



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

ACKNOWLEDGMENT	I
TABLE OF CONTENTS	II
ABBREVIATIONS	IV
ABSTRACT	V
LIST OF FIGURES	6
LIST OF TABLES	IX
CHAPTER 1: INTRODUCTION	1
1.1 Problem statement	1
1.2 Project aims	2
1.3 Applications	3
CHAPTER 2: SYSTEM ARCHITECTURE AND FEATURE	4
2.1 Provided Features to the system	5
2.2 System architectures	6
CHAPTER 3: APPLICATIONS	8
3.1 Robust Security System	8
3.2 Fire Safety system	11
3.3 Time Dependent Illumination	13
3.4 Adaptive Air Conditioning System	15
3.5 Auto Irrigation	18
CHAPTER 4: IMPLEMENTATION DETAILS	19
4.1 The Layer Architecture	19

4.2 LIB Layer	21
4.3 MCAL Layer	23
4.4 HAL Layer	50
4.5 Main APP Layer	59

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

Figure 2-1: Conceptualization of modern household	4
Figure 1-1: System architecture	6
Figure 2-2: System implementation	7
Figure 3-1: Security system	8
Figure 3-2 Security System Implementation:	9
Figure 3-3: Fire Alarm	11
Figure 3-4: Safety system state	11
Figure 3-5: Controlled illumination	13
Figure 3-6: Illumination state	13
Figure 3-7: AC Unit	15
Figure 3-8: Adaptive AC state	16
Figure 3-9: Irrigation system states	18
Figure 4-1: Layer Architecture	19
Figure 4-2: Standard data types in C	21
Figure 4-3: STD_Types.h	21
Figure 4-4: Shifting bits	22
Figure 4-5: BIT_Math.h	22
Figure 4-6: PORTS and PINS arrangement in the ATmega32 Microcontroller	23
Figure 4-7: DIO.h 1	24
Figure 4-8: DIO.h 2	25
Figure 4-9: DIO.c 1	26
Figure 4-10: DIO.c 2	27
Figure 4-11: Interrupt using ISR	28

Figure 4-12: INT.h	29
Figure 4-13: INT.c 1	30
Figure 4-14: INT.c 2	31
Figure 4-15: INT.c 3	32
Figure 4-16: Timer	33
Figure 4-17: TIMER0.h 1	34
Figure 4-18: TIMER0.h 2	35
Figure 4-19: TIMER0.c 1	36
Figure 4-20: TIMER0.c	37
Figure 4-21: TIMER0.c 3	38
Figure 4-22: TIMER0.c 4	39
Figure 4-23: TIMER0.c 5	40
Figure 4-24: ADC	41
Figure 4-25: ADC.h	42
Figure 4-26: ADC.c 1	43
Figure 4-27: ADC.c 2	44
Figure 4-28: SPI	45
Figure 4-29: SPI.h	46
Figure 4-30: SPI.c 1	47
Figure 4-31: SPI.c 2	48
Figure 4-32: SPI.c 3	49
Figure 4-33: LCD	50
Figure 4-34: LCD running in simulation	50
Figure 4-35: LCD.h	51
Figure 4-36: LCD.c 1	52

Figure 4-37: LCD.c 2	53
Figure 4-38: LCD.c 3	54
Figure 4-39:KEYPAD	55
Figure 4-40: KEYPAD in simulation	55
Figure 4-41: KEYPAD.h	56
Figure 4-42: KEYPAD.c 1	57
Figure 4-43: KEYPAD.c 2	58
Figure 4-44: Master Microcontroller in simulation	61
Figure 4-45 Maser Microcontroller main.c 1	62
Figure 4-46: Maser Microcontroller main.c 2	63
Figure 4-47: Maser Microcontroller main.c 3	64
Figure 4-48: Maser Microcontroller main.c 4	65
Figure 4-49: Maser Microcontroller main.c 5	66
Figure 4-50: Maser Microcontroller main.c 6	67
Figure 4-51: Slave Microcontroller in simulation	68
Figure 4-52: Slave Microcontroller main.c 1	69
Figure 4-53: Slave Microcontroller main.c 2	70
Figure 4-54:Slave Microcontroller main.c 3	71
Figure 4-55: Slave Microcontroller main.c 4	72
Figure 4-56: Slave Microcontroller main.	73

List of Tables

Table 1-1: Abbreviations	IV
Table 3-1: Security System Implementation	9
Table 3-2: Safety System implementation	12
Table 3-3: Illumination implementation	14
Table 3-4: Adaptive AC implementation	16

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 an 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 mental 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:

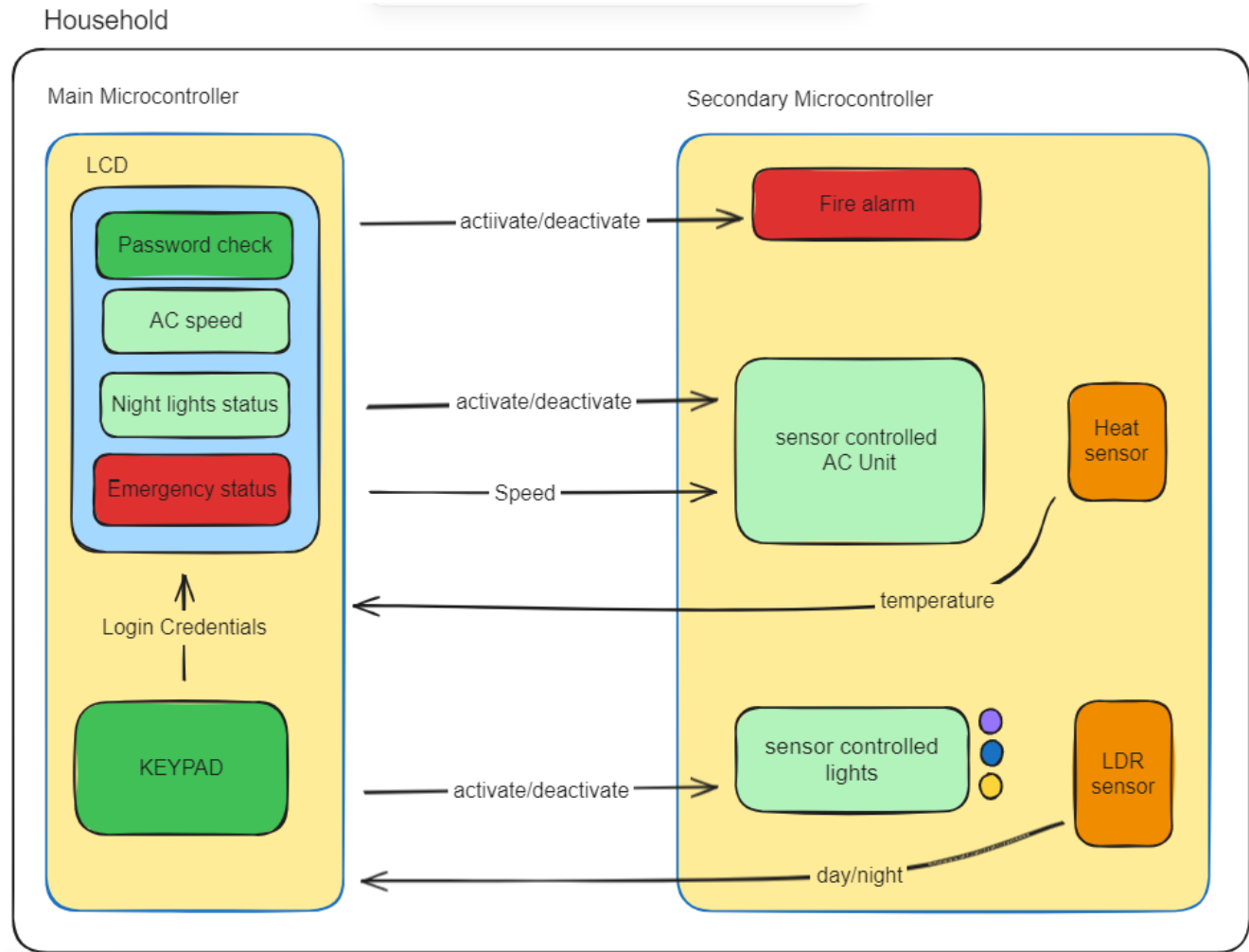


Figure 1-1: System architecture

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

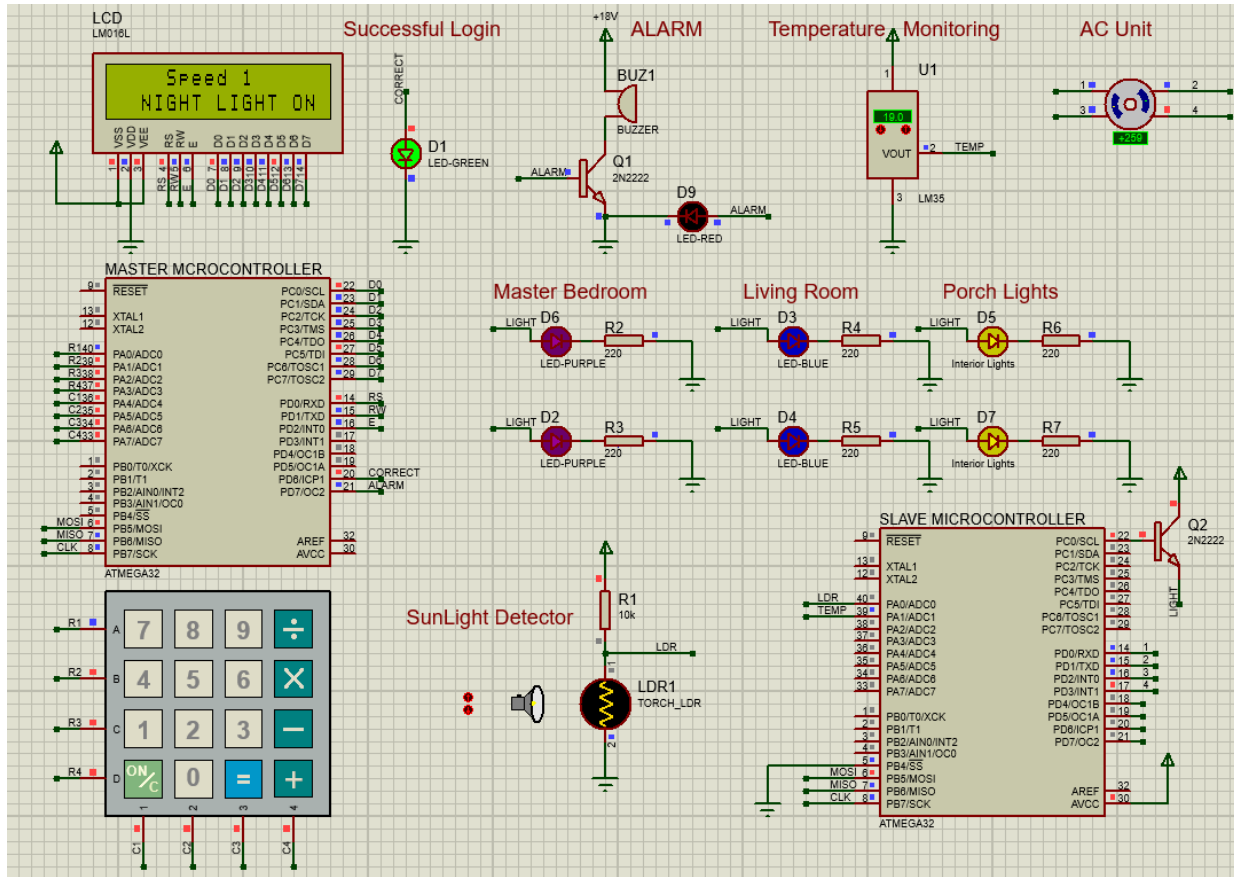


Figure 2-2: System implementation

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

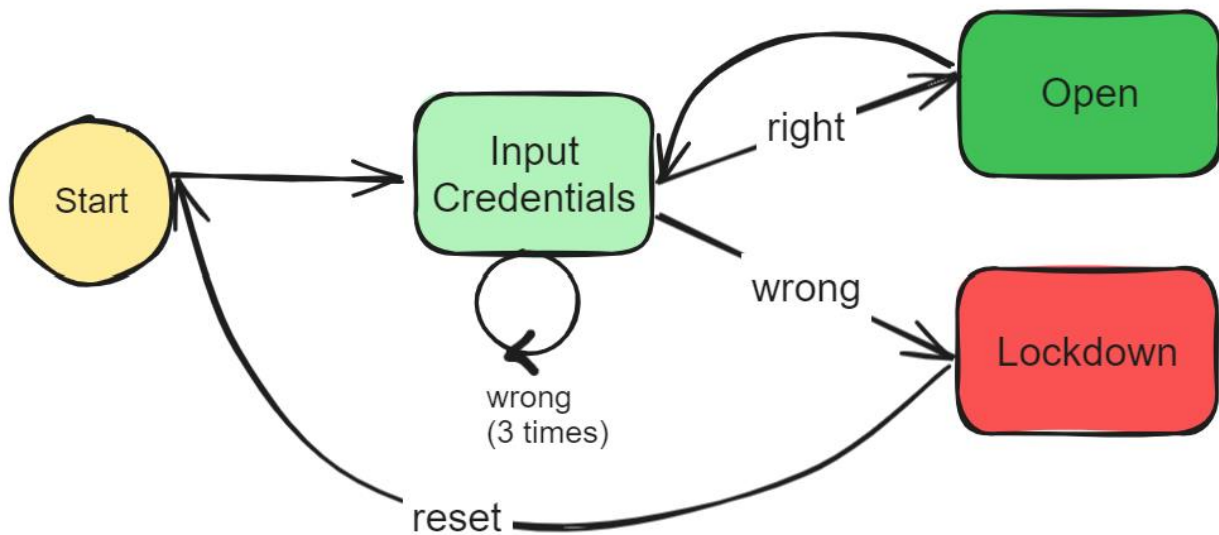
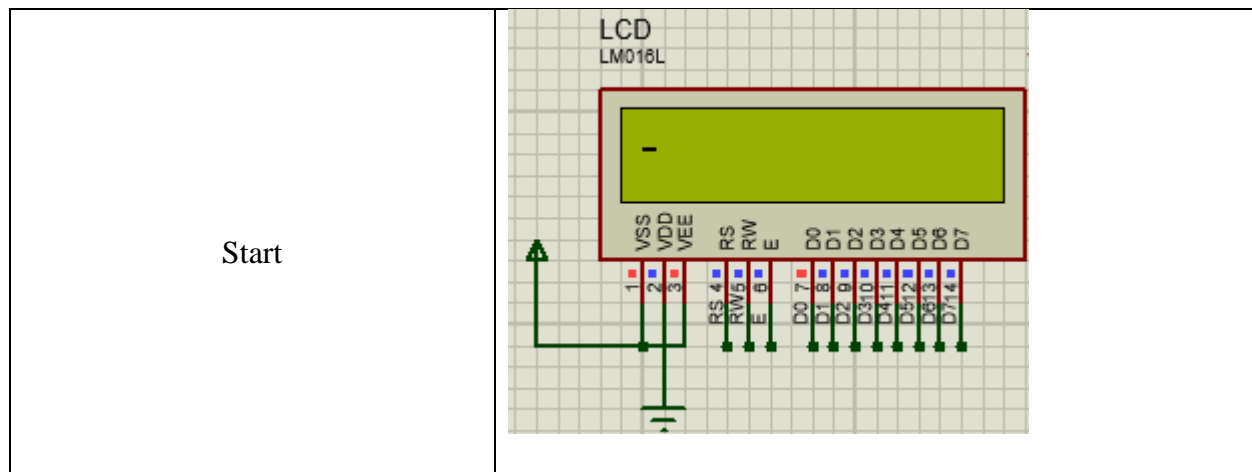
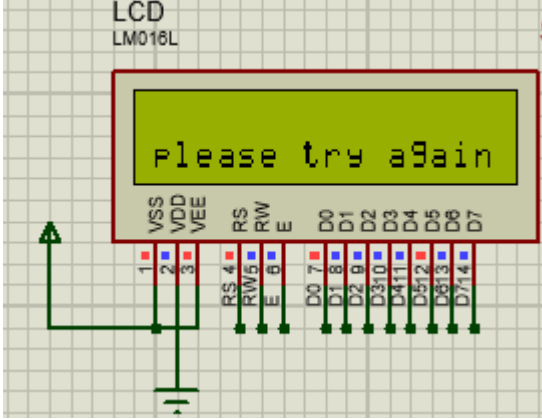
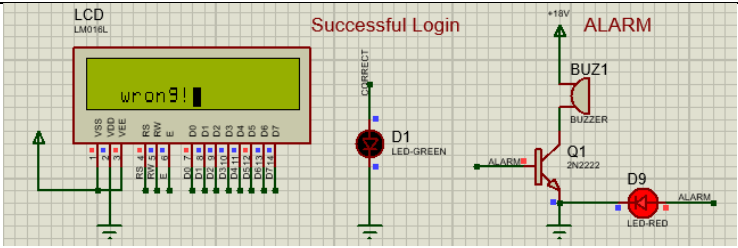
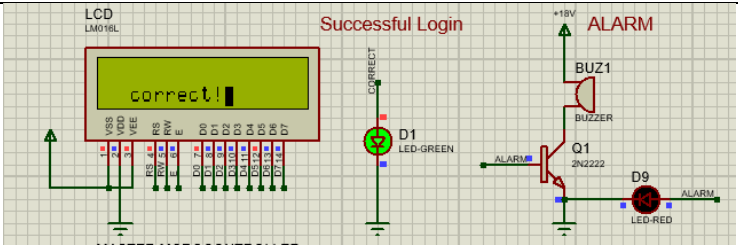


Figure 3-2 Security System Implementation:

3.1.2 Implementation

Table 3-1: Security System Implementation



<p>Input wrong credentials</p>	 <p>The diagram shows an LCD LM016L with the text 'please try again' on its screen. The pins are connected as follows: VSS (1) to ground, VDD (2) to +5V, VEE (3) to ground, RS (4) to ground, RW (5) to ground, E (6) to ground, D0 (7) to ground, D1 (8) to ground, D2 (9) to ground, D3 (10) to ground, D4 (11) to ground, D5 (12) to ground, D6 (13) to ground, and D7 (14) to ground.</p>
<p>Exceeding margin of error</p>	 <p>The diagram shows the LCD LM016L displaying 'wrong!'. The circuit includes a buzzer (BUZ1) and a red LED (D9) connected to an alarm signal. A green LED (D1) is also present, labeled 'LED-GREEN'. The text 'Successful Login' is visible above the LCD. The alarm circuit is powered by +18V and includes a 2N2222 transistor (Q1).</p>
<p>Right Credentials</p>	 <p>The diagram shows the LCD LM016L displaying 'correct!'. The circuit is identical to the previous one, but the green LED (D1) is now lit, indicating a successful login. The text 'Successful Login' is visible above the LCD. The alarm circuit is powered by +18V and includes a 2N2222 transistor (Q1).</p>

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

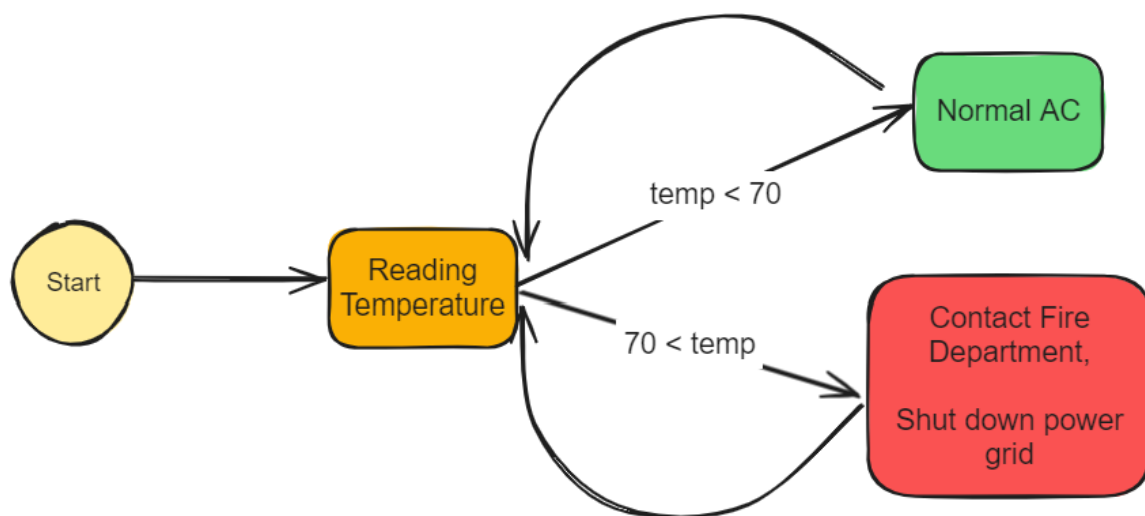
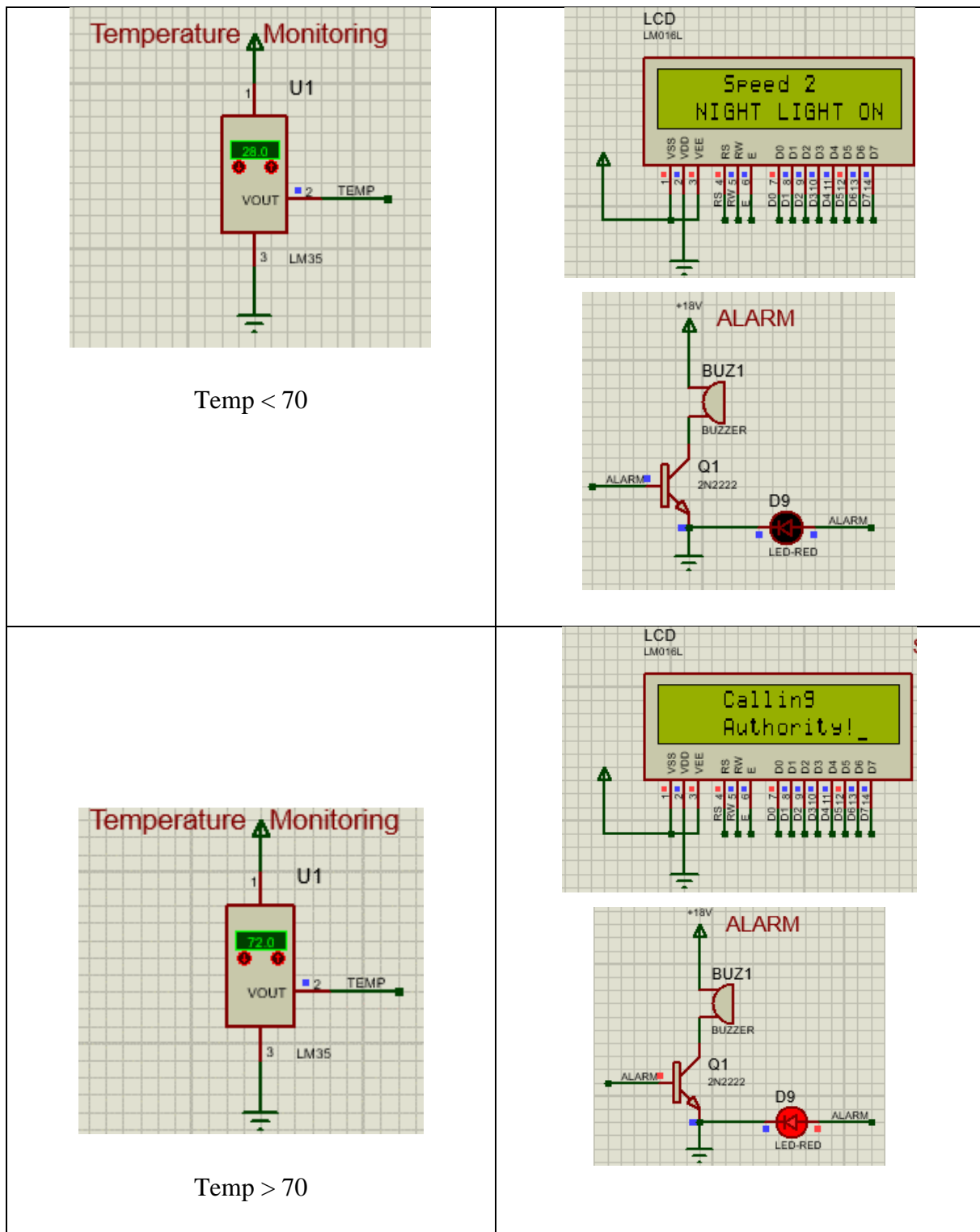


Figure 3-4: Safety system state

3.2.2 Implementation

Table 3-2: Safety System implementation



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

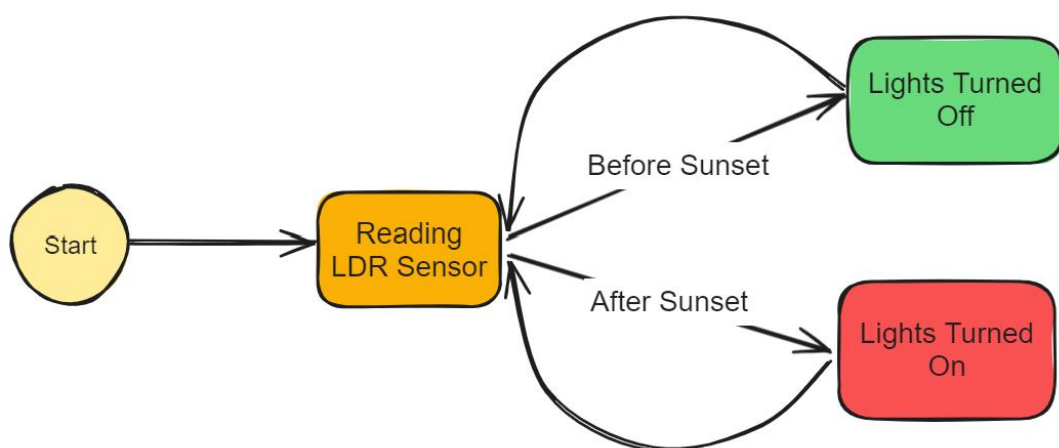
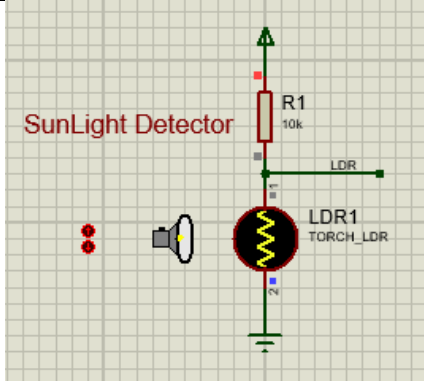
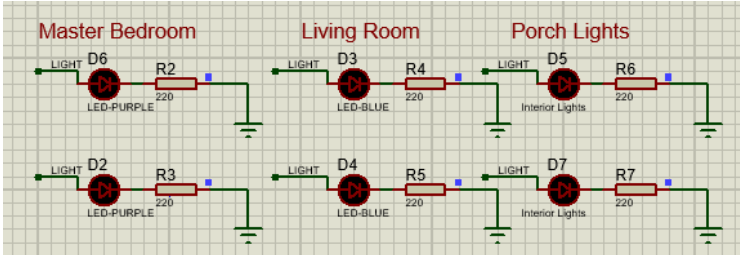
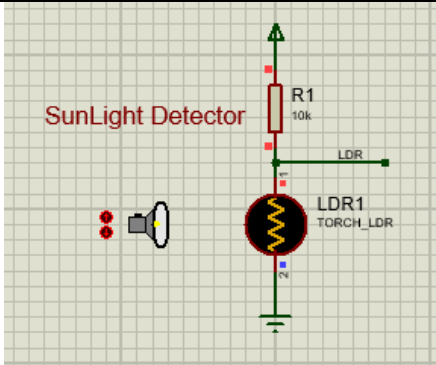
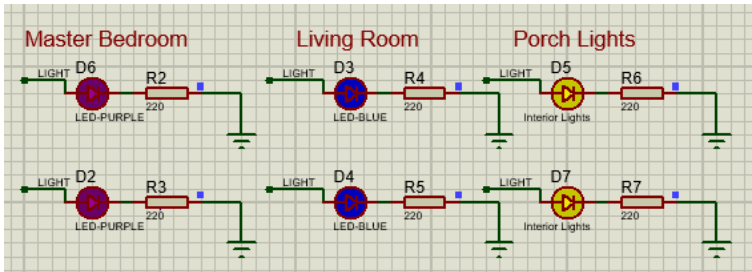


Figure 3-6: Illumination state

3.3.2 Implementation

Table 3-3: Illumination implementation

 <p>Before Sunset</p>	
 <p>After Sunset</p>	

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 heat 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

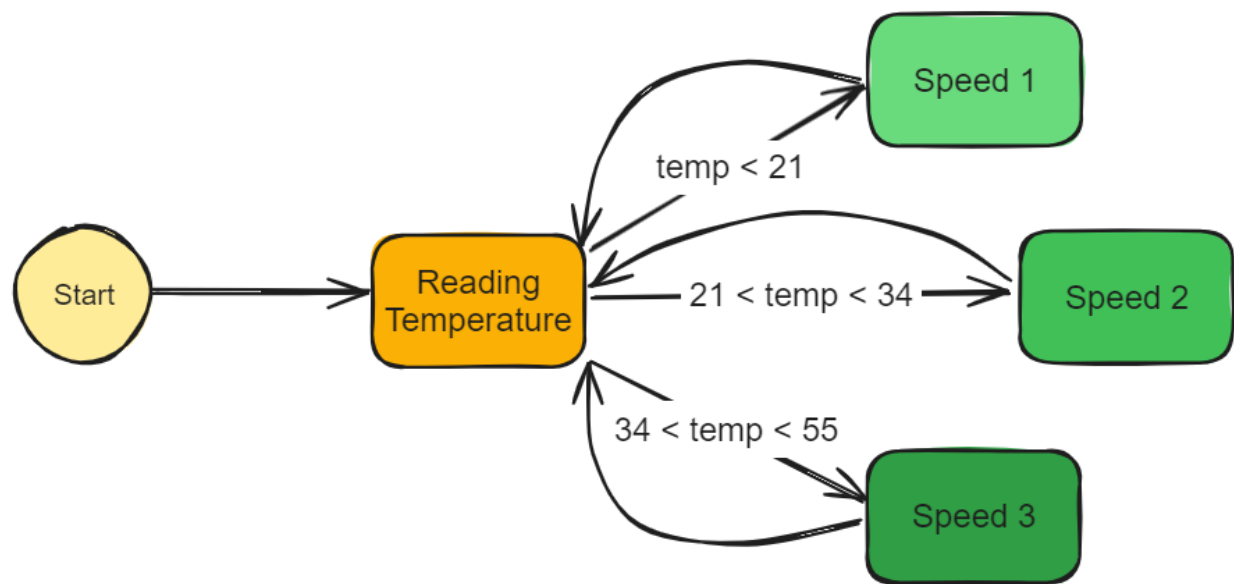
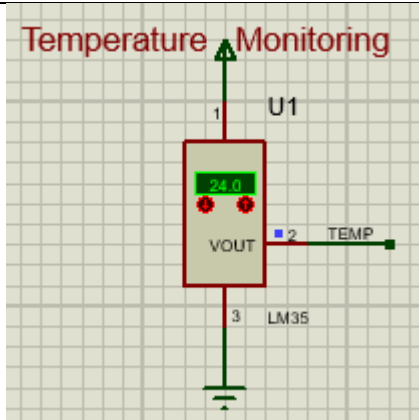


Figure 3-8: Adaptive AC state

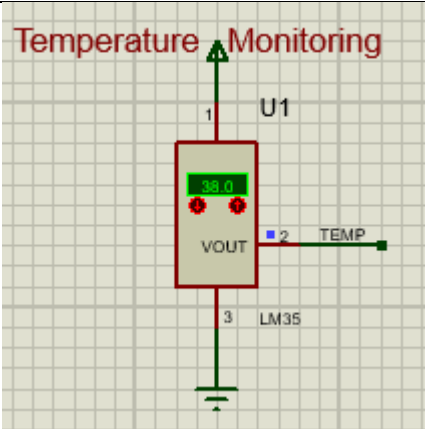
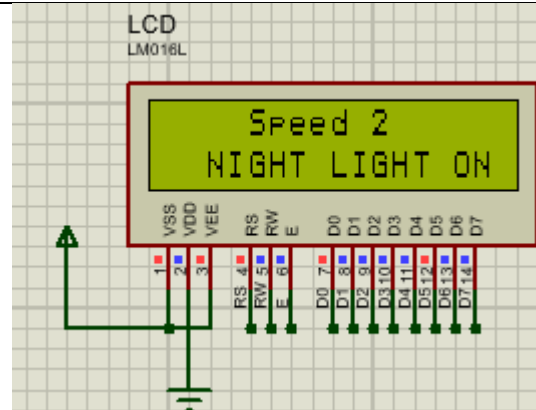
3.4.2 State representation

Table 3-4: Adaptive AC implementation

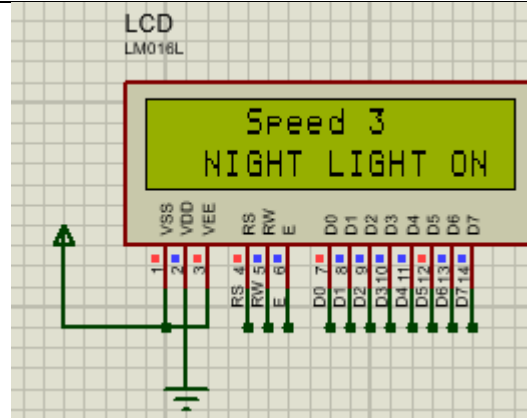
<p>The diagram shows a 'Temperature Monitoring' circuit. It features a component labeled 'U1' (LM35) with a digital display showing '19.0'. The output is labeled 'VOUT' and 'TEMP'. The component is connected to a ground symbol at the bottom. The label 'LM35' is also present.</p>	<p>The diagram shows an 'LCD LM016L' connected to a power source. The display shows 'Speed 1' and 'NIGHT LIGHT ON'. The pins are labeled: VSS, VDD, VEE, RS, RW, E, D0, D1, D2, D3, D4, D5, D6, D7. The display is connected to a ground symbol at the bottom.</p>
Temp < 21	



$21 < \text{Temp} < 34$



$34 < \text{Temp} < 55$



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

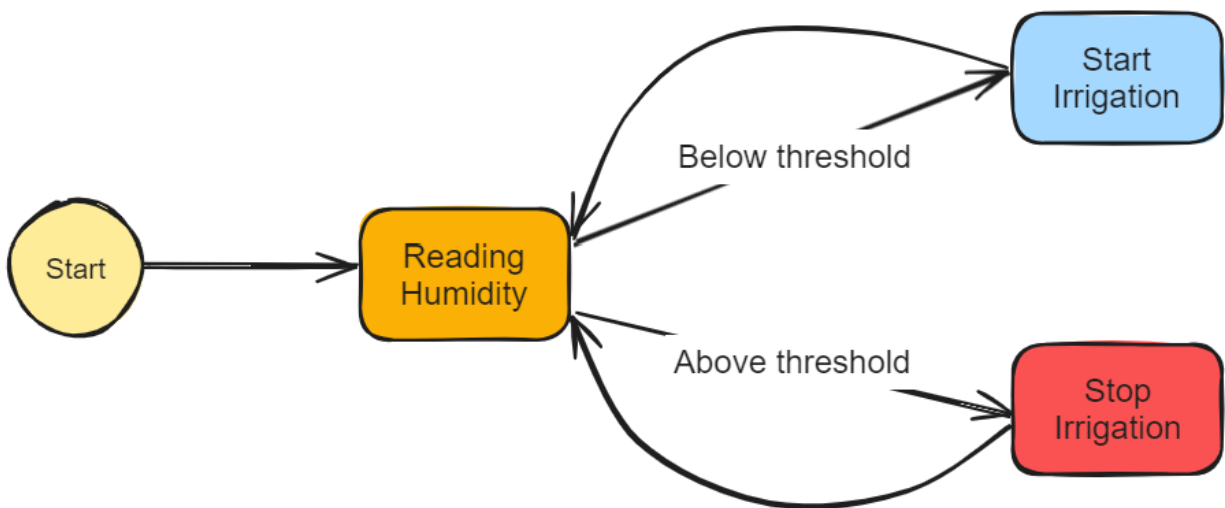


Figure 3-9: Irrigation system states

CHAPTER 4: Implementation Details

4.1 THE LAYER ARCHITECTURE

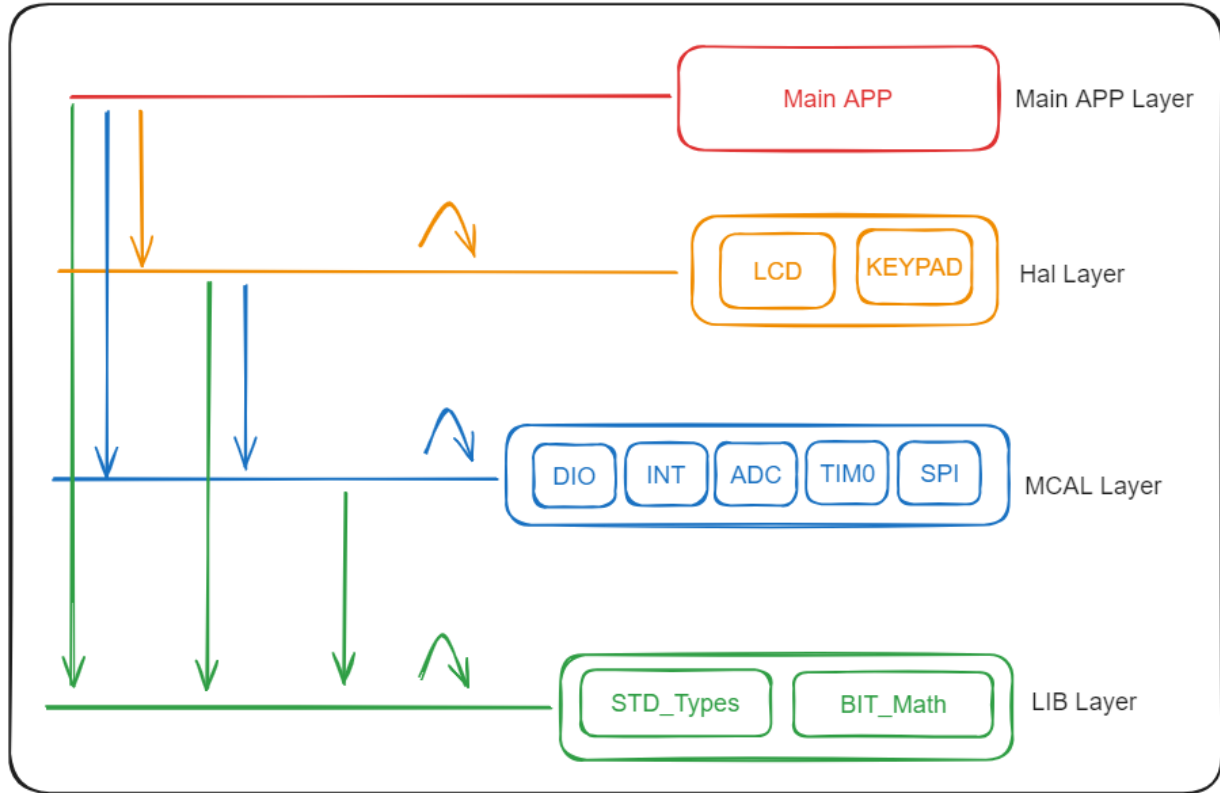


Figure 4-1: Layer Architecture

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.

Implementation Details

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 same layers can call each other, 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 signed int	2byte	-32768 to 32767
Unsigned int	2byte	0 to 65535

Figure 4-2: Standard data types in C

```

1  /*
2   * STD_Types.h
3   *
4   * Standard library - modified types
5   */
6
7  #ifndef STD_TYPES_H
8  #define STD_TYPES_H
9  /******
10
11  typedef unsigned char    u8    ;
12  typedef unsigned short int u16  ;
13  typedef unsigned long  int u32  ;
14  typedef signed char     s8     ;
15  typedef signed short  int s16  ;
16  typedef signed long   int s32  ;
17  typedef float          f32    ;
18  typedef double         f64    ;
19  typedef long double    f128   ;
20  typedef unsigned long long int u64 ;
21
22  /*user expected errors*/
23  typedef enum {
24      FUN_OK,
25      FUN_NOK,
26      PARAM_OUT_OF_RANGE ,
27  }enuErrorStatus;
28
29
30  #endif

```

Figure 4-3: STD_Types.h

4.2.2 BIT_Math.h

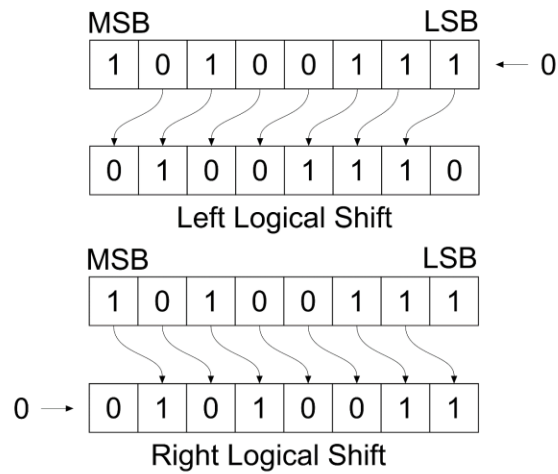


Figure 4-4: Shifting bits

```

1  /*
2   * BIT_Math.h
3   *
4   * Bitwise math operations on individual bits
5   */
6
7  #ifndef BIT_MATH_H_
8  #define BIT_MATH_H_
9
10 #define SET_BIT(REG,PIN) (REG |= (1<<PIN))
11 #define CLR_BIT(REG,PIN) (REG &= ~(1<<PIN))
12 #define TOG_BIT(REG,PIN) (REG ^= (1<<PIN))
13 #define GET_BIT(REG,PIN) ((REG>>PIN)&0x01)
14 #define ROT_L(REG,PIN) (REG = (REG<< PIN) | (REG >> (8-PIN)))
15 #define ROT_R(REG,PIN) (REG = (REG >> PIN) | (REG <<> (8-PIN)))
16
17
18 #endif /* BIT_MATH_H_ */

```

Figure 4-5: BIT_Math.h

4.3 MCAL LAYER

4.3.1 DIO

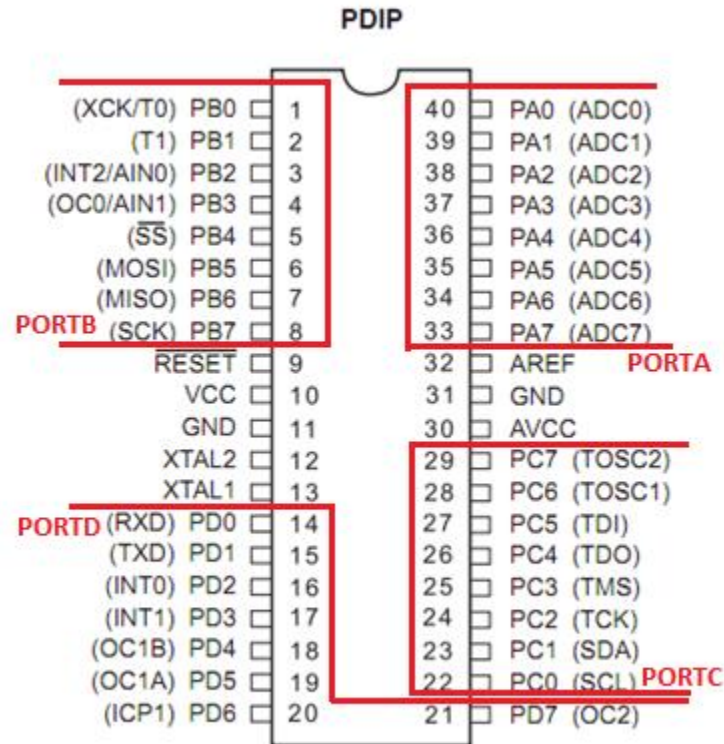


Figure 4-6: PORTS and PINS arrangement in the ATmega32 Microcontroller

DIO.h

```

1  /*
2   * DIO.h
3   *
4   * Digital Input Output
5   */
6
7  /*
8   ***** The Includes *****
9   */
10
11 #ifndef DIO_H
12 #define DIO_H
13 /*
14 ***** Defining Ports *****
15 */
16
17 #define DIO_PORTA 0
18 #define DIO_PORTB 1
19 #define DIO_PORTC 2
20 #define DIO_PORTD 3
21
22 /*
23 ***** Defining Pins *****
24 */
25 #define DIO_PIN0 0
26 #define DIO_PIN1 1
27 #define DIO_PIN2 2
28 #define DIO_PIN3 3
29 #define DIO_PIN4 4
30 #define DIO_PIN5 5
31 #define DIO_PIN6 6
32 #define DIO_PIN7 7
33
34 /*
35 ***** Defining Pin Input/Output *****
36 */
37 #define DIO_OUTPUT 1
38 #define DIO_INPUT 0
39
40 /*
41 ***** Defining Port Input/Output *****
42 */
43 #define DIO_OUTPUT_PORT 0xff
44 #define DIO_INPUT_PORT 0x00
45

```

Figure 4-7: DIO.h 1

Implementation Details

```
--
46 /*
47  ***** Defining pin High/Low *****
48  */
49  #define DIO_HIGH 1
50  #define DIO_LOW 0
51
52 /*
53  ***** Defining Port High/Low *****
54  */
55  #define DIO_HIGH_PORT 0xff
56  #define DIO_LOW_PORT 0x00
57
58 /*
59  ***** APIs *****
60  */
61
62 /*
63  ***** Port APIs *****
64  */
65  void DIO_voidSetPortDirection (u8 Copy_u8Port, u8 Copy_u8Direction);
66  void DIO_voidSetPortValue     (u8 Copy_u8Port, u8 Copy_u8Value);
67
68 /*
69  ***** Pin APIs *****
70  */
71  void DIO_voidSetPinDirection (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Direction);
72  void DIO_voidSetPinValue     (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Value);
73  u8   DIO_u8GetPinValue       (u8 Copy_u8Port, u8 Copy_u8Pin);
74
75  void DIO_voidTogglePin       (u8 Copy_u8Port, u8 Copy_u8Pin);
76
77
78  #endif
79
```

Figure 4-8: DIO.h 2

DIO.c

```

1  /*
2   * Dio.c
3   *
4   * Digital Input Output
5   */
6
7  /*
8   ***** The Includes *****
9   */
10 #include <avr/io.h>
11 #include "../LIB/STD_types.h"
12 #include "../LIB/BIT_Math.h"
13 #include "DIO.h"
14
15 /*
16 ***** Settings Port Direction *****
17 */
18 void DIO_voidSetPortDirection(u8 Copy_u8Port, u8 Copy_u8Direction){
19     switch(Copy_u8Port){
20         case DIO_PORTA:    DDRA = Copy_u8Direction;    break;
21         case DIO_PORTB:    DDRB = Copy_u8Direction;    break;
22         case DIO_PORTC:    DDRC = Copy_u8Direction;    break;
23         case DIO_PORTD:    DDRD = Copy_u8Direction;    break;
24     }
25
26 }
27
28 /*
29 ***** Settings Port Value *****
30 */
31 void DIO_voidSetPortValue    (u8 Copy_u8Port, u8 Copy_u8Value){
32     switch(Copy_u8Port){
33         case DIO_PORTA:    PORTA = Copy_u8Value;        break;
34         case DIO_PORTB:    PORTB = Copy_u8Value;        break;
35         case DIO_PORTC:    PORTC = Copy_u8Value;        break;
36         case DIO_PORTD:    PORTD = Copy_u8Value;        break;
37     }
38 }
39
40 /*
41 ***** Settings Pin Direction *****
42 */
43 void DIO_voidSetPinDirection (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Direction){
44     if(Copy_u8Direction == DIO_OUTPUT){
45         switch(Copy_u8Port){
46             case DIO_PORTA:    SET_BIT(DDRA,Copy_u8Pin);    break;
47             case DIO_PORTB:    SET_BIT(DDRB,Copy_u8Pin);    break;
48             case DIO_PORTC:    SET_BIT(DDRC,Copy_u8Pin);    break;
49             case DIO_PORTD:    SET_BIT(DDRD,Copy_u8Pin);    break;
50         }

```

Figure 4-9: DIO.c 1

Implementation Details

```
50     },
51     }
52     else{
53         switch(Copy_u8Port){
54             case DIO_PORTA: CLR_BIT(DDRA,Copy_u8Pin); break;
55             case DIO_PORTB: CLR_BIT(DDRB,Copy_u8Pin); break;
56             case DIO_PORTC: CLR_BIT(DDRC,Copy_u8Pin); break;
57             case DIO_PORTD: CLR_BIT(DDRD,Copy_u8Pin); break;
58         }
59     }
60 }
61
62 /*
63  ***** Settings Pin Value *****
64  */
65 void DIO_voidSetPinValue (u8 Copy_u8Port, u8 Copy_u8Pin, u8 Copy_u8Value){
66     if(Copy_u8Value == DIO_HIGH){
67         switch(Copy_u8Port){
68             case DIO_PORTA: SET_BIT(PORTA,Copy_u8Pin); break;
69             case DIO_PORTB: SET_BIT(PORTB,Copy_u8Pin); break;
70             case DIO_PORTC: SET_BIT(PORTC,Copy_u8Pin); break;
71             case DIO_PORTD: SET_BIT(PORTD,Copy_u8Pin); break;
72         }
73     }
74     else{
75         switch(Copy_u8Port){
76             case DIO_PORTA: CLR_BIT(PORTA,Copy_u8Pin); break;
77             case DIO_PORTB: CLR_BIT(PORTB,Copy_u8Pin); break;
78             case DIO_PORTC: CLR_BIT(PORTC,Copy_u8Pin); break;
79             case DIO_PORTD: CLR_BIT(PORTD,Copy_u8Pin); break;
80         }
81     }
82 }
83
84 /*
85  ***** Reading Pin Value *****
86  */
87 u8 DIO_u8GetPinValue (u8 Copy_u8Port, u8 Copy_u8Pin){
88     u8 value;
89     switch(Copy_u8Port){
90         case DIO_PORTA: value = GET_BIT(PINA,Copy_u8Pin); break;
91         case DIO_PORTB: value = GET_BIT(PINB,Copy_u8Pin); break;
92         case DIO_PORTC: value = GET_BIT(PINC,Copy_u8Pin); break;
93         case DIO_PORTD: value = GET_BIT(PIND,Copy_u8Pin); break;
94     }
95     return value;
96 }
97
```

Figure 4-10: DIO.c 2

4.3.2 Interrupt

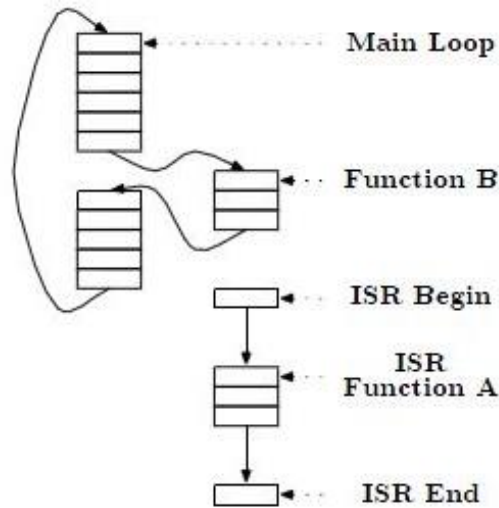


Figure 4-11: Interrupt using ISR

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

```

1  /*
2   * INT.h
3   *
4   * Interrupt
5   */
6
7  #ifndef INT_H_
8  #define INT_H_
9
10 /*
11  ***** Configuration *****
12  */
13
14 #define INT_INT0          0
15 #define INT_INT1          1
16 #define INT_INT2          2
17
18 /*
19  ***** Sense *****
20  */
21 #define INT_FALLING_EDGE    0
22 #define INT_RISING_EDGE    1
23 #define INT_ANY_CHANGE     2
24 #define INT_LOW_LEVEL      3
25
26 /*
27  ***** APIs *****
28  */
29 void INT_voidEnable(u8 Copy_u8INTIndex, u8 Copy_u8INTSense);
30 void INT_voidDisable(u8 Copy_u8INTIndex);
31
32
33 #endif /* MCAL_INT_INT_H_ */

```

Figure 4-12: INT.h

INT.c

```

1  /*
2   * INT.c
3   *
4   * Interrupt
5   */
6
7  /*
8   ***** The Includes *****
9   */
10 #include "../LIB/STD_Types.h"
11 #include "../LIB/BIT_Math.h"
12 #include <avr/interrupt.h>
13 #include "INT.h"
14
15 /*
16 ***** Enabling Interrupt *****
17 */
18 void INT_voidEnable(u8 Copy_u8INTIndex, u8 Copy_u8INTSense){
19
20     switch (Copy_u8INTIndex) {
21         /*
22          ***** In case of Interrupt 0 *****
23          */
24         case INT_INT0:
25             /*
26              ***** Selecting Sense *****
27              */
28             switch (Copy_u8INTSense) {
29                 case INT_FALLING_EDGE:
30                     CLR_BIT(MCUCR, ISC00);
31                     SET_BIT(MCUCR, ISC01);
32                     break;
33
34                 case INT_RISING_EDGE:
35                     SET_BIT(MCUCR, ISC00);
36                     SET_BIT(MCUCR, ISC01);
37                     break;
38
39                 case INT_ANY_CHANGE:
40                     CLR_BIT(MCUCR, ISC01);
41                     SET_BIT(MCUCR, ISC00);
42                     break;
43
44                 case INT_LOW_LEVEL:
45                     CLR_BIT(MCUCR, ISC00);
46                     CLR_BIT(MCUCR, ISC01);
47                     break;
48             }
49

```

Figure 4-13: INT.c 1

Implementation Details

```
50      /*
51          ***** Enabling Interrupt 0 *****
52      */
53      SET_BIT(GICR, INT0);
54      break;
55
56      /*
57          ***** In case of Interrupt 1 *****
58      */
59      case INT_INT1:
60          /*
61              ***** Selecting Sense *****
62          */
63          switch (Copy_u8INTSense) {
64              case INT_FALLING_EDGE:
65                  CLR_BIT(MCUCR, ISC10);
66                  SET_BIT(MCUCR, ISC11);
67                  break;
68
69              case INT_RISING_EDGE:
70                  SET_BIT(MCUCR, ISC10);
71                  SET_BIT(MCUCR, ISC11);
72                  break;
73
74              case INT_ANY_CHANGE:
75                  CLR_BIT(MCUCR, ISC11);
76                  SET_BIT(MCUCR, ISC10);
77                  break;
78
79              case INT_LOW_LEVEL:
80                  CLR_BIT(MCUCR, ISC10);
81                  CLR_BIT(MCUCR, ISC11);
82                  break;
83          }
84          /*
85              ***** Enabling Interrupt 1 *****
86          */
87          SET_BIT(GICR, INT1);
88          break;
89
90      /*
91          ***** In case of Interrupt 2 *****
92      */
```

Figure 4-14: INT.c 2

Implementation Details

```
93         case INT_INT2:
94             /*
95              * ***** Selecting Sense *****
96              */
97             switch (Copy_u8INTSense) {
98                 case INT_FALLING_EDGE:
99                     CLR_BIT(MCUCSR, ISC2);
100                     break;
101
102                 case INT_RISING_EDGE:
103                     SET_BIT(MCUCSR, ISC2);
104                     break;
105             }
106             /*
107              * ***** Enabling Interrupt 2 *****
108              */
109             SET_BIT(GICR, INT2);
110             break;
111     }
112 }
113
114
115 /*
116  * ***** Disabling Interrupt *****
117  */
118 void INT_voidDisable(u8 Copy_u8INTIndex){
119
120     switch (Copy_u8INTIndex) {
121         /*
122          * ***** Interrupt 0 *****
123          */
124         case INT_INT0:
125             CLR_BIT(GICR, INT0);
126             break;
127         /*
128          * ***** Interrupt 1 *****
129          */
130         case INT_INT1:
131             CLR_BIT(GICR, INT1);
132             break;
133         /*
134          * ***** Interrupt 2 *****
135          */
136         case INT_INT2:
137             CLR_BIT(GICR, INT2);
138             break;
139     }
140 }
```

Figure 4-15: INT.c 3

4.3.3 TIMER 0

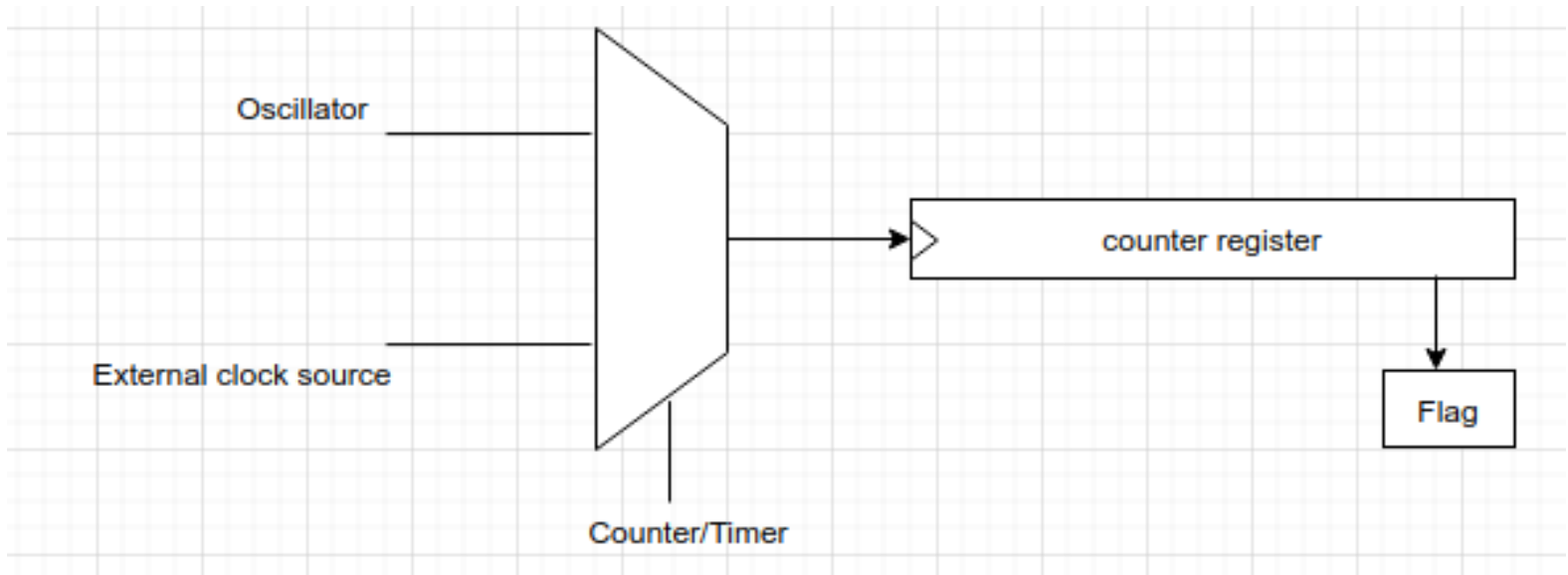


Figure 4-16: Timer

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

```

1  /*
2   * TIM0.h
3   *
4   * Timer 0
5   */
6
7  /*
8   * ***** Head Guard *****
9   */
10 #ifndef TIM0_H
11 #define TIM0_H
12
13 /*
14  * ***** Configuration (CFG) *****
15  */
16
17
18 /*
19  * ***** Select Mode *****
20  */
21 #define TIM0_MODE FAST_PWM
22 /* NORMAL      CTC      PWM_PHASE      FAST_PWM */
23
24
25 /*
26  * ***** Prescaler *****
27  */
28 #define TIM0_STOPPED 0
29 #define TIM0_NO_PRESCALER 1
30 #define TIM0_PRESCALER_8 2
31 #define TIM0_PRESCALER_64 3
32 #define TIM0_PRESCALER_256 4
33 #define TIM0_PRESCALER_1024 5
34 #define TIM0_EXT_FALLING 6
35 #define TIM0_EXT_RISING 7
36
37 /*
38  * ***** Interrupt control *****
39  */
40 #define TIM0_TIN_EABLE 1
41 #define TIM0_TIN_DISABLE 0
42
43
44 /*
45  * ***** CO0 Pin Action (NORMAL, CTC) *****
46  */
47 #define TIM0_OC0_IN_OUT_PIN 0
48 #define TIM0_OC0_TOGGLE_PIN 1
49 #define TIM0_OC0_CLEAR_PIN 2
50 #define TIM0_OC0_SET_PIN 3

```

Figure 4-17: TIMER0.h 1

Implementation Details

```
51
52
53 /*
54  ***** CO0 Pin Action (FAST PWM) *****
55  */
56 #define TIM0_OC0_SET_CMP_CLR_OVF          1
57 #define TIM0_OC0_CLR_CMP_SET_OVF          2
58
59
60 /*
61  ***** CO0 Pin Action (PHASE PWM) *****
62  */
63 #define TIM0_OC0_SET_UPC_CLR_DNC          1
64 #define TIM0_OC0_CLR_UPC_SET_DNC          2
65
66
67 /*
68  ***** APIs *****
69  */
70
71 void TIM0_voidInit          (void);
72 void TIM0_voidSetPreValue   (u8 Copy_u8CounterStartVal);
73 void TIM0_voidTimerStart    (u8 Copy_u8Prescaler );
74 void TIM0_voidOvrINTControl (u8 Copy_u8InterruptConl );
75 void TIM0_voidSetCompareMat (u8 Copy_u8CmpMatVal, u8 Copy_u8OC0PinAction);
76 void TIM0_voidPWMSGenerator (u8 Copy_u8DutyCycle, u8 Copy_u8OC0PinAction);
77 void TIM0_voidSetCmpValue   (u8 Copy_u8CmpMatVal );
78 void TIM0_voidCmpINTControl (u8 Copy_u8InterruptConl );
79 void TIM0_voidTimerStop     ( void);
80 void TIM0_voidOvfCallback    ( void (*Copy_pfTIM0Ovf)(void) );
81 void TIM0_voidCmpCallback    ( void (*Copy_pfTIM0Cmp)(void) );
82
83 // returned type (* pf) (input parameters)
84
85
86 #endif /* MCAL_TIMER0_TIMER0_H_ */
87
```

Figure 4-18: TIMER0.h 2

TIMER0.c

```

1  /*
2   * TIMER0.c
3   *
4   * TIMER 0
5   *
6   */
7
8  /*
9   ***** The Includes *****
10  */
11  #include "../LIB/BIT_Math.h"
12  #include "../LIB/STD_Types.h"
13  #include <avr/io.h>
14  #include <avr/interrupt.h>
15
16  /*
17   ***** Defining Timer0 Parameters *****
18  */
19  #define NORMAL      0
20  #define CTC          1
21  #define PWM_PHASE   2
22  #define FAST_PWM     3
23
24
25  #include "TIMER0.h"
26
27
28  /*
29   ***** Global pointer to function *****
30  */
31  void (*TIM0_pfTIM0Ovf)(void);
32  void (*TIM0_pfTIM0Cmp)(void);
33
34
35  /*
36   ***** Initializing Timer0 *****
37  */
38  void TIM0_voidInit (void){
39      /*
40       ***** Selecting Timer0 mode *****
41      */
42      #if TIM0_MODE == NORMAL
43          CLR_BIT(TCCR0, WGM00);
44          CLR_BIT(TCCR0, WGM01);
45      #elif TIM0_MODE == CTC
46          CLR_BIT(TCCR0, WGM00);
47          SET_BIT(TCCR0, WGM01);
48      #elif TIM0_MODE == PWM_PHASE
49          CLR_BIT(TCCR0, WGM01);
50          SET_BIT(TCCR0, WGM00);

```

Figure 4-19: TIMER0.c 1

```

51 #else
52     SET_BIT(TCCR0, WGM00);
53     SET_BIT(TCCR0, WGM01);
54 #endif
55
56 }
57
58 /*
59  ***** Getting PreValue for the timer0
60  ***** to start counting from *****
61  */
62 void TIM0_voidSetPreValue (u8 Copy_u8CounterStartVal){
63     /* set timer start value */
64     TCNT0 = Copy_u8CounterStartVal;
65
66 }
67
68 /*
69  ***** Starting the Timer0 *****
70  */
71 void TIM0_voidTimerStart (u8 Copy_u8Prescaler){
72     /*
73      ***** Setting Prescaler *****
74      */
75     TCCR0 &= 0b11111000;
76     TCCR0 |= Copy_u8Prescaler;
77
78 }
79
80 /*
81  *****
82  */
83 void TIM0_voidOvrINTControl (u8 Copy_u8InterruptCon1){
84
85     if(Copy_u8InterruptCon1 == TIM0_TIN_EABLE)
86         {SET_BIT(TIMSK, TOIE0);}
87     else
88         {CLR_BIT(TIMSK, TOIE0);}
89
90 }
91

```

Figure 4-20: TIMER0.c

Implementation Details

```
92  /*
93  ***** Sets the value of the
94  ***** Output Compare Register(OCR0) for Timer0
95  ***** and controlling OC0 *****
96  */
97  void TIM0_voidSetCompareMat (u8 Copy_u8CmpMatVal, u8 Copy_u8OC0PinAction){
98
99      switch(Copy_u8OC0PinAction)
100      {
101          case TIM0_OC0_TOGGLE_PIN :
102              SET_BIT(TCCR0, COM00);
103              CLR_BIT(TCCR0, COM01);
104              break;
105
106          case TIM0_OC0_CLEAR_PIN :
107              SET_BIT(TCCR0, COM01);
108              CLR_BIT(TCCR0, COM00);
109              break;
110
111          case TIM0_OC0_SET_PIN :
112              SET_BIT(TCCR0, COM00);
113              SET_BIT(TCCR0, COM01);
114              break;
115
116          default :
117              CLR_BIT(TCCR0, COM00);
118              CLR_BIT(TCCR0, COM01);
119              break;
120      }
121
122  /*
123  ***** Setting compare value *****
124  */
125  OCR0 = Copy_u8CmpMatVal;
126
127  }
128
129  /*
130  ***** Generate Pulse Width Modulation
131  ***** (PWM) signal *****
132  */
133  void TIM0_voidPWMGenerator (u8 Copy_u8DutyCycle, u8 Copy_u8OC0PinAction){
134
```

Figure 4-21: TIMER0.c 3

Implementation Details

```
135 /*
136  ***** Fast PWM Mode *****
137  */
138 #if TIM0_MODE == FAST_PWM
139     if(Copy_u8OC0PinAction == TIM0_OC0_SET_CMP_CLR_OVF){
140         SET_BIT(TCCR0, COM00);
141         SET_BIT(TCCR0, COM01);
142         OCR0 = (256 - ((Copy_u8DutyCycle/100.0)*256));
143     }
144     else{
145         SET_BIT(TCCR0, COM01);
146         CLR_BIT(TCCR0, COM00);
147         OCR0 = ((Copy_u8DutyCycle/100.0)*256);
148     }
149 /*
150  ***** Phase Correct PWM Mode *****
151  */
152 #elif TIM0_MODE == PWM_PHASE
153     if(Copy_u8OC0PinAction == TIM0_OC0_SET_UPC_CLR_DNC){
154         SET_BIT(TCCR0, COM00);
155         SET_BIT(TCCR0, COM01);
156         OCR0 = (255 - (((Copy_u8DutyCycle/100.0)*510)/2));
157     }
158     else{
159         SET_BIT(TCCR0, COM01);
160         CLR_BIT(TCCR0, COM00);
161         OCR0 = (((Copy_u8DutyCycle/100.0)*510)/2);
162     }
163 #endif
164 }
165
166
167 }
168
169
170
171 /*
172  ***** Setting value for
173  ***** Output Compare Register (OCR0)*****
174  */
175 void TIM0_voidSetCmpValue (u8 Copy_u8CmpMatVal){
176     /* Set Compare value */
177     OCR0 = Copy_u8CmpMatVal;
178 }
179
180
```

Figure 4-22: TIMER0.c 4

Implementation Details

```
181 /*
182 ***** Enabling/Disabling compare match interrupt **
183 */
184 void TIM0_voidCmpINTControl (u8 Copy_u8InterruptCon1){
185
186     if(Copy_u8InterruptCon1 == TIM0_TIN_EABLE)
187         {SET_BIT(TIMSK, OCIE0);}
188     else
189         {CLR_BIT(TIMSK, OCIE0);}
190 }
191
192 /*
193 ***** Stopping Timer0 *****
194 */
195 void TIM0_voidTimerStop      (void){
196
197     TCCR0 &= 0b11111000;
198 }
199 /*
200 ***** Setting a callback function for
201 ***** when OverFlow Interrupt occurs*****
202 */
203 void TIM0_voidOvfCallback    ( void (*Copy_pfTIM0Ovf)(void) ){
204
205     TIM0_pfTIM0Ovf = Copy_pfTIM0Ovf;
206 }
207 /*
208 ***** Setting a callback function for
209 ***** when Compare Match Interrupt occurs*****
210 */
211 void TIM0_voidCmpCallback    ( void (*Copy_pfTIM0Cmp)(void) ){
212
213     TIM0_pfTIM0Cmp = Copy_pfTIM0Cmp;
214 }
215 /*
216 ***** Interrupt Service Routine (ISR)
217 ***** for the Timer0 Overflow Interrupt *****
218 */
219 ISR(TIMERO_OVF_vect){
220
221     TIM0_pfTIM0Ovf();
222 }
223 /*
224 ***** Interrupt Service Routine (ISR)
225 ***** for the Timer0 Compare Match Interrupt *****
226 */
227 ISR(TIMERO_COMP_vect){
228
229     TIM0_pfTIM0Cmp();
230 }
```

Figure 4-23: TIMERO.c 5

4.3.4 ADC

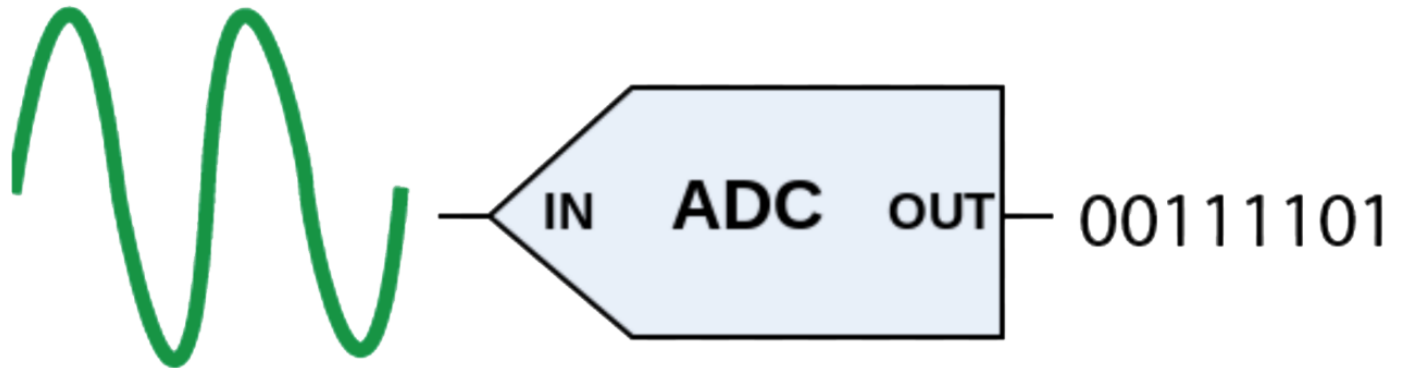


Figure 4-24: ADC

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

```

1  /*
2   * ADC.h
3   *
4   * Analog Digital Converter
5   */
6  /*
7   ***** Head Guard*****
8   */
9  #ifndef ADC_H
10 #define ADC_H
11
12 /*
13 ***** Configuration (CFG) *****
14 */
15 /*
16 ***** Choosing Reference Voltage *****
17 */
18 #define ADC_REF_VOLT      ADC_AVCC
19 /* ADC_AVCC      ADC_AREF      ADC_2_56_V */
20
21 /*
22 ***** Prescaler *****
23 */
24 #define ADC_DIV_2          1
25 #define ADC_DIV_4          2
26 #define ADC_DIV_8          3
27 #define ADC_DIV_16         4
28 #define ADC_DIV_32         5
29 #define ADC_DIV_64         6
30 #define ADC_DIV_128        7
31 /*
32 ***** Defining Channels *****
33 */
34 #define ADC_Channel_0      0
35 #define ADC_Channel_1      1
36 #define ADC_Channel_2      2
37 #define ADC_Channel_3      3
38 #define ADC_Channel_4      4
39 #define ADC_Channel_5      5
40 #define ADC_Channel_6      6
41 #define ADC_Channel_7      7
42 /*
43 ***** APIs *****
44 */
45 void ADC_voidInit(u8 Copy_u8Prescaler);
46 u16 ADC_u16GetDigitalValue(u8 Copy_u8Channel);
47 void ADC_voidDisable(void);
48
49 #endif /* MCAL_ADC_ADC_H */

```

Figure 4-25: ADC.h

ADC.c

```

1  /*
2   * ADC.c
3   *
4   * Analog Digital Converter
5   */
6
7  /*
8   * ***** The Includes *****
9   */
10 #include "../LIB/BIT_Math.h"
11 #include "../LIB/STD_Types.h"
12 #include <avr/io.h>
13
14 /*
15  * ***** Defining ADC parameters *****
16  */
17 #define ADC_AVCC      0
18 #define ADC_AREF      1
19 #define ADC_2_56_V    2
20
21 #include "ADC.h"
22
23 /*
24  * ***** Initializing ADC *****
25  */
26 void ADC_voidInit(u8 Copy_u8Prescaler){
27
28     /* Enable ADC */
29     SET_BIT(ADCSRA, ADEN);
30
31     /*
32      * ***** Select Vref *****
33      */
34     #if ADC_REF_VOLT == ADC_AVCC
35         SET_BIT(ADMUX, REFS0);
36         CLR_BIT(ADMUX, REFS1);
37     #elif ADC_REF_VOLT == ADC_AREF
38         CLR_BIT(ADMUX, REFS0);
39         CLR_BIT(ADMUX, REFS1);
40     #else
41         SET_BIT(ADMUX, REFS0);
42         SET_BIT(ADMUX, REFS1);
43     #endif
44
45     /*
46      * ***** Select right adjust *****
47      */
48     CLR_BIT(ADMUX, ADLAR);
49
50     /*

```

Figure 4-26: ADC.c 1

Implementation Details

```
51      ***** Prescaler *****
52      */
53      ADCSRA &= 0b11111000;
54      ADCSRA |= Copy_u8Prescaler; // 0b00000110
55
56  }
57
58  /*
59  ***** Reading the sensor data *****
60  */
61  u16 ADC_u16GetDigitalValue(u8 Copy_u8Channel){
62
63      /*
64      ***** select Channel *****
65      */
66      ADMUX &= 0b11100000;
67      ADMUX |= Copy_u8Channel;
68
69      /*
70      ***** Start conversion *****
71      */
72      SET_BIT(ADCSRA, ADSC);
73
74      /*
75      ***** Wait flag = 1 *****
76      */
77      while(GET_BIT(ADCSRA, ADIF) == 0);
78
79      /*
80      ***** clear flag *****
81      */
82      CLR_BIT(ADCSRA, ADIF);
83
84      /*
85      ***** Read ADC value *****
86      */
87      return ADC;
88  }
89
90
91
92  /*
93  ***** Disabling ADC *****
94  */
95  void ADC_voidDisable(void){
96
97      /* Disable ADC */
98      CLR_BIT(ADCSRA, ADEN);
99  }
100
```

Figure 4-27: ADC.c 2

4.3.5 SPI

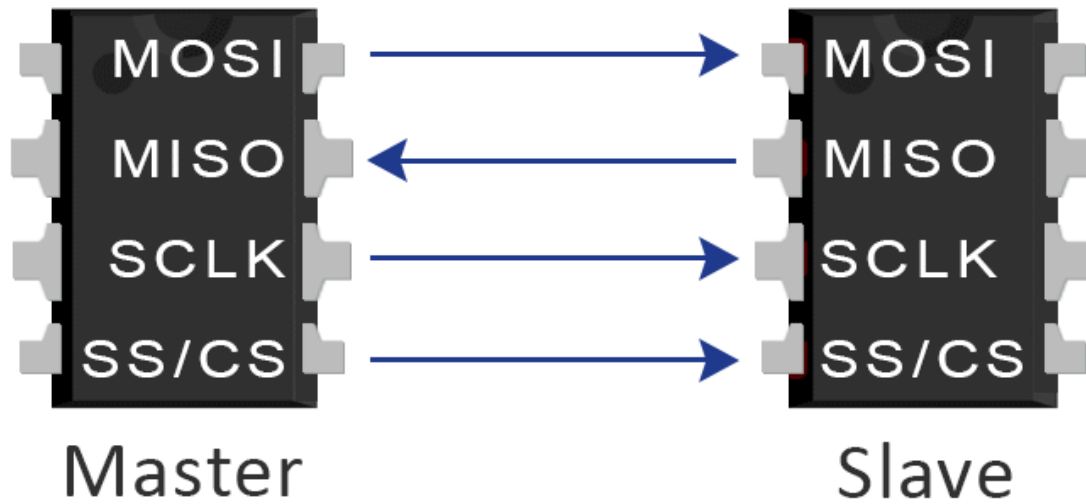


Figure 4-28: SPI

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

```

1  /*
2   * SPI.h
3   *
4   * Serial Peripheral Interface
5   */
6
7  /*
8   ***** Head Guard *****
9   */
10 #ifndef SPI_H
11 #define SPI_H
12
13 /*
14 ***** Selecting SPI Mode *****
15 */
16 #define SPI_MODE                SPI_SLAVE_MODE
17 /* select one option:          SPI_MASTER_MODE          SPI_SLAVE_MODE */
18
19
20 /*
21 ***** APIs *****
22 */
23 void SPI_voidInit                (void);
24 u8  SPI_u8ReceiveData            (void);
25 void SPI_voidMasterSendData      (u8 Copy_u8Data);
26 void SPI_voidSalveSendData       (u8 Copy_u8Data);
27 u8  SPI_u8ReceiveTrans           (u8 Copy_u8Data);
28
29
30 #endif /* MCAL_SPI_SPI_H_ */

```

Figure 4-29: SPI.h

SPI.c

```

1  /*
2   * SPI.c
3   *
4   * Serial Peripheral Interface
5   */
6
7  /*
8   ***** The Includes *****
9   */
10 #include "../LIB/BIT_Math.h"
11 #include "../LIB/STD_Types.h"
12 #include <avr/io.h>
13 /*
14 ***** Defining Communication Modes *****
15 */
16 #define SPI_MASTER_MODE      1
17 #define SPI_SLAVE_MODE      2
18
19
20 #include "SPI.h"
21
22
23 /*
24 ***** Initializing SPI *****
25 */
26 void SPI_voidInit  (void){
27
28     /*
29     ***** Setting SPI Master *****
30     */
31     #if SPI_MODE == SPI_MASTER_MODE
32         /*
33         ***** 1- Select order ==> MSB *****
34         */
35         CLR_BIT(SPCR , DORD);
36         /*
37         ***** 2- select Master mode *****
38         */
39         SET_BIT(SPCR, MSTR);
40         /*3- Clock Polarity & Phase ==> Rising_Falling ... Setup_Sample */
41         CLR_BIT(SPCR , CPOL);
42         SET_BIT(SPCR, CPHA);
43         /*
44         ***** 4 -Set clock rate fck/16 *****
45         */
46         SET_BIT(SPCR, SPR0);
47         CLR_BIT(SPCR, SPR1);
48         CLR_BIT(SPCR, SPI2X);

```

Figure 4-30: SPI.c 1

Implementation Details

```
50  /*
51  ***** Setting SPI Slave *****
52  */
53  #elif SPI_MODE == SPI_SLAVE_MODE
54
55  /*
56  ***** 1- Select order ==> MSB *****
57  */
58  CLR_BIT(SPCR, DORD);
59  /*
60  ***** 2- select Slave mode *****
61  */
62  CLR_BIT(SPCR, MSTR);
63  /*3- Clock Polarity & Phase ==> Rising_Falling ... Setup_Sample */
64  CLR_BIT(SPCR, CPOL);
65  SET_BIT(SPCR, CPHA);
66
67  #endif
68
69  /*
70  ***** Enabling SPI *****
71  */
72  SET_BIT(SPCR, SPE);
73
74  }
75
76  /*
77  ***** Receiving data *****
78  */
79  u8 SPI_u8ReceiveData (void){
80
81  /*
82  ***** Wait for reception complete *****
83  */
84  while(GET_BIT(SPSR, SPIF) == 0);
85  /*
86  ***** Return data register *****
87  */
88  return SPDR;
89  }
90
```

Figure 4-31: SPI.c 2

Implementation Details

```
91  /*
92  ***** Sending data as Master *****
93  */
94  void SPI_voidMasterSendData (u8 Copy_u8Data){
95
96      /*
97      ***** Start transmission *****
98      */
99      SPDR = Copy_u8Data;
100     /*
101     ***** Wait for transmission complete *****
102     */
103     while(GET_BIT(SPSR, SPIF) == 0);
104 }
105
106 /*
107 ***** Sending data as Slave *****
108 */
109 void SPI_voidSalveSendData (u8 Copy_u8Data){
110     SPDR = Copy_u8Data;
111 }
112
113
114 /*
115 ***** Receiving data *****
116 */
117 u8 SPI_u8ReceiveTrans (u8 Copy_u8Data){
118
119     /*
120     ***** Send data *****
121     */
122     SPDR = Copy_u8Data;
123     /*
124     ***** Wait for transmission complete *****
125     */
126     while(GET_BIT(SPSR, SPIF) == 0);
127     /*
128     ***** Return data register *****
129     */
130     return SPDR;
131 }
```

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.

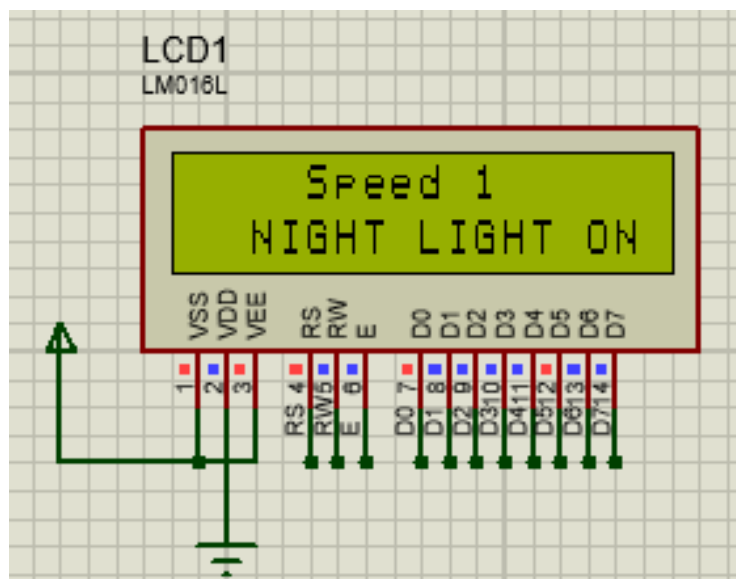


Figure 4-34: LCD running in simulation

LCD.h

```

1  #ifndef LCD_H
2  #define LCD_H
3
4  /***** Cfg *****/
5  #define LCD_DATA_PORT      DIO_PORTC
6  #define LCD_CONTROL_PORT  DIO_PORTD
7  #define RS                 DIO_PIN0
8  #define RW                 DIO_PIN1
9  #define EN                 DIO_PIN2
10 /*****/
11 #define FUNCTION_SET       0b00111000
12 #define DISPLAY_ON_OFF     0b00001111
13 #define CLEAR               0b00000001
14 /*****/
15
16 #define LCD_LINE1          1
17 #define LCD_LINE2          2
18
19 /***** APIs *****/
20
21 void LCD_voidInit          (void);
22 void LCD_voidSendCommand   (u8 Copy_u8Command);
23 void LCD_voidSendChar      (u8 Copy_u8Char);
24 void LCD_voidSendString    (u8 *Copy_u8String);
25 void LCD_voidSetLocation   (u8 Copy_u8LineNum, u8 Copy_u8CharNum);
26 void LCD_voidSendNumber    (u32 Copy_u32Number);
27 void LCD_voidDrawSpecialChar(u8 Copy_u8CharIndex, u8 *Copy_u8SpecialChar);
28 void LCD_voidSendSpecialChar(u8 Copy_u8CharIndex);
29
30 #endif

```

Figure 4-35: LCD.h

LCD.c

```

1  /*
2  * ***** The includes *****
3  */
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7  #include "../LIB/STD_Types.h"
8  #include "../LIB/BIT_Math.h"
9  #include "../MCAL/DIO/DIO.h"
10 #include "LCD.h"
11
12 /*
13 * ***** Initializing LCD *****
14 */
15 void LCD_voidInit(void){
16     /*
17     * ***** setting Data port as output *****
18     */
19     DIO_voidSetPortDirection(LCD_DATA_PORT, 0xFF);
20     /*
21     * ***** Setting RS, RW and EN pins out output *****
22     */
23     DIO_voidSetPinDirection(LCD_CONTROL_PORT, RS, DIO_OUTPUT);
24     DIO_voidSetPinDirection(LCD_CONTROL_PORT, RW, DIO_OUTPUT);
25     DIO_voidSetPinDirection(LCD_CONTROL_PORT, EN, DIO_OUTPUT);
26
27     _delay_ms(35);
28     LCD_voidSendCommand(FUNCTION_SET);
29     _delay_us(50);
30     LCD_voidSendCommand(DISPLAY_ON_OFF);
31     _delay_us(50);
32     LCD_voidSendCommand(CLEAR);
33     _delay_ms(2);
34 }
35
36 /*
37 * ***** sequence for preparing LCD to receive commands *****
38 */
39 void LCD_voidSendCommand(u8 Copy_u8Command){
40
41     DIO_voidSetPinValue(LCD_CONTROL_PORT, RS, DIO_LOW);
42     DIO_voidSetPinValue(LCD_CONTROL_PORT, RW, DIO_LOW);
43     DIO_voidSetPortValue(LCD_DATA_PORT, Copy_u8Command);
44     DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_HIGH);
45     _delay_us(1);
46     DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_LOW);
47 }
--

```

Figure 4-36: LCD.c 1

Implementation Details

```
49 /*
50 * ***** Sequence for sending a single English character to LCD *****
51 */
52 void LCD_voidSendChar(u8 Copy_u8Char){
53
54     DIO_voidSetPinValue(LCD_CONTROL_PORT, RS, DIO_HIGH);
55     DIO_voidSetPinValue(LCD_CONTROL_PORT, RW, DIO_LOW);
56     DIO_voidSetPortValue(LCD_DATA_PORT, Copy_u8Char);
57     DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_HIGH);
58     _delay_us(1);
59     DIO_voidSetPinValue(LCD_CONTROL_PORT, EN, DIO_LOW);
60
61 }
62
63 /*
64 * ***** Sequence for sending English characters (whole string) to LCD *****
65 */
66 void LCD_voidSendString (u8 *Copy_u8String){
67
68     u8 Local_U8Counter ;
69     for (Local_U8Counter = 0 ; Copy_u8String[Local_U8Counter] != '\0' ; Local_U8Counter++){
70         LCD_voidSendChar(Copy_u8String[Local_U8Counter]);
71     }
72
73 }
74
75 /*
76 * ***** Sequence for selecting the position of display on LCD *****
77 */
78 void LCD_voidSetLocation(u8 Copy_u8LineNum, u8 Copy_u8CharNum){
79
80     switch(Copy_u8LineNum)
81     {
82     case LCD_LINE1 : LCD_voidSendCommand(0x80 + Copy_u8CharNum); break;
83
84     case LCD_LINE2 : LCD_voidSendCommand(0xC0 + Copy_u8CharNum); break;
85
86     }
87
88 }
89
```

Figure 4-37: LCD.c 2

Implementation Details

```
89
90 /*
91 * ***** Sequence for sending numbers (integers) to LCD *****
92 */
93 void LCD_voidSendNumber (u32 Copy_u32Number){
94     u8 Local_u8ASingleNum[11], Local_u8Count = 9;
95
96     if (Copy_u32Number == 0) { LCD_voidSendChar('0');}
97     else{
98         Local_u8ASingleNum[10] = '\0' ;
99         while (Copy_u32Number != 0)
100         {
101             Local_u8ASingleNum[Local_u8Count] = ((Copy_u32Number % 10) + '0') ;
102             Copy_u32Number /= 10 ;
103             Local_u8Count--;
104         }
105         /* send address of the first number in my array till the '\0' */
106         LCD_voidSendString(Local_u8ASingleNum + Local_u8Count + 1);
107     }
108 }
109
110
111 /*
112 * ***** Sequence for defining a special character to LCD *****
113 */
114 void LCD_voidDrawSpecialChar(u8 Copy_u8CharIndex, u8 *Copy_u8SpecialChar){
115
116     LCD_voidSendCommand(0b01000000+(Copy_u8CharIndex * 8));
117     _delay_us(40);
118     u8 LCD_U8Counter ;
119     for (LCD_U8Counter=0 ; LCD_U8Counter<8 ; LCD_U8Counter++)
120     {
121         LCD_voidSendChar(Copy_u8SpecialChar[LCD_U8Counter]);
122     }
123
124
125 }
126
127 /*
128 * ***** Sequence for sending a special character to LCD *****
129 */
130 void LCD_voidSendSpecialChar(u8 Copy_u8CharIndex){
131
132     LCD_voidSendChar(Copy_u8CharIndex);
133 }
134
```

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.

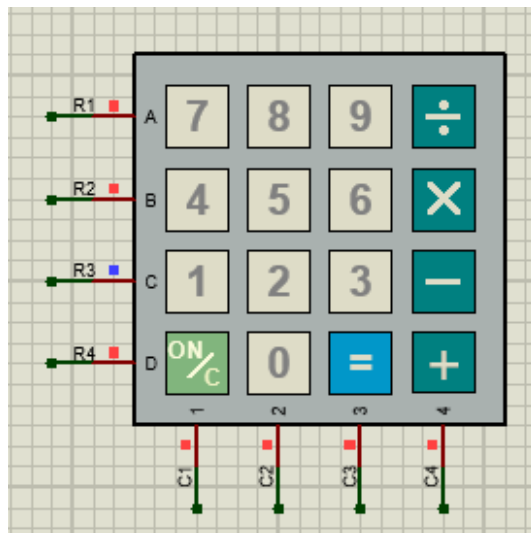


Figure 4-40: KEYPAD in simulation

KEYPAD.h

```

1  /*
2   * KEYPAD.h
3   */
4  #ifndef KEYPAD_H_
5  #define KEYPAD_H_
6
7  /*
8   ***** Configuration *****
9   */
10
11 /*
12 ***** Select kpd port *****
13 */
14 #define KPD_PORT      DIO_PORTA
15
16 /*
17 ***** Row pins ( Output pins) *****
18 */
19 #define KPD_R1_PIN    DIO_PIN0
20 #define KPD_R2_PIN    DIO_PIN1
21 #define KPD_R3_PIN    DIO_PIN2
22 #define KPD_R4_PIN    DIO_PIN3
23
24 /*
25 ***** Columns pins ( Input pins) *****
26 */
27 #define KPD_C1_PIN    DIO_PIN4
28 #define KPD_C2_PIN    DIO_PIN5
29 #define KPD_C3_PIN    DIO_PIN6
30 #define KPD_C4_PIN    DIO_PIN7
31
32 /*
33 ***** Configuring Keys layout *****
34 */
35 #define KPD_KEYS      {{ '7', '8', '9', '/' }, \
36                        { '4', '5', '6', '*' }, \
37                        { '1', '2', '3', '-' }, \
38                        { 'C', '0', '=', '+' }}
39
40 /*
41 ***** Defining keyPad specific macro *****
42 */
43 #define KPD_CHECK_BUTTON_PRESSED_OR_NOT    0xFF
44
45 /*
46 ***** APIs *****
47 */
48
49 void KPD_voidInit(void);
50 u8 KPD_u8GetPressedKey(void);
51
52 #endif /* HAL KEYPAD KEYPAD H */

```

Figure 4-41: KEYPAD.h

KEYPAD.c

```

1  /*
2   * KEYPAD.c
3   */
4
5  /*
6   * ***** The Includes *****
7   */
8  #include <avr/io.h>
9  #include <util/delay.h>
10 #include "../LIB/STD_Types.h"
11 #include "../LIB/BIT_Math.h"
12 #include "../MCAL/DIO/DIO.h"
13 #include "KEYPAD.h"
14
15 /*
16 * ***** Mapping Keys to Pins *****
17 */
18 u8 KPD_Au8Keys[4][4] = KPD_KEYS;
19
20 /*
21 * ***** Rows are set to be output *****
22 */
23 u8 KPD_Au8RowPins[4] = {KPD_R1_PIN, KPD_R2_PIN, KPD_R3_PIN, KPD_R4_PIN};
24 /*
25 * ***** Columns are set to be input *****
26 */
27 u8 KPD_Au8ColPins[4] = {KPD_C1_PIN, KPD_C2_PIN, KPD_C3_PIN, KPD_C4_PIN};
28
29
30 /*
31 * ***** Initializing KeyPad *****
32 */
33 void KPD_voidInit(void){
34
35     u8 Local_u8Count;
36     for (Local_u8Count = 0; Local_u8Count < 8; ++Local_u8Count) {
37         if (Local_u8Count < 4) {
38             DIO_voidSetPinDirection(KPD_PORT, KPD_Au8RowPins[Local_u8Count], DIO_OUTPUT);
39         } else {
40             DIO_voidSetPinDirection(KPD_PORT, KPD_Au8ColPins[Local_u8Count-4], DIO_INPUT);
41         }
42     }
43     DIO_voidSetPortValue(KPD_PORT, 0xff);
44 }
45

```

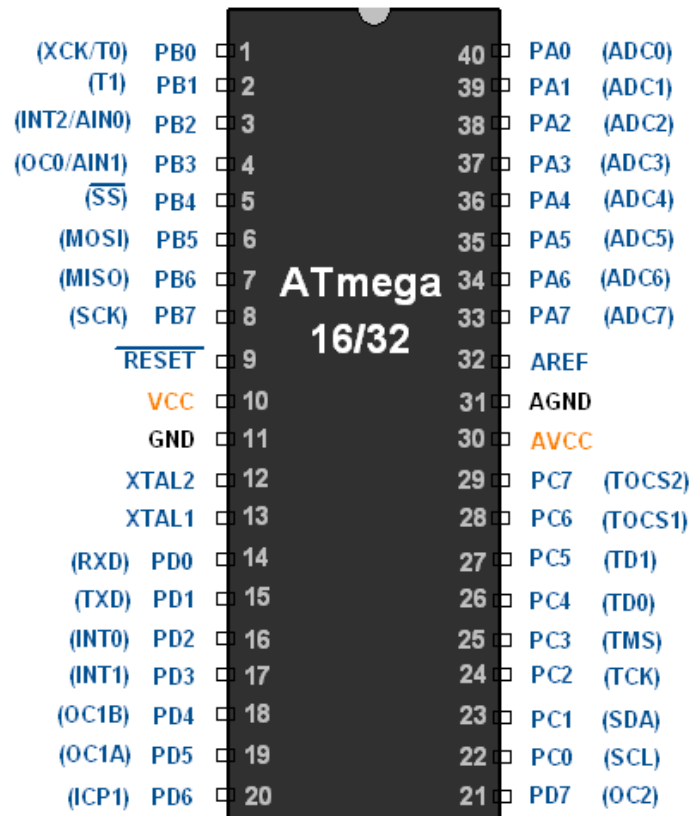
Figure 4-42: KEYPAD.c 1

Implementation Details

```
46- /*
47-  ***** Getting Key Values *****
48-  */
49- u8 KPD_u8GetPressedKey(void){
50-
51-     u8 Local_u8RowCount, Local_u8ColCount, Local_u8Pressed , Local_u8Flag =0;
52-     u8 Local_u8ReturnedVal = KPD_CHECK_BUTTON_PRESSED_OR_NOT;
53-
54-     for (Local_u8RowCount = 0; Local_u8RowCount < 4; ++Local_u8RowCount) {
55-
56-         /*
57-         ***** Applying zero voltage individually on rows *****
58-         */
59-         DIO_voidSetPinValue(KPD_PORT, KPD_Au8RowPins[Local_u8RowCount], DIO_LOW);
60-
61-         for (Local_u8ColCount = 0; Local_u8ColCount < 4; ++Local_u8ColCount) {
62-             /*
63-             ***** Checking if any button was pressed *****
64-             */
65-             Local_u8Pressed = DIO_u8GetPinValue(KPD_PORT, KPD_Au8ColPins[Local_u8ColCount]);
66-             if(Local_u8Pressed == 0){
67-                 /*
68-                 ***** Waiting for Debouncing *****
69-                 */
70-                 _delay_ms(25);
71-                 /*
72-                 ***** Checking for noise *****
73-                 */
74-                 Local_u8Pressed = DIO_u8GetPinValue(KPD_PORT, KPD_Au8ColPins[Local_u8ColCount]);
75-                 if(Local_u8Pressed == 0){
76-                     Local_u8ReturnedVal = KPD_Au8Keys[Local_u8RowCount][Local_u8ColCount];
77-
78-                     /*
79-                     ***** Handling a long press *****
80-                     */
81-                     while(Local_u8Pressed == 0){
82-                         Local_u8Pressed = DIO_u8GetPinValue(KPD_PORT, KPD_Au8ColPins[Local_u8ColCount]);
83-                     }
84-                     Local_u8Flag =1;
85-                     break;
86-                 }
87-             }
88-         }
89-         DIO_voidSetPinValue(KPD_PORT, KPD_Au8RowPins[Local_u8RowCount], DIO_HIGH);
90-         if (Local_u8Flag == 1) {break;}
91-     }
92-     return Local_u8ReturnedVal;
93- }
```

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.

Implementation Details

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.

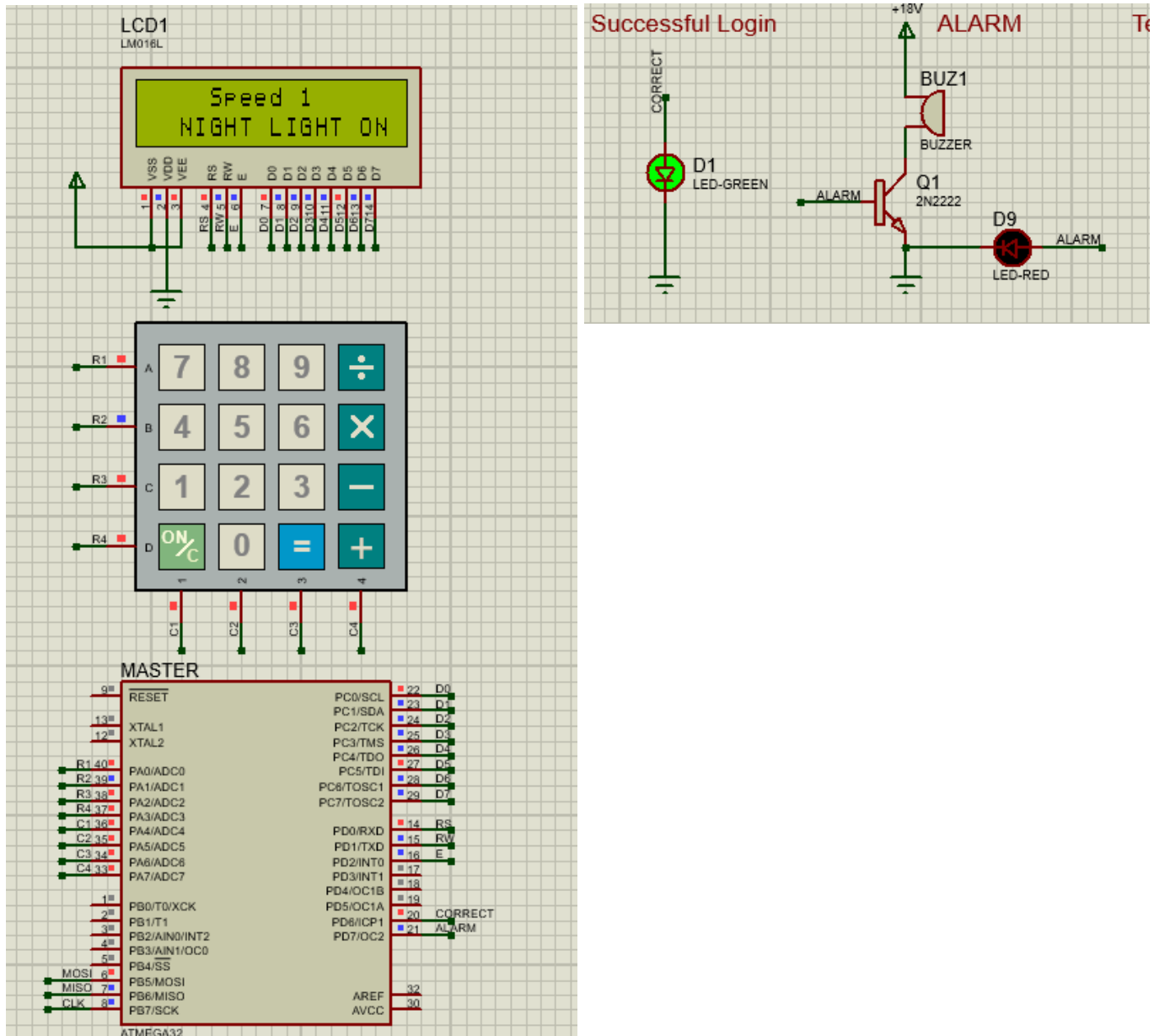


Figure 4-44: Master Microcontroller in simulation

Implementation Details

```
1  /*
2  * main.c
3  * *****
4  * *****
5  * Master MicroController portion
6  *
7  * *****
8  * *****
9  */
10
11 /*
12 * ***** THE INCLUDES *****
13 */
14 #include <avr/io.h>
15 #include <util/delay.h>
16 #include "LIB/BIT_Math.h"
17 #include "LIB/STD_Types.h"
18 #include "MCAL/DIO/DIO.h"
19 #include "HAL/KEYPAD/KEYPAD.h"
20 #include "HAL/LCD/LCD.h"
21 #include "MCAL/INT/INT.h"
22 #include <avr/interrupt.h>
23 #include "MCAL/ADC/ADC.h"
24 #include "MCAL/TIMER0/TIM0.h"
25 #include "MCAL/SPI/SPI.h"
26
27 /*
28 * ***** SPI Communication dependent flag *****
29 */
30 u8 Global_SPI_Flag = '0';
31 void SPICommunication(void);
32
33 /*
34 * Main Code base for the master controller in the home system
35 */
36 int main(void){
37
38     /*
39     * ***** Configuration for the SPI protocol *****
40     */
41     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN5, DIO_OUTPUT); // MOSI
42     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN6, DIO_INPUT); // MISO
43     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN7, DIO_OUTPUT); // CK
44     _delay_ms(200);
45     SPI_voidInit();
46     _delay_ms(200);
47 }
```

Figure 4-45 Maser Microcontroller main.c 1

Implementation Details

```
48-  /*
49-   * ***** Configuration for the timer *****
50-   */
51-   TIM0_voidInit();
52-   TIM0_voidTimerStart(TIM0_PRESCALER_8);
53-   sei();
54-   TIM0_voidOvrINTControl(TIM0_TIN_EABLE);
55-   TIM0_voidOvfCallback(SPICommunication);
56-
57-  /*
58-   * ***** Pin configuration Alarm and Home security indicator *****
59-   */
60-   DIO_voidSetPinDirection(DIO_PORTD,DIO_PIN6,DIO_OUTPUT); //correct
61-   DIO_voidSetPinDirection(DIO_PORTD,DIO_PIN7,DIO_OUTPUT); //ALARM
62-
63-  /*
64-   * ***** Initializing LCD and KeyPad *****
65-   */
66-   LCD_voidInit();
67-   KPD_voidInit();
68-
69-  /*
70-   * ***** The password is '000' *****
71-   */
72-   u8 Local_u8APass[3] ;
73-   u8 Local_u8Key;
74-   u8 local_u8flag = 0;
75-
76-  /*
77-   * ***** Super Loop *****
78-   */
79-  while(1){
80-
81-    /*
82-     * ***** Registering the keys pressed *****
83-     */
84-    for(u8 i =0; i<3;){
85-      Local_u8Key = KPD_u8GetPressedKey();
86-      if( Local_u8Key != KPD_CHECK_BUTTON_PRESSED_OR_NOT ){
87-        if(Local_u8Key == 'C'){
88-          LCD_voidSendCommand(CLEAR);
89-          _delay_ms(20);
90-          DIO_voidSetPinValue(DIO_PORTD,DIO_PIN6,DIO_LOW);
91-          DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_LOW);
92-          local_u8flag =0;
93-          i =0;
94-          continue;
95-        }
```

Figure 4-46: Maser Microcontroller main.c 2

Implementation Details

```
96      /*
97      * ***** Showing the pressed key on LCD *****
98      */
99      Local_u8APass[i] = Local_u8Key;
100      LCD_voidSetLocation(LCD_LINE1,i);
101      LCD_voidSendChar(Local_u8Key);
102      i++;
103  }
104  }
105  /*
106  * ***** Checking if the password is correct *****
107  */
108  if((Local_u8APass[0] == '0')&&(Local_u8APass[1] == '0')&&(Local_u8APass[2] == '0')){
109
110      LCD_voidSendCommand(CLEAR);
111      _delay_ms(20);
112      LCD_voidSetLocation(LCD_LINE2,2);
113      LCD_voidSendString("correct!");
114      DIO_voidSetPinValue(DIO_PORTD,DIO_PIN6,DIO_HIGH);
115      DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_LOW);
116      local_u8flag = 0;
117      Global_SPI_Flag = 'c';
118  }
119  /*
120  * ***** Allowing for three tries if entered incorrectly *****
121  */
122  else{
123      if(local_u8flag<=2){
124          LCD_voidSendCommand(CLEAR);
125          _delay_ms(20);
126          LCD_voidSetLocation(LCD_LINE2,0);
127          LCD_voidSendString("please try again!");
128          Global_SPI_Flag = 'p';
129          local_u8flag ++;
130      }
131  }
132  /*
133  * ***** Calling Authority And setting Alarm off if password is
134  * ***** entered wrongly three times *****
135  */
136  if(local_u8flag == 3){
137      LCD_voidSendCommand(CLEAR);
138      _delay_ms(20);
139      LCD_voidSetLocation(LCD_LINE2,2);
140      LCD_voidSendString("wrong!");
141      Global_SPI_Flag = 'w';
142      DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_HIGH); //ALARM ON
143      DIO_voidSetPinValue(DIO_PORTD,DIO_PIN6,DIO_LOW); //GREEN OFF
144  }
```

Figure 4-47: Maser Microcontroller main.c 3

Implementation Details

```
145     }
146     return 0;
147 }
148 /*
149  * ***** The SPI communication protocol code portion *****
150  * ***** to send and receive with the slave MicroController *****
151  */
152 void SPICommunication(void){
153     static u16 Local_u16Count =0;
154     /*
155      * ***** Setting the right time to have SPI communicate *****
156      */
157     if (Local_u16Count == 4000){
158         SPI_voidMasterSendData(Global_SPI_Flag);
159         Local_u16Count = 0;
160
161         u8 Local_u8Data = SPI_u8ReceiveData();
162         switch(Local_u8Data){
163             /*
164              * ***** Calling Authority *****
165              * ***** In case of: 1- Wrong password three times *****
166              * ***** 2- Fire Alarm / temperature of the house *****
167              * ***** above 70 degrees *****
168              */
169             case 'A' :
170                 LCD_voidSendCommand(CLEAR);
171                 _delay_ms(2);
172                 LCD_voidSetLocation(LCD_LINE1,4);
173                 LCD_voidSendString("Calling");
174                 LCD_voidSetLocation(LCD_LINE2,4);
175                 LCD_voidSendString("Authority!");
176                 DIO_voidSetPinValue(DIO_PORTD,DIO_PIN7,DIO_HIGH); //ALARM ON
177                 _delay_ms(750);
178             break;
179             /*
180              * ***** Showing the state of:
181              * ***** 1- Time of day is before sunset
182              * ***** 2 Temperature of the house is less than
183              * ***** twenty one (21) degrees
184              */
185             case '1' :
186                 LCD_voidSendCommand(CLEAR);
187                 _delay_ms(2);
188                 LCD_voidSetLocation(LCD_LINE1,4);
189                 LCD_voidSendString("Speed 1");
190                 LCD_voidSetLocation(LCD_LINE2,2);
191                 LCD_voidSendString("NIGHT LIGHT OFF!");
192                 _delay_ms(750);
193             break;
```

Figure 4-48: Maser Microcontroller main.c 4

Implementation Details

```
194- /*
195-  * ***** Showing the state of:
196-  * ***** 1- Time of day is before sunset
197-  * ***** 2- Temperature of the house is between
198-  * ***** twenty one (21) and thirty four (34) degrees
199-  */
200- case '2' :
201-     LCD_voidSendCommand(CLEAR);
202-     _delay_ms(2);
203-     LCD_voidSetLocation(LCD_LINE1,4);
204-     LCD_voidSendString("Speed 2");
205-     LCD_voidSetLocation(LCD_LINE2,2);
206-     LCD_voidSendString("NIGHT LIGHT OFF!");
207-     _delay_ms(750);
208- break;
209- /*
210-  * ***** Showing the state of:
211-  * ***** 1- Time of day is before sunset
212-  * ***** 2- Temperature of the house is between
213-  * ***** thirty four (34) and fifty five (55) degrees
214-  */
215- case '3' :
216-     LCD_voidSendCommand(CLEAR);
217-     _delay_ms(2);
218-     LCD_voidSetLocation(LCD_LINE1,4);
219-     LCD_voidSendString("Speed 3");
220-     LCD_voidSetLocation(LCD_LINE2,2);
221-     LCD_voidSendString("NIGHT LIGHT OFF!");
222-     _delay_ms(750);
223- break;
224- /*
225-  * ***** Showing the state of:
226-  * ***** 1- Time of day is after sunset
227-  * ***** 2- Temperature of the house is less than
228-  * ***** twenty one (21) degrees
229-  */
230- case '4' :
231-     LCD_voidSendCommand(CLEAR);
232-     _delay_ms(2);
233-     LCD_voidSetLocation(LCD_LINE1,4);
234-     LCD_voidSendString("Speed 1");
235-     LCD_voidSetLocation(LCD_LINE2,2);
236-     LCD_voidSendString("NIGHT LIGHT ON!");
237-     _delay_ms(750);
238- break;
```

Figure 4-49: Maser Microcontroller main.c 5

Implementation Details

```
239- /*
240-  * ***** Showing the state of:
241-  * ***** 1- Time of day is after sunset
242-  * ***** 2- Temperature of the house is between
243-  * ***** twenty one (21) and thirty four (34) degrees
244-  */
245- case '5' :
246-     LCD_voidSendCommand(CLEAR);
247-     _delay_ms(2);
248-     LCD_voidSetLocation(LCD_LINE1,4);
249-     LCD_voidSendString("Speed 2");
250-     LCD_voidSetLocation(LCD_LINE2,2);
251-     LCD_voidSendString("NIGHT LIGHT ON!");
252-     _delay_ms(750);
253- break;
254- /*
255-  * ***** Showing the state of:
256-  * ***** 1- Time of day is after sunset
257-  * ***** 2- Temperature of the house is between
258-  * ***** thirty four (34) and fifty five (55) degrees
259-  */
260- case '6' :
261-     LCD_voidSendCommand(CLEAR);
262-     _delay_ms(2);
263-     LCD_voidSetLocation(LCD_LINE1,4);
264-     LCD_voidSendString("Speed 3");
265-     LCD_voidSetLocation(LCD_LINE2,2);
266-     LCD_voidSendString("NIGHT LIGHT ON!");
267-     _delay_ms(750);
268- break;
269- default:
270-     /*
271-     * ***** reset manually with 'C' on keypad *****
272-     */
273-     break;
274- }
275- }
276- Local_u16Count++;
277- }
278- }
279- }
```

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.

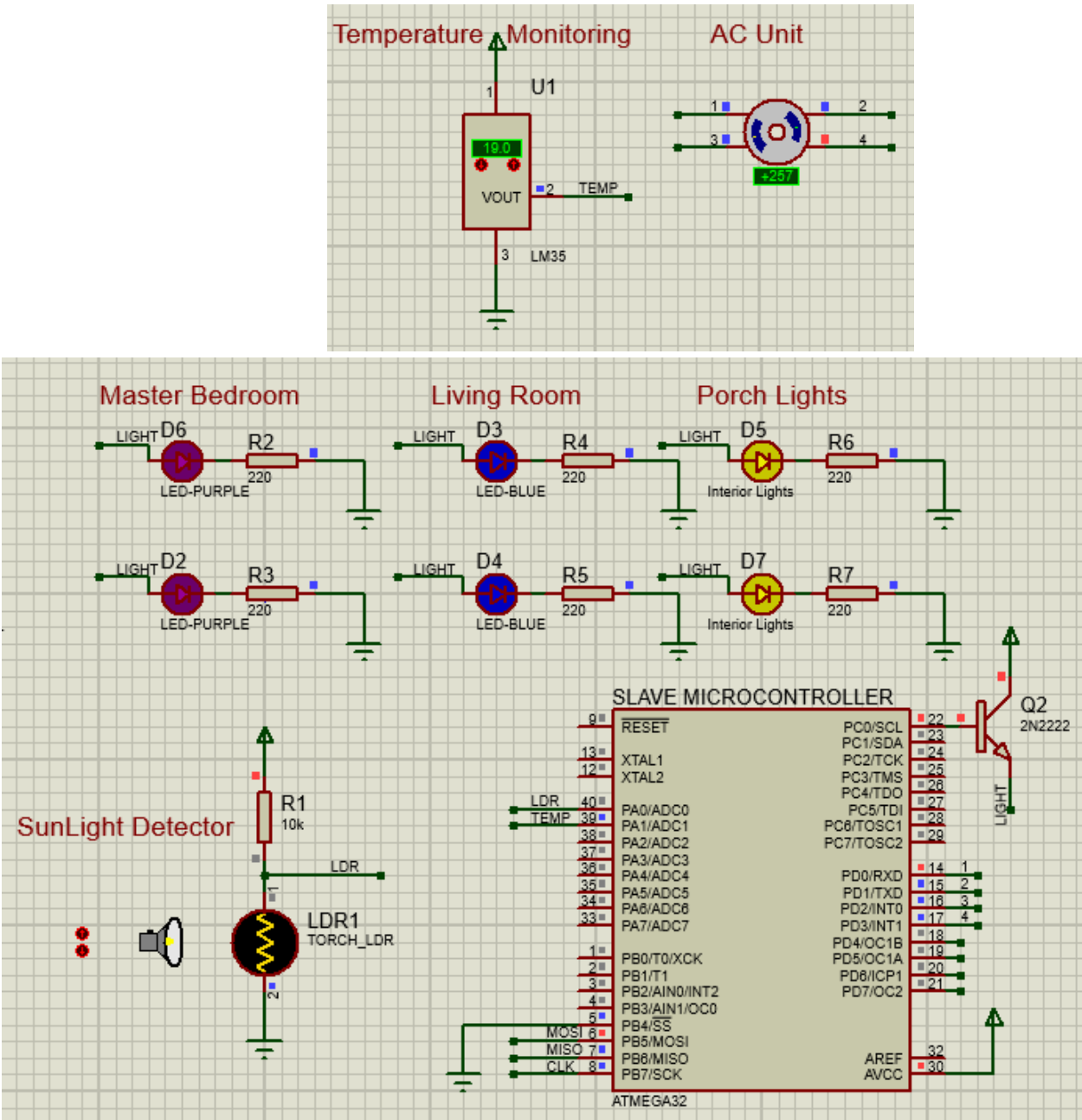


Figure 4-51: Slave Microcontroller in simulation

Implementation Details

```
1  /*
2  * main.c
3  * *****
4  * *****
5  * Slave MicroController portion
6  *
7  * *****
8  * *****
9  */
10
11 /*
12 * ***** THE INCLUDES *****
13 */
14
15 #include <avr/io.h>
16 #include <util/delay.h>
17 #include "LIB/BIT_Math.h"
18 #include "LIB/STD_Types.h"
19 #include "MCAL/DIO/DIO.h"
20 #include "HAL/KEYPAD/KEYPAD.h"
21 #include "HAL/LCD/LCD.h"
22 #include "MCAL/INT/INT.h"
23 #include <avr/interrupt.h>
24 #include "MCAL/ADC/ADC.h"
25 #include "HAL/Stepper_Motor/STPMR.h"
26 #include "MCAL/TIMER0/TIM0.h"
27 #include "MCAL/SPI/SPI.h"
28
29
30 /*
31 * Main Code base for the slave controller in the home system
32 */
33 int main(void){
34
35     /*
36     * ***** Configuration for the SPI protocol *****
37     */
38     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN4, DIO_INPUT);    // SS
39     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN5, DIO_INPUT);    // MOSI
40     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN6, DIO_OUTPUT);    // MISO
41     DIO_voidSetPinDirection(DIO_PORTB, DIO_PIN7, DIO_INPUT);    // CK
42     _delay_ms(200);
43     SPI_voidInit();
44     _delay_ms(200);
45
46     DIO_voidSetPinDirection(DIO_PORTC, DIO_PIN0, DIO_OUTPUT);
47 }
```

Figure 4-52: Slave Microcontroller main.c 1

Implementation Details

```
48  /*
49  * ***** Configuration ADC *****
50  */
51  DIO_voidSetPinDirection(DIO_PORTA, DIO_PIN0, DIO_INPUT);
52  ADC_voidInit(ADC_DIV_64);
53  STP_voidInit();
54  /*
55  * ***** Setting up variables for readings *****
56  */
57  u16 Local_u16DigitalVlaueLDR, Local_u16AnaloglValueLDR;
58  u16 Local_u16DigitalVlaueTEMP, Local_u16AnaloglValueTEMP;
59
60
61  /*
62  * ***** Super Loop *****
63  */
64  while(1){
65      /*
66      * ***** Default value for LCD of nothing *****
67      */
68      u8 Local_u8TempFlag = 'z';
69      u8 Local_u8NightLightFlag = 'z';
70      u8 Local_u8SPIDataReceived = SPI_u8ReceiveData();
71
72      /*
73      * ***** TEMP Sensor (always running ) *****
74      */
75      Local_u16DigitalVlaueTEMP = ADC_u16GetDigitalValue(ADC_Channel_1);
76      Local_u16AnaloglValueTEMP = (Local_u16DigitalVlaueTEMP * 5000UL)/1024;
77
78      /*
79      * ***** Fire Alarm (always running ) *****
80      */
81      if(Local_u16AnaloglValueTEMP >= 570){
82          Local_u8TempFlag = 'A';
83          Local_u8NightLightFlag = 'A';
84          //SPI_voidSalveSendData('1'); //1 for alarm
85          Local_u8SPIDataReceived = 'x'; //x is arbitrary constant
86      }
87  }
```

Figure 4-53: Slave Microcontroller main.c 2

Implementation Details

```
88-  /*
89-   * ***** Correct login credentials case *****
90-   */
91-   if(Local_u8SPIDataReceived == 'c'){
92-
93-
94-   /*
95-    * ***** Measuring illumination with LDR *****
96-    */
97-    Local_u16DigitalVlaueLDR = ADC_u16GetDigitalValue(ADC_Channel_0);
98-    Local_u16AnaloglValueLDR = (Local_u16DigitalVlaueLDR * 5000UL)/1024;
99-
100-  /*
101-   * ***** Controlling the day cycles lights *****
102-   */
103-   if(Local_u16AnaloglValueLDR <= 1500){
104-     DIO_voidSetPinValue(DIO_PORTC, DIO_PIN0, DIO_LOW);
105-     Local_u8NightLightFlag = '0';
106-   }
107-   else if(Local_u16AnaloglValueLDR > 1500 ){
108-     DIO_voidSetPinValue(DIO_PORTC, DIO_PIN0, DIO_HIGH);
109-     Local_u8NightLightFlag = '1';
110-   }
111-
112-  /*
113-   * ***** Controlling the AC Unit *****
114-   */
115-   if(Local_u16AnaloglValueTEMP <= 210){
116-     STP_voidRotate(STP_SPEED_1,STP_DIRECTION_CW);
117-     Local_u8TempFlag = '1';
118-   }
119-   else if((Local_u16AnaloglValueTEMP > 210)&&(Local_u16AnaloglValueTEMP <= 340)){
120-     STP_voidRotate(STP_SPEED_2,STP_DIRECTION_CW);
121-     Local_u8TempFlag = '2';
122-   }
123-   else if((Local_u16AnaloglValueTEMP > 340)&&(Local_u16AnaloglValueTEMP < 550)){
124-     STP_voidRotate(STP_SPEED_3, STP_DIRECTION_CW);
125-     Local_u8TempFlag = '3';
126-   }
127-  }
```

Figure 4-54:Slave Microcontroller main.c 3

Implementation Details

```
128- /*
129-  * ***** Wrong login credentials/ Fire Alarm case *****
130-  */
131- }else{
132- /*
133-  * ***** Cutting off power to the house *****
134-  */
135-  DIO_voidSetPinValue(DIO_PORTC, DIO_PIN0, DIO_LOW); //NIGHT_LIGHT off
136-  STP_voidOff(); //AC Unit off
137- }
138-
139-
140- /*
141-  * ***** Determining what will Show on the LCD *****
142-  */
143- switch(Local_u8NightLightFlag){
144- /*
145-  * ***** Calling Authority *****
146-  * ***** In case of: 1- Wrong password three times *****
147-  * ***** 2- Fire Alarm / temperature of the house *****
148-  * ***** above 70 degrees *****
149-  */
150-  case 'A':
151-    SPI_voidSalveSendData('A');
152-    break;
153-  case '0':
154-    switch(Local_u8TempFlag){
155- /*
156-  * ***** The state of:
157-  * ***** 1- Time of day is before sunset
158-  * ***** 2- Temperature of the house is less than
159-  * ***** twenty one (21) degrees
160-  */
161-  case '1': SPI_voidSalveSendData('1'); break;
162- /*
163-  * ***** Showing the state of:
164-  * ***** 1- Time of day is before sunset
165-  * ***** 2- Temperature of the house is between
166-  * ***** twenty one (21) and thirty four (34) degrees
167-  */
168-  case '2': SPI_voidSalveSendData('2'); break;
169- /*
170-  * ***** Showing the state of:
171-  * ***** 1- Time of day is before sunset
172-  * ***** 2- Temperature of the house is between
173-  * ***** thirty four (34) and fifty five (55) degrees
174-  */
175-  case '3': SPI_voidSalveSendData('3'); break;
176-    }
177-    break;
```

Figure 4-55: Slave Microcontroller main.c 4

Implementation Details

```
178     case '1':
179         switch(Local_u8TempFlag){
180             /*
181              * ***** Showing the state of:
182              * ***** 1- Time of day is after sunset
183              * ***** 2- Temperature of the house is less than
184              * ***** twenty one (21) degrees
185              */
186             case '1': SPI_voidSalveSendData('4'); break;
187             /*
188              * ***** Showing the state of:
189              * ***** 1- Time of day is after sunset
190              * ***** 2- Temperature of the house is between
191              * ***** twenty one (21) and thirty four (34) degrees
192              */
193
194             case '2': SPI_voidSalveSendData('5'); break;
195             /*
196              * ***** Showing the state of:
197              * ***** 1- Time of day is after sunset
198              * ***** 2- Temperature of the house is between
199              * ***** thirty four (34) and fifty five (55) degrees
200              */
201
202             case '3': SPI_voidSalveSendData('6'); break;
203         }
204     }
205     break;
206 }
207 }
208 return 0;
209 }
210
```

Figure 4-56: Slave Microcontroller main.