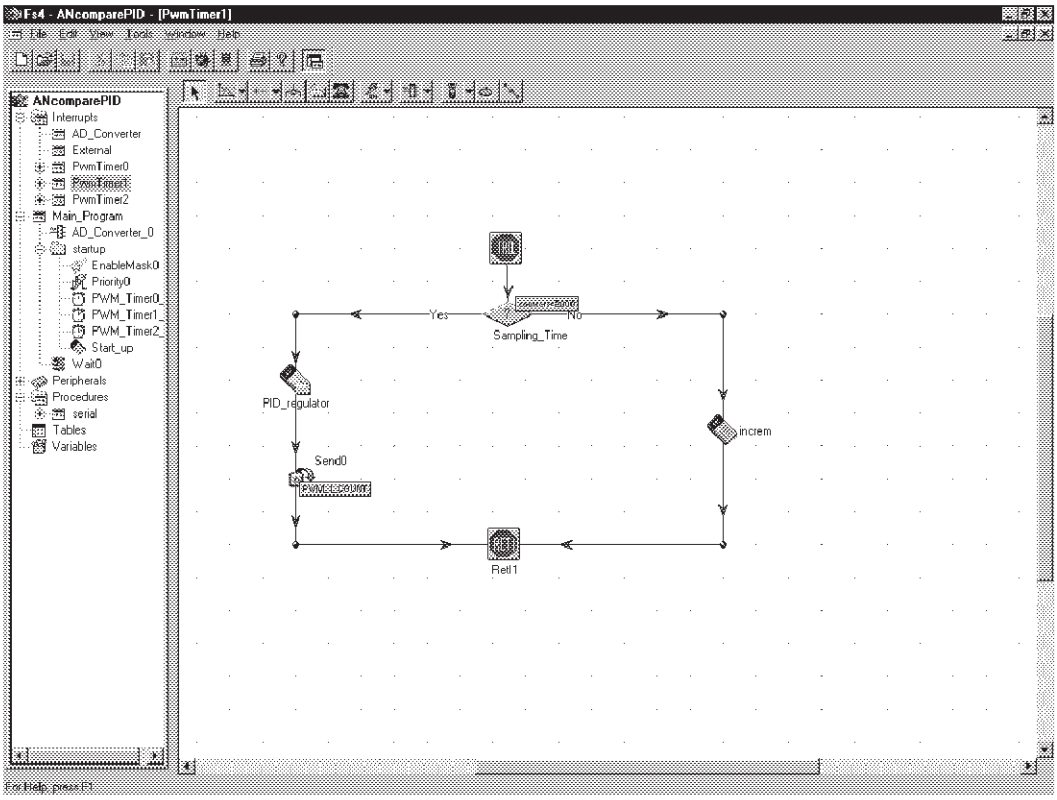


Figure 4. Interrupt routine "PWM1" in FUZZYSTUDIO™4 implementation



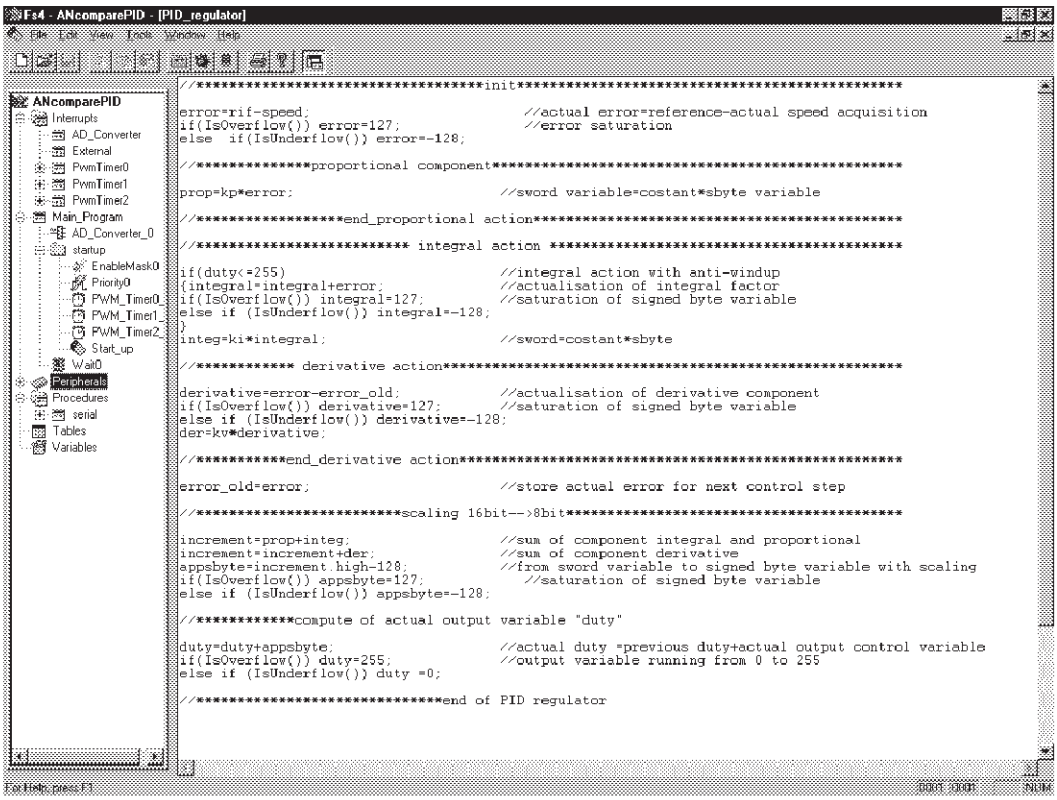
In each control's step, the variable called *error* (signed-byte), is computed and manipulated to perform the appropriate control law with the following procedure:

- The acquired signal speed is compared now with the requested reference *ref* and the difference is stored in the variable called *"error"*.
- An underflow/overflow control is actuated using the preset FUZZYSTUDIO™ 4 function.
- A variable called *"prop"* implements the proportional action, multiplying the error by the constant K_p. This constant is representative of the proportional PID parameter so like K_i and K_d represents respectively the integral and the derivative parameters, set like constants in FUZZYSTUDIO™ 4.
- The integral action is achieved updating the variable integral adding the current value of "*error*"; after the overflow/underflow control, the variable "*integ*" is computed like *Ki*error* [signed-word]. In this firmware an integrator anti-windup has already been implemented thus avoiding the wind-up problem by stopping the action of the integrator when the output variable "*duty*" is saturated.
- The derivative action is obtained calculating the difference between the current and the previous step "*error" (error_old)*. The result is multiplied by *Kd* to obtain the signed-word "*der*".
- The actual value of "*error*" to be used in the next step is stored in a support variable called *"error_old*".
- After this phase, summing the quantity above, we obtain a signed-word variable called "*increment*". Now, we need to scale them in a signed-byte with an opportune algorithm to maintain the 16-bit information content in an 8-bit signed variable. So we use a variable called "*appsbyte*" to perform an increment or a decrement of the output control variable (precisely the duty cycle of the 20 KHz PWM).

In the following figure is represented the ALU block inside the firmware that controls the washing machine's motor.

In FUZZYSTUDIO™4 the ALU blocks allow to design a firmware in a "C"-like coding, with powerful dedicated functions to manipulate byte, word, signed, under/overflow control, peripherals' setting, mathematical functions, variable assignation and more.

Figure 5. ALU block"PID_Regulator" in FUZZYSTUDIO™4 implementation



5. RESULTS AND CONCLUSION

This application shows a simple way to implement a PID controller with the ST52x420 micro. Using this approach it is possible to obtain a regulator with performances suited to the application, calibrating the typology of control action in function of the system.

The strategy of calibration has not been treated here, but in reference [1] various techniques are discussed in depth.

The following figures show the performances of a washing machine motor's regulator in start-up condition, changing speed reference on-fly and load variation.

This acquisition performed with an on-board serial communication software, represents the comparison between the simultaneous input speed and output duty-cycle of the asynchronous motor driver. The targets achieved are:

1. Reach the reference speed.
2. Noise and error disturbance reduction.
3. Low employment of the micro resources to allow other applications' functionalities.

Figure 6. Start-up

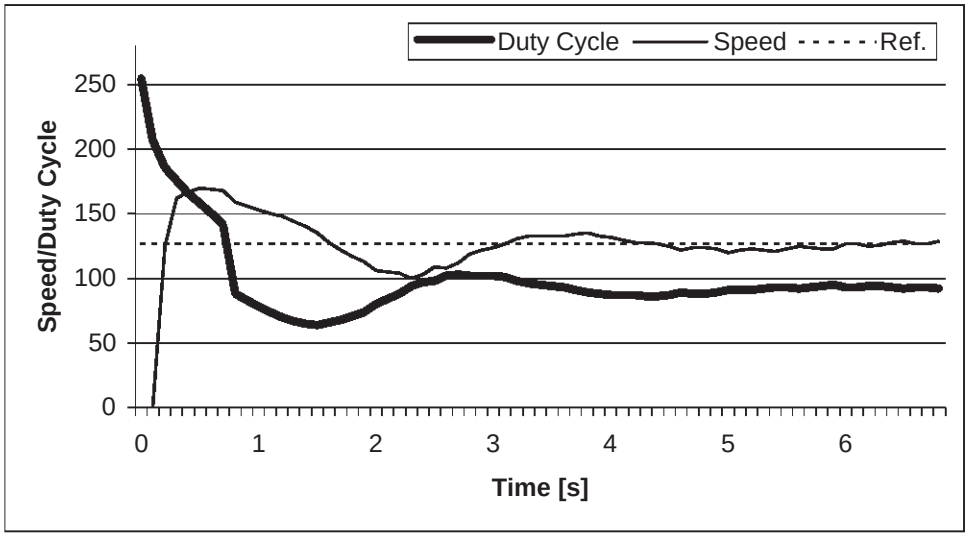


Figure 7. Change reference with anti-wind-up action

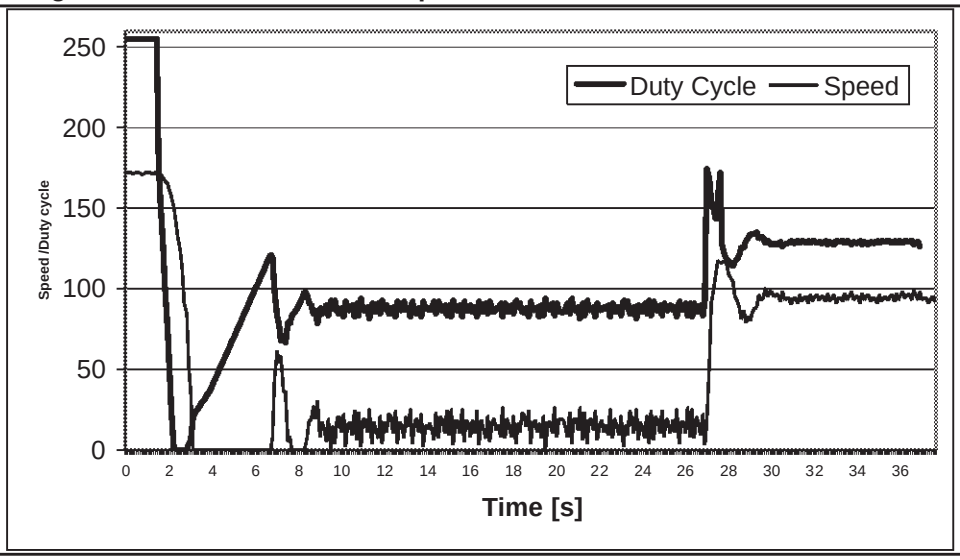
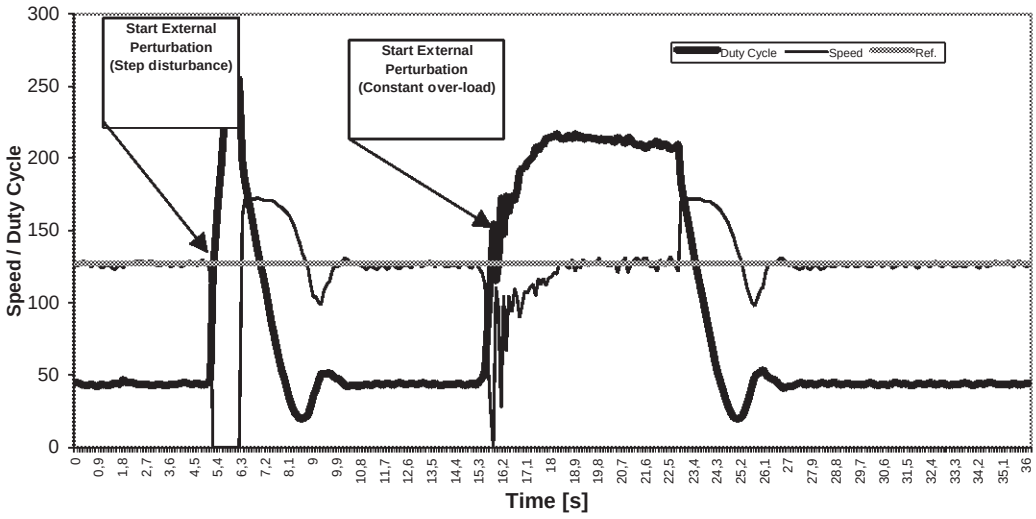


Figure 8. Step stimulus and load variation



REFERENCES

[1] K.J. Åström, T. Hägglund ”*PID Controllers*”, 2nd Edition, Instrument Society of America 1995.
[2] FUZZYSTUDIO™ 4.0 User Manual, STMicroelectronics 2000

APPENDIX: ST52 FS CODE

```
//***********************Init***************
error=rif-speed;                    //actual error=reference-actual  speed acquisition
if(IsOverflow()) error=127;         //error saturation
else if(IsUnderflow()) error=-128;
//********** Proportional action****************
prop=kp*error;                      //sword variable=costant*sbyte  variable
//************** End_proportional action ******************
//************** Integral action *************************
if(duty<=255)                       //integral action with anti-windup
{integral=integral+error;           //actualisation of integral factor
if(IsOverflow()) integral=127;      //saturation of signed byte variable
else if (IsUnderflow()) integral=-128
};
integ=ki*integral;                  //sword=costant*sbyte
//***************** End_integral action********************
//******************** Derivative action *****************
derivative=error-error_old;         //actualisation of derivative component
if(IsOverflow()) derivative=127;    //saturation of signed byte variable
else if (IsUnderflow()) derivative=-128;
der=kv*derivative;
//*************** End_derivative action ******************
error_old=error;                    //store actual error for next control step
//*************** Scaling 16bit-->8bit ******************
increment=prop+integ;               //sum of component integral and proportional
increment=increment+der;            //sum of component derivative
appsbyte=increment.high-128;        //from sword variable to signed byte variable with
                                      scaling
if(IsOverflow()) appsbyte=127;      //saturation of signed byte variable
else if (IsUnderflow()) appsbyte=-128;
//******* Compute of actual output variable ”duty” ********
duty=duty+appsbyte;                 //actual duty previous duty+actual output control
                                      variable
if(IsOverflow()) duty=255;          //output variable running from 0 to 255
else if (IsUnderflow()) duty =0;
//****************** End of PID regulator *************
```

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.