DEVELOPMENT OF REAL TIME DIGITAL CONTROLLER FOR A LIQUID LEVEL SYSTEM USING ATMEGA32 MICROCONTROLLER

A REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY
IN ELECTRICAL ENGINEERING

By

AMRUTA PATRA

107EE042

DEPARTMENT OF ELECTRICAL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA
MAY 2011

DEVELOPMENT OF REAL TIME DIGITAL CONTROLLER FOR A LIQUID LEVEL SYSTEM USING ATMEGA32 MICROCONTROLLER

A REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY
IN ELECTRICAL ENGINEERING

By

AMRUTA PATRA

107EE042

Under the guidance of
Prof. Bidyadhar Subudhi

DEPARTMENT OF ELECTRICAL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA
MAY 2011

Certificate

This is to certify that the project titled **“**DEVELOPMENT OF REAL TIME DIGITAL CONTROLLER FOR A LIQUID LEVEL SYSTEM USING ATMEGA32 MICROCONTROLLER**”** is a bonafide record of work done by Amruta Patra in partial fulfillment of the requirements for the award of Bachelor of Technology degree in electrical engineering at the National Institute of Technology, Rourkela, under my supervision and guidance.

To the best of my knowledge the matter embodied in this project had not been submitted to any other Institute / University for the award of any Degree or Diploma.

Date : - 10/5/2011

Prof. Bidyadhar Subudhi
Head of the Department
Department of Electrical Engineering
National Institute of Technology
Rourkela

Abstract

This project describes how to implement a digital controller algorithm like PI controller in real time using a simple yet effective digital control device-ATMEGA32 microcontroller for controlling a prototype model of a liquid level control system. A liquid level sensor(rotary potentiometer) detects the present level of the liquid in the tank in terms of the voltage across the potentiometer and feeds it to the microcontroller and the control action generated by the microcontroller is amplified through a suitable amplifier that actuates the actuator(the pump) and finally controls the flow output of the pump. The operator has to set the desired level in the microcontroller and accordingly the feedback control in the real time will get operated for achieving the desired level.

In the present project work , a dicretized model of the liquid level system is developed using PI controller and is simulated in MATLAB Simulink so as to observe the nature of the PI controlled system output. It is also compared with the uncontrolled liquid level system model simulation output so as to see the advantage of using a PI controller. Also, an attempt has been made for developing a dummy representation of the actual prototype model for automatic controlling the liquid level so as to know the proper functioning of the algorithm.

The hardware is set up and the devices like the microcontroller, D/A, power amplifier are interfaced with each other .The devices are also tested for their proper functioning. The PI controller is developed in discrete domain and its parameters are determined using the open loop Ziegler Nichols tuning method. The dicretized control algorithm is then implemented in the microcontroller using C language for coding. The nature of the controller is observed and results are shown . Also the experimental results are compared with the simulated results to show the similarity and accuracy of the controller.

This liquid level study will be useful for several industrial and household applications like boiler level control , household supplies and many more.

Keywords: Control, Proportional and Integral(PI), microcontroller, analog, digital, sensor

Acknowledgement

I am indebted to my mentor, Prof. Bidyadhar Subudhi, Head of Department of Electrical Engineering, for giving me an opportunity to work under his guidance. Like a true mentor, he motivated and inspired me throughout the entire duration of my work.

I am also greatful to Mr.Ayaskanta Swain of Electronics and Communication Engineering department and Mr.Raja Rout of Electrical Engineering department for assisting me and guiding me throughout the project and furthered the project till this extent. I also extend my thanks to the supportive staff for providing me all the necessary facilities to accomplish this project.

Last but not the least, I express my profound gratitude to the Almighty and my parents for their blessings and support without which this task could have never been accomplished.

Amruta Patra

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

The liquid level digital control system automatically maintains the desired level of water in a tank/container, i.e., it switches on the pump when the water level in the tank/container goes below a predetermined maximum level and switches it off as soon as the water level reaches the pre-determined maximum level in the tank/container to prevent it from overflowing, thus maintains the water level at a fixed level always .The user has the flexibility to decide by himself the water level set-points for operations of pump. It ensures no overflows there by saves electricity and water. Moreover the system consumes very little energy and hence is ideal for continuous operation.

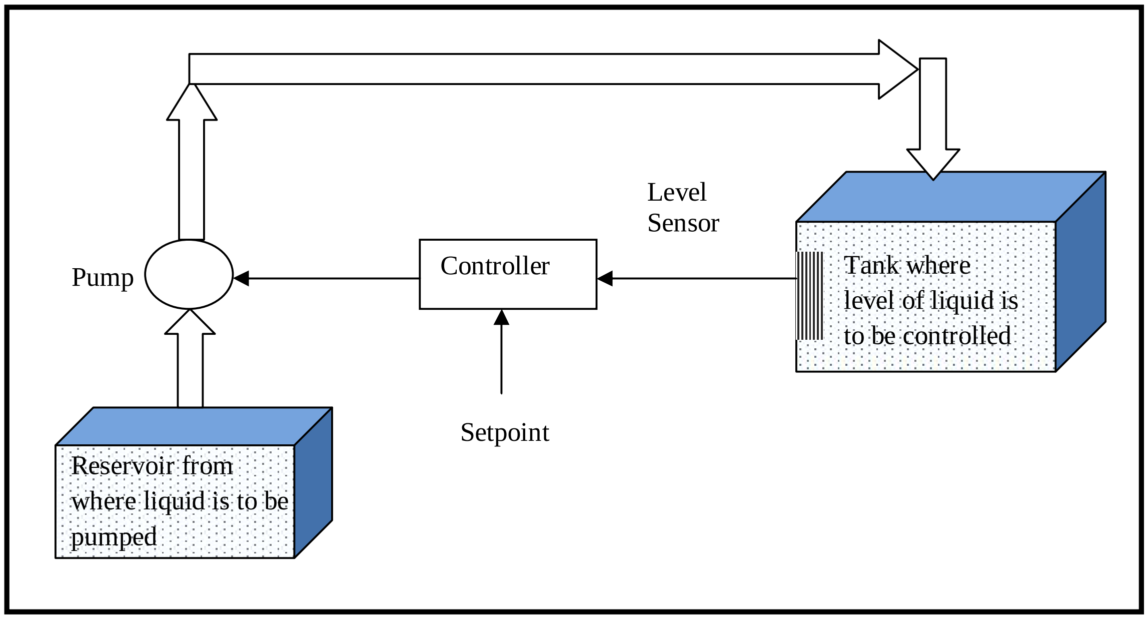

Level
Sensor
Pump
Controller
Tank where level of liquid is to be controlled
Setpoint
Reservoir from where liquid is to be pumped

Fig 1.1 : A Typical liquid level control system

1.1 MOTIVATION

In the modern world of today, automation is encompassing nearly every walk of life. Automation solutions are more accurate, reliable and flexible and so have replaced human efforts right from agriculture to space technologies, may it be for monitoring a process, recording its parameters, analyzing the trend of output or controlling the desired parameter. These days plant automation is the necessity of the manufacturing industries to survive in the globally competitive markets. For any process to be automated , we need most essentially a real time automatic controller.

Most of the process industries involve liquid at some point or the other of its production process. So it is highly essential for accurate liquid level measurement and control at a desired level in most process industries like -

- Food Processing, Dairy and Beverages Industry.
- Chemical production, processing and storage Plants
- Petroleum and Petro chemical Industry.
- Water and Waste Water Treatment Plants.
- Pollution control plants.
- Textiles, Pulp and Paper Industries.
- Energy and Power generation Plants.
- Shipping and Marine Industry.

And many more[9],[10] and so comes the need for a liquid level controller.

Also with population blooming each day, water scarcity is a global concern, which needs to be immediately taken care of else drastic circumstances would have to be faced since we all know life without water is impossible. With plenty of water available the problem is not with its scarcity but its undue wastage. Normally in the houses, water is first stored in an underground tank (UGT) and from the UGT, water is pumped up to the overhead tank (OHT) located on the roof. People generally switch on the pump when their taps get dry and switch off the pump when the overhead tank starts overflowing. This results in the unnecessary wastage of water by tank overflow and sometimes non-availability of water in the case of emergency due to drying of overhead tank. So come the need of an automatic water level controller.

Earlier humans used to do control manually but this always involved errors. So these controllers had to be automated. With the advent of digital electronics and hence invention of microprocessors and microcontrollers, came the concept of automation. The controllers developed, could be implemented in real time with the help of these microprocessors or microcontrollers. Hence we could control the level of a liquid at a desired set point with the help of a proper controller using an embedded device like microprocessor or microcontroller to implement the control algorithm.

1.2 WORK SUMMARY

The classical controller - PI controller is used for this liquid level control system as PI is the most widely used controller in process industries due to its simple structure, assured acceptable performance and their tuning is well known among all industrial operators. ATMEGA32 microcontroller is chosen for implementing this algorithm in real time as it has inbuilt ADC, timer/counter and so simplifies external circuitry and it is 10 times faster than conventional microcontrollers like 8051. At first the system and the controller are modeled and simulated to get an idea of their behavior. Then the set up is made, also the circuit connections are made. The set up is interfaced with the circuitry and then the control algorithm is implemented in real time with the help of the microcontroller. Hence we can list down our objectives as follows :

- ❖ Modeling the Liquid level system
- ❖ Discretization of the liquid level system model
- ❖ Simulation of the system without and with the PI controller in MATLAB
- ❖ Analysis of the controller performance
- ❖ Development of the prototype model
    - Planning for hardware implementation
    - Constructing the liquid level system

    - Selection of hardware devices microcontroller, D/A, Power Amplifier, Pump, Sensor
    - Testing and interfacing the hardware devices
- ❖ Development of controller for maintaining a desired liquid level
- ❖ Implementation of the control algorithm in the model with the help of the microcontroller
- ❖ Analysing experimental Results

1.3 REPORT ORGANISATION

Chapter 1 deals with the introduction to this project narrating the motivation, objectives, summary of the project and the description of the liquid level system. Chapter 2 deals with the modeling and discretizing the liquid level plant and its simulation results of MATLAB Simulink with and without a PI controller. Chapter 3 describes the components used in the project and explains(in italics) the need and method for using them. Chapter 4 narrates the procedure and results of testing and interfacing the devices. Chapter 5 deals with development of the PI controller in discrete domain and determining its coefficients by Ziegler Nichols open loop test tuning method. The discrete domain PI control algorithm is also shown and a simple code is given for implementing it. Chapter 6 deals with the development of the final set up of the controller and then implementation of the control algorithm in real time, programming ATMEGA32 in C. Chapter 7 narrates the and explains the experimental results obtained and compares it with the simulated results and gives the conclusion of the project . At the end there is the list of references followed by the list of prices of components used in the project.

1.4 LIQUID LEVEL SYSTEM DESCRIPTION

The liquid level system consists of two water tanks, a water pump, a liquid level sensor, a microcontroller with inbuilt ADC , a D/A converter and a power amplifier. The schematic block diagram of the system is as follows :

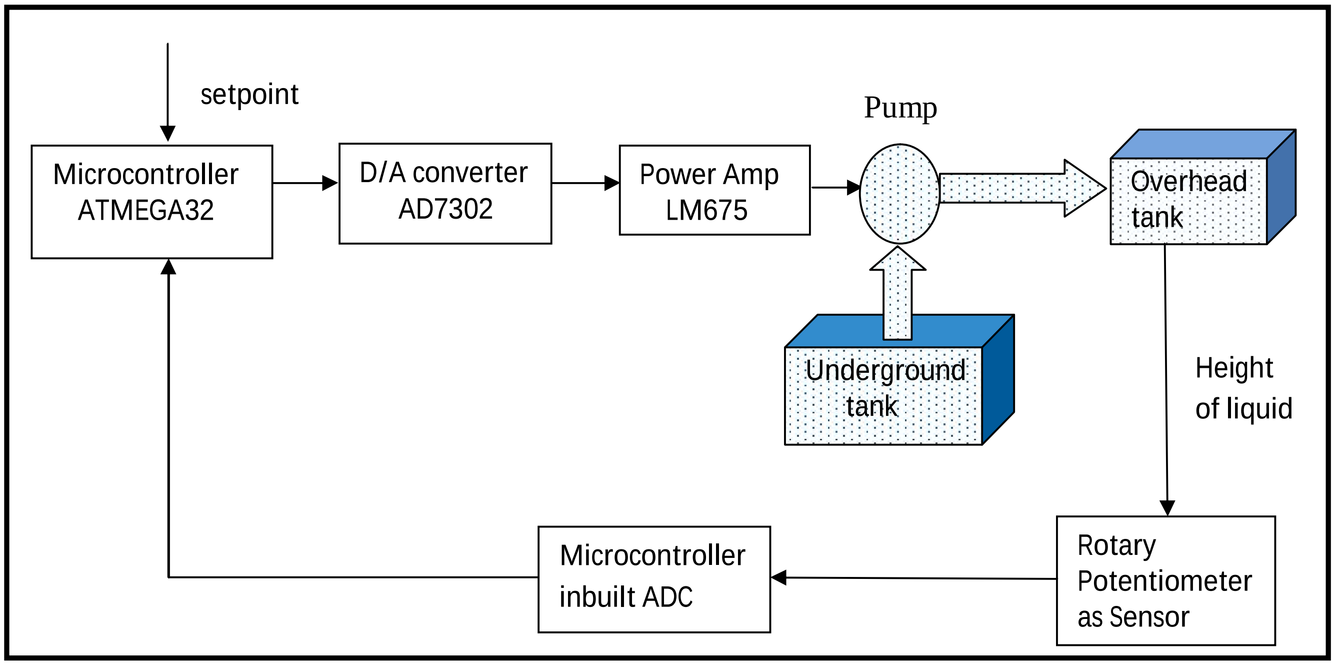


Fig 1.2 : Schematic of the liquid level system

The description of the liquid level system components are as follows –

Water tank - This is the tank inside which the level of the liquid has to be controlled. Water is pumped to the tank from a pipe coming down into the tank from above and a rotary potentiometer type liquid level sensor measures the height of the water inside the tank. The microcontroller controls the pump so that the liquid is stopped at the desired level. The tank used in this project is a plastic container with measurements 18cm × 10cm × 18cm.

Water pump- The pump is a small 12V water pump which draws around 3A current when it operates at the full-scale voltage.

Level sensor - A rotary potentiometer type level sensor is used in this project. The sensor consists of a floating arm connected to the sliding arm of a rotary potentiometer. The level of the floating arm, and hence the resistance of the rotary potentiometer changes as the liquid level changes inside the tank. A voltage is applied across this potentiometer and the change of voltage is measured across the arm of the potentiometer, which is the source of analog input for the microcontroller.

<u>Microcontroller</u> **-** An ATMEGA32 type microcontroller is used in the project as the digital controller.

<u>D/A converter</u> **-** An 8-bit AD7302 type D/A converter is used in the project.

<u>Power amplifier</u> **-** The output power of the D/A converter is in the range of few hundred milliwatts, which is not capable of driving the water pump. So an LM675 type power amplifier is used in this project to increase the power output of the D/A converter so as to be capable of driving the pump. The LM675 can provide around 30W of power.[1],[2]

CHAPTER 2

SYSTEM MODELING, DISCRETIZING AND SIMULATION

CHAPTER 2

SYSTEM MODELING, DISCRETIZING AND SIMULATION

2.1 SYSTEM MODELING [1]

The system is modeled as a first-order system. The tank acts as a fluid capacitor where fluid enters the tank(behaving as charged particles entering a capacitor) and leaves the tank. According to mass balance relation between the incoming fluid and outgoing fluid,

$$Q_{in} = Q + Q_{out} \qquad (2.1)$$

where Q_{in} is the flow rate of water coming into the tank, Q the net rate of water storage in the tank, and Q_{out} is the flow rate of water going out from the tank. If A is the cross-sectional area of the tank, and h is the height of water inside the tank at any instant, Equation (2.1) can be written as

$$Q_{in} = A\frac{dh}{dt} + Q_{out} \qquad (2.2)$$

where $\frac{dh}{dt}$ is the rate of change of height of water inside the tank.

The net flow rate (Q_{out}) of water coming out of the tank depends on the discharge coefficient of the tank, the height of the liquid at any instant inside the tank (h), the gravitational constant (g), and the area of the tank outlet (a),and can be expressed as

$$Q_{out} = C_d\, a\sqrt{(2gh)} \qquad (2.3)$$

where C_d is the discharge coefficient of the tank outlet, a is the area of the tank outlet, and g is the gravitational constant (9.8m/s^2).

From (2.2) and (2.3) we obtain

$$Q_{in} = A\frac{dh}{dt} + C_d\, a\sqrt{(2gh)} \qquad (2.4)$$

As we can see from equation (2.4) , it is a nonlinear relationship between the inflow rate (Q_{in}) and the height of the water inside the tank(h).This equation can be linearized for small perturbations about an operating point.

When the input flow rate Q_{in} becomes constant i.e. water comes in a constant rate, the flow rate of water coming out from the tank through the orifice would reach a steady-state value $Q_{out} = Q_0$, and the height of the water h becomes a constant value h_0, and we can write

$$Q_0 = C_d\, a\sqrt{(2gh_0)} \qquad (2.5)$$

If we now consider a small perturbation (δQ_{in})in input flow rate around the steady-state value Q_0 , we obtain

$$\delta Q_{in} = Q_{in} - Q_0 \qquad (2.6)$$

and, as a result, the fluid level h will be perturbed around the steady-state value h_0 by

$$\delta h = h - h_0 \qquad (2.7)$$

Now, substituting (2.6) and (2.7) into (2.4) we obtain

$$A\frac{d\delta h}{dt} + C_d a\sqrt{2(\delta h + h_0)} = \delta Q_{\text{in}} + Q_0. \qquad (2.8)$$

Equation (2.8) can be linearized by using the Taylor series and all terms are neglected except the first term. From Taylor series,

$$f(x) = f(x_0) + \left.\frac{df}{dx}\right|_{x=x_0}\frac{(x-x_0)}{1!} + \left.\frac{d^2f}{dx^2}\right|_{x=x_0}\frac{(x-x_0)^2}{2!} + \ldots. \qquad (2.9)$$

Considering only the first term,

$$f(x) - f(x_0) \approx \left.\frac{df}{dx}\right|_{x=x_0}(x - x_0) \qquad (2.10)$$

Or

$$\delta f(x) \approx \left.\frac{df}{dx}\right|_{x=x_0}\delta x. \qquad (2.11)$$

Now linearizing Equation (2.8) using Equation (2.11), we obtain

$$A\frac{d\delta h}{dt} + \frac{Q_0}{2h_0}\delta h = \delta Q_{\text{in}}. \qquad (2.12)$$

Taking the Laplace transform of Equation (2.12), we obtain the transfer function of the tank for small perturbations about the steady-state value as a first-order system:

$$\frac{h(s)}{Q_{\text{in}}(s)} = \frac{1}{As + Q_0/2h_0}. \qquad (2.13)$$

The pump, level sensor, and the power amplifier are simple units and can be approximated to have just proportional gains and no system dynamics. The input–output relations of these units can be written as follows:

For the pump,

$$Q_p = K_p V_p;$$

for the level sensor,

$$V_l = K_l\, h;$$

and for the power amplifier,

$$V_0 = K_0 V_i\,.$$

Here Q_p is the pump flow rate, V_p the voltage applied to the pump, V_l the level sensor output voltage, V_0 the output voltage of the power amplifier, and V_i the input voltage of the power amplifier; K_p, K_l , K_0 are constants.

The D/A converter can be approximated to have a transfer function of $\frac{1-e\text{^}(-st)}{s}$

So the block diagram of the level control system can be obtained as shown in Figure 2.1 below.

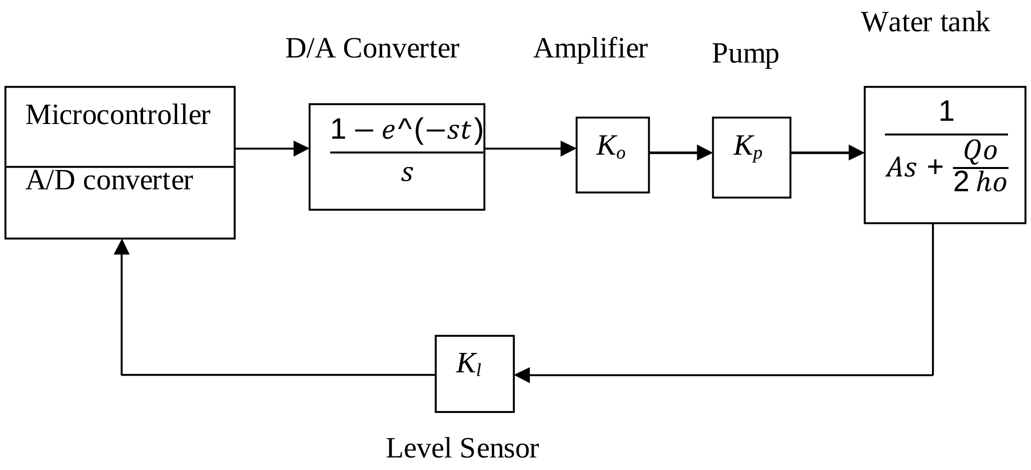


Fig 2.1 : Block diagram of the liquid level controller system[1]

2.2 SYSTEM DISCRETIZING

As we are dealing with discrete domain, we convert the palnt model in s-domain to z-domain to get the discretized plant and then to simulate the discretized plant in MATLAB simulink. Zero-order-hold equivalent method is used to dicretize the Liquid level system i.e. to convert G(s) to G(z) , where is G(s) is the combined transfer function of the three blocks of amplifier pump and the water tank ,i.e.

$$G(s) = \frac{KoKp}{As+Qo/2ho} = \frac{KoKp/A}{s+Qo/2Aho} \qquad (2.14)$$

A zero order hold transfer function is represented by ZOH = $G_o(s)$

$$G_o(s) = \frac{1-\exp(-sT)}{s} \qquad (2.15)$$

Now to discretize the plant model (*G(s)*) , a Zero order hold is followed by the plant as shown in the block diagram as follows

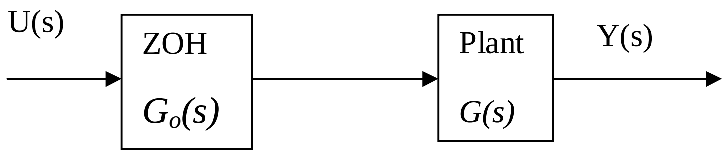


Fig 2.2 : Block diagram representation of the discretized plant model

The above block diagram can be reduced to a single block by combining the two transfer functions $G_o(s)$ and $G(s)$ in s-domain and converting the s-domain transfer function to z-domain, gives the dicretized transfer function model of the plant as follows.

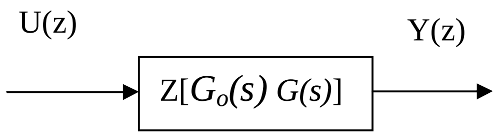


Fig 2.3 : Reduced Block diagram representation of the discretized plant model

Assuming T = 1 sec, and all coefficients and constants are assumed to be = 1 , we get

$$Z[G_o(s)\, G(s)] = \frac{0.63}{z-0.37} \qquad (2.16)$$

Now the controlled plant model can be represented by the following diagram

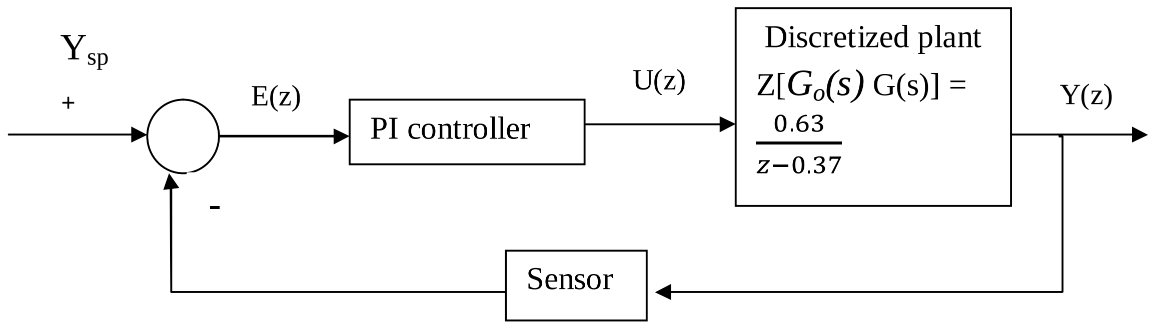


Fig 2.4 : Block Diagram representation of the plant model in discrete domain

2.3 SYSTEM SIMULATION

The final block diagram representation shown at the end of the previous section is simulated in MATLAB Simulink to see the behavior of Y(z), U (z) and E(z) with time. With repeated trial and error process, the PI constants are tuned and at the end the proportional constant is fixed at 1.9 and integral constant is fixed at 0.2 and the nature is recorded. The set point value y_{sp} is fixed at 3.8 (i.e. 3.8 volts is the corresponding voltage across the rotary potentiometer liquid level sensor for the desired height of the liquid level in the tank till which we want to fill the tank), i.e. the height of the liquid in the tank has to be controlled to be restricted at this level and beyond this level the pump should stop thus preventing overflow of the tank.

The block diagram of the uncontrolled discretized plant in MATLAB Simulink is as follows:

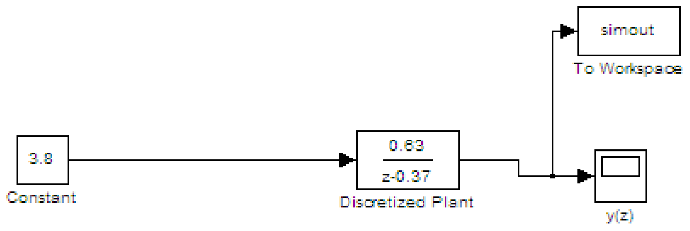


Fig 2.5 : Uncontrolled discretized plant in MATLAB Simulink

This on simulation gives the following nature of the output Y(z) :

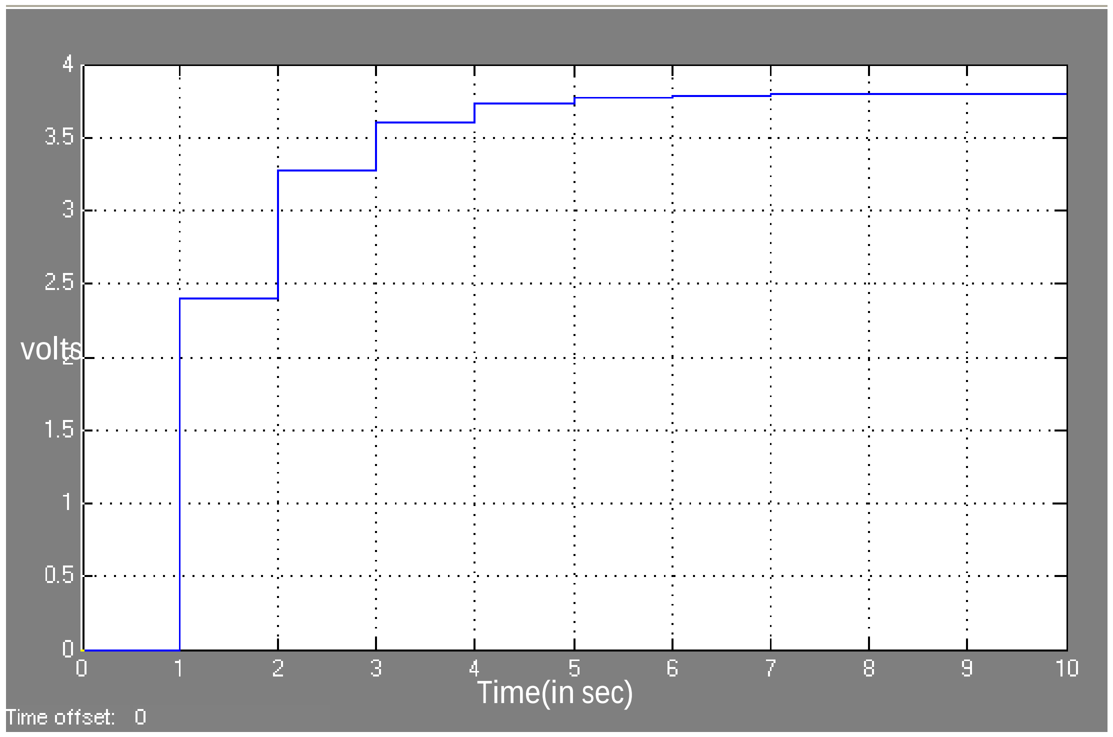


Fig 2.6 : Nature of output of Discretized plant as observed in MATLAB Simulink

Now the Discretised plant being controlled by a PI controller looks as follows is MATLAB Simulink :

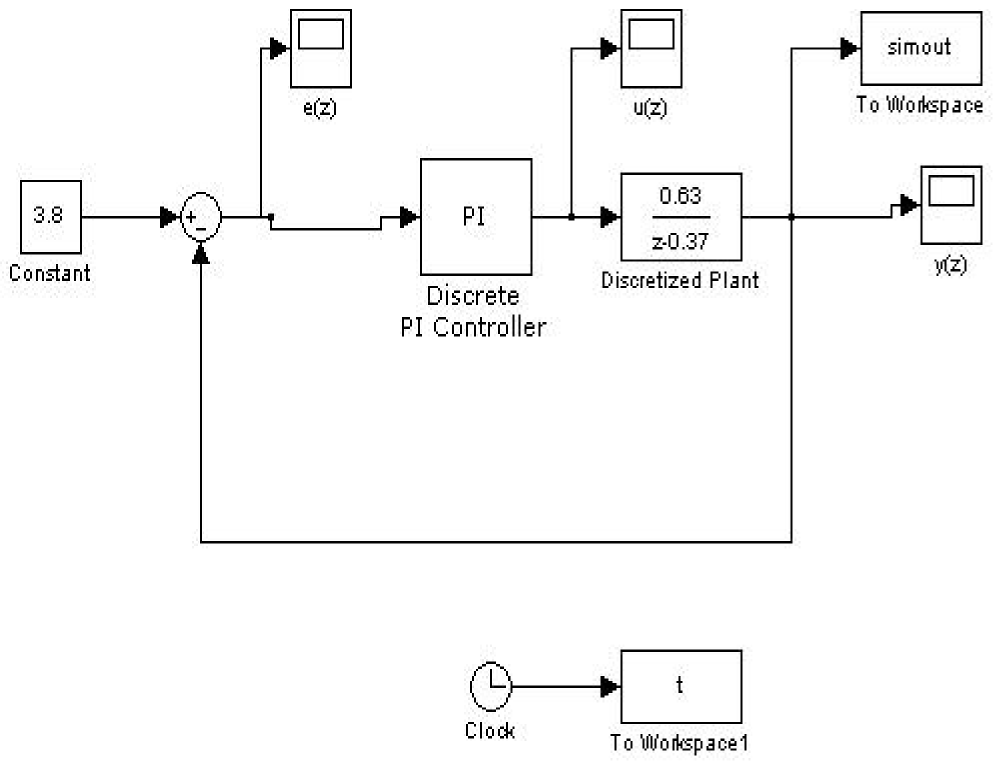


Fig 2.7 : PI controlled Discretized plant model in MATLAB Simulink

This on simulation gives the following nature of the output Y(z) :

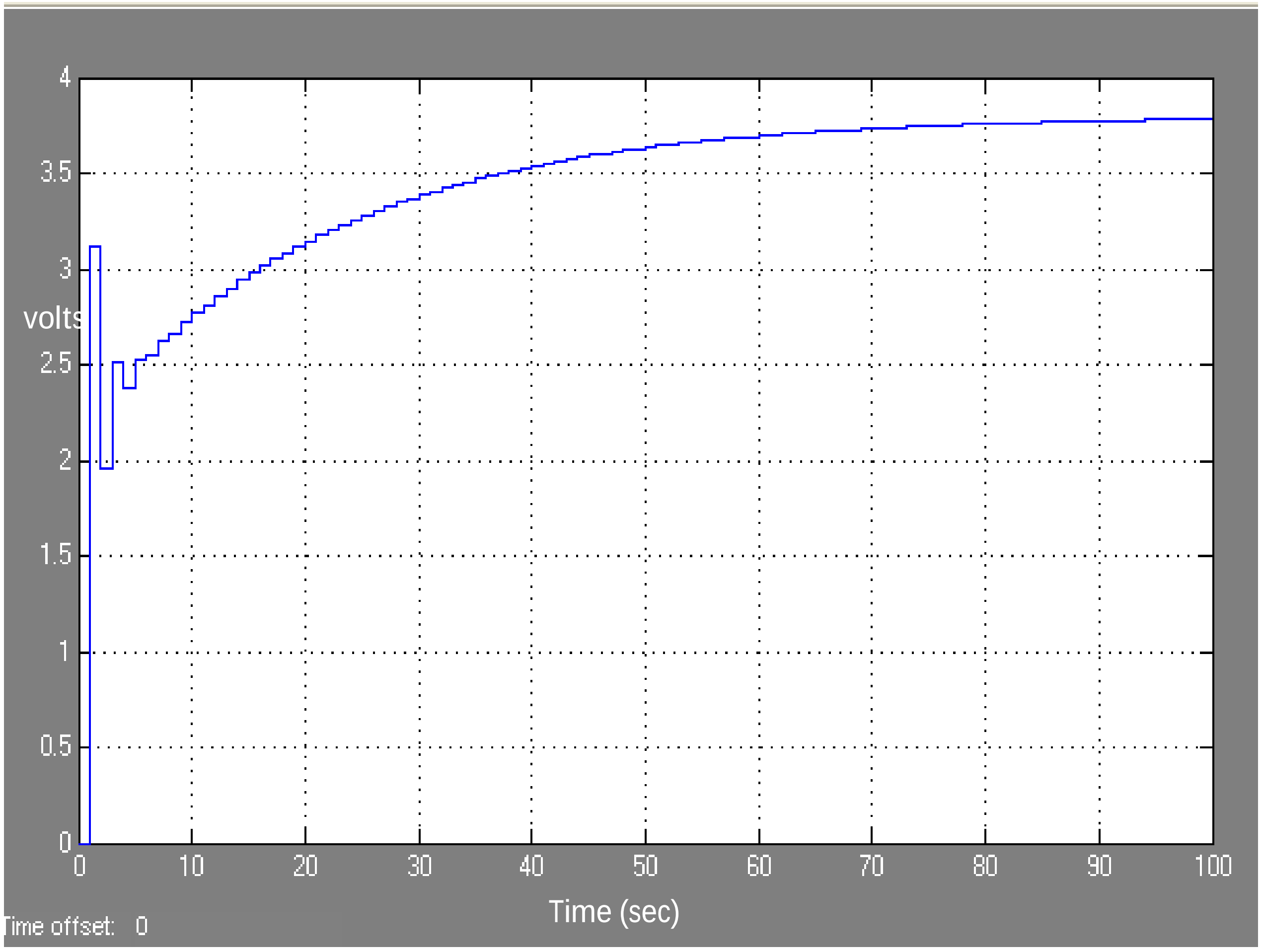


Fig 2.8 : Nature of output of PI controlled discretized plant model as observerd in MATLAB Simulink

The nature of the control signal U(z) from the PI controller is as follows :

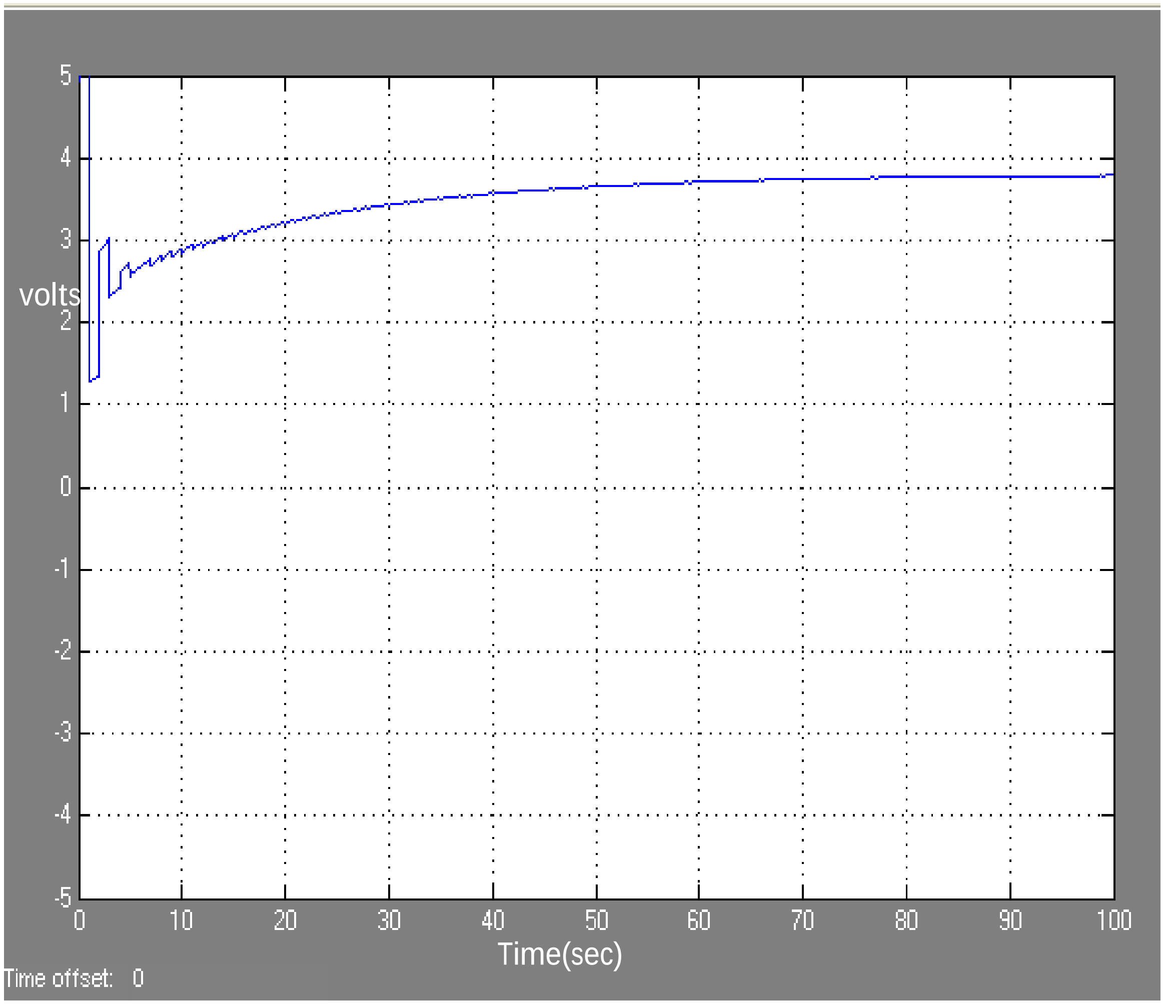


Fig 2.9 : Nature of control signal from the PI controller as observed in MATLAB Simulink

The nature of the error signal E(z) generated in the plant model is as follows :

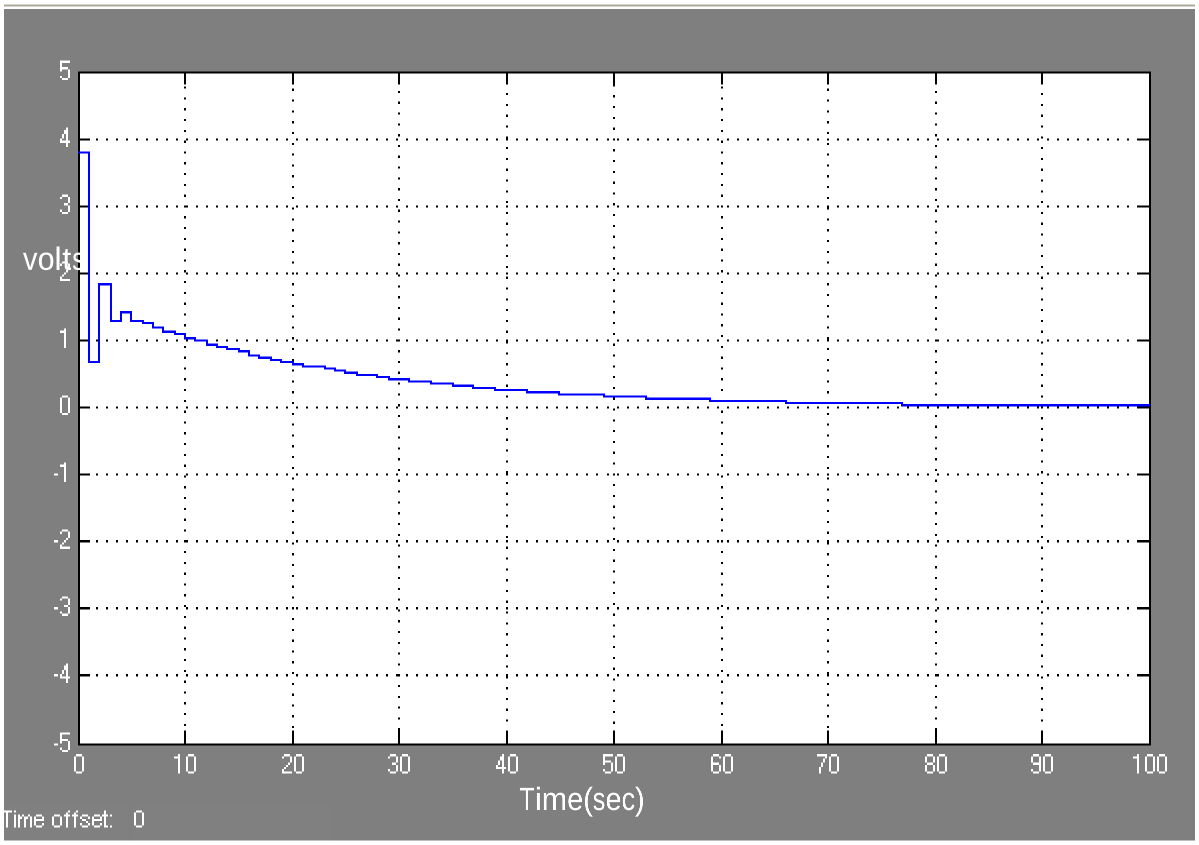


Fig 2.10 : Nature of error signal generated as observed in MATLAB Simulink

Thus by comparing the simulation results of an uncontrolled discretised plant and a PI controlled discretised plant , we observe that by implementing a PI controller the desired set value in the output is reached more steadily with less deviations and fluctuations . Hence we use a PI controller algorithm in our project to control the liquid level plant system to prevent overflowing of the tank.

CHAPTER 3

COMPONENTS USED IN THE PROJECT

CHAPTER 3

COMPONENTS USED IN THE PROJECT

3.1 ATMEGA32 MICROCONTROLLER

ATMEGA32 microcontroller was chosen as it has many benefits in comparison to other microcontrollers. Its features and benefits over other microprocessors are as follows:

3.1.1 Features of ATMEGA32 [6]

• High-performance, Low-power AVR 8-bit Microcontroller

• Advanced RISC (reduced instruction set computer) Architecture

– 131 Powerful Instructions – *(Most instructions operate in 1 clock cycle, and this leads to an almost 10-times performance improvement over conventional processors (e.g., the 8051) operating at equal Clock frequency)*

– 32 × 8 General purpose working Registers *(A large register set means that variables can be stored inside the CPU rather than storing the variables in memory, as accessing memory, is time expensive. Thus the program will run faster)*

–Wide range of inbuilt Clock frequency between 0 - 16 MHz

– on-chip hardware for 2-cycle Multiplier *(In many other microcontroller architectures, multiplication typically requires many more clock cycles)*

• High Endurance Non-volatile Memory segments

– 32 K bytes of In-System Self-programmable Flash program memory

– 1024 Bytes EEPROM

– 2 K bytes of internal SRAM *(the EEPROM and the RAM is seen as DATA memory for storing constants and variables and SRAM is used for stack)*

– Write/Erase Cycles: 10,000 times for Flash memory and 100,000 times for EEPROM

– 20 years of data retention at 85°C/100 years at 25°C

– Optional Boot Code Section with Independent Lock Bits

-In-System Programming by On-chip Boot Program *(This means we don't have to have external EPROMs or ROMs containing your program code. Also, the program memory can be programmed while the processor is in the target without removing it. This allows faster and easier system software upgrades.)*

– Programming Lock for Software Security

• Peripheral Features

– Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes

– One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

– Real Time Counter with Separate Oscillator

– Four PWM Channels

– 8-channel, 10-bit ADC

– Byte-oriented Two-wire Serial Interface

– Programmable Serial USART

– Master/Slave SPI Serial Interface

– Programmable Watchdog Timer with Separate On-chip Oscillator *(This is used to recover in case of software crash but can also be used for other interesting applications)*

– On-chip Analog Comparator

• Special Microcontroller Features

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated RC Oscillator

– External and Internal Interrupt Sources

– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby

• I/O and Packages

– 32 Programmable I/O Lines

– 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

• Operating Voltages

– 4.5V - 5.5V for ATmega32

• Speed Grades

– 0 - 16 MHz for ATmega32

• Power Consumption at 1 MHz, 3V, 25°C for ATmega32

– Active: 1.1 mA

– Idle Mode: 0.35 mA

– Power-down Mode: < 1 µA

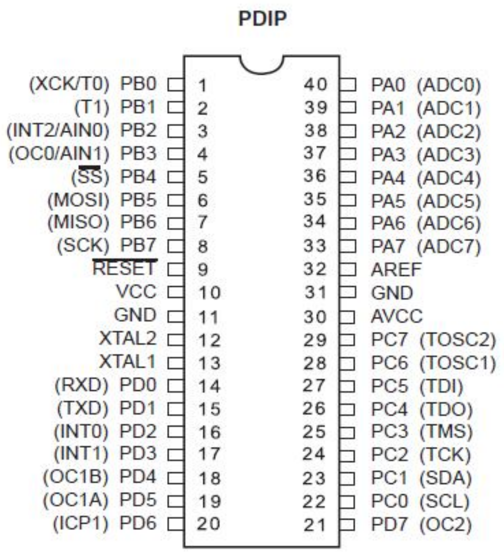


Fig 3.1 : Pinout ATMEGA 32 [6]

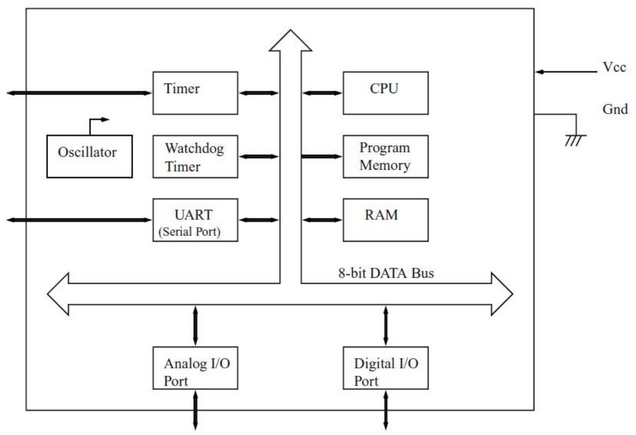


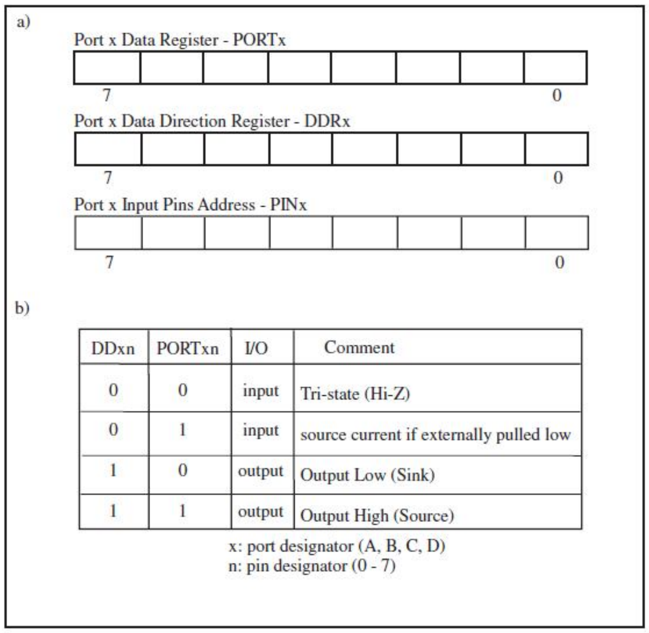
Fig 3.2 : ATMEGA 32 microcontroller[6]

3.1.2 ATMEGA32 Architecture

The Atmel ATmega16 is a register-based architecture. The processor is designed following the Harvard architecture format. That is, has separate, dedicated memories and buses for program and data information. The register-based Harvard Architecture coupled with the RISC-based instruction set, allows for faster and efficient program execution and allows the processor to complete an assembly language instruction every clock cycle [1].

3.1.3 PORT system

The Atmel ATmega32 is equipped with four 8-bit general-purpose, digital I/O PORTs designated as PORTA, PORTB, PORTC, and PORTD. All of these ports also have alternate functions as well.


a)
Port x Data Register - PORTx
7
0
Port x Data Direction Register - DDRx
7
0
Port x Input Pins Address - PINx
7
0
b)

DDxn	PORTxn	I/O	Comment
0	0	input	Tri-state (Hi-Z)
0	1	input	source current if externally pulled low
1	0	output	Output Low (Sink)
1	1	output	Output High (Source)

x: port designator (A, B, C, D)
n: pin designator (0 - 7)

Fig 3.3 : ATmega32 port configuration registers: (a) port-associated registers and (b) port pin configuration[3].

As shown in Figure 3.3 (a), each port has three registers associated with it:

• Data Register (PORTx)---This register is used to write output data to the port,

• Data Direction Register (DDRx)---This register is used to set a specific port pin to either output (by assigning 1) or input (by assigning 0), and

• Input Pin Address (PINx)---used to read present configuration of the port if the port behaves as a input port.

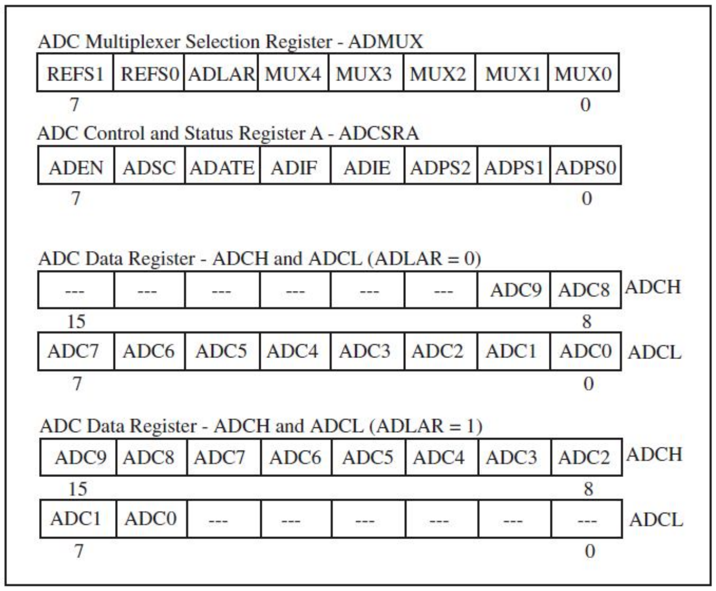
Figure 3.3(b) describes the settings required to configure a specific port pin to either input or output. Port pins should be normally configured at the beginning of a program to behave as either input or output, and their initial values are then set. It is a usual practice to configure all eight pins for a given port simultaneously. The data direction register (DDRx) is first used to set the pins as either input or output, and then the data register (PORTx) is used to set the initial value of the output port pins[3].

3.1.4 Analog-to-Digital Converter

The ATmega32 is equipped with an eight-channel ADC subsystem. PORTA alternatively acts as the ADC channel for input of analog signal to the microcontroller . The ADC converts an analog signal from the outside world into a binary representation suitable for use by the microcontroller. The ATMEGA32 ADC has by default 10-bit resolution. This means that an analog voltage between 0 and 5V will be encoded into one of 1024 binary representations between $(000)_{16}$ and $(3FF)_{16}$. This provides the ATmega32 with a voltage resolution of approximately 4.88 mV. It has ±2 LSB absolute accuracy i.e. ±9.76 mV at this resolution. The ADC can also be configured for 8-bit resolution[3].

ADC Register set :

The key registers for the ADC system are shown in Figure 3.4.



ADC Multiplexer Selection Register - ADMUX

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
7							0

ADC Control and Status Register A - ADCSRA

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
7							0

ADC Data Register - ADCH and ADCL (ADLAR = 0)

---	---	---	---	---	---	ADC9	ADC8	ADCH
15							8	
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
7							0	

ADC Data Register - ADCH and ADCL (ADLAR = 1)

ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
15							8	
ADC1	ADC0	---	---	---	---	---	---	ADCL
7							0	

Fig 3.4 : ADC Registers [3]

ADC Multiplexer Selection Register (ADMUX) :

The analog input channel for conversion is selected using the MUX[4:0] bits in the ADC Multiplexer Selection Register (ADMUX). The 10-bit result from the conversion process is placed in the ADC Data Registers, ADCH and ADCL. These two registers provide 16 bits for the 10-bit result. The result may be either left justified by setting the ADLAR (ADC Left Adjust Result) bit of the ADMUX register. Right justification is provided by clearing this bit. The REFS[1:0] bits of the ADMUX register are also used to determine the reference voltage source for the ADC system. These bits may be set to the following values:

• REFS[0:0] = 00: AREF used for ADC voltage reference

• REFS[0:1] = 01: AVCC with external capacitor at the AREF pin

• REFS[1:0] = 10: reserved

• REFS[1:1] = 11: internal 2.56-VDC voltage reference with an external capacitor at the AREF pin[3]

ADC Control and Status Register A (ADCSRA):

The ADCSRA register contains the ADC Enable (ADEN) bit. This bit is the ‘‘on/off’’ switch for the ADC system. The ADC is turned on by setting this bit to a logic 1. Setting the ADC Start Conversion (ADSC) bit to logic 1 initiates an ADC. The ADCSRA register also contains the ADC Interrupt flag (ADIF) bit. This bit sets to logic 1 when the ADC is complete. The ADIF bit is reset by writing a logic 1 to this bit. The ADPS[2:0] bits are used to set the ADC clock frequency. The ADC clock is derived from dividing down the main microcontroller clock. The ADPS[2:0] may be set to the following values:

• ADPS[2:0] = 000: division factor: 2

• ADPS[2:0] = 001: division factor: 2

• ADPS[2:0] = 010: division factor: 4

• ADPS[2:0] = 011: division factor: 8

• ADPS[2:0] = 100: division factor: 16

• ADPS[2:0] = 101: division factor: 32

• ADPS[2:0] = 110: division factor: 64

• ADPS[2:0] = 111: division factor: 128[3]

ADC Data Registers (ADCH and ADCL) :

The ADC Data Register contains the result of the ADC. These two registers provide 16 bits for the 10-bit result. The result may be left justified by setting the ADLAR (ADC Left Adjust Result) bit of the ADMUX register. Right justification is provided by clearing this bit. If we left justify and , just take the ADCH value, neglecting the two LSBs in ADCL,then we can get an 8-bit resolution[3].

In this project analog signal in form of voltage from the level sensor is fed to the microcontroller. As the microcontroller is a digital device and cannot process analog signals so we need to convert this analog signal to digital form and hence we use the ATMEGA32 ADC.

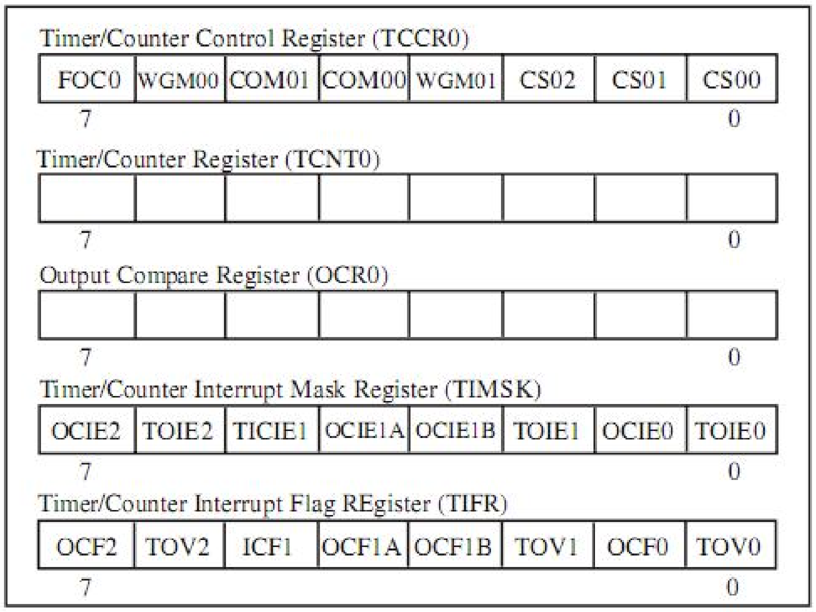
The ATMEGA32 ADC channel 0 is used for the Analog input and main clock frequency / 8 = 125 kHz is taken as the sampling time. The ADC result is left justified.

3.1.5 Timer Subsystem

The Atmel ATmega32 has a flexible and powerful three-channel timing system. The three timer channels Timer 0(8-bit timer), Timer 1(16-bit timer), and Timer 2(8-bit timer).
Timer0 is only used in this project and so only that is explained in detail.

Timer0 Register Set :

The following figure shows the Timer0 Registers:


Timer/Counter Control Register (TCCR0)

7							0
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Timer/Counter Register (TCNT0)

7							0

Output Compare Register (OCR0)

7							0

Timer/Counter Interrupt Mask Register (TIMSK)

7							0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

Timer/Counter Interrupt Flag REgister (TIFR)

7							0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

Fig 3.5 : Timer0 Registers[3]

Timer/Counter Control Register 0(TCCR0) :

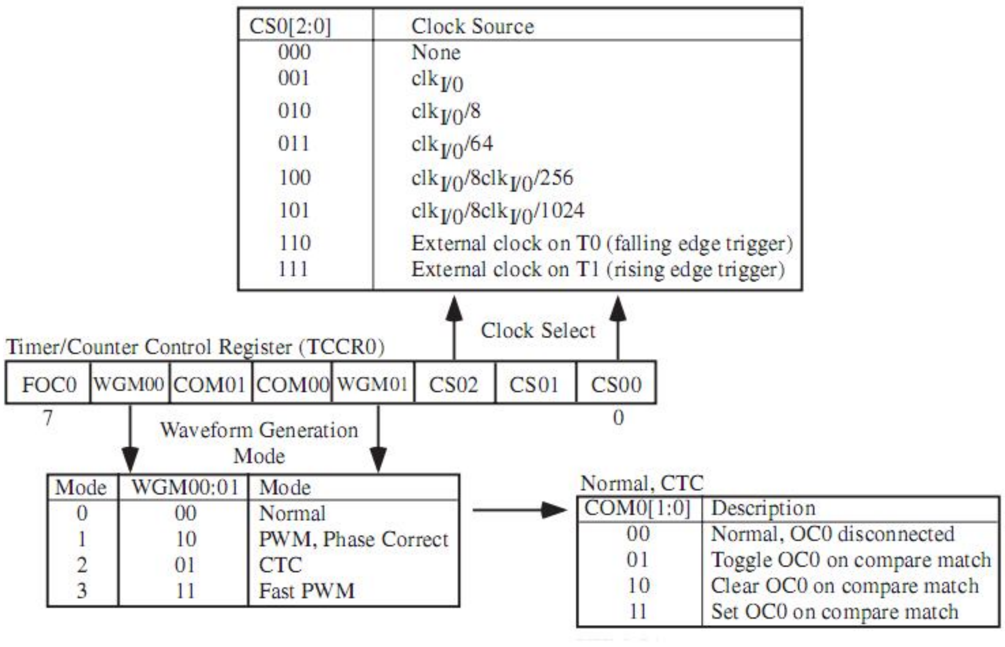
The TCCR0 register bits are used to -

• select the operational mode of Timer 0 using the Waveform Mode Generation (WGM0[1:0]) bits,

• determine the operation of the timer within a specific mode with the Compare Match Output Mode (COM0[1:0]) bits, and

• select the source of the Timer 0 clock and the prescaler to subdivide the main clock frequency down to timer system frequency (clk_{Tn})using CS0[2:0] bits. [3]

The bit settings for the TCCR0 register are summarized in the following figure :



CS0[2:0]	Clock Source
000	None
001	$clk_{I/0}$
010	$clk_{I/0}/8$
011	$clk_{I/0}/64$
100	$clk_{I/0}/8clk_{I/0}/256$
101	$clk_{I/0}/8clk_{I/0}/1024$
110	External clock on T0 (falling edge trigger)
111	External clock on T1 (rising edge trigger)

Mode	WGM00:01	Mode
0	00	Normal
1	10	PWM, Phase Correct
2	01	CTC
3	11	Fast PWM

COM0[1:0]	Description
00	Normal, OC0 disconnected
01	Toggle OC0 on compare match
10	Clear OC0 on compare match
11	Set OC0 on compare match

Fig 3.6 : TCCR0 Register configuration[3]

Timer/Counter Register 0 (TCNT0) :

The TCNT0 is the 8-bit counter for Timer 0. The timer clock source (clk_{Tn}) is fed to the 8-bit Timer/Counter Register (TCNT0). This register is incremented (or decremented) on every clock pulse clk_{Tn} [3].

Output Compare Register 0(OCR0) :

The OCR0 register holds a user-defined 8-bit value that is continuously compared with the TCNT0 register [3].

Timer/Counter Interrupt Mask Register (TIMSK):

The TIMSK register is used by all three timer channels. Timer 0 uses the Timer/Counter0 Output Compare Match Interrupt Enable (OCIE0) bit and the Timer/Counter 0 Overflow Interrupt Enable (TOIE0) bit. When the OCIE0 bit and the I-bit in the Status Register are both set to 1, the Timer/Counter 0 Compare Match interrupt is enabled. When the TOIE0 bit and the I-bit in the Status Register are both set to 1, the Timer/Counter 0 Overflow interrupt is enabled [3].

Timer/Counter Interrupt Flag Register(TIFR) :

The TIMSK register is used by all three timer channels. Timer 0 uses the OCF0(Output compare Flag), which sets for an output compare match. Timer 0 also uses the TOV0(Timer/Counter overflow flag), which sets when Timer/Counter 0 Overflows [3].

Modes of Operation :

The following diagram shows the modes of operation of the Timer0.

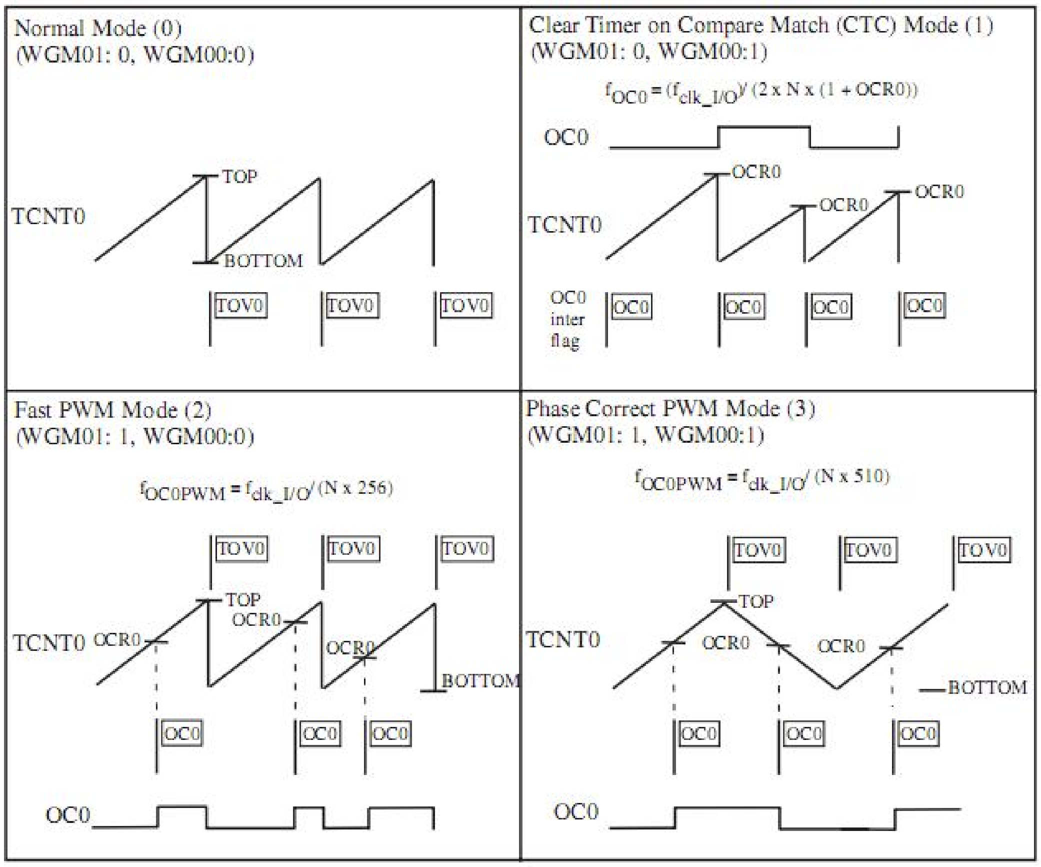

Normal Mode (0)
(WGM01: 0, WGM00:0)
TCNT0
TOP
BOTTOM
TOV0
Clear Timer on Compare Match (CTC) Mode (1)
(WGM01: 0, WGM00:1)
$f_{OC0} = (f_{clk\_I/O})/(2 \times N \times (1 + OCR0))$
OC0
OCR0
OC0 inter flag
Fast PWM Mode (2)
(WGM01: 1, WGM00:0)
$f_{OC0PWM} = f_{clk\_I/O}/(N \times 256)$
Phase Correct PWM Mode (3)
(WGM01: 1, WGM00:1)
$f_{OC0PWM} = f_{clk\_I/O}/(N \times 510)$

Fig 3.7 : Modes of operation of Timer0[3]

Mode 1 (i.e. Clear timer on compare match (CTC)mode) is used in this project .In this mode , the TCNT0 timer register is reset to 0 every time the TCNT0 counter reaches the value set in OCR0. The Output Compare Flag 0 (OCF0) is set when this event occurs[3]. A 1 is written to this flag from the program to clear it.

3.1.6 Interrupt Subsystem

The interrupt system on board a microcontroller allows it to respond to higher-priority or unscheduled events occurring during a program execution. These events may be planned, but we do not know when they will occur. When an interrupt event occurs, the microcontroller will normally complete the instruction it is currently executing and then transit the program control to tasks related to the interrupt event. When one interrupt is being served, then no other interrupt can occur as the microcontroller deactivates the interrupt system for preventing further interrupts. The tasks, which correspond to the interrupt event, are organized in a function called the interrupt service routine (ISR). Each interrupt will normally have its own ISR. Once the ISR is complete, the microcontroller will resume processing where it left off before the interrupt event occurred.

The ATmega16 can handle 21 interrupt sources. Three of the interrupts can be from external interrupt sources, and the remaining 18 interrupts are for the peripheral subsystems of the microcontroller. The ATmega16 interrupt sources are shown in Figure 21. The interrupts are listed in descending order of priority. RESET, INT0 (pin 16) and INT1 (pin 17) are external interrupts and the remaining interrupt sources are internal to the ATmega16 [3].

To program an interrupt, the user has to do the following actions:

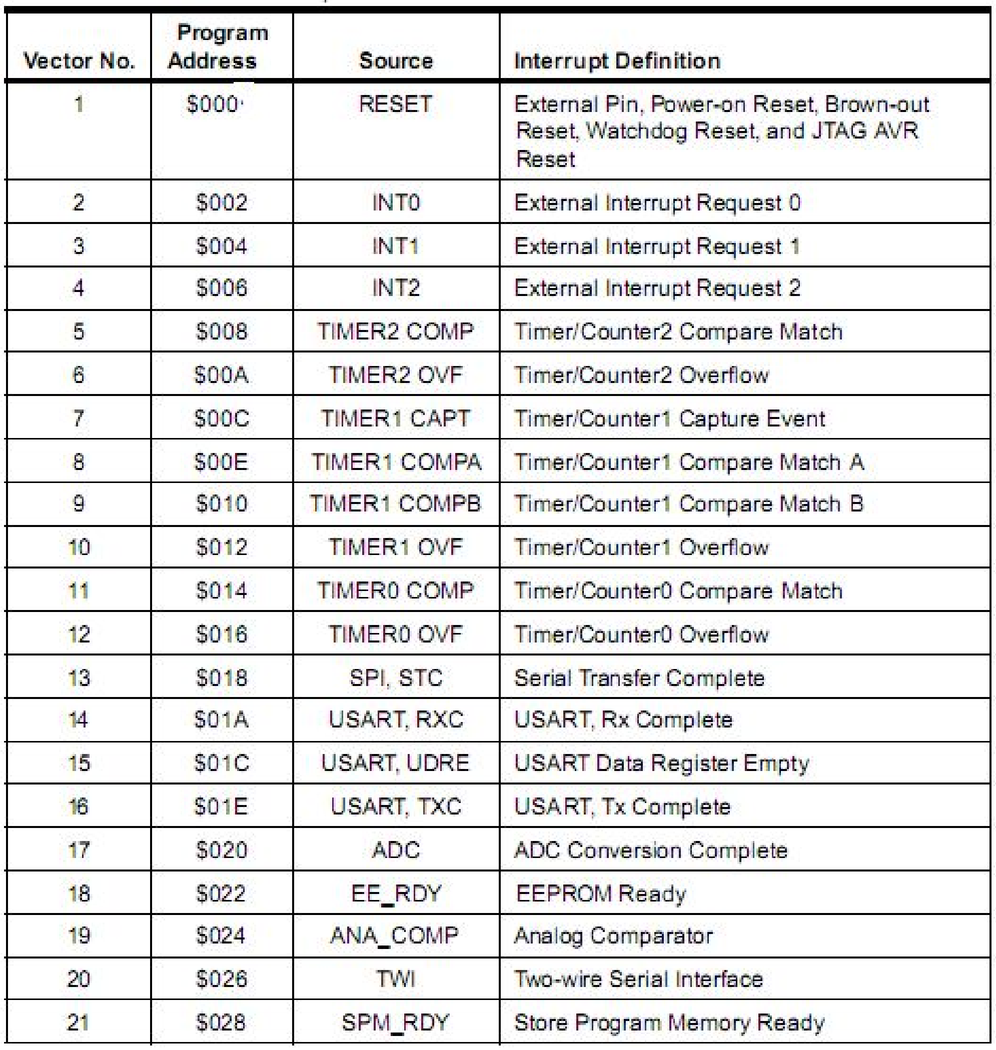
• Associate the ISR for a specific interrupt to the correct interrupt vector address, which points to the starting address of the ISR.

• Enable the interrupt system globally. This is accomplished with the assembly language instruction SEI.

• Enable the specific interrupt subsystem locally .

• Configure the registers associated with the specific interrupt correctly.

*In this project Timer0 output compare match interrupt has been used. The OCR0 is assigned decimal value 155 and TCCR0 is assigned 0X0B i.e. the timer is configured for CTC mode and the inbuilt set main clock source frequency (1MHz) is divided by 64 to give timing system frequency of 15.6 kHz so that the counter TCNT0 increments every 64 microseconds and it would set the OCF0 in 156 ticks i.e. when the TCNT0 rolls over to 0 at the 156th clock tick after becoming equal to 155 (at the 155th clock tick)which is the value stored in OCR0 and hence cause the TIMER0 COMP interrupt to occur in 156 * 64µs = 0.01s.*



Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Fig 3.8 : Atmel AVR ATMEGA32 Interrupts [6]

3.2 AD7302 DIGITAL-TO-ANALOG CONVERTER

A digital-to-analog converter (DAC) is a device which is used to convert incoming digital pulses to analog signals to be sent to the next device. Although there are a few microcontrollers that incorporate the D/A converter on chip, most microcontrollers still need to use an off-chip D/A converter to perform the D/A conversion function. Majority of integrated circuit DACs use R/2R ladder to convert digital value to analog value.

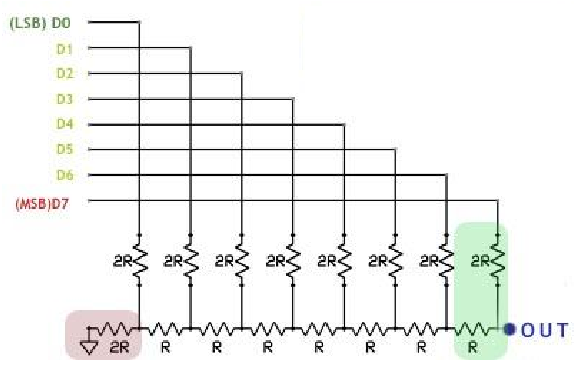


Fig 3.9 : R/2R network for converting digital value to analog value[11]

The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. For n number of data bit inputs, number of analog output levels is equal to 2^n .Therefore , an 8-bit DAC provides 256 discrete voltage(or current) levels of output.

In this project the signal coming from the microcontroller is in digital format and an analog power amplifier has to be fed with this control signal. So a DAC is needed to convert the digital control signal to analog control signal. So DAC AD7302 is used for the conversion as it has a WR write pin capable of enabling and disabling the DAC , thus latching the values and preventing them from getting changed when needed.

The AD7302 is a dual channel 8-bit D/A converter chip from Analog Devices that has a parallel interface with the microcontroller. The AD7302 converts an 8-bit digital value into an analog voltage.The AD7302 is designed to be a memory-mapped device. Its pin configuration is given in the following figure.

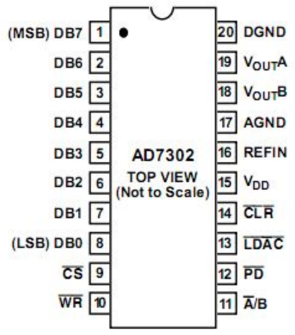


Fig 3.10 : Pin configuration of AD7302[7]

In order to send data to AD7302 ,the CS signal must be pulled to low. On the rising edge of the WR signal the values on D7-D0 will be latched into the input register. When the signal LDAC is low , the data in input register will be transferred to the DAC register and a new D/A conversion is started. The PD pin has to be pulled high for its functioning.The AD7302 needs a reference voltage to perform the D/A conversion. The reference voltage can come from either the external REFIN input or the internal V_{DD} . The A/B signal selects the channel A or B to perform the D/A conversion.

The AD7302 operates from a single +2.7V to +5.5V supply and typically consumes 15mW at 5V, making it suitable for battery powered applications.Each digital sample takes about 2μs to convert .

The output voltage V_{outA} or V_{outB} from either DAC is given by:

$V_{out\ A/B} = 2 * V_{ref} * (N/256)$

Where V_{ref} = is the voltage applied to the external REFIN pin or VDD/2 when the internal reference is selected.

N is the decimal equivalent of the code loaded to the DAC register and ranges from 0 to 255.

If the voltage applied to REFIN pin is within 1V of V_{DD}, V_{DD} is used as the reference voltage automatically. Otherwise voltage applied at the REFIN pin is used as the reference voltage [7][13].

3.3 LM675 POWER AMPLIFIER

The LM675 is a monolithic power operational amplifier featuring wide bandwidth and low input offset voltage. It is equally suitable for AC and DC applications. The LM675 is capable of delivering output currents in excess of 3 amps, giving upto 30W power, operating at supply voltages of up to 60V. The amplifier is also internally compensated for gains of 10 or greater. Its applications are

- High performance power op amp
- Bridge amplifiers
- Motor speed controls
- Servo amplifiers
- Instrument systems[8]

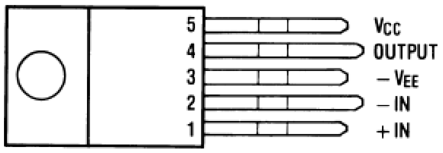


Fig 3.11 : LM675 Pin diagram[8]

In this project the analog control signal has to drive a 12V water pump needing 3A current but the signal coming from the DAC is of 5V and milliwatts range and hence is insufficient to drive the pump. So we need to connect a power amplifier to boost the signal coming from the DAC so as to be capable of driving the pump.

3.4 Water Pump

The pump is a small 12V water pump drawing about 3A when operating at the full-scale voltage

Fig 3.12 : Water pump used in the project[2]

3.5 Liquid level sensor :

A rotary potentiometer type level sensor is used in this project. The sensor consists of a floating arm connected to the sliding arm of a rotary potentiometer. As the level of the floating arm changes due to the changing height of the liquid in the tank, the resistance of the rotary potentiometer changes.

A voltage is applied across the potentiometer to form a voltage divider and the change of voltage is measured across the arm of the potentiometer. The voltage changes from 3.2 V when the floating arm is at the bottom to 4.6 V when the floating arm is at the top. Depending on the desired height of the liquid to be set as set-point, the corresponding voltage across the rotary potentiometer is our required set-point which has to be maintained by the controller.

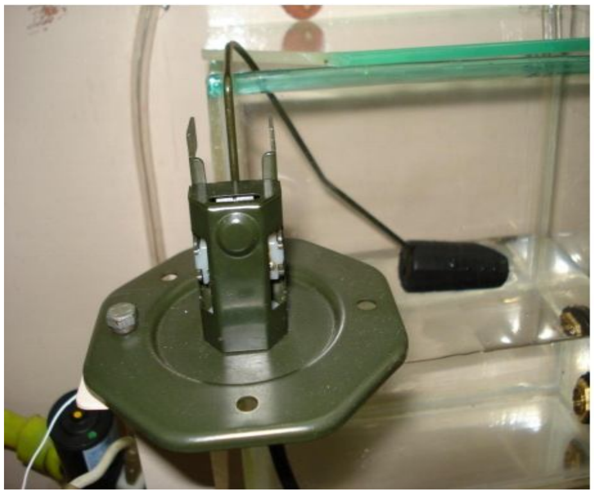


Fig 3.13 : Rotary potentiometer liquid level sensor used in the project[2]

CHAPTER 4

TESTING AND INTERFACING OF THE DEVICES

CHAPTER 4

TESTING AND INTERFACING OF THE DEVICES

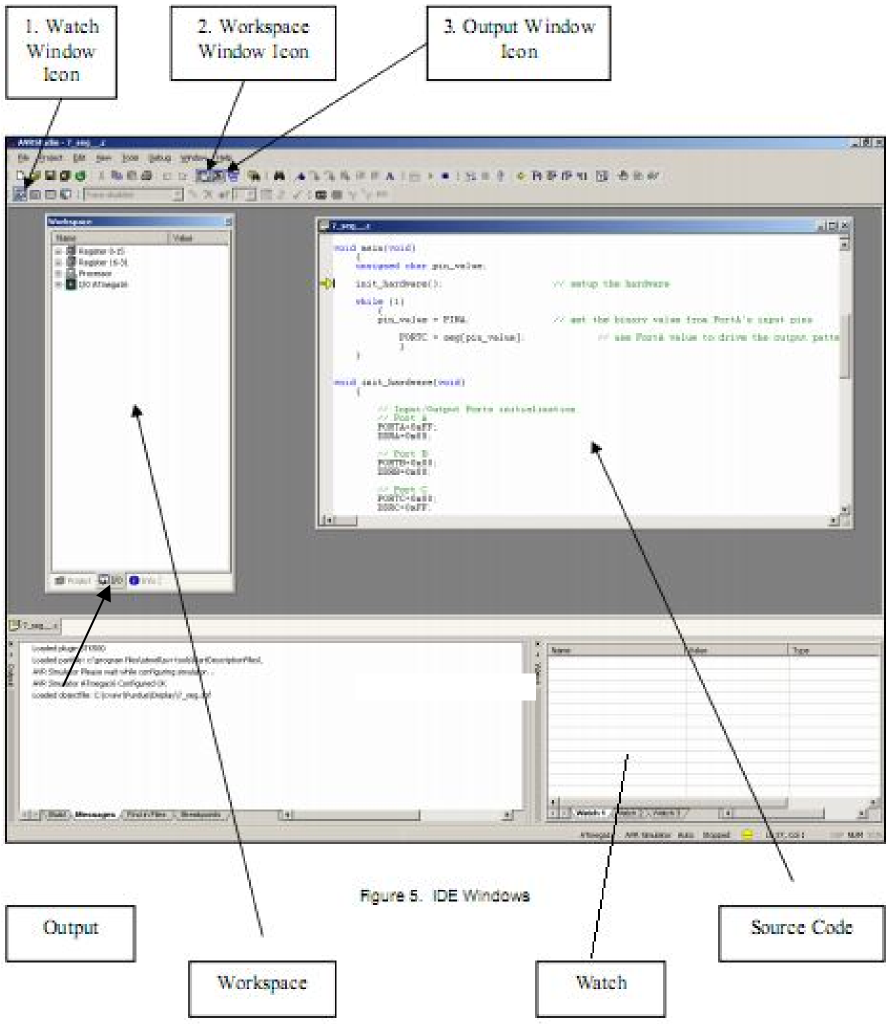
4.1 TESTING OF DEVICES

4.1.1 Testing of ATMEGA32

Before using ATMEGA32 in real time, programs are written and simulated to see if they execute as desired or not and give the correct output or not.AVR Studio 4's simulator is used for this purpose. This helps in easy debugging. If programs are directly used in real time then in case the desired result is not obtained then it becomes difficult to trace where the fault lies. So it is advisable to first simulate and then use the program in real time. So during the whole course of project work, all programs were first tested by simulating in AVR Studio and then dumped into the microcontroller for execution in real time.

The AVR Studio 4 is an Integrated Development Environment(IDE) for debugging AVR software. The AVR Studio allows chip simulation and in-circuit emulation for the AVR family of microcontrollers. The user interface is designed in such a way that it is easy to use and gives complete information overview. The IDE has several windows that provide important information to the user. The main windows of interest are the Workspace, Source Code, Output, and Watch windows. These can be seen in the next page in figure 4.1.

The AVR uses the same user interface for both simulation and emulation. The AVR Studio uses a COF object file for simulation. This file is created through the C compiler by selecting COF as the output file type[12].

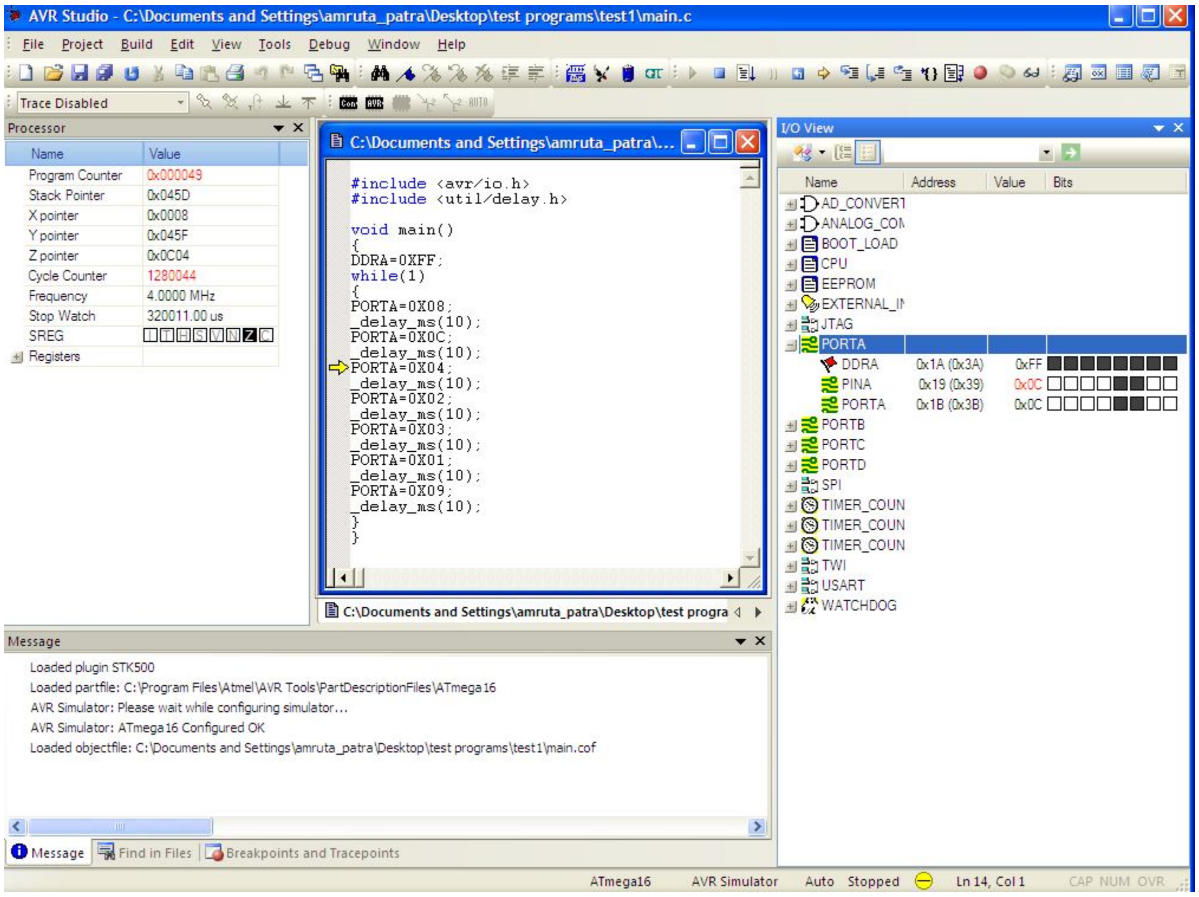


```
void main(void)
    {
    unsigned char pin_value;

    init_hardware();                    // setup the hardware

    while (1)
        {
        pin_value = PINA;               // get the binary value from PortA's input pins
            PORTC = seg[pin_value];           // use PortA value to drive the output patte
            }
    }

void init_hardware(void)
    {

        // Input/Output Ports initialization
        // Port A
        PORTA=0xFF;
        DDRA=0x00;

        // Port B
        PORTB=0x00;
        DDRB=0x00;

        // Port C
        PORTC=0x00;
        DDRC=0xFF;
```

```
Loaded plugin STK500
Loaded partfile: c:\program files\atmel\avr tools\partDescriptionFiles\[illegible]
AVR Simulator: Please wait while configuring simulator...
AVR Simulator: ATmega16 Configured OK
Loaded objectfile: C:\[illegible]\7_seg.[illegible]
```

Fig 4.1 : A sample of IDE windows[12]

In the following figure we can see that in the right side we can see the status of the different PORTs , the ADC registers , the timer registers etc


AVR Studio - C:\Documents and Settings\amruta_patra\Desktop\test programs\test1\main.c

Name	Value
Program Counter	0x000049
Stack Pointer	0x045D
X pointer	0x0008
Y pointer	0x045F
Z pointer	0x0C04
Cycle Counter	1280044
Frequency	4.0000 MHz
Stop Watch	320011.00 us
SREG	I T H S V N Z C
Registers	

```
#include <avr/io.h>
#include <util/delay.h>

void main()
{
DDRA=0XFF;
while(1)
{
PORTA=0X08;
_delay_ms(10);
PORTA=0X0C;
_delay_ms(10);
PORTA=0X04;
_delay_ms(10);
PORTA=0X02;
_delay_ms(10);
PORTA=0X03;
_delay_ms(10);
PORTA=0X01;
_delay_ms(10);
PORTA=0X09;
_delay_ms(10);
}
}
```

Name	Address	Value	Bits
AD_CONVERT			
ANALOG_COM			
BOOT_LOAD			
CPU			
EEPROM			
EXTERNAL_IN			
JTAG			
PORTA			
DDRA	0x1A (0x3A)	0xFF	■■■■■■■■
PINA	0x19 (0x39)	0x0C	□□□□■■□□
PORTA	0x1B (0x3B)	0x0C	□□□□■■□□
PORTB			
PORTC			
PORTD			
SPI			
TIMER_COUN			
TIMER_COUN			
TIMER_COUN			
TWI			
USART			
WATCHDOG			

```
Loaded plugin STK500
Loaded partfile: C:\Program Files\Atmel\AVR Tools\PartDescriptionFiles\ATmega16
AVR Simulator: Please wait while configuring simulator...
AVR Simulator: ATmega16 Configured OK
Loaded objectfile: C:\Documents and Settings\amruta_patra\Desktop\test programs\test1\main.cof
```

ATmega16 AVR Simulator Auto Stopped Ln 14, Col 1 CAP NUM OVR

Fig 4.2 A snapshot of a program being simulated

4.1.2 Testing of ATMEGA32 ADC

Varying voltage was given at pin no 40 PA0 of the microcontroller and the ADC outputs ADCL and ADCH were transferred to PORTC and PORTD respectively to record and analyze the outputs. The results are tabulated as follows :

Table 4.1 - Observations for output of ADC of ATMEGA32 for varying input given at ADC channel 0 of ATMEGA32

I/P volt (in volts)	O/P voltage at the respective pins of PORTC and PORTD (in volts)										O/P voltage (converted to binary)
	PD1	PD0	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	
0.37	0	0	0	0	4.92	0	4.92	4.92	0	0	0000101100
0.482	0	0.001	0.005	4.95	4.93	0.07	4.93	4.93	0.06	4.89	0001101101
0.751	0.002	0.002	4.95	0.001	4.94	0.01	4.94	4.94	4.95	0.002	0010101110
0.932	0.001	0.001	4.95	0.003	4.94	0.1	4.94	4.94	0.06	4.94	0010101101
1.224	0.002	0.001	4.95	4.95	4.93	0.008	4.93	4.93	4.95	4.95	0011101111
1.49	0.005	0.002	0.002	0.001	4.93	0.013	4.93	4.94	4.95	0.003	0000101110
1.98	0.004	0.004	4.95	0.002	4.93	0.012	4.93	4.93	0.02	4.9	0010101101
1.765	0.02	4.95	0.002	4.95	4.93	0.01	4.93	4.93	0.001	0.006	0111101111
2.28	0.002	4.95	4.95	4.95	4.93	0.015	4.93	4.93	4.95	4.95	0111101111
2.504	4.95	0.001	0.001	0.001	4.93	0.011	4.933	4.94	4.3	4.3	1000101111
2.771	4.95	0.001	0.001	0.001	4.93	0.015	4.93	4.93	0.001	4.95	1000101101
2.911	4.944	0.001	0.001	4.944	4.927	0.073	4.928	4.927	4.944	4.944	1001101111
3.027	4.95	0.000	0.002	4.95	4.93	0.013	4.93	4.93	4.95	0.001	1001101110
3.261	4.95	0.001	4.95	0.003	4.93	0.015	4.94	4.9	0.002	4.7	1010101101
3.48	4.95	0.001	4.95	4.95	4.93	0.014	4.93	4.93	4.95	4.95	1011101111
3.73	4.95	4.95	0.001	0.001	4.93	0.011	4.93	4.93	4.95	4.95	1100101111
4.032	4.95	4.95	0.06	4.95	4.93	0.013	4.93	4.93	0.011	4.952	1101101101
4.26	4.95	4.95	0.005	4.94	4.93	0.011	4.93	4.94	4.95	0.008	1101101110
4.497	4.95	0.005	4.951	0.002	4.936	0.011	4.9	4.9	4.9	4.9	1010101111
4.75	4.95	0.05	4.95	4.95	4.93	0.01	4.93	4.93	0.004	4.95	1011101101
4.94	4.95	4.95	4.95	4.95	4.93	0.021	4.93	4.93	4.95	4.95	1111101111

From the tabulated data we find that the microcontroller gives approximately the correct expected values after converting the analog i/p voltage to its equivalent 10-bit digital value. The formula it obeys is –

decimal equivalent of i/p voltage = i/p voltage * 1024 / 5

The decimal equivalent is then converted to binary and compared with the last column of the above table and is almost found nearby.

4.1.3 AD7302 Testing

Different combinations of binary 1 and 0 i.e. +5 volts and 0 volts were given at the input pins of the DAC and the outputs were recorded in the following table.

Table 4.2 - Observations for outputs of DAC for different combinations of its binary inputs

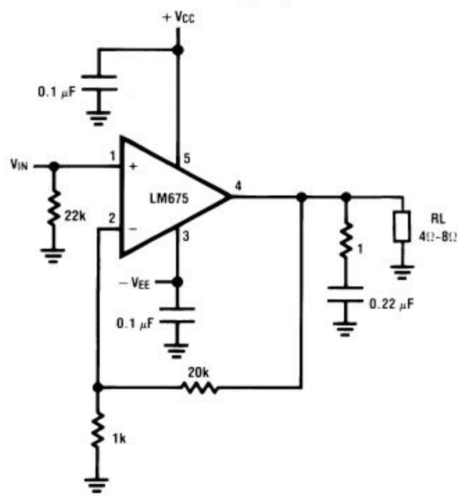
I/P voltage(binary)								O/P voltage(volts)
D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	0	0	0.018
0	0	0	0	0	0	0	1	0.037
0	0	0	0	0	0	1	0	0.057
0	0	0	0	0	0	1	1	0.077
0	0	0	0	0	1	0	0	0.096
0	0	0	0	0	1	0	1	0.115
0	0	0	0	0	1	1	0	0.135
0	0	0	0	0	1	1	1	0.154
0	0	0	0	1	0	0	0	0.174
0	0	0	0	1	0	0	1	0.193
0	0	0	0	1	0	1	0	0.212
0	0	0	0	1	0	1	1	0.232
0	0	0	0	1	1	0	0	0.251
0	0	0	0	1	1	0	1	0.271
0	0	0	0	1	1	1	0	0.291
0	0	0	0	1	1	1	1	0.310
0	0	0	1	0	0	0	0	0.327
0	0	0	1	0	0	0	1	0.346
0	0	0	1	0	0	1	0	0.366
0	0	0	1	0	0	1	1	0.386
0	0	0	1	0	1	0	0	0.405
0	0	0	1	0	1	0	1	0.424
0	0	0	1	0	1	1	0	0.444
0	0	0	1	0	1	1	1	0.464
0	0	0	1	1	0	0	0	0.482
0	0	0	1	1	0	0	1	0.502
1	1	1	1	1	0	0	0	4.822
1	1	1	1	1	0	0	1	4.841
1	1	1	1	1	0	1	0	4.861
1	1	1	1	1	0	1	1	4.880
1	1	1	1	1	1	0	0	4.900
1	1	1	1	1	1	0	1	4.920
1	1	1	1	1	1	1	0	4.938
1	1	1	1	1	1	1	1	4.940

It was found that the outputs were almost equal to the theoretically calculated values of D/A conversion value given by the following relation-

DAC o/p value = (decimal equivalent of i/p binary + 1) * 5 V / 256

4.1.4 LM675 Testing

The circuit for testing the power amplifier is as follows.Varying input was given at the input terminal and outputs were tabulated. A plot was made with input voltages as x-axis and output voltages as y-axis and the curve was found to become horizontal at input voltage = 4.02V and the corresponding output voltage was found to be 11.03 volts.



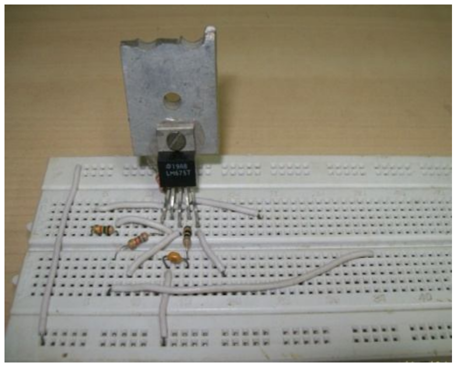


Fig 4.3: LM675 application circuit diagram[8] Fig 4.4: LM675 circuit used in the project

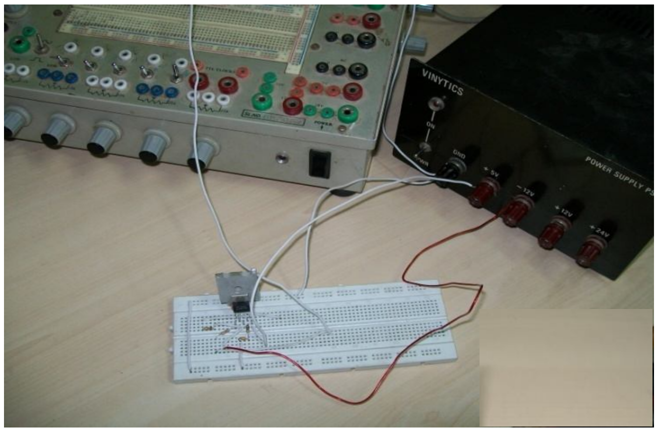


Fig 4.5: LM675 circuit testing

Table 4.3 - Observations for outputs of LM675 for varying inputs

I/P voltage(in volts)	O/P voltage (in volts)
0.47	1.18
0.75	1.90
0.98	2.51
1.22	3.14
1.45	3.78
1.76	4.56
2.00	5.22
2.24	5.84
2.50	6.55
2.76	7.27
2.95	7.77
3.23	8.52
3.55	9.4
3.706	9.82
3.76	9.97
3.838	10.17
3.934	10.43
4.024	10.64
4.076	10.80
4.08	10.83
4.091	10.84
4.13	10.98
4.16	11.01
4.20	11.03
4.25	11.03
4.5	11.03
4.7	11.03
5.0	11.03

The maximum output obtained from the LM675 power amplifier is 11.03 volts , which is sufficient to drive the water pump. It can be seen that the power amplifier has an almost constant amplification gain of around 2.65 i.e. the ratio of o/p voltage and i/p voltage is fixed and they obey a linear relation to each other .Thus the amplifier linearly amplifies the i/p voltage till saturation. This is evident by the following graph :

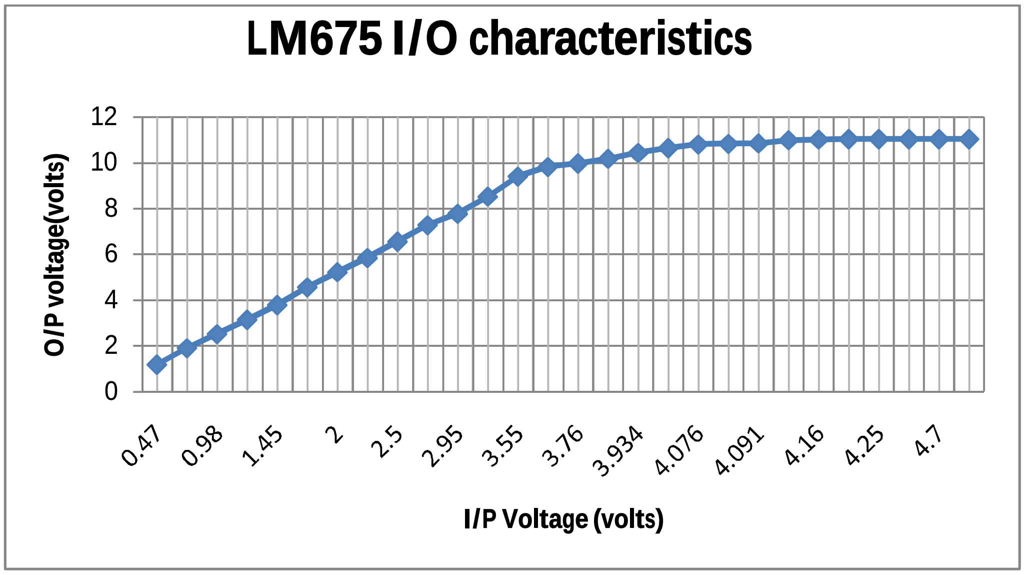


Fig 4.6 : I/O characteristics plot for LM675

4.2 INTERFACING OF DEVICES

4.2.1 Interfacing of AD7302 to ATMEGA32

The input pins of the DAC were connected to PORTD of the ATMEGA to receive the digital signal from the controller output and the write pin of the DAC was connected to the pin 0 of PORTC for receiving a periodical square wave from the microcontroller to enable/disable the DAC .

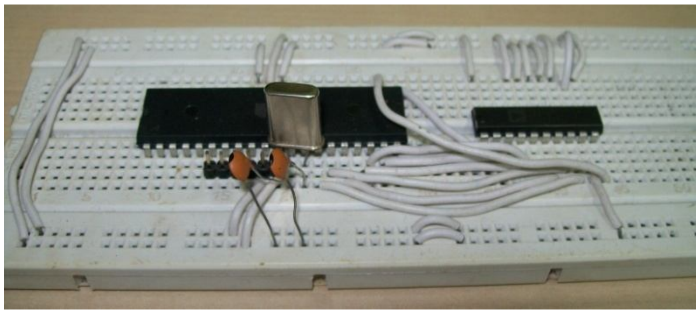


Fig 4.7 : ATMEGA32 interfaced with AD7302

4.2.2 Testing of the interfaced AD7302 and ATMEGA32 circuit with a small algorithm

The following circuit was made in the breadboard:

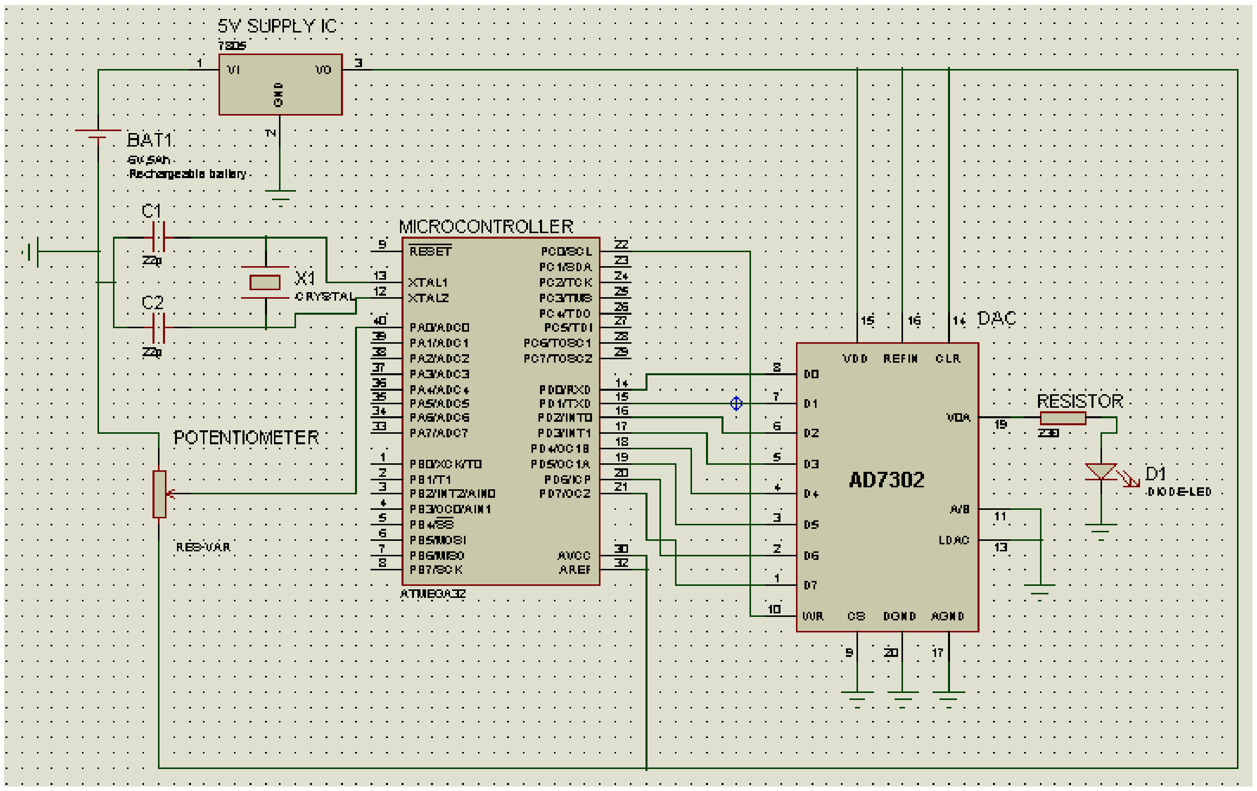


Fig 4.8 : Circuit for the dummy representation of the actual liquid level control circuit

The potentiometer which represents the ‘rotary potentiometer type liquid level sensor’ is interfaced to pin 0 of PORTA of the microcontroller which is configured as input port. The 8 pins PORTD of the microcontroller are respectively connected to the 8 input pins of the DAC AD7302 for transferring the digital value of the desired output voltage so as to operate the actuator which is represented by the LED in this circuit.

In the DAC the CS signal is permanently pulled low so as to continuously transfer data to the it. Pin 0 of PORTC of the microcontroller feeds high value to the WR signal of the DAC due to which the input data is latched to the input register. LDAC is permanently pulled low so as to transfer data continuously from the input register to the DAC register and start a conversion. Making A/B signal low stores the output value in DAC A. At the output pin A of the DAC , the LED is connected so as to see its response to the output voltage generated by the DAC. If the circuit and the code function properly then the brightness of the LED should vary proportionately with varying input to the microcontroller.

The above circuit is just a demo of the actual liquid level control circuit. This is a logical representation of it as the potentiometer is used to represent the liquid level sensor(which is nothing but a rotary potentiometer which would give input voltage as per to the level of water) and the LED in the output circuit is used to represent the pump of the actual circuit. The brightness and the ON/OFF of the LED will represent the opening and closing of the pump. In the actual circuit the controller part hardware is same but the input and output sections are different. The output of AD7302 is taken to a power amplifier LM675 to boost the DAC output current so as to be sufficient to be able to operate the pump.

The assembled hardware for the circuit in the previous page is as shown :

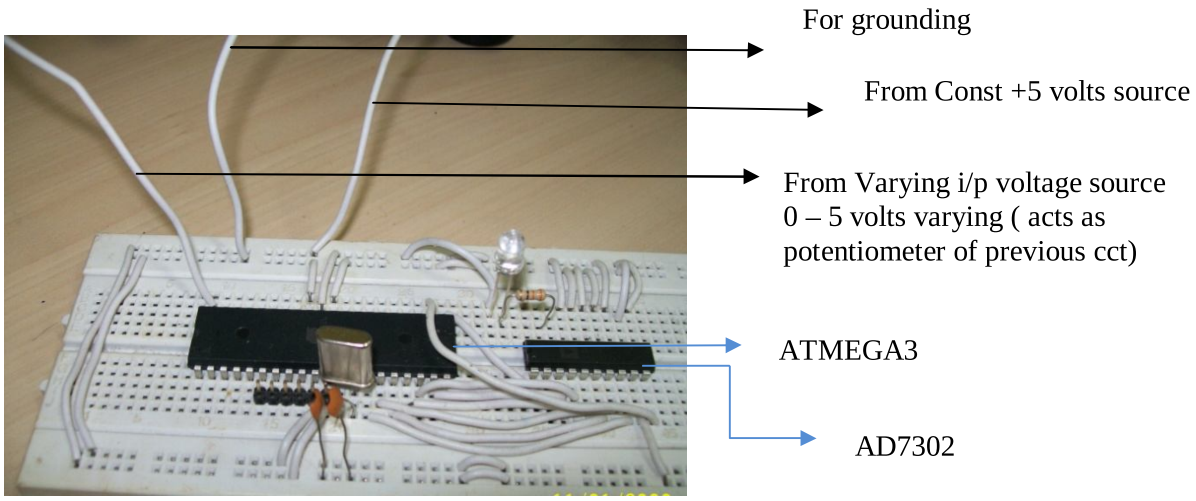

For grounding
From Const +5 volts source
From Varying i/p voltage source
0 – 5 volts varying (acts as
potentiometer of previous cct)
ATMEGA3
AD7302

Fig 4.9 : Dummy representation of the actual liquid level control circuit using LED in place of pump

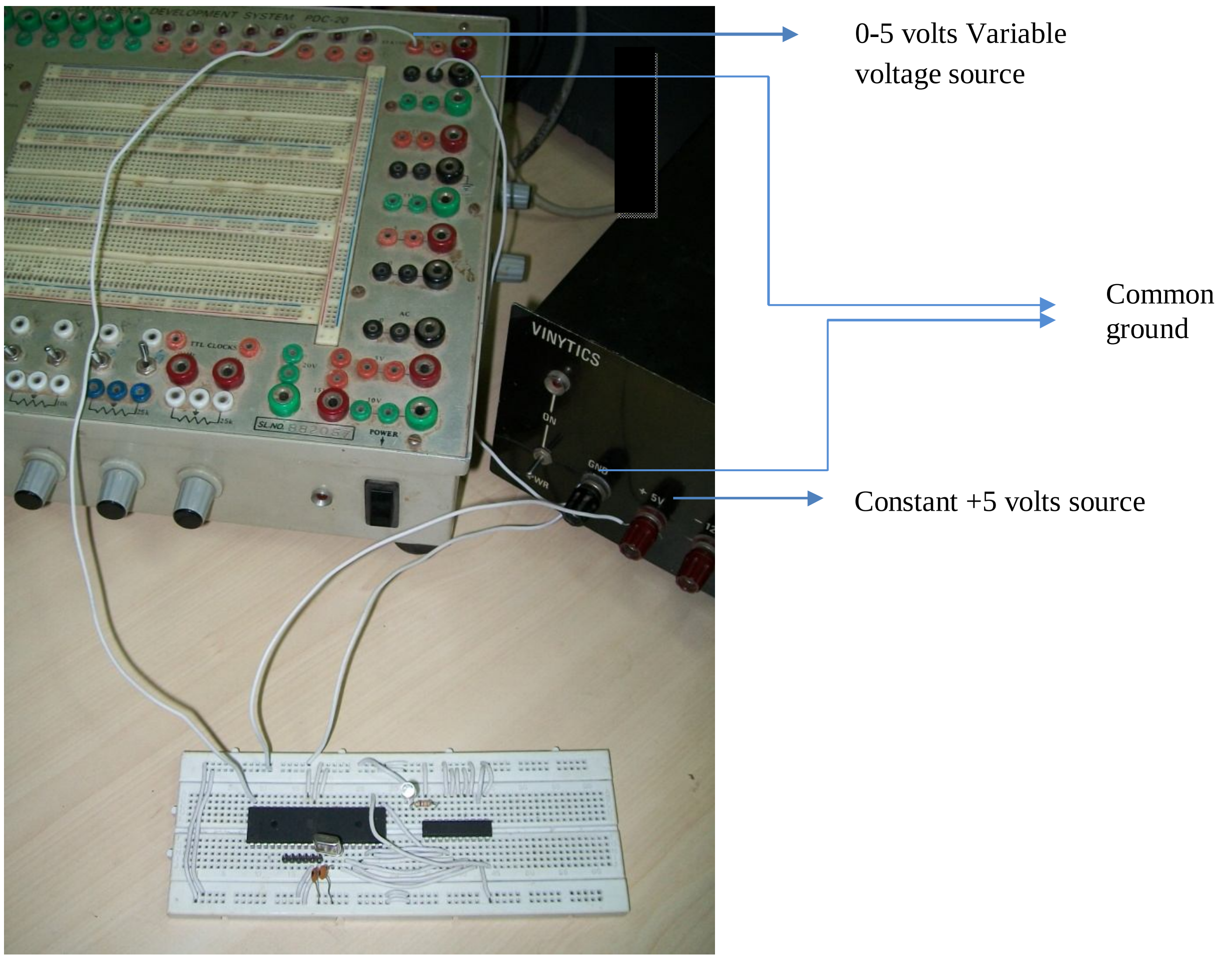


Fig 4.10 : Varying and fixed voltage supplies being fed to the circuit

The following simple control algorithm was tried at first to see the proper functioning of all the circuit components and to learn implementation of a control algorithm using a microcontroller for a real time system. The C code implementing the algorithm was compiled and fed to the microcontroller to test the working of the circuit. The code checks for the working of the circuit by glowing the LED if the input potentiometer voltage is more than the mid-value.

Algorithm:

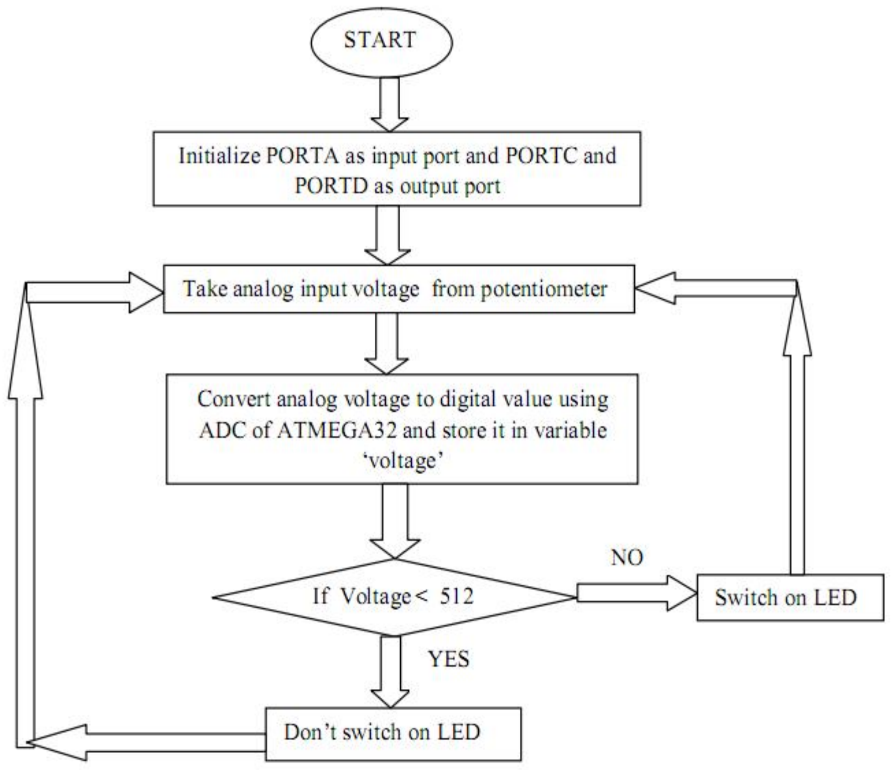
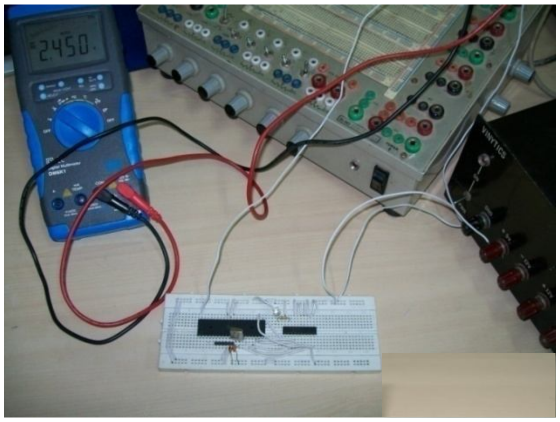

START
Initialize PORTA as input port and PORTC and PORTD as output port
Take analog input voltage from potentiometer
Convert analog voltage to digital value using ADC of ATMEGA32 and store it in variable 'voltage'
If Voltage < 512
NO
Switch on LED
YES
Don't switch on LED

Fig 4.11 : A simple control algorithm flowchart

Code:

```
#include<avr/io.h>
void main()
{
int voltage;
DDRA=0X00; // configure PORTA as i/p port
DDRC=0XFF; // configure PORTC as o/p port
DDRD=0XFF; // configure PORTD as o/p port
while(1)
{
PORTC=0X01; //disables D/A
ADMUX=0XC0; //select ADC channel,reference
             voltage,left/right justification
ADCSRA=0XC3; //enable and start A/D conversion and select
             prescaler bits for ADC sampling frequecny
while(!(ADCSRA&0X10)); // wait for conversion to complete
ADCSRA=ADCSRA|0X10; // clear interrupt flag
voltage=ADC ; // store ADC value in voltage variable
if(voltage<512) //if voltage less than mid-value
PORTD=0X00; //LED in OFF state
else
PORTD=0XFF; // LED in ON state
PORTC=0X00; //enables D/A
PORTC=0X01; //disables D/A
}
}
```

While testing this code the LED was found to respond to the algorithm.



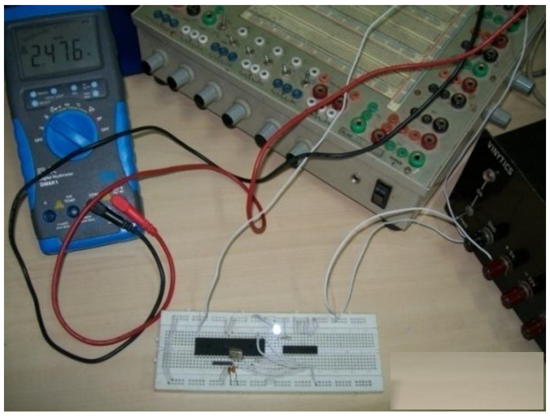


Fig 4.12: LED OFF as i/p voltage below mid value

Fig 4.13: LED ON as i/p voltage above mid-value

The glowing of the LED shows that the interfaced circuit of ATMEGA32 and AD7302 are responding correctly and hence the LED can be replaced with the LM675 circuit.

4.2.3 Interfacing of LM675 to AD7302

The output of the DAC was connected to the input of the power amplifier so that the current magnitude will be raised so as to be sufficient to drive the pump.

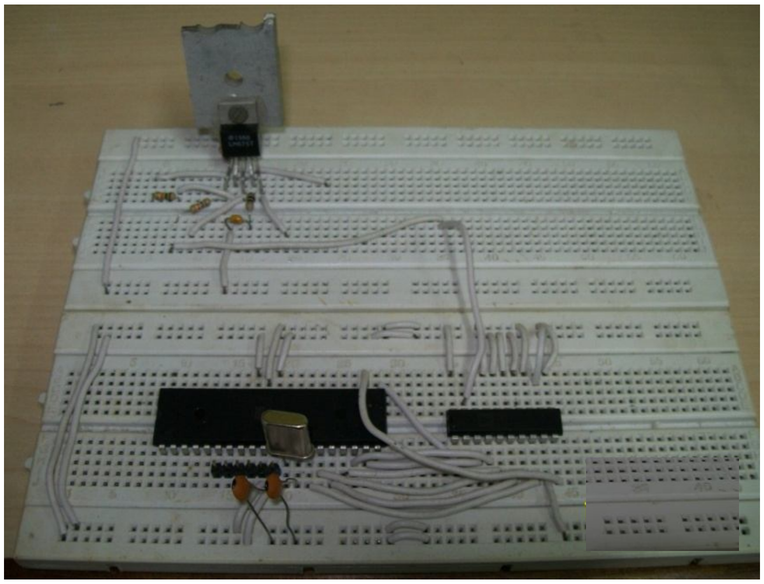


Fig 4.14 Interfacing of LM675 with AD7302

CHAPTER 5

DEVELOPMENT OF DISCRETE PI CONTROLLER

CHAPTER 5

DEVELOPMENT OF DISCRETE PI CONTROLLER

5.1 CONTROLLER REALIZATION IN DISCRETE DOMAIN

A standard equation of PID controller is

$$u(t) = k\{e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau\} \qquad (5.1)$$

where the error e(t) is the controller input and is the difference between the setpoint and plant output, and the control variable u(t) is the controller output. The two parameters are k (the proportional gain), Ti (integral time constant) , which are to be appropriately fixed by tuning.

Performing Laplace transform on (4.1), we get

$$G(s) = k\left(1 + \frac{1}{sT_i}\right) E(s) \qquad (5.2)$$

This can be written in an alternative way ,a widely used form of PI algorithm is called the 'Parallel form' which is as follows:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau \qquad (5.3)$$

which on taking its laplace transform becomes

$$G(s) = k_p E(s) + \frac{k_i}{s} E(s) \qquad (5.4)$$

We can compare Equations (4.2) and (4.4) and note that

$k_p = k$, $k_i = \frac{kT}{T_i}$

For using PI algorithm in our project ,digital implementation of PI is needed ,so we need a Z-transform of the parallel form Equation (4.3) , as z-transform is for discrete domain just like the s-transform which is for continuous domain. So by converting to z-domain ,we get

$$U(z) = \left[k_p + \frac{k_i}{1-z^{-1}} \right] E(z)$$

$$= \left[\frac{(k_p + k_i) + (-k_p)z^{-1}}{1-z^{-1}} \right] E(z)$$

$$= \frac{k_1 + k_2 z^{-1}}{1-z^{-1}} \; E(z) \qquad (5.5)$$

Where $k_1 = k_p + k_i$

$k_2 = - k_p$

Rearranging the terms in Equation (4.5) gives

$$U(z) - Z^{-1} U(z) = [\, k_1 + k_2 z^{-1}]\, E(z)$$

which when converted back to difference equation gives us a relation between the controller o/p and the recent and previous errors,as follows

$$u[k] - u[k-1] = k_1 e[k] + k_2 e[k-1]$$

$$\text{or} \quad u[k] = k_1 e[k] + k_2 e[k-1] + u[k-1] \qquad (5.6)$$

Thus we obtain a form suitable for implementation in digital controllers like microcontroller. The above form is called *velocity form* of PI algorithm [5].

5.2 DETERMINATION OF CONTROLLER COEFFICIENTS

The PI controller coefficients are determined by Ziegler-Nichols tuning method by giving a step response to the liquid level system and conducting an open loop step response test. A step of 200 decimal value whose corresponding voltage is 200 * 5 V /256 = 3.906 volts is sent to the DAC from the microcontroller. The height of the water inside the tank is recorded in real time in terms of the voltage across of the level sensor with the help of a DAQ 6221 card and the Labview software. DAQ 6221 card is a small electronic card which is connected to the parallel port of the PC. The card has sensors to measure physical quantities such as intensity, sound level, voltage, humidity, and temperature of different systems. Labview software runs on a PC and can be used to record the measurements of the DAQ 6221 card in real time. The software includes a graphical option which enables the measurements to be plotted [1].

The following figure shows the circuit diagram and hardware set up for conducting the open loop step response test.

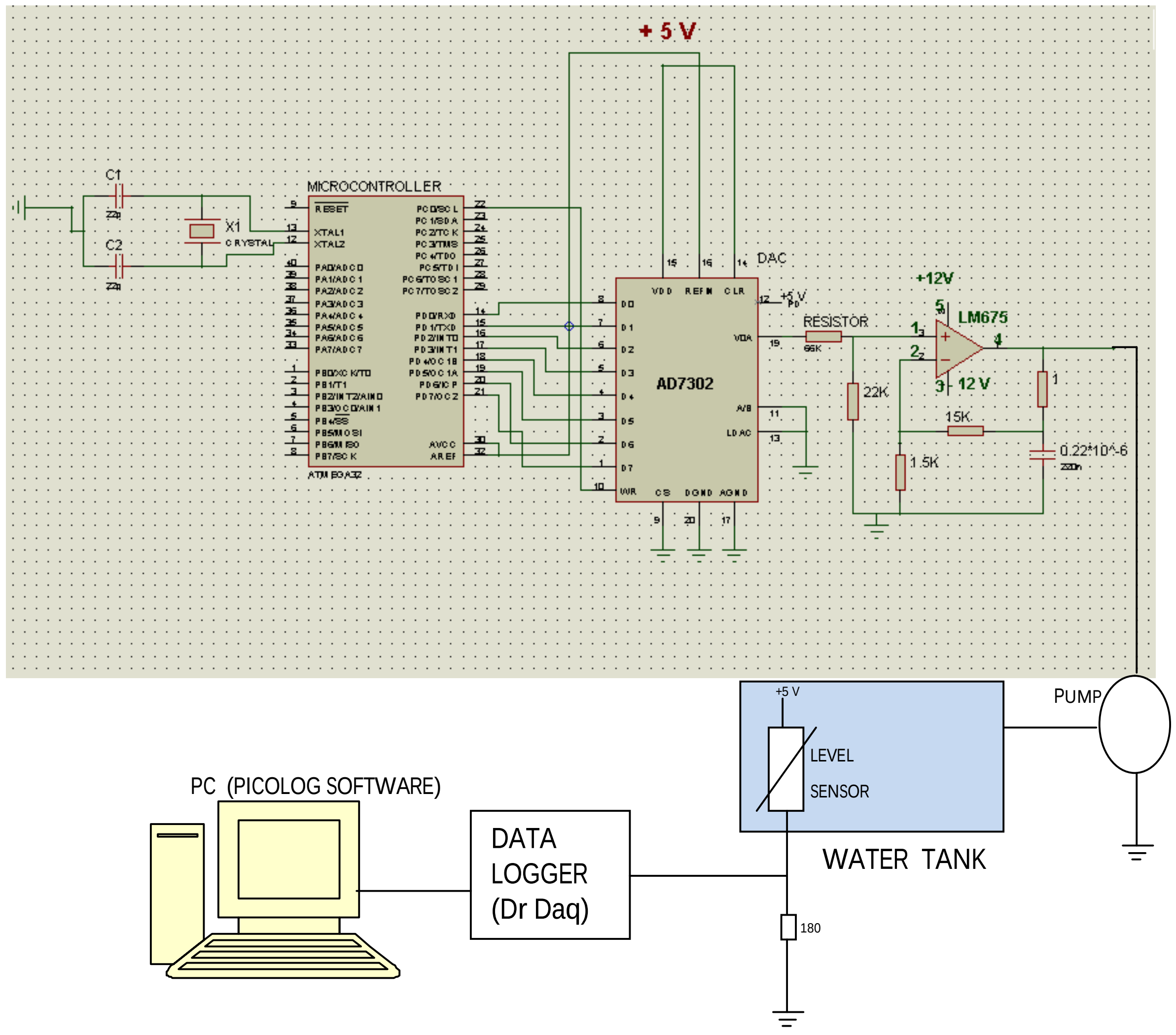


Fig 5.1 : Circuit diagram and Hardware set up to record the open loop step response

The following is the code for sending the step signal to the liquid level system. As explained in section 3.1.3 the PORTs are first configured as i/p or o/p port. Then the D/A is disabled by setting its WR pin so that the value sent to the D/A is latched. A step signal of 200 is sent and the D/A is enabled by clearing the WR pin so that the step signal can be transferred to the DAC. Then again the D/A is disabled to latch this value so that it doesn't accidentally change.

```
#include<avr/io.h>
void main()
{
DDRC=0XFF; //configure PORTC as o/p port
DDRD=0XFF; //configure PORTD as o/p port
PORTC=0X01; //disable D/A
PORTD=0XC8;  //send step signal of value 200
PORTC=0X00; //enable D/A
PORTC=0X01; //disable D/A
while(1)
{
}
}
```

The following figure shows the output of the LABVIEW program for recording the step response.

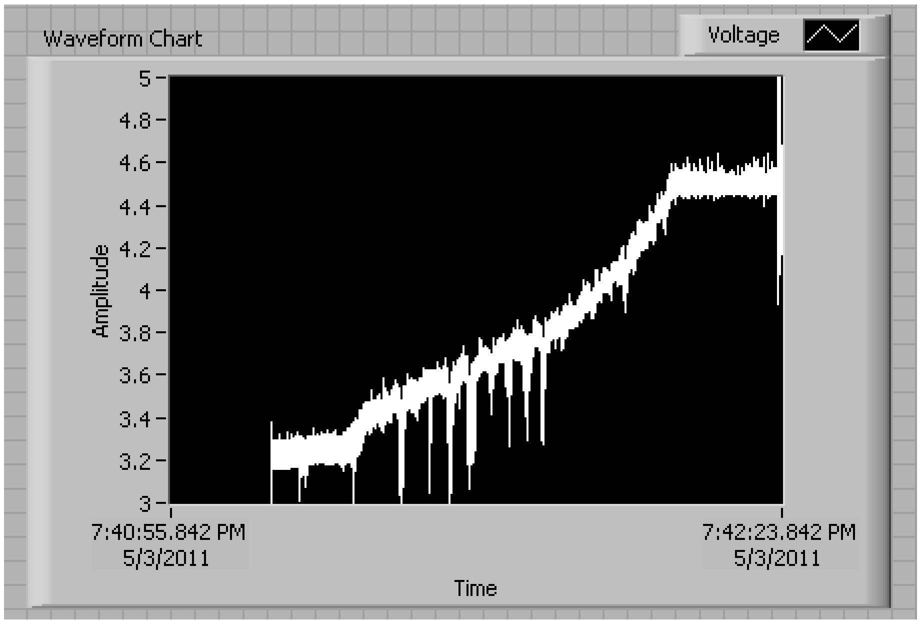


Fig 5.2 : Output of the open loop step response test

The above curve is transferred to MATLAB and tangent is drawn to the curve such that it has the maximum possible slope and its x-intercept determines the PI parameter T_D (the delay constant). The rise in voltage from starting to stable point determines the PI parameter K and the time difference between the intersection of the tangent with the line passing through the stable part of the curve and T_D would give the PI parameter T_L (the time constant of the system).These can be experimentally determined for the liquid level system as follows:

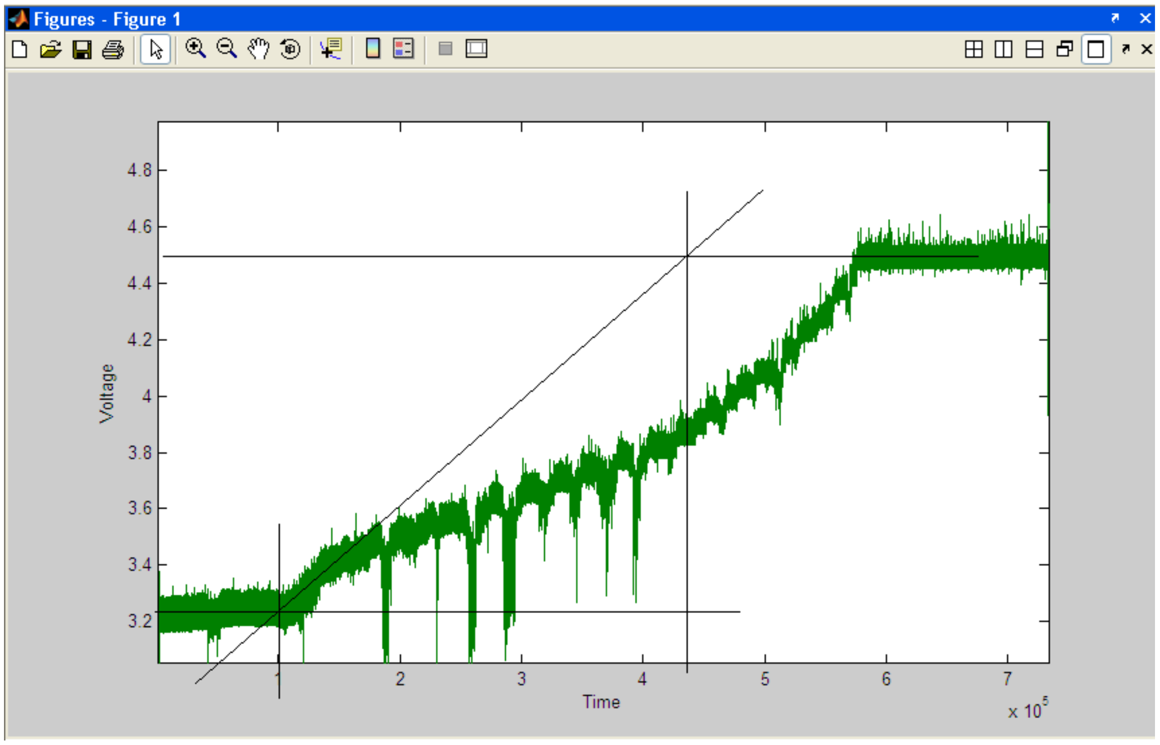


Fig 5.3 : Experimental determination of PI parameters from the open loop step response test

From the above plot we get $T_D = 2.53$ sec , $T_L = 30.12$ sec and K = 5.64 .

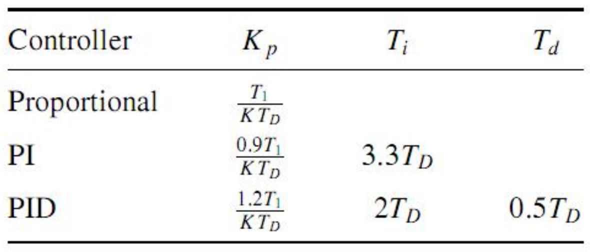
Thus the transfer function of the system is :

$$G(s) = \frac{5.64\ e^{-2.53}}{1+30.12s}$$

As sampling time is normally chosen less than one-tenth of the system time constant i.e. (30.12 sec / 10) in this case , so we chose sampling time of 0.1 second in this project.

The PI coefficients K_p and T_i can be determined by open loop Ziegler Nichols settings as shown below in the following Table :

Table 5.1 – Open loop Ziegler Nichols settings[1]



Controller	K_p	T_i	T_d
Proportional	$\frac{T_1}{KT_D}$		
PI	$\frac{0.9T_1}{KT_D}$	$3.3T_D$	
PID	$\frac{1.2T_1}{KT_D}$	$2T_D$	$0.5T_D$

Thus we obtain $k_p = 1.9$ and $T_i = 8.345$.

Hence we can write the PI equation given in equation (5.6) as follows :

$$u[k] = k_1\, e[k] + k_2\, e[k-1] + u[k-1] \qquad (5.7)$$

Where $k_1 = k_p + k_i = k_p + k_p\, T/\, T_i = 1.9 + 1.9*0.1/8.345 = 0.02277$

$k_2 = -\, k_p = -1.9$

5.3 DISCRETE TIME PI ALGORITHM

The PI equation (5.7) can be directly implemented in microcontroller following the below algorithm

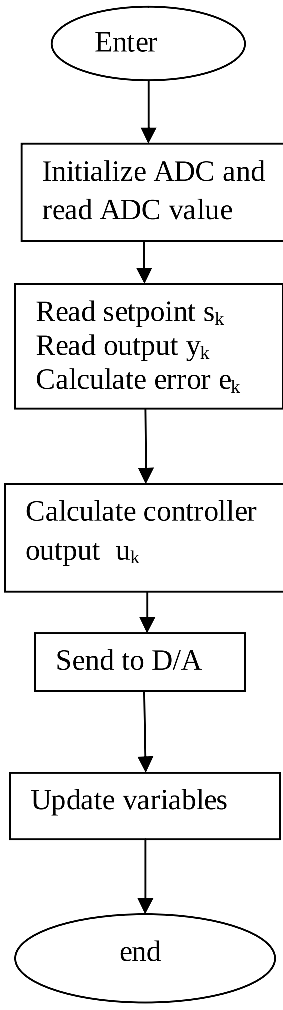


Fig 5.4 : Flowchart for the discrete PI algorithm[1]

The code for implementing the above algorithm is as follows :

```
double e,e1,u,delta_u
k1 = kp + ki;
k2 = - kp;
void pid()
{
e1 = e  ;// update error variable
y = read ADC; // read variable from sensor o/p
e = setpoint – y ; // compute new error
delta_u = k1 * e + k2 *e1 ;
u = u + delta_u ;
if(u> UMAX ) u = UMAX ; //limit the controller o/p to DAC
range
if (u<UMIN) u = UMIN ;
write DA(u) // send to DAC chip
}
```
[5]

CHAPTER 6

IMPLEMENTATION OF CONTROLLER ALGORITHM

CHAPTER 6

IMPLEMENTATION OF CONTROLLER ALGORITHM

The circuit diagram and the hardware set up for the closed loop liquid level control sytem is as follows :

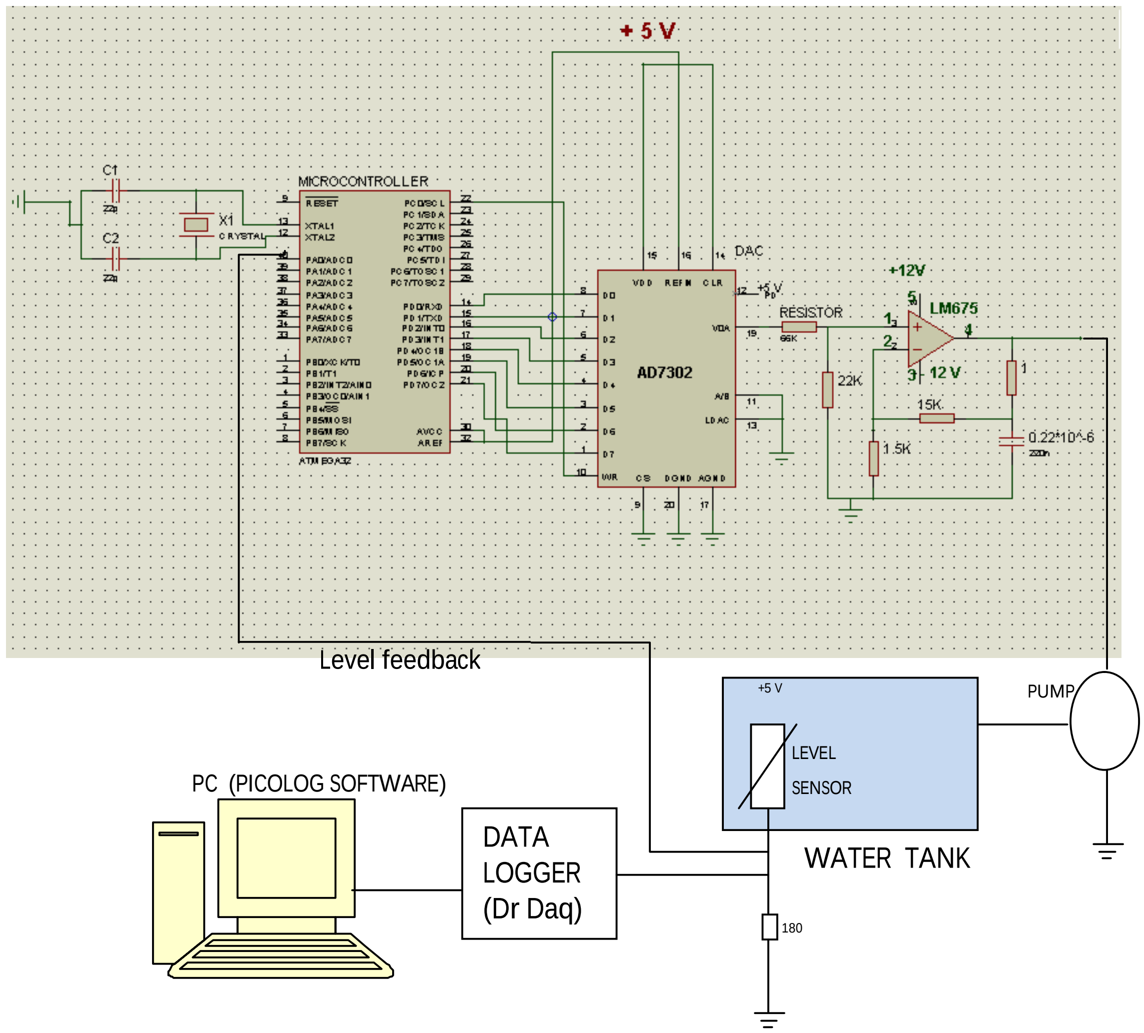


Fig 6.1 : Circuit diagram and hardware set up for the closed loop liquid level control system

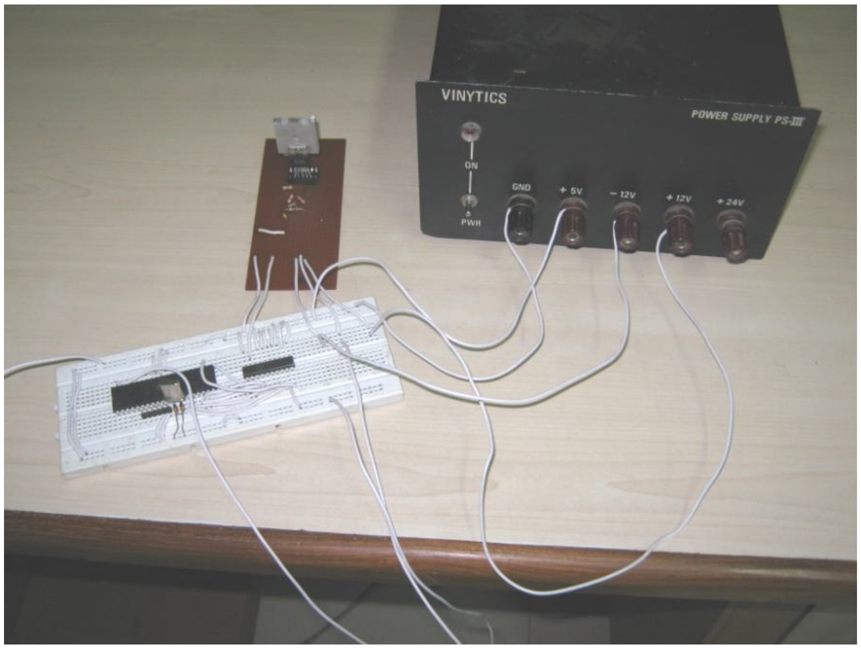


Fig 6.2 : A snapshot of the Circuitry used in the project

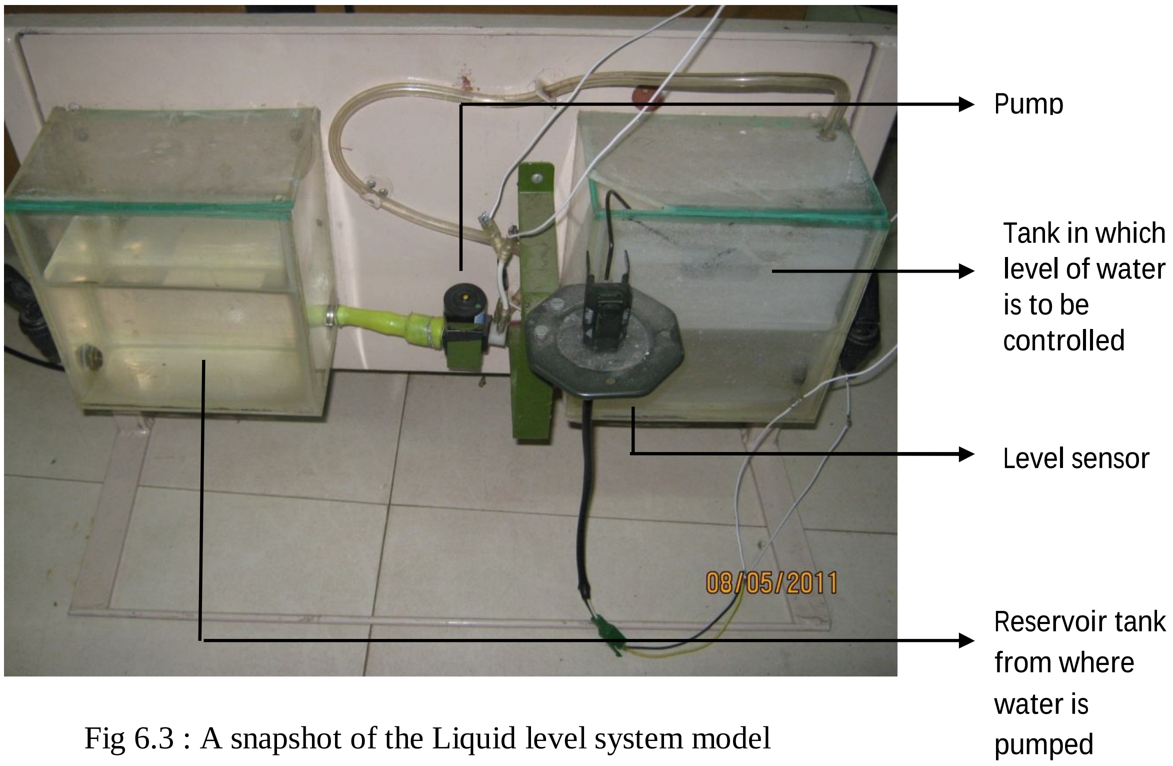

Pump
Tank in which level of water is to be controlled
Level sensor
Reservoir tank from where water is pumped
08/05/2011

Fig 6.3 : A snapshot of the Liquid level system model

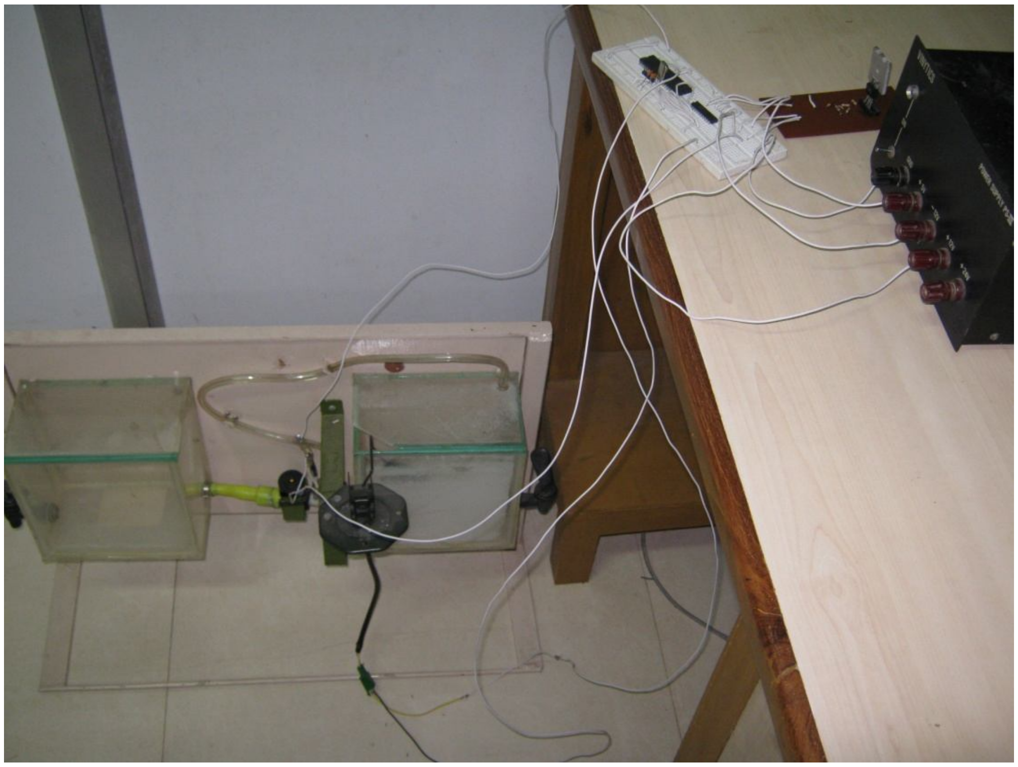


Fig 6.4 : A snapshot of the Liquid level system interfaced with the control circuit and being controlled implementing the controller algorithm

The microcontroller operates at 1 MHz frequency. The ADC channel 0 is connected to the output of the level sensor of the plant (y). The PORTD output of the microcontroller is connected to AD7302 type D/A converter. The output of the DAC is connected to the LM675 power amplifier which drives the pump. The sampling interval is 0.1s (100ms) and the timer interrupt service routine is used to obtain the required sampling interval.

The program for the liquid level system controller is written following the below algorithm :

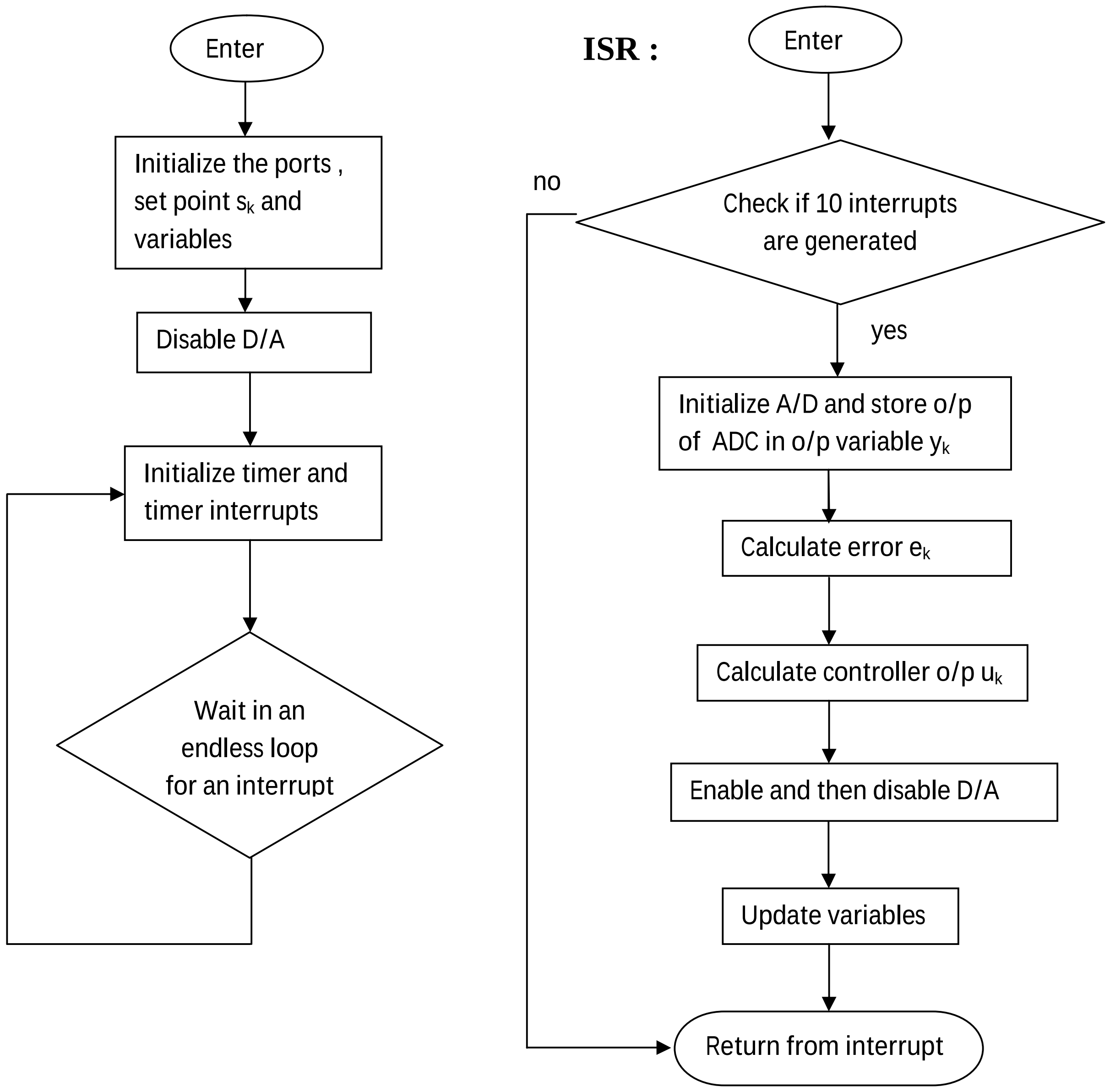

Enter
Initialize the ports , set point s_k and variables
Disable D/A
Initialize timer and timer interrupts
Wait in an endless loop for an interrupt
ISR :
Enter
Check if 10 interrupts are generated
no
yes
Initialize A/D and store o/p of ADC in o/p variable y_k
Calculate error e_k
Calculate controller o/p u_k
Enable and then disable D/A
Update variables
Return from interrupt

Fig 6.5 : Flowchart for the liquid level control system program

In the beginning of the main program, the controller parameters are initialized. The program consists of the functions Initialize_Timer, Initialize_Read_ADC, and the interrupt service routine (ISR). The timer is initialized in such a way that it interrupts every 10 ms. As ISR routine is entered ,it is first checked if 10 interrupts are over i.e 100ms is lapsed(ensuring that controller sample time is 100ms) and then the Initialize_Read_ADC function is called which initializes the ADC and reads the analog i/p from the level sensor and after converting it to digital value stores it in variable y_k .The error e_k is then calculated and the PI algorithm is then implemented. The controller output u_k is sent to the DAC by enabling the DAC and also taking into consideration that the D/A converter is limited to full scale, i.e. 255. After sending an output to D/A, the DAC is disable to latch the current value so that it doesn't change accidentally. The variables are then updated and at the end the ISR routine re-enables the timer interrupts and the program waits for the occurrence of the next interrupt[1].

The program is as follows :

```
#include<avr/io.h>
#include<avr/interrupt.h>
#include<stdint.h>
#include<stdio.h>

volatile unsigned int time_now=0;
float kp,b,uk_1,ek_1,ek,sk,yk,wk,T,Ti;
unsigned int uk;

/*The following function initializes the timer0 so that
interrupts can be generated at 10ms intervals */
void Initialize_Timer(void)
{
TCCR0=0X0B; //CTC mode is selected and 1MHz/64 = 15.6 kHz
             timing frequency is chosen so that counter TCNT0
             increments every 64µs
OCR0=0x9B;// compare value is 155 in decimal
TCNT0=0X00;// counter starts from 0
asm("sei") ; // global interrupt is enabled
TIMSK=1<<OCIE0;//output compare interrupt is enabled
}
```

```
/* The following function reads data from A/D converter and
stores it in variable yk */
void Initialize_Read_ADC(void)
{
ADMUX=0XE0;  //selects the channel and reference voltage for
              A/D conversion
ADCSRA=0XC3;  //starts conversion & selects 1MHz/8  = 125kHz
                as the clock for sampling
while(!(ADCSRA&0X10)); //wait till conversion completes
ADCSRA=ADCSRA|0X10;//clear the AD interrupt flag
yk=ADCH; //sensor output in V is stored in yk
}

/* The following function is the ISR and program jumps here
every 10ms */
ISR (TIMER0_COMP_vect)
{
TCNT0=0X00; // reload the timer counter for next interrupt
TIFR=0X02;// clear timer overflow bit by setting it so that
             next timer overflow interrupt can
             be detected
time_now++; //counts number of interrupts
if(time_now==10)  //ADC is started every 100ms , so 100ms
                    becomes the sampling time
 {
 time_now=0;
 Initialize_Read_ADC();
  ek = sk - yk; //error is calculated
  wk = kp*ek + b*ek - kp*ek_1 + uk_1;//PI algorithm
  if (wk > 255) //exceeding 8-bit value,so limit it to
                    maximum 8-bit value
  uk=255;
  else
  uk=(unsigned int)wk;//converts the float to int
  PORTD=(unsigned int)uk;
  PORTC=0X00; //enable D/A
  PORTC=0X01; //disable D/A
  ek_1=ek;
  uk_1=uk;
  }}

/* The main program initializes the variables, setpoint, DAC
etc and then waits in an endless loop for timer interrupts to
occur every 100ms */
int main(void)
{
  DDRC=0XFF;
  DDRA=0X00;
  DDRD=0XFF;
  kp = 1.9;
  T=0.1;
  Ti=8.345;
  b =  kp*T/Ti;
  uk_1 = 0.0;
  ek_1 = 0.0;
  sk = 200;  //=3.830
  PORTC=0X01; //disable D/A
  Initialize_Timer();
  for(;;); // wait for an interrupt
 }
```

CHAPTER 7

RESULTS AND CONCLUSION

CHAPTER 7

RESULTS AND CONCLUSION

7.1 RESULTS

The closed loop response of the liquid level control system is recorded in LABVIEW as follows :

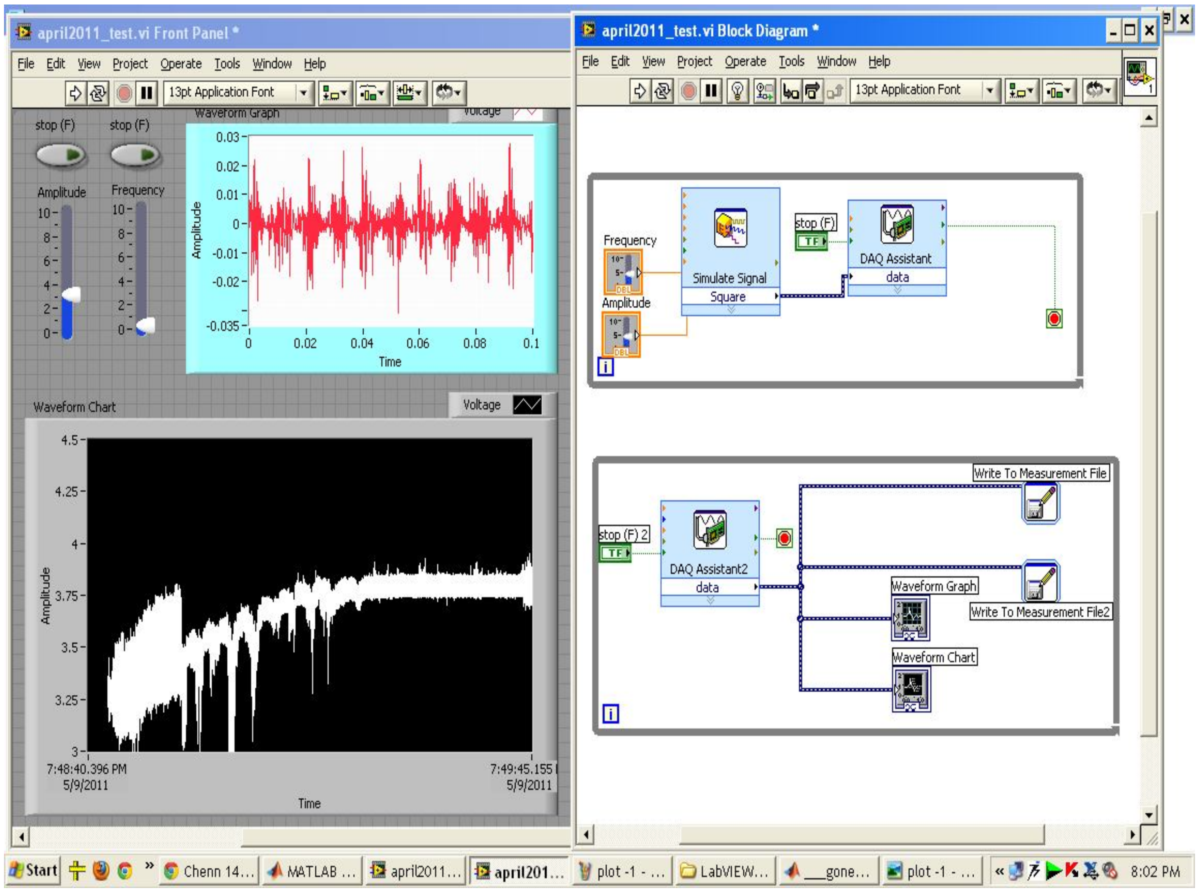


Fig 7.1 : Closed loop response of the liquid level control system recorded in LABVIEW

The output response of the PI controlled liquid level closed loop system is as follows :

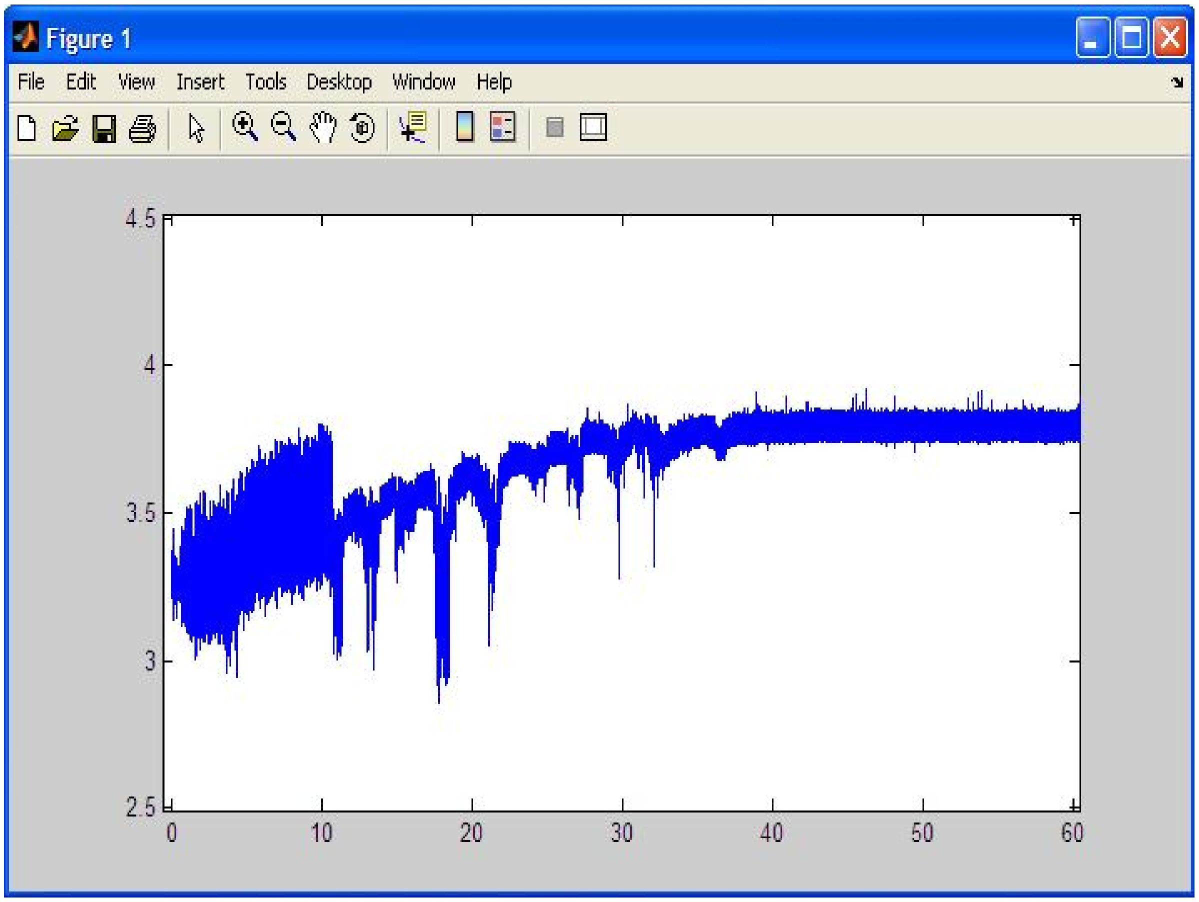


Fig 7.2 : closed response of PI controlled liquid level system

It can be seen that the liquid stably reaches its desired height which corresponding to the level sensor output voltage of 3.8 volts is set as the set point and stops there without increasing further. On draining out the water from the tank, the pump starts again to fill the water till the set point and then stops again. So we can see that the PI algorithm is properly followed and hence we have successfully designed an automatic digital controller for a liquid level system. There is no need for manually switching on or off the pump as it can be automatically controlled.

We can also compare the simulated result and the experimental result to see if the controller behaves as expected with almost same PI coefficients used in both simulation and experiment. The compared results are as follows :

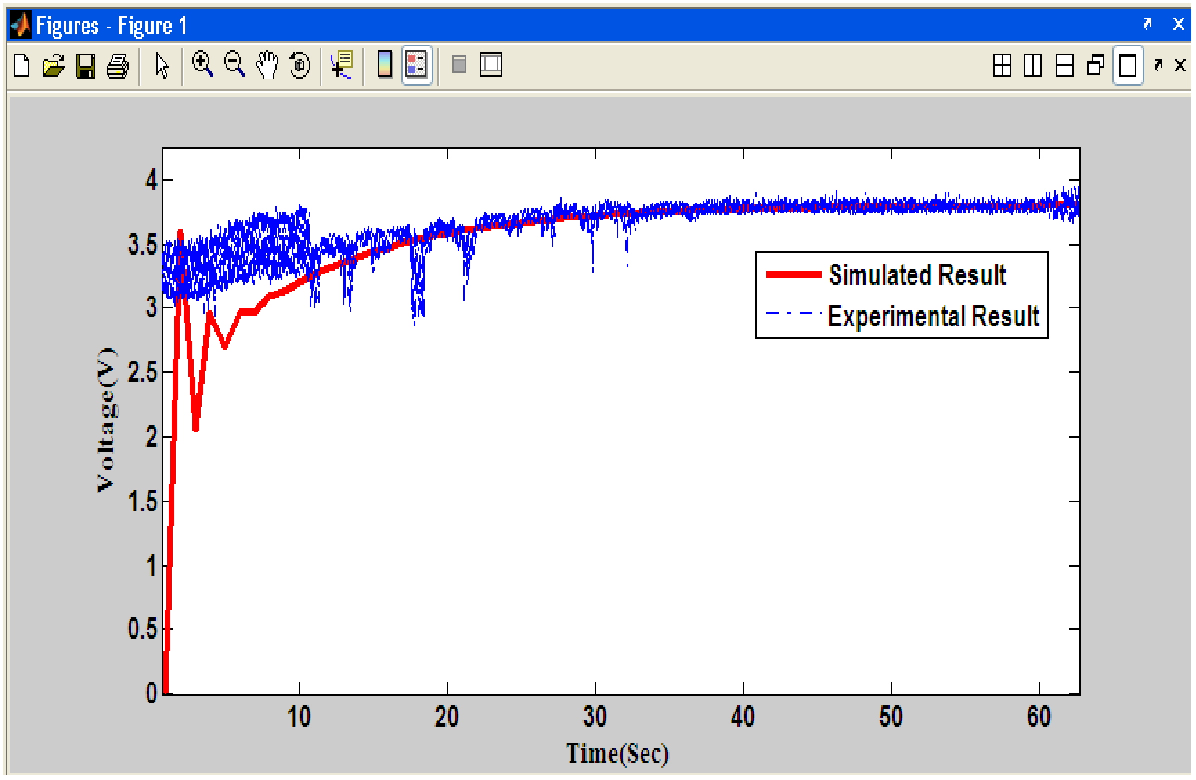


Fig 7.3 : Comparison of experimental and simulated results

7.2 CONCLUSION

- The liquid level digital system is controlled using a simple PI controller.
- The ATMEGA32 microcontroller , AD7302 DAC , LM675 power amplifier are quite inexpensive and hence we could design the controller at a cheap price.
- The testing of the devices – the microcontroller, the DAC, the power amplifier gave approximately expected results and thus give correct results during the implementation of the controller algorithm.
- The whole controller set up gave similar simulated and experimental results thus resulting in fine and accurate control of level of liquid at the desired height.

- Selecting ATMEGA32 helps in 10 times faster execution(compared to conventional microcontrollers like 8051) and more effective performance as its each instruction takes only one clock cycle for execution and multiplication process takes 2 clock cycles for execution , whereas other microcontrollers take more number of cycles.It also reduces the need of using external ADC thus simplifying the circuit.
- Selecting AD7302 helps for accurate performance as it has the WR pin which when pulled high, latches its value thus preventing it from accidentally changing.
- The system can be of use in industrial applications for accurately measuring and controlling the level of liquid as needed in boilers in power plants , petroleum industries ,pharmaceutical or chemical industries etc.

  It can also be used for household applications for preventing overflowing of tanks thus saving electricity and water.

REFERENCES

1. Ibrahim Dogan . *"Microcontroller Based Applied Digital Control"* . Chichester : John Wiley & Sons, Ltd. , 2006
2. Mittal Shivani . "*Liquid level digital control system*". B.Tech Project, National Institute of Technology, Rourkela : 2009
3. Barrett Steven F. and Pack Daniel J. *"Atmel AVR Microcontroller Primer: Programming and Interfacing"* . Morgan & Claypool , 2008
4. Mazidi Muhammad Ali, Naimi Sarmad, Naimi Sepher.*"The AVR Microcontroller and embedded and embedded systems using assemble and C"*. Upper Saddle River : Prentice Hall, 2011
5. Dr. Toochinda Varodom . *"Digital PID Controllers"*. http://www.dewinz.com , July 2009
6. Atmel AVR . *ATMEGA32 datasheet* , 2010
7. Analog devices. *AD7302 datasheet* , Norwood : Analog Devices , 1997
8. National Semiconductor . *LM675 datasheet* , National Semiconductor corporation, 2004
9. http://www.shridhan.com/
10. http://www.nationalmagnetic.com/applications.html
11. Kamal Ibrahim.*"8-bit Digital to Analog converter(DAC) Using R/2R resistor network"*, http://www.ikalogic.com/dac08.php
12. *"AVR Simulation with the ATMEL AVR Studio 4"* , Purdue university , 2005, pp 1 – 56
13. Huang Han-Way. *"MC68HC12 an introduction : software and hardware interfacing"* . New York : Thomson Learning , 2003
14. Stefanovic Miladin, Cvijetkovic Vladimir, Matijevic Milan, Simic Visnja, *"A LabVIEW-Based Remote Laboratory Experiments for Control Engineering Education"*, Wiley Interscience, Wiley Periodicals , 2009, DOI 10.1002/cae.20334
15. Bera S.C., Ray J.K. , Chattopadhyay S.,*"A Novel Technique of Boiler Drum Level Measurement using Non-Contact Capacitive Sensor"* , IE(I) Journal-ET, Vol 84, July 2003

APPENDIX

List and cost of components is given below. The components were procured from different places from India. The prices given are approximate and vary depending on manufacturer and place of purchase.

S.No.	Components name	Quantity	Price (in Rs)
1.	ATMEGA32 Microcontroller	1	230
2.	AD7302 Digital-to-Analog Converter	1	320
3.	LM675 power amplifier	1	340
4.	Water pump(12V, 3A)	1	200
5.	Rotary potentiometer level sensor	1	300
6.	Crystal oscillator	1	25
7.	Resistor(68k, 22k ,15k,1.5k,1,180)	1 each	0.5 each
8.	Capacitor(33pF,33pF,0.22µF)	1 each	2 each
9.	Wires for connection	As per to need	10

Sum total cost of all components = Rs 1434

This is the approximate total cost of major components needed in the Liquid level control system. Other components like for the stand , water tank , pipe for water flow etc can be procured from scraps and constructed in situ and hence the price of those components are not included.