

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

185,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



# Design of Three-Term Controller Using a PIC18F452 Microcontroller

*Mostefa Ghassoul*

## Abstract

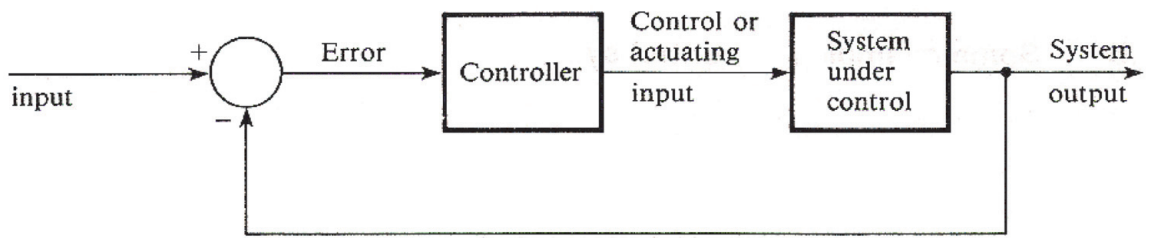
Microcontrollers are used in almost any applications that come across one's mind, from small control applications such as home appliances to aerospace. Microcontroller-based controllers are cost-effective and flexible to modify the design to meet the requirement for any control of any industrial plant. Microcontrollers do not require external hardware interface, memories, counter/timers, and ADCs, because they are all integrated inside the chip. Those controllers could be programmed online and do not require any backup memories except for big applications. This chapter presents the implementation of the three-term PID controller using a Microchip PIC18F452 microcontroller. To read data into the controller, a 10-bit integrated ADC is used; and to read data out of the machine, an external 12-bit serial DAC is used. Before programming the PIC microcontroller, the task to be tested could be off-line using a software simulator to make sure that it is working according. When that is the case, it could be then fired into the controller on-line in a matter of seconds. Not only that, if the user decides to use a different algorithm, he only programs the controller again online.

**Keywords:** PIC microcontroller, PID, timer, Digital to analog converter, serial digital to converter, liquid crystal display

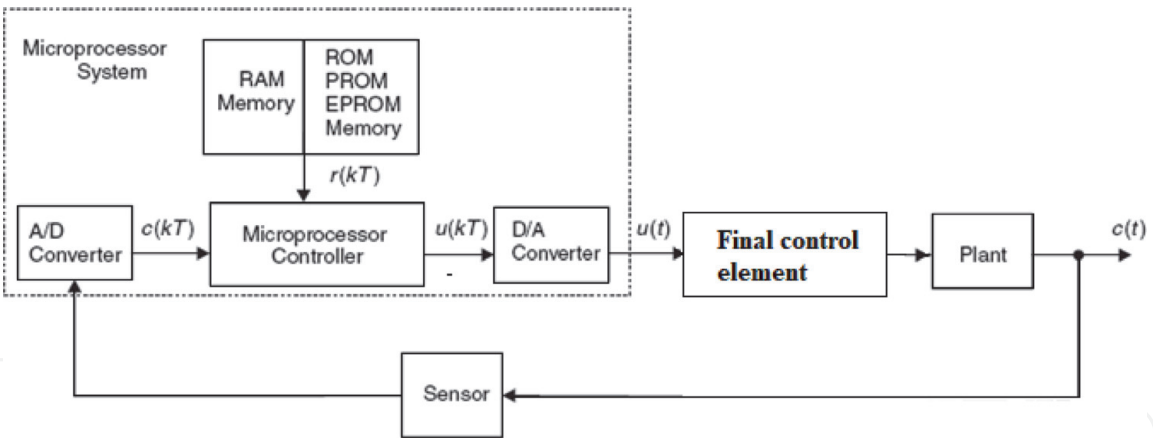
## 1. Introduction

The PID (Proportional Integral Differential) algorithm is the most popular feedback controller used within the process industries. It has been successfully used for over 50 years. It is a robust easily understood algorithm that can provide excellent control performance despite the varied dynamic characteristics of process plant. It is designed to generate an output that causes some corrective effort to be applied to a process so as to drive a measurable process variable towards a desired value, known as the set point. The concept is based (as shown in **Figure 1**) on the re-input of the system own output according to certain laws (hence the name “feedback”). It is desired for the system output to follow the set point. All feedback controllers determine their output by observing the difference, called error, between the set point and the actual process variable measurement. The PID looks at (a) the current value of the error, (b) the integral of the error over a recent time interval, and (c) the current derivative of the error signal to determine not only how much of a correction to apply, but for how long. Each of those three quantities are multiplied by a (tuning constant) and added together. Thus the PID output is a weighted sum. Depending on the application one may want a faster convergence speed or a lower overshoot. By adjusting the weighting constants,  $K_p$ ,  $K_i$ , and  $K_d$ , the PID is set to give the most desired performance.

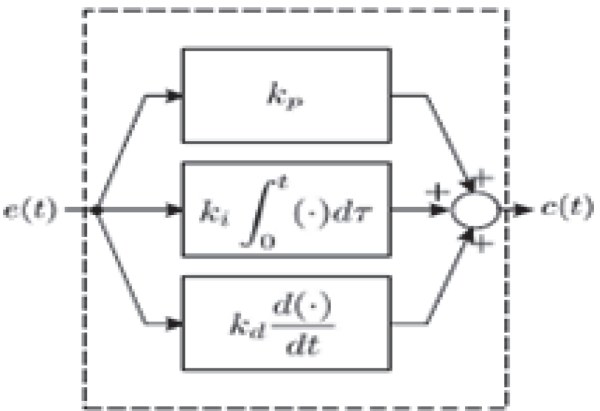
As a result of enormous development in microcomputer technology, analog controllers have been replaced by digital controllers either in small or large industry. It is now a common practice to implement PID controllers in its digital version, which means that they operate in discrete time domain and deal with analog signals quantized in a limited number of levels. The trend toward digital rather than analog control is mainly due to: (1) versatility where programs can be easily modified or completely changed, (2) sophistication where advanced control laws could be implemented, (3) cost effectiveness where microcontrollers are available at very low costs compared to PLCs, industrial computers, RTUs or DCS. A typical digital feedback control system is shown in **Figure 2**. In digital feedback systems, the controller input and output are digital (sampled) rather than continuous signals. Thus, the continuous signal from the measurement device (sensor/transmitter) is sampled and converted periodically to a digital signal by an analog-to-digital converter (ADC). A digital control algorithm is then used to calculate the controller output as a digital signal. Because most final



**Figure 1.**  
*Typical closed loop control system.*



**Figure 2.**  
*Digital closed loop based on a microcontroller.*



**Figure 3.**  
*The three-term PID controller.*

control elements are analog devices, the digital output signal is usually converted to a corresponding analog signal by a digital-to-analog converter (DAC).

In feedback control, the objective is to reduce the error signal to zero where

$$e(t) = y_{sp}(t) - y_m(t) \quad (1)$$

where  $e(t)$  = error signal,  $y_{sp}(t)$  = set point and  $y_m(t)$  = measured value of the controlled variable. For the PID controller, the three terms (proportional, integral, and derivative) are combined to minimize the error as much as possible. The most common combination of these three terms is in parallel as shown in **Figure 3**. The PID equation [1, 2] is given by:

$$P(t) = \bar{P} + K_C \left[ e(t) + \frac{1}{\tau_I} \int_0^t e(t) dt + \tau_D \frac{de(t)}{dt} \right] \quad (2)$$

where  $P(t)$  = controller output;  $\bar{P}$  = bias (steady-state) value;  $K_C$  = controller gain;  $\tau_I$  = integral time;  $\tau_D$  = derivative time.

A straightforward way of deriving a digital version of the parallel form of the PID controller is to replace the integral and derivative terms by finite difference approximations,

$$\int_0^t e(t) dt \approx \sum_{j=1}^k e_j T_S \quad (3)$$

$$\frac{de}{dt} \approx \frac{e_k - e_{k-1}}{T_S} \quad (4)$$

where  $T_S$  = the sampling time (the time between successive measurements of the controlled variable); and  $e_k$  = error at the  $k$ th sampling instant for  $k = 1, 2, 3, \dots$

There are two alternative forms of the digital PID control equation, the position form and the velocity form. Substituting Eqs. (3) and (4) into (2) gives the *position form*:

$$P_k = \bar{P} + K_C \left[ e_k + \frac{T_S}{\tau_I} \sum_{j=1}^k e_j + \frac{\tau_D}{T_S} (e_k - e_{k-1}) \right] \quad (5)$$

where  $P_k$  is the controller output at the  $K$ th sampling instant. Eq. (5) is referred to as the position form of the PID control algorithm because the actual value of the controller output is calculated [3, 4].

In the *velocity form*, the change in controller output is calculated. The velocity form can be derived by writing the position form of Eq. (5) for the  $(k-1)$  sampling instant:

$$P_{k-1} = \bar{P} + K_C \left[ e_{k-1} + \frac{T_S}{\tau_I} \sum_{j=1}^{k-1} e_j + \frac{\tau_D}{T_S} (e_{k-1} - e_{k-2}) \right] \quad (6)$$

Note that the summation still begins at  $j = 1$  because it is assumed that the process is at the desired steady state for  $j \leq 0$ , and thus  $e_j = 0$  for  $j \leq 0$ . Subtracting Eq. (6) from (5) gives the velocity form of the digital PID algorithm:

$$\Delta P_k = P_k - P_{k-1} = K_C \left[ (e_k - e_{k-1}) + \frac{T_S}{\tau_I} e_k + \frac{\tau_D}{T_S} (e_k - 2e_{k-1} + e_{k-2}) \right] \quad (7)$$

$$P_k = P_{k-1} + K_C \left[ (e_k - e_{k-1}) + \frac{T_S}{\tau_I} e_k + \frac{\tau_D}{T_S} (e_k - 2e_{k-1} + e_{k-2}) \right] \quad (8)$$

In this study, velocity form is chosen because of the following advantages:

1. It does not need initialization. The position form requires the initial value of the controller output  $\bar{P}$ , which is not normally known in practice. For example, an operator keeps the control loop in the manual mode until a desired steady state operation has been reached. At this point the error is zero and the position of the control valve would correspond to the  $\bar{P}$  value. Therefore, if the operator would like to transfer the control from manual to automatic, he or she should enter in the position control algorithm the value of  $\bar{P}$  which is not normally known. This difficulty can be bypassed with the velocity form of the control algorithms, which do not need initialization.
2. It is protected against integral windup. The integral mode of a controller causes its output to continue changing as long as there is a nonzero error. Often the errors cannot be eliminated quickly enough and given enough time they produce larger and larger values for the integral term, which in turn keeps increasing the control action until it is “saturated” (e.g., the valve completely opens or closes). This condition is called *integral windup*. Then, even if the error returns to zero, the control action will remain saturated. The position form with its continuous summation of errors will produce integral windup and special attention will be required. The velocity form, on the other hand, is protected from integral windup for the following reason: The control action changes continuously until it becomes saturated. But then as soon as the error changes sign, the control action can return within the control range in one sampling period.
3. It protects the process against computer failure. With the velocity algorithm one can send out a signal which is used to drive an integrating amplifier or a stepper motor. These devices will retain the last calculated position of the control valve (or other final control element) in case the computer fails, thus avoiding total loss of control of the process.

## 2. PIC18F452 background

As mentioned earlier, the implementation is based on a Microchip PIC18F452 microcontroller, where the controller plays the role of the brain of the control system [5]. The right choice of the microcontroller is essential, as it will be the core of the final design. The PIC18F452 from Microchip has been chosen for the following advantages:

1. Speed: with its maximum internal clock rate of 20 MHz and its 16-bit-wide instruction bus, the CPU can execute most of its instructions at a single machine cycle of four clocks which is equivalent to a 0.2  $\mu$ s.
2. Math support: unlike classical microprocessors, the controller in hand has got a hardware multiplier and divider for multiple-bytes, fixed-point numbers and for floating-point numbers so multiplication is carried out in a single instruction.



3. Flexible timer resources: four independent timers modules support timing measurements and output interval control with a timing resolution as fine as 0.1  $\mu$ s. Those timers could be used to produce up to three pulse width modulations which could be used for electrical motor control.
4. Free software tools: Microchip's Development Package MPLAB<sup>®</sup> (consisting of assembler, simulator, and user interface) as well as all manuals and application notes are available at no cost from their Web site ([www.microchip.com](http://www.microchip.com)).
5. Development tool versatility: it supports in-circuit debugger which permits the loading and execution of a user program as well as the use of breakpoints, memory/ register modification, and single stepping.
6. Build-in ADCs: it has analogue-to-digital converters with 10 bits resolution.
7. Built-in serial peripheral interface: it has a variety of serial bus interfaces like USART, I<sup>2</sup>C & SPI.
8. C programmable: it could be programmed using C language with the use of a variety of built in C libraries developed by microchip.

The PIC18F452 microcontroller is a 40 or 44-pin depending on the package, where in the 40 pins configuration, a dual inline package is used; whereas in the 44 pins configuration, either thin quad flat package or dual flat no leads package is used. Its design is based on Harvard technology where the program and data have different buses. This type of microcontrollers is very cheap, small in size, and could be customized. It could be easily programmed on-line using either assembly language, BASIC or C language. In fact, it is ideal for small application such as the one in hand. The controller has a 24 kbytes of flash memory and 2048 bytes of SDRAM. It also has a  $8 \times 10$  bits analog to digital channels. It also has 5 bidirectional digital ports with 33 inputs/outputs, configured as follows:  $3 \times 8$  digital I/O ports (PORTB, PORTC and PORTD), one six digital I/O port (PORTA) and one three digital I/O port (PORTE). Unfortunately, one of the drawbacks of microcontrollers, it is very seldom to find one with a digital to analog converter. Luckily, there are few manufacturers around including microchip, which make serial DACs which could be programmed through Serial Port Interface (SPI) using only three wires. The PIC18F452 has four timer/counters which could be programmed either as 8 or 16 bit timers/counters. It also has two ports which could be configured either as capture, compare or pulse width modulation (PWM). It has two serial peripheral interfaces: (SPI) and an inter-integrated circuit (I<sup>2</sup>C). An asynchronous port (USART) is also provided. For the microcontroller to output analogue data, an MCP4921 device is used. The device is a 12-bit buffered single voltage output Digital-to-Analog Converter (DAC). The device operates from a single 2.7 V to 5.5 V supply with an SPI compatible Serial Peripheral Interface. The user can configure the full-scale range of the device to be VREF or 2\*VREF by setting the gain selection option bit (gain of 1 of 2). The user can shut down the device by setting the Configuration Register bit. In Shutdown mode, most of the internal circuits are turned off for power savings, and the output amplifier is configured to present a known high resistance output load (500 k $\Omega$ , typical). The device includes double-buffered registers, allowing synchronous updates of the DAC output using the LDAC pin. The device also incorporates a Power-on Reset (POR) circuit to ensure reliable powerup. The device utilizes a resistive string architecture, with its inherent advantages of low Differential Non-Linearity (DNL) error and fast settling time. The device is specified over

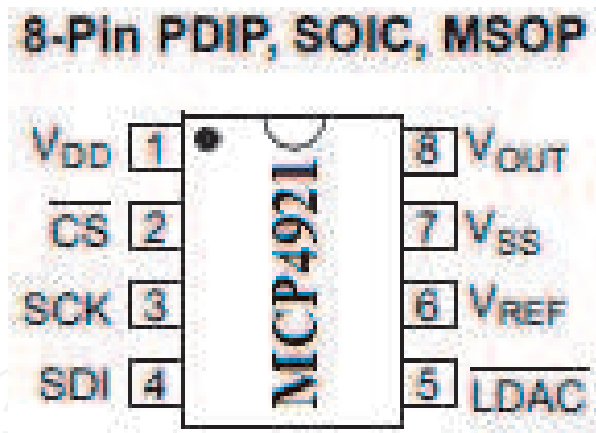


Figure 4.  
MCP4921 pin configuration.

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
0	BUF	GA	SHDN	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
bit 15								bit 0							

Figure 5.  
Write command register for MCP4921 (12-bit DAC).

the extended temperature range (+125°C). It provides high accuracy and low noise performance for consumer and industrial applications where calibration or compensation of signals (such as temperature, pressure and humidity) is required. The MCP4921 device is available in the PDIP, SOIC, MSOP and DFN packages. **Figure 4** shows the chip pin configuration. The MCP4921 device is designed to interface directly with the Serial Peripheral Interface (SPI) port, which is available on the PIC18F452 microcontroller and supports Mode 0,0 and Mode 1,1. Commands and data are sent to the device via the SDI pin, with data being clocked-in on the rising edge of SCK. The communication is unidirectional; this means the data cannot be read out of the MCP4921. The CS (chip select active low) pin must be held low for the duration of a write command. The write command consists of 16 bits and is used to configure the DAC’s control and data latches. Register shown in **Figure 5**, details the write command which is loaded into the input register that is used to configure and load the DAC register [6].

The write command is initiated by driving the CS pin low, followed by clocking the four Configuration bits and the 12 data bits into the SDI pin on the rising edge of SCK. The CS pin is then raised, causing the data to be latched into the DAC’s input register. The MCP4921 utilizes a double-buffered latch structure to allow the analog output to be synchronized with the LDAC pin, if desired. By bringing the LDAC pin down to a low state, the content stored in the DAC’s input register is transferred into the DAC’s output register (VOUT), and VOUT is updated. The write to the MCP4921 device is 16-bit words. Any clocks past the 16th clock will be ignored. The Most Significant 4 bits are Configuration bits. The remaining 12 bits are data bits. No data can be transferred into the device with CS high. This transfer will only occur if 16 clocks have been transferred into the device. If the rising edge of CS occurs prior to that, shifting of data into the input register will be aborted. The most four significant bits are defined as follows:

- bit 15 0 = Write to DAC register  
1 = Ignore this command
- bit 14 BUF: VREF Input Buffer Control bit  
1 = Buffered  
0 = Unbuffered

bit 13 **GA**: Output Gain Selection bit

1 = 1x ( $V_{OUT} = V_{REF} * D/4096$ )

0 = 2x ( $V_{OUT} = 2 * V_{REF} * D/4096$ )

bit 12 **SHDN**: Output Shutdown Control bit

1= Active mode operation.  $V_{OUT}$  is available.

0 = Shutdown the device. Analog output is not available.

$V_{OUT}$  pin is connected to 500 k $\Omega$  (typical).

### 3. Liquid crystal display (LCD)

This module is designed to display the value of the temperature detected by the temperature sensor and to guide the user in changing the parameters of the controller. The LCD is a  $16 \times 2$  alphanumeric display with the built-in Hitachi 44780 controller and LED backlighting. It works with an 8-bit data bus, which means it will require a total of 11 data lines. Three control lines (connected to port E) plus the 8 lines for the data bus (connected to port D) [7].

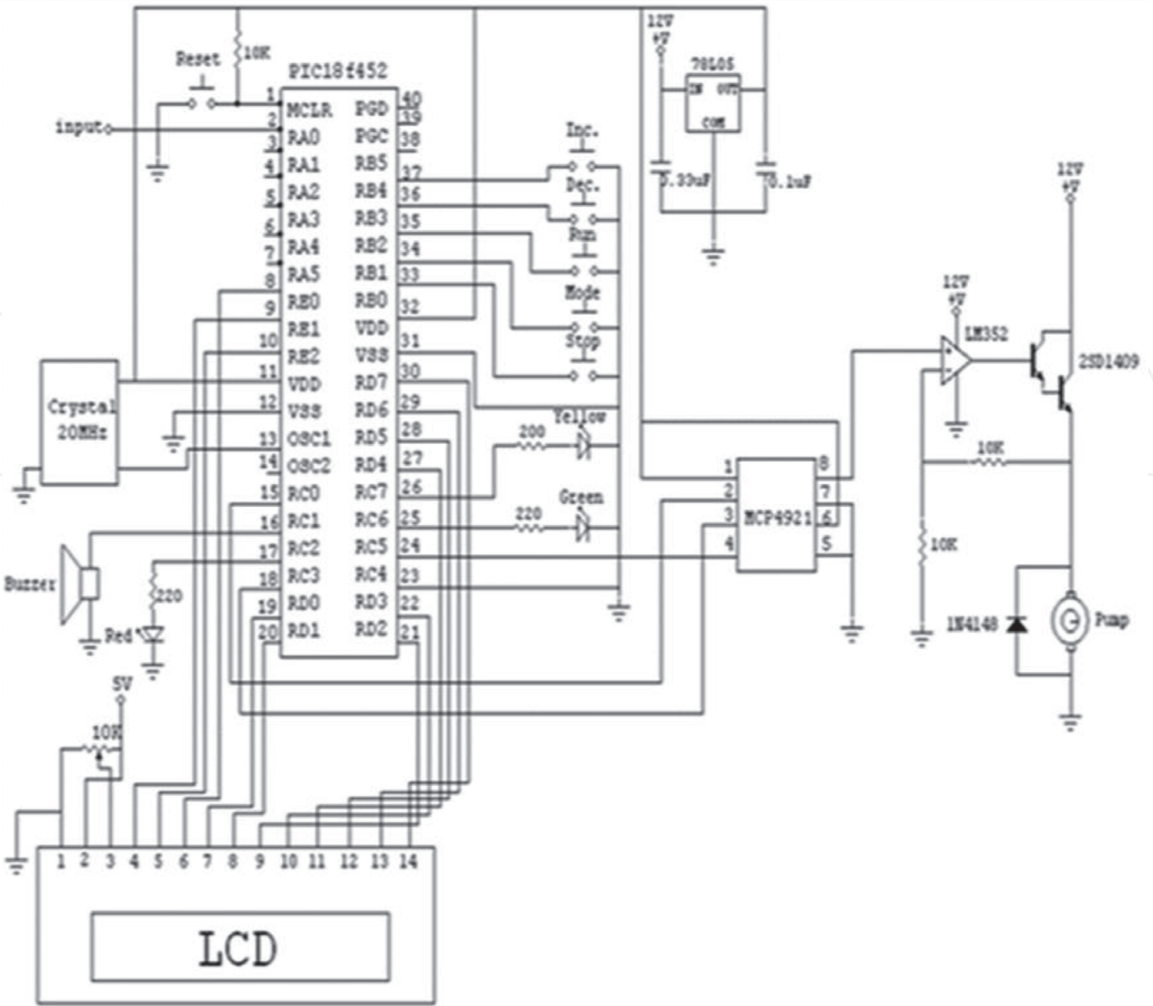
### 4. System design

The system is design around a stand-alone PIC18F452 controller, where the measured variable (MV) is read through channel0 (pin 2). The MV is subtracted from the set point automatically by the controller. The error is treated by the PIC PID and produces a digital control variable. This control variable is outputted through PIC serial data output pin (SDO pin 24) together with serial clock pin (SCK pin 18) to synchronize the conversion process. For the conversion to take place, the serial DAC chip select (CS) has to be pulled low. The CS is connected to pin RC0. The positive reference voltage is connected to +5 V (pin 6) and the negative reference voltage (pin 7) is tied to zero volt. The analog output is read through pin8 ( $V_{out}$ ). This voltage is small to drive an electric motor. This voltage is pulled up to +12 V through the non-inverting operational amplifier (LM358). The Darlington transistor 2SD1409 is used to bust the current. The motor is connected to the emitter follower so that the driving current is sufficient enough to drive the motor. Needless to say that the diode 1N4148 is used to protect the Darlington transistor against any spike due to the change of current. **Figure 6** shows the schematic of the system. The LCD is used to display the measured temperature. To manipulate the setting of different parameters, six push buttons are used as follows:

Six push buttons were used in the project to allow the user to change the setting and the controller parameters. Their functions are as follows:

1. Reset: To reset the microcontroller.
2. Stop: interrupt the program to allow the user to change the controller settings
3. Run: To run the program
4. Mode: To allow the user to change between setting modes.
5. Increment: To increment the controller variables by 1 or 0.1.
6. Decrement: To decrement the controller variables by 1 or 0.1.





**Figure 6.** System schematic circuit showing all the connection to the microcontroller, as well as the liquid crystal display and the final control element.

These switches are connected to PIC PORTB to allow the user to use the internal build-in pull up resistors to prevent floating instead of using external pull-down resistors. The reset has got a separate button connected to MCLR pin. A buzzer is used as an alarm to indicate that the temperature is more than what the user specifies. Three LEDs were used to show the user the status of the microcontroller program. The three colors green, yellow, and orange were used as follows:

1. Green: means that the PID controller is working properly.
2. Yellow: means that the program is interrupted by (STOP) push button.
3. Red: means that the alarm is triggered.

5. Software design

To implement the control program, three major routines are used; the main routine along with the timer and external interrupts. The program starts with the main routine which contains all the configurations of the external pins whether outputs or inputs. It also contains the configurations of timer and external interrupts, so when one of these interrupts is triggered, the microcontroller will stop its

current execution and perform another action. The trigger will be caused by either an overflow in timer register or a change on an external pin (RB0/INT0).

Because the time is a crucial element in digital control, the PID algorithm is controlled through a timer interrupt. This choice allows the user the ability to calculate the sampling time accurately. On the other hand, an external interrupt (INT0) is used to interrupt the program in order to allow the user a chance to modify the controller parameters. In the following we discuss in some details about the functions of each routine.

### 5.1 Main routine

#### 5.1.1 Routine function

This routine, as mentioned earlier, is dedicated to configure the direction of external pins as well as interrupt sources. It also allows the user to choose the measured variable (temperature, flow, level or others). The flow chart of this routine is shown in **Figure 7**.

#### 5.1.2 External ports configuration

First PORTA (pin RA0) is configured as an analog input channel0 and PORTB as input digital port which is connected to the push button switches; while all other pins are configured as outputs.

#### 5.1.3 LCD configuration

The configuration of the LCD was performed by separate software from Microchip called Application Maestro [8]. With the aid of this software, a configuration code was produced after modifying the module parameters. It was then incorporated into the project. Once incorporated, the LCD is configured and ready to work. One feature of using Application Maestro is its ability to use the prewritten code that this software provides to initialize or to write to the LCD.

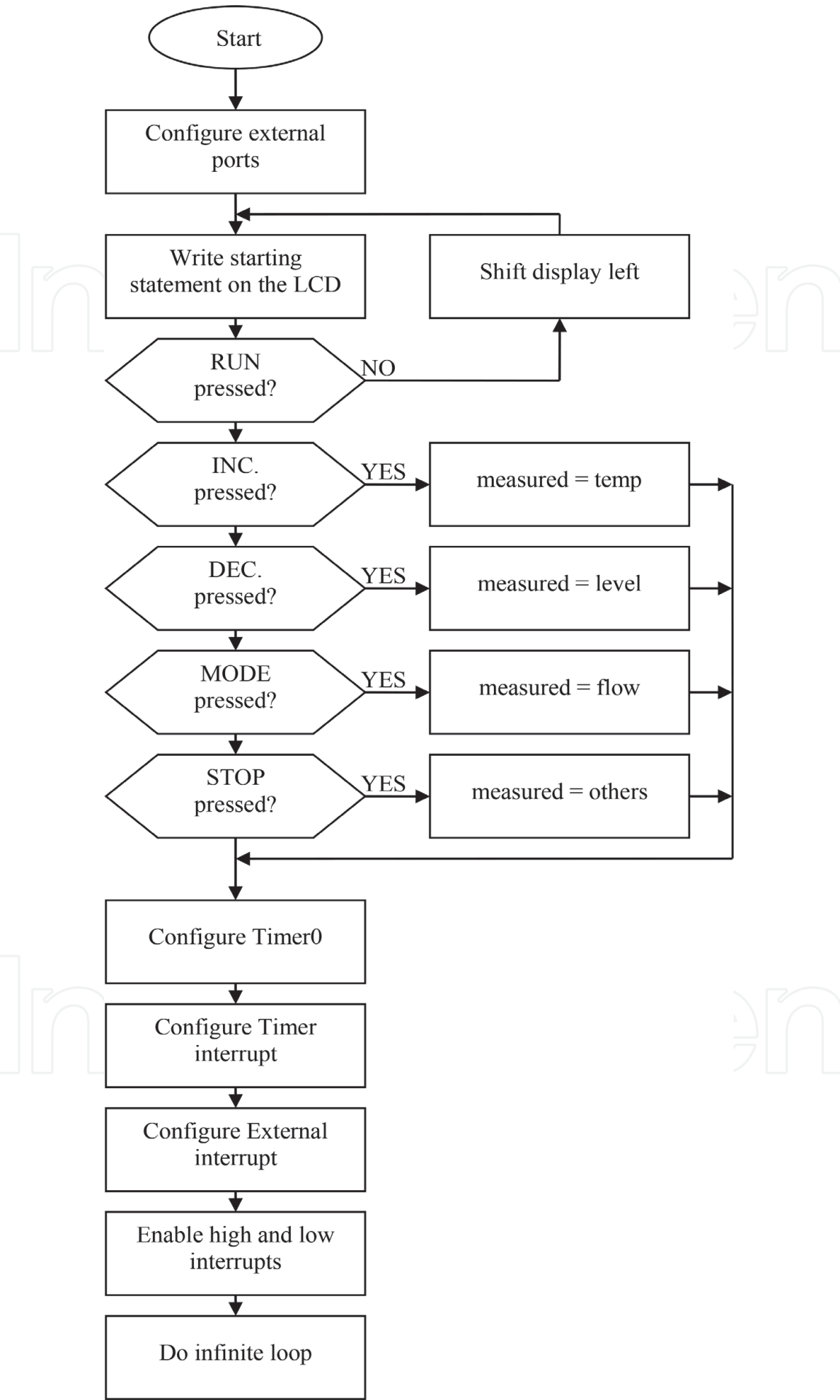
#### 5.1.4 Timer0 configuration

Timer0 can operate as a timer or as a counter. In Timer mode, the Timer0 module will increment with every instruction cycle (without prescaler). It is configured by setting a special function register called T0CON (timer0 control byte). This register is a readable and writable register that controls all the aspects of Timer0, including the prescale selection. In the design in hand, T0CON register is set to 0x85 (0b10000101) as shown below [9, 10].

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
1	0	0	0	0	1	0	1
bit 7							bit 0

This value will configure the timer0 as follows:

●Bit7	TMR0ON	= 1	: Timer0 is enabled
●Bit6	T08BIT	= 0	: Timer0 is configured as a 16-bit timer
●Bit5	T0CS	= 0	: Internal instruction cycle clock
●Bit4	T0SE	= 0	: This bit is used only with external clock
●Bit3	PSA	= 0	: Timer0 prescaler is assigned
●Bit2	T0PS2	= 1	: Bit2: T0PS2 =1: } 1:64 prescaler value
●Bit1	T0PS1	= 0	: Bit1: T0PS1 =0:
●Bit0	T0PS0	= 1	: Bit0: T0PS0 =1:



**Figure 7.**  
*Main routine.*

### 5.1.5 Interrupt configuration

There are ten registers which are used to control internal and external interrupt operations to accommodate a variety of interrupts [11]. In the project in hand, only two interrupts are required INT0 and timer0 interrupt. To do so, only three control registers are required. These registers are INTCON, INTCON2, and RCON. INTCON register contains various enable bits as well as several interrupt flags. RCON is the Reset Control register which contains flag bits that allow differentiation between the sources RESET. Timer0 interrupt is enabled by setting TMR0IE bit (<5>) while external interrupt is enabled by setting INT0IE (INTCON<4>). Note that the interrupt flags are reset before enabling the interrupt in order to avoid unwanted interruptions.

To start the interrupt, the global interrupt bit GIE/GIEH (INTCON<7>) must be set. If set, it enables all unmasked interrupts, so if more than one interrupt source is used (as in our case) the Interrupt Priority Enable bit IPEN (RCON<7>) must be set and the interrupt sources should be specified either as high or low priority interrupt. The interrupt priority bit TMR0IP (INTCON2<2>) is used to specify the interrupt priority for Timer0. This bit is reset so timer0 interrupt is set to low priority. On the other hand, no need to specify the priority of the external interrupt (INT0), because it is already set to high priority by default.

After configuring the interrupts, the program will enter an infinite loop until one of the interrupt sources is triggered.

## 5.2 Timer interrupt routine

### 5.2.1 Routine function

The main purpose of this routine is to calculate the controller output and send it to the DAC serially through the synchronous SPI module [12]. **Figure 8** shows the routine function.

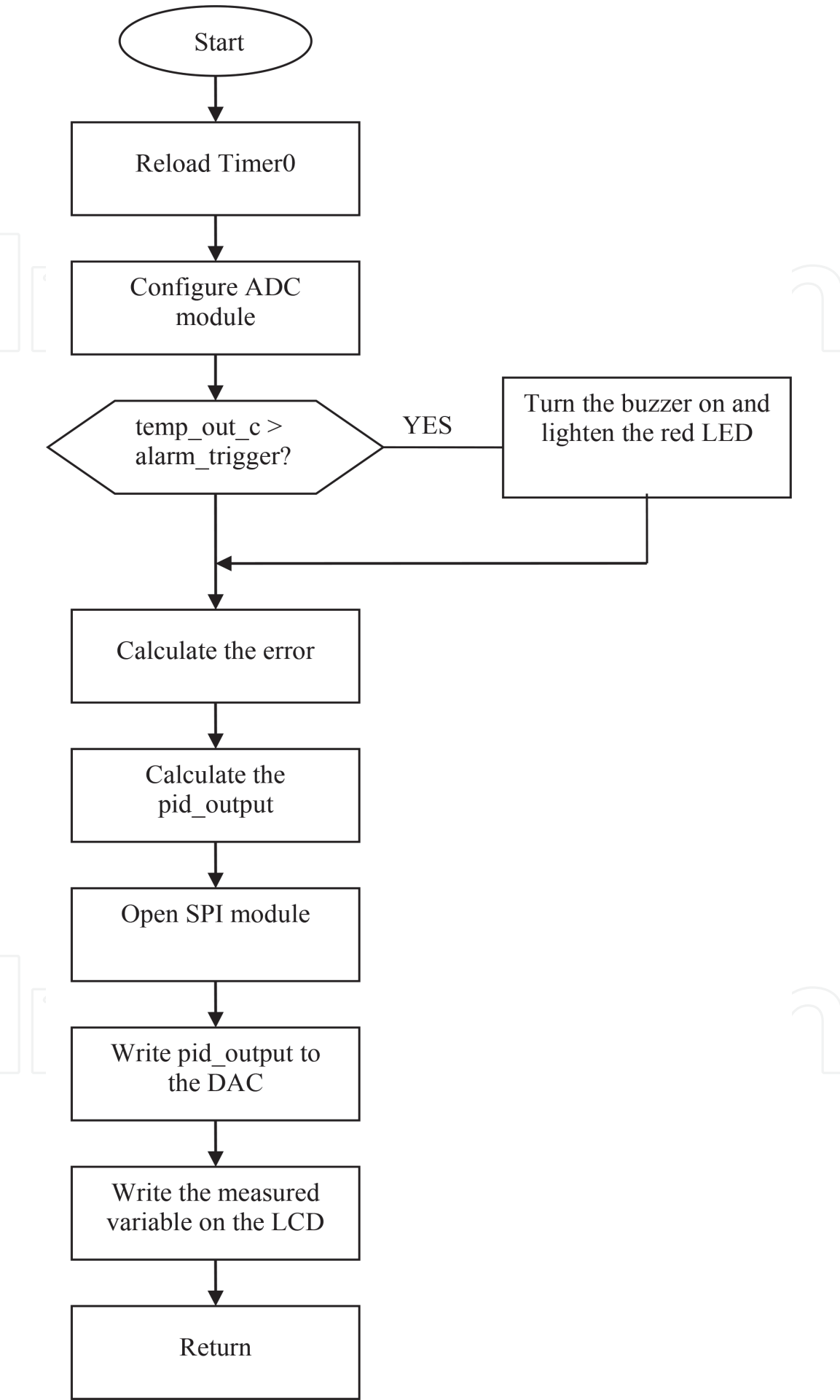
### 5.2.2 Timer reloading

Because of the importance of time in calculating the timed controller output, timer0 is used as an accurate hardware timer. The source clock of the timer is the crystal oscillator which is fed to the clock pin of Timer0 internally. The clock used is a 20 MHz derived from a stable crystal oscillator. This frequency is automatically divided by 4 because the controller machine cycle is 4 clocks to give a 5 MHz which is fed to the timer. The timer is exactly clocked every 0.2  $\mu$ s and takes 13107.2  $\mu$ s (16-bit mode) to count from zero to zero again. However, by loading the timer with a suitable value, a smaller time interval could be obtained. For example, by loading the timer with the value 4095 (0xFFF), the overflow would occur after 12288.2  $\mu$ s. Alternatively, the time period can be extended by using a prescaler as was done in the main routine. If a divide by 64 prescaler is selected, timer0 only overflows after 838.848 ms. This is obtained as follows:

$$\frac{5 \text{ MHz}}{64} = 78,125 \text{ Hz}$$

$$(78,125 \text{ Hz})^{-1} = 12.8 \text{ } \mu\text{s}$$

$$12.8 \text{ } \mu\text{s} \times 65,535 = 838.848 \text{ ms}$$



**Figure 8.**  
*Timer interrupt routine.*



This time period is less than one second, while a one second sampling time is required for the design in hand. To obtain a one second sampling time, the timer should count 78,125 pulses.

Because timer0 register is only 16 bit wide, it is only limited to count up to 65,535 pulses. The interruption is triggered several times to obtain one second timing, after which the controller computes the control action and sends it to the DAC. By using MPLAB simulator, it was found that 5362 cycles are required to calculate the controller output and send it to the DAC besides 51 extra cycles needed to reload the timer with time constant. If the interruption is required to repeat itself five times before calculating the controller output, one needs  $5362 + 51 \times 5 = 5617$  cycles (1.1234 ms). Thus, in order to get exactly one second sampling time, the timer register (TMR0) has to be reloaded with a value that interrupts the program every 998.8766 ms (1 s–1.1234 ms). The following shows how this value is obtained:

$$\text{No. of cycles for 1 s} = 5 \times 10^6 \text{ cycles}$$

$$\text{Therefore no. of cycles between interrupts} = 5 \times 10^6 - 5617 = 4,997,383 \text{ cycles}$$

By using a timer with 64 prescaler:

$$\text{No. of counts} = \frac{4,997,383}{64} = 78,084.10938$$

When we repeat the interrupt for 5 times:

$$\text{No. of counts} = \frac{78,084.10938}{5} = 15,616.82188$$

But because the timer counts in ascending order (from 0x0000 to 0xFFFF):

$$\text{Reload value} = 65,535 - 15616.82188 = 49918.17812$$

```
repeat=5*sampling_time;
term_1=((repeat)*51.0+5362.0);
term_2=(5000000.0*sampling_time-term_1)/64.0;
term_3=term_2/repeat;
cycle=65535-term_3;
```

However, the timer register accepts only integer numbers, thus the final value that should be added to the timer register is 49918. Because we omitted the numbers after the decimal point, our error will be  $\pm 1$  count which is equal to 64 cycles. Therefore, our error in calculating the sampling time will be:

$$\text{Timer error} = 64 \times 0.2 \text{ } \mu\text{s} = 12.8 \text{ } \mu\text{s}$$

This calculation is for getting 1 s sampling time. To expand the calculation in order to enable the user to change the sampling time, one defines two integer variables (repeat and cycle). The first variable repeat is to determine how many times we need to repeat the interrupt, while the second one cycle is the final value that should be added to the timer register. The following pseudo code shows the general formula used to reload the timer register.

### 5.2.3 Analog to digital converter module

The ADC module normally operates at 10-bits resolution, giving output digital values 0–1024 [13]. It needs a reference voltage to set the maximum and minimum

values for the input conversion. This reference can be provided internally as  $V_{dd}$  and  $V_{ss}$  (supply values) or externally through  $V_{ref+}$  and  $V_{ref-}$  pins. To configure this module, OpenADC function from Microchip C library is used. This function performs a bitwise AND operation (“&”) between its arguments which are defined in the file adc.h. The parameters of this function along with their meaning of each argument are discussed below [1]

```
OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_8ANA_0REF,
ADC_CH0 & ADC_INT_OFF);
```

- ADC\_FOSC\_32: FOSC/32.

A clock divider to allow the minimum specified conversion time (about 20  $\mu$ s). A 32 prescaler was chosen because the clock source is 20 MHz

- ADC\_RIGHT\_JUST: Right justified.

Because the ADRES register pair (where the converted values are loaded) is 16-bit wide. But the ADC is only 10bit wide. The ADC module could either be configured as right or left justified. In this project, right justified is chosen as shown in **Figure 9**. This sets the 6 most significant bits of register ADRES to zeros.

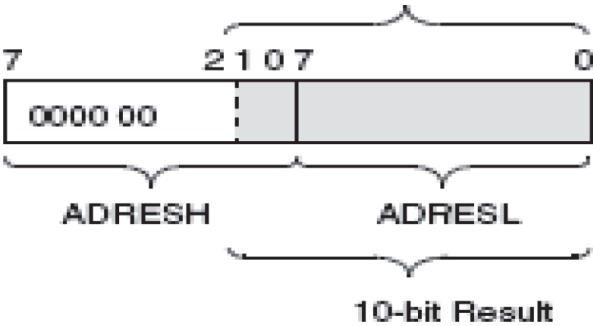
- ADC\_8ANA\_0REF:  $V_{REF+} = V_{DD}$ ,  $V_{REF-} = V_{SS}$

The supply values are chosen as the voltage references to the ADC.

- ADC\_CH0: Channel0 (AN0) is selected
- ADC\_INT\_OFF: Interrupts of ADC interrupts are disabled.
- Once the A/D conversion is completed, the result is stored in an integer variable called result. After reading the analog value by the ADC module, the result will be compared with the variable alarm-trigger which was previously specified. If the result is greater than this value, the microcontroller triggers the buzzer and lights the red LED.

5.2.4 Controller calculation

Due to the limitation in the microcontroller’s memory, the PID equation is divided into three terms (term\_1, term\_2, and term\_3) and after calculating each term separately, they are added together along with the previous output to give the controller



**Figure 9.**  
*Choosing right justified for data input.*

output which will be sent to the DAC. The following code shows how to calculate the controller output

```
term_1 = KC*(error-last_error);  
term_2 = KC*(sampling_time/TI)*error;  
term_3 = KC*(TD/sampling_time)*(error-(2*last_error)+prev_error);  
pid_output_v = term_1 + term_2 + term_3+pid_output_v;
```

5.2.5 SPI module

To send the control variable to the final control element, the serial DAC, which is interfaced to the Serial Peripheral Interface (SPI) port, is used. The SPI is initiated using Microchip C library called OpenSPI. This function also performs a bitwise AND operation between its arguments which are defined in the file SPI.h according to the following formula.

- SPI\_FOSC\_16: Master mode and the clock = FOSC/16
- MODE\_00: Mode 0,0 (change takes place on the rising edge)
- SMPEND: Input data sample at end of data out

```
OpenSPI(SPI_FOSC_16, MODE_00, SMPEND);
```

After configuring the module, it is time to write a command to the DAC in order to convert it into analog signal. The write command is initiated by driving the CS pin low, followed by clocking the four configuration bits and the 12 data bits into the SDI pin on the rising edge of SCK. The CS pin is then raised, causing the data to be latched into the DAC’s input registers and when the LDAC pin is pulled down through RC1, the values held in the DAC’s input registers are transferred into the DAC’s output registers to provide the analog signal. It is important to mention here that we wrote the write command in two steps (as shown in the following code) because the SPI module send only 8 bit at a time.

```
LATCbits.LATC0 = 0; // Chip Select is set  
WriteSPI(pid_output_16_high); //4MSB as command + 4LSB as data  
WriteSPI(pid_output_16_low); // 8 bit data  
LATCbits.LATC0 = 1; // Chip Select is reset  
LATCbits.LATC1=0; // enable LDAC for data output to DAC  
LATCbits.LATC1=1; // disable LDAC  
CloseSPI();
```

5.2.6 Writing on the LCD

To write characters to the LCD, required prewritten functions are provided by Application Maestro. Some of These functions are listed in following table:

XLCDInit()	It is used to initialize the LCD module according to the Application Maestro options
XLCDPut(data)	It sends the clocking signal and data to be displayed to the LCD

XLCDL1home()	Points to the first address location of line one of the LCD
XLCDL2home()	Points to the first address location of line two of the LCD
XLCDClear()	Clears the DDRAM content of the LCD and points to the 00 address location
XLCDPutRomString(addr)	Displays String in Program memory
XLCDPutRamString(addr)	Displays String in Data memory
XLCDCommand(Command)	It sends clocking signal and instructions to the LCD

For numbers to be displayed, they are first converted into strings (characters) before being sent to the LCD, since the latter only accepts strings. To do so a C function called `sprintf` is called upon. This function saves the number in an array after converting it into string. The subroutine to do so is shown below [4]:

```
sprintf (buf,"%d",temp_set);
XLCDPutRamString(buf);
```

5.3 External interrupt routine

5.3.1 Routine function

The main function of this routine is to allow the user to change the controller parameters. The routine is initiated by pressing the push button (STOP) which is connected to the external interrupt pin (RB0/INT0). Once initiated, the user is able to change all the parameters of the controller ( $K_C$ ,  $\tau_I$ ,  $\tau_D$ , sampling time, alarm trigger and sensitivity) by using three push buttons (MODE, INCREMENT and DECREMENT [11].

To determine which action the microcontroller should take if any push button is pressed, we defined two integer variables (`present_button` and `present_mode`) to be used as statuses. That is, each bit of them has specific meaning as described below:

• `present_button`

Np	_____	dec	inc	Mod
bit15				bit0

- bit 15      **Np**: set if there are no push buttons pressed
- bit 14-3    Unimplemented
- bit 2        **dec**: set if the DECREMENT push button pressed
- bit 1        **inc**: set if the INCREMENT push button pressed
- bit 0        **mod**: set if the MODE push button pressed

• `present_mode`

_____	Srt	Spt	KC	TI	TD	Stm	Sen	Alm	Tun	Dp	Tp	Kp	_____	_____	_____
bit15															bit0

- bit 15      Unimplemented
- bit 14      **Srt**: set in the starting mode

- bit 13     **Spt:** set in the set point mode
- bit 12     **KC:** set in the controller gain mode
- bit 11     **TI:** set in the integral time mode
- bit 10     **TD:** set in the derivative time mode
- bit 9       **Stm:** set in the sampling time mode
- bit 8       **Sen:** set in the sensitivity mode
- bit 7       **Alm:** set in the alarm mode
- bit 6       **Tun:** set in the tuning mode
- bit 5       **Dp:** set in the process delay mode
- bit 4       **Tp:** set in the process time constant mode
- bit 3       **Kp:** set in the process gain mode
- bit 2-0     Unimplemented

Initially, before pressing any push button, present\_button variable is loaded with 0x8000 (no push button pressed), and present\_mode with 0x2000 (starting mode). Then if any push button is pressed, the corresponding bit of that push button will be set, giving a specific value of present\_button which indicates the push button that was pressed by the user. So by performing a bitwise OR operation between the two variables (present\_button and present\_mode) we will come up with a number indicates the push button pressed and the present mode and based on that number we can decide the proper action to be taken by the microcontroller. The following code shows how to perform the OR operation after checking which of the push buttons was pressed. Beside changing the controller variables, this routine has another feature, it gives the user preliminary values of the controller parameters after entering the process variables. The result is derived based on Cohen-Koon tuning method. However, this feature is impractical if the sampling time is big [14].

```
if (mode_pin==0){
    Delay10KTCYx(70);
    present_button=mode_pushed;
}
else if (inc_pin==0){
    Delay10KTCYx(70);
    present_button=inc_pushed;
}
else if (dec_pin==0){
    Delay10KTCYx(70);
    present_button=dec_pushed;
}
else if(end_pin==0)
    return;
action= present_mode | present_button;
```

6. Testing and verification

To test the system, a first order system given by the equation below was used. To run the control action, the system was converted into a difference equation given by Eq. (10).

$$G_p = \frac{10}{5S + 1} \tag{9}$$



The process transfer function is first order, thus the discrete transfer function obtained using Zero-Order Hold will be:

$$HG(z) = \frac{az^{-1}}{1 - bz^{-1}} \quad (10)$$

where:

$$a = k_p \left[ 1 - \exp\left(\frac{-T_s}{\tau_p}\right) \right]$$

$$b = \exp\left(\frac{-T_s}{\tau_p}\right)$$

If  $k_p = 10$ ,  $\tau_p = 5$ , and  $T_s = 1$ , The discrete transfer function will be:

$$HG(z) = \frac{1.813z^{-1}}{1 - 0.8187z^{-1}} \quad (11)$$

$$\Rightarrow (1 - 0.8187z^{-1})Y(z) = 1.813z^{-1}C(z)$$

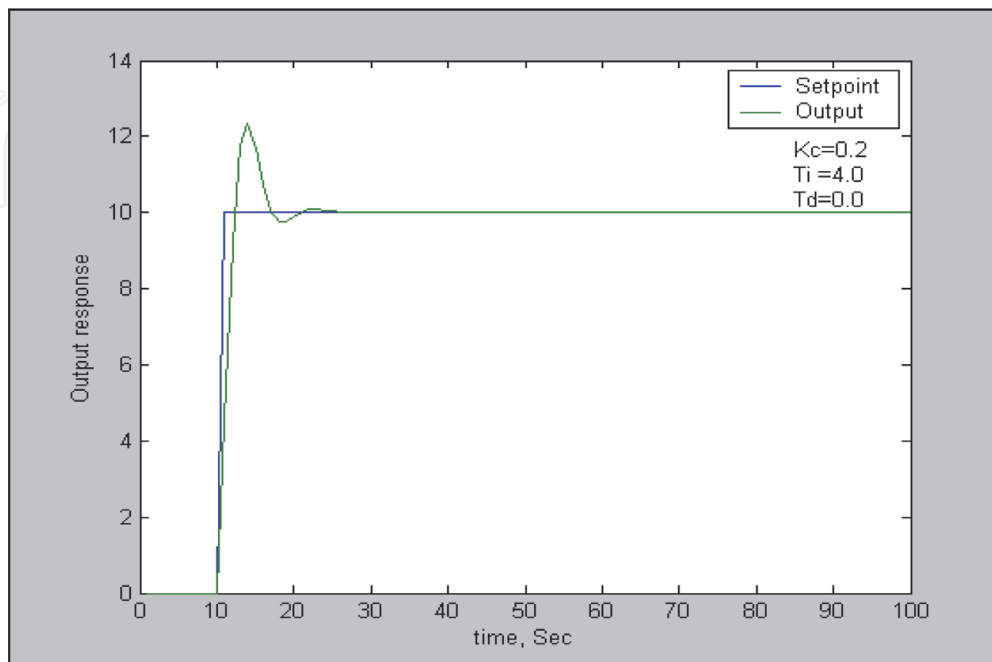
$$Y(z) - 0.8187z^{-1}Y(z) = 1.813z^{-1}C(z)$$

$$y_n - 0.8187y_{n-1} = 1.813c_{n-1}$$

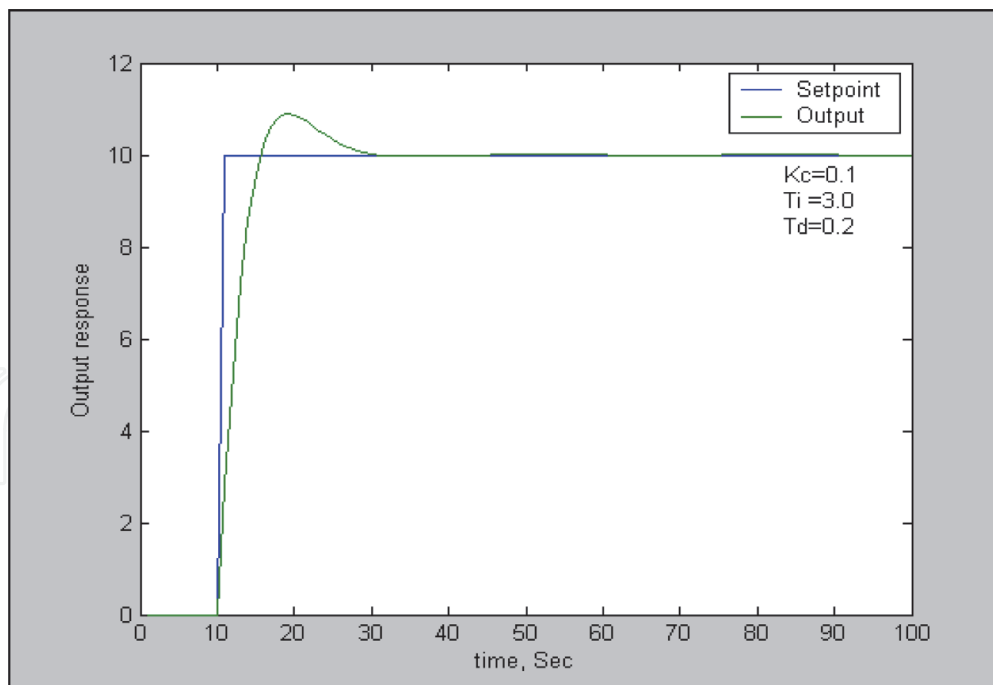
Therefore the difference equation of the output is:

$$y_n = 0.8187y_{n-1} + 1.813c_{n-1} \quad (12)$$

After getting the difference equation, the control scheme was tested and the output of **Figure 10** was obtained with the parameters set to:  $K_c = 0.2$ ,  $T_i = 4.0$  and  $T_d = 0.0$ . The parameters were then changed to:  $K_c = 0.1$ ,  $T_i = 3.0$  and  $T_d = 0.2$ . The response is shown in **Figure 11**.



**Figure 10.**  
Controller response with  $K_c = 0.2$ ,  $T_i = 4.0$  and  $T_d = 0$ .



**Figure 11.**  
 Controller response with  $K_c = 0.1$ ,  $T_i = 3.0$  and  $T_d = 0.2$ .

## 7. Conclusion


By referring to the previous graphs, it could be concluded that the response tracks the set point as expected. In addition, the increase in controller gain ( $K_c$ ) does speed up the response but at the expense of the overshoot. Based on these results, it could be concluded that the three-term controller is working according to plan. Because of the flexibility of the microcontroller and its programming, any control scheme could be developed and implemented in the manner as described in this chapter. Not only that, the scheme could be transferred to several high range microcontrollers from the same company such as 16 or 32 bits with the use of the benefits those types of controllers offer.

## Author details

Mostefa Ghassoul  
 Chemical Engineering, University of Bahrain, Bahrain

\*Address all correspondence to: [mghassoul@uob.edu.bh](mailto:mghassoul@uob.edu.bh)

## IntechOpen

© 2020 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

## References

- [1] Ogunnaike BA, Ray WH. Process Dynamics, Modeling, and Control. 1st ed. USA: Oxford University Press; 1994
- [2] Stephanopoulos G. Chemical Process Control: An Introduction to Theory and Practice. USA: Prentice Hall; 1983
- [3] Seborg DE, Edgar TF, Mellichamp DA, Doyle FJ. Process Dynamics and Control. 4th ed. New York, USA: John Wiley and Son; 2017. pp. 115-117
- [4] Phillips CL, Nagle HT, Chakraborty A. Digital Control System Analysis and Design. UK: Pearson; 2015. pp. 279-335
- [5] Microchip PIC18FXX2 Data Sheet “High-Performance, Enhanced Flash Microcontrollers” Microchip 2006 (DS39564C)
- [6] Microchip MCP4921/4922 datasheet “12-Bit DAC with SPI™ Interface” Microchip 2007
- [7] HD44780 LCD starter guide. 2001
- [8] Microchip application maestro software user’s guide. 2003
- [9] Microchip PIC18FXX2 Data Sheet “High-Performance, Enhanced Flash Microcontrollers: Timers” Microchip 2006 (DS39564C). pp. 103-115
- [10] Bates M. PIC Microcontrollers: An Introduction to Microelectronics. Newnes; 2011
- [11] Microchip PIC18FXX2 Data Sheet “High-Performance, Enhanced Flash Microcontrollers: External interrupts” Microchip 2006 (DS39564C). pp. 73-85
- [12] Microchip PIC18FXX2 Data Sheet “High-Performance, Enhanced Flash Microcontrollers: Timer0 interrupt” Microchip 2006 (DS39564C). p. 85
- [13] Microchip PIC18FXX2 Data Sheet “High-Performance, Enhanced Flash Microcontrollers: 10 bit ADC” Microchip 2006 (DS39564C). Chandler, Arizona USA; pp. 181-188
- [14] Deitel PJ, Deitel HM. C How to Program. 6th ed. New Jersey USA: Pearson Prentice Hall; 2010