Modern Day Electrical and Communication Systems Applied on Residential Households

By:

Fady Gamal Abdel Fattah

Mohammed Tohamy Mohammed Rizq

Noha Mohammed Hamed Asr

A Project Graduation Document Submitted in Partial Fulfillment of the Requirements for the BSc. Degree in Electrical Engineering
(Electronics and Communication Engineering)

Under the Supervision of

Dr. Eng. Mohamed Amr Mokhtar

Faculty of Engineering - Alexandria University

Faculty of Engineering, Alexandria University

Alexandria, Egypt

2024

Dedication

We would like to thank our families and loved ones for the love and support provided. This journey wouldn't have been possible without the encouraging environments we were blessed with. Our families have been our pillars of strength, offering unwavering support through the challenges and triumphs alike.

The sacrifices made by our parents and loved ones have not gone unnoticed. They have been our constant source of motivation, pushing us to reach for our dreams and never give up. Their belief in us has been a driving force, fueling our determination to succeed.

Moreover, the nurturing environments our families have cultivated have played a pivotal role in shaping our characters and instilling in us the values of hard work, perseverance, and resilience. These virtues have been instrumental in helping us navigate the academic landscape and overcome obstacles along the way.

We are forever grateful for the love, encouragement, and sacrifices made by our families. Their unwavering support has been the foundation upon which we have built our successes, and we owe a debt of gratitude that can never be fully repaid.

Acknowledgment

We are honored to have had Dr. Amr Mokhtar as our supervisor for this graduation project. His guidance, expertise, and unwavering support have been invaluable throughout the entire process. Dr. Mokhtar's dedication to his students and his passion for the subject matter truly sets him apart as an exceptional mentor.

We would like to thank him and the body of the faculty for their contributions to our academic journey. The knowledge and skills we have acquired during our time at this esteemed institution have laid a solid foundation for our future endeavors. The professors' commitment to excellence has inspired us to strive for greatness in all our pursuits.

Furthermore, we extend our gratitude to the administrative staff and support personnel who have played a crucial role in ensuring a seamless academic experience. Their tireless efforts behind the scenes have not gone unnoticed, and we are truly appreciative of their contributions to our success. And Specially to Dr. Noha Korany for the supportive role she played throughout our journey.

We also thank The Information Technology Institute (ITI), and The I Make Technology School (IMT School) for providing the particular knowledge and experience required for our journey into this project.

Table of Contents

Abbreviations

Table 1-1: Abbreviations

Abbreviation	Description
AC Unit	Air Conditioning Unit
ADC	Analog Digital Converter
APP	Application
AVR	Alf and Vegard's RISC processor, also Advanced Virtual RISC
DIO	Digital Input Output
EEPROM	Electrically Erasable Programmable Read-Only Memory
HAL	Hardware Abstraction Layer
I2C	Inter-Integrated Circuit
INT	Interrupt
LCD	Liquid Crystal Display
LIB	Library
MCAL	Microcontroller Abstraction Layer
RISC	Reduced Instruction Set Computers
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
STD	Standard
UART	Universal Asynchronous Receiver / Transmitter

Abstract

One major aspect that can have a great impact on our mentality and lifestyle is the house hold, it's the place we spend most time at, and maybe most of our efforts, between cleaning and maintenance and generally making sure that it's a smooth sailing ship. And sometimes, we can't do it all efficiently. In various areas we can use a hand managing the household and maintaining it, and this is where our idea takes place, providing a solution for automating some of the basic day to day operations that take place withing the household.

We demonstrate that, using microcontrollers and a communication line set up between them, in addition to sensors to interact with the environment, and having integration with the main electrical grid of the house, to control various devices, we can transform a regular home to a better more versatile and helpful asset, all with minimal cost and effort.

The fact that we can achieve this functional result opens up the way for us to imagine the possibilities and opportunities when implementing this solution on a wide range area that can all be interconnected and to thrive together as well as individually being able to keep each single household safe and functional.

List of Figures

List of Tables

CHAPTER 1: Introduction

1.1 PROBLEM STATEMENT

1.1.1 Typical households can't guarantee safety

Countless unfortunate incidents, like fires, take place each and every day in residential areas, these incidents, these incidents can be prevented, or at least contained to a minimum if caught early. This can be made possible with a dedicated fire-fighting system. And thus, absence of safety measures leads to loss of valuables and lives.

1.1.2 Repetitive tasks waste time and effort

Tending to daily chores and tasks might be necessary, but going about it in the wrong way wastes time, effort and even money. Sometimes a system can be put in place to automate away these exhaustive tasks and take the monotony out of it.

1.1.3 Manual cognition is unreliable

When it comes to creative work, a mindful approach is always preferred, but when relying on a person to be awake and aware all the time for and important task can be unreliable, and unproductive. It's always preferred to establish systems for predefined tasks to be done consistently and efficiently.

1.2 PROJECT AIMS

1.2.1 Integrate a modern electronic system to a household

This project aims to combine a full-fledged control system with a regular household facility to transform it to a better and more advanced living vessel, one that can offer smart and adaptive features and services to its residents, all whilst keeping them and the facility as a whole in the best and safest condition possible. This involves the usage of microcontrollers, sensors, actuators, and a complete integration with the power grid of the house and its available machines and appliances.

1.2.2 Facilitating day to day life

Utilizing the system that was integrated with the household, this project aspires to make everyday life easier and more enjoyable, all while automating repetitive tasks or processes that don’t require complete focus and metal effort, freeing up the residents to tend to their interests and life demands and minimizing manual labor required each day.

1.3 APPLICATIONS

Target application can be sorted in several categories:

1.3.1 Security

The methods put in place to keep the household secure and prevent unwanted parties from accessing the premises.

This includes a complete security system.

1.3.2 Safety

Keeping the house and its residents safe in case of fires or life-threatening situations.

This includes a full fire monitoring and containing system.

1.3.3 Convenience

Providing a better and smoother experience for everyday life tasks.

This includes an adaptive air conditioning system.

1.3.4 Automation

Preserving time and effort by automating repetitive manual labor carried out throughout the house hold.

This includes automatically adjusting the house lighting conditions according to the current time of the day.

CHAPTER 2: System Architecture and Feature

Figure 2-1: Conceptualization of modern household

2.1 PROVIDED FEATURES TO THE SYSTEM

2.1.1 Power Grid Control

Having control over the whole power grid of the house, the system can precisely choose what appliances to turn on or off, it can also choose to feed power to the household or to cut it all off entirely in case of emergency.

2.1.2 External Communication

The system has the ability to conduct communication outside the bounds of the household, this includes sending notifications to the owner of the house or establishing a hotline call with Local Law Enforcement Department and Fire Fighting Department.

2.1.3 Environment Monitoring

Utilizing various sensors, the system has the ability to monitor the surrounding environment in different methods, whether the temperature rises or falls or the humidity changes or other environmental changes occur, it’s all measured, processed, and relevant actions are taken in return.

2.1.4 Day Cycle Awareness

The system is made aware of the time of day at all times, this helps it make relevant decisions and act according to the current phase of the day, as it can act differently whether the time is in the morning, afternoon, evening, or night.

2.2 SYSTEM ARCHITECTURES

The system involves of mainly two Microcontrollers, communicating with each other through SPI communication protocol, and between them there's a multitude of sensors and controls available to monitor the environment and take appropriate actions when necessary.

2.2.1 High level overview of the system architecture:

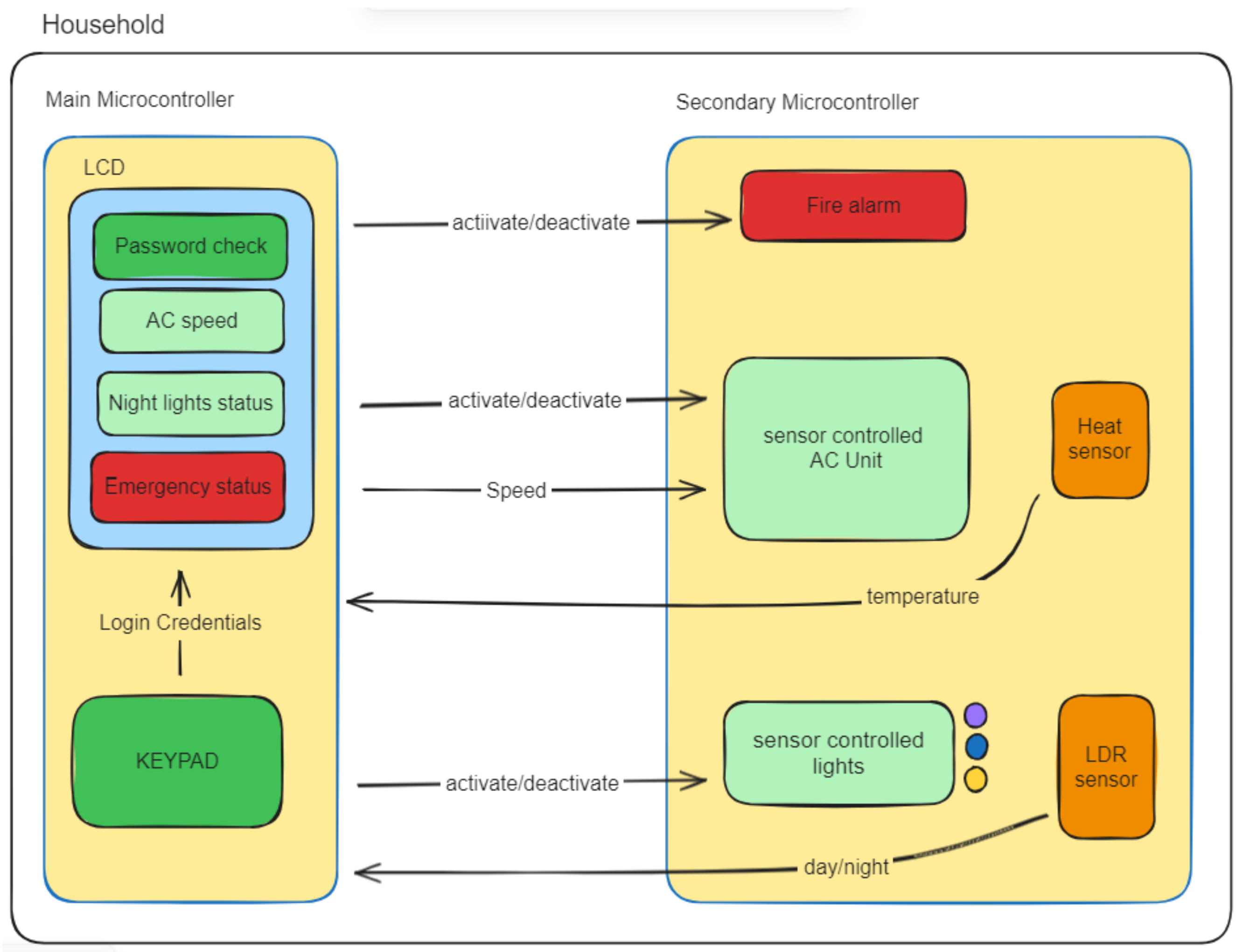


Figure 1-1: System architecture

2.2.2 Implementation of the systems architecture using simulation software (Proteus):

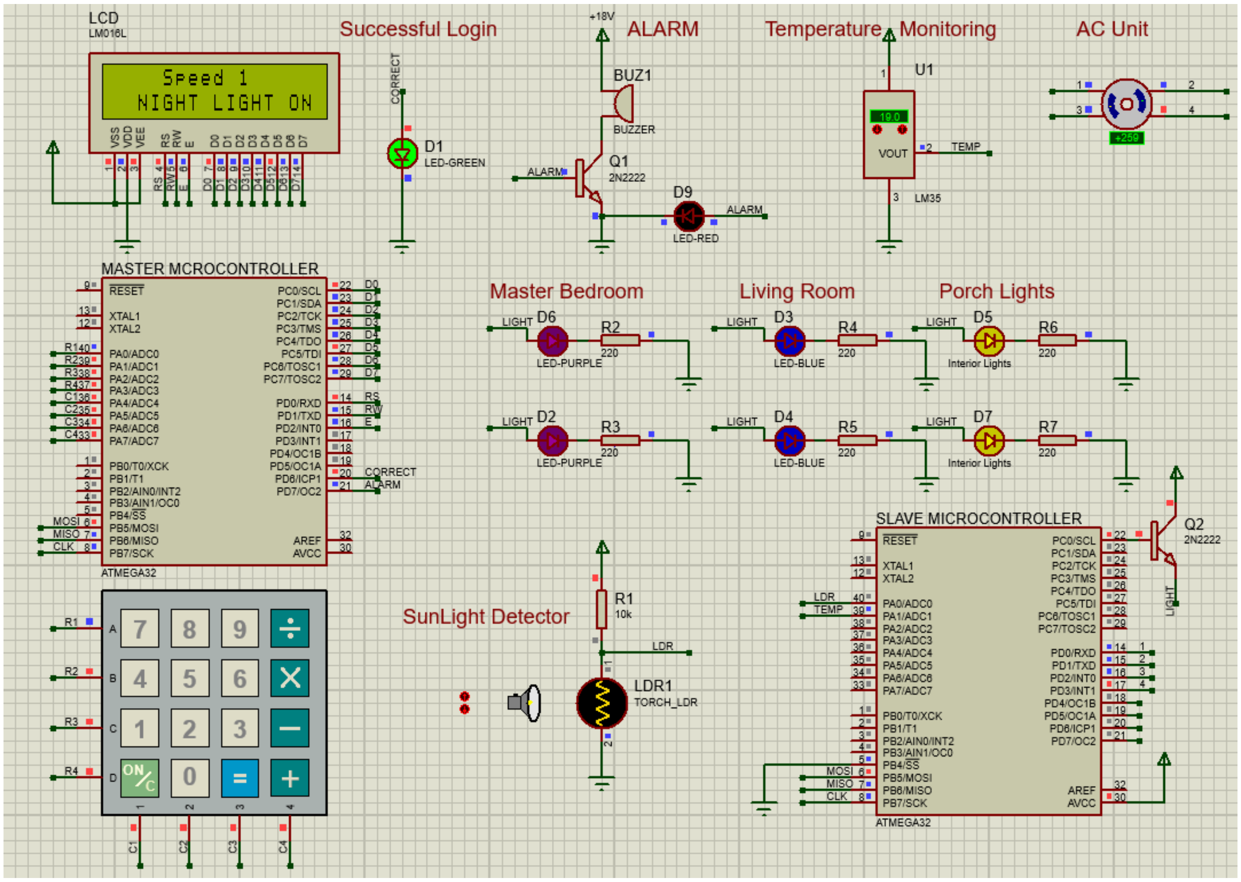


Figure 2-2: System implementation

CHAPTER 3: Applications

There is a multitude of Applications that can be utilized with the concept of integrating electronics and communication with normal households, this can truly transform the way we conduct our day and think about everyday life.

3.1 ROBUST SECURITY SYSTEM

Figure 3-1: Security system

This system provides a security system for the household to keep it safe, secure measures are put in place to allow for margins of error on the user's end, while having a plan of actions in case of security breach attempt, this includes firing and alarm and notifying the police department.

3.1.1 State representation

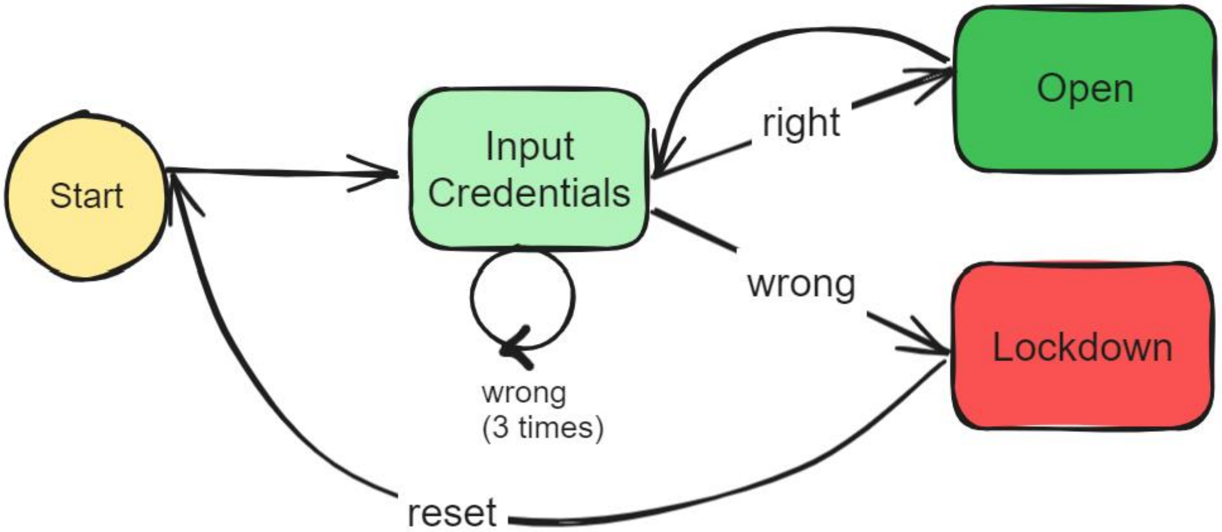
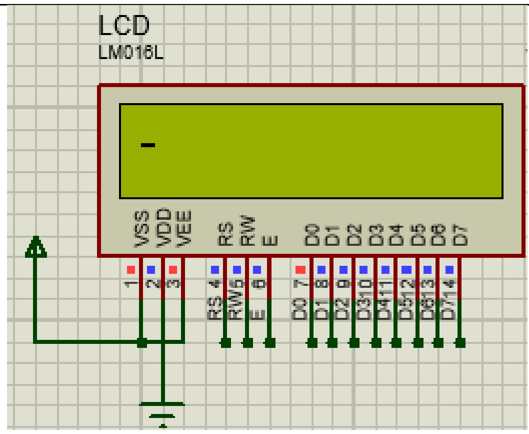


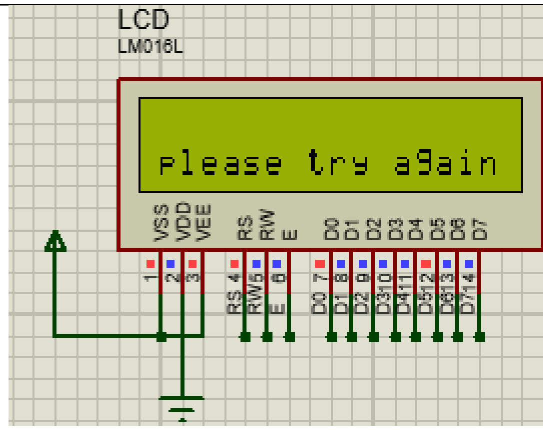
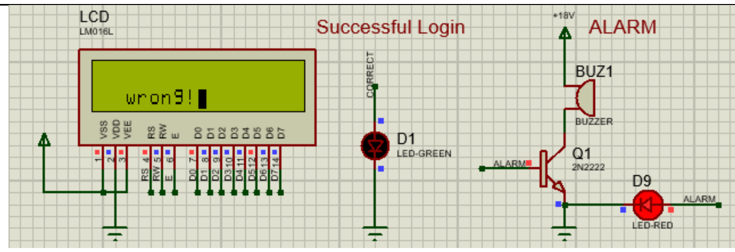
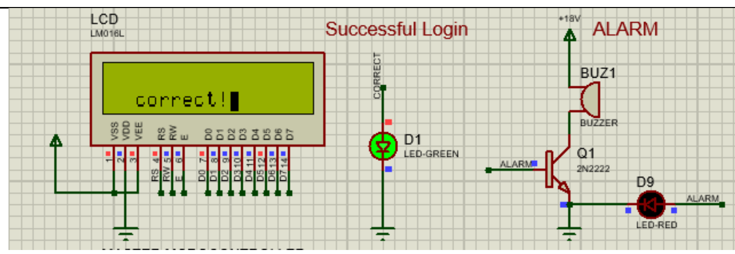
Figure 3-2 Security System Implementation:

3.1.2 Implementation

Table 3-1: Security System Implementation

Start	

Applications

Input wrong credentials	
Exceeding margin of error	
Right Credentials	

3.2 FIRE SAFETY SYSTEM

Figure 3-3: Fire Alarm

The system keeps track of the state of the household at all times and starts deploying safety measures at any sign of the danger of a fire, to ensure the soundness of the household and the wellbeing of its residents, this includes cutting off all power sources and contacting the fire department.

3.2.1 State representation

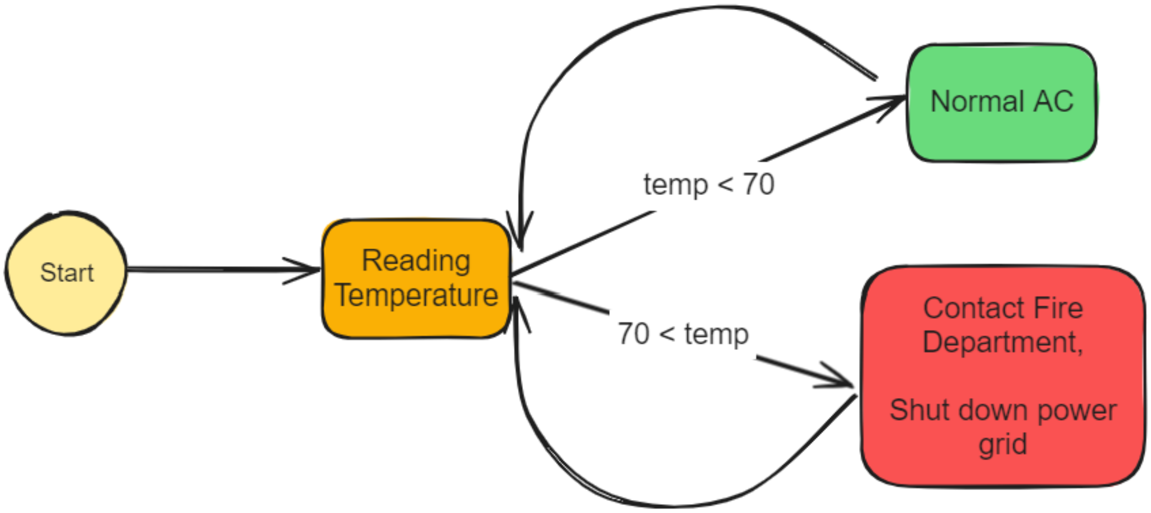
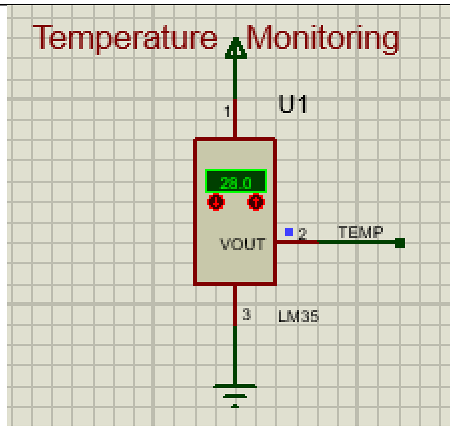
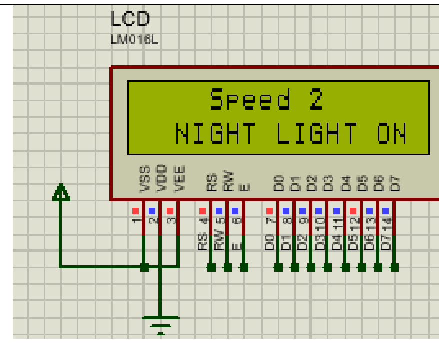


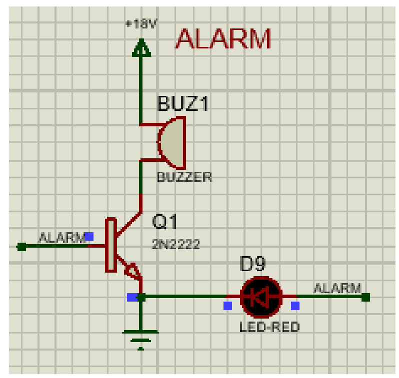
Figure 3-4: Safety system state

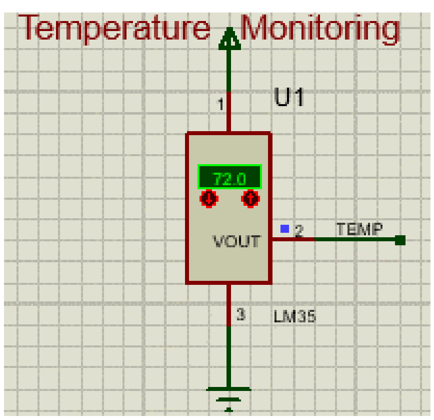
3.2.2 Implementation

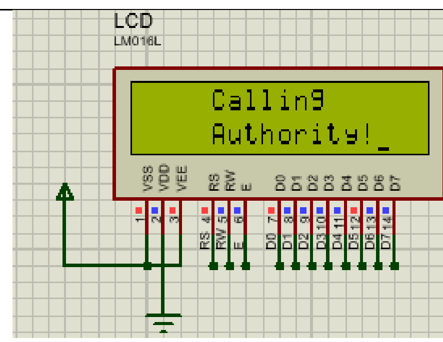
Table 3-2: Safety System implementation

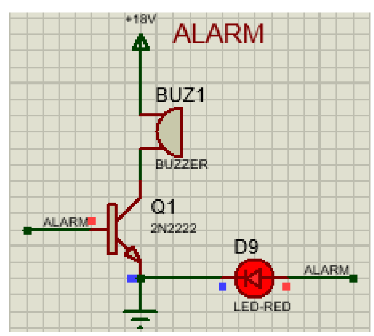
<table>
<tr>
<td>


Temp < 70
</td>
<td>




</td>
</tr>
<tr>
<td>


Temp > 70
</td>
<td>




</td>
</tr>
</table>

3.3 TIME DEPENDENT ILLUMINATION

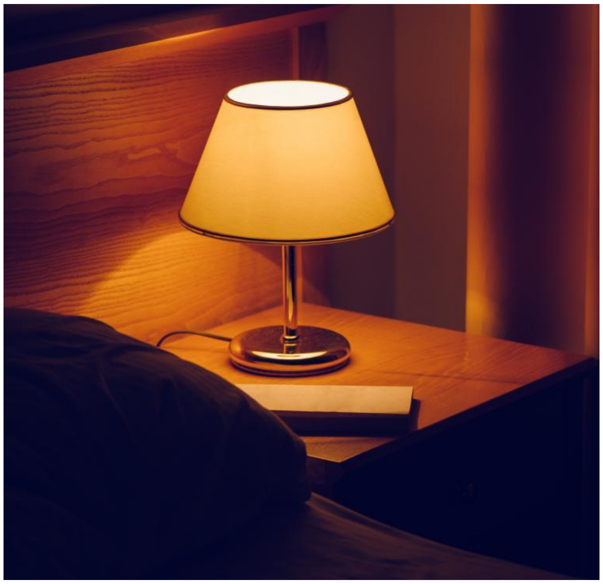


Figure 3-5: Controlled illumination

According to the phase of the day, the system controls the lighting of the whole household, having the illumination levels related to the state of the exterior environment, making them brighter when the sun goes down and turning them off when the sun comes up to save power.

3.3.1 State representation

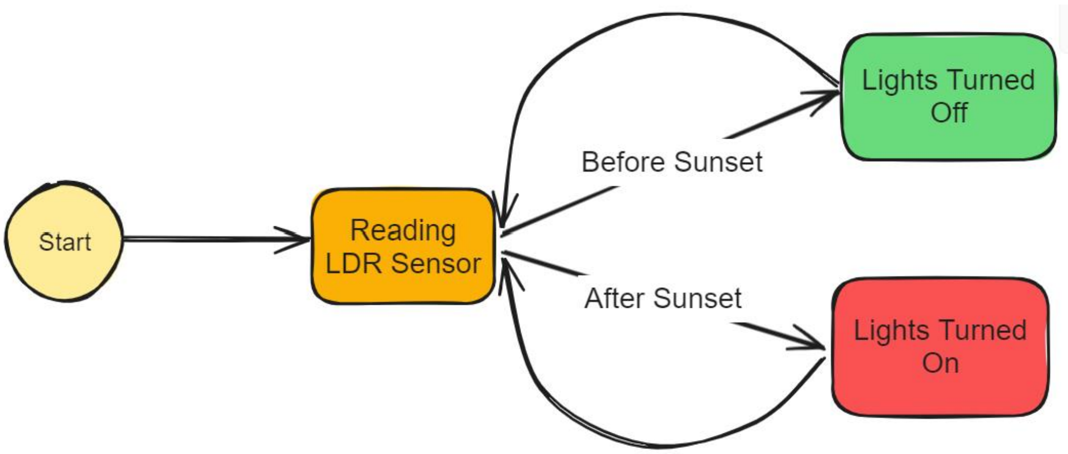
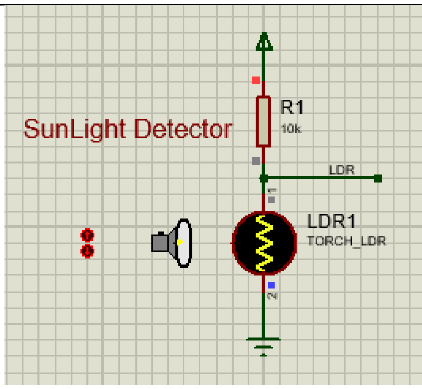
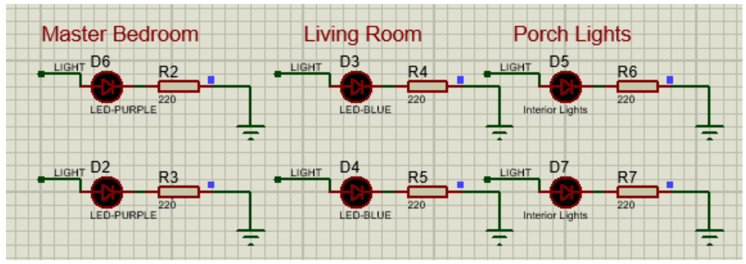
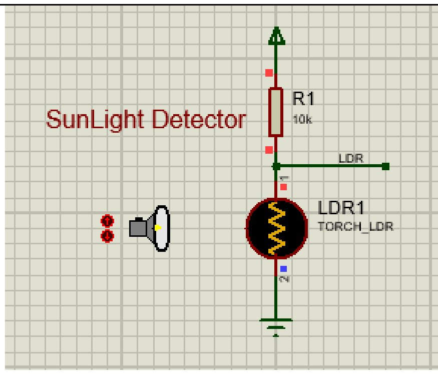


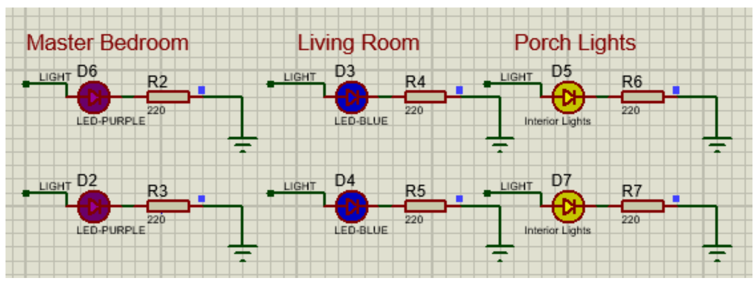
Figure 3-6: Illumination state

3.3.2 Implementation

Table 3-3: Illumination implementation

<table>
<tr>
<td>

Before Sunset</td>
<td>
</td>
</tr>
<tr>
<td>

After Sunset</td>
<td>
</td>
</tr>
</table>

3.4 ADAPTIVE AIR CONDITIONING SYSTEM

Figure 3-7: AC Unit

The system can change the strength and level of the AC Unit in order to make the environment inside the house more compliant to acceptable living conditions, so when the heat shoots up in the summer day time, the AC Unit will go full force on cooling the house, and when the head goes way down in the winter nights, the AC Unit adds some warmth to the air, and it’s all done adaptively according to the measured temperatures.

3.4.1 Implementation

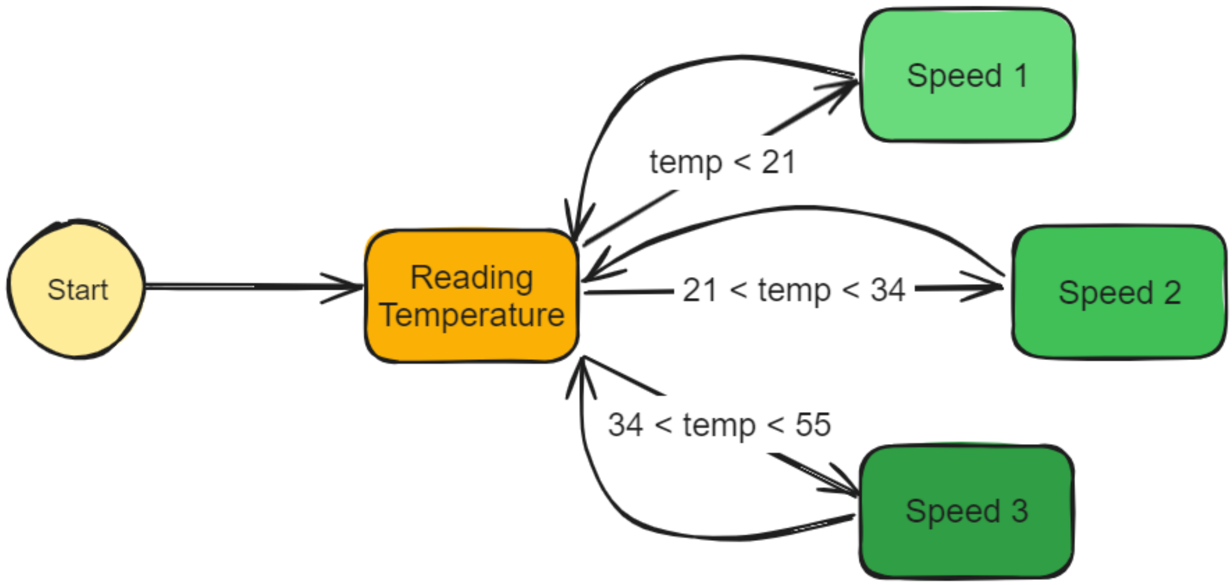
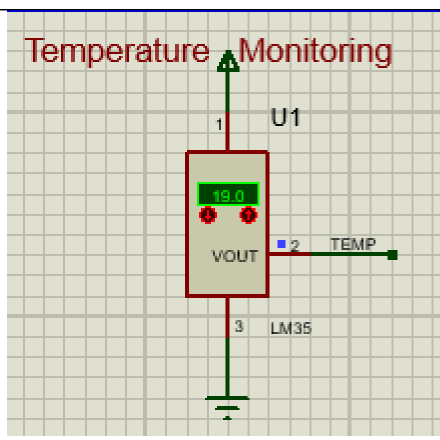
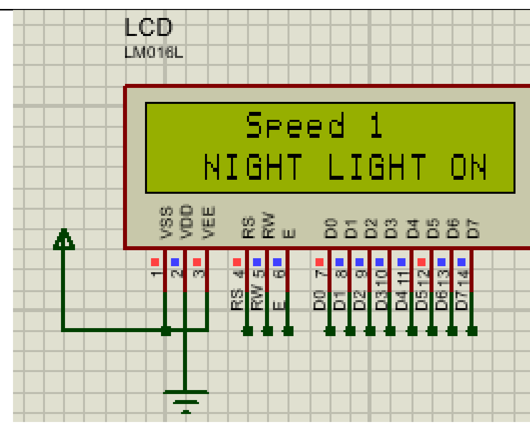


Figure 3-8: Adaptive AC state

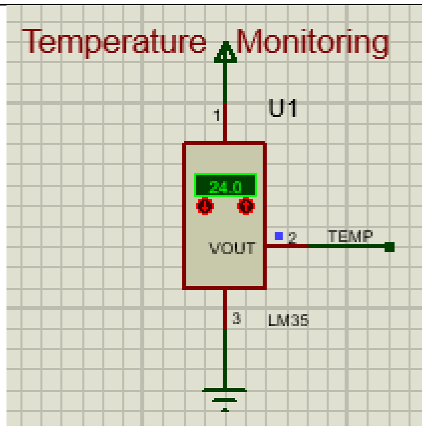
3.4.2 State representation

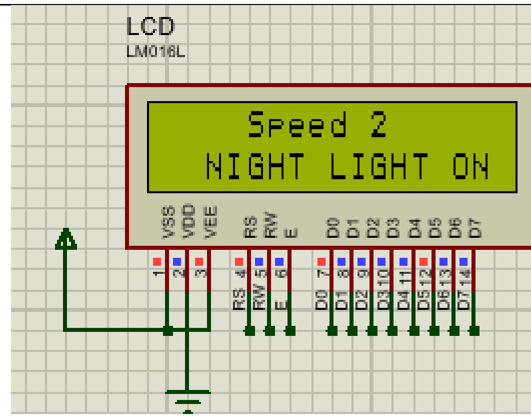
Table 3-4: Adaptive AC implementation

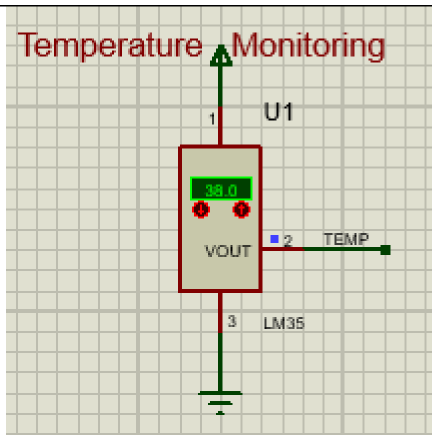
<table>
<tr>
<td>

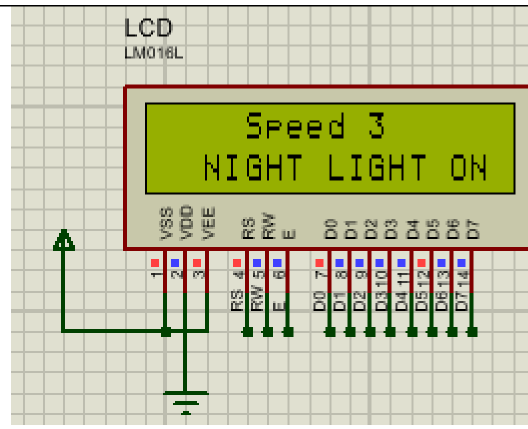
Temp < 21</td>
<td>
</td>
</tr>
</table>

Applications

<table>
<tr>
<td>


21 < Temp < 34
</td>
<td>


</td>
</tr>
<tr>
<td>


34< Temp < 55
</td>
<td>


</td>
</tr>
</table>

3.5 AUTO IRRIGATION

Watering plants in home garden has to happen with care and consistency, the system can schedule the irrigation times, but rain can also be a valuable source for home gardens, and so the system monitors the state of the rain and humidity and shuts off the scheduled watering sessions in case of rainfall.

3.5.1 State representation

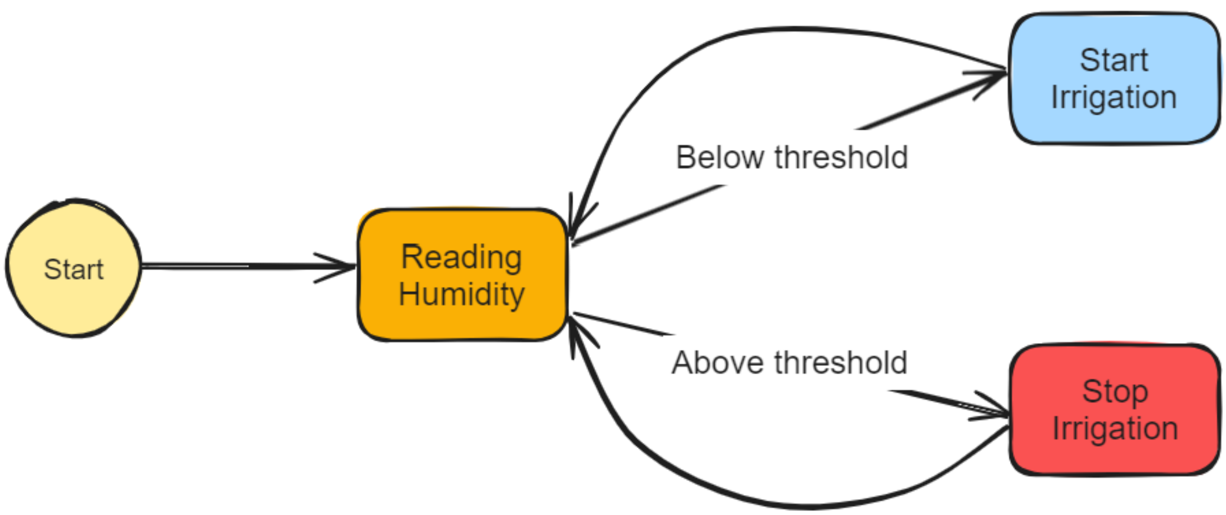


Figure 3-9: Irrigation system states

CHAPTER 4: Implementation Details

4.1 THE LAYER ARCHITECTURE

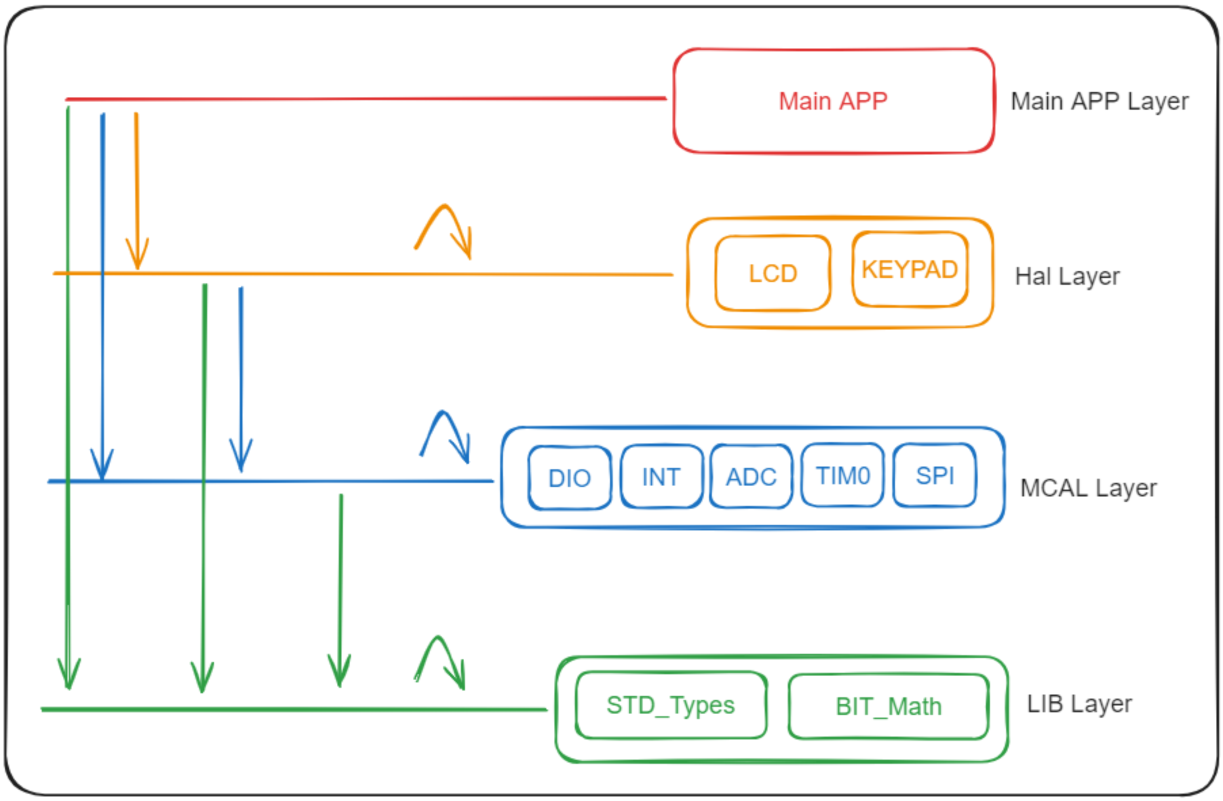


Figure 4-1: Layer Architecture

The layers are:

1- **LIB**: Library Layer.

Here we have the basic fundamental functionalities needed to communicate with the microcontroller.

2- **MCAL**: Microcontroller Abstraction Layer.

here we have the firmware for all microcontroller peripherals like DIO, ADC, Timers, Interrupts etc.

3- **HAL**: Hardware Abstraction Layer.

here we have the device driver for all hardware not inside the microcontroller itself like KEYPAD, LCD etc.

4- **Main APP**: Application Layer.

here we write the code related to our application.

In this Diagram, each layer can call the layer beneath it to functionality and data, but the other way around is prohibited.

Different modules in the <u>same layers</u> can <u>call each other</u>, and modules from the layers beneath them.

This ensures portability of code, as the **LIB** and **MCAL** layers can be used with different **HAL** Layer components and for different **Applications** without depending of any other components

4.2 LIB LAYER

4.2.1 STD_Types.h

Data Type	Size	Range
Char or signed char	1byte	-128 to +128
Unsigned char	1byte	0 to 255
Int or singed int	2byte	-32768 to 32767
Unsigned int	2byte	0 to 65535

Figure 4-2: Standard data types in C

```
/*
 * STD_Types.h
 *
 * Standard library - modified types
*/

 #ifndef   STD_TYPES_H
 #define   STD_TYPES_H
 /*****************************************************/

typedef  unsigned char      u8    ;
typedef  unsigned short int u16   ;
typedef  unsigned long  int u32   ;
typedef  signed char        s8    ;
typedef  signed short  int  s16   ;
typedef  signed long   int  s32   ;
typedef  float              f32   ;
typedef  double             f64   ;
typedef  long double        f128  ;
typedef  unsigned long long  int u64 ;

/*user expected errors*/
typedef enum {
    FUN_OK,
    FUN_NOK,
    PARAM_OUT_OF_RANGE ,
}enuErrorStatus;


#endif
```

Figure 4-3: STD_Types.h

4.2.2 BIT_Math.h

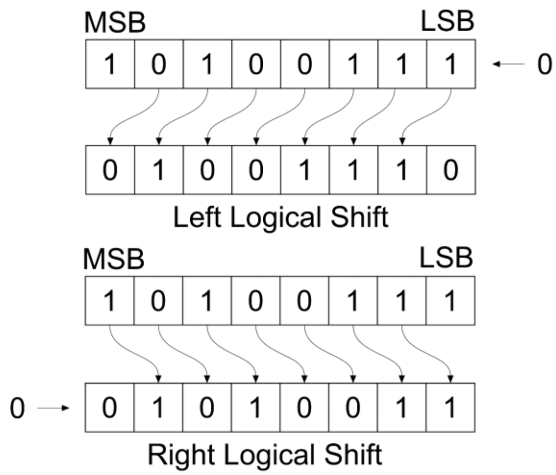


Figure 4-4: Shifting bits

```c
/*
 * BIT_Math.h
 *
 * Bitwise math operations on individual bits
 */

#ifndef BIT_MATH_H_
#define BIT_MATH_H_

#define SET_BIT(REG,PIN) (REG |= (1<<PIN))
#define CLR_BIT(REG,PIN) (REG &= ~(1<<PIN))
#define TOG_BIT(REG,PIN) (REG ^= (1<<PIN))
#define GET_BIT(REG,PIN) ((REG>>PIN)&0x01)
#define ROT_L(REG,PIN) (REG = (REG<< PIN) | (REG >> (8-PIN)))
#define ROT_R(REG,PIN) (REG = (REG >> PIN) | (REG <<> (8-PIN)))


#endif /* BIT_MATH_H_ */
```

Figure 4-5: BIT_Math.h

4.3 MCAL LAYER

4.3.1 DIO

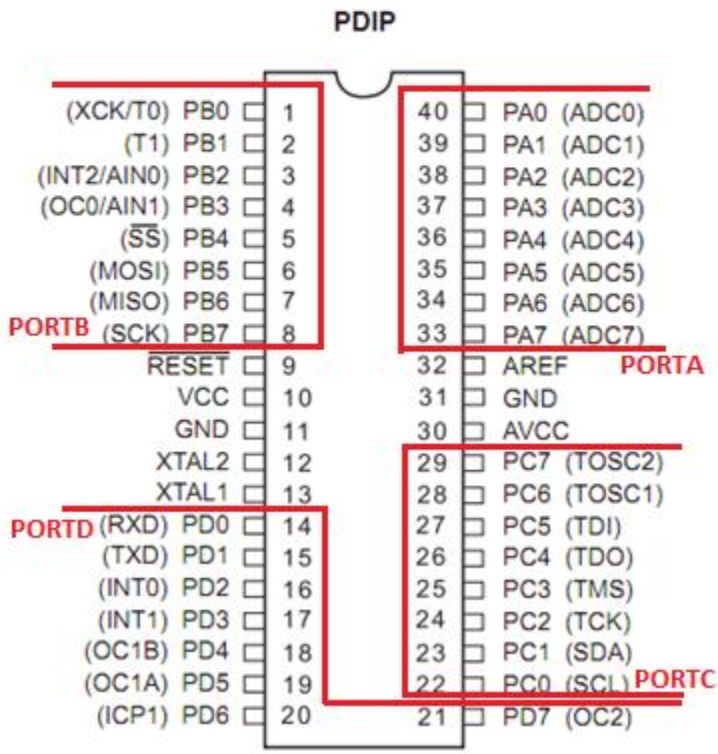


Figure 4-6: PORTS and PINS arrangement in the ATMega32 Microcontroller

DIO.h

```
/*
 * DIO.h
 *
 * Digital Input Output
 */

/*
 **************** The Includes *****************************
 */

#ifndef DIO_H
#define DIO_H
/*
 **************** Defining Ports ****************************
 */

#define DIO_PORTA 0
#define DIO_PORTB 1
#define DIO_PORTC 2
#define DIO_PORTD 3

/*
 **************** Defining Pins *****************************
 */
#define DIO_PIN0 0
#define DIO_PIN1 1
#define DIO_PIN2 2
#define DIO_PIN3 3
#define DIO_PIN4 4
#define DIO_PIN5 5
#define DIO_PIN6 6
#define DIO_PIN7 7

/*
 **************** Defining Pin Input/Output *****************
 */
#define DIO_OUTPUT 1
#define DIO_INPUT 0

/*
 **************** Defining Port Input/Output ***************
 */
#define DIO_OUTPUT_PORT 0xff
#define DIO_INPUT_PORT  0x00

```

Figure 4-7: DIO.h 1

```c
/*
 **************** Defining pin High/Low ********************
 */
#define DIO_HIGH 1
#define DIO_LOW  0

/*
 **************** Defining Port High/Low*********************
 */
#define DIO_HIGH_PORT 0xff
#define DIO_LOW_PORT  0x00

/*
 **************** APIs **********************************************************
 */

/*
 **************** Port APIs*********************************
 */
void DIO_voidSetPortDirection (u8 Copy_u8Port, u8 Copy_u8Direction);
void DIO_voidSetPortValue     (u8 Copy_u8Port, u8 Copy_u8Value);

/*
 **************** Pin APIs**********************************
 */
void DIO_voidSetPinDirection (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Direction);
void DIO_voidSetPinValue     (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Value);
u8   DIO_u8GetPinValue       (u8 Copy_u8Port, u8 Copy_u8Pin);

void DIO_voidTogglePin       (u8 Copy_u8Port, u8 Copy_u8Pin);


#endif
```

Figure 4-8: DIO.h 2

DIO.c

```c
/*
 * Dio.c
 *
 * Digital Input Output
 */

/*
 ***************** THe Includes ************************************
 */
#include <avr/io.h>
#include "../../LIB/STD_types.h"
#include "../../LIB/BIT_Math.h"
#include "DIO.h"

/*
 ***************** Settings Port Direction *************************
 */
void DIO_voidSetPortDirection(u8 Copy_u8Port, u8 Copy_u8Direction){
    switch(Copy_u8Port){
    case DIO_PORTA:     DDRA = Copy_u8Direction;    break;
    case DIO_PORTB:     DDRB = Copy_u8Direction;    break;
    case DIO_PORTC:     DDRC = Copy_u8Direction;    break;
    case DIO_PORTD:     DDRD = Copy_u8Direction;    break;
    }

}

/*
 ***************** Settings Port Value *****************************
 */
void DIO_voidSetPortValue    (u8 Copy_u8Port, u8 Copy_u8Value){
    switch(Copy_u8Port){
    case DIO_PORTA:     PORTA = Copy_u8Value;       break;
    case DIO_PORTB:     PORTB = Copy_u8Value;       break;
    case DIO_PORTC:     PORTC = Copy_u8Value;       break;
    case DIO_PORTD:     PORTD = Copy_u8Value;       break;
    }
}

/*
 ***************** Settings Pin Direction **************************
 */
void DIO_voidSetPinDirection (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Direction){
    if(Copy_u8Direction == DIO_OUTPUT){
        switch(Copy_u8Port){
        case DIO_PORTA:     SET_BIT(DDRA,Copy_u8Pin);       break;
        case DIO_PORTB:     SET_BIT(DDRB,Copy_u8Pin);       break;
        case DIO_PORTC:     SET_BIT(DDRC,Copy_u8Pin);       break;
        case DIO_PORTD:     SET_BIT(DDRD,Copy_u8Pin);       break;
        }
```

Figure 4-9: DIO.c 1

```c
        }
    }
    else{
        switch(Copy_u8Port){
        case DIO_PORTA:     CLR_BIT(DDRA,Copy_u8Pin);       break;
        case DIO_PORTB:     CLR_BIT(DDRB,Copy_u8Pin);       break;
        case DIO_PORTC:     CLR_BIT(DDRC,Copy_u8Pin);       break;
        case DIO_PORTD:     CLR_BIT(DDRD,Copy_u8Pin);       break;
        }
    }
}

/*
 ***************** Settings Pin Value *****************************
 */
void DIO_voidSetPinValue     (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Value){
    if(Copy_u8Value == DIO_HIGH){
        switch(Copy_u8Port){
        case DIO_PORTA:     SET_BIT(PORTA,Copy_u8Pin);      break;
        case DIO_PORTB:     SET_BIT(PORTB,Copy_u8Pin);      break;
        case DIO_PORTC:     SET_BIT(PORTC,Copy_u8Pin);      break;
        case DIO_PORTD:     SET_BIT(PORTD,Copy_u8Pin);      break;
        }
    }
    else{
        switch(Copy_u8Port){
        case DIO_PORTA:     CLR_BIT(PORTA,Copy_u8Pin);      break;
        case DIO_PORTB:     CLR_BIT(PORTB,Copy_u8Pin);      break;
        case DIO_PORTC:     CLR_BIT(PORTC,Copy_u8Pin);      break;
        case DIO_PORTD:     CLR_BIT(PORTD,Copy_u8Pin);      break;
        }
    }
}

/*
 ***************** Reading Pin Value ********************************
 */
u8   DIO_u8GetPinValue       (u8 Copy_u8Port, u8 Copy_u8Pin){
    u8 value;
        switch(Copy_u8Port){
        case DIO_PORTA:     value = GET_BIT(PINA,Copy_u8Pin);       break;
        case DIO_PORTB:     value = GET_BIT(PINB,Copy_u8Pin);       break;
        case DIO_PORTC:     value = GET_BIT(PINC,Copy_u8Pin);       break;
        case DIO_PORTD:     value = GET_BIT(PIND,Copy_u8Pin);       break;
        }
    return value;
}
```

Figure 4-10: DIO.c 2

4.3.2 Interrupt

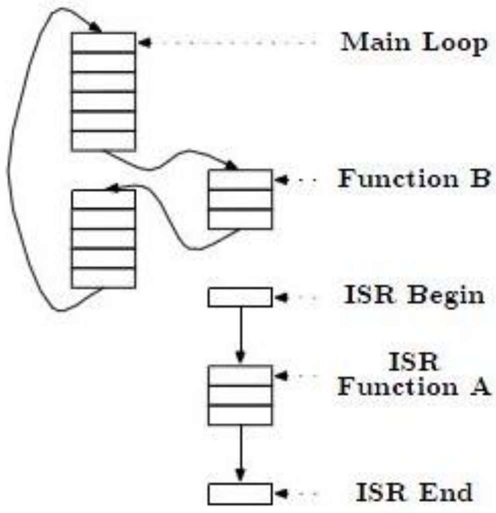


Figure 4-11: Interrupt using ISR

The ATmega32 microcontroller utilizes interrupts to handle unexpected events without halting the main program. When an interrupt, like a button press or timer overflow, occurs, the ATmega32 jumps to a designated function called an Interrupt Service Routine (ISR). The ISR deals with the event quickly, then the microcontroller returns to the main program seamlessly. This allows efficient multitasking and responsiveness to external stimuli.

INT .h

```
/*
 * INT.h
 *
 * Interrupt
 */

#ifndef INT_H_
#define INT_H_

/*
 ***************** Configuration ************
 */

#define INT_INT0            0
#define INT_INT1            1
#define INT_INT2            2

/*
 ***************** Sense *******************
 */
#define INT_FALLING_EDGE        0
#define INT_RISING_EDGE         1
#define INT_ANY_CHANGE          2
#define INT_LOW_LEVEL           3

/*
 ***************** APIs *****************************************
 */
void INT_voidEnable(u8 Copy_u8INTINdex, u8 Copy_u8INTSense);
void INT_voidDisable(u8 Copy_u8INTINdex);


#endif /* MCAL_INT_INT_H_ */
```

Figure 4-12: INT.h

INT.c

```c
/*
 * INT.c
 *
 * Interrupt
 */

/*
 ************* The Includes *********************************
 */
#include "../../LIB/STD_Types.h"
#include "../../LIB/BIT_Math.h"
#include <avr/interrupt.h>
#include "INT.h"

/*
 ************* Enabling Interrupt ***************************
 */
void INT_voidEnable(u8 Copy_u8INTINdex, u8 Copy_u8INTSense){

    switch (Copy_u8INTINdex) {
        /*
         ************* In case of Interrupt 0 ***************
         */
        case INT_INT0:
            /*
             ************* Selecting Sense ******************
             */
            switch (Copy_u8INTSense) {
                case INT_FALLING_EDGE:
                    CLR_BIT(MCUCR, ISC00);
                    SET_BIT(MCUCR, ISC01);
                    break;

                case INT_RISING_EDGE:
                    SET_BIT(MCUCR, ISC00);
                    SET_BIT(MCUCR, ISC01);
                    break;

                case INT_ANY_CHANGE:
                    CLR_BIT(MCUCR, ISC01);
                    SET_BIT(MCUCR, ISC00);
                    break;

                case INT_LOW_LEVEL:
                    CLR_BIT(MCUCR, ISC00);
                    CLR_BIT(MCUCR, ISC01);
                    break;
            }

```

Figure 4-13: INT.c 1

```
            /*
             ************* Enabling Interrupt 0 **************
             */
            SET_BIT(GICR, INT0);
            break;

        /*
         ************* In case of Interrupt 1 ****************
         */
        case INT_INT1:
            /*
             ************* Selecting Sense *******************
             */
            switch (Copy_u8INTSense) {
                case INT_FALLING_EDGE:
                    CLR_BIT(MCUCR, ISC10);
                    SET_BIT(MCUCR, ISC11);
                    break;

                case INT_RISING_EDGE:
                    SET_BIT(MCUCR, ISC10);
                    SET_BIT(MCUCR, ISC11);
                    break;

                case INT_ANY_CHANGE:
                    CLR_BIT(MCUCR, ISC11);
                    SET_BIT(MCUCR, ISC10);
                    break;

                case INT_LOW_LEVEL:
                    CLR_BIT(MCUCR, ISC10);
                    CLR_BIT(MCUCR, ISC11);
                    break;
            }
            /*
             ************* Enabling Interrupt 1 ***************
             */
            SET_BIT(GICR, INT1);
            break;

        /*
         ************* In case of Interrupt 2 *****************
         */
```

Figure 4-14: INT.c 2

```c
        case INT_INT2:
            /*
             ************* Selecting Sense ********************
             */
            switch (Copy_u8INTSense) {
                case INT_FALLING_EDGE:
                    CLR_BIT(MCUCSR, ISC2);
                    break;

                case INT_RISING_EDGE:
                    SET_BIT(MCUCSR, ISC2);
                    break;
            }
            /*
             ************* Enabling Interrupt 2 ***************
             */
            SET_BIT(GICR, INT2);
            break;
    }
}


/*
 ************* Disabling Interrupt ****************************
 */
void INT_voidDisable(u8 Copy_u8INTINdex){

    switch (Copy_u8INTINdex) {
        /*
         ************* Interrupt 0 ****************************
         */
        case INT_INT0:
            CLR_BIT(GICR, INT0);
            break;
            /*
             ************* Interrupt 1 ************************
             */
        case INT_INT1:
            CLR_BIT(GICR, INT1);
            break;
            /*
             ************* Interrupt 2 ************************
             */
        case INT_INT2:
            CLR_BIT(GICR, INT2);
            break;
    }
}
```

Figure 4-15: INT.c 3

4.3.3 TIMER 0

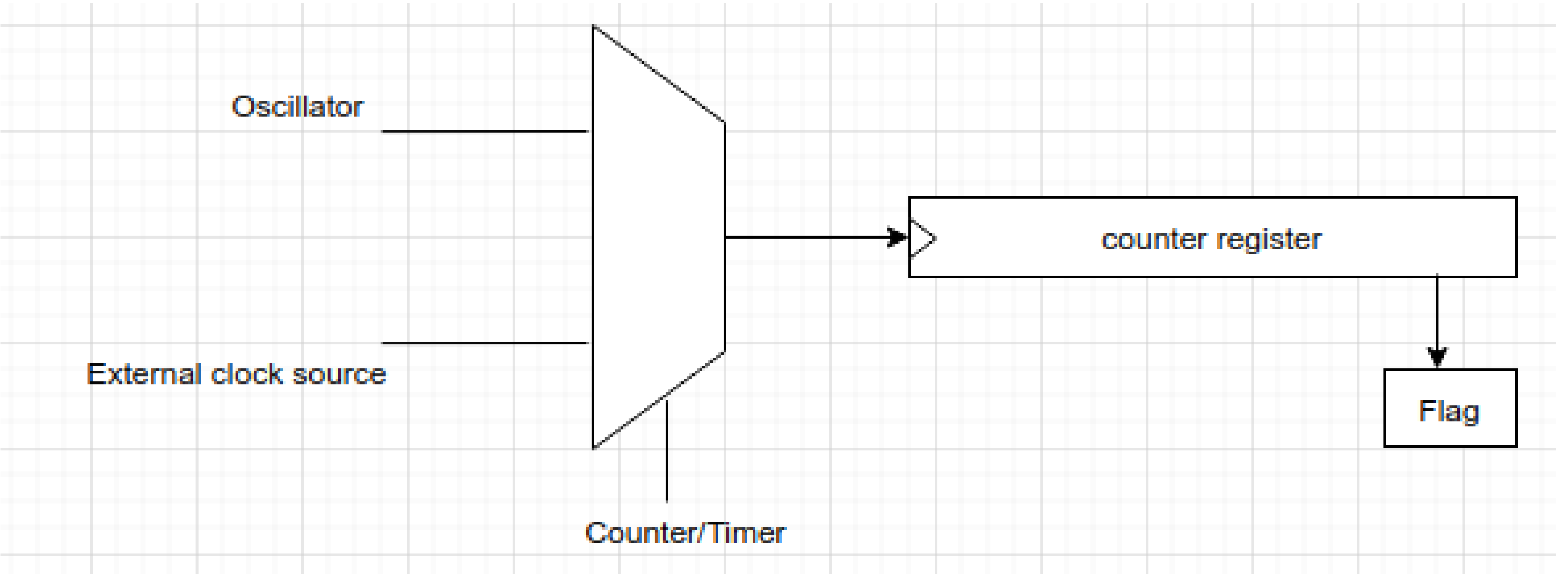


Figure 4-16: Timer

The ATmega32 packs three timers: two 8-bit (Timer 0 and Timer 2) and one 16-bit (Timer 1). These timers aren't just for counting time. They can generate precise delays, create Pulse Width Modulation (PWM) signals for dimming LEDs or controlling motors, and even keep track of external events. Each timer has dedicated registers for configuration and control, allowing for flexible control over your project's timing needs.

TIMER0.h

```c
/*
 * TIM0.h
 *
 * Timer 0
 */

/*
 *************** Head Guard*************************
 */
#ifndef TIM0_H
#define TIM0_H

/*
 *************** Configuration (CFG) ***************************
 */


/*
 *************** Select Mode **********************
 */
#define TIM0_MODE                       FAST_PWM
/* NORMAL       CTC      PWM_PHASE        FAST_PWM */


/*
 *************** Prescaler ************************
 */
#define TIM0_STOPPED                    0
#define TIM0_NO_PRESCALER               1
#define TIM0_PRESCALER_8                2
#define TIM0_PRESCALER_64               3
#define TIM0_PRESCALER_256              4
#define TIM0_PRESCALER_1024             5
#define TIM0_EXT_FALLING                6
#define TIM0_EXT_RISINGING              7

/*
 *************** Interrupt control*****************
 */
#define TIM0_TIN_EABLE                  1
#define TIM0_TIN_DISABLE                0


/*
 *************** CO0 Pin Action (NORMAL, CTC)*******
 */
#define TIM0_OC0_IN_OUT_PIN             0
#define TIM0_OC0_TOGGLE_PIN             1
#define TIM0_OC0_CLEAR_PIN              2
#define TIM0_OC0_SET_PIN                3
```

Figure 4-17: TIMER0.h 1

```c

/*
 *************** CO0 Pin Action (FAST PWM) *********
 */
#define TIM0_OC0_SET_CMP_CLR_OVF            1
#define TIM0_OC0_CLR_CMP_SET_OVF            2


/*
 *************** CO0 Pin Action (PHASE PWM) ********
 */
#define TIM0_OC0_SET_UPC_CLR_DNC            1
#define TIM0_OC0_CLR_UPC_SET_DNC            2


/*
 *************** APIs ****************************************************
 */

void TIM0_voidInit           (void);
void TIM0_voidSetPreValue    (u8 Copy_u8CounterStartVal);
void TIM0_voidTimerStart     (u8 Copy_u8Prescaler      );
void TIM0_voidOvrINTControl  (u8 Copy_u8InterruptConl  );
void TIM0_voidSetCompareMat  (u8 Copy_u8CmpMatVal, u8 Copy_u8OC0PinAction);
void TIM0_voidPWMGenerator   (u8 Copy_u8DutyCycle, u8 Copy_u8OC0PinAction);
void TIM0_voidSetCmpValue    (u8 Copy_u8CmpMatVal      );
void TIM0_voidCmpINTControl  (u8 Copy_u8InterruptConl  );
void TIM0_voidTimerStop      ( void);
void TIM0_voidOvfCallback    ( void (*Copy_pfTIM0Ovf)(void) );
void TIM0_voidCmpCallback    ( void (*Copy_pfTIM0Cmp)(void) );

// returned type (* pf) (input parameters)


#endif /* MCAL_TIMER0_TIM0_H_ */

```

Figure 4-18: TIMER0.h 2

TIMER0.c

```c
/*
 * TIM0.c
 *
 * TIMER 0
 *
 */

/*
 **************** The Includes ***************************
 */
#include "../../LIB/BIT_Math.h"
#include "../../LIB/STD_Types.h"
#include <avr/io.h>
#include <avr/interrupt.h>

/*
 **************** Defining Timer0 Parameters ***********
 */
#define NORMAL          0
#define CTC             1
#define PWM_PHASE       2
#define FAST_PWM        3


#include "TIM0.h"


/*
 **************** Global pointer to function ************
 */
void (*TIM0_pfTIM0Ovf)(void);
void (*TIM0_pfTIM0Cmp)(void);


/*
 **************** Initializing Timer0 *******************
 */
void TIM0_voidInit  (void){
    /*
     **************** Selecting Timer0 mode *************
     */
#if TIM0_MODE == NORMAL
    CLR_BIT(TCCR0, WGM00);
    CLR_BIT(TCCR0, WGM01);
#elif TIM0_MODE == CTC
    CLR_BIT(TCCR0, WGM00);
    SET_BIT(TCCR0, WGM01);
#elif TIM0_MODE == PWM_PHASE
    CLR_BIT(TCCR0, WGM01);
    SET_BIT(TCCR0, WGM00);
```

Figure 4-19: TIMER0.c 1

```
#else
    SET_BIT(TCCR0, WGM00);
    SET_BIT(TCCR0, WGM01);
#endif

}

/*
 **************** Getting PreValue for the timer0
 **************** to start counting from ***************
 */
void TIM0_voidSetPreValue    (u8 Copy_u8CounterStartVal){
    /* set timer start value */
    TCNT0 = Copy_u8CounterStartVal;

}

/*
 **************** Starting the Timer0 ******************
 */
void TIM0_voidTimerStart     (u8 Copy_u8Prescaler){
    /*
     **************** Setting Prescaler ****************
     */
    TCCR0 &= 0b11111000;
    TCCR0 |= Copy_u8Prescaler;

}

/*
 ****************  *****************
 */
void TIM0_voidOvrINTControl (u8 Copy_u8InterruptConl){

    if(Copy_u8InterruptConl == TIM0_TIN_EABLE)
        {SET_BIT(TIMSK, TOIE0);}
    else
        {CLR_BIT(TIMSK, TOIE0);}

}
```

Figure 4-20: TIMER0.c

```c
/*
 ***************** Sets the value of the
 ***************** Output Compare Register(OCR0) for Timer0
 ***************** and controlling OC0 **********************
 */
void TIM0_voidSetCompareMat (u8 Copy_u8CmpMatVal, u8 Copy_u8OC0PinAction){

    switch(Copy_u8OC0PinAction)
    {
        case TIM0_OC0_TOGGLE_PIN :
            SET_BIT(TCCR0, COM00);
            CLR_BIT(TCCR0, COM01);
            break;

        case TIM0_OC0_CLEAR_PIN  :
            SET_BIT(TCCR0, COM01);
            CLR_BIT(TCCR0, COM00);
            break;

        case TIM0_OC0_SET_PIN    :
            SET_BIT(TCCR0, COM00);
            SET_BIT(TCCR0, COM01);
            break;

        default                  :
            CLR_BIT(TCCR0, COM00);
            CLR_BIT(TCCR0, COM01);
            break;
    }

    /*
     *************** Setting compare value ***************************
     */
    OCR0 = Copy_u8CmpMatVal;

}

/*
 ***************** Generate Pulse Width Modulation
 ***************** (PWM) signal **************************
 */
void TIM0_voidPWMGenerator  (u8 Copy_u8DutyCycle, u8 Copy_u8OC0PinAction){

```

Figure 4-21: TIMER0.c 3

```c
/*
 *************** Fast PWM Mode ***************************
 */
#if TIM0_MODE == FAST_PWM
    if(Copy_u8OC0PinAction == TIM0_OC0_SET_CMP_CLR_OVF){
        SET_BIT(TCCR0, COM00);
        SET_BIT(TCCR0, COM01);
        OCR0 = (256 - ((Copy_u8DutyCycle/100.0)*256));
    }
    else{
        SET_BIT(TCCR0, COM01);
        CLR_BIT(TCCR0, COM00);
        OCR0 = ((Copy_u8DutyCycle/100.0)*256);
    }
/*
 *************** Phase Correct PWM Mode ********************
 */
#elif TIM0_MODE == PWM_PHASE
    if(Copy_u8OC0PinAction == TIM0_OC0_SET_UPC_CLR_DNC){
        SET_BIT(TCCR0, COM00);
        SET_BIT(TCCR0, COM01);
        OCR0 = (255 -(((Copy_u8DutyCycle/100.0)*510)/2));
    }
    else{
        SET_BIT(TCCR0, COM01);
        CLR_BIT(TCCR0, COM00);
        OCR0 = (((Copy_u8DutyCycle/100.0)*510)/2);
    }

#endif


}



/*
 **************** Setting value for
 **************** Output Compare Register (OCR0)***********
 */
void TIM0_voidSetCmpValue   (u8 Copy_u8CmpMatVal){

    /* Set Compare value */
    OCR0 = Copy_u8CmpMatVal;
}

```

Figure 4-22: TIMER0.c 4

```c
/*
 **************** Enabling/Disabling compare match interrupt **
 */
void TIM0_voidCmpINTControl (u8 Copy_u8InterruptConl){

	if(Copy_u8InterruptConl == TIM0_TIN_EABLE)
		{SET_BIT(TIMSK, OCIE0);}
	else
		{CLR_BIT(TIMSK, OCIE0);}
}

/*
 **************** Stopping Timer0 ***************************
 */
void TIM0_voidTimerStop 	(void){

	TCCR0 &= 0b11111000;
}
/*
 **************** Setting a callback function for
 **************** when OverFlow Interrupt occurs*************
 */
void TIM0_voidOvfCallback 	( void (*Copy_pfTIM0Ovf)(void) ){

	TIM0_pfTIM0Ovf = Copy_pfTIM0Ovf;
}
/*
 **************** Setting a callback function for
 ****************  when Compare Match Interrupt occurs********
 */
void TIM0_voidCmpCallback 	( void (*Copy_pfTIM0Cmp)(void) ){

	TIM0_pfTIM0Cmp = Copy_pfTIM0Cmp;
}
/*
 **************** Interrupt Service Routine (ISR)
 **************** for the Timer0 Overflow Interrupt **********
 */
ISR(TIMER0_OVF_vect){

	TIM0_pfTIM0Ovf();
}
/*
 **************** Interrupt Service Routine (ISR)
 **************** for the Timer0 Compare Match Interrupt ******
 */
ISR(TIMER0_COMP_vect){

	TIM0_pfTIM0Cmp();
}
```

Figure 4-23: TIMER0.c 5

4.3.4 ADC

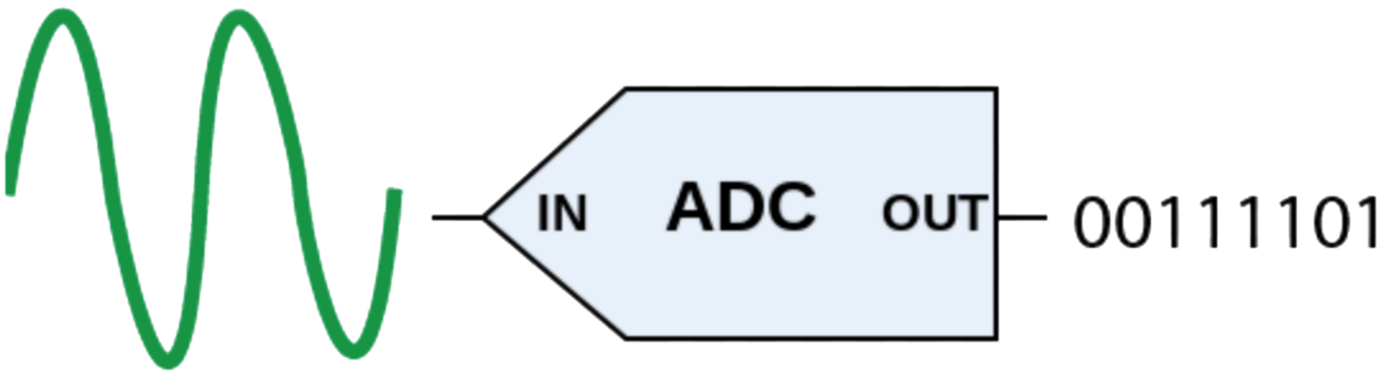


Figure 4-24: ADC

The ATmega32 boasts a 10-bit Analog-to-Digital Converter (ADC), allowing it to convert analog voltages from the real world into digital values the microcontroller can understand. With eight selectable channels, you can connect various sensors like temperature sensors or light sensors. The ADC converts the analog input voltage to a digital value between 0 and 1023, providing a precise representation of the analog signal for further processing within your program.

ADC.h

```c
/*
 * ADC.h
 *
 * Analog Digital Converter
 */
/*
 *************** Head Guard*************************
 */
#ifndef ADC_H
#define ADC_H

/*
 *************** Configuration (CFG) ****************
 */
/*
 *************** Choosing Reference Voltage *********
 */
#define ADC_REF_VOLT        ADC_AVCC
/* ADC_AVCC     ADC_AREF     ADC_2_56_V */

/*
 *************** Prescaler **************************
 */
#define ADC_DIV_2           1
#define ADC_DIV_4           2
#define ADC_DIV_8           3
#define ADC_DIV_16          4
#define ADC_DIV_32          5
#define ADC_DIV_64          6
#define ADC_DIV_128         7
/*
 *************** Defining Channels *******************
 */
#define ADC_Channel_0       0
#define ADC_Channel_1       1
#define ADC_Channel_2       2
#define ADC_Channel_3       3
#define ADC_Channel_4       4
#define ADC_Channel_5       5
#define ADC_Channel_6       6
#define ADC_Channel_7       7
/*
 *************** APIs ******************************************************
 */
void ADC_voidInit(u8 Copy_u8Prescaler);
u16  ADC_u16GetDigitalValue(u8 Copy_u8Channel);
void ADC_voidDisable(void);

#endif /* MCAL ADC ADC H  */
```

Figure 4-25: ADC.h

ADC.c

```c
/*
 * ADC.c
 *
 * Analog Digital Converter
 */

/*
 **************** The Includes *****************
 */
#include "../../LIB/BIT_Math.h"
#include "../../LIB/STD_Types.h"
#include <avr/io.h>

/*
 **************** Defining ADC parameters *****************
 */
#define ADC_AVCC        0
#define ADC_AREF        1
#define ADC_2_56_V      2

#include "ADC.h"

/*
 **************** Initializing ADC *****************
 */
void ADC_voidInit(u8 Copy_u8Prescaler){

    /* Enable ADC */
    SET_BIT(ADCSRA, ADEN);

    /*
     **************** Select Vref *****************
     */
#if ADC_REF_VOLT == ADC_AVCC
    SET_BIT(ADMUX, REFS0);
    CLR_BIT(ADMUX, REFS1);
#elif ADC_REF_VOLT == ADC_AREF
    CLR_BIT(ADMUX, REFS0);
    CLR_BIT(ADMUX, REFS1);
#else
    SET_BIT(ADMUX, REFS0);
    SET_BIT(ADMUX, REFS1);
#endif

    /*
     **************** Select right adjust *****************
     */
    CLR_BIT(ADMUX, ADLAR);

    /*
```

Figure 4-26: ADC.c 1

```c
     ****************** Prescaler ******************
     */
    ADCSRA &= 0b11111000;
    ADCSRA |= Copy_u8Prescaler; // 0b00000110

}

/*
 ****************** Reading the sensor data ******************
 */
u16  ADC_u16GetDigitalValue(u8 Copy_u8Channel){

    /*
     ****************** select Channel ******************
     */
    ADMUX &= 0b11100000;
    ADMUX |= Copy_u8Channel;

    /*
     ****************** Start conversion ******************
     */
    SET_BIT(ADCSRA, ADSC);

    /*
     ****************** Wait flag = 1 ******************
     */
    while(GET_BIT(ADCSRA, ADIF) == 0);

    /*
     ****************** clear flag ******************
     */
    CLR_BIT(ADCSRA, ADIF);

    /*
     ****************** Read ADC value ******************
     */
    return ADC;

}


/*
 ****************** Disabling ADC ******************
 */
void ADC_voidDisable(void){

    /* Disable ADC */
    CLR_BIT(ADCSRA, ADEN);
}

```

Figure 4-27: ADC.c 2

4.3.5 SPI

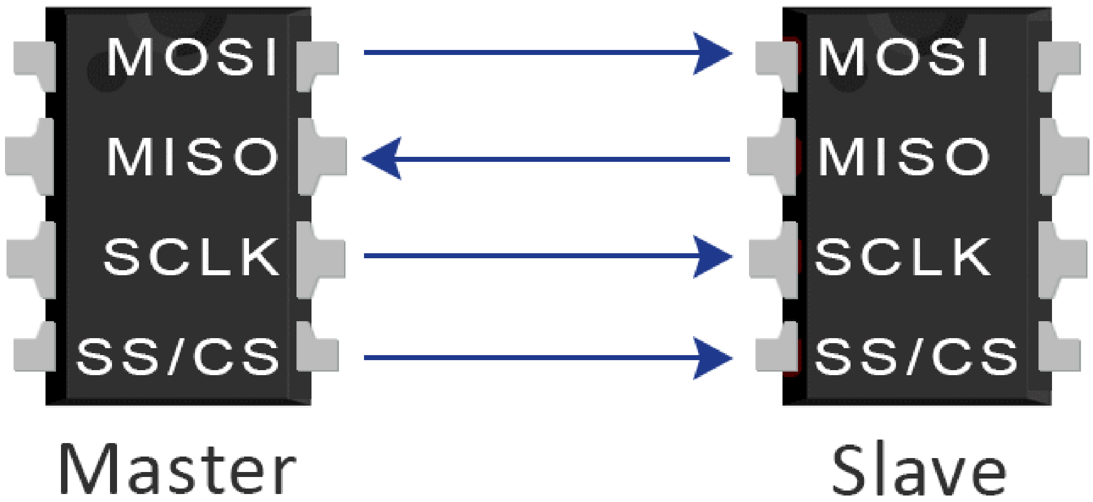


Figure 4-28: SPI

SPI (Serial Peripheral Interface) is a synchronous communication protocol ideal for high-speed data exchange between microcontrollers and peripherals like LCD screens, memory cards, or high-performance sensors. Unlike UART (Universal Asynchronous Receiver/Transmitter), SPI utilizes a dedicated clock signal to synchronize data transfer, making it significantly faster. Compared to I2C (Inter-Integrated Circuit), SPI boasts a simpler design with fewer wires (four versus two) and avoids the overhead of device addressing present in I2C. This translates to faster communication speeds, especially for short bursts of data.

SPI.h

```
/*
 * SPI.h
 *
 * Serial Peripheral Interface
 */

/*
 *************** Head Guard ********************
 */
#ifndef SPI_H
#define SPI_H

/*
 *************** Selecting SPI Mode ************
 */
#define SPI_MODE                SPI_SLAVE_MODE
/* select one option:           SPI_MASTER_MODE         SPI_SLAVE_MODE  */


/*
 *************** APIs ********************************************
 */
void SPI_voidInit           (void);
u8   SPI_u8ReceiveData      (void);
void SPI_voidMasterSendData (u8 Copy_u8Data);
void SPI_voidSalveSendData  (u8 Copy_u8Data);
u8   SPI_u8ReceiveTrans     (u8 Copy_u8Data);


#endif /* MCAL_SPI_SPI_H_ */
```

Figure 4-29: SPI.h

SPI.c

```c
/*
 * SPI.c
 *
 * Serial Peripheral Interface
 */

/*
 **************** The Includes *************************
 */
#include "../../LIB/BIT_Math.h"
#include "../../LIB/STD_Types.h"
#include <avr/io.h>
/*
 **************** Defining Communication Modes *********
 */
#define SPI_MASTER_MODE             1
#define SPI_SLAVE_MODE              2


#include "SPI.h"


/*
 **************** Initializing SPI **********************
 */
void SPI_voidInit   (void){

    /*
     **************** Setting SPI Master *****************
     */
#if SPI_MODE == SPI_MASTER_MODE
    /*
     **************** 1- Select order ==> MSB ************
     */
    CLR_BIT(SPCR , DORD);
    /*
     **************** 2- select Master mode **************
     */
    SET_BIT(SPCR, MSTR);
    /*3- Clock Polarity & Phase ==> Rising_Falling ... Setup_Sample */
    CLR_BIT(SPCR , CPOL);
    SET_BIT(SPCR, CPHA);
    /*
     **************** 4 -Set clock rate fck/16 ***********
     */
    SET_BIT(SPCR, SPR0);
    CLR_BIT(SPCR, SPR1);
    CLR_BIT(SPSR, SPI2X);
```

Figure 4-30: SPI.c 1

```c
    /*
     **************** Setting SPI Slave *******************
     */
#elif SPI_MODE == SPI_SLAVE_MODE

    /*
     **************** 1- Select order ==> MSB *************
     */
    CLR_BIT(SPCR , DORD);
    /*
     **************** 2- select Slave mode ****************
     */
    CLR_BIT(SPCR, MSTR);
    /*3- Clock Polarity & Phase ==> Rising_Falling ... Setup_Sample */
    CLR_BIT(SPCR , CPOL);
    SET_BIT(SPCR, CPHA);

#endif

    /*
     **************** Enabling SPI **********************
     */
    SET_BIT(SPCR, SPE);

}

/*
 **************** Receiving data *************************
 */
u8   SPI_u8ReceiveData      (void){

    /*
     **************** Wait for reception complete ********
     */
    while(GET_BIT(SPSR, SPIF) == 0);
    /*
     **************** Return data register ***************
     */
    return SPDR;
}
```

Figure 4-31: SPI.c 2

```c
/*
 **************** Sending data as Master******************
 */
void SPI_voidMasterSendData (u8 Copy_u8Data){

	/*
	 **************** Start transmission *****************
	 */
	SPDR = Copy_u8Data;
	/*
	 **************** Wait for transmission complete *****
	 */
	while(GET_BIT(SPSR, SPIF) == 0);
}

/*
 **************** Sending data as Slave ******************
 */
void SPI_voidSalveSendData  (u8 Copy_u8Data){

	SPDR = Copy_u8Data;
}

/*
 **************** Receiving data  ************************
 */
u8   SPI_u8ReceiveTrans     (u8 Copy_u8Data){

	/*
	 **************** Send data **************************
	 */
	SPDR = Copy_u8Data;
	/*
	 ****************  Wait for transmission complete ****
	 */
	while(GET_BIT(SPSR, SPIF) == 0);
	/*
	 **************** Return data register ****************
	 */
	return SPDR;
}
```

Figure 4-32: SPI.c 3

4.4 HAL LAYER

4.4.1 LCD

Figure 4-33: LCD

The ubiquitous 2x16 LCD, or Liquid Crystal Display, is a workhorse in the world of electronics. This compact display boasts two rows, each capable of showing 16 characters. That's a total of 32 alphanumeric characters for you to display messages, sensor readings, or control prompts. It's relatively simple to control using microcontrollers and comes in various configurations with features like backlights and custom character sets. This makes the 2x16 LCD a perfect choice for providing user interaction and informative displays in countless hobbyist and professional projects.

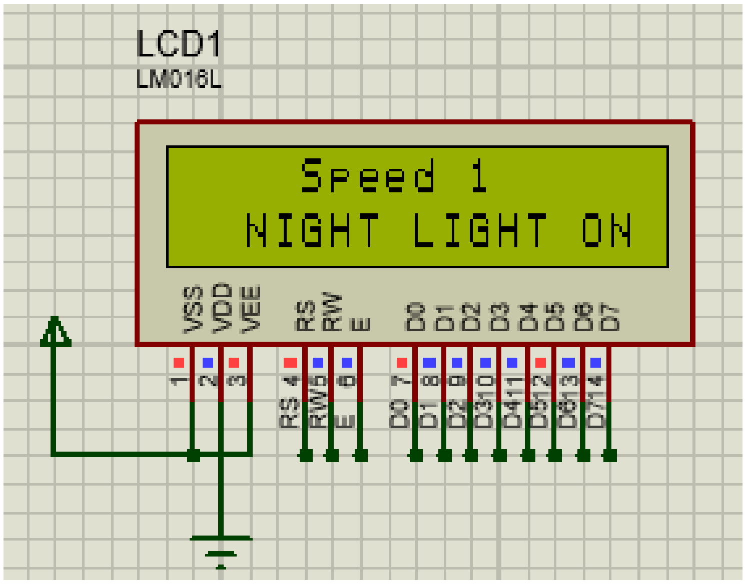


Figure 4-34: LCD running in simulation

LCD.h

```c
#ifndef LCD_H
#define LCD_H

/********************************** Cfg ***************************/
#define LCD_DATA_PORT       DIO_PORTC
#define LCD_CONTROL_PORT    DIO_PORTD
#define RS                  DIO_PIN0
#define RW                  DIO_PIN1
#define EN                  DIO_PIN2
/*****************************************************************/
#define FUNCTION_SET        0b00111000
#define DISPLAY_ON_OFF      0b00001111
#define CLEAR               0b00000001
/*****************************************************************/

#define LCD_LINE1           1
#define LCD_LINE2           2

/********************************** APIs *************************/

void LCD_voidInit           (void);
void LCD_voidSendCommand    (u8 Copy_u8Command);
void LCD_voidSendChar       (u8 Copy_u8Char);
void LCD_voidSendString     (u8 *Copy_u8String);
void LCD_voidSetLocation    (u8 Copy_u8LineNum, u8 Copy_u8CharNum);
void LCD_voidSendNumber     (u32 Copy_u32Number);
void LCD_voidDrawSpecialChar(u8 Copy_u8CharIndex, u8 *Copy_u8SpecialChar);
void LCD_voidSendSpecialChar(u8 Copy_u8CharIndex);

#endif
```

Figure 4-35: LCD.h

LCD.c

```
/*
* ****************** The includes ************
*/

#include <avr/io.h>
#include <util/delay.h>
#include "../../LIB/STD_Types.h"
#include "../../LIB/BIT_Math.h"
#include "../../MCAL/DIO/DIO.h"
#include "LCD.h"

/*
* ****************** Initializing LCD ************
*/
void LCD_voidInit(void){
    /*
    * ****************** setting Data port as output ************
    */
    DIO_voidSetPortDirection(LCD_DATA_PORT, 0XFF);
    /*
    * ****************** Setting RS, RW and EN pins out output ************
    */
    DIO_voidSetPinDirection(LCD_CONTROL_PORT, RS, DIO_OUTPUT);
    DIO_voidSetPinDirection(LCD_CONTROL_PORT, RW, DIO_OUTPUT);
    DIO_voidSetPinDirection(LCD_CONTROL_PORT, EN, DIO_OUTPUT);

    _delay_ms(35);
    LCD_voidSendCommand(FUNCTION_SET);
    _delay_us(50);
    LCD_voidSendCommand(DISPLAY_ON_OFF);
    _delay_us(50);
    LCD_voidSendCommand(CLEAR);
    _delay_ms(2);
}

/*
* ****************** sequence for preparing LCD to recieve commands ************
*/
void LCD_voidSendCommand(u8 Copy_u8Command){

    DIO_voidSetPinValue(LCD_CONTROL_PORT, RS, DIO_LOW);
    DIO_voidSetPinValue(LCD_CONTROL_PORT, RW, DIO_LOW);
    DIO_voidSetPortValue(LCD_DATA_PORT, Copy_u8Command);
    DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_HIGH);
    _delay_us(1);
    DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_LOW);
}
```

Figure 4-36: LCD.c 1

```
/*
* ****************** Sequence for sending a single English character to LCD ************
*/
void LCD_voidSendChar(u8 Copy_u8Char){

    DIO_voidSetPinValue(LCD_CONTROL_PORT, RS, DIO_HIGH);
    DIO_voidSetPinValue(LCD_CONTROL_PORT, RW, DIO_LOW);
    DIO_voidSetPortValue(LCD_DATA_PORT, Copy_u8Char);
    DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_HIGH);
    _delay_us(1);
    DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_LOW);

}

/*
* ****************** Sequence for sending English characters (whole string) to LCD ************
*/
void LCD_voidSendString (u8 *Copy_u8String){

    u8 Local_U8Counter ;
    for (Local_U8Counter = 0 ; Copy_u8String[Local_U8Counter] != '\0' ; Local_U8Counter++){
        LCD_voidSendChar(Copy_u8String[Local_U8Counter]);}


}

/*
* ****************** Sequence for selecting the position of display on LCD ************
*/
void LCD_voidSetLocation(u8 Copy_u8LineNum, u8 Copy_u8CharNum){

    switch(Copy_u8LineNum)
    {
    case LCD_LINE1 : LCD_voidSendCommand(0x80 + Copy_u8CharNum); break;

    case LCD_LINE2 : LCD_voidSendCommand(0xC0 + Copy_u8CharNum); break;

    }

}

```

Figure 4-37: LCD.c 2

```
/*
* ****************** Sequence for sending numbers (integers) to LCD ************
*/
void LCD_voidSendNumber (u32 Copy_u32Number){
    u8 Local_u8ASingleNum[11], Local_u8Count = 9;

    if (Copy_u32Number == 0) { LCD_voidSendChar('0');;}
    else{
        Local_u8ASingleNum[10] = '\0' ;
        while (Copy_u32Number != 0)
        {
            Local_u8ASingleNum[Local_u8Count] = ((Copy_u32Number % 10) + '0') ;
            Copy_u32Number /= 10 ;
            Local_u8Count--;
        }
        /* send address of the first number in my array till the '\0' */
        LCD_voidSendString(Local_u8ASingleNum + Local_u8Count + 1);
    }
}


/*
* ****************** Sequence for defining a special character to LCD ************
*/
void LCD_voidDrawSpecialChar(u8 Copy_u8CharIndex, u8 *Copy_u8SpecialChar){

    LCD_voidSendCommand(0b01000000+(Copy_u8CharIndex * 8));
    _delay_us(40);
    u8 LCD_U8Counter ;
    for (LCD_U8Counter=0 ; LCD_U8Counter<8 ; LCD_U8Counter++)
    {
        LCD_voidSendChar(Copy_u8SpecialChar[LCD_U8Counter]);
    }


}

/*
* ****************** Sequence for sending a special character to LCD ************
*/
void LCD_voidSendSpecialChar(u8 Copy_u8CharIndex){

    LCD_voidSendChar(Copy_u8CharIndex);
}
```

Figure 4-38: LCD.c 3

4.4.2 KEYPAD

Figure 4-39:KEYPAD

A 4x4 keypad packs 16 buttons into a compact grid, offering a versatile input method for microcontrollers. Each key sits at the intersection of a row and column, creating a matrix that simplifies wiring. By scanning the rows and columns electrically, the microcontroller can identify which button is pressed. This allows for easy user input for data entry, menu navigation, or triggering various functions in your project. These keypads come in pre-built modules or individual buttons for custom layouts, making them a popular choice for building user interfaces for embedded systems.

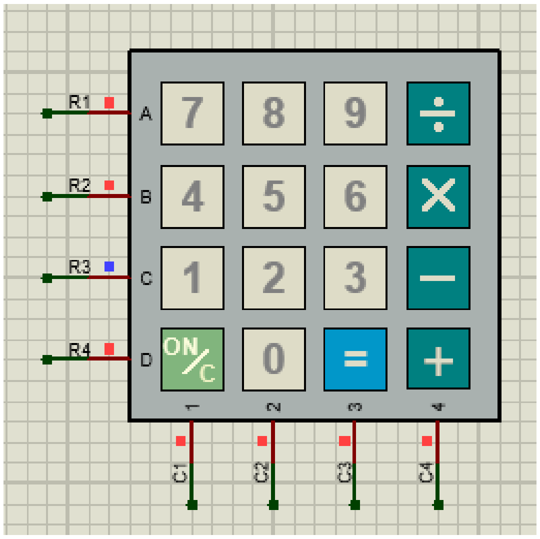


Figure 4-40: KEYPAD in simulation

KEYPAD.h

```c
/*
 * KEYPAD.h
 */
#ifndef KEYPAD_H_
#define KEYPAD_H_

/*
 ************* Configuration ************************************************
 */

/*
 ************* Select kpd port ************************
 */
#define KPD_PORT        DIO_PORTA

/*
 ************* Row pins ( Output pins) ****************
 */
#define KPD_R1_PIN      DIO_PIN0
#define KPD_R2_PIN      DIO_PIN1
#define KPD_R3_PIN      DIO_PIN2
#define KPD_R4_PIN      DIO_PIN3

/*
 ************* Columns pins ( Input pins) ************
 */
#define KPD_C1_PIN      DIO_PIN4
#define KPD_C2_PIN      DIO_PIN5
#define KPD_C3_PIN      DIO_PIN6
#define KPD_C4_PIN      DIO_PIN7
/*
 ************* Configuring Keys layout ****************
 */
#define KPD_KEYS        {{ '7', '8', '9', '/'},\
                         { '4', '5', '6', '*'}, \
                         { '1', '2', '3', '-'},\
                         { 'C', '0', '=', '+'}}
/*
 ************* Defining keyPad specific macro *********
 */
#define KPD_CHECK_BUTTON_PRESSED_OR_NOT     0XFF
/*
 ************* APIs ****************************************************
 */

void KPD_voidInit(void);
u8   KPD_u8GetPressedKey(void);

#endif /* HAL_KEYPAD_KEYPAD_H_ */
```

Figure 4-41: KEYPAD.h

KEYPAD.c

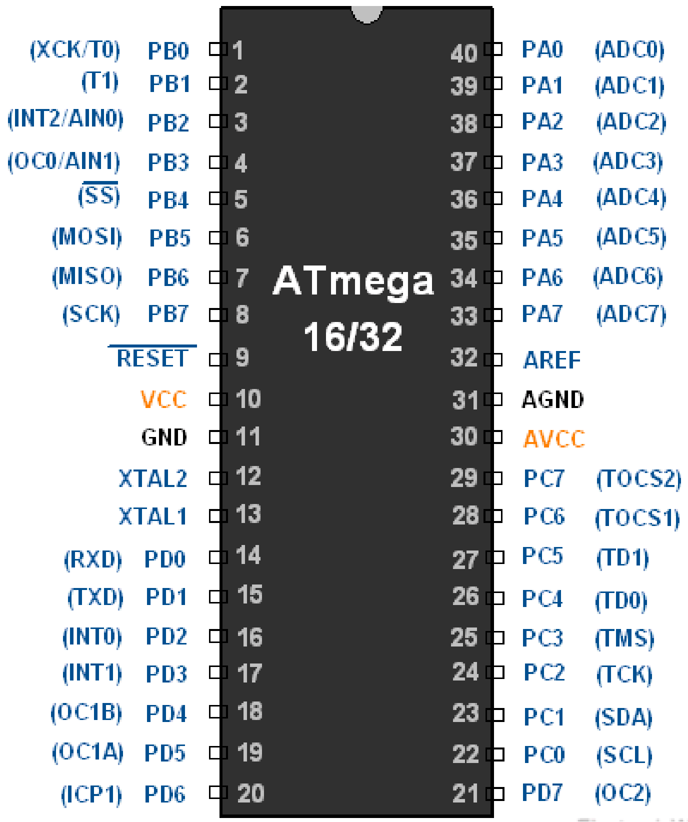
```c
/*
 * KEYPAD.c
 */

/*
 ************** The Includes ****************
 */
#include <avr/io.h>
#include <util/delay.h>
#include "../../LIB/STD_Types.h"
#include "../../LIB/BIT_Math.h"
#include "../../MCAL/DIO/DIO.h"
#include "KEYPAD.h"

/*
 ************** Mapping Keys to Pins ****************
 */
u8 KPD_Au8Keys[4][4] = KPD_KEYS;

/*
 ************** Rows are set to be output ****************
 */
u8 KPD_Au8RowPins[4] = {KPD_R1_PIN, KPD_R2_PIN, KPD_R3_PIN, KPD_R4_PIN};
/*
 ************** Columns are set to be input ****************
 */
u8 KPD_Au8ColPins[4] = {KPD_C1_PIN, KPD_C2_PIN, KPD_C3_PIN, KPD_C4_PIN};


/*
 ************** Initializing KeyPad ****************
 */
void KPD_voidInit(void){

    u8 Local_u8Count;
    for (Local_u8Count = 0; Local_u8Count < 8; ++Local_u8Count) {
        if (Local_u8Count < 4) {
            DIO_voidSetPinDirection(KPD_PORT, KPD_Au8RowPins[Local_u8Count], DIO_OUTPUT);
        } else {
            DIO_voidSetPinDirection(KPD_PORT, KPD_Au8ColPins[Local_u8Count-4], DIO_INPUT);
        }
    }
    DIO_voidSetPortValue(KPD_PORT, 0xff);
}

```

Figure 4-42: KEYPAD.c 1

```c
/*
 ************* Getting Key Values ****************
 */
u8 KPD_u8GetPressedKey(void){

    u8 Local_u8RowCount, Local_u8ColCount, Local_u8Pressed , Local_u8Flag =0;
    u8 Local_u8ReturnedVal = KPD_CHECK_BUTTON_PRESSED_OR_NOT;

    for (Local_u8RowCount = 0; Local_u8RowCount < 4; ++Local_u8RowCount) {

        /*
         ************* Applying zero voltage individually on rows ****************
         */
        DIO_voidSetPinValue(KPD_PORT, KPD_Au8RowPins[Local_u8RowCount], DIO_LOW);

        for (Local_u8ColCount = 0; Local_u8ColCount < 4; ++Local_u8ColCount) {
            /*
             ************* Checking if any button was pressed ****************
             */
            Local_u8Pressed = DIO_u8GetPinValue(KPD_PORT, KPD_Au8ColPins[Local_u8ColCount]);
            if(Local_u8Pressed == 0){
                /*
                 ************* Waiting for Debounceing ****************
                 */
                _delay_ms(25);
                /*
                 ************* Checking for noise ****************
                 */
                Local_u8Pressed = DIO_u8GetPinValue(KPD_PORT, KPD_Au8ColPins[Local_u8ColCount]);
                if(Local_u8Pressed == 0){
                    Local_u8ReturnedVal = KPD_Au8Keys[Local_u8RowCount][Local_u8ColCount];

                    /*
                     ************* Handling a long press ****************
                     */
                    while(Local_u8Pressed == 0){
                        Local_u8Pressed = DIO_u8GetPinValue(KPD_PORT, KPD_Au8ColPins[Local_u8ColCount]);
                    }
                    Local_u8Flag =1;
                    break;
                }
            }
        }
        DIO_voidSetPinValue(KPD_PORT, KPD_Au8RowPins[Local_u8RowCount], DIO_HIGH);
        if (Local_u8Flag == 1) {break;}
    }
    return Local_u8ReturnedVal;
}
```

Figure 4-43: KEYPAD.c 2

4.5 MAIN APP LAYER



The ATmega32 is a powerful yet versatile 8-bit microcontroller from Microchip Technology. Its heart lies in a high-performance AVR RISC architecture, allowing it to execute instructions swiftly and efficiently.

This translates to impressive processing power for a microcontroller of its size. Packed within its 40-pin frame, the ATmega32 boasts a treasure trove of features.

It offers a generous 32KB of flash memory for storing your program code, 2KB of SRAM for temporary data storage, and 1KB of EEPROM for non-volatile data that persists even after power cycles.

This memory combination empowers you to create feature-rich projects. But the ATmega32 doesn't stop at processing power. It integrates a multitude of peripherals that bridge the gap between the digital world of the microcontroller and the analog world we interact with.

An 8-channel 10-bit Analog-to-Digital Converter (ADC) lets you convert sensor readings like temperature or light into digital values.

Three flexible timers provide precise timing for delays, generating control signals, or keeping track of events.

Serial communication interfaces like SPI and USART enable communication with external devices such as displays or sensors.

Rounding out this feature set are power-saving modes, making the ATmega32 suitable for battery-powered applications.

Overall, the ATmega32 combines of processing power, memory, versatile peripherals, and low-power capabilities.

4.5.1 Master_ Microcontroller

The Main Microcontroller is responsible for processing the KEYPAD inputs and running the Security System, displaying data on the LCD, and sounding the alarm in case of a security breach of a fire, and cutting off the power grid when need be.

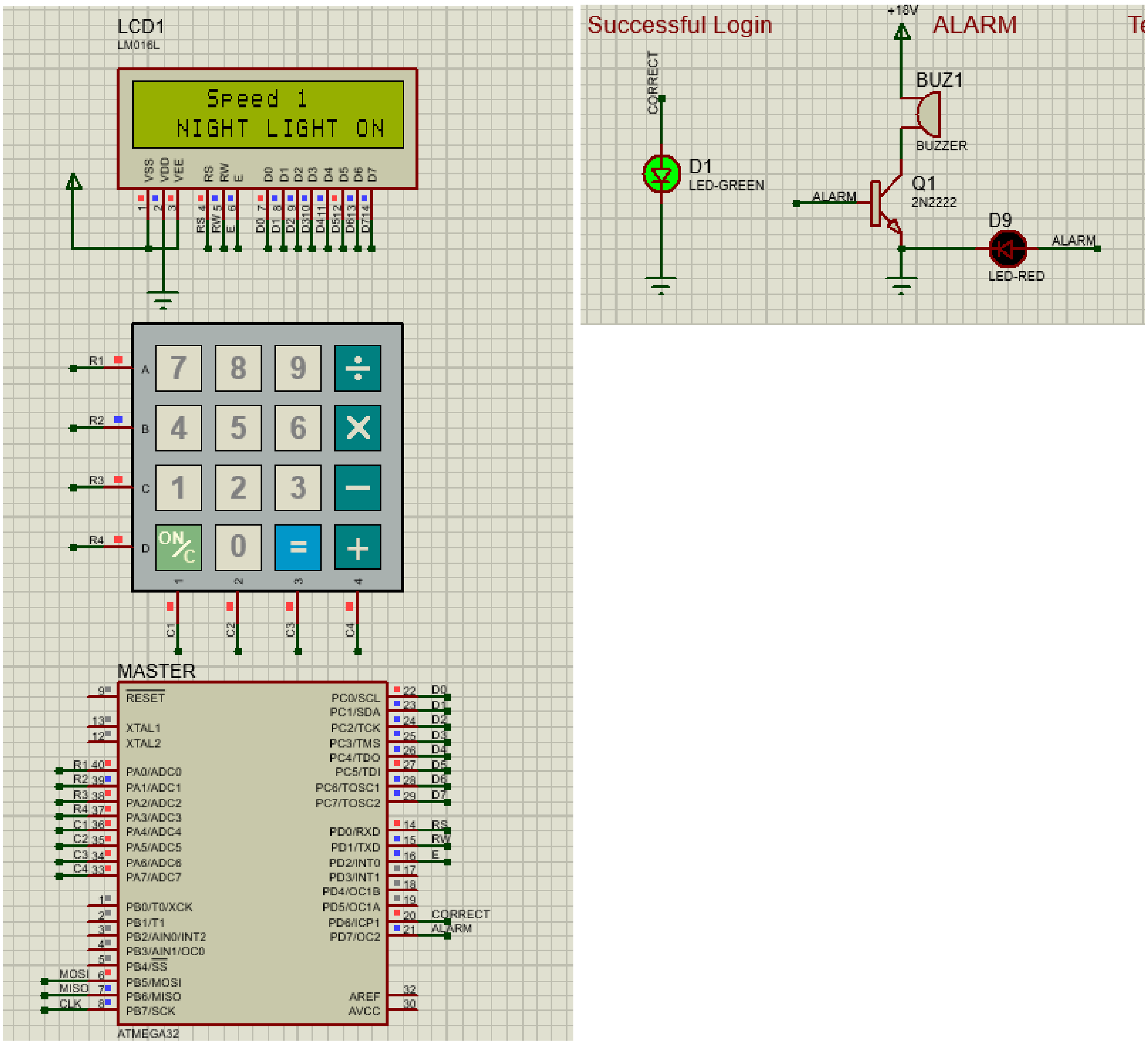


Figure 4-44: Master Microcontroller in simulation

```c
/*
 * main.c
 *  **************************************************************
 *  **************************************************************
 *  Master MicroController portion
 *
 *  **************************************************************
 *  **************************************************************
 */

/*
 * **************************** THE INCLUDES *********************
 */
#include <avr/io.h>
#include <util/delay.h>
#include "LIB/BIT_Math.h"
#include "LIB/STD_Types.h"
#include "MCAL/DIO/DIO.h"
#include "HAL/KEYPAD/KEYPAD.h"
#include "HAL/LCD/LCD.h"
#include "MCAL/INT/INT.h"
#include <avr/interrupt.h>
#include "MCAL/ADC/ADC.h"
#include "MCAL/TIMER0/TIM0.h"
#include "MCAL/SPI/SPI.h"

/*
 * ****************** SPI Communication dependent flag ***********
 */
u8 Global_SPI_Flag ='0';
void SPICommunication(void);

/*
 * Main Code base for the master controller in the home system
 */
int main(void){

    /*
     * ****************** Configuration for the SPI protocol *****
     */
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN5, DIO_OUTPUT);   // MOSI
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN6, DIO_INPUT);    // MISO
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN7, DIO_OUTPUT);   // CK
    _delay_ms(200);
    SPI_voidInit();
    _delay_ms(200);

```

Figure 4-45 Maser Microcontroller main.c 1

```
    /*
     * ****************** Configuration for the timer **************
     */
    TIM0_voidInit();
    TIM0_voidTimerStart(TIM0_PRESCALER_8);
    sei();
    TIM0_voidOvrINTControl(TIM0_TIN_EABLE);
    TIM0_voidOvfCallback(SPICommunication);

    /*
     * ******* Pin configuration Alarm and Home security indicator  *************
     */
    DIO_voidSetPinDirection(DIO_PORTD,DIO_PIN6,DIO_OUTPUT); //correct
    DIO_voidSetPinDirection(DIO_PORTD,DIO_PIN7,DIO_OUTPUT); //ALARM

    /*
     * ****************** Initializing LCD and KeyPad ***************************
     */
    LCD_voidInit();
    KPD_voidInit();

    /*
     * ****************** The password is '000' *********************************
     */
    u8 Local_u8APass[3] ;
    u8 Local_u8Key;
    u8 local_u8flag = 0;

    /*
     * ****************** Super Loop **********************************************
     */
    while(1){

        /*
         * ****************** Registering the keys pressed ***************************
         */
        for(u8 i =0; i<3;){
            Local_u8Key = KPD_u8GetPressedKey();
            if( Local_u8Key != KPD_CHECK_BUTTON_PRESSED_OR_NOT ){
                if(Local_u8Key == 'C'){
                    LCD_voidSendCommand(CLEAR);
                    _delay_ms(20);
                    DIO_voidSetPinValue(DIO_PORTD,DIO_PIN6,DIO_LOW);
                    DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_LOW);
                    local_u8flag =0;
                    i =0;
                    continue;
                }
```

Figure 4-46: Maser Microcontroller main.c 2

```c
            /*
             * ****************** Showing the pressed key on LCD *****************
             */
            Local_u8APass[i] = Local_u8Key;
            LCD_voidSetLocation(LCD_LINE1,i);
            LCD_voidSendChar(Local_u8Key);
            i++;
        }
    }
    /*
     * ****************** Checking if the password is correct  ***************************
     */
    if((Local_u8APass[0] == '0')&&(Local_u8APass[1] == '0')&&(Local_u8APass[2] == '0')){

        LCD_voidSendCommand(CLEAR);
        _delay_ms(20);
        LCD_voidSetLocation(LCD_LINE2,2);
        LCD_voidSendString("correct!");
        DIO_voidSetPinValue(DIO_PORTD,DIO_PIN6,DIO_HIGH);
        DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_LOW);
        local_u8flag = 0;
        Global_SPI_Flag = 'c';
    }
    /*
     * ****************** Allowing for three tries if entered incorrectly ****************
     */
    else{
        if(local_u8flag<=2){
            LCD_voidSendCommand(CLEAR);
            _delay_ms(20);
            LCD_voidSetLocation(LCD_LINE2,0);
            LCD_voidSendString("please try again!");
            Global_SPI_Flag = 'p';
            local_u8flag ++;
        }
    }
    /*
     * ****************** Calling Authority And setting Alarm off if password is
     * ****************** entered wrongly three times *****************************
     */
    if(local_u8flag == 3){
        LCD_voidSendCommand(CLEAR);
        _delay_ms(20);
        LCD_voidSetLocation(LCD_LINE2,2);
        LCD_voidSendString("wrong!");
        Global_SPI_Flag = 'w';
        DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_HIGH); //ALARM ON
        DIO_voidSetPinValue(DIO_PORTD,DIO_PIN6,DIO_LOW); //GREEN OFF
    }
```

Figure 4-47: Maser Microcontroller main.c 3

```
    }
    return 0;
}
/*
 * *************************** The SPI communication protocol code portion ************
 * *************************** to send and receive with the slave MicroController *****
 */
void SPICommunication(void){
    static u16 Local_u16Count =0;
    /*
     * *************************** Setting the right time to have SPI communicate ******
     */
    if (Local_u16Count == 4000){
        SPI_voidMasterSendData(Global_SPI_Flag);
        Local_u16Count = 0;

        u8 Local_u8Data = SPI_u8ReceiveData();
        switch(Local_u8Data){
        /*
         * ***************** Calling Authority  **********************************
         * ****************** In case of: 1- Wrong password three times **********
         * ***************************** 2- Fire Alarm / temperature of the house
         * *****************************     above 70 degrees
         */
        case'A' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Calling");
            LCD_voidSetLocation(LCD_LINE2,4);
            LCD_voidSendString("Authority!");
            DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_HIGH); //ALARM ON
            _delay_ms(750);
        break;
        /*
         * *************************** Showing the state of:
         *  ************************** 1- Time of day is before sunset
         *  ************************** 2 Temperature of the house is less than
         *  **************************   twenty one (21) degrees
         */
        case'1' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Speed 1");
            LCD_voidSetLocation(LCD_LINE2,2);
            LCD_voidSendString("NIGHT LIGHT OFF!");
            _delay_ms(750);
        break;
```

Figure 4-48: Maser Microcontroller main.c 4

```
        /*
         * **************************** Showing the state of:
         *  *************************** 1- Time of day is before sunset
         *  *************************** 2- Temperature of the house is between
         *  ***************************   twenty one (21) and thirty four (34) degrees
         */
        case'2' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Speed 2");
            LCD_voidSetLocation(LCD_LINE2,2);
            LCD_voidSendString("NIGHT LIGHT OFF!");
            _delay_ms(750);
        break;
        /*
         * **************************** Showing the state of:
         *  *************************** 1- Time of day is before sunset
         *  *************************** 2- Temperature of the house is between
         *  ***************************   thirty four (34) and fifty five (55) degrees
         */
        case'3' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Speed 3");
            LCD_voidSetLocation(LCD_LINE2,2);
            LCD_voidSendString("NIGHT LIGHT OFF!");
            _delay_ms(750);
        break;
        /*
         * **************************** Showing the state of:
         *  *************************** 1- Time of day is after sunset
         *  *************************** 2- Temperature of the house is less than
         *  ***************************   twenty one (21) degrees
         */
        case'4' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Speed 1");
            LCD_voidSetLocation(LCD_LINE2,2);
            LCD_voidSendString("NIGHT LIGHT ON!");
            _delay_ms(750);
        break;
```

Figure 4-49: Maser Microcontroller main.c 5

```c
        /*
         * ************************** Showing the state of:
         *  ************************** 1- Time of day is after sunset
         *  ************************** 2- Temperature of the house is between
         *  **************************   twenty one (21) and thirty four (34) degrees
         */
        case'5' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Speed 2");
            LCD_voidSetLocation(LCD_LINE2,2);
            LCD_voidSendString("NIGHT LIGHT ON!");
            _delay_ms(750);
        break;
        /*
         * ************************** Showing the state of:
         *  ************************** 1- Time of day is after sunset
         *  ************************** 2- Temperature of the house is between
         *  **************************   thirty four (34) and fifty five (55) degrees
         */
        case'6' :
            LCD_voidSendCommand(CLEAR);
            _delay_ms(2);
            LCD_voidSetLocation(LCD_LINE1,4);
            LCD_voidSendString("Speed 3");
            LCD_voidSetLocation(LCD_LINE2,2);
            LCD_voidSendString("NIGHT LIGHT ON!");
            _delay_ms(750);
        break;
        default:
            /*
             * ************************** reset manually with 'C' on keypad ********
             */
        break;

        }
    }
    Local_u16Count++;
}
```

Figure 4-50: Maser Microcontroller main.c 6

4.5.2 Slave _Microcontroller

The Slave Microcontroller is responsible for processing the data from all sensors, running the safety fire monitoring system, controlling the AC Unit and the illumination of the household, and sending relative data to the Main Microcontroller.

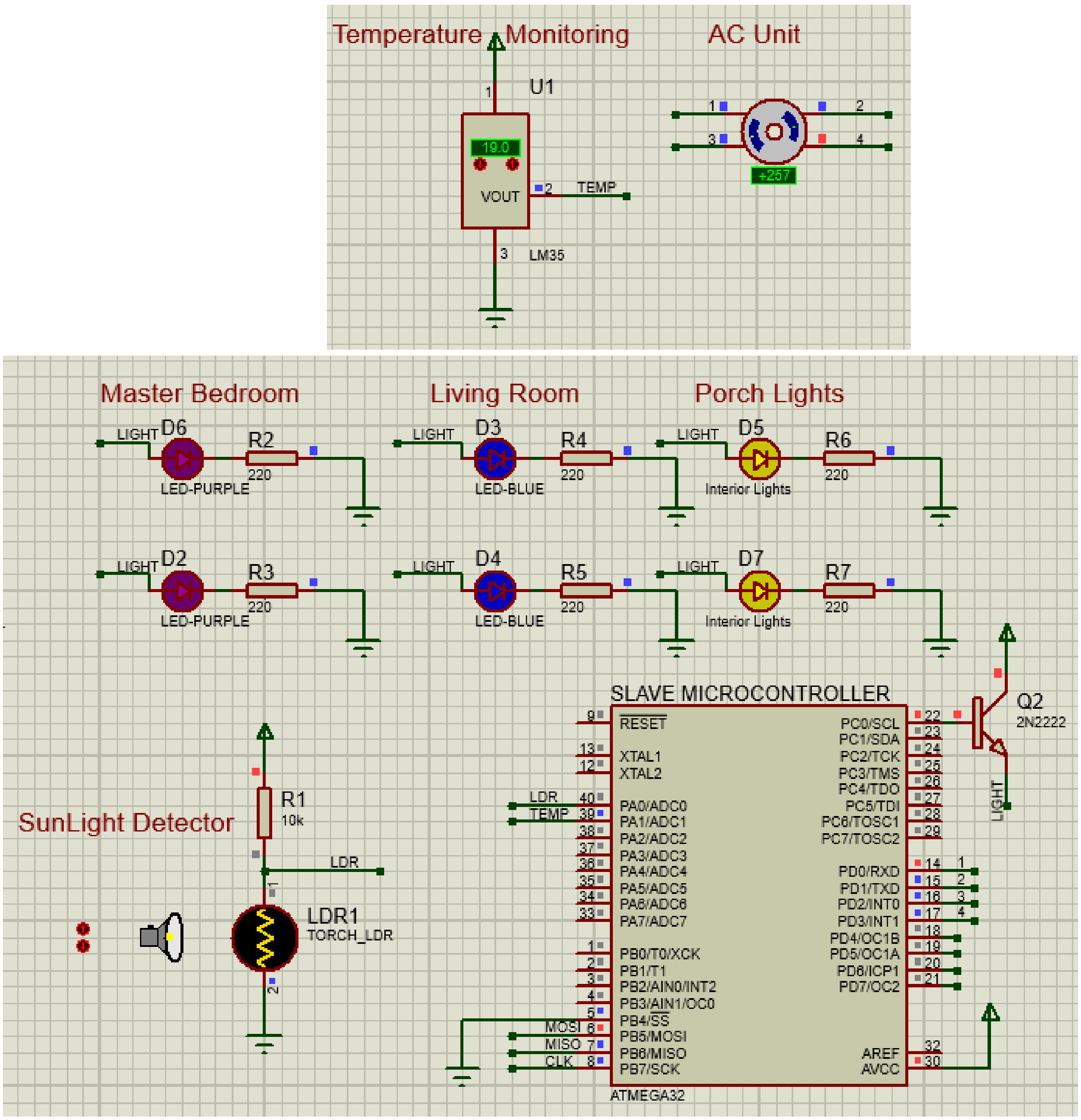


Figure 4-51: Slave Microcontroller in simulation

```c
/*
 * main.c
 * ************************************************************
 * ************************************************************
 * Slave MicroController portion
 *
 * ************************************************************
 * ************************************************************
 */

/*
 * *************************** THE INCLUDES *********************
 */

#include <avr/io.h>
#include <util/delay.h>
#include "LIB/BIT_Math.h"
#include "LIB/STD_Types.h"
#include "MCAL/DIO/DIO.h"
#include "HAL/KEYPAD/KEYPAD.h"
#include "HAL/LCD/LCD.h"
#include "MCAL/INT/INT.h"
#include <avr/interrupt.h>
#include "MCAL/ADC/ADC.h"
#include "HAL/Stepper_Motor/STPMR.h"
#include "MCAL/TIMER0/TIM0.h"
#include "MCAL/SPI/SPI.h"


/*
 * Main Code base for the slave controller in the home system
 */
int main(void){

    /*
     * ****************** Configuration for the SPI protocol *****
     */
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN4, DIO_INPUT);    // SS
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN5, DIO_INPUT);    // MOSI
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN6, DIO_OUTPUT);   // MISO
    DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN7, DIO_INPUT);    // CK
    _delay_ms(200);
    SPI_voidInit();
    _delay_ms(200);

    DIO_voidSetPinDirection(DIO_PORTC, DIO_PIN0, DIO_OUTPUT);

```

Figure 4-52: Slave Microcontroller main.c 1

```c
/*
 * ****************** Configuration ADC ********************
 */
DIO_voidSetPinDirection(DIO_PORTA, DIO_PIN0, DIO_INPUT);
ADC_voidInit(ADC_DIV_64);
STP_voidInit();
/*
 * ****************** Setting up variables for readings *****
 */
u16 Local_u16DigitalVlaueLDR, Local_u16AnaloglValueLDR;
u16 Local_u16DigitalVlaueTEMP, Local_u16AnaloglValueTEMP;


/*
 * ****************** Super Loop *****
 */
while(1){
    /*
     * ****************** Default value for LCD of nothing *****
     */
    u8 Local_u8TempFlag = 'z';
    u8 Local_u8NightLightFlag = 'z';
    u8 Local_u8SPIDataReceived = SPI_u8ReceiveData();

    /*
     * ****************** TEMP Sensor (always running ) *************
     */
    Local_u16DigitalVlaueTEMP = ADC_u16GetDigitalValue(ADC_Channel_1);
    Local_u16AnaloglValueTEMP = (Local_u16DigitalVlaueTEMP * 5000UL)/1024;

    /*
     * ****************** Fire Alarm (always running ) ****************
     */
    if(Local_u16AnaloglValueTEMP >= 570){
        Local_u8TempFlag = 'A';
        Local_u8NightLightFlag = 'A';
        //SPI_voidSalveSendData('l'); //l for alarm
        Local_u8SPIDataReceived = 'x'; //x is arbitrary constant
    }

```

Figure 4-53: Slave Microcontroller main.c 2

```c
        /*
         * ****************** Correct login credentials case ************
         */
        if(Local_u8SPIDataReceived == 'c'){


            /*
             * ****************** Measuring illumination with LDR *****
             */
            Local_u16DigitalVlaueLDR = ADC_u16GetDigitalValue(ADC_Channel_0);
            Local_u16AnaloglValueLDR = (Local_u16DigitalVlaueLDR * 5000UL)/1024;

            /*
             * ****************** Controlling the day cycles lights *****
             */
            if(Local_u16AnaloglValueLDR <= 1500){
                DIO_voidSetPinValue(DIO_PORTC, DIO_PIN0, DIO_LOW);
                Local_u8NightLightFlag = '0';
            }
            else if(Local_u16AnaloglValueLDR > 1500 ){
                DIO_voidSetPinValue(DIO_PORTC, DIO_PIN0, DIO_HIGH);
                Local_u8NightLightFlag = '1';
            }

            /*
             * ****************** Controlling the AC Unit ***************
             */
            if(Local_u16AnaloglValueTEMP <= 210){
                STP_voidRotate(STP_SPEED_1,STP_DIRECTION_CW);
                Local_u8TempFlag = '1';
            }
            else if((Local_u16AnaloglValueTEMP > 210)&&(Local_u16AnaloglValueTEMP <= 340)){
                STP_voidRotate(STP_SPEED_2,STP_DIRECTION_CW);
                Local_u8TempFlag = '2';
            }
            else if((Local_u16AnaloglValueTEMP > 340)&&(Local_u16AnaloglValueTEMP < 550)){
                STP_voidRotate(STP_SPEED_3, STP_DIRECTION_CW);
                Local_u8TempFlag = '3';
            }

```

Figure 4-54:Slave Microcontroller main.c 3

```
        /*
         * ****************** Wrong login credentials/ Fire Alarm case ************
         */
        }else{
            /*
             * ****************** Cutting off power to the house ************
             */
            DIO_voidSetPinValue(DIO_PORTC, DIO_PIN0, DIO_LOW); //NIGHT_LIGTH off
            STP_voidOff();                                      //AC Unit off
        }


        /*
         * ****************** Determining what will Show on the LCD ************
         */
        switch(Local_u8NightLightFlag){
        /*
         * ***************** Calling Authority  ***********************************
         * ****************** In case of: 1- Wrong password three times **********
         * *****************************  2- Fire Alarm / temperature of the house
         * *****************************     above 70 degrees
         */
            case 'A':
                SPI_voidSalveSendData('A');
            break;
            case '0':
                switch(Local_u8TempFlag){
                /*
                 * *************************** The state of:
                 *  ************************** 1- Time of day is before sunset
                 *  ************************** 2 Temperature of the house is less than
                 *  **************************   twenty one (21) degrees
                 */
                    case '1': SPI_voidSalveSendData('1'); break;
                /*
                 * *************************** Showing the state of:
                 *  ************************** 1- Time of day is before sunset
                 *  ************************** 2- Temperature of the house is between
                 *  **************************   twenty one (21) and thirty four (34) degrees
                */
                    case '2': SPI_voidSalveSendData('2'); break;
                /*
                 * *************************** Showing the state of:
                 *  ************************** 1- Time of day is before sunset
                 *  ************************** 2- Temperature of the house is between
                 *  **************************   thirty four (34) and fifty five (55) degrees
                 */
                    case '3': SPI_voidSalveSendData('3'); break;
                }
            break;
```

Figure 4-55: Slave Microcontroller main.c 4

```
            case '1':
                switch(Local_u8TempFlag){
                /*
                 * *************************** Showing the state of:
                 *  ************************** 1- Time of day is after sunset
                 *  ************************** 2- Temperature of the house is less than
                 *  **************************   twenty one (21) degrees
                 */
                    case '1': SPI_voidSalveSendData('4'); break;
                /*
                 * *************************** Showing the state of:
                 *  ************************** 1- Time of day is after sunset
                 *  ************************** 2- Temperature of the house is between
                 *  **************************   twenty one (21) and thirty four (34) degrees
                 */

                    case '2': SPI_voidSalveSendData('5'); break;
                /*
                 * *************************** Showing the state of:
                 *  ************************** 1- Time of day is after sunset
                 *  ************************** 2- Temperature of the house is between
                 *  **************************   thirty four (34) and fifty five (55) degrees
                 */

                    case '3': SPI_voidSalveSendData('6'); break;
                }

            break;
        }
    }
    return 0;
}
```

Figure 4-56: Slave Microcontroller main.