Design & Construction of Universal Electronic Component Tester

A Project submitted in partial fulfillment of the requirements for the Award of Degree of Bachelor of Science in Electrical and Electronic Engineering

Prepared by

Ahnaf Tajwar karim
ID: 161-33-3142

Md. Bachu Howlader
ID: 143-33-2139

Supervised by

Md. Dara Abdus Satter
Associate Professor & Associate Head
Department of EEE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
FACULTY OF ENGINEERING
DAFFODIL INTERNATIONAL UNIVERSITY

March, 2020

Certification

This is to certify that this project entitled "**Design & Construction of Universal Electronic Component Tester**" is done by the following students under my direct supervision and this work has been carried out by them in the laboratories of the Department of Electrical and Electronic Engineering under the Faculty of Engineering of Daffodil International University in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering. The presentation of the work was held on March 2020.

Signature of the Candidates

____

Name: Ahnaf Tajwar Karim

ID: 161-33-3142

____

Name: Md. Bachu Howlader

ID: 143-33-2139

This is to certify that this project entitled "**Design & Construction of Universal Electronic Component Tester**" is done by the following students under my direct supervision and this work has been carried out by them in the laboratories of the Department of Electrical and Electronic Engineering under the Faculty of Engineering of Daffodil International University in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering. The presentation of the work was held on March 2020.

Signature of the Supervisor

Md. Dara Abdus Satter
Associate Professor & Associate Head
Department of Electrical and Electronic Engineering
Daffodil International University

Dedicated to

Our Parents

CONTENTS

List of Abbreviations

CPU	Central Processing Unit
CD	Chromatic Dispersion
EMI	Immune to Electromagnetic Interference
FBG	Fiber Bragg Gratings
FWHM	Full Width at Half Maximum
GVD	Group Velocity Dispersion
LED	Light Emitting Diodes
MD	Material Dispersion
NLSE	Nonlinear Schrödinger Equation
PMD	Polarization Mode Dispersion
PUA	Piecewise Uniform Approach
RMS	Root Mean Square
SSMF	Standard Single Mode Fiber
TFBG	Tilted Fiber Bragg Gratings
UV	Ultraviolet
WD	Wave-guide Dispersion
WDM	Wavelength Division Multiplexed

List of Figures

ACKNOWLEDGEMENT

First of all, we give thanks to Allah or God. Then we would like to take this opportunity to express our appreciation and gratitude to our project and thesis supervisor Md. Dara Abdus Satter, Associate Professor & Assistant Head of Department of EEE & Deputy Proctor of Daffodil International University for being dedicated in supporting, motivating and guiding us through this project. This project can’t be done without his advice and helps. Also thank you very much for giving us opportunity to choose this project.

We also want to convey our thankfulness to Mr. Nazmus Saqib (Lecturer), of the Department of EEE for his help, support and constant encouragement.

Apart from that, we would like to thank our entire friends Gazi Md. Mahedi Hassan & Engr. Mithun Das (KUET) for sharing knowledge; information and helping us in making this project a success. Also thanks for lending us some tools and equipment.

ABSTRACT

This project presents the design and construction of an universal electronic components tester. In this tester we have used the flexible programmable features of Atmega328 microcontroller for its application. The microcontroller used as a CPU for the keypad, liquid crystal (LCD), input and out components under test via a ZIF socket. Keypad functions as a medium for alerting the microcontroller about the component to be tested. The LCD function as a platform for displaying the functionally condition of the component under and results. ZIF socket function as a platform to place component under test. In this system we have two section controlled by microcontroller. Our developed system is less complex, easy to install, cost effective, easy to operate and more convenient both in lab and industrial field as well. So, this project can use for this purpose in electronics lab and electronics servicing center & also in R&D Lab.

Chapter 1

Introduction

1.1 Introduction

This universal component tester of testing is able to detect whether electronic components is operational. If you are new to start to build electronic circuits then the important thing to do is to get familiar with few basic logic gates ICs and basic electronic components. Without understanding these basic electronic components i.e. their values, ratings, purpose etc. your circuit design might not function as expected.

1.2 Objectives

The aim of the project is to design a component tester and testing electronic components. The project would mainly focus on the testing of electronic components. Therefore, we want to develop microcontroller based equipment that can able to detect problems very quickly and easily with low cost and reliable circuit. The general objectives of this system are summarized below:

- To develop electronic component tester.
- To build this project can be used for this purpose without multimeter.
- To useful in electronics laboratory and servicing center.

1.3 Organization of the Project Report

In this book, you can get a brief overview of few of the most common basic electronic components and basic logic gate. For more information about a particular component and basic logic gate, you can check out the link associated with individual component. In this report, chapter one covers introduction, motivation and objective.Chapter two represents theory, description of circuit elements, such as power supply, microcontroller, Voltage regulator, LCD, PCB making and other parts. Chapter four describes Basic logic gate ICs.Describes the working principle of the project Result, discussion and cost estimation has been discussed in chapter five. Finally we have concluded this work by writing chapter six that includes conclusion and future scope.

Chapter 2

Theory of the Project

2.1 Introduction

In this chapter we introducedthose entire components which were used in our project to complete project properly.The whole system is controlled by an AVR microcontroller. Wehave used Atmega328P microcontroller for interfacing, data processing & controlling the system. We can control a good number of travelable devices by this method.

2.2 Theory

We have controlled the whole circuit by the Microcontroller ATmega328.It is a free bandwidth in most of the countries. Now a day's ready-made LCD display module is very cheap. The system will displaythe conditions ofthe device on the LCD display.

There are many ways to classify different types of electronic components but the most common way is to classify them in to three types:

1) Active Electronic Components,

2) Passive Electronic Components and

3) Electromechanical Components.

It is a 28 pin device in which 23 pins can be used for input and output purpose.

2.3 Flow Chart

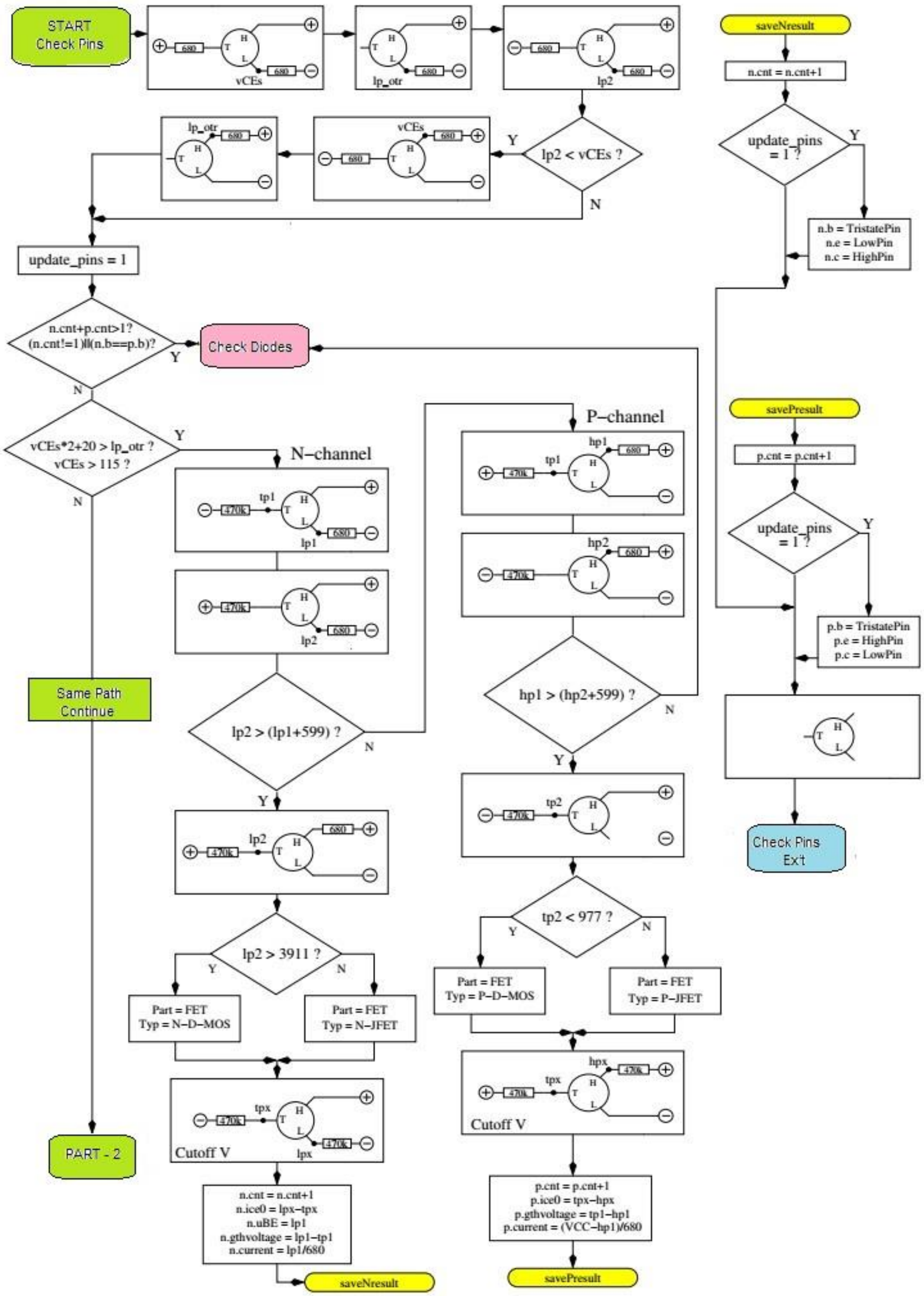


Fig 2.0: Flow Chart

2.4 Description of the Circuit Elements

Apparatus used for project:

(i) Atmega 328P Microcontroller
(ii) 16× 2 LCD Display
(iii) LM7805 Voltage Regulator
(iv) Power Supply Unit (12V Step down Transformer with Bridge rectifier).
(v) Power Switch
(vi) Electrolytic Capacitor
(vii) Ceramic Capacitor
(viii) Pin Headers & Connectors
(ix) IC Socket
(x) LED
(xi) Resistor
(xii) Push button switch
(xiii) Diode
(xiv) DC socket
(xv) ZIF socket

2.4.1 Atmega328P Microcontroller

(a) Introduction

Atmega328 is an 8-bit AVR Microcontroller. It is 28 Pins AVR Microcontroller and manufactured by Microchip follows RISC Architecture and has a flash type program memory of 32KB.

- Atmega328P Microcontroller has an Electrically Erasable Programmable Read-Only Memory memory of 1KB and its SRAM memory is of 2KB.
- Atmega328P Microcontroller has 8 Pin for Analog to digital converter operations, which all combines to form PortA (PA0 – PA7).

- It has 3 builtin Timers of them two of them are 8 Bit timers while the third one is 16-Bit Timer.
- You must have heard of Arduino UNO, which is based on atmega328 Microcontroller.
- It's UNO's heart.
- It's operates ranging from 3.3-5.5V but normally we use 5V as a standard.
- Its excellent features include the cost efficiency, programming lock for security purposes, and real timer counter with separate oscillator.

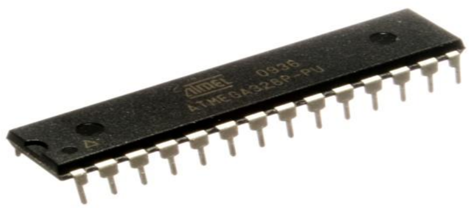


Fig 2.1: Atmega328P Microcontroller

(b) Features

- To perform any task we can select a device on the basis of its features. its the main features of an AVR Microcontroller ATmega328 are shown in the table given in the figure below.

ATmega328 Features	
Sr. No	Features
1	Non programmable data and program memory
2	High performance
3	Low power consumption
4	Fully static operation
5	On chip analog comparator
6	Advance RISC architecture
7	32KB flash memory
8	2KB SRAM

(c) Block Diagram of ATmega328P

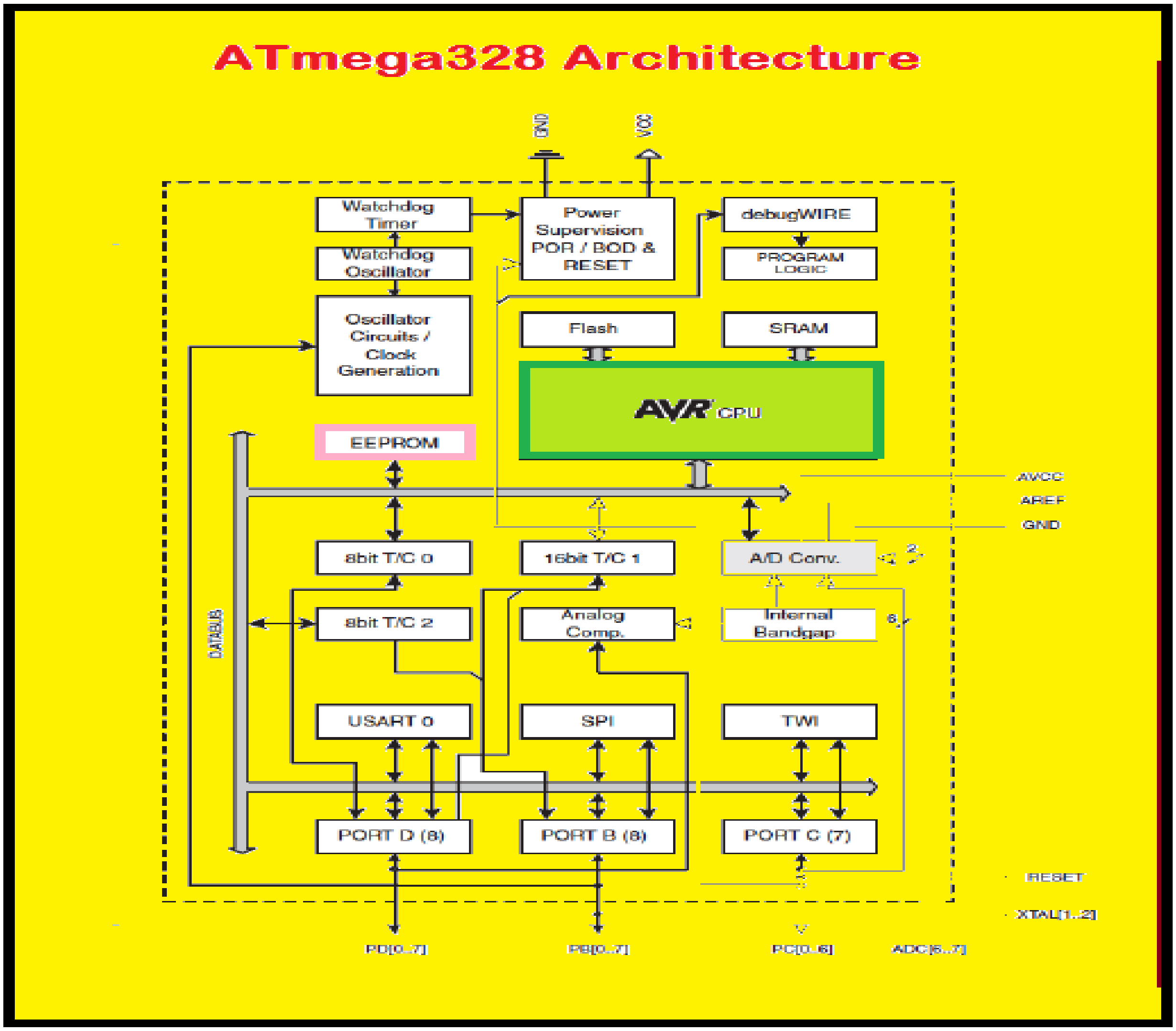


Fig 2.2: Block Diagram of ATmega 328P

Atmega32 avr microcontroller (8 bit) introduction to.Avr 11v internal ADC reference over voltage electrical. Atmega328 8 bit avr microcontroller’s microchip. This Introduction To Atmega328 The Engineering Projects image has 20 dominated colors, which include white, tomb blue, basalt grey, cerebral grey, royal navy blue, niblet green, smoked purple, bluebell, black, wisp, rebecca purple, tatzelwurm green, red blood, woodgrain, duck tail, lizard belly, earthy khaki green, canopy, boston university red, persian red.

(d) Pin Configurations of ATmega328P

ATmega 328P has 23 I/O pins, 2 power pins, 2 ground pins, reset pin and 2 clock pins shown in Figure 2.3.

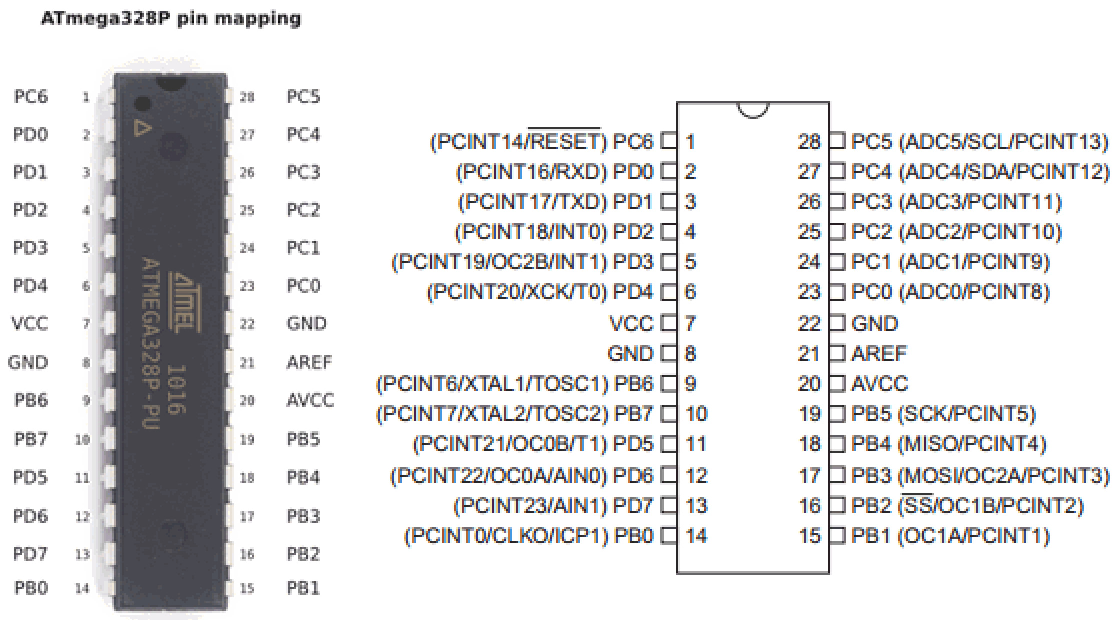


Fig 2.3: Pin Diagram of ATmega 328P

(e) Pin Descriptions

VCC is a digital voltage supply. (Pin No: 07)AVCC is a supply voltage pin for analog to digital converter. (Pin No: 20)GND denotes Ground and it has a 0V (Pin No: 08 & 22)Port A consists of the pins from PA0 to PA7. If analog to digital converter is not used, port A acts as an eight (8) bit bidirectional input/output port.Port B is consists of the pins from PB0 to PB7.Port C of the pins from PC0 to PC7. Output port C has symmetrical drive characteristics with source capability as well high sink.Port D consists of the pins from PD0 to PD7. It is also an 8 bit input/output port having an internal pull-up resistor.PC6/RESET If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Shorter pulses arenot guaranteed to generate a Reset.The various special features of Port C are elaborated in the Alternate Functions of Port Csection. Analogue REFerence is the analog reference pin for the A/D Converter.

(f) Memory Organizations

1. ATmega 328 has three types of memories e.g. EEPROM, SRAM etc.
2. The capacity of each memory is explained in detail below.

Flash Memory has 32KB capacity. It has an address of 15 bits. It is a Programmable Read Only **Memory (ROM).** It is non volatile memory.

SRAM stands for Static Random Access Memory. It is a volatile memory i.e. data will be removed after removing the power supply.

EEPROM stands for Electrically Erasable Programmable Read Only Memory and it has a long termdata.

ATmega328 and Arduino

ATmega-328 is the most micro-controller that is used while designing and it is the most important part of Arduino.The program is uploaded on the automatic voltage regulator micro-controller attached on Arduino.ATmega328 attached on Arduino is shown in the figure given below.

Fig 2.4: ATmega328 and Arduino

(g) Clock Options

The following figure illustrates the principal clock systems in the device and their distribution and all the clocks need not be active at a given time. The clock systems are described in the following sections and refers to the frequency generated from the System Clock Presale.After that all clock outputs from the AVR Clock Control Unit runs in the same frequency.

Clock Sources

The clock signal can come from an internaloscillator, an external crystal/resonator, or an external signal. Arduinos normally use an external 16MHz crystal.Clock selection table is shown in Fig 2.4.

Device Clocking Option	CKSEL3 CKSEL2 CKSEL1 CKSEL0
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved / Not used	0001

Fig 2.5: Clock Source Selection

Arduinos normally use a low power crystal oscillator and The ATmega has 2 built in oscillators, a 128 kHz RC oscillator and a calibrated RC oscillator. The external clock signal needs to be used crystal oscillator for clock source. A 16MHz Crystal is connected to XTAL1 & XTAL2 pin.

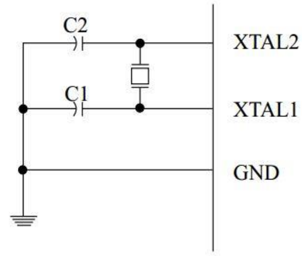


Fig 2.6: Crystal Oscillator Connection

(h) Analog to Digital Converter

The Atmel ATmega328P microcontroller used on the Arduino Uno has an A/D conversion module capable of converting an analog voltage into a 10-bit number from 0 to 1023 or an 8-bit number from 0 to 255 and the input to the module can be selected to come from any one of six inputs on the chip.

Features of ADC

- 10-bit Resolution
- 0.5 LSB Integral Non-Linearity
- ±2 LSB Absolute Accuracy
- 13µs - 260µs Conversion Time
- Temperature Sensor Input Channel
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Sleep Mode Noise Canceler

(i) General I/O Ports

General I/O Ports have true Read-Modify-Write functionality when used this ports that the direction of one port pin can be changed without unintentionally changing any other pin withthe SBI and CBI instructions.The pin driver is strongenough to drive LED displays directly. After all port have individually selectable pull-up resistors with asupply-voltage invariant resistance. The I/O port as General Digital I/O is described in next section and most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in Alternate Port Functions section in this chapter.After that Refer to the individual module sections for a full description of the alternate functions.

I/O pin configuration of Atmega 328P is shown below:

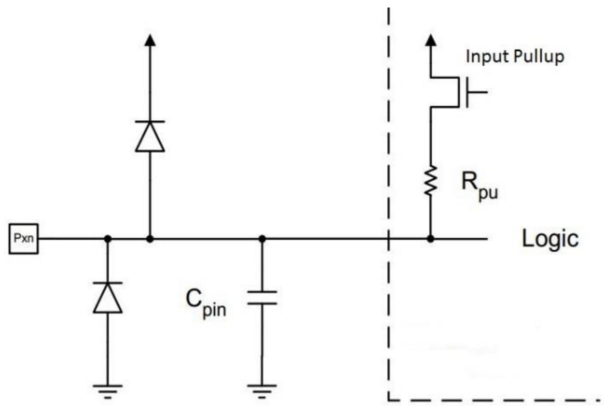


Fig 2.7: I/O Pin Configuration

(j) SPI – Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a very useful data transfer protocol for microcontrollers and it is the method used by programming devices like the usbtiny to transfer programs to AVR microcontrollers and is a way to interface with SD cards, among other things.

SPI communication connection is shown in Fig 2.7 below:

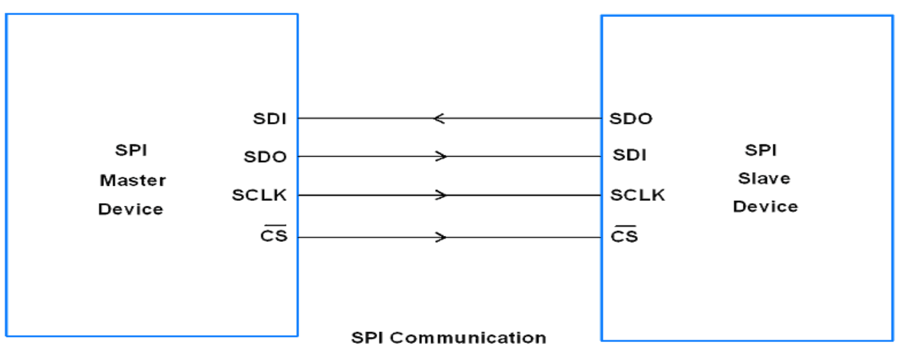


Fig 2.8: SPI Connection Diagram

Features

- Full-duplex, Three-wire Synchronous Data Transfer and Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

2.1.1 LCD Display Module

LCD means Liquid Crystal Display.It is an electronic display which is commonly used nowadays in applications such as calculators, laptops, tablets, mobile phones etc and it can display 2 lines of 16 characters and each character is displayed using 5×7 or 5×10 pixel matrix and it is displayed in 5x7 pixel matrix. This Liquid Crystal Display has two registers, that name Command and Data.

LCD Display Module is shown in Figure 2.8.

Fig 2.9: LCD Display Module

A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD. The 16×2 translates a display 16 characters per line in 2 such lines. In this LCD each character is displayed in a 5×7 pixel matrix.RS (Register select): A 16X2 LCD has two registers, namely, command and data. The register select is used to switch from one register to other. RS=0 for command register, whereas RS=1 for data register.

Command Register: The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc.Processing for commands happen in the command register.

Data Register: The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD. When we send data to LCD it goes to the data register and is processed there. When RS=1, data register is selected.

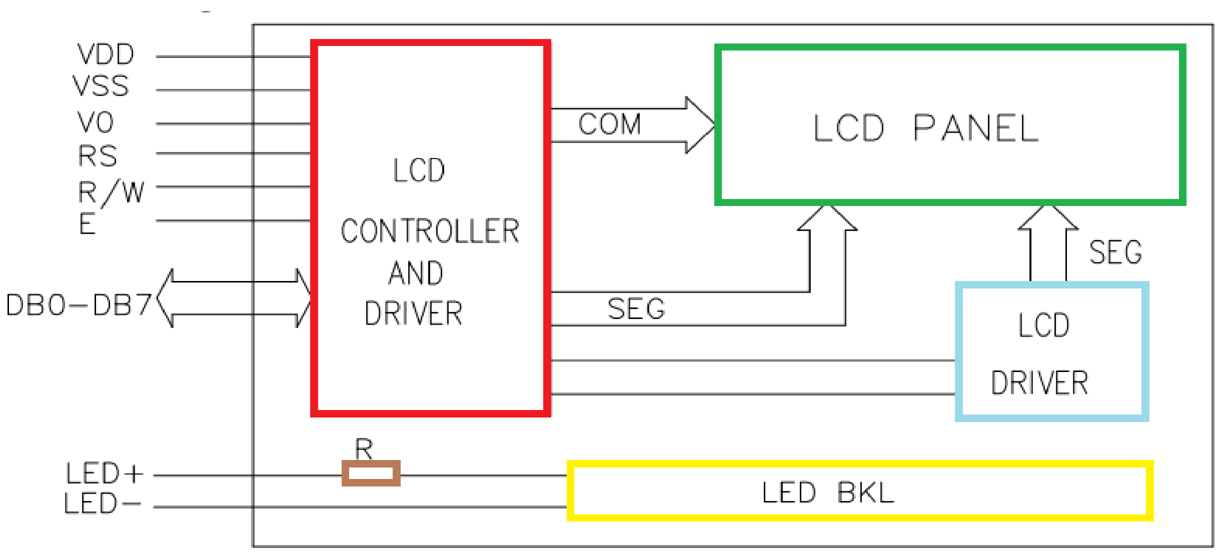


Fig 2.10: LCD Block Diagram

2.1.2 ZIF socket

ZIF means Zero Insertion Force. It is a type of CPU socket on a computer motherboard that allows for the simple replacement or upgrade of the processor and uses a ZIF socket can easily be removed by pulling a small release lever next to the processor and lifting it out and connectors are used to secure delicate ribbon cables, such as flat flex cables or flexible printed circuit cables. To disconnect the cable, use the tip of a spudger or your fingernail to flip up the small locking flap.ZIF Socket Pinout is shown in Figure 2.12 below.

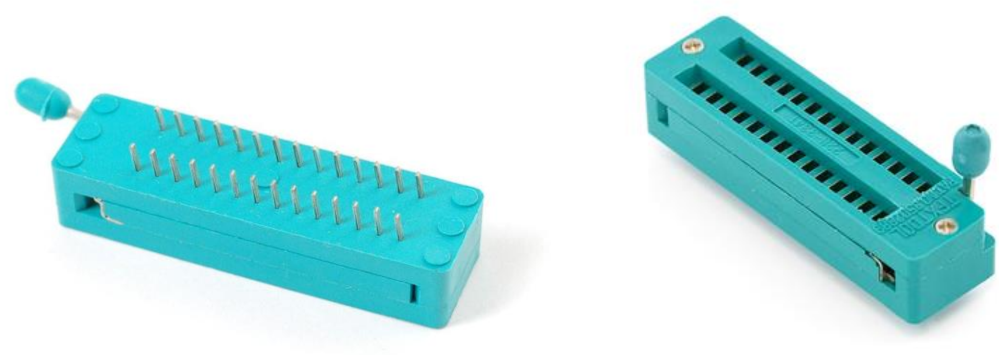


Fig 2.11: Pin Configuration of ZIF Socket.

2.1.3 Push Button Switch

A push button switch is a small sealedmechanism. When it’s on makes contact with two wires, all owing electricity to flow and after that it is off the spring retracts, contact is interrupted, and current won't flow.

Fig 2.12: Push Button Switch

It is devoted and opposed to a typical on or off switch which latches in its set position and momentary switches may be normally normally closed.

2.1.4 DPDT Mini Push Switch

Adding another pole to the SPDT creates a double-pole, double-throw (DPDT) switch. Basically two SPDT switches, which can control two separate circuits, but are always switched together by a single actuator. DPDTs have six terminals. A DPDT Mini Push Switch Pinout is shown below:

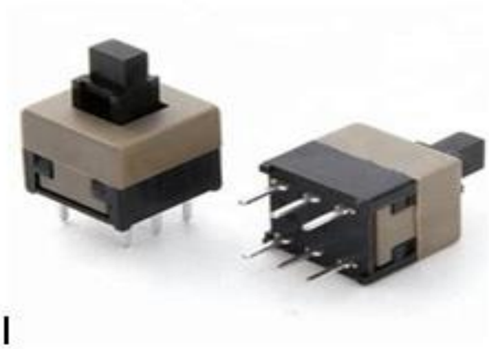


Fig 2.13: DPDT Mini Push Switch

A 6 Pin Push Switch also known as Mini DPDT Push Switch, is nothing but a combination of two push switches placed together inside one package. Unlike momentary switches which connect the wires of the switch only for a second; this switch retains its ON-OFF state till pushed later on.

2.1.5 LM 7812 Voltage Regulator

The LM 7812 Voltage regulator is a type of selfcontained fixed linear voltage regulator circuit that it's do operate at their optimal capability if the input voltage is at least 2.5 volt greater than the output voltage and the current is 1 or 1.5 Amperes more we used in this 1.25.

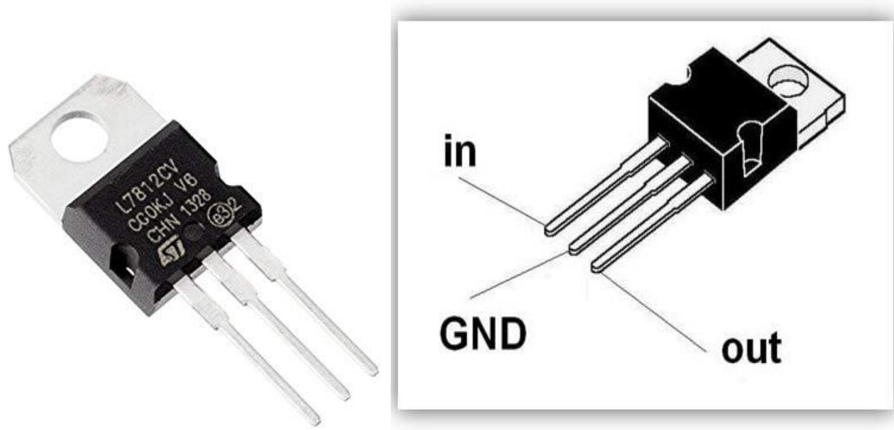


Fig 2.14: LM7812 5.0 Voltage Regulator Pin-out

A voltage regulator is used to regulate voltage level. When a steady, reliable voltage is needed, then voltage regulator is the preferred device. It generates a fixed output voltage that remains constant for any changes in an input voltage or load conditions. It acts as a buffer for protecting components from damages.It is simple to build: connect the 12V wire at the left most terminal of the IC, while looking at the inscription and with the pins down. Connect the 5V output for USB port at the right most pin. Connect the ground of your 12V supply and of your USB to the middle pin or to the heatsink.

Circuit Diagram

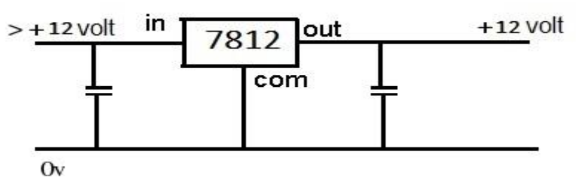
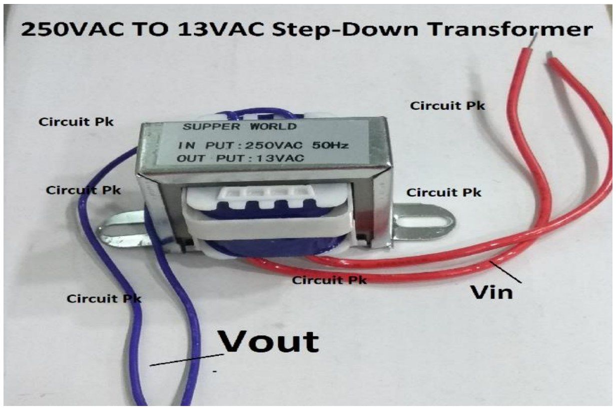


Fig 2.15: Voltage Regulator Circuit Diagram

2.1.6 220/12 V AC to 12V DC Power Supply Unit:

We have used the 220V-12V Ac step down transformer andthe Bridge Rectifier for rectifying & Filtering the 12V DC output supply. After that, we divided the 5V DC ultimate pure driect current power supply into two separate Arduino UNO set.



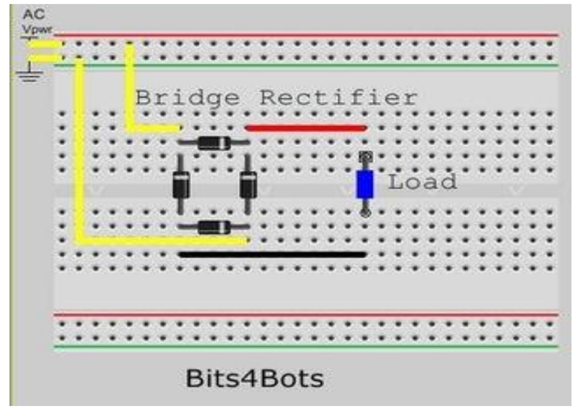


Fig 2.16: 12V DC Power Supply Unit

Circuit Diagram

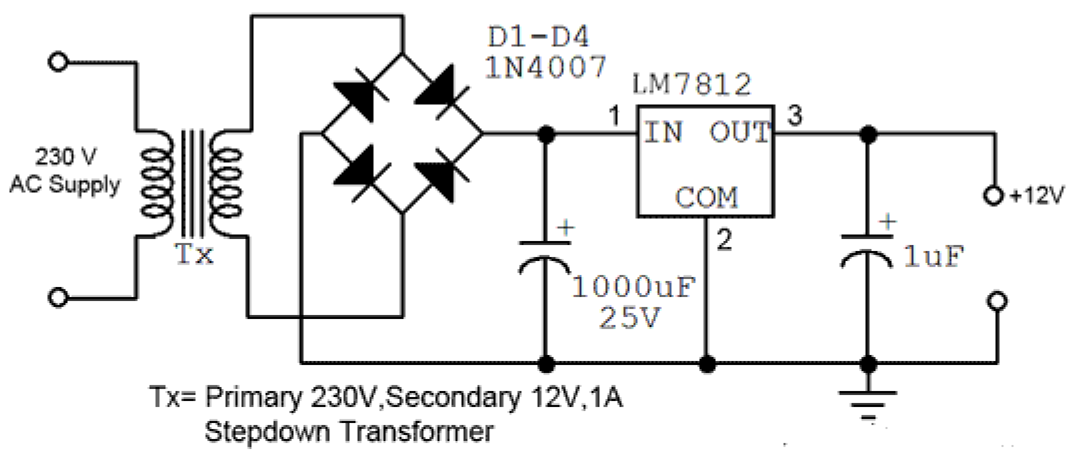


Fig 2.17: Circuit Diagram of 12V DC Power Supply Unit

Firstly, 220V AC is converted into 12V autometicecurrent by using simple step-down transformer.After that output of this transformer is given to the rectifier circuit, which will convert the ac supply into dc supply. The output of the rectifier circuit that isDC contains the ripples in the output voltage.

2.1.7 Electrolytic Capacitor

Electrolytic capacitors are madeby layering the electrolytic paper between anode and cathode foils, and then coiling the result and the process of preparing an electrode facing the etched anode foil surface is extremely difficult.Due to this process, the electrolyte essentially functions as the cathode.A second aluminum foil called “cathode foil” contacts the electrolyte and serves as the electrical connection to the negative terminal of the capacitor. [14]

Fig 2.18: Electrolytic Capacitor

2.1.8 Ceramic Capacitor

A ceramic capacitor is a fixed-value capacitor where the ceramic material acts as the dielectric. It is constructed of two or more alternating layers of ceramic and a metal layer acting as the electrodes. The composition of the ceramic material defines the electrical behavior and therefore applications. Ceramic capacitors are majorly used in the resonant circuit in transmitter stations. Class 2 high-power capacitors are used in high voltage laser power supplies, power circuit breakers, induction furnaces etc. Surface mount capacitors are often used in printed circuit boards and high-density applications.

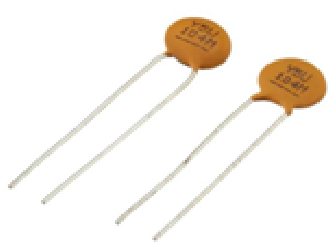


Fig 2.19: Ceramic Capacitor

Ceramic Capacitors or Disc Capacitors as they are generally called are made by coating two sides of a small porcelain or ceramic disc with silver and are then stacked together to make a capacitor. For very low capacitance values a single ceramic disc of about 3-6mm is used.

2.1.9 AC Power Cable 2 prong

AC power Cable 2 Prong is a common barrel-type power cable for AC wall supplies. These are compatible with AC wall supplies and have a 1mm diameter whole to connect to the Transformer Primary side wire connection.

Fig 2.20: AC Power Jack

2.1.10 Header Pin

Pin header is a type of electrical connector and used in electronic or instrumentation of Print Circuit Board (PCB) function as bridge between two Print Circuit Boardes which were blocked, and used to take current or signal transmission.

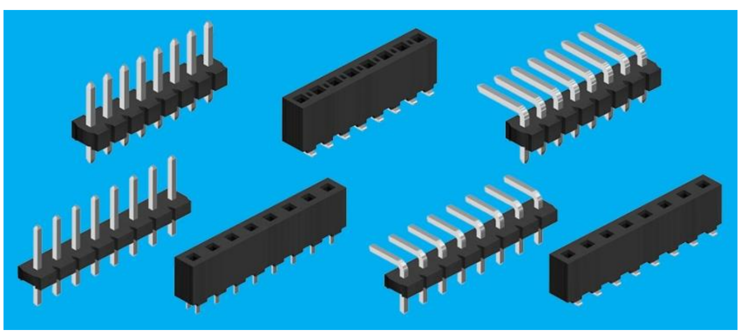


Fig 2.21: Different Types of Header pins

PCB headers are a type of electrical connector and that use to make multiple electrical connections to a PCB using one connection block or cable.

Chapter 3

Design & Fabrication

3.1 Introduction

In this chapter the project design and fabrication will be fully described. A circuit diagram has been designed. Then the project is implemented according diagrams. Here overall project description, implementation procedure and working principle will be discussed. Project flow chart is also available in this chapter.

3.2 Block Diagram

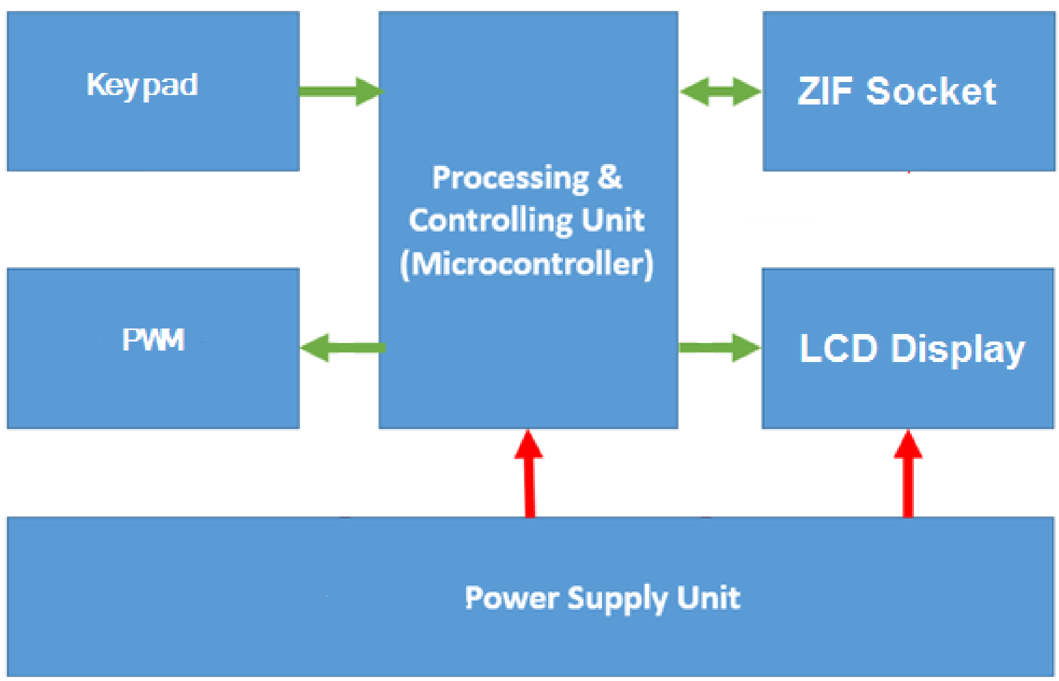


Fig 3.1: Block diagram of Component Tester Device

3.3 Designing Circuit

The circuit diagram of the project is designed by Proteus ISIS 8.1.

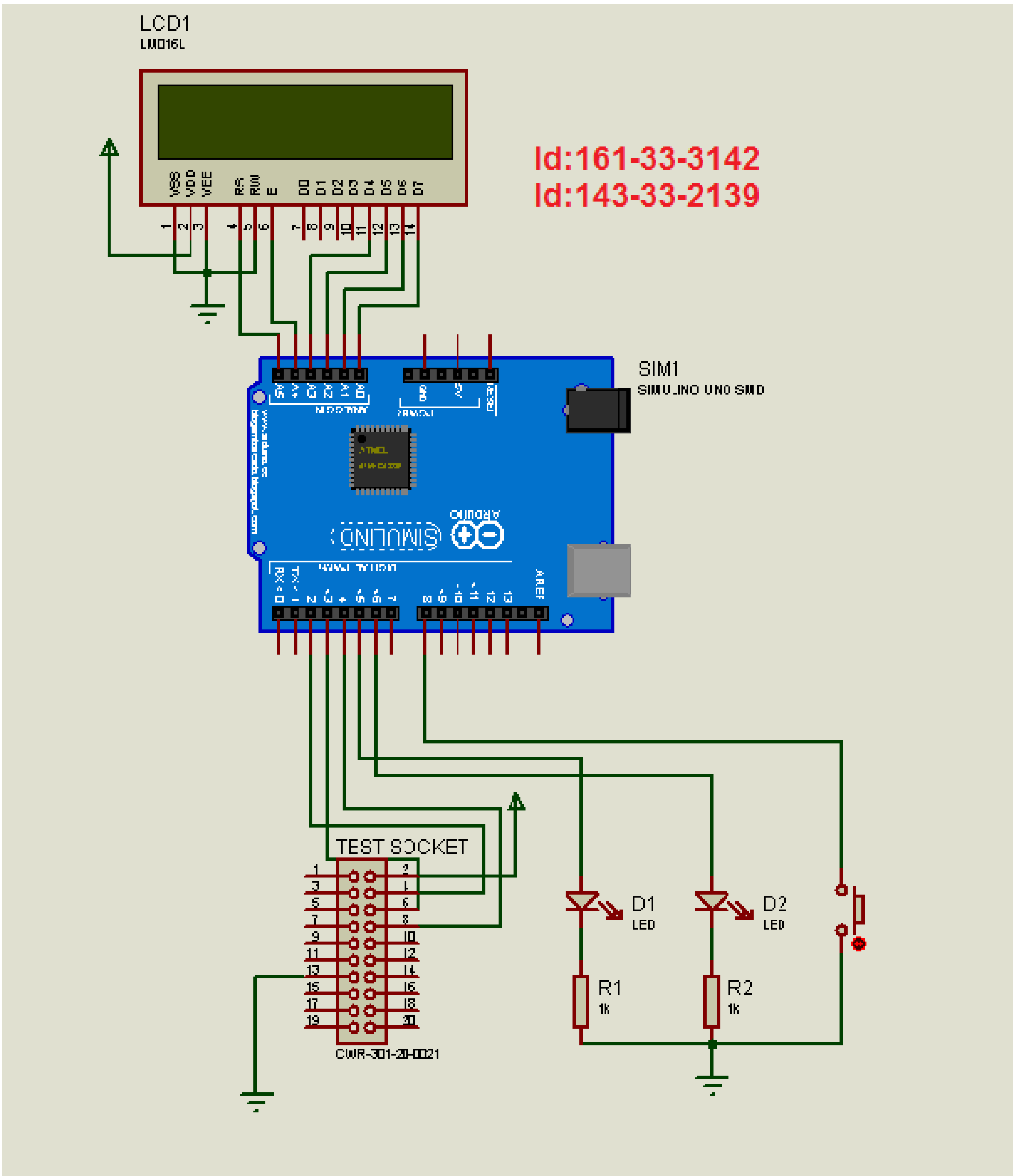


Fig 3.2: Complete Circuit diagram of Project

3.4 Circuit Description

The complete is built with microcontroller, LCD Display module, ZIF Socket, power switch, connectors & power supply. The microcontroller acts as a CPU of the system. It needs 5v to power-up. Constant 5V from power supply is connected to Vcc pin of microcontroller. For clock circuit of microcontroller a 16 MHz crystal is connected to clock pin of the microcontroller.Two 22pF disc ceramic capacitor is also connected to clock pin to filter noise. The LCD Display module is connected to microcontroller port.For powering LCD Display module with 5.0V. Push Button switch is connected to microcontrollers through a Analog pin. As a displaying 16*2 display is used. Six I/O pins are used to interface the display. The display needs 5V to power-up. Power pin is connected to 5V power rail and backlight LED pin is connected to 5V power rail. As a signal LED is connected to pin of microcontroller.

3.5 Printed Circuit Board Design

The printed circuit board is designed by Express PCB software.

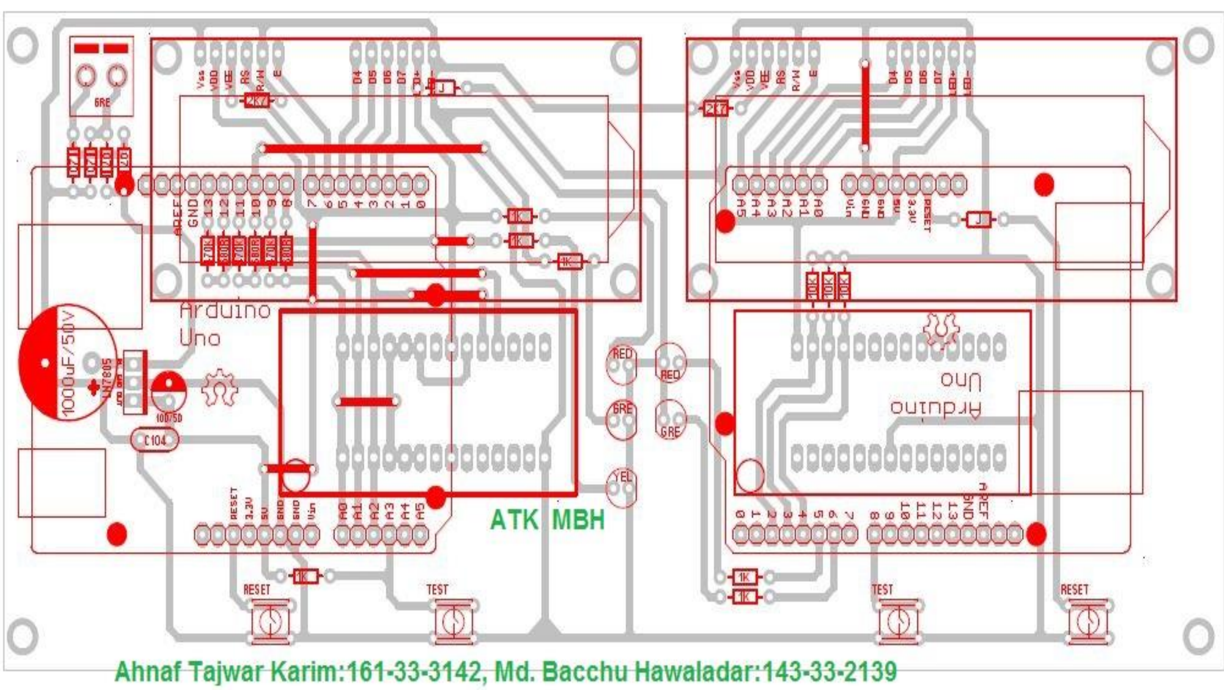


Fig 3.3: PCB Design

3.6 Working Principle

When device power is on than LCD Display show that ready to component testing.When electronic components are plug in ZIF socket and press test button AVR microcontroller analysis the component.Therefore, LCD display module show the component status on screen. Reset button clear the status previous data.

When test button press long time than feature PWM operation mode active. Every test pin (measurement port) can be used as analog input. This measurement every test pin can be switched to output and in this mode it can be directly connected to680Ω resistor or a 470kΩ resistor.

3.7 Firmware and Programming Procedure

Firmware is held in non-volatile memory devices such as ROM, EPROM, or flash memory Changing the firmware of a device was rarely or never done during its lifetime in the past but is nowadays a common procedure some firmware memory devices are permanently installed and cannot be changed after manufacture in computer programming, a procedure is a set of coded instructions that tell a computer how to run a program or calculation. Different types of programming languages can beused to build a procedure.

3.7.1 Programming Microcontroller

A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, Java, FORTRAN, Ada, and Pascal. As the process of writing executable code was endlessly tiring, the first 'higher' programming language called assembly language was created. The truth is that it made the process of programming more complicated, but on the other hand the process of writing program stopped being a nightmare. Instructions in assembly language are represented in the form of meaningful abbreviations, and the process of their compiling into executable code is left over to a special program on a PC called compiler. The main advantage of this programming language is its simplicity, i.e. each program instruction corresponds to one memory location in the Microcontroller. It enables a complete control of what is going on within the chip and thus making this language commonly used in the world.

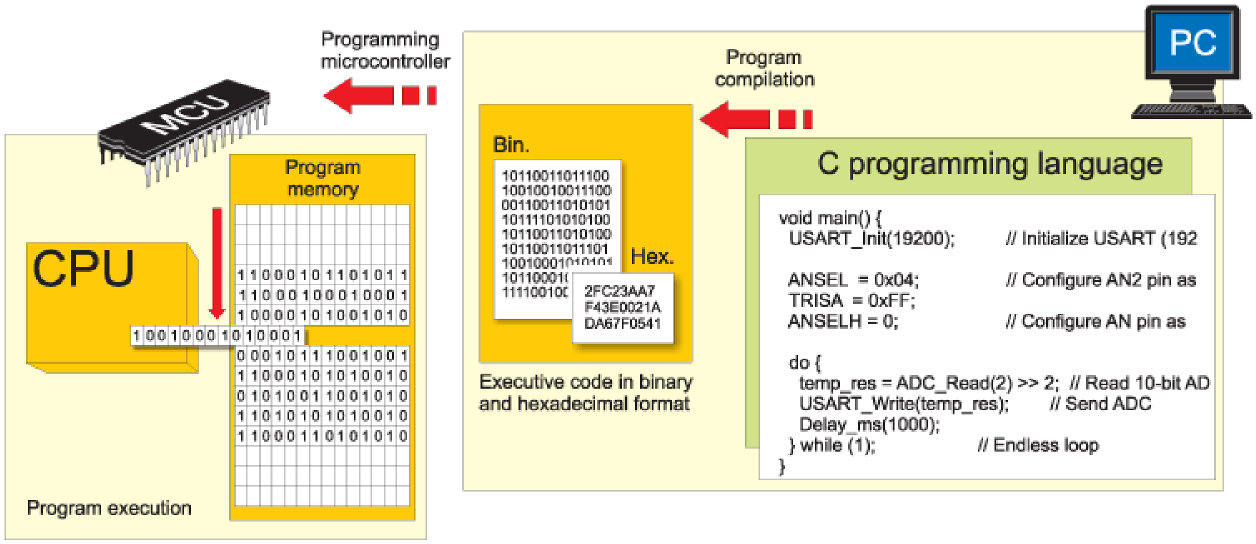


Fig 3.4: Compiling Program

3.7.2 Programming IDE

IDE means integrated development environment .An integrated developmentenvironment is a software application that provides comprehensive facilities to computer programmers for software development and it is very important for us. An IDE normally consists of at least a source code editor, build automation tools and a debugger.It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. It can be used with any Arduino board and a worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

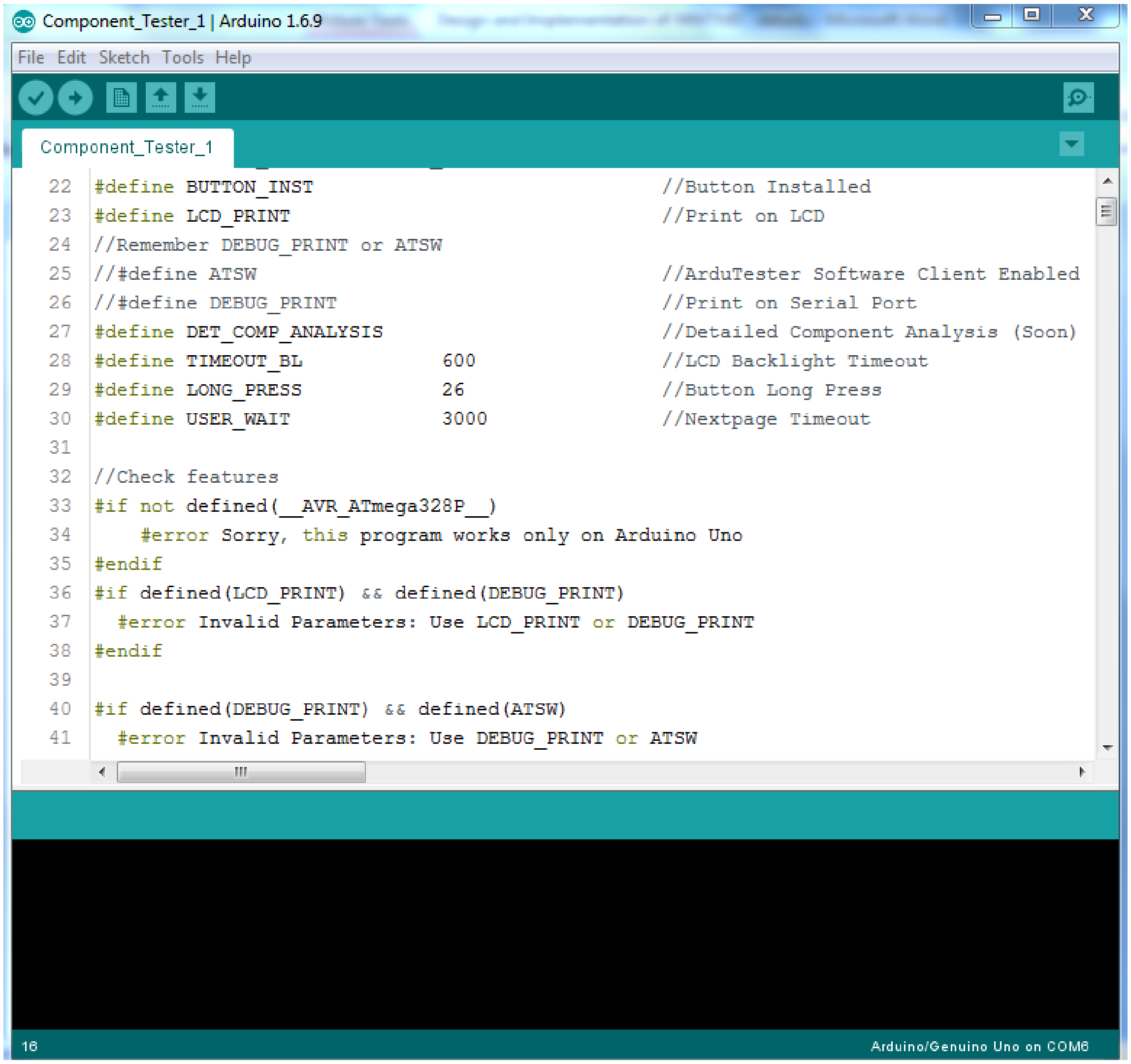


Fig 3.5: Compilation of Program Using Arduino IDE

Arduino was born at the Ivrea Interaction Design Institute as an easy toolfor fast prototyping aimed at students without a background in electronics and programming. Arduino boards are completely open-source and empowering users to build them independently and eventually adapt them to their particular needs. The software is open-source and it is growing through the contributions of users worldwide.

3.7.3 Prototyping Board

In this topic, we will go through different hardware components of an Arduino Board. Arduinos have the majority of the components in common.

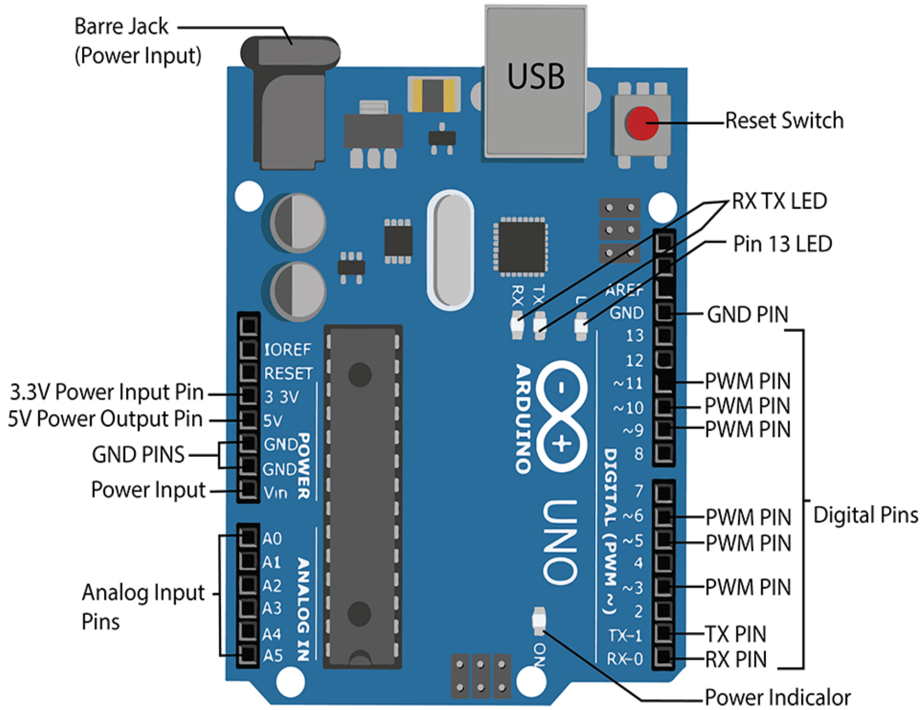


Fig 3.6: Arduino Uno Board

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered three ways: DC Power Jack can be used to power your Arduino board. The barrel jack The recommended voltage for most Arduino models is between 6 and 12 Volts.VIN Pin is used to power the Arduino Uno board using an external power source. The voltage should be within the range mentioned above.USB cable are connected to the computer provides 5 volts at 500mA.The pins on your Arduino are the places where you connectwires to construct a circuit probably in conjunction with a breadboard and some wire. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions:Ground are used to ground circuits.5V: This pin provides 5V Voltage to the circuits.Analog Pins are for reading analog voltage value from sensors and convert them into a digital value that can be read. In Arduino Uno, there are 6 analog pins labeled A0-A5.Digital Pins are for both digital input (reading the state of the switch) and digital output (controlling the LED). In Arduino Uno, there are 14 digital pins (0 -13).PWM Pins: You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). But can also be used for something called Pulse-Width Modulation (PWM). They areused analog output like fading an LED in and out.

RX – TX is serial communication pins and used to communicate with other Arduino boards as well as computers.

Reset Button-This button is used to restart the code that is loaded on the Arduino.
Power Indicator LED-This LED should light up whenever you plug your Arduino into a power source.
RX – TX LEDs-These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data on the RX TX Pins.
Pin 13 LED-Arduino Uno has an inbuilt LED connected to digital pin 13. Whenever the pin is HIGH, LED lights up and when it is LOW, LED is off.

3.7.4 Microcontroller Programmer / Program Burner

A microcontroller burner is a hardware device accompanied with software which is used to transfer the machine language code to the microcontroller/EEPROM from the PC.

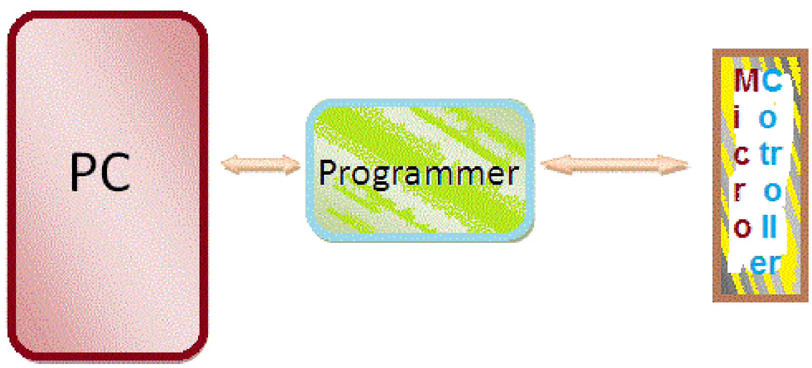


Fig 3.7: Download program to MCU

Most Arduino boards consist of an Atmel 8-bit AVR microcontroller (ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560) with varying amounts of flash memory, pins, and features.Boards are loaded with program code via a serial connection to another computer.
AVR refers to the architectureused on many of Atmel's microprocessors. AVR programming is the process if programming a chip with this architecture which is important to understand as each architecture comes with its own set of quirks and nightnares.Short for automatic voltage regulator, AVR is a hardware device used to maintain a voltage to electronic devices. 2. Short for automatic voice recognition, AVR is the ability of a computer or other electronic devices to identify and understand human voice.

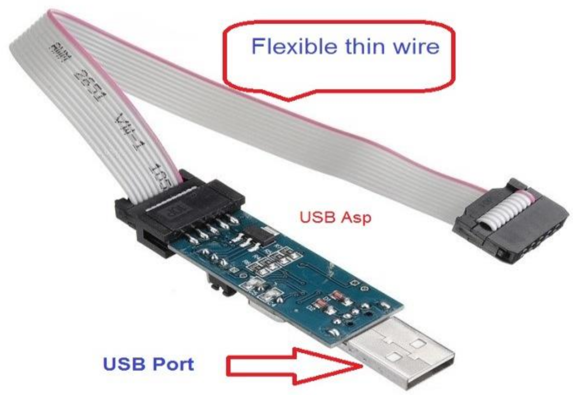


Fig 3.8: (USBasp) AVR Programmer

It contain on chip central processing unit (CPU), Read only memory (ROM), Random access Memory (RAM), input/output unit, interrupts controller etc. Therefore it is used for high speed signal processing operation inside an embedded system. AVR is a microcontroller of the ATMEL family, used in Arduino. ARM is a microprocessor. Arduino Boards come with AVR controllers. Update Arduino Due has been launched which is based on ARM processor. So if you want to compare arduinos with AVRs (Uno, Nano, Leonardo) and Arduinos with ARMs (Due, Zero, Teensy), the big difference IS that the AVR is an 8-bit architecture.

Features of avr microcontroller:

- 32 x 8 general working purpose registers.
- 32K bytes of insystem self programmable flash program memory.
- 2K bytes of internal SRAM.
- 1024 bytes EEPROM.
- 40 pin DIP, 44 lead QTFP, 44-pad QFN/MLF.
- 32 programmable I/O lines.
- 8 Channel, 10 bit ADC.

3.8 Image of the Project

Fig 3.9: Image of the Project

Chapter 4

Basic Logic Gates ICs

4.1 Introduction

Basic logic gate ICs are an electronic device which is used to compute a function on a two valued signal.Basic logic gates are the basic building block of digital circuits and all logic gates have one output and two inputs. Some logic gates like NOT gate has only one input and one output.Digital logic gates can have more than one input, for example, inputs A, B, C, D etc but generally only have one digital output, (Q). Some logic gates can be cascaded together to form a logic gate function with any desired number of inputs and to form combinational and sequential type circuits and differnt logic gate functions from standard gates.

4.2 Logic Gates ICs:

Binary logic gates ICs have been in use since inception with advancement,technology and millennium gate design area. Now it has become tedious andcomplicated. For this purpose the lowpower and voltagearithmetic and logic circuit designed. In this paper we will presents the designand performance of Arithmetic Ternary logic and CMOS design styles. Thedesign is targeted for the 45nm CMOS technology. Design tool for simulation will be MICROWIND 3.1 software and DSCHtool. We will estimate area of power and delay and the designof arithmetic circuit with optimized number of transistors ascompared to binary circuit.In this section, ternary logic system is described. Thesystem includes a set of logic gate operators. The circuitscan be designed using them. As discussed earlier ternarylogic offers significant advantages in development. It is usedto design entry method for our planned project.In ternary logic system, logic levels ranges from 0 to 2 asagainst 0 and 1 in binary logic. The logic systems uses logicgates and their operations is known as operators. The gatesand operators can be interchangeably used.

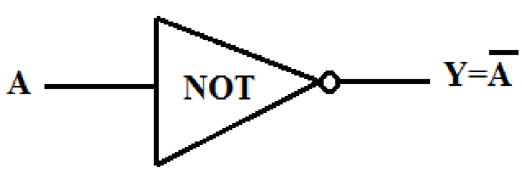
4.3 Types of Logic Gates

There are seven types of Logic Gates. They are:

- NOT Gate (Inverter)
- AND Gate (Multiplication)
- OR Gate (Addition)
- NAND Gate
- NOR Gate
- XOR Gate
- XNOR Gate

4.3.1 NOT Gate (Inverter):

A NOT gate have one input and produceone output in ternary algebra. These gates are known as fundamental operator. Truth tablegives the output 0; in case of when input is 2. Gives output 1 in case of input is 1. And when output is 2 then input would be 0. Similarly, a false input result in a true output.It is a gate which has a single input and a single output. Their truthtable and symbolic representation given below .



Input	Output
A	$Y=\overline{A}$
0	1
1	0

Figure 4.1: NOT Gate & Truth Table

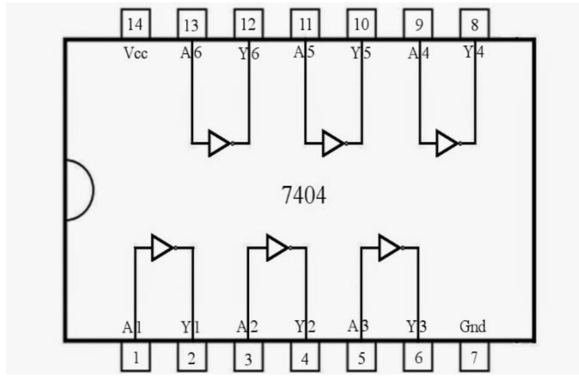
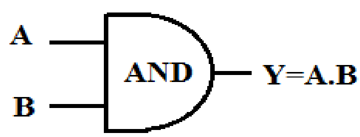


Figure 4.2: NOT GATE IC

4.3.2 AND Gate (Multiplication):

The AND Gate is called Multiplication gate .It has two or many inputs and one output when it is high or 1 when all of the inputs are 1 or high and the output is 0or Low when any of the inputs are 0 or Low.



Inputs		Output
A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 4.3: AND Gate Symbol & Truth Table

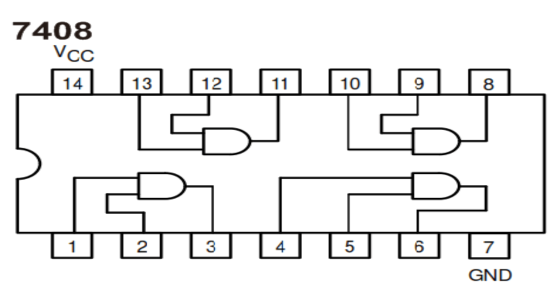
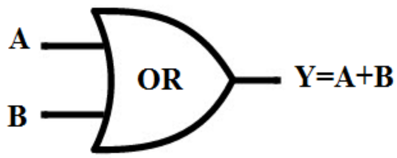


Figure 4.4: AND GATE IC

4.3.3 OR Gate (Addition):

The OR gate has two or many inputs and one output. This is a gate which performs addition which is commonly known as OR functions. This gate is called an OR gate because the gate gives output will be highor 1 only if any or all input values are high or 1 i.e., the output is high or 1 when any one of the inputs is high or 1 and The output is low or 0 when both the inputs are low or 0.



Inputs		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 4.5: OR Gate & Truth Table

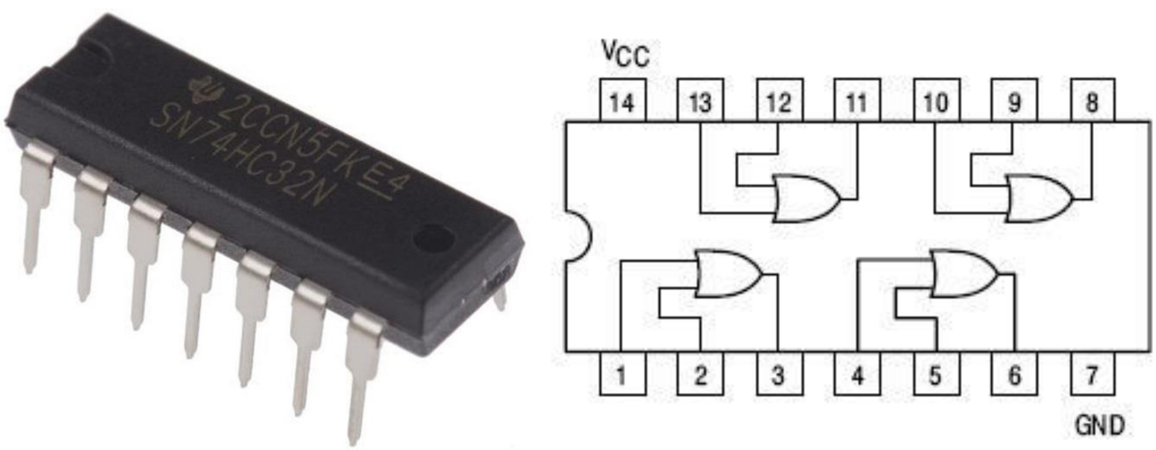
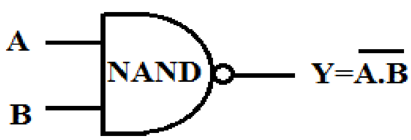


Figure 4.6: OR GATE IC

4.3.4 NAND Gate :

A NAND gate have two inputs and produceone output in ternary algebra. These gates are known as a functional operator. Truthtable gives output 1 in case of when input is 0. Gives output 1 in case input is 0. Givesoutput 1 in casein put is 0, respectively. Their truthtable and symbolic representation given below.



Inputs		Output
A	B	$Y=\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Figure 4.7: NAND Gate & Truth Table

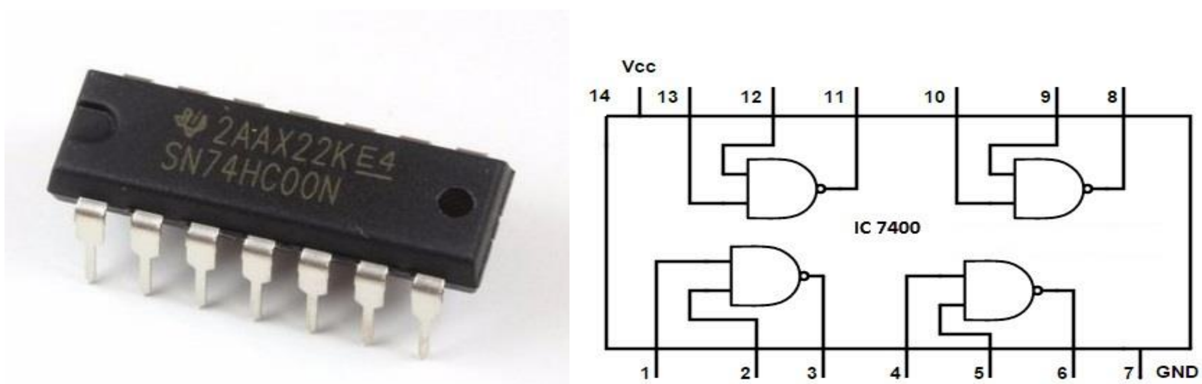
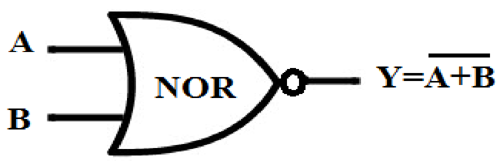


Figure 4.8: NAND GATE IC

A ternary NOT gate or ternary inverter, have one input andproduced one output. Below explains the basic inverter or NOT gate. If input is 0 then output is 1 and if input is 1 then output is 0. In the truth in and out is input and outputrespectively both the input is high then output is low; otherwise outputs high. Below explains the basic NAND gate.The truth table of NAND gate where A0 and A1 are inputs and Vought is output respectively.

4.3.5 NOR GATE:

A NOR gate have two inputs andproduce one output in ternary algebra. These gates are known as a functional operator. Truthtable gives output 1 in case when input is 00. Gives output 1 in case when input is 0. Gives output 0 when input is 1. Theirtruth table and symbolic representation given below



Inputs		Output
A	B	$Y=\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Figure 4.9: NOR Gate & Truth Table

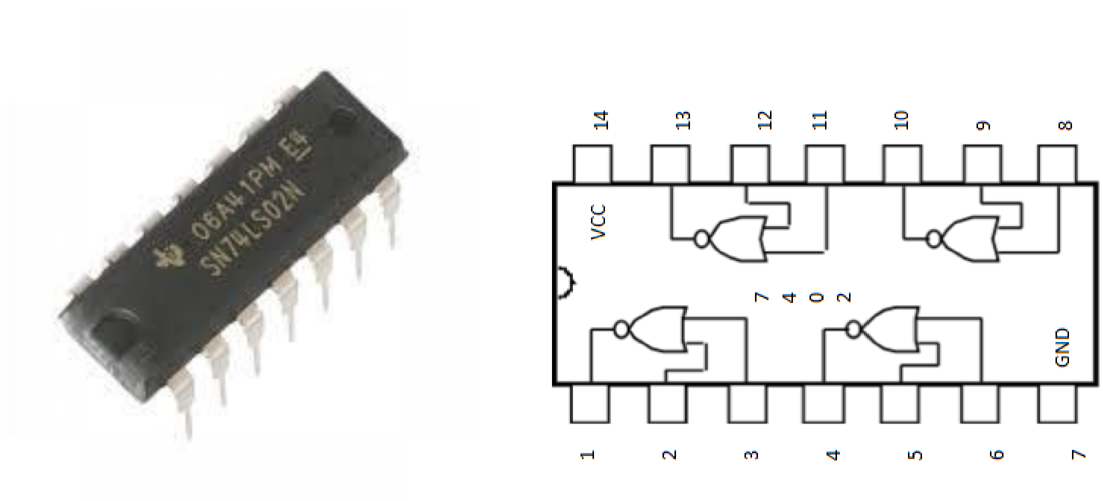
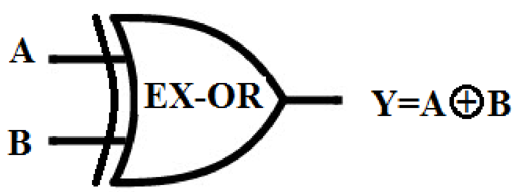


Figure 4.10: NOR GATE IC

4.3.6 ExCLUSIVE-OR GATE (XOR) IC:

The XOR gate is used in digital data processing circuits. In this gate has two or more input terminals and one output terminal. The EX-OR Gate has the output only high means 1 when an odd number of inputs are high or 1 and the output is low or 0 when both the inputs are low or 0 and both the inputs are high or 1 Given below :



Inputs		Output
A	B	$Y=A\oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.11: EX-OR Gate & Truth Table

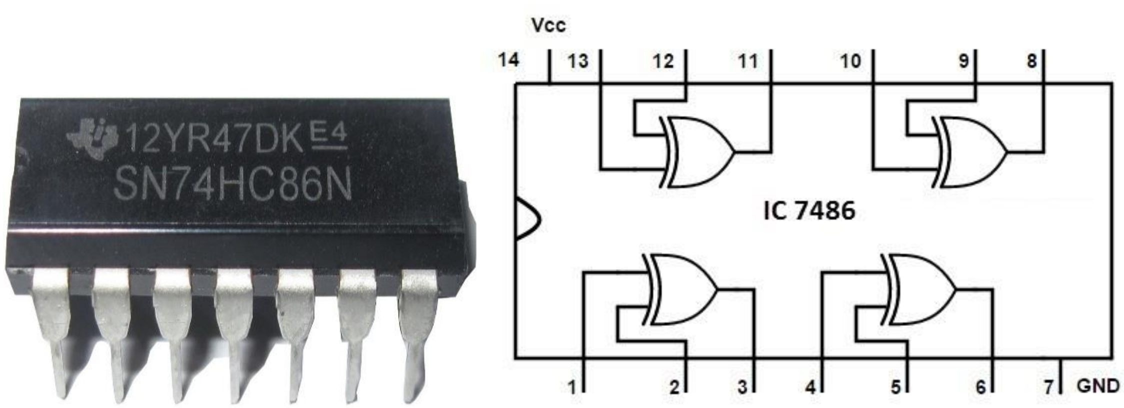


Figure 4.12: ExCLUSIVE-OR GATE (XOR) IC

4.4 Advantages of Logic Gates

The advantages of Logic Gates are:

- Logical Operations are performed using Boolean algebra which makes the circuit design more economical and simple.
- When Logic '1' and Logic '0' can be easily distinguished.

4.5 Disadvantages of Logic Gates

The disadvantages of Logic Gates are:

- Operating Voltage is limited.
- Time delay occurs between input and output.

Chapter 5

Result and Discussion

5.1 Introduction

In this chapter we have discussed and obtainedto contain the results about the full project. Here Part-01 is for Component Testing & Part-02 for the Logic IC testing.We have also covered discussions about advantages, disadvantages and limitation of current version of the **Universal Electronic Component Tester.**

5.2 Result

The project has been run as desired.These universal component tester is able to detect these problems very quickly and easily.It can analysis electronic components. It takes several mA current when on standby. After all, the project showed satisfactory results.

So, here are Component Tester can testing -

1. Resistance-1KΩ:Our component tester value 1019Ω and digital multimeter value 0.981KΩ.

2KΩ: Our component tester value 2266Ω and digital multimeter value 2.16 KΩ.

2.0-5 Volt DC Battery: Our component tester value 143mV and digital multimeter value 1.51V.

3. Inductor: Our component tester value 0.30Ω 118uh digital multimeter value 0.24Ω 108 uh

4. Transistor Model C828-R16: Our component tester value NPN EBC=312, V_BE=159mV and different side NPN EBC=132, h_FE=250mV.

Model A733-GR338: Our component tester value PNP EBC=312, h_FE=289mV

5. MOSFET Model IRF840, N78K: Our component tester value Mosfet N_channel GDS=123, Vth=112Mv, Cgs=2757pF other side GDS=321, Vth=29Mv, Cgs=2844pF

6. Thyristor, TRIAC-Model PH-600E, BT134: Our component tester value Triac GAC=321 other side SCR- Model PH-600E, BT134: Our component tester value SCR GAC=321.

7. Capacitor 50V/10 uF: Our component tester value 51.85 uF digital multimeter value 9.60 uF

16V/47 uF: Our component tester value 45.77 uF digital multimeter value 47.31uF

8. Ceramic Capacitor: Our component tester value 51.81uF digital multimeter value 48.37uF

9. Rectifier Diode: Our component tester value C= 51.28uF, VF=169mV digital multimeter valueC= 52.12uF, VF=168mV.

10. LED: Our component tester value VF=1940mV, C=48.98 uF.

11. Inferred LED: Our component tester value VF=1201mV, C=46.07 uF.

12. Logic IC's:

IC Model	Binary Code	Gate Name
SN74HC08N	0001	AND Gate
SN74HC86N	0110	ExOR Gate
SN74HC32N	0111	OR Gate
SN74HC04N	1000	NOT Gate
SN74HC00N	1110	NAND Gate

5.1 Advantages

- It can testing electronic components.
- It can test Logic ICs with Binary Output.
- It can test 0-5Volt DC Battery.
- The parts used are commonly available and easily replaceable.
- Very low power consumption.
- Cost effective. (Low Price)
- Easy to use. Do not need any high or deep skill for operate this tester.
- It can be repair easily.
- Increased flexibility, quick operation & accuracy of data.

5.2 Disadvantages

- Testing is limited.
- 2-3 seconds need to test each equipment.
- Only Logic IC can testing.
- Timer IC, Op-Amp IC & Memory IC can't testing.
- Not easy to upgrade for adjust

5.5 Cost Analysis

Cost analysis is rendered on table below:

S.N	Name of Part	Specification	Quantity	Price/Unit	Total Price
01.	Atmega 328P	8-bit MCU	2	180/-	360/-
02.	LCD Display	16*2	2	160/-	320/-
03.	ZIF Socket	28 pin	2	100/-	200/-
04.	Push Button	4-pin	4	20/-	80/-
05.	DC Jack/Connector	Mini	1	10/-	10/-
06.	Voltage Regulator	LM7805	1	15/-	15/-
07.	12V AC to DC PwrSply Unit	1000mAh	1	250/-	250/-
08.	Diode	1N4007	1	5/-	5/-
09.	Crystal Oscillator	16 MHz	1	10/-	10/-
10.	Disc Ceramic Capacitor	22pF	2	1/-	2/-
11.	Electrolytic Capacitor	100µF	2	5/-	10/-
12.	Power Switch	Push Switch	1	8/-	8/-
13.	Pin Headers & Connectors	Male, Female	4	15/-	60/-
14.	IC Socket	28 Pin	1	20/-	20/-
15.	Copper Spacer	5 mm	4	6/-	24/-
16.	Copper Clad Board	Single Side	2 Sq. Ft.	150/-	300/-
17.	Ferric Chloride	Liquid	1 mg	100/-	100/-
18.	LED	5 mm	3	2/-	6/-
19.	Resistor	0.25 Watt	11	1/-	11/-
20.	Power Cable	AC cord	1	16/-	16/-
21.	Drill Bit	(0.8-4) mm	4	20/-	80/-
22.	Raping Tape	Carton Size	1	35/-	35/-
23.	Rosin	Smalll	1	20/-	20/-
24.	Prototyping Board	Arduino Uno	2	600/-	1200/-
25.	Prototyping Board	Arduino Mega	1	1250/-	1250/-
26.	AVR Burner	Custom size	1	300/-	300/-
27.	Soldering Lead	60/40 Grade	1 Reel	50/-	50/-
27.	Transportation Expenses	N.A	N.A	N.A	500/-
28.	Arduino UNO		2	400/-	800/-
Total =					**6042/-**

Table 5.1: Cost Analysis

Chapter 6

Conclusion

6.1 Conclusion

In conclusion, the objective to build a **Universal Electronic Component Testing Device** has successfully achieved. In the pastdecades microprocessor based embedded system ruled the market. The last decade witnessed the revolution of Microcontroller based embedded systems it was very importente. Withregards to the requirements gathered the manualwork and the complexity in counting can be achieved with the help of electronic devices it is very sensitive in the world. These simple component testers are able to detect these problems very quickly and easily. However it is still quite easy to perform a simple going test using the simplest of equipment. So, this project can be used for this purpose without multimeter.It will be useful in electronics laboratory and servicing center, personal lab.As the system will be travelable therefore more suitable and cost effective.

6.2 Future Work

There are some scopes for further work in improving **Electronic Component Testing Device**

Those are:

- Adding a feature where user can testing IC (Integrated Circuit).
- Adding a feature where user can testing Logic gates.
- Controlling appliances using PWM.
- Implement sleep mode to gain more power efficiency.

REFERENCES

[1]https://www.arduino.cc/
[2] Book: Digital Logic Families, Written by: Sateesh Babu
[3]https://www.instructables.com/id/Arduino-Components-Tester/
[3] http://www.mikrocontroller.net/articles/AVR-Transistortester
[4] https://en.wikipedia.org/wiki/ISM_band
[5]https://www.eeweb.com/profile/max-maxfield/articles/want-to-build-an-arduino-based-component-tester
[6] http://www.atmel.com/devices/ATMEGA328.aspx
[7] https://www.youtube.com/watch?v=4lVIVKr36iM
[8] Atmega328P Datasheet from Atmel Corporation
[9] https://www.youtube.com/watch?v=b3dPBnYixs4
[10] Atmega328P Datasheet from Atmel Corporation
[11] https://www.electronicshub.org/electronics-projects-ideas/
[12] Atmega328P Datasheet from Atmel Corporation
[13] https://blog.adafruit.com/2013/05/02/ardutester-arduino-component-tester/
[14] https://emguide.wordpress.com
[15] By Sharon D. Nelson, John W. Simek, Michael C. Maschke, Michael Maschke
[16] http://batteryuniversity.com/learn/archive/is_lithium_ion_the_ideal_battery
[17] https://en.wikipedia.org/wiki/Crystal_oscillator
[18] https://en.wikipedia.org/wiki/Iron(III)_chloride
[19] https://www.arduino.cc/en/Guide/Introduction
[20]https://www.clemson.edu/cecas/departments/ece/document_resource/undergrad/electronics/CInquiryLabManual.pdf
[21]www.researchgate.net/publication/297734853_Arduino_and_Open_Source_Computer_Hardware_and_Software
[22]http://www.engineersgarage.com/tutorials/microcontroller-programmer-burner
[23] https://www.academia.edu/3342803/A_capacitive_threshold-logic_gate
[24] http://www.circuitstoday.com/category/testing-components
[25] https://www.electronics-diy.com/esr-meter.php

Appendix

Name of the Project	: **Design & Construction of Universal Electronic Component Tester**
Target Device	: Atmega 328PU (Arduino UNO)
	Arduino Uno Bootloader
	16MHz external crystal
Coding By	: **Ahnaf Tajwar Karim** & **Md. Bachu Howlader**
Last Update	: 07 January, 2020

```
Universal Electronic Component Tester


#define BUTTON_INST                       //Button Installed
#define LCD_PRINT                         //Print on LCD
#define DET_COMP_ANALYSIS   //Detailed Component Analysis (Soon)
#define TIMEOUT_BL            600        //LCD Backlight Timeout
#define LONG_PRESS            26             //Button Long Press
#define USER_WAIT             3000           //Next page Timeout
#if not defined(__AVR_ATmega328P__)
#endif
#if defined(LCD_PRINT) && defined(DEBUG_PRINT)
#error Invalid Parameters: Use LCD_PRINT or DEBUG_PRINT
#endif
#if defined(DEBUG_PRINT) && defined(ATSW)
#error Invalid Parameters: Use DEBUG_PRINT or ATSW
#endif
#include <avr/wdt.h>
#include <avr/sleep.h>
#include <avr/power.h>
#include <EEPROM.h>
#ifdef LCD_PRINT
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);          //RS,E,D4,D5,D6,D7
#endif
#define UINT32_MAX            ((uint32_t)-1)
#define ADC_PORT              PORTC     //ADC port data register
#define ADC_DDR          DDRC //ADC port data direction register
#define ADC_PIN               PINC    //Port input pins register
#define TP1                   0                //Test pin 1 (=0)
#define TP2                   1                //Test pin 2 (=1)
#define TP3                   2                //Test pin 3 (=2)
#define R_PORT                PORTB         //Port data register
#define R_DDR               DDRB //Port data direction register
```

```
#define TEST_BUTTON     A3 //Test/start push button (low active)
#define CYCLE_DELAY           3000
#define CYCLE_MAX             5
#define UREF_VCC              5001
#define UREF_OFFSET           1250
#define R_LOW                 680
#define R_HIGH                470000
#define RH_OFFSET             700
#define R_ZERO                20
#define CAP_WIRES             15
#define CAP_PROBELEADS        9
#define CAP_DISCHARGED        2
#define ADC_SAMPLES           25
#define R_MCU_LOW             200               //Default: 209
#define R_MCU_HIGH            220               //Default: 235
#define COMPARATOR_OFFSET     15
#define CAP_PCB               42
#define C_ZERO              CAP_PCB + CAP_WIRES + CAP_PROBELEADS
#define ADC_CLOCK_DIV   (1 << ADPS2) | (1 << ADPS1)|(1 << ADPS0)
#define CPU_FREQ              F_CPU
#define OSC_STARTUP           16384
#define COMP_NONE             0
#define COMP_ERROR            1
#define COMP_MENU             2
#define COMP_RESISTOR         10
#define COMP_CAPACITOR        11
#define COMP_INDUCTOR         12
#define COMP_DIODE            20
#define COMP_BJT              21
#define COMP_FET              22
#define COMP_IGBT             23
#define COMP_TRIAC            24
#define COMP_THYRISTOR        25
#define LCD_CHAR_UNSET        0            //Just a place holder
#define LCD_CHAR_DIODE1       1            //Diode icon '>|'
#define LCD_CHAR_DIODE2       2            //Diode icon '|<'
#define LCD_CHAR_CAP          3            //Capacitor icon '||'
#define LCD_CHAR_FLAG         4              //Flag Icon
#define LCD_CHAR_RESIS1       6         //Resistor left icon '['
#define LCD_CHAR_RESIS2       7        //Resistor right icon ']'
#ifdef DEBUG_PRINT
#define LCD_CHAR_OMEGA      79
#define LCD_CHAR_MICRO '\u00B5'//Code for Arduino Serial Monitor
#else
#define LCD_CHAR_OMEGA      244                 //Default: 244
#define LCD_CHAR_MICRO      228
#endif
```

```
#define TYPE_DISCHARGE        1                //Discharge error
#define TYPE_N_CHANNEL        0b00000001         //n channel
#define TYPE_P_CHANNEL        0b00000010         //p channel
#define TYPE_ENHANCEMENT      0b00000100      //Enhancement mode
#define TYPE_DEPLETION        0b00001000       //Depletion mode
#define TYPE_MOSFET           0b00010000         //MOSFET
#define TYPE_JFET             0b00100000         //JFET
#define TYPE_IGBT             0b01000000         //IGBT (no FET)
#define MODE_LOW_CURRENT      0b00000001      //Low test current
#define MODE_HIGH_CURRENT     0b00000010     //High test current
#define MODE_DELAYED_START    0b00000100         //Delayed start
#define TYPE_NPN              1                  //NPN
#define TYPE_PNP              2                  //PNP
#define MODE_CONTINOUS        0                  //Continuous
#define MODE_AUTOHOLD         1                  //Auto hold
#define TABLE_SMALL_CAP       1
#define TABLE_LARGE_CAP       2
#define TABLE_INDUCTOR        3
#define FLAG_PULLDOWN         0b00000000
#define FLAG_PULLUP           0b00000001
#define FLAG_1MS              0b00001000
#define FLAG_10MS             0b00010000
typedef struct
{
  byte                   TesterMode;  //Tester operation mode
  byte                   SleepMode;    //MCU sleep mode
  byte                    Samples;     //Number of ADC samples
  byte      AutoScale;           //Flag to disable/enable ADC
auto scaling
  byte        RefFlag;           //Internal control flag for ADC
  unsigned int  U_Bandgap;      //Voltage of internal bandgap
reference (mV)
  unsigned int               RiL;          //Internal pin
resistance of Aâ€šÃ‚lC in low mode (0.1 Ohms)
  unsigned int               RiH;                //Internal pin
resistance of Aâ€šÃ‚lC in high mode (0.1 Ohms)
  unsigned int               RZero;              //Resistance of
probe leads (2 in series) (0.01 Ohms)
  byte                       CapZero;            //Capacity zero
offset (input + leads) (pF)
  signed char                RefOffset;          //Voltage
offset of bandgap reference (mV)
  signed char                CompOffset;         //Voltage
offset of analog comparator (mV)
} Config_Type;
typedef struct
{
```

```
  byte                        Pin_1;          //Probe-1
  byte                        Pin_2;          //Probe-2
  byte                        Pin_3;          //Probe-3
  byte                        Rl_1;        //Rl mask for probe-1
  byte                        Rh_1;        //Rh mask for probe-1
  byte                        Rl_2;        //Rl mask for probe-2
  byte                        Rh_2;        //Rh mask for probe-2
  byte                        Rl_3;        //Rl mask for probe-3
  byte                        Rh_3;        //Rh mask for probe-3
  byte                        ADC_1;      //ADC mask for probe-1
  byte                        ADC_2;      //ADC mask for probe-2
} Probe_Type;
typedef struct
{
  byte                Done; //Flag for transistor detection done
  byte                Found;    //Component type which was found
  byte                  Type;      //Component specific subtype
  byte                Resistors;  //Number of resistors found
  byte                   Diodes;      //Number of diodes found
  byte                    Probe;             //Error: probe pin
  unsigned int                U;   //Error: voltage left in mV
} Check_Type;
typedef struct
{
  byte                   A;                  //Probe pin #1
  byte                   B;                  //Probe pin #2
  byte               Scale;  //Exponent of factor (value * 10^x)
  unsigned long           Value;       //Resistance
} Resistor_Type;
typedef struct
{
  byte                        A;                  //Probe pin #1
  byte                        B;                  //Probe pin #2
  signed char                 Scale;              //Exponent of
factor (value * 10^x)
  unsigned long           Value; //Capacitance incl. zero offset
  unsigned long          Raw;   //Capacitance excl. zero offset
} Capacitor_Type;
typedef struct
{
  signed char        Scale;  //Exponent of factor (value * 10^x)
  unsigned long      Value;        //Inductance
} Inductor_Type;
typedef struct
{
  byte                      A;    //Probe pin connected to anode
  byte                      C;   //Probe pin connected to cathode
```

```
  unsigned int       V_f; //Forward voltage in mV (high current)
  unsigned int       V_f2; //Forward voltage in mV (low current)
} Diode_Type;
typedef struct
{
  byte                     B;  //Probe pin connected to base
  byte                     C; //Probe pin connected to collector
  byte                     E;  //Probe pin connected to emitter
  unsigned long        hFE;      //Current amplification factor
  unsigned int         I_CE0;    //Leakage current (in Aâ€šÃ,lA)
} BJT_Type;
typedef struct
{
  byte                     G;    //Test pin connected to gate
  byte                     D;   //Test pin connected to drain
  byte                      S;   //Test pin connected to source
  unsigned int          V_th;  //Threshold voltage of gate in mV
} FET_Type;
typedef struct
{
} Error_Type;
char                    OutBuffer[12];
char                    PRGBuffer[32];
Config_Type          Config;  //Tester modes, offsets and values
//Probing
Probe_Type           Probes;   //Test probes
Check_Type           Check;             //Checking/testing
Resistor_Type       Resistors[3];   //Resistors (3 combinations)
Capacitor_Type       Caps[3];     //Capacitors (3 combinations)
Diode_Type  Diodes[6]; //Diodes (3 combinations in 2 directions)
BJT_Type                 BJT;    //Bipolar junction transistor
FET_Type         FET;           //FET
Inductor_Type           Inductor;    //Inductor
class __FlashStringHelper;
#define X(str) (strcpy_P(PRGBuffer, PSTR(str)), PRGBuffer)
const unsigned char Mode_str[] PROGMEM = "Mode:";
const unsigned char Continous_str[] PROGMEM = "Continous";
const unsigned char AutoHold_str[] PROGMEM = "Auto Hold";
const unsigned char Running_str[] PROGMEM = "Testing...";
const unsigned char Weak_str[] PROGMEM = "weak";
const unsigned char Low_str[] PROGMEM = "low";
const unsigned char Failed1_str[] PROGMEM = "  No Component";
const unsigned char Failed2_str[] PROGMEM = "     Found!";
const unsigned char Thyristor_str[] PROGMEM = "SCR";
const unsigned char Triac_str[] PROGMEM = "Triac";
const unsigned char GAK_str[] PROGMEM = "GAC=";
const unsigned char Done_str[] PROGMEM = "       OK";
```

```
const unsigned char Select_str[] PROGMEM = "Select";
const unsigned char Selftest_str[] PROGMEM = "Selftest";
const unsigned char Adjustment_str[] PROGMEM = "Adjustment";
const unsigned char Default_str[] PROGMEM = "Default Values";
const unsigned char Save_str[] PROGMEM = "Save";
const unsigned char Show_str[] PROGMEM = "Show Values";
const unsigned char Remove_str[] PROGMEM = "Remove";
const unsigned char Create_str[] PROGMEM = "Create";
const unsigned char ShortCircuit_str[] PROGMEM = "Short
Circuit!";
const unsigned char DischargeFailed_str[] PROGMEM = "Battery?";
const unsigned char Error_str[] PROGMEM = "Error!";
const unsigned char Battery_str[] PROGMEM = "Bat.";
const unsigned char OK_str[] PROGMEM = "ok";
const unsigned char MOS_str[] PROGMEM = "MOS";
const unsigned char FET_str[] PROGMEM = "FET";
const unsigned char Channel_str[] PROGMEM = "-ch";
const unsigned char Enhancement_str[] PROGMEM = "enh.";
const unsigned char Depletion_str[] PROGMEM = "dep.";
const unsigned char IGBT_str[] PROGMEM = "IGBT";
const unsigned char GateCap_str[] PROGMEM = "Cgs=";
const unsigned char GDS_str[] PROGMEM = "GDS=";
const unsigned char GCE_str[] PROGMEM = "GCE=";
const unsigned char NPN_str[] PROGMEM = "NPN";
const unsigned char PNP_str[] PROGMEM = "PNP";
const unsigned char EBC_str[] PROGMEM = "EBC=";
const unsigned char hFE_str[] PROGMEM = "h_FE=";
const unsigned char V_BE_str[] PROGMEM = "V_BE=";
const unsigned char I_CEO_str[] PROGMEM = "I_CEO=";
const unsigned char Vf_str[] PROGMEM = "Vf=";
const unsigned char DiodeCap_str[] PROGMEM = "C=";
const unsigned char Vth_str[] PROGMEM = "Vth=";
const unsigned char I_R_str[] PROGMEM = "I_R=";
const unsigned char URef_str[] PROGMEM = "Vref";
const unsigned char RhLow_str[] PROGMEM = "Rh-";
const unsigned char RhHigh_str[] PROGMEM = "Rh+";
const unsigned char RiLow_str[] PROGMEM = "Ri-";
const unsigned char RiHigh_str[] PROGMEM = "Ri+";
const unsigned char Rl_str[] PROGMEM = "+Rl-";
const unsigned char Rh_str[] PROGMEM = "+Rh-";
const unsigned char ProbeComb_str[] PROGMEM = "12 13 23";
const unsigned char CapOffset_str[] PROGMEM = "C0";
const unsigned char ROffset_str[] PROGMEM = "R0";
const unsigned char CompOffset_str[] PROGMEM = "AComp";
const unsigned char PWM_str[] PROGMEM = "PWM";
const unsigned char Hertz_str[] PROGMEM = "Hz";
const unsigned char Title_str[] PROGMEM = "Component Tester";
```

```
const unsigned char Organigation_str[] PROGMEM = "      v1.0";
#ifdef DEBUG_PRINT
const unsigned char Cap_str[] PROGMEM = {'-', '|', '|', '-', 0};
const unsigned char Diode_AC_str[] PROGMEM = {'-', '>', '-', 0};
const unsigned char Diode_CA_str[] PROGMEM = {'-', '<', '-', 0};
const unsigned char Diodes_str[] PROGMEM = {'*', '>', ' ', ' ',
0};
const unsigned char Resistor_str[] PROGMEM = {'-', '[', ']', '-
', 0};
#else
const unsigned char Cap_str[] PROGMEM = {'-', LCD_CHAR_CAP, '-',
0};
const unsigned char Diode_AC_str[] PROGMEM = {'-',
LCD_CHAR_DIODE1, '-', 0};
const unsigned char Diode_CA_str[] PROGMEM = {'-',
LCD_CHAR_DIODE2, '-', 0};
const unsigned char Diodes_str[] PROGMEM = {'*',
LCD_CHAR_DIODE1, ' ', ' ', 0};
const unsigned char Resistor_str[] PROGMEM = {'-',
LCD_CHAR_RESIS1, LCD_CHAR_RESIS2, '-', 0};
#endif
byte DiodeIcon1[8]  = {0x11, 0x19, 0x1d, 0x1f, 0x1d, 0x19, 0x11,
0x00};
byte DiodeIcon2[8]  = {0x11, 0x13, 0x17, 0x1f, 0x17, 0x13, 0x11,
0x00};
byte CapIcon[8]  = {0x1b, 0x1b, 0x1b, 0x1b, 0x1b, 0x1b, 0x1b,
0x00};
byte ResIcon1[8]  = {0x00, 0x0f, 0x08, 0x18, 0x08, 0x0f, 0x00,
0x00};
byte ResIcon2[8]  = {0x00, 0x1e, 0x02, 0x03, 0x02, 0x1e, 0x00,
0x00};
byte FlagIcon[8] = {0x1f, 0x11, 0x0e, 0x04, 0x0a, 0x15, 0x1f,
0x00};
const unsigned char Prefix_table[]  = {'p', 'n', LCD_CHAR_MICRO,
'm', 0, 'k', 'M'};
const unsigned int PWM_Freq_table[]  = {100, 250, 500, 1000,
2500, 5000, 10000, 25000};
const unsigned int LargeCap_table[]  = {23022, 21195, 19629,
18272, 17084, 16036, 15104, 14271, 13520, 12841, 12224, 11660,
11143, 10668, 10229, 9822, 9445, 9093, 8765, 8458, 8170, 7900,
7645, 7405, 7178, 6963, 6760, 6567, 6384, 6209, 6043, 5885,
5733, 5589, 5450, 5318, 5191, 5069, 4952, 4839, 4731, 4627,
4526, 4430, 4336};
const unsigned int SmallCap_table[]  = {954, 903, 856, 814, 775,
740, 707, 676, 648};
const unsigned int Inductor_table[]  = {4481, 3923, 3476, 3110,
2804, 2544, 2321, 2128, 1958, 1807, 1673, 1552, 1443, 1343,
```

```
1252, 1169, 1091, 1020, 953, 890, 831, 775, 721, 670, 621, 574,
527, 481, 434, 386, 334, 271};
const unsigned char Rl_table[]  = {(1 << (TP1 * 2)), (1 << (TP2
* 2)), (1 << (TP3 * 2))};
const unsigned char ADC_table[]  = {(1 << TP1), (1 << TP2), (1
<< TP3)};
byte SmallCap(Capacitor_Type *Cap);
byte LargeCap(Capacitor_Type *Cap);
byte MeasureInductor(Resistor_Type *Resistor);
void ShowDiode_Uf(Diode_Type *Diode);
void ShowDiode_C(Diode_Type *Diode);
byte                          RunsPassed;        //Counter for
successful measurements
byte                          RunsMissed;        //Counter for
failed/missed measurements
byte                          ErrFnd;            //An Error is
occured

/****************************************************************
****************/
/****************************************************************
****************/
/****************************************************************
****************/


void setup()
{
  byte                        Test;              //Test value
  power_spi_disable();
  power_twi_disable();
  power_timer2_disable();
#ifdef LCD_PRINT
  lcd.begin(16, 2);
  delay(5);
  lcd.createChar(LCD_CHAR_DIODE1, DiodeIcon1); //Diode symbol
|<|
  lcd.createChar(LCD_CHAR_DIODE2, DiodeIcon2); //Diode symbol
|<|
  lcd.createChar(LCD_CHAR_CAP, CapIcon);       //Capacitor
symbol ||
  lcd.createChar(LCD_CHAR_RESIS1, ResIcon1);   //Resistor symbol
[
  lcd.createChar(LCD_CHAR_RESIS2, ResIcon2);   //Resistor symbol
]
  lcd.createChar(LCD_CHAR_FLAG, FlagIcon);     //Flag symbol
  lcd.home();
```

```
  lcd_fixed_string(Title_str);
  lcd_line(2);
  lcd_fixed_string(Organigation_str);
#endif
#ifdef ATSW                                              //Client Begin
  Serial.begin(19200);
#endif
#ifdef DEBUG_PRINT
  Serial.begin(9600);                                    //Serial Output
#endif
  ADCSRA = (1 << ADEN) | ADC_CLOCK_DIV;                    //Enable ADC
and set clock divider
  MCUSR &= ~(1 << WDRF);                                   //Reset
watchdog flag
  DIDR0 = 0b00110111;
  wdt_disable();                                           //Disable
watchdog
  Config.Samples = ADC_SAMPLES;                            //Number of ADC
samples
  Config.AutoScale = 1;                                    //Enable ADC
auto scaling
  Config.RefFlag = 1;                                      //No ADC
reference set yet
  delay(100);
  RunsMissed = 0;
  RunsPassed = 0;
  Config.TesterMode = MODE_CONTINOUS;                      //Set default
mode: continous
#ifdef BUTTON_INST
  pinMode(TEST_BUTTON, INPUT_PULLUP);                    //Initialize the
pushbutton pin as an input
#endif
  LoadAdjust();                                            //Load
adjustment values
#ifdef DEBUG_PRINT
  Serial.print(X("A  R  D  U  T  E  S  T  E  R "));
  lcd_fixed_string(Organization_str);                         //Print
Ardutester Version
  Serial.println();
  Serial.println(X("      By PighiXXX & PaoloP"));
  Serial.println(X("original version by Markus Reschke"));
  Serial.println();
#ifdef BUTTON_INST
  Serial.print(X("Press Button to Probe"));
  Serial.println(X(", long press enter Menu"));
#endif
#endif
```

```
  delay(100);
}

/*************************************************************
***************/
/*************************************************************
***************/
/*************************************************************
***************/
void loop()
{
  byte Test;
#ifdef BUTTON_INST
  Test = TestKey(0, 0);                              //Wait user
#else
  delay(3000);                                       //No button
installed, Wait 3 seconds
  Test = 1;                                          //No button, no
menu :-)
#endif
#ifdef WDT_enabled
  wdt_enable(WDTO_2S);                               //Enable watchdog
(timeout 2s)
#endif
  Check.Found = COMP_NONE;
  Check.Type = 0;
  Check.Done = 0;
  Check.Diodes = 0;
  Check.Resistors = 0;
  BJT.hFE = 0;
  BJT.I_CE0 = 0;
  SetADCHiz();                                        //Set all pins
of ADC port as input
  lcd_clear();                                        //Clear LCD
#ifdef LCD_PRINT
  lcd_fixed_string(Title_str);
  lcd_line(2);
  lcd_fixed_string(Organigation_str);
#endif
  Config.U_Bandgap = ReadU(0x0e);                     //Dummy read
for bandgap stabilization
  Config.Samples = 200;                               //Do a lot of
samples for high accuracy
  Config.U_Bandgap = ReadU(0x0e);                     //Get voltage
of bandgap reference
  Config.Samples = ADC_SAMPLES;                       //Set samples
back to default
```

```
  Config.U_Bandgap += Config.RefOffset;          //Add voltage
offset
  if (Test == 2)                                 //Long Press
  {
    wdt_disable();                               //Disable
watchdog
    MainMenu();                                  //Main Menu
  }
  else
  {
    if (AllProbesShorted() == 3)                 //All probes
Shorted!
    {
#ifdef DEBUG_PRINT
      Serial.println();
#endif
      lcd_fixed_string(Remove_str);             //Display:
Remove/Create
      lcd_line(2);
      lcd_fixed_string(ShortCircuit_str);       //Display: short
circuit!
    }
    else
    {
      lcd_line(2);                              //Move to line #2
      lcd_fixed_string(Running_str);            //Display:
Testing...
      DischargeProbes();
      if (Check.Found == COMP_ERROR)            //Discharge
failed
      { //Only for Standalone Version!
        lcd_fixed_string(DischargeFailed_str); //Display:
Battery?
        lcd_line(2);
        lcd_testpin(Check.Probe);
        lcd_data(':');
        lcd_space();
        DisplayValue(Check.U, -3, 'V');
      }
      else                                      //Skip all other
checks
      {
        CheckProbes(TP1, TP2, TP3);
        CheckProbes(TP2, TP1, TP3);
        CheckProbes(TP1, TP3, TP2);
        CheckProbes(TP3, TP1, TP2);
        CheckProbes(TP2, TP3, TP1);
```

```
        CheckProbes(TP3, TP2, TP1);
        if ((Check.Found == COMP_NONE) ||
            (Check.Found == COMP_RESISTOR))
        {
#ifdef DEBUG_PRINT
          Serial.println();
          Serial.println(X("Wait a moment..."));
#else
          lcd_clear_line(2);
          lcd_fixed_string(Running_str);
          lcd_data('.');
#endif
          MeasureCap(TP3, TP1, 0);
#ifdef LCD_PRINT
          lcd_data('.');
#endif
          MeasureCap(TP3, TP2, 1);
#ifdef LCD_PRINT
          lcd_data('.');
#endif
          MeasureCap(TP2, TP1, 2);
        }
        lcd_clear();
#ifdef BUTTON_INST
        pinMode(TEST_BUTTON, INPUT_PULLUP);  //Reinitialize the
pushbutton pin as an input
#endif
#ifdef DEBUG_PRINT
        Serial.print("Found: ");
        switch (Check.Found)
        {
          case COMP_ERROR:
            Serial.println(X("Component Error!"));
            break;
          case COMP_NONE:
            Serial.println(X("No Component!"));
            break;
          case COMP_RESISTOR:
            Serial.println(X("Resistor"));
            break;
          case COMP_CAPACITOR:
            Serial.println(X("Capacitor"));
            break;
          case COMP_INDUCTOR:
            Serial.println(X("Inductor"));
            break;
          case COMP_DIODE:
```

```
            Serial.println(X("Diode"));
            break;
          case COMP_BJT:
            Serial.println(X("BJT"));
            break;
          case COMP_FET:
            Serial.println(X("FET"));
            break;
          case COMP_IGBT:
            Serial.println(X("IGBT"));
            break;
          case COMP_TRIAC:
            Serial.println(X("TRIAC"));
            break;
          case COMP_THYRISTOR:
            Serial.println(X("Thyristor"));
            break;
        }
#endif
        switch (Check.Found)
        {
          case COMP_ERROR:
            ShowError();
            break;
          case COMP_DIODE:
            ShowDiode();
            break;
          case COMP_BJT:
            ShowBJT();
            break;
          case COMP_FET:
            ShowFET();
            break;
          case COMP_IGBT:
            ShowIGBT();
            break;
          case COMP_THYRISTOR:
            ShowSpecial();
            break;
          case COMP_TRIAC:
            ShowSpecial();
            break;
          case COMP_RESISTOR:
            ShowResistor();
            break;
          case COMP_CAPACITOR:
            ShowCapacitor();
```

```
            break;
          default:                                      //No component
found
            ShowFail();
        }
#ifdef ATSW                             //Client output
        Serial.println("@>");
        Serial.println(Check.Found);
        Serial.println("|");
        Serial.println(Check.Type);
        Serial.println("|");
        Serial.println(Check.Done);
        Serial.println("|");
        Serial.println("@<");
#endif
        RunsMissed = 0;                                 //Reset counter
        RunsPassed++;                                   //Increase
counter
      }
    }
  }
  delay(1000);                                            //Let the user
read the text
  wdt_disable();                                          //Disable
watchdog
}//end of void loop

/****************************************************************
****************/
/****************************************************************
****************/
/****************************************************************
****************/

void SetADCHiz(void)
{
  ADC_DDR &= ~(1 << TP1);
  ADC_DDR &= ~(1 << TP2);
  ADC_DDR &= ~(1 << TP3);
}
void SetADCLow(void)
{
  ADC_PORT &= ~(1 << TP1);
  ADC_PORT &= ~(1 << TP2);
  ADC_PORT &= ~(1 << TP3);
}
void UpdateProbes(byte Probe1, byte Probe2, byte Probe3)
```

```
{
  Probes.Pin_1 = Probe1;
  Probes.Pin_2 = Probe2;
  Probes.Pin_3 = Probe3;
  Probes.Rl_1 = Rl_table[Probe1];
  Probes.Rh_1 = Probes.Rl_1 + Probes.Rl_1;
  Probes.ADC_1 = ADC_table[Probe1];
  Probes.Rl_2 = Rl_table[Probe2];
  Probes.Rh_2 = Probes.Rl_2 + Probes.Rl_2;
  Probes.ADC_2 = ADC_table[Probe2];
  Probes.Rl_3 = Rl_table[Probe3];
  Probes.Rh_3 = Probes.Rl_3 + Probes.Rl_3;
}
byte ShortedProbes(byte Probe1, byte Probe2)
{
  byte                        Flag = 0;          //Return value
  unsigned int                U1;                //Voltage at
probe #1 in mV
  unsigned int                U2;                //Voltage at
probe #2 in mV
  R_PORT = Rl_table[Probe1];
  R_DDR = Rl_table[Probe1] | Rl_table[Probe2];
  U1 = ReadU(Probe1);
  U2 = ReadU(Probe2);
  if ((U1 > UREF_VCC / 2 - 30) && (U1 < UREF_VCC / 2 + 30))
  {
    if ((U2 > UREF_VCC / 2 - 30) && (U2 < UREF_VCC / 2 + 30))
    {
      Flag = 1;
    }
  }
  R_DDR = 0;
  return Flag;
}
byte AllProbesShorted(void)
{
  byte                        Flag = 0;          //Return value
  Flag = ShortedProbes(TP1, TP2);
  Flag += ShortedProbes(TP1, TP3);
  Flag += ShortedProbes(TP2, TP3);
  return Flag;
}
void DischargeProbes(void)
{
  byte                        Counter;           //Loop control
  byte                        Limit = 40;        //Sliding
timeout (2s)
```

```
  byte                        ID;                 //Test pin
  byte                        DischargeMask;      //Bitmask
  unsigned int                U_c;                //Current
voltage
  unsigned int                U_old[3];           //Old voltages
  SetADCHiz();
  SetADCLow();
  R_PORT = 0;
  R_DDR = (2 << (TP1 * 2)) | (2 << (TP2 * 2)) | (2 << (TP3 *
2));
  R_DDR |= (1 << (TP1 * 2)) | (1 << (TP2 * 2)) | (1 << (TP3 *
2));
  U_old[0] = ReadU(TP1);
  U_old[1] = ReadU(TP2);
  U_old[2] = ReadU(TP3);
  Counter = 1;
  ID = 2;
  DischargeMask = 0;
  while (Counter > 0)
  {
    ID++;                                         //Next probe
    if (ID > 2) ID = 0;                           //Start with
probe #1 again
    if (DischargeMask & (1 << ID))                //Skip
discharged probe
      continue;
    U_c = ReadU(ID);                              //Get voltage
of probe
    if (U_c < U_old[ID])                          //Voltage
decreased
    {
      U_old[ID] = U_c;                            //Update old
value
      if ((Limit - Counter) < 20)
      {
        if (Limit < (255 - 20)) Limit += 20;
      }
      Counter = 1;                                //Reset no-
changes counter
    }
    else                                          //Voltage not
decreased
    {
      if ((U_c < 10) && (Limit <= 40)) Limit = 80;
      Counter++;                                  //Increase no-
changes counter
    }
```

```
    if (U_c <= CAP_DISCHARGED)                       //Seems to be
discharged
    {
      DischargeMask |= (1 << ID);                    //Set flag
    }
    else if (U_c < 800)                              //Extra pull-
down
    {
      ADC_DDR |= ADC_table[ID];
    }
    if (DischargeMask == 0b00000111)                 //All probes
discharged
    {
      Counter = 0;                                   //End loop
    }
    else if (Counter > Limit)                        //No decrease
for some time
    {
      //Might be a battery or a super cap
      Check.Found = COMP_ERROR;                      //Report error
      Check.Type = TYPE_DISCHARGE;                   //Discharge
problem
      Check.Probe = ID;                              //Save probe
      Check.U = U_c;                                 //Save voltage
      Counter = 0;                                   //End loop
    }
    else                                             //Go for
another round
    {
      wdt_reset();                                   //Reset
watchdog
      delay(50);                                     //Wait for 50ms
    }
  }
  R_DDR = 0;                                         //Set resistor
port to input mode
  SetADCHiz();                                       //Set ADC port
to input mode
}
void PullProbe(byte Mask, byte Mode)
{
  if (Mode & FLAG_PULLUP) R_PORT |= Mask;            //Pull-up
  else R_PORT &= ~Mask;                              //Pull-down
  R_DDR |= Mask;                                     //Enable
pulling
  if (Mode & FLAG_1MS) delay(1);                     //Wait 1ms
  else delay(10);                                    //Wait 10ms
```

```
  R_DDR &= ~Mask;                                      //Set to HiZ
mode
  R_PORT &= ~Mask;                                     //Set 0
}
unsigned long RescaleValue(unsigned long Value, signed char
Scale, signed char NewScale)
{
  unsigned long              NewValue;
  NewValue = Value;                                    //Take old
value
  while (Scale != NewScale)                            //Processing
loop
  {
    if (NewScale > Scale)                              //Upscale
    {
      NewValue /= 10;
      Scale++;
    }
    else                                               //Downscale
    {
      NewValue *= 10;
      Scale--;
    }
  }
  return NewValue;
}
unsigned int GetFactor(unsigned int U_in, byte ID)
{
  unsigned int               Factor;             //Return value
  unsigned int               U_Diff;             //Voltage
difference to table start
  unsigned int               Fact1, Fact2;       //Table entries
  unsigned int               TabStart;           //Table start
voltage
  unsigned int               TabStep;            //Table step
voltage
  unsigned int               TabIndex;           //Table entries
(-2)
  unsigned int               *Table;
  byte                       Index;              //Table index
  byte                       Diff;               //Difference to
next entry
  if (ID == TABLE_SMALL_CAP)
  {
    TabStart = 1000;                                   //Table starts
at 1000mV
```

```
    TabStep = 50;                                //50mV steps
between entries
    TabIndex = 7;                                //Entries in
table - 2
    Table = (unsigned int *)&SmallCap_table[0];  //Pointer to
table start
  }
  else if (ID == TABLE_LARGE_CAP)
  {
    TabStart = 300;                              //Table starts
at 1000mV
    TabStep = 25;                                //25mV steps
between entries
    TabIndex = 42;                               //Entries in
table - 2
    Table = (unsigned int *)&LargeCap_table[0];  //Pointer to
table start
  }
  else if (ID == TABLE_INDUCTOR)
  {
    TabStart = 200;                              //Table starts
at 200
    TabStep = 25;                                //Steps between
entries
    TabIndex = 30;                               //Entries in
table - 2
    Table = (unsigned int *)&Inductor_table[0];  //Pointer to
table start
  }
  else
  {
    return 0;
  }
  if (U_in >= TabStart) U_Diff = U_in - TabStart;
  else U_Diff = 0;
  Index = U_Diff / TabStep;                      //Index
(position in table)
  Diff = U_Diff % TabStep;                       //Difference to
index
  Diff = TabStep - Diff;                         //Difference to
next entry
  if (Index > TabIndex) Index = TabIndex;
  Table += Index;                                //Advance to
index
  Fact1 = *(Table);
  Table++;                                       //Next entry
  Fact2 = *(Table);
```

```
  Factor = Fact1 - Fact2;
  Factor *= Diff;
  Factor += TabStep / 2;
  Factor /= TabStep;
  Factor += Fact2;
  return Factor;
}
void CheckProbes(byte Probe1, byte Probe2, byte Probe3)
{
  byte                        Flag;              //Temporary
value
  unsigned int                U_Rl;              //Voltage
across Rl (load)
  unsigned int                U_1;               //Voltage #1
  if (Check.Found == COMP_ERROR) return;         //Skip check on
any error
  wdt_reset();                                   //Reset
watchdog
  UpdateProbes(Probe1, Probe2, Probe3);          //Update
bitmasks
  R_PORT = 0;                                    //Set resistor
port to Gnd
  R_DDR = Probes.Rl_2;                           //Pull down
probe-2 via Rl
  ADC_DDR = Probes.ADC_1;                        //Set probe-1
to output
  ADC_PORT = Probes.ADC_1;                       //Pull-up
probe-1 directly
  PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLDOWN);
  U_Rl = ReadU_5ms(Probes.Pin_2);                //Get voltage
at Rl
  if (U_Rl >= 977)                               // > 1.4mA
  {
    PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLUP);
    U_Rl = ReadU_5ms(Probes.Pin_2);              //Get voltage
at Rl
  }
  if (U_Rl > 490)                                // >
700Aâ€šÃ‚lA (was 92mV/130Aâ€šÃ‚lA)
  {
    CheckDepletionModeFET(U_Rl);
  }
  if (U_Rl < 977)                                //Load current
< 1.4mA
  {
    if (Check.Done == 0)                         //Not sure yet
    {
```

```
      R_DDR = Probes.Rl_2;                            //Enable Rl for
probe-2
      R_PORT = 0;                                     //Pull down
collector via Rl
      ADC_DDR = Probes.ADC_1;                         //Set probe 1
to output
      ADC_PORT = Probes.ADC_1;                        //Pull up
emitter directly
      delay(5);
      R_DDR = Probes.Rl_2 | Probes.Rl_3;              //Pull down
base via Rl
      U_1 = ReadU_5ms(Probe2);                        //Get voltage
at collector
      if (U_1 > 3422)                                 //Detected
current > 4.8mA
      {
        CheckBJTorEnhModeMOSFET(TYPE_PNP, U_Rl);
      }
    }
    if (Check.Done == 0)                              //Not sure yet
    {
      ADC_DDR = Probes.ADC_2;                         //Set probe-2
to output mode
      SetADCLow();                                    //Pull down
probe-2 directly
      R_DDR = Probes.Rl_1 | Probes.Rl_3;              //Select Rl for
probe-1 & Rl for probe-3
      R_PORT = Probes.Rl_1 | Probes.Rl_3;             //Pull up
collector & base via Rl
      U_1 = ReadU_5ms(Probe1);                        //Get voltage
at collector
      if (U_1 < 1600)                                 //Detected
current > 4.8mA
      {
        Flag = CheckThyristorTriac();
        if (Flag == 0)                                //No thyristor
or triac
        {
          CheckBJTorEnhModeMOSFET(TYPE_NPN, U_Rl);
        }
      }
    }
  }
  else                                                //Load current
> 1.4mA
  {
    CheckDiode();
```

```
  }
  if ((Check.Found == COMP_NONE) ||
      (Check.Found == COMP_RESISTOR))
  {
    CheckResistor();
  }
  else
  {
    if ((Check.Found == COMP_FET) && (Check.Type & TYPE_MOSFET))
      VerifyMOSFET();
  }
  SetADCHiz();                                        //Set ADC port
to HiZ mode
  SetADCLow();                                        //Set ADC port
low
  R_DDR = 0;                                          //Set resistor
port to HiZ mode
  R_PORT = 0;                                         //Set resistor
port low
}
unsigned int ReadU(byte Probe)
{
  unsigned int              U;                        //Return value
(mV)
  byte                      Counter;                  //Loop counter
  unsigned long             Value;                    //ADC value
  boolean                   cycle;
  Probe |= (1 << REFS0);                              //Use internal
reference anyway
  do {
    cycle = false;
    ADMUX = Probe;                                      //Set input
channel and U reference
    Counter = Probe & (1 << REFS1);                     //Get REFS1
bit flag
    if (Counter != Config.RefFlag)
    {
      waitus(100);                                    //Time for
voltage stabilization
      ADCSRA |= (1 << ADSC);                          //Start
conversion
      while (ADCSRA & (1 << ADSC));                   //Wait until
conversion is done
      Config.RefFlag = Counter;                       //Update flag
    }
    Value = 0UL;                                      //Reset
sampling variable
```

```
    Counter = 0;                                        //Reset counter
    while (Counter < Config.Samples)                    //Take samples
    {
      ADCSRA |= (1 << ADSC);                            //Start
conversion
      while (ADCSRA & (1 << ADSC));                     //Wait until
conversion is done
      Value += ADCW;                                    //Add ADC
reading
      if (Counter == 4)
      {
        if (((unsigned int)Value < 1024) && !(Probe & (1 <<
REFS1)) && (Config.AutoScale == 1))
        {
          Probe |= (1 << REFS1);                        //Select
internal bandgap reference
          cycle = true;                                 //Re-run
sampling
          break;
        }
      }
      Counter++;                                        //One less to
do
    }
  } while (cycle);
  if (Probe & (1 << REFS1)) U = Config.U_Bandgap;//Bandgap
reference
  else U = UREF_VCC;                                    //Vcc reference
  Value *= U;                                           //ADC readings
* U_ref
  Value /= 1024;                                        // / 1024 for
10bit ADC
  Value /= Config.Samples;
  U = (unsigned int)Value;
  return U;
}
unsigned int ReadU_5ms(byte Probe)
{
  delay(5);                                            //Wait 5ms
  return (ReadU(Probe));
}
unsigned int ReadU_20ms(byte Probe)
{
  delay(20);                                            //Wait 20ms
  return (ReadU(Probe));
}
void waitus(byte microsec) {
```

```
  delayMicroseconds(microsec);
}
unsigned long Get_hFE_C(byte Type)
{
  unsigned long              hFE;                //Return value
  unsigned int               U_R_e;              //Voltage
across emitter resistor
  unsigned int               U_R_b;              //Voltage
across base resistor
  unsigned int               Ri;                 //Internal
resistance of Aâ€šÃ,lC
  if (Type == TYPE_NPN)                          //NPN
  {
    ADC_DDR = Probes.ADC_1;                      //Set probe 1
to output
    ADC_PORT = Probes.ADC_1;                     //Pull up
collector directly
    R_DDR = Probes.Rl_2 | Probes.Rl_3;           //Select Rl for
probe-2 & Rl for probe-3
    R_PORT = Probes.Rl_3;                        //Pull up base
via Rl
    U_R_e = ReadU_5ms(Probes.Pin_2);             //U_R_e = U_e
    U_R_b = UREF_VCC - ReadU(Probes.Pin_3);      //U_R_b = Vcc -
U_b
  }
  else                                           //PNP
  {
    SetADCLow();                                 //Set ADC port
low
    ADC_DDR = Probes.ADC_2;                      //Pull down
collector directly
    R_PORT = Probes.Rl_1;                        //Pull up
emitter via Rl
    R_DDR = Probes.Rl_1 | Probes.Rl_3;           //Pull down
base via Rl
    U_R_e = UREF_VCC - ReadU_5ms(Probes.Pin_1);  //U_R_e = Vcc -
U_e
    U_R_b = ReadU(Probes.Pin_3);                 //U_R_b = U_b
  }
  if (U_R_b < 10)                                //I_b <
14Aâ€šÃ,lA -> Darlington
  {
    if (Type == TYPE_NPN)                        //NPN
    {
      R_DDR = Probes.Rl_2 | Probes.Rh_3;         //Select Rl for
probe-2 & Rh for probe-3
```

```
      R_PORT = Probes.Rh_3;                      //Pull up base
via Rh
      U_R_e = ReadU_5ms(Probes.Pin_2);           //U_R_e = U_e
      U_R_b = UREF_VCC - ReadU(Probes.Pin_3);    //U_R_b = Vcc -
U_b
      Ri = Config.RiL;                           //Get internal
resistor
    }
    else                                         //PNP
    {
      R_DDR = Probes.Rl_1 | Probes.Rh_3;         //Pull down
base via Rh
      U_R_e = UREF_VCC - ReadU_5ms(Probes.Pin_1);//U_R_e = Vcc -
U_e
      U_R_b = ReadU(Probes.Pin_3);               //U_R_b = U_b
      Ri = Config.RiH;                           //Get internal
resistor
    }
    if (U_R_b < 1) U_R_b = 1;                    //Prevent
division by zero
    hFE =  U_R_e * R_HIGH;                       //U_R_e * R_b
    hFE /= U_R_b;                                // / U_R_b
    hFE *= 10;                                   //Upscale to
0.1
    hFE /= (R_LOW * 10) + Ri;                    // / R_e in 0.1
Ohm
  }
  else                                           //I_b >
14Aâ€šÃ‚lA -> standard
  {
    hFE = (unsigned long)((U_R_e - U_R_b) / U_R_b);
  }
  return hFE;
}
void GetGateThreshold(byte Type)
{
  unsigned long            Uth = 0;              //Gate
threshold voltage
  byte                     Drain_Rl;             //Rl bitmask
for drain
  byte                     Drain_ADC;            //ADC bitmask
for drain
  byte                     PullMode;
  byte                     Counter;              //Loop counter
  if (Type & TYPE_N_CHANNEL)                     //n-channel
  {
    Drain_Rl =  Probes.Rl_1;
```

```
    Drain_ADC = Probes.ADC_1;
    PullMode = FLAG_10MS | FLAG_PULLDOWN;
  }
  else                                                  //p-channel
  {
    Drain_Rl =  Probes.Rl_2;
    Drain_ADC = Probes.ADC_2;
    PullMode = FLAG_10MS | FLAG_PULLUP;
  }
  Drain_ADC &= 0b00000111;                              //drain
  ADMUX = Probes.Pin_3 | (1 << REFS0);                  //Select probe-
3 for ADC input
  for (Counter = 0; Counter < 10; Counter++)
  {
    wdt_reset();                                        //Reset
watchdog
    PullProbe(Probes.Rl_3, PullMode);
    R_DDR = Drain_Rl | Probes.Rh_3;
    if (Type & TYPE_N_CHANNEL)                          //n-channel
    {
      while (ADC_PIN & Drain_ADC);
    }
    else                                                //p-channel
    {
      while (!(ADC_PIN & Drain_ADC));
    }
    R_DDR = Drain_Rl;                                   //Set probe-3
to HiZ mode
    ADCSRA |= (1 << ADSC);                              //Start ADC
conversion
    while (ADCSRA & (1 << ADSC));                       //Wait until
conversion is done
    if (Type & TYPE_N_CHANNEL)                          //n-channel
    {
      Uth += ADCW;                                      //U_g =
U_measued
    }
    else                                                //p-channel
    {
      Uth += (1023 - ADCW);                             //U_g = Vcc -
U_measured
    }
  }
  Uth /= 10;                                            //Average of 10
samples
  Uth *= UREF_VCC;                                      //Convert to
voltage
```

```
  Uth /= 1024;                                          //Using 10 bit
resolution
  FET.V_th = (unsigned int)Uth;
}
unsigned int GetLeakageCurrent(void)
{
  unsigned int                I_leak = 0;             //Return value
  unsigned int                U_Rl;                   //Voltage at Rl
  unsigned int                R_Shunt;                //Shunt
resistor
  uint32_t                    Value;
  R_PORT = 0;                                         //Set resistor
port to Gnd
  R_DDR = Probes.Rl_2;                                //Pull down
probe-2 via Rl
  ADC_DDR = Probes.ADC_1;                             //Set probe-1
to output
  ADC_PORT = Probes.ADC_1;                            //Pull-up
probe-1 directly
  U_Rl = ReadU_5ms(Probes.Pin_2);                     //Get voltage
at Rl
  R_Shunt = Config.RiL + (R_LOW * 10);                //Consider
internal resistance of MCU (0.1 Ohms)
  R_Shunt += 5;                                       //For rounding
  R_Shunt /= 10;                                      //Scale to Ohms
  Value = U_Rl * 100000;                              //Scale to 10nV
  Value /= R_Shunt;                                   //in 10nA
  Value += 55;                                        //For rounding
  Value /= 100;                                       //Scale to
Aâ€šÃ‚lA
  I_leak = Value;
  SetADCHiz();                                        //Set ADC port
to HiZ mode
  SetADCLow();                                        //Set ADC port
low
  R_DDR = 0;                                          //Set resistor
port to HiZ mode
  R_PORT = 0;                                         //Set resistor
port low
  return I_leak;
}
void CheckDiode(void)
{
  Diode_Type                  *Diode;                 //Pointer to
diode
  unsigned int                U1_Rl;                  //Vf #1 with Rl
pull-up
```

```
  unsigned int                U1_Rh;              //Vf #1 with Rh
pull-up
  unsigned int                U1_Zero;            //Vf #1 zero
  unsigned int                U2_Rl;              //Vf #2 with Rl
pull-up
  unsigned int                U2_Rh;              //Vf #2 with Rh
pull-up
  unsigned int                U2_Zero;            //Vf #2 zero
  wdt_reset();                                    //Reset
watchdog
  DischargeProbes();                              //Try to
discharge probes
  if (Check.Found == COMP_ERROR) return;          //Skip on error
  SetADCLow();
  ADC_DDR = Probes.ADC_2;                         //Pull down
cathode directly
  U1_Zero = ReadU(Probes.Pin_1);                  //Get voltage
at anode
  R_DDR = Probes.Rh_1;                            //Enable Rh for
probe-1
  R_PORT = Probes.Rh_1;                           //Pull up anode
via Rh
  PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLUP);
  U1_Rh = ReadU_5ms(Probes.Pin_1);                //Get voltage
at anode, neglect voltage at cathode
  R_DDR = Probes.Rl_1;                            //Enable Rl for
probe-1
  R_PORT = Probes.Rl_1;                           //Pull up anode
via Rl
  PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLUP);
  U1_Rl = ReadU_5ms(Probes.Pin_1);                //Get voltage
at anode
  U1_Rl -= ReadU(Probes.Pin_2);                   //Substract
voltage at cathode
  DischargeProbes();                              //Try to
discharge probes
  if (Check.Found == COMP_ERROR) return;          //Skip on error
  SetADCLow();
  ADC_DDR = Probes.ADC_2;                         //Pull down
cathode directly
  U2_Zero = ReadU(Probes.Pin_1);                  //Get voltage
at anode
  R_DDR = Probes.Rh_1;                            //Enable Rh for
probe-1
  R_PORT = Probes.Rh_1;                           //Pull up anode
via Rh
  PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLDOWN);
```

```
  U2_Rh = ReadU_5ms(Probes.Pin_1);               //Get voltage
at anode, neglect voltage at cathode
  R_DDR = Probes.Rl_1;                           //Enable Rl for
probe-1
  R_PORT = Probes.Rl_1;                          //Pull up anode
via Rl
  PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLDOWN);
  U2_Rl = ReadU_5ms(Probes.Pin_1);               //Get voltage
at anode
  U2_Rl -= ReadU(Probes.Pin_2);                  //Substract
voltage at cathode
  R_PORT = 0;                                    //Stop pulling
up
  if (U1_Rl > U2_Rl)                             //The higher
voltage wins
  {
    U2_Rl = U1_Rl;
    U2_Rh = U1_Rh;
    U2_Zero = U1_Zero;
  }
  if (U2_Rh <= 10) return;                       //Small
resistor or very large cap
  U1_Zero = U2_Rh - U2_Zero;                     //Voltage
difference
  if ((U2_Zero > 2) && (U1_Zero < 100)) return;  //Capacitor
  if (U2_Rh < 40)                                //Resistor (<
3k)
  {
    uint32_t                  a, b;
    b = (R_HIGH * 10) / ((R_LOW * 10) + Config.RiH +
Config.RiL);
    a = b - 1;                                   //k - 1
    a /= 5;                                      // / 5V
    a *= U2_Rh;                                  // *U_Rh
    a += 1000;                                   // +1 (1000 for
mV)
    b *= 1000;                                   //For mV
    b *= U2_Rh;                                  // *U_Rh
    b /= a;                                      //U_Rl in mV
    U1_Zero = (unsigned int)b;
    U1_Rl = U1_Zero;
    U1_Rh = U1_Zero;
    U1_Zero /= 50;                               //2%
    U1_Rh += U1_Zero;                            //102%
    U1_Zero = (unsigned int)b;
    U1_Zero /= 33;                               //3%
```

```
    U1_Rl -= U1_Zero;                                //97% (for
resistors near 1k)
    if ((U2_Rl >= U1_Rl) && (U2_Rl <= U1_Rh)) return;
  }
  if ((U2_Rl > 150) && (U2_Rl < 4640))
  {
    if ((Check.Found == COMP_NONE) ||
        (Check.Found == COMP_RESISTOR))
    {
      Check.Found = COMP_DIODE;
    }
    Diode = &Diodes[Check.Diodes];
    Diode->A = Probes.Pin_1;
    Diode->C = Probes.Pin_2;
    Diode->V_f = U2_Rl;                              //Vf for high
measurement current
    Diode->V_f2 = U2_Rh;                             //Vf for low
measurement current
    Check.Diodes++;
  }
}
void VerifyMOSFET(void)
{
  byte                        Flag = 0;
  byte                        n = 0;
  byte                        Anode;
  byte                        Cathode;
  Diode_Type                  *Diode;                //Pointer to
diode
  if (Check.Type & TYPE_N_CHANNEL)                   //n-channel
  {
    Anode = FET.S;
    Cathode = FET.D;
  }
  else                                               //p-channel
  {
    Anode = FET.D;
    Cathode = FET.S;
  }
  Diode = &Diodes[0];                                //First diode
  while (n < Check.Diodes)
  {
    if ((Diode->A == Cathode) && (Diode->C == Anode))
    {
      Flag = 1;                                      //Signal match
      n = 10;                                        //End loop
    }
```

```
    n++;                                                //Next diode
    Diode++;
  }
  if (Flag == 1)                                        //Found
reversed diode
  {
    Check.Found = COMP_NONE;
    Check.Type = 0;
    Check.Done = 0;
  }
}
void CheckBJTorEnhModeMOSFET(byte BJT_Type, unsigned int U_Rl)
{
  byte                          FET_Type;               //MOSFET type
  unsigned int                  U_R_c;                  //Voltage
across collector resistor
  unsigned int                  U_R_b;                  //Voltage
across base resistor
  unsigned int                  BJT_Level;              //Voltage
threshold for BJT
  unsigned int                  FET_Level;              //Voltage
threshold for FET
  unsigned int                  I_CE0;                  //Leakage
current
  unsigned long                 hFE_C;                  //hFE (common
collector)
  unsigned long                 hFE_E;                  //hFE (common
emitter)
  if (BJT_Type == TYPE_NPN)                             //NPN / n-
channel
  {
    BJT_Level = 2557;                                   //Voltage
across base resistor (5.44Aâ€šÃ,lA)
    FET_Level = 3400;                                   //Voltage
across drain resistor (4.8mA)
    FET_Type = TYPE_N_CHANNEL;
    R_DDR = Probes.Rl_1 | Probes.Rh_3;                  //Enable Rl for
probe-1 & Rh for probe-3
    R_PORT = Probes.Rl_1 | Probes.Rh_3;                 //Pull up
collector via Rl and base via Rh
    delay(50);                                          //Wait to skip
gate charging of a FET
    U_R_c = UREF_VCC - ReadU(Probes.Pin_1);             //U_R_c = Vcc -
U_c
    U_R_b = UREF_VCC - ReadU(Probes.Pin_3);             //U_R_b = Vcc -
U_b
  }
```

```
  else                                              //PNP / p-
channel
  {
    BJT_Level = 977;                                //Voltage
across base resistor (2.1Aâ€šÃ‚lA)
    FET_Level = 2000;                               //Voltage
across drain resistor (2.8mA)
    FET_Type = TYPE_P_CHANNEL;
    R_DDR = Probes.Rl_2 | Probes.Rh_3;              //Pull down
base via Rh
    U_R_c = ReadU_5ms(Probes.Pin_2);                //U_R_c = U_c
    U_R_b = ReadU(Probes.Pin_3);                    //U_R_b = U_b
  }
  if (U_R_b > BJT_Level)                            //U_R_b exceeds
minimum level of BJT
  {
    if (Check.Found == COMP_BJT) Check.Done = 1;
    Check.Found = COMP_BJT;
    Check.Type = BJT_Type;
    I_CE0 = GetLeakageCurrent();                    //Get leakage
current (in Aâ€šÃ‚lA)
    if (U_R_c > U_Rl) U_R_c -= U_Rl;                // - U_Rl
(leakage)
    hFE_E = U_R_c * R_HIGH;                         //U_R_c * R_b
    hFE_E /= U_R_b;                                 // / U_R_b
    hFE_E *= 10;                                    //Upscale to
0.1
    if (BJT_Type == TYPE_NPN)                       //NPN
      hFE_E /= (R_LOW * 10) + Config.RiH;           // / R_c in 0.1
Ohm
    else                                            //PNP
      hFE_E /= (R_LOW * 10) + Config.RiL;           // / R_c in 0.1
Ohm
    hFE_C = Get_hFE_C(BJT_Type);
    if (hFE_C > hFE_E) hFE_E = hFE_C;
    if (hFE_E > BJT.hFE)
    {
      BJT.hFE = hFE_E;
      BJT.I_CE0 = I_CE0;
      BJT.B = Probes.Pin_3;
      if (BJT_Type == TYPE_NPN)                     //NPN
      {
        BJT.C = Probes.Pin_1;
        BJT.E = Probes.Pin_2;
      }
      else                                          //PNP
      {
```

```
        BJT.C = Probes.Pin_2;
        BJT.E = Probes.Pin_1;
      }
    }
#if 0
    SetADCHiz();                                      //Set ADC port
to HiZ mode
    R_DDR = 0;                                        //Set resistor
port to HiZ mode
    if (BJT_Type == TYPE_NPN)                         //NPN
    {
      SetADCLow();
      ADC_DDR = Probes.ADC_1;                         //Pull-down
emitter directly
      R_PORT = Probes.Rl_2 | Probes.Rh_3;             //Pull-up base
via Rh
      R_DDR = Probes.Rl_2 | Probes.Rh_3;              //Enable probe
resistors
      U_R_b = UREF_VCC - ReadU_5ms(Probes.Pin_2);//U_R_c = Vcc -
U_c
    }
    else                                              //PNP
    {
      R_PORT = 0;
      R_DDR = Probes.Rl_1 | Probes.Rh_3;              //Pull down
base via Rh
      ADC_DDR = Probes.ADC_2;
      ADC_PORT = Probes.ADC_2;                        //Pull-up
emitter directly
      U_R_b = ReadU_5ms(Probes.Pin_1);                //U_R_c = U_c
    }
    if (U_R_c > U_R_b)                                //I_c >
I_c_reversed
    {
      Check.Done = 1;
    }
#endif
  }
  else if ((U_Rl < 97) && (U_R_c > FET_Level))    //No BJT
  {
    I_CE0 = ReadU(Probes.Pin_1) - ReadU(Probes.Pin_2);
    if (I_CE0 < 250)                                  //MOSFET
    {
      Check.Found = COMP_FET;
      Check.Type = FET_Type | TYPE_ENHANCEMENT | TYPE_MOSFET;
    }
    else                                              //IGBT
```

```
    {
      Check.Found = COMP_IGBT;
      Check.Type = FET_Type | TYPE_ENHANCEMENT;
    }
    Check.Done = 1;                                      //Transistor
found
    GetGateThreshold(FET_Type);
    FET.G = Probes.Pin_3;
    if (FET_Type == TYPE_N_CHANNEL)                      //n-channel
    {
      FET.D = Probes.Pin_1;
      FET.S = Probes.Pin_2;
    }
    else                                                 //p-channel
    {
      FET.D = Probes.Pin_2;
      FET.S = Probes.Pin_1;
    }
  }
}
void CheckDepletionModeFET(unsigned int U_Rl_L)
{
  unsigned int              U_1;                         //Voltage #1
  unsigned int              U_2;                         //Voltage #2
  if (Check.Done == 0)                                   //No transistor
found yet
  {
    R_DDR = Probes.Rl_2 | Probes.Rh_3;                   //Pull down
gate via Rh
    U_1 = ReadU_20ms(Probes.Pin_2);                      //Voltage at
source
    R_PORT = Probes.Rh_3;                                //Pull up gate
via Rh
    U_2 = ReadU_20ms(Probes.Pin_2);                      //Voltage at
source
    if (U_2 > (U_1 + 488))
    {
      SetADCLow();                                       //Set ADC port
to low
      ADC_DDR = Probes.ADC_2;                            //Pull down
source directly
      R_DDR = Probes.Rl_1 | Probes.Rh_3;                 //Enable Rl for
probe-1 & Rh for probe-3
      R_PORT = Probes.Rl_1 | Probes.Rh_3;                //Pull up drain
via Rl / pull up gate via Rh
      U_2 = ReadU_20ms(Probes.Pin_3);                    //Get voltage
at gate
```

```
      if (U_2 > 3911)                                //MOSFET
      {
        Check.Type = TYPE_N_CHANNEL | TYPE_DEPLETION |
TYPE_MOSFET;
      }
      else                                           //JFET
      {
        Check.Type = TYPE_N_CHANNEL | TYPE_JFET;
      }
      Check.Found = COMP_FET;
      Check.Done = 1;
      FET.G = Probes.Pin_3;
      FET.D = Probes.Pin_1;
      FET.S = Probes.Pin_2;
    }
  }
  if (Check.Done == 0)                               //No transistor
found yet
  {
    SetADCLow();                                     //Set ADC port
to Gnd
    ADC_DDR = Probes.ADC_2;                          //Pull down
drain directly
    R_DDR = Probes.Rl_1 | Probes.Rh_3;               //Enable Rl for
probe-1 & Rh for probe-3
    R_PORT = Probes.Rl_1 | Probes.Rh_3;              //Pull up
source via Rl / pull up gate via Rh
    U_1 = ReadU_20ms(Probes.Pin_1);                  //Get voltage
at source
    R_PORT = Probes.Rl_1;                            //Pull down
gate via Rh
    U_2 = ReadU_20ms(Probes.Pin_1);                  //Get voltage
at source
    if (U_1 > (U_2 + 488))
    {
      ADC_PORT = Probes.ADC_1;                       //Pull up
source directly
      ADC_DDR = Probes.ADC_1;                        //Enable pull
up for source
      U_2 = ReadU_20ms(Probes.Pin_3);                //Get voltage
at gate
      if (U_2 < 977)                                 //MOSFET
      {
        Check.Type =  TYPE_P_CHANNEL | TYPE_DEPLETION |
TYPE_MOSFET;
      }
      else                                           //JFET
```

```
      {
        Check.Type = TYPE_P_CHANNEL | TYPE_DEPLETION |
TYPE_JFET;
      }
      Check.Found = COMP_FET;
      Check.Done = 1;
      FET.G = Probes.Pin_3;
      FET.D = Probes.Pin_2;
      FET.S = Probes.Pin_1;
    }
  }
}
byte CheckThyristorTriac(void)
{
  byte                        Flag = 0;           //Return value
  unsigned int                U_1;                //Voltage #1
  unsigned int                U_2;                //Voltage #2
  PullProbe(Probes.Rl_3, FLAG_10MS | FLAG_PULLDOWN);
  U_1 = ReadU_5ms(Probes.Pin_1);                  //Get voltage
at anode
  R_PORT = 0;                                     //Pull down
anode
  delay(5);
  R_PORT = Probes.Rl_1;                           //And pull up
anode again
  U_2 = ReadU_5ms(Probes.Pin_1);                  //Get voltage
at anode (below Rl)
  if ((U_1 < 1600) && (U_2 > 4400))
  {
    Check.Found = COMP_THYRISTOR;                 //If not
detected as a triac below
    Check.Done = 1;
    R_DDR = 0;                                    //Disable all
probe resistors
    R_PORT = 0;
    ADC_PORT = Probes.ADC_2;                      //Pull up MT1
directly
    delay(5);
    R_DDR = Probes.Rl_1;                          //Pull down MT2
via Rl
    U_1 = ReadU_5ms(Probes.Pin_1);                //Get voltage
at MT2
    if (U_1 <= 244)
    {
      R_DDR = Probes.Rl_1 | Probes.Rl_3;          //And pull down
gate via Rl
```

```
      U_1 = ReadU_5ms(Probes.Pin_3);             //Get voltage
at gate
      U_2 = ReadU(Probes.Pin_1);                 //Get voltage
at MT2
      if ((U_1 >= 977) && (U_2 >= 733))
      {
        R_DDR = Probes.Rl_1;                     //Set probe3 to
HiZ mode
        U_1 = ReadU_5ms(Probes.Pin_1);           //Get voltage
at MT2
        if (U_1 >= 733)
        {
          R_PORT = Probes.Rl_1;                  //Pull up MT2
via Rl
          delay(5);
          R_PORT = 0;                            //And pull down
MT2 via Rl
          U_1 = ReadU_5ms(Probes.Pin_1);         //Get voltage
at MT2
          if (U_1 <= 244)
          {
            Check.Found = COMP_TRIAC;
          }
        }
      }
    }
    BJT.B = Probes.Pin_3;
    BJT.C = Probes.Pin_1;
    BJT.E = Probes.Pin_2;
    Flag = 1;                                    //Signal that
we found a component
  }
  return Flag;
}
unsigned int SmallResistor(byte ZeroFlag)
{
  unsigned int               R = 0;              //Return value
  byte                       Probe;              //Probe ID
  byte                       Mode;               //Measurement
mode
  byte                       Counter;            //Sample
counter
  unsigned long              Value;              //ADC sample
value
  unsigned long              Value1 = 0;         //U_Rl temp.
value
```

```
  unsigned long               Value2 = 0;          //U_R_i_L temp.
value
  DischargeProbes();                               //Try to
discharge probes
  if (Check.Found == COMP_ERROR) return R;         //Skip on error
  SetADCLow();                                     //Set ADC port
to low
  ADC_DDR = Probes.ADC_2;                          //Pull-down
probe 2 directly
  R_PORT = 0;                                      //Low by
default
  R_DDR = Probes.Rl_1;                             //Enable
resistor
#define MODE_HIGH           0b00000001
#define MODE_LOW            0b00000010
  Mode = MODE_HIGH;
  while (Mode > 0)
  {
    if (Mode & MODE_HIGH) Probe = Probes.Pin_1;
    else Probe = Probes.Pin_2;
    wdt_reset();                                   //Reset
watchdog
    Counter = 0;                                   //Reset loop
counter
    Value = 0;                                     //Reset sample
value
    Probe |= (1 << REFS0) | (1 << REFS1);
    ADMUX = Probe;                                 //Set input
channel and U reference
    waitus(100);                                   //Time for
voltage stabilization
    ADCSRA |= (1 << ADSC);                         //Start
conversion
    while (ADCSRA & (1 << ADSC));                  //Wait until
conversion is done
    while (Counter < 100)
    {
      ADC_DDR = Probes.ADC_2;                      //Pull-down
probe-2 directly
      R_PORT = Probes.Rl_1;
      ADCSRA |= (1 << ADSC);                       //Start
conversion
      waitus(20);
      R_PORT = 0;
      ADC_DDR = Probes.ADC_2 | Probes.ADC_1;
      while (ADCSRA & (1 << ADSC));                //Wait until
conversion is done
```

```
      Value += ADCW;                                    //Add ADC
reading
      waitus(900);
      Counter++;                                        //Next round
    }
    Value *= Config.U_Bandgap;
    Value /= 1024;                                      // / 1024 for
10bit ADC
    Value /= 10;                                        //De-sample to
0.1mV
    if (Mode & MODE_HIGH)                               //Probe #1 / Rl
    {
      Mode = MODE_LOW;                                  //Switch to low
side
      Value1 = Value;                                   //Save measured
value
    }
    else                                                //Probe #2 /
R_i_L
    {
      Mode = 0;                                         //End loop
      Value2 = Value;                                   //Save measured
value
    }
  }
  if (Value1 > Value2)                                  //Sanity check
  {
    Value = 10UL * UREF_VCC;                            //in 0.1 mV
    Value -= Value1;
    Value *= 1000;                                      //Scale to
Aâ€šÃ‚µA
    Value /= ((R_LOW * 10) + Config.RiH);               //in 0.1 Ohms
    Value1 -= Value2;                                   //in 0.1 mV
    Value1 *= 10000;                                    //Scale to 0.01
Aâ€šÃ‚µV
    Value1 /= Value;                                    //in 0.01 Ohms
    R = (unsigned int)Value1;                           //Copy result
    if (ZeroFlag == 1)                                  //Auto-zero
    {
      if (R > Config.RZero) R -= Config.RZero;
      else R = 0;
    }
  }
#undef                        MODE_LOW
#undef                        MODE_HIGH
  Config.RefFlag = (1 << REFS1);                        //Set REFS1 bit
flag
```

```
  return R;
}
void CheckResistor(void)
{
  Resistor_Type               *Resistor;         //Pointer to
resistor
  unsigned long               Value1;            //Resistance of
measurement #1
  unsigned long               Value2;            //Resistance of
measurement #2
  unsigned long               Value;             //Resistance
value
  unsigned long               Temp;              //Temp. value
  signed char                 Scale;             //Resistance
scale
  signed char                 Scale2;            //Resistance
scale
  byte                        n;                 //Counter
  unsigned int                U_Rl_H;            //Voltage #1
  unsigned int                U_Ri_L;            //Voltage #2
  unsigned int                U_Rl_L;            //Voltage #3
  unsigned int                U_Ri_H;            //Voltage #4
  unsigned int                U_Rh_H;            //Voltage #5
  unsigned int                U_Rh_L;            //Voltage #6
  wdt_reset();                                   //Reset
watchdog
  SetADCLow();                                   //Set ADC port
low low
  ADC_DDR = Probes.ADC_2;                        //Pull down
probe-2 directly
  R_DDR = Probes.Rl_1;                           //Enable Rl for
probe-1
  R_PORT = Probes.Rl_1;                          //Pull up
probe-1 via Rl
  U_Ri_L = ReadU_5ms(Probes.Pin_2);              //Get voltage
at internal R of Aâ€šÃ‚lC
  U_Rl_H = ReadU(Probes.Pin_1);                  //Get voltage
at Rl pulled up
  R_PORT = 0;                                    //Set resistor
port low
  R_DDR = Probes.Rh_1;                           //Pull down
probe-1 via Rh
  U_Rh_L = ReadU_5ms(Probes.Pin_1);              //Get voltage
at probe 1
  if (U_Rh_L <= 20)
  {
```

```
    R_PORT = Probes.Rh_1;                              //Pull up
probe-1 via Rh
    U_Rh_H = ReadU_5ms(Probes.Pin_1);                  //Get voltage
at Rh pulled up
    ADC_DDR = Probes.ADC_1;                            //Set probe-1
to output
    ADC_PORT = Probes.ADC_1;                           //Pull up
probe-1 directly
    R_PORT = 0;                                        //Set resistor
port to low
    R_DDR = Probes.Rl_2;                               //Pull down
probe-2 via Rl
    U_Ri_H = ReadU_5ms(Probes.Pin_1);                  //Get voltage
at internal R of Aâ€šÃ‚lC
    U_Rl_L = ReadU(Probes.Pin_2);                      //Get voltage
at Rl pulled down
    R_DDR = Probes.Rh_2;                               //Pull down
probe-2 via Rh
    U_Rh_L = ReadU_5ms(Probes.Pin_2);                  //Get voltage
at Rh pulled down
    if ((U_Rl_H >= 4400) || (U_Rh_H <= 97))            //R >= 5.1k / R
< 9.3k
    {
      if (U_Rh_H < 4972)                               //R < 83.4M &
prevent division by zero
      {
        Value = 0;                                     //Reset value
of resistor
        if (U_Rl_L < 169)                              //R > 19.5k
        {
          if (U_Rh_L >= 38)                            //R < 61.4M &
prevent division by zero
          {
            Value1 = R_HIGH * U_Rh_H;
            Value1 /= (UREF_VCC - U_Rh_H);
            Value2 = R_HIGH * (UREF_VCC - U_Rh_L);
            Value2 /= U_Rh_L;
            if (U_Rh_H < 990)                          //Below bandgap
reference
            {
              Value = (Value1 * 4);
              Value += Value2;
              Value /= 5;
            }
            else if (U_Rh_L < 990)                     //Below bandgap
reference
            {
```

```
            Value = (Value2 * 4);
            Value += Value1;
            Value /= 5;
          }
          else                                      //Higher than
bandgap reference
          {
            Value = (Value1 + Value2) / 2;
          }
          Value += RH_OFFSET;                       //Add offset
value for Rh
          Value *= 10;                              //Upscale to
0.1 Ohms
        }
      }
      else                                          //U_Rl_L: R <=
19.5k
      {
        if ((U_Rl_H >= U_Ri_L) && (U_Ri_H >= U_Rl_L))
        {
          if (U_Rl_H == UREF_VCC) U_Rl_H = UREF_VCC - 1;
          Value1 = (R_LOW * 10) + Config.RiH;  //Rl + RiH in
0.1 Ohm
          Value1 *= (U_Rl_H - U_Ri_L);
          Value1 /= (UREF_VCC - U_Rl_H);
          Value2 = (R_LOW * 10) + Config.RiL;  //Rl + RiL in
0.1 Ohms
          Value2 *= (U_Ri_H - U_Rl_L);
          Value2 /= U_Rl_L;
          if (U_Rl_H < 990)                         //Below bandgap
reference
          {
            Value = (Value1 * 4);
            Value += Value2;
            Value /= 5;
          }
          else if (U_Rl_L < 990)                    //Below bandgap
reference
          {
            Value = (Value2 * 4);
            Value += Value1;
            Value /= 5;
          }
          else                                      //Higher than
bandgap reference
          {
            Value = (Value1 + Value2) / 2;
```

```
            }
          }
          else                                  //May happen
for very low resistances
          {
            if (U_Rl_L > 4750) Value = 1;       //U_Rl_L: R <
15 Ohms
          }
        }
        if (Value > 0)                          //Valid
resistor
        {
          Scale = -1;                           //0.1 Ohm by
default
          if (Value < 100UL)
          {
            Value2 = (unsigned long)SmallResistor(1);
            Scale2 = -2;                        //0.01 Ohm
            Value1 = Value * 2;                 //Allow 100%
tolerance
            Value1 *= 10;                       //Re-scale to
0.01 Ohms
            if (Value1 > Value2)                //Got expected
value
            {
              Value = Value2;                   //Update data
              Scale = Scale2;
            }
          }
          n = 0;
          while (n < Check.Resistors)           //Loop through
resistors
          {
            Resistor = &Resistors[n];           //Pointer to
element
            if ((Resistor->A == Probes.Pin_1) && (Resistor->B ==
Probes.Pin_2))
            {
              if (CmpValue(Value, Scale, 2, 0) == -1)
              {
                Temp = Value / 2;               //50%
              }
              else                              //>= 2 Ohm
              {
                Temp = Value / 20;              //5%
              }
              Value1 = Value - Temp;            //95% or 50%
```

```
              Value2 = Value + Temp;              //105% or 150%
              if (CmpValue(Value, Scale, 1, -1) == -1)
              {
                Value1 = 0;                       //0
                Value2 = Value * 5;               //500%
                if (Value2 == 0) Value2 = 5;      //Special case
              }
              if ((CmpValue(Resistor->Value, Resistor->Scale,
Value1, Scale) >= 0) &&
                  (CmpValue(Resistor->Value, Resistor->Scale,
Value2, Scale) <= 0))
              {
                Check.Found = COMP_RESISTOR;
                n = 100;                          //End loop and
signal match
              }
              else                                //No match
              {
                n = 200;                          //End loop and
signal mis-match
              }
            }
            else                                  //No match
            {
              n++;                                //Next one
            }
          }
          if (n != 100)                           //Not a known
resistor
          {
            if (Check.Resistors < 3)              //Prevent array
overflow
            {
              Resistor = &Resistors[Check.Resistors];
              Resistor->A = Probes.Pin_2;
              Resistor->B = Probes.Pin_1;
              Resistor->Value = Value;
              Resistor->Scale = Scale;
              Check.Resistors++;                  //Another one
found
            }
          }
        }
      }
    }
  }
}
```

```
signed char CmpValue(unsigned long Value1, signed char Scale1,
unsigned long Value2, signed char Scale2)
{
  signed char                 Flag;                //Return value
  signed char                 Len1, Len2;          //Length
  Len1 = NumberOfDigits(Value1) + Scale1;
  Len2 = NumberOfDigits(Value2) + Scale2;
  if ((Value1 == 0) || (Value2 == 0))              //Special case
  {
    Flag = 10;                                     //Perform
direct comparison
  }
  else if (Len1 > Len2)                            //More digits -
> larger
  {
    Flag = 1;
  }
  else if (Len1 == Len2)                           //Same length
  {
    Len1 -= Scale1;
    Len2 -= Scale2;
    while (Len1 > Len2)                            //Up-scale
Value #2
    {
      Value2 *= 10;
      Len2++;
    }
    while (Len2 > Len1)                            //Up-scale
Value #1
    {
      Value1 *= 10;
      Len1++;
    }
    Flag = 10;                                     //Perform
direct comparison
  }
  else                                             //Less digits -
> smaller
  {
    Flag = -1;
  }
  if (Flag == 10)                                  //Perform
direct comparison
  {
    if (Value1 > Value2) Flag = 1;
    else if (Value1 < Value2) Flag = -1;
    else Flag = 0;
```

```
  }
  return Flag;
}
byte NumberOfDigits(unsigned long Value)
{
  byte                        Counter = 1;
  while (Value >= 10)
  {
    Value /= 10;
    Counter++;
  }
  return Counter;
}
byte LargeCap(Capacitor_Type *Cap)
{
  byte                        Flag = 3;           //Return value
  byte                        TempByte;           //Temp. value
  byte                        Mode;               //Measurement
mode
  signed char                 Scale;              //Capacitance
scale
  unsigned int                TempInt;            //Temp. value
  unsigned int                Pulses;             //Number of
charging pulses
  unsigned int                U_Zero;             //Voltage
before charging
  unsigned int                U_Cap;              //Voltage of
DUT
  unsigned int                U_Drop = 0;         //Voltage drop
  unsigned long               Raw;                //Raw
capacitance value
  unsigned long               Value;              //Corrected
capacitance value
  boolean                     rerun;
  Mode = FLAG_10MS | FLAG_PULLUP;                 //Start with
large caps
  do {
    rerun = false;                                //One-Time
    DischargeProbes();                              //Try to
discharge probes
    if (Check.Found == COMP_ERROR) return 0;        //Skip on
error
    SetADCLow();                                    //Set ADC
port to low
    ADC_DDR = Probes.ADC_2;                         //Pull-down
probe 2 directly
```

```
    R_PORT = 0;                                             //Set
resistor port to low
    R_DDR = 0;                                              //Set
resistor port to HiZ
    U_Zero = ReadU(Probes.Pin_1);                           //Get zero
voltage (noise)
    Pulses = 0;
    TempByte = 1;
    while (TempByte)
    {
      Pulses++;
      PullProbe(Probes.Rl_1, Mode);                         //Charging
pulse
      U_Cap = ReadU(Probes.Pin_1);                          //Get voltage
      U_Cap -= U_Zero;                                      //Zero offset
      if ((Pulses == 126) && (U_Cap < 75)) TempByte = 0;
      if (U_Cap >= 300) TempByte = 0;
      if (Pulses == 500) TempByte = 0;
      wdt_reset();                                          //Reset
watchdog
    }
    if (U_Cap < 300)
    {
      Flag = 1;
    }
    if ((Pulses == 1) && (U_Cap > 1300))
    {
      if (Mode & FLAG_10MS)                                 //<47Aâ€šÃ,lF
      {
        Mode = FLAG_1MS | FLAG_PULLUP;                      //Set mode
(1ms charging pulses)
        rerun = true;                                       //And re-run
      }
      else
//<4.7Aâ€šÃ,lF
      {
        Flag = 2;
      }
    }
  } while (rerun);
  if (Flag == 3)
  {
    TempInt = Pulses;
    while (TempInt > 0)
    {
      TempInt--;                                          //Descrease
timeout
```

```
      U_Drop = ReadU(Probes.Pin_1);                  //Get voltage
      U_Drop -= U_Zero;                              //Zero offset
      wdt_reset();                                   //Reset
watchdog
    }
    if (U_Cap > U_Drop) U_Drop = U_Cap - U_Drop;
    else U_Drop = 0;
    if (U_Drop > 100) Flag = 0;
  }
  if (Flag == 3)
  {
    Scale = -9;                                      //Factor is
scaled to nF
    Raw = GetFactor(U_Cap + U_Drop, TABLE_LARGE_CAP);
    Raw *= Pulses;                                   //C = pulses *
factor
    if (Mode & FLAG_10MS) Raw *= 10;                 // *10 for 10ms
charging pulses
    if (Raw > UINT32_MAX / 1000)                     //Scale down if
C >4.3mF
    {
      Raw /= 1000;                                   //Scale down by
10^3
      Scale += 3;                                    //Add 3 to the
exponent
    }
    Value = Raw;                                     //Copy raw
value
    Value *= 100;
    if (Mode & FLAG_10MS) Value /= 109;              //-9% for large
cap
    else Value /= 104;                               //-4% for mid
cap
    Cap->A = Probes.Pin_2;                           //Pull-down
probe pin
    Cap->B = Probes.Pin_1;                           //Pull-up probe
pin
    Cap->Scale = Scale;                              //-9 or -6
    Cap->Raw = Raw;
    Cap->Value = Value;                              //Max.
4.3*10^6nF or 100*10^3Aâ€šÃ,lF
  }
  return Flag;
}
byte SmallCap(Capacitor_Type *Cap)
{
  byte                        Flag = 3;          //Return value
```

```
  byte                        TempByte;          //Temp. value
  signed char                 Scale;             //Capacitance
scale
  unsigned int                Ticks;             //Timer counter
  unsigned int                Ticks2;            //Timer
overflow counter
  unsigned int                U_c;               //Voltage of
capacitor
  unsigned long               Raw;               //Raw
capacitance value
  unsigned long               Value;             //Corrected
capacitance value
  Ticks2 = 0;                                    //Reset timer
overflow counter
  DischargeProbes();                             //Try to
discharge probes
  if (Check.Found == COMP_ERROR) return 0;       //Skip on error
  R_PORT = 0;                                    //Set resistor
port to low
  ADC_DDR = (1 << TP1) | (1 << TP2) | (1 << TP3);
  SetADCLow();                                   //Set ADC port
to low
  R_DDR = Probes.Rh_1;                           //Pull-down
probe-1 via Rh
  ADCSRB = (1 << ACME);                          //Use ADC
multiplexer as negative input
  ACSR =  (1 << ACBG) | (1 << ACIC);             //Use bandgap
as positive input, trigger timer1
  ADMUX = (1 << REFS0) | Probes.Pin_1;  //Switch ADC multiplexer
to probe 1 and set AREF to Vcc
  ADCSRA = ADC_CLOCK_DIV;                        //Disable ADC,
but keep clock dividers
  waitus(200);
  TCCR1A = 0;                                    //Set default
mode
  TCCR1B = 0;                                    //Set more
timer modes
  TCNT1 = 0;                                     //Set Counter1
to 0
  TIFR1 = (1 << ICF1) | (1 << OCF1B) | (1 << OCF1A) | (1 <<
TOV1);
  R_PORT = Probes.Rh_1;                          //Pull-up
probe-1 via Rh
  if (Check.Found == COMP_FET)
  {
    TempByte = (((1 << TP1) | (1 << TP2) | (1 << TP3)) & ~(1 <<
Probes.Pin_1));
```

```
  }
  else
  {
    TempByte = Probes.ADC_2;                              //Keep just
probe-1 pulled down
  }
  TCCR1B = (1 << CS10);
  ADC_DDR = TempByte;                                     //Start
charging DUT
  while (1)
  {
    TempByte = TIFR1;                                    //Get timer1
flags
    if (TempByte & (1 << ICF1)) break;
    if (TempByte & (1 << TOV1))
    {
      TIFR1 = (1 << TOV1);                               //Reset flag
      wdt_reset();                                       //Reset watchdog
      Ticks2++;                                          //Increase
overflow counter
      if (Ticks2 == (CPU_FREQ / 5000)) break;
    }
  }
  TCCR1B = 0;                                             //Stop timer
  TIFR1 = (1 << ICF1);                                    //Reset Input
Capture flag
  Ticks = ICR1;                                           //Get counter
value
  R_DDR = 0;                                              //Set resistor
port to HiZ mode
  if ((TCNT1 > Ticks) && (TempByte & (1 << TOV1)))
  {
    TIFR1 = (1 << TOV1);                                  //Reset
overflow flag
    Ticks2++;                                             //Increase
overflow counter
  }
  ADCSRA = (1 << ADEN) | (1 << ADIF) | ADC_CLOCK_DIV;
  U_c = ReadU(Probes.Pin_1);                              //Get voltage
of cap
  R_PORT = 0;                                             //Pull down
probe-2 via Rh
  R_DDR = Probes.Rh_1;                                    //Enable Rh for
probe-1 again
  if (Ticks2 >= (CPU_FREQ / 5000)) Flag = 1;
  if (Flag == 3)
  {
```

```
    Raw = (unsigned long)Ticks;                           //Set lower 16
bits
    Raw |= (unsigned long)Ticks2 << 16;                   //Set upper 16
bits
    if (Raw > 2) Raw -= 2;                                //Subtract
processing time overhead
    Scale = -12;                                          //Default
factor is for pF scale
    if (Raw > (UINT32_MAX / 1000))                        //Prevent
overflow (4.3*10^6)
    {
      Raw /= 1000;                                        //Scale down by
10^3
      Scale += 3;                                         //Add 3 to the
exponent (nF)
    }
    Raw *= GetFactor(Config.U_Bandgap + Config.CompOffset,
TABLE_SMALL_CAP);
    Raw /= (CPU_FREQ / 10000);
    Value = Raw;                                          //Take raw
value
    if (Scale == -12)                                     //pF scale
    {
      if (Value >= Config.CapZero)                        //If value is
larger than offset
      {
        Value -= Config.CapZero;                          //Substract
offset
      }
      else                                                //If value is
smaller than offset
      {
        Value = 0;                                        //Set value to
0
      }
    }
    Cap->A = Probes.Pin_2;                                //Pull-down
probe pin
    Cap->B = Probes.Pin_1;                                //Pull-up probe
pin
    Cap->Scale = Scale;                                   //-12 or -9
    Cap->Raw = Raw;
    Cap->Value = Value;                                   //Max.
5.1*10^6pF or 125*10^3nF
    if (((Scale == -12) && (Value >= 100000)) ||
        ((Scale == -9) && (Value <= 20000)))
    {
```

```
      signed int              Offset;
      signed long             TempLong;
      while (ReadU(Probes.Pin_1) > 980)
      {
      }
      R_DDR = 0;                                  //Stop
discharging
      Config.AutoScale = 0;                       //Disable auto
scaling
      Ticks = ReadU(Probes.Pin_1);                //U_c with Vcc
reference
      Config.AutoScale = 1;                       //Enable auto
scaling again
      Ticks2 = ReadU(Probes.Pin_1);               //U_c with
bandgap reference
      R_DDR = Probes.Rh_1;                        //Resume
discharging
      Offset = Ticks - Ticks2;
      if ((Offset < -4) || (Offset > 4))          //Offset too
large
      {
        TempLong = Offset;
        TempLong *= Config.U_Bandgap;             // * U_ref
        TempLong /= Ticks2;                       // / U_c
        Config.RefOffset = (signed char)TempLong;
      }
      Offset = U_c - Config.U_Bandgap;
      if ((Offset > -50) && (Offset < 50)) Config.CompOffset =
Offset;
    }
  }
  return Flag;
}
void MeasureCap(byte Probe1, byte Probe2, byte ID)
{
  byte                        TempByte;          //Temp. value
  Capacitor_Type              *Cap;              //Pointer to
cap data structure
  Diode_Type                  *Diode;            //Pointer to
diode data structure
  Resistor_Type               *Resistor;         //Pointer to
resistor data structure
  Cap = &Caps[ID];
  Cap->A = 0;
  Cap->B = 0;
  Cap->Scale = -12;                              //pF by default
  Cap->Raw = 0;
```

```
  Cap->Value = 0;
  if (Check.Found == COMP_ERROR) return;         //Skip check on
any error
  if (Check.Found == COMP_RESISTOR)
  {
    Resistor = &Resistors[0];                    //Pointer to
first resistor
    TempByte = 0;
    while (TempByte < Check.Resistors)
    {
      if (((Resistor->A == Probe1) && (Resistor->B == Probe2))
||
          ((Resistor->A == Probe2) && (Resistor->B == Probe1)))
      {
        if (CmpValue(Resistor->Value, Resistor->Scale, 10UL, 0)
== -1)
          TempByte = 99;                         //Signal low
resistance and end loop
      }
      TempByte++;                                //Next one
      Resistor++;                                //Next one
    }
    if (TempByte != 100) return;                 //Skip this one
  }
  Diode = &Diodes[0];                            //Pointer to
first diode
  for (TempByte = 0; TempByte < Check.Diodes; TempByte++)
  {
    if ((Diode->C == Probe2) &&
        (Diode->A == Probe1) &&
        (Diode->V_f < 1500))
    {
      return;
    }
    Diode++;                                     //Next one
  }
  UpdateProbes(Probe1, Probe2, 0);               //Update
bitmasks and probes
  TempByte = LargeCap(Cap);
  if (TempByte == 2)
  {
    TempByte = SmallCap(Cap);
  }
  if (Check.Diodes == 0)
  {
    if (Check.Found == COMP_RESISTOR)
    {
```

```
      if (Cap->Scale >= -6) Check.Found = COMP_CAPACITOR;
    }
    else if ((Cap->Scale > -12) || (Cap->Value >= 5UL))
    {
      Check.Found = COMP_CAPACITOR;              //Report
capacitor
    }
  }
  DischargeProbes();                             //Discharge DUT
  SetADCHiz();                                   //Set ADC port
to input
  SetADCLow();                                   //Set ADC port
low
  R_DDR = 0;                                     //Set resistor
port to input
  R_PORT = 0;                                    //Set resistor
port low
}
byte MeasureInductance(uint32_t *Time, byte Mode)
{
  byte                        Flag = 3;          //Return value
  byte                        Test;              //Test flag
  signed char                 Offset;            //Counter offet
  unsigned int                Ticks_L;           //Timer counter
  unsigned int                Ticks_H;           //Timer
overflow counter
  unsigned long               Counter;           //Counter
  if (Time == NULL) return 0;
  DischargeProbes();                             //Try to
discharge probes
  if (Check.Found == COMP_ERROR) return 0;
  R_PORT = 0;                                    //Set resistor
port to low
  SetADCLow();                                   //Set ADC port
to low
  if (Mode & MODE_LOW_CURRENT)                   //Low current
  {
    R_DDR = Probes.Rl_2;                         //Pull down
probe-2 via Rl
    ADC_DDR = Probes.ADC_1;                      //Pull down
probe-1 directly
  }
  else                                           //High current
  {
    R_DDR = 0;                                   //Disable probe
resistors
    ADC_DDR = Probes.ADC_1 | Probes.ADC_2;
```

```
  }
  ADCSRB = (1 << ACME);                                    //Use ADC
multiplexer as negative input
  ACSR =  (1 << ACBG) | (1 << ACIC);                       //Use bandgap
as positive input, trigger timer1
  ADMUX = (1 << REFS0) | Probes.Pin_2;  //Switch ADC multiplexer
to probe-2 and set AREF to Vcc
  ADCSRA = ADC_CLOCK_DIV;                                  //Disable ADC,
but keep clock dividers
  waitus(200);                                             //Allow bandgap
reference to settle
  Ticks_H = 0;                                             //Reset timer
overflow counter
  TCCR1A = 0;                                              //Set default
mode
  TCCR1B = 0;                                              //Set more
timer modes
  TCNT1 = 0;                                               //Set Counter1
to 0
  TIFR1 = (1 << ICF1) | (1 << OCF1B) | (1 << OCF1A) | (1 <<
TOV1);
  if (Mode & MODE_DELAYED_START)                           //Delayed start
  {
    Test = (CPU_FREQ / 1000000);                           //Cycles per
Aâ€šÃ‚µs
    ADC_PORT = Probes.ADC_1;                               //Pull up
probe-1 directly
    while (Test > 0)
    {
      Test--;
      asm volatile("nop\n\t"::);
    }
    TCCR1B |= (1 << CS10);                                 //Start timer
(1/1 clock divider)
  }
  else                                                     //Immediate
start
  {
    TCCR1B |= (1 << CS10);                                 //Start timer
(1/1 clock divider)

    ADC_PORT = Probes.ADC_1;                               //Pull up
probe-1 directly
  }
  while (1)
  {
```

```
    Test = TIFR1;                                 //Get timer1
flags
    if (Test & (1 << ICF1)) break;
    if (Test & (1 << TOV1))
    {
      TIFR1 = (1 << TOV1);                        //Reset flag
      wdt_reset();                                //Reset watchdog
      Ticks_H++;                                  //Increase
overflow counter
      if (Ticks_H == (CPU_FREQ / 250000))
      {
        Flag = 0;                                 //Signal timeout
        break;                                    //End loop
      }
    }
  }
  TCCR1B = 0;                                      //Stop timer
  TIFR1 = (1 << ICF1);                             //Reset Input
Capture flag
  Ticks_L = ICR1;                                  //Get counter
value
  R_DDR = Probes.Rl_2 | Probes.Rl_1;
  SetADCHiz();
  if ((TCNT1 > Ticks_L) && (Test & (1 << TOV1)))
  {
    TIFR1 = (1 << TOV1);                           //Reset
overflow flag
    Ticks_H++;                                     //Increase
overflow counter
  }
  ADCSRA = (1 << ADEN) | (1 << ADIF) | ADC_CLOCK_DIV;
  Counter = (unsigned long)Ticks_L;                //Lower 16 bits
  Counter |= (unsigned long)Ticks_H << 16;         //Upper 16 bits
  Offset = -4;                                     //Subtract
processing overhead
  if (Mode & MODE_DELAYED_START)                   //Delayed start
  {
  }
  else                                             //Immediate
start
  {
    Offset -= 1;                                   //Timer started
one cycle too early
  }
  if (Offset >= 0)                                 //Positive
offet
  {
```

```
    Counter += Offset;
  }
  else                                               //Negative
offset
  {
    Offset *= -1;                                    //Make it
positive
    if (Counter < Offset) Counter = 0;               //Prevent
underflow
    else Counter -= Offset;                          //Subtract
offset
  }
  if (Counter > 0)
  {
    Counter += (CPU_FREQ / 2000000);                 //Add half of
cycles for rounding
    Counter /= (CPU_FREQ / 1000000);                 //Divide by
frequeny and scale to Aâ€šÃ‚ls
  }
  if (Counter <= 1) Flag = 2;                        //Signal
inductance too low
  *Time = Counter;                                   //Save time
  return Flag;
}
byte MeasureInductor(Resistor_Type *Resistor)
{
  byte                        Test = 0;              //Return value
/ measurement result
  byte                        Mode;                  //Measurement
mode
  byte                        Scale;                 //Scale of
value
  unsigned int                R_total;               //Total
resistance
  unsigned int                Factor;                //Factor
  unsigned long               Value;                 //Value
  unsigned long               Time1;                 //Time #1
  unsigned long               Time2;                 //Time #2
  Inductor.Scale = 0;
  Inductor.Value = 0;
  if (Resistor == NULL) return Test;
  if (CmpValue(Resistor->Value, Resistor->Scale, 2000, 0) >= 0)
return Test;
  UpdateProbes(Resistor->A, Resistor->B, 0);         //Update probes
  Mode = MODE_LOW_CURRENT;
  Test = MeasureInductance(&Time1, Mode);
```

```
  if (Test == 2)                                    //Inductance
too low
  {
    if (CmpValue(Resistor->Value, Resistor->Scale, 40, 0) < 0)
    {
      Mode = MODE_HIGH_CURRENT;
      Test = MeasureInductance(&Time1, Mode);
    }
  }
  else if (Test == 3)                               //Valid time
  {
    Mode = MODE_LOW_CURRENT | MODE_DELAYED_START;
    Test = MeasureInductance(&Time2, Mode);
    if (Time1 > Time2) Time1 = Time2;               //Lower value
wins
  }
  if (Test != 3) Test = 0;                          //Measurements
faile
  if (Test == 3)
  {
    R_total = RescaleValue(Resistor->Value, Resistor->Scale, -
1);
    R_total += Config.RiH + Config.RiL;
    Factor = Config.RiL;
    if (Mode & MODE_LOW_CURRENT)                    //Low current
measurement mode
    {
      R_total += (R_LOW * 10);
      Factor += (R_LOW * 10);
    }
    Value = Config.U_Bandgap + Config.CompOffset;
    Value *= R_total;                               // * R_total
(in 0.1 Ohms)
    Value /= Factor;                                // / R_shunt
(in 0.1 Ohms)
    Value /= 5;                                     // / 5000mV, *
10^3
    Scale = -6;                                     //Aâ€šÃ‚lH by
default
    Value = Time1;                                  //t_stop
    Value *= Factor;                                // * factor
(Aâ€šÃ‚ls * 10^-3)
    while (Value > 100000)                          //Re-scale to
prevent overflow
    {
      Value /= 10;
      Scale++;
```

```
    }
    Value *= R_total;                                      // * R_total
(in 0.1 Ohms)
    Value /= 10000;
    Inductor.Scale = Scale;
    Inductor.Value = Value;
    Test = 1;                                              //Signal
success
  }
  return Test;
}
void lcd_clear(void)
{
#ifdef LCD_PRINT
  lcd.clear();
  delay(2);                                              //LCD needs some
time for processing
#endif
#ifdef DEBUG_PRINT
  Serial.println();
#endif
}
void lcd_line(unsigned char Line)
{
#ifdef LCD_PRINT
  lcd.setCursor(0, Line);
#endif
#ifdef DEBUG_PRINT
  Serial.println();
#endif
}
void lcd_clear_line(unsigned char Line)
{
  unsigned char               Pos;
#ifdef LCD_PRINT
  lcd_line(Line);                                        //Go to beginning
of line
  for (Pos = 0; Pos < 20; Pos++)                         //For 20 times
  {
    lcd_data(' ');                                       //Send space
  }
  lcd_line(Line);                                        //Go back to
beginning of line
#endif
#ifdef DEBUG_PRINT
  Serial.println();
#endif
```

```
}
void lcd_testpin(unsigned char Probe)
{
  lcd_data('1' + Probe);                                //Send data
}
void lcd_space(void)
{
  lcd_data(' ');
}
void lcd_string(char *String)
{
  while (*String)                                       //Loop until
trailing 0 is reached
  {
    lcd_data(*String);                                  //Send
character
    String++;                                           //Next one
  }
}
void lcd_fixed_string(const unsigned char *String)
{
  while (pgm_read_byte(String) != 0x00)
    lcd_data(pgm_read_byte(String++));                  //Send
character
}
void lcd_data(unsigned char Data)
{
#ifdef LCD_PRINT
  lcd.write(Data);                                    //Send data to
LCD
#endif
#ifdef DEBUG_PRINT
  Serial.write(Data);                                 //Send data to
Serial
#endif
}
void DisplayValue(unsigned long Value, signed char Exponent,
unsigned char Unit)
{
  unsigned char              Prefix = 0;              //Prefix
character
  byte                       Offset = 0;              //Exponent
offset to next 10^3 step
  byte                       Index;                   //Index ID
  byte                       Length;                  //String length
  while (Value >= 10000)
  {
```

```
    Value += 5;                                            //For automagic
rounding
    Value = Value / 10;                                    //Scale down by
10^1
    Exponent++;                                            //Increase
exponent by 1
  }
  if (Exponent >= -12)                                     //Prevent index
underflow
  {
    Exponent += 12;                                        //Shift
exponent to be >= 0
    Index = Exponent / 3;                                  //Number of
10^3 steps
    Offset = Exponent % 3;                                 //Offset to
lower 10^3 step
    if (Offset > 0)                                        //Dot required
    {
      Index++;                                             //Upscale
prefix
      Offset = 3 - Offset;                                 //Reverse value
(1 or 2)
    }
    if (Index <= 6) Prefix = *(&Prefix_table[Index]);
  }
  utoa((unsigned int)Value, OutBuffer, 10);
  Length = strlen(OutBuffer);
  Exponent = Length - Offset;                              //Calculate
position
  if (Exponent <= 0)                                       //We have to
prepend "0."
  {
    lcd_data('0');
    lcd_data('.');
    if (Exponent < 0) lcd_data('0');                       //Extra 0 for
factor 100
  }
  if (Offset == 0) Exponent = -1;                          //Disable dot
if not needed
  Exponent--;
  Index = 0;
  while (Index < Length)                                   //Loop through
string
  {
    lcd_data(OutBuffer[Index]);                            //Display char
    if (Index == Exponent) lcd_data('.');                  //Display dot
    Index++;                                               //Next one
```

```
  }
  if (Prefix) lcd_data(Prefix);
  if (Unit) lcd_data(Unit);
}
void DisplaySignedValue(signed long Value, signed char Exponent,
unsigned char Unit)
{
  if (Value < 0)                                        //Negative
value
  {
    lcd_data('-');                                      //Display: "-"
    Value = -Value;                                     //Make value
positive
  }
  DisplayValue((signed long)Value, Exponent, Unit);
}
void ShortCircuit(byte Mode)
{
  byte                         Run = 0;                 //Loop control
  byte                         Test;                    //Test feedback
  unsigned char                *String = NULL;          //Display
string pointer
  Test = AllProbesShorted();                            //Get current
status
  if (Mode == 0)                                        //Remove short
  {
    if (Test != 0) String = (unsigned char *)Remove_str;
  }
  else                                                  //Create short
  {
    if (Test != 3) String = (unsigned char *)Create_str;
  }
  if (String)
  {
    lcd_clear();
    lcd_fixed_string(String);                           //Display:
Remove/Create
    lcd_line(2);
    lcd_fixed_string(ShortCircuit_str);                 //Display:
short circuit!
    Run = 1;                                            //Enter loop
  }
  while (Run == 1)
  {
    Test = AllProbesShorted();                          //Check for
short circuits
    if (Mode == 0)                                      //Remove short
```

```
    {
      if (Test == 0) Run = 0;                         //End loop if
all removed
    }
    else                                              //Create short
    {
      if (Test == 3) Run = 0;                         //End loop if
all shorted
    }
    if (Run == 1)                                     //If not done
yet
      delay(50);                                      //Wait a little
bit
    else                                              //If done
      delay(200);                                     //Time to
debounce
  }
}
byte TestKey(unsigned int Timeout, byte Mode)
{
  byte                        Flag = 0;               //Return value
  byte                        Run = 1;                //Loop control
  byte                        Counter = 0;            //Time counter
  byte                        ButtonStatus = 0;       //Button Status
  if (Mode > 10)                                      //Consider
operation mode
  {
    if (Config.TesterMode == MODE_AUTOHOLD)           //Auto hold
mode
    {
      Timeout = 0;                                    //Disable
timeout
      Mode -= 10;                                     //Set cursor
mode
    }
    else                                              //Continous
mode
    {
      Mode = 0;                                       //Disable
cursor
    }
  }
  if (Mode > 0)                                       //Cursor
enabled
  {
#ifdef LCD_PRINT
    lcd.setCursor(15, 2);
```

```
    lcd.cursor();
#endif
  }
  while (Run)
  {
    //Take care about timeout
    if (Timeout > 0)                                  //Timeout
enabled
    {
#ifdef LCD_PRINT
      lcd.setCursor(15, 2);
      lcd_data(LCD_CHAR_FLAG);
#endif
      if (Timeout > 5) Timeout -= 5;                  //Decrease
timeout by 5ms
      else Run = 0;                                   //End loop on
timeout
    }
    if (!(digitalRead(TEST_BUTTON)))                  //If key is
pressed
    {
      Counter = 0;                                    //Reset counter
      delay(30);                                      //Time to
debounce
      while (Run)                                     //Detect how
long key is pressed
      {
        if (!(digitalRead(TEST_BUTTON)))              //Key still
pressed
        {
          Counter++;                                  //Increase
counter
          if (Counter > LONG_PRESS) Run = 0;          //End loop if
LONG_PRESS are reached
          else delay(10);                             //Otherweise
wait 10ms
        }
        else                                          //Key released
        {
          Run = 0;                                    //End loop
        }
      }
      if (Counter > LONG_PRESS) Flag = 2;             //Long (>=
LONG_PRESS)
      else Flag = 1;                                  //Short (<
LONG_PRESS)
    }
```

```
    else                                              //No key press
    {
      delay(5);                                       //Wait a little
bit more (5ms)
      if (Mode == 2)                                  //Blinking
cursor
      {
        Counter++;                                    //Increase
counter
        if (Counter == 100)                           //Every 500ms
(2Hz)
        {
          Counter = 0;                                //Reset counter
          if (Run == 1)                               //Turn off
          {
#ifdef LCD_PRINT
            lcd.noCursor();
#endif
            Run = 2;                                  //Toggle flag
          }
          else                                        //Turn on
          {
#ifdef LCD_PRINT
            lcd.cursor();
#endif
            Run = 1;                                  //Toggle flag
          }
        }
      }
    }
  }
  if (Mode > 0)                                       //Cursor
enabled
  {
#ifdef LCD_PRINT
    lcd.noCursor();
#endif
  }
  return Flag;
}
void ShowFail(void)
{
  lcd_fixed_string(Failed1_str);                      //Display: No
component
  lcd_line(2);                                        //Move to line
#2
```

```
  lcd_fixed_string(Failed2_str);                  //Display:
found!
  if (Check.Diodes > 0)                           //Diodes found
  {
    lcd_space();                                  //Display space
    lcd_data(Check.Diodes + '0');                 //Display
number of diodes found
    lcd_fixed_string(Diode_AC_str);               //Display: -
|>|-
  }
  RunsMissed++;                                   //Increase
counter
  RunsPassed = 0;                                 //Reset counter
}
void ShowError()
{
  if (Check.Type == TYPE_DISCHARGE)               //Discharge
failed
  {
    lcd_fixed_string(DischargeFailed_str);        //Display:
Battery?
    lcd_line(2);
    lcd_testpin(Check.Probe);
    lcd_data(':');
    lcd_space();
    DisplayValue(Check.U, -3, 'V');
  }
}
void ShowDiode_Uf(Diode_Type *Diode)
{
  if (Diode == NULL) return;
  DisplayValue(Diode->V_f, -3, 'V');
}
void ShowDiode_C(Diode_Type *Diode)
{
  if (Diode == NULL) return;
  MeasureCap(Diode->C, Diode->A, 0);
  DisplayValue(Caps[0].Value, Caps[0].Scale, 'F');
}
void ShowDiode(void)
{
  Diode_Type                  *D1;                //Pointer to
diode #1
  Diode_Type                  *D2 = NULL;         //Pointer to
diode #2
  byte                        SkipFlag = 0;       //Flag for
anti-parallel diodes
```

```
  byte                        A = 5;            //ID of common
anode
  byte                        C = 5;            //ID of common
cothode
  unsigned int                I_leak;           //Leakage
current
  D1 = &Diodes[0];                              //Pointer to
first diode
  if (Check.Diodes == 1)                        //Single diode
  {
    C = D1->C;                                  //Make anode
first pin
  }
  else if (Check.Diodes == 2)                   //Two diodes
  {
    D2 = D1;
    D2++;                                       //Pointer to
second diode
    if (D1->A == D2->A)                         //Common anode
    {
      A = D1->A;                                //Save common
anode
    }
    else if (D1->C == D2->C)                    //Common
cathode
    {
      C = D1->C;                                //Save common
cathode
    }
    else if ((D1->A == D2->C) && (D1->C == D2->A))
    {
      A = D1->A;                                //Anode and
cathode
      C = A;                                    //Are the same
      SkipFlag = 1;                             //Signal anti-
parallel diodes
    }
  }
  else if (Check.Diodes == 3)                   //Three diodes
  {
    byte                      n;
    byte                      m;
    for (n = 0; n <= 2; n++)                    //Loop for
first diode
    {
      D1 = &Diodes[n];                          //Get pointer
of first diode
```

```
      for (m = 0; m <= 2; m++)                        //Loop for
second diode
      {
        D2 = &Diodes[m];                              //Get pointer
of second diode
        if (n != m)                                   //Don't check
same diode :-)
        {
          if (D1->C == D2->A)                         //Got match
          {
            n = 5;                                    //End loops
            m = 5;
          }
        }
      }
    }
    if (n < 5) D2 = NULL;                             //No match
found
    C = D1->C;                                        //Cathode of
first diode
    A = 3;                                            //In series
mode
  }
  else                                                //To much
diodes
  {
    D1 = NULL;                                        //Don't display
any diode
    ShowFail();                                       //And tell user
    return;
  }
  if (A < 3) lcd_testpin(D1->C);                      //Common anode
  else lcd_testpin(D1->A);                            //Common
cathode
  if (A < 3) lcd_fixed_string(Diode_CA_str);          //Common anode
  else lcd_fixed_string(Diode_AC_str);                //Common
cathode
  if (A < 3) lcd_testpin(A);                          //Common anode
  else lcd_testpin(C);                                //Common
cathode
  if (D2)                                             //Second diode
  {
    if (A <= 3) lcd_fixed_string(Diode_AC_str);       //Common anode
or in series
    else lcd_fixed_string(Diode_CA_str);              //Common
cathode
    if (A == C) lcd_testpin(D2->A);                   //Anti parallel
```

```
    else if (A <= 3) lcd_testpin(D2->C);          //Common anode
or in series
    else lcd_testpin(D2->A);                      //Common
cathode
  }
  lcd_line(2);                                    //Go to line #2
  lcd_fixed_string(Vf_str);                       //Display: Vf=
  ShowDiode_Uf(D1);                               //First diode
  lcd_space();
  if (D2 == NULL)                                 //Single diode
  {
    if (D1->V_f2 < 250)
    {
      lcd_data('(');
      DisplayValue(D1->V_f2, 0, 0);
      lcd_data(')');
    }
    UpdateProbes(D1->C, D1->A, 0);                //Reverse diode
    I_leak = GetLeakageCurrent();                 //Get current
(in Aâ€šÃ‚µA)
    if (I_leak > 0)                               //Show if not
zero
    {
#ifdef BUTTON_INST
      TestKey(USER_WAIT, 11);                    //Next page
#else
      delay(3000);
#endif
      lcd_clear_line(2);                          //Only change
line #2
      lcd_fixed_string(I_R_str);                  //Display: I_R=
      DisplayValue(I_leak, -6, 'A');              //Display
current
    }
  }
  else
  {
    ShowDiode_Uf(D2);                             //Second diode
(optional)
  }
  if (SkipFlag == 0)
  {
#ifdef BUTTON_INST
    TestKey(USER_WAIT, 11);                      //Next page
#else
    delay(3000);
#endif
```

```
    lcd_clear_line(2);                               //Only change
line #2
    lcd_fixed_string(DiodeCap_str);                  //Display: C=
    ShowDiode_C(D1);                                 //First diode
    lcd_space();
    ShowDiode_C(D2);                                 //Second diode
(optional)
  }
}
void ShowBJT(void)
{
  Diode_Type                 *Diode;                 //Pointer to
diode
  unsigned char              *String;                //Display
string pointer
  byte                       Counter;                //Counter
  byte                       A_Pin;                  //Pin acting as
anode
  byte                       C_Pin;                  //Pin acting as
cathode
  unsigned int               V_BE;                   //V_BE
  signed int                 Slope;                  //Slope of
forward voltage
  if (Check.Type == TYPE_NPN)                        //NPN
    String = (unsigned char *)NPN_str;
  else                                               //PNP
    String = (unsigned char *)PNP_str;
  lcd_fixed_string(String);                          //Display: NPN
/ PNP
  if (Check.Diodes > 2)                              //Transistor is
a set of two diodes :-)
  {
    lcd_space();
    if (Check.Type == TYPE_NPN)                      //NPN
      String = (unsigned char *)Diode_AC_str;
    else                                             //PNP
      String = (unsigned char *)Diode_CA_str;
    lcd_fixed_string(String);                        //Display: -
|>|- / -|<|-
  }
  lcd_space();
  lcd_fixed_string(EBC_str);                         //Display: EBC=
  lcd_testpin(BJT.E);                                //Display
emitter pin
  lcd_testpin(BJT.B);                                //Display base
pin
```

```
  lcd_testpin(BJT.C);                              //Display
collector pin
  lcd_line(2);                                     //Move to line
#2
  lcd_fixed_string(hFE_str);                       //Display:
h_FE=
  DisplayValue(BJT.hFE, 0, 0);
  Diode = &Diodes[0];                              //Get pointer
of first diode
  Counter = 0;
  while (Counter < Check.Diodes)                   //Check all
diodes
  {
    if (Check.Type == TYPE_NPN)                    //NPN
    {
      A_Pin = BJT.B;
      C_Pin = BJT.E;
    }
    else                                           //PNP
    {
      A_Pin = BJT.E;
      C_Pin = BJT.B;
    }
    if ((Diode->A == A_Pin) && (Diode->C == C_Pin))
    {
#ifdef BUTTON_INST
      TestKey(USER_WAIT, 11);                    //Next page
#else
      delay(3000);
#endif
      lcd_clear_line(2);                           //Update line
#2
      lcd_fixed_string(V_BE_str);                  //Display:
V_BE=
      Slope = Diode->V_f - Diode->V_f2;
      Slope /= 3;
      if (BJT.hFE < 100)                           //Low hFE
      {
        V_BE = Diode->V_f;
      }
      else if (BJT.hFE < 250)                      //Mid-range hFE
      {
        V_BE = Diode->V_f - Slope;
      }
      else                                         //High hFE
      {
        V_BE = Diode->V_f2 + Slope;
```

```
      }
      DisplayValue(V_BE, -3, 'V');
      if (BJT.I_CE0 > 0)                         //Show if not
zero
      {
#ifdef BUTTON_INST
        TestKey(USER_WAIT, 11);                 //Next page
#else
        delay(3000);
#endif
        lcd_clear_line(2);                       //Only change
line #2
        lcd_fixed_string(I_CEO_str);             //Display:
I_CE0=
        DisplayValue(BJT.I_CE0, -6, 'A');        //Display
current
      }
      Counter = Check.Diodes;                    //End loop
    }
    else
    {
      Counter++;                                 //Increase
counter
      Diode++;                                   //Next one
    }
  }
}
void Show_FET_IGBT_Extras(byte Symbol)
{
  if (Check.Diodes > 0)
  {
    lcd_space();                                 //Display space
    lcd_data(Symbol);                            //Display diode
symbol
  }
#ifdef BUTTON_INST
  TestKey(USER_WAIT, 11);                       //Next page
#else
  delay(3000);
#endif
  lcd_clear();
  lcd_fixed_string(Vth_str);                     //Display: Vth
  DisplayValue(FET.V_th, -3, 'V');               //Display V_th
in mV
  lcd_line(2);
  //Display gate capacitance
  lcd_fixed_string(GateCap_str);                 //Display: Cgs=
```

```
  MeasureCap(FET.G, FET.S, 0);                       //Measure
capacitance
  DisplayValue(Caps[0].Value, Caps[0].Scale, 'F');
}
void ShowFET(void)
{
  byte                        Data;                  //Temp. data
  byte                        Symbol;                //Intrinsic
diode
  if (Check.Type & TYPE_N_CHANNEL)                   //n-channel
  {
    Data = 'N';
    Symbol = LCD_CHAR_DIODE2;                        // '|<|'
cathode pointing to drain
  }
  else                                               //p-channel
  {
    Data = 'P';
    Symbol = LCD_CHAR_DIODE1;                        // '|>|'
cathode pointing to source
  }
  if (Check.Type & TYPE_MOSFET)                      //MOSFET
    lcd_fixed_string(MOS_str);                       //Display: MOS
  else                                               //JFET
    lcd_data('J');                                   //Display: J
  lcd_fixed_string(FET_str);                         //Display: FET
  lcd_space();
  lcd_data(Data);                                    //Display: N /
P
  lcd_fixed_string(Channel_str);                     //Display: -ch
  if (Check.Type & TYPE_MOSFET)                      //MOSFET
  {
    lcd_space();
    if (Check.Type & TYPE_ENHANCEMENT)               //Enhancement
mode
      lcd_fixed_string(Enhancement_str);
    else                                             //Depletion
mode
      lcd_fixed_string(Depletion_str);
  }
  lcd_line(2);                                       //Move to line
#2
  lcd_fixed_string(GDS_str);                         //Display: GDS=
  lcd_testpin(FET.G);                                //Display gate
pin
  if (Check.Type & TYPE_JFET)
  {
```

```
    lcd_data('?');
    lcd_data('?');
  }
  else
  {
    lcd_testpin(FET.D);                               //Display drain
pin
    lcd_testpin(FET.S);                               //Display
source pin
  }
  if (Check.Type & (TYPE_ENHANCEMENT | TYPE_MOSFET))
  {
    Show_FET_IGBT_Extras(Symbol);
  }
}
void ShowIGBT(void)
{
  byte                        Data;                   //Temp. data
  byte                        Symbol;                 //Intrinsic
diode
  if (Check.Type & TYPE_N_CHANNEL)                    //n-channel
  {
    Data = 'N';
    Symbol = LCD_CHAR_DIODE2;                         // '|<|'
cathode pointing to drain
  }
  else                                                //p-channel
  {
    Data = 'P';
    Symbol = LCD_CHAR_DIODE1;                         // '|>|'
cathode pointing to source
  }
  lcd_fixed_string(IGBT_str);                         //Display: IGBT
  lcd_space();
  lcd_data(Data);                                     //Display: N /
P
  lcd_fixed_string(Channel_str);                      //Display: -ch
  lcd_space();
  if (Check.Type & TYPE_ENHANCEMENT)                  //Enhancement
mode
    lcd_fixed_string(Enhancement_str);
  else                                                //Depletion
mode
    lcd_fixed_string(Depletion_str);
  lcd_line(2);                                        //Move to line
#2
  lcd_fixed_string(GCE_str);                          //Display: GCE=
```

```
  lcd_testpin(FET.G);                              //Display gate
pin
  lcd_testpin(FET.D);                              //Display
collector pin
  lcd_testpin(FET.S);                              //Display
emitter pin
  Show_FET_IGBT_Extras(Symbol);
}
void ShowSpecial(void)
{
  if (Check.Found == COMP_THYRISTOR)
  {
    lcd_fixed_string(Thyristor_str);               //Display:
thyristor
  }
  else if (Check.Found == COMP_TRIAC)
  {
    lcd_fixed_string(Triac_str);                   //Display:
triac
  }
  lcd_line(2);                                     //Move to line
#2
  lcd_fixed_string(GAK_str);                       //Display: GAK
  lcd_testpin(BJT.B);                              //Display gate
pin
  lcd_testpin(BJT.C);                              //Display anode
pin
  lcd_testpin(BJT.E);                              //Display
cathode pin
}
void ShowResistor(void)
{
  Resistor_Type              *R1;                  //Pointer to
resistor #1
  Resistor_Type              *R2;                  //Pointer to
resistor #2
  byte                       Pin;                  //ID of common
pin
  R1 = &Resistors[0];                              //Pointer to
first resistor
  if (Check.Resistors == 1)                        //Single
resistor
  {
    R2 = NULL;                                     //Disable
second resistor
    Pin = R1->A;                                   //Make B the
first pin
```

```
  }
  else                                                      //Multiple
resistors
  {
    R2 = R1;
    R2++;                                                   //Pointer to
second resistor
    if (Check.Resistors == 3)                               //Three
resistors
    {
      Resistor_Type     *Rmax;                              //Pointer to
largest resistor
      Rmax = R1;                                            //Starting
point
      for (Pin = 1; Pin <= 2; Pin++)
      {
        if (CmpValue(R2->Value, R2->Scale, Rmax->Value, Rmax-
>Scale) == 1)
        {
          Rmax = R2;                                        //Update
largest one
        }
        R2++;                                               //Next one
      }
      if (R1 == Rmax) R1++;
      R2 = R1;
      R2++;
      if (R2 == Rmax) R2++;
    }
    if ((R1->A == R2->A) || (R1->A == R2->B)) Pin = R1->A;
    else Pin = R1->B;
  }
  if (R1->A != Pin) lcd_testpin(R1->A);
  else lcd_testpin(R1->B);
  lcd_fixed_string(Resistor_str);
  lcd_testpin(Pin);
  if (R2)                                                   //Second
resistor
  {
    lcd_fixed_string(Resistor_str);
    if (R2->A != Pin) lcd_testpin(R2->A);
    else lcd_testpin(R2->B);
  }
  lcd_line(2);
  DisplayValue(R1->Value, R1->Scale, LCD_CHAR_OMEGA);
  if (R2)                                                   //Second
resistor
```

```
  {
    lcd_space();
    DisplayValue(R2->Value, R2->Scale, LCD_CHAR_OMEGA);
  }
  else                                          //Single
resistor
  {
    if (MeasureInductor(R1) == 1)
    {
      lcd_space();
      DisplayValue(Inductor.Value, Inductor.Scale, 'H');
    }
  }
}
void ShowCapacitor(void)
{
  Capacitor_Type              *MaxCap;          //Pointer to
largest cap
  Capacitor_Type              *Cap;             //Pointer to
cap
  byte                        Counter;          //Loop counter
  MaxCap = &Caps[0];                            //Pointer to
first cap
  Cap = MaxCap;
  for (Counter = 1; Counter <= 2; Counter++)
  {
    Cap++;                                      //Next cap
    if (CmpValue(Cap->Value, Cap->Scale, MaxCap->Value, MaxCap-
>Scale) == 1)
    {
      MaxCap = Cap;
    }
  }
  lcd_testpin(MaxCap->A);                       //Display pin
#1
  lcd_fixed_string(Cap_str);                    //Display
capacitor symbol
  lcd_testpin(MaxCap->B);                       //Display pin
#2
  lcd_line(2);                                  //Move to line
#2
  DisplayValue(MaxCap->Value, MaxCap->Scale, 'F');
}
void LoadAdjust(void)
{
  if (EEPROM.read(10) == 126)
  {
```

```
    ReadEEP();
  }
  else
  {
    Config.RiL = R_MCU_LOW;
    Config.RiH = R_MCU_HIGH;
    Config.RZero = R_ZERO;
    Config.CapZero = C_ZERO;
    Config.RefOffset = UREF_OFFSET;
    Config.CompOffset = COMPARATOR_OFFSET;
    SaveEEP();
  }
}
byte SelfTest(void)
{
  byte                        Flag = 0;          //Return value
  byte                        Test = 1;          //Test counter
  byte                        Counter;           //Loop counter
  byte                        DisplayFlag;       //Display flag
  unsigned int                Val0;              //Voltage/value
  signed int                  Val1 = 0, Val2 = 0, Val3 = 0;
  ShortCircuit(1);                               //Make sure all
probes are shorted
  while (Test <= 6)
  {
    Counter = 1;
    while (Counter <= 5)
    {
      lcd_clear();
      lcd_data('T');                             //Display: T
      lcd_data('0' + Test);                      //Display test
number
      lcd_space();
      DisplayFlag = 1;                           //Display
values by default
      switch (Test)
      {
        case 1:                                  //Reference
voltage
          Val0 = ReadU(0x0e);                    //Dummy read
for bandgap stabilization
          Val0 = ReadU(0x0e);                    //Read bandgap
reference voltage
          lcd_fixed_string(URef_str);            //Display: Vref
          lcd_line(2);
          DisplayValue(Val0, -3, 'V');           //Display
voltage in mV
```

```
          DisplayFlag = 0;                              //Reset flag
          break;
        case 2:                                         //Compare Rl
resistors (probes still shorted)
          lcd_fixed_string(Rl_str);                     //Display: +Rl-
          lcd_space();
          lcd_fixed_string(ProbeComb_str);              //Display: 12
13 23
          R_PORT = 1 << (TP1 * 2);
          R_DDR = (1 << (TP1 * 2)) | (1 << (TP2 * 2));
          Val1 = ReadU_20ms(TP3);
          Val1 -= ((long)UREF_VCC * (R_MCU_LOW + R_LOW)) /
(R_MCU_LOW + R_LOW + R_LOW + R_MCU_HIGH);
          //TP1: Gnd -- Rl -- probe-3 -- probe-1 -- Rl -- Vcc
          R_DDR = (1 << (TP1 * 2)) | (1 << (TP3 * 2));
          Val2 = ReadU_20ms(TP2);
          Val2 -= ((long)UREF_VCC * (R_MCU_LOW + R_LOW)) /
(R_MCU_LOW + R_LOW + R_LOW + R_MCU_HIGH);
          R_PORT = 1 << (TP2 * 2);
          R_DDR = (1 << (TP2 * 2)) | (1 << (TP3 * 2));
          Val3 = ReadU_20ms(TP2);
          Val3 -= ((long)UREF_VCC * (R_MCU_LOW + R_LOW)) /
(R_MCU_LOW + R_LOW + R_LOW + R_MCU_HIGH);
          break;
        case 3:                                         //Compare Rh
resistors (probes still shorted)
          lcd_fixed_string(Rh_str);                     //Display: +Rh-
          lcd_space();
          lcd_fixed_string(ProbeComb_str);              //Display: 12
13 23
          R_PORT = 2 << (TP1 * 2);
          R_DDR = (2 << (TP1 * 2)) | (2 << (TP2 * 2));
          Val1 = ReadU_20ms(TP3);
          Val1 -= (UREF_VCC / 2);
          R_DDR = (2 << (TP1 * 2)) | (2 << (TP3 * 2));
          Val2 = ReadU_20ms(TP2);
          Val2 -= (UREF_VCC / 2);
          R_PORT = 2 << (TP2 * 2);
          R_DDR = (2 << (TP2 * 2)) | (2 << (TP3 * 2));
          Val3 = ReadU_20ms(TP1);
          Val3 -= (UREF_VCC / 2);
          break;
        case 4:                                         //Un-short
probes
          ShortCircuit(0);                              //Make sure
probes are not shorted
          Counter = 100;                                //Skip test
```

```
          DisplayFlag = 0;                        //Reset flag
          break;
        case 5:                                   //Rh resistors
pulled down
          lcd_fixed_string(RhLow_str);            //Display: Rh-
          R_PORT = 0;
          R_DDR = 2 << (TP1 * 2);
          Val1 = ReadU_20ms(TP1);
          R_DDR = 2 << (TP2 * 2);
          Val2 = ReadU_20ms(TP2);
          R_DDR = 2 << (TP3 * 2);
          Val3 = ReadU_20ms(TP3);
          break;
        case 6:                                   //Rh resistors
pulled up
          lcd_fixed_string(RhHigh_str);           //Display: Rh+
          R_DDR = 2 << (TP1 * 2);
          R_PORT = 2 << (TP1 * 2);
          Val1 = ReadU_20ms(TP1);
          R_DDR = 2 << (TP2 * 2);
          R_PORT = 2 << (TP2 * 2);
          Val2 = ReadU_20ms(TP2);
          R_DDR = 2 << (TP3 * 2);
          R_PORT = 2 << (TP3 * 2);
          Val3 = ReadU_20ms(TP3);
          break;
      }
      R_DDR = 0;                                  //Input mode
      R_PORT = 0;                                 //All pins low
      if (DisplayFlag)
      {
        lcd_line(2);                              //Move to line
#2
        DisplaySignedValue(Val1, 0 , 0);          //Display TP1
        lcd_space();
        DisplaySignedValue(Val2, 0 , 0);          //Display TP2
        lcd_space();
        DisplaySignedValue(Val3, 0 , 0);          //Display TP3
      }
      if (Counter < 100)                          //When we don't
skip this test
      {
#ifdef BUTTON_INST
        DisplayFlag = TestKey(1000, 0);         //Catch key press
or timeout
#else
        delay(1000);
```

```
        DisplayFlag = 0;
#endif
        if (DisplayFlag > 0)
        {
          Counter = 100;                        //Skip current
test anyway
          if (DisplayFlag == 2) Test = 100;      //Also skip
selftest
        }
      }
      Counter++;                                //Next run
    }
    Test++;                                     //Next one
  }
  Flag = 1;                                     //Signal
success
  return Flag;
}
byte SelfAdjust(void)
{
  byte                        Flag = 0;         //Return value
  byte                        Test = 1;         //Test counter
  byte                        Counter;          //Loop counter
  byte                        DisplayFlag;      //Display flag
  unsigned int                Val1 = 0, Val2 = 0, Val3 = 0;
  byte                        CapCounter = 0;   //Number of
C_Zero measurements
  unsigned int                CapSum = 0;       //Sum of C_Zero
values
  byte                        RCounter = 0;     //Number of
R_Zero measurements
  unsigned int                RSum = 0;         //Sum of R_Zero
values
  byte                        RiL_Counter = 0;  //Number of
U_RiL measurements
  unsigned int                U_RiL = 0;        //Sum of U_RiL
values
  byte                        RiH_Counter = 0;  //Number of
U_RiH measurements
  unsigned int                U_RiH = 0;        //Sum of U_RiL
values
  unsigned long               Val0;             //Temp. value
  ShortCircuit(1);                              //Make sure all
probes are shorted
  while (Test <= 5)
  {
    Counter = 1;
```

```
    while (Counter <= 5)
    {
      lcd_clear();
      lcd_data('A');                                    //Display: a
      lcd_data('0' + Test);                             //Display
number
      lcd_space();
      DisplayFlag = 1;                                  //Display
values by default
      switch (Test)
      {
        case 1:                                         //Resistance of
probe leads (probes shorted)
          lcd_fixed_string(ROffset_str);                //Display: R0
          lcd_space();
          lcd_fixed_string(ProbeComb_str);              //Display: 12
13 23
          UpdateProbes(TP2, TP1, 0);
          Val1 = SmallResistor(0);
          if (Val1 < 100)                               //Within limit
          {
            RSum += Val1;
            RCounter++;
          }
          UpdateProbes(TP3, TP1, 0);
          Val2 = SmallResistor(0);
          if (Val2 < 100)                               //Whithin limit
          {
            RSum += Val2;
            RCounter++;
          }
          UpdateProbes(TP3, TP2, 0);
          Val3 = SmallResistor(0);
          if (Val3 < 100)                               //Within limit
          {
            RSum += Val3;
            RCounter++;
          }
          break;
        case 2:                                         //Un-short
probes
          ShortCircuit(0);                              //Make sure
probes are not shorted
          Counter = 100;                                //Skip test
          DisplayFlag = 0;                              //Reset display
flag
          break;
```

```
        case 3:                                       //Internal
resistance of Aâ€šÃ‚lC in pull-down mode
          lcd_fixed_string(RiLow_str);            //Display: Ri-
          SetADCLow();
          ADC_DDR = 1 << TP1;
          R_PORT = 1 << (TP1 * 2);
          R_DDR = 1 << (TP1 * 2);
          Val1 = ReadU_5ms(TP1);
          U_RiL += Val1;
          ADC_DDR = 1 << TP2;
          R_PORT =  1 << (TP2 * 2);
          R_DDR = 1 << (TP2 * 2);
          Val2 = ReadU_5ms(TP2);
          U_RiL += Val2;
          ADC_DDR = 1 << TP3;
          R_PORT =  1 << (TP3 * 2);
          R_DDR = 1 << (TP3 * 2);
          Val3 = ReadU_5ms(TP3);
          U_RiL += Val3;
          RiL_Counter += 3;
          break;
        case 4:                                       //Internal
resistance of Aâ€šÃ‚lC in pull-up mode
          lcd_fixed_string(RiHigh_str);           //Display: Ri+
          R_PORT = 0;
          ADC_PORT = 1 << TP1;
          ADC_DDR = 1 << TP1;
          R_DDR = 1 << (TP1 * 2);
          Val1 = UREF_VCC - ReadU_5ms(TP1);
          U_RiH += Val1;
          ADC_PORT = 1 << TP2;
          ADC_DDR = 1 << TP2;
          R_DDR = 1 << (TP2 * 2);
          Val2 = UREF_VCC - ReadU_5ms(TP2);
          U_RiH += Val2;
          ADC_PORT = 1 << TP3;
          ADC_DDR = 1 << TP3;
          R_DDR = 1 << (TP3 * 2);
          Val3 = UREF_VCC - ReadU_5ms(TP3);
          U_RiH += Val3;
          RiH_Counter += 3;
          break;
        case 5:                                       //Capacitance
offset (PCB and probe leads)
          lcd_fixed_string(CapOffset_str);        //Display: C0
          lcd_space();
```

```
          lcd_fixed_string(ProbeComb_str);       //Display: 12
13 23
          MeasureCap(TP2, TP1, 0);
          Val1 = (unsigned int)Caps[0].Raw;
          if ((Caps[0].Scale == -12) && (Caps[0].Raw <= 100))
          {
            CapSum += Val1;
            CapCounter++;
          }
          MeasureCap(TP3, TP1, 1);
          Val2 = (unsigned int)Caps[1].Raw;
          if ((Caps[1].Scale == -12) && (Caps[1].Raw <= 100))
          {
            CapSum += Val2;
            CapCounter++;
          }
          MeasureCap(TP3, TP2, 2);
          Val3 = (unsigned int)Caps[2].Raw;
          if ((Caps[2].Scale == -12) && (Caps[2].Raw <= 100))
          {
            CapSum += Val3;
            CapCounter++;
          }
          break;
      }
      SetADCHiz();                                       //Input mode
      SetADCLow();                                       //All pins low
      R_DDR = 0;                                         //Input mode
      R_PORT = 0;                                        //All pins low
      if (DisplayFlag)
      {
        lcd_line(2);                                     //Move to line
#2
        DisplayValue(Val1, 0 , 0);                       //Display TP1
        lcd_space();
        DisplayValue(Val2, 0 , 0);                       //Display TP2
        lcd_space();
        DisplayValue(Val3, 0 , 0);                       //Display TP3
      }
      if (Counter < 100)                                 //When we don't
skip this test
      {
#ifdef BUTTON_INST
        DisplayFlag = TestKey(1000, 0);                //Catch key press
or timeout
#else
        delay(1000);
```

```
        DisplayFlag = 0;
#endif
        if (DisplayFlag > 0)
        {
          Counter = 100;                        //Skip current
test anyway
          if (DisplayFlag == 2) Test = 100;      //Also skip
selftest
        }
      }
      Counter++;                                //Next run
    }
    Test++;                                     //Next one
  }
  if (CapCounter == 15)
  {
    Config.CapZero = CapSum / CapCounter;
    Flag++;
  }
  if (RCounter == 15)
  {
    Config.RZero = RSum / RCounter;
    Flag++;
  }
  if ((RiL_Counter == 15) && (RiH_Counter == 15))
  {
    U_RiL /= 5;                                 //Average sum
of 3 U_RiL
    U_RiH /= 5;                                 //Average sum
of 3 U_RiH
    Val1 = (UREF_VCC * 3) - U_RiL - U_RiH;      //U_Rl * 3
    Val0 = ((unsigned long)R_LOW * 100 * U_RiL) / Val1;
    Val0 += 5;                                  //For automagic
rounding
    Val0 /= 10;                                 //Scale down to
0.1 Ohm
    if (Val0 < 250UL)                           // < 25 Ohms
    {
      Config.RiL = (unsigned int)Val0;
      Flag++;
    }
    Val0 = ((unsigned long)R_LOW * 100 * U_RiH) / Val1;
    Val0 += 5;                                  //For automagic
rounding
    Val0 /= 10;                                 //Scale down to
0.1 Ohm
    if (Val0 < 280UL)                           // < 29 Ohms
```

```
    {
      Config.RiH = (unsigned int)Val0;
      Flag++;
    }
  }
  ShowAdjust();
  if (Flag == 4) Flag = 1;                       //All
adjustments done -> success
  else Flag = 0;                                 //Signal error
  return Flag;
}
void ShowAdjust(void)
{
  lcd_clear();
  lcd_fixed_string(RiLow_str);                   //Display: Ri-
  lcd_space();
  DisplayValue(Config.RiL, -1, LCD_CHAR_OMEGA);
  lcd_line(2);
  lcd_fixed_string(RiHigh_str);                  //Display: Ri+
  lcd_space();
  DisplayValue(Config.RiH, -1, LCD_CHAR_OMEGA);
#ifdef BUTTON_INST
  TestKey(USER_WAIT, 11);                       //Let the user
read
#else
  delya(3000);
#endif
  lcd_clear();
  lcd_fixed_string(CapOffset_str);               //Display: C0
  lcd_space();
  DisplayValue(Config.CapZero, -12, 'F');        //Display C0
offset
  lcd_line(2);
  lcd_fixed_string(ROffset_str);                 //Display: R0
  lcd_space();
  DisplayValue(Config.RZero, -2, LCD_CHAR_OMEGA);//Display R0
#ifdef BUTTON_INST
  TestKey(USER_WAIT, 11);                       //Let the user
read
#else
  delay(3000);
#endif
  lcd_clear();
  lcd_fixed_string(URef_str);                    //Display: Vref
  lcd_space();
  DisplaySignedValue(Config.RefOffset, -3, 'V');
  lcd_line(2);
```

```
  lcd_fixed_string(CompOffset_str);              //Display: 
AComp
  lcd_space();
  DisplaySignedValue(Config.CompOffset, -3, 'V');
#ifdef BUTTON_INST
  TestKey(USER_WAIT, 11);                       //Let the user 
read
#else
  delay(3000);
#endif
}
void PWM_Tool(unsigned int Frequency)
{
  byte                        Test = 1;          //Loop control 
and user feedback
  byte                        Ratio;             //PWM ratio
  byte                        Prescaler;         //Timer 
prescaler
  unsigned int                Top;               //Top value
  unsigned int                Toggle;            //Counter value 
to toggle output
  uint32_t                    Value;             //Temporary 
value
  ShortCircuit(0);                               //Make sure 
probes are not shorted
  lcd_clear();
  lcd_fixed_string(PWM_str);                     //Display: PWM
  lcd_data(' ');
  DisplayValue(Frequency, 0, 'H');               //Display 
frequency
  lcd_data('z');                                 //Make it Hz :-
)
  R_PORT = 0;                                    //Make probe #1 
and #3 ground
  R_DDR = (1 << (TP1 * 2)) | (1 << (TP2 * 2)) | (1 << (TP3 * 
2));
  Value = CPU_FREQ / 2;
  Value /= Frequency;
  if (Value > 2000000)                           //Low frequency
  {
    Value /= 256;
    Prescaler = (1 << CS12);                     //256
  }
  else if (Value > 16000)                        //Mid-range 
frequency
  {
    Value /= 64;
```

```
    Prescaler = (1 << CS11) | (1 << CS10);       //64
  }
  else                                           //High
frequency
  {
    Prescaler = (1 << CS10);                     //1
  }
  Top = (unsigned int)Value;
  Ratio = 50;                                    //Default ratio
is 50%
  Toggle = (Top / 2) - 1;                        //Compare value
for 50%
  Config.SleepMode = SLEEP_MODE_IDLE;            //Change sleep
mode to Idle
  TCCR1B = 0;                                    //Disable timer
  TCCR1A = (1 << WGM11) | (1 << WGM10) | (1 << COM1B1);
  TCCR1B = (1 << WGM13);
  TCNT1 = 0;                                     //Set counter
to 0
  OCR1A = Top - 1;                               //Set top value
(-1)
  OCR1B = Toggle;                                //Set value to
compare with
  TCCR1B = (1 << WGM13) | Prescaler;
  while (Test > 0)
  {
    lcd_clear_line(2);
    DisplayValue(Ratio, 0, '%');                 //Show ratio in
%
    delay(500);                                  //Smooth UI
#ifdef BUTTON_INST
    Test = TestKey(0, 0);                       //Wait for user
feedback
#else
    delay(3000);
    Test = 1;
#endif
    if (Test == 1)                               //Short key
press
    {
      delay(50);                                 //Debounce
button a little bit longer
#ifdef BUTTON_INST
      Prescaler = TestKey(200, 0);              //Check for
second key press
#else
      delay(3000);
```

```
      Prescaler = 0;
#endif
      if (Prescaler > 0)                                 //Second key
press
      {
        Test = 0;                                        //End loop
      }
      else                                               //Single key
press
      {
        if (Ratio <= 95) Ratio += 5;                     // +5% and
limit to 100%
      }
    }
    else                                                 //Long key
press
    {
      if (Ratio >= 5) Ratio -= 5;                        // -5% and
limit to 0%
    }
    Value = (uint32_t)Top * Ratio;
    Value /= 100;
    Toggle = (unsigned int)Value;
    Toggle--;
    OCR1B = Toggle;                                      //Update
compare value
  }
  TCCR1B = 0;                                            //Disable timer
  TCCR1A = 0;                                            //Reset flags
(also frees PB2)
  R_DDR = 0;                                             //Set HiZ mode
  Config.SleepMode = SLEEP_MODE_PWR_SAVE;                //Reset sleep
mode to default
}
void SaveEEP(void)
{
  EEPROMWriteInt(1, Config.RiL);
  EEPROMWriteInt(3, Config.RiH);
  EEPROMWriteInt(5, Config.RZero);
  EEPROM.write(7, Config.CapZero);
  delay(10);
  EEPROM.write(8, Config.RefOffset);
  delay(10);
  EEPROM.write(9, Config.CompOffset);
  delay(10);
  EEPROM.write(10, 126);                                 //Saved :-)
  delay(10);
```

```
}
void ReadEEP(void)
{
  Config.RiL = EEPROMReadInt(1);
  Config.RiH = EEPROMReadInt(3);
  Config.RZero = EEPROMReadInt(5);
  Config.CapZero = EEPROM.read(7);
  Config.RefOffset = EEPROM.read(8);
  Config.CompOffset = EEPROM.read(9);
}
unsigned int EEPROMReadInt(int p_address)
{
  byte                        lowByte = EEPROM.read(p_address);
  byte                        highByte = EEPROM.read(p_address +
1);
  return ((lowByte << 0) & 0xFF) + ((highByte << 8) & 0xFF00);
}
void EEPROMWriteInt(int p_address, int p_value)
{
  byte                        lowByte = ((p_value >> 0) & 0xFF);
  byte                        highByte = ((p_value >> 8) &
0xFF);
  EEPROM.write(p_address, lowByte);
  delay(10);
  EEPROM.write(p_address + 1, highByte);
  delay(10);
}
void MainMenu(void)
{
#ifdef DEBUG_PRINT
  unsigned int              Frequency;          //Frequency for
PWM Tool
  boolean                   doexit = false;     //Exit Menu Flag
  do
  {
    boolean                 cmdexec = false;    //CMD Exec Flag
    Serial.println();
    Serial.println(X("** MAIN MENU"));
    Serial.println();
    Serial.println(X("  1) PWM"));
    Serial.println(X("  2) SelfTest"));
    Serial.println(X("  3) Adjust"));
    Serial.println(X("  4) Save"));
    Serial.println(X("  5) Show"));
    Serial.println(X("  6) Default"));
    Serial.print(X("  0) Exit       >"));
    do
```

```
    {
      if (Serial.available() > 0)
      {
        char inChar = Serial.read();
        Serial.println(inChar);
        switch ((byte)inChar - 48)
        {
          case 1:                              //Pwm Menu
            Serial.println();
            Frequency = selFreq();
            Serial.println();
            Serial.println(X("Info:"));
            Serial.println(X("  Short  Press +"));
            Serial.println(X("  Long   Press -"));
            Serial.println(X("  Double Press Exit"));
            PWM_Tool(Frequency);
            Serial.println();
            cmdexec = true;
            break;
          case 2:                              //Selftest
            SelfTest();
            Serial.println();
            cmdexec = true;
            break;
          case 3:                              //Adjust
            SelfAdjust();
            Serial.println();
            cmdexec = true;
            break;
          case 4:                              //Save
            SaveEEP();
            Serial.println();
            cmdexec = true;
          case 5:                              //Show
            ShowAdjust();
            Serial.println();
            cmdexec = true;
            break;
          case 6:                               //Default
Parameters
            DefaultPar();
            Serial.println();
            cmdexec = true;
            break;
          case 0:                              //Exit
            cmdexec = true;
            doexit = true;
```

```
          Serial.println();
          Serial.println(X("Done. Exit"));
          return;
        default:
          Serial.print(X("                >"));
          cmdexec = false;
          doexit = false;
      }
    }
  } while (cmdexec == false);
 } while (doexit == false);
#else
 delay(800);
 LcdMenu();
#endif
}
unsigned int selFreq(void)
{
 boolean                cmdexec = false;   //CMD Exec Flag
 Serial.println(X("Select Frequency:"));
 for (int f; f < 8; f++)
 {
   Serial.print(X("  "));
   Serial.print(f + 1);
   Serial.print(X(") "));
   DisplayValue(PWM_Freq_table[f], 0, 0);
   Serial.println(X("Hz"));
 }
 Serial.print(X("                >"));
 do
 {
   if (Serial.available() > 0)
   {
     char inChar = Serial.read();
     byte selNum = (byte)inChar - 48;
     if (selNum > 0 && selNum < 9)
     {
       Serial.println(inChar);
       cmdexec = true;
       return PWM_Freq_table[selNum - 1];
     }
     else
     {
       Serial.println(X("                >"));
       cmdexec = false;
     }
   }
```

```
  } while (cmdexec == false);
  return 100;
}
void LcdMenu(void)
{
  byte                        Flag = 1;          //Control flag
  byte                        Selected;          //ID of
selected item
  byte                        ID;                //ID of
selected item
  unsigned int                Frequency;         //PWM frequency
  void                        *Menu[6];
  //Setup menu
  Menu[0] = (void *)PWM_str;
  Menu[1] = (void *)Selftest_str;
  Menu[2] = (void *)Adjustment_str;
  Menu[3] = (void *)Save_str;
  Menu[4] = (void *)Show_str;
  Menu[5] = (void *)Default_str;
  lcd_clear();
  lcd_fixed_string(Select_str);
  Selected = MenuTool(6, 1, Menu, NULL);
  switch (Selected)
  {
    case 0:                                      //PWM tool
      lcd_clear();
      lcd_fixed_string(PWM_str);
      ID = MenuTool(8, 2, (void **)PWM_Freq_table, (unsigned
char *)Hertz_str);
      Frequency = PWM_Freq_table[ID];
      PWM_Tool(Frequency);                       //And run PWM
tool
      break;
    case 1:                                      //Self test
      Flag = SelfTest();
      break;
    case 2:                                      //Self
adjustment
      Flag = SelfAdjust();
      break;
    case 3:                                      //Save self
adjument values
      SaveEEP();
      break;
    case 4:                                      //Show self
adjument values
      ShowAdjust();
```

```
      break;
  }
  lcd_clear();
  if (Flag == 1)
    lcd_fixed_string(Done_str);                  //Display:
done!
  else
    lcd_fixed_string(Error_str);                 //Display:
error!
}
byte MenuTool(byte Items, byte Type, void *Menu[], unsigned char
*Unit)
{
  byte                        Selected = 0;      //Return value
/ ID of selected item
  byte                        Run = 1;           //Loop control
flag
  byte                        n;                 //Temp value
  void                        *Address;          //Address of
menu element
  unsigned int                Value;             //Temp. value
  Items--;                                       //To match
array counter
  lcd_data(':');                                 //Whatever:
  while (Run)
  {
    lcd_clear_line(2);
    Address = &Menu[Selected];                   //Get address
of element
    if (Type == 1)                               //Fixed string
    {
      lcd_fixed_string(*(unsigned char **)Address);
    }
    else
    {
      Value = PWM_Freq_table[Selected];
      DisplayValue(Value, 0, 0);
    }
    if (Unit)                                    //Optional
fixed string
    {
      lcd_fixed_string(Unit);
    }
    delay(100);                                  //Smooth UI
#ifdef LCD_PRINT
    lcd.setCursor(15, 2);
#endif
```

```
    if (Selected < Items) n = 126;                      //Another item
follows
    else n = 127;                                       //Last item
    lcd_data(n);
    n = TestKey(0, 0);                                  //Wait for
testkey
    if (n == 1)                                         //Short key
press: moves to next item
    {
      Selected++;                                       //Move to next
item
      if (Selected > Items)
      {
        Selected = 0;                                   //Roll over to
first one
      }
    }
    else if (n == 2)                                    //Long key
press: select current item
    {
      Run = 0;
    }
  }
  lcd_clear();
  delay(500);                                           //Smooth UI
  return Selected;
}
void DefaultPar(void)
{
  Config.RiL = R_MCU_LOW;
  Config.RiH = R_MCU_HIGH;
  Config.RZero = R_ZERO;
  Config.CapZero = C_ZERO;
  Config.RefOffset = UREF_OFFSET;
  Config.CompOffset = COMPARATOR_OFFSET;
  SaveEEP();
}
```

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(A5, A4, A3, A2, A1, A0);
const int test = 8;
const int pin1 = 2;
const int pin2 = 3;
const int pin3 = 4;
const int test_ok_led = 6;
const int test_fail_led = 5;
int result[4];
int logic_gate_result = 0;
#define   AND_gate   1
#define   OR_gate   2
#define   NAND_gate   3
#define   NOR_gate   4
#define   EX_OR_gate   5
#define   EX_NOR_gate   6
#define   INV_gate  7
void setup()
{
  pinMode(test_ok_led, OUTPUT);
  pinMode(test_fail_led, OUTPUT);
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Component test");
  lcd.setCursor(0, 1);
  lcd.print("Part2");
  pinMode(test, INPUT_PULLUP);
  delay(1000);

}

void loop()
{
  if (digitalRead(test) == 0) //button pressed
  {
    //begin testing
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Testing...");
    delay(2000);
    find_gate();
    if (logic_gate_result > 0)
```

```
    {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Result");
      lcd.setCursor(0, 1);
      lcd.print("Found:");
      digitalWrite(test_fail_led, LOW);
      digitalWrite(test_ok_led, HIGH);
      if (logic_gate_result == AND_gate)
      {
        lcd.print(" AND Gate   ");
        delay(2000);
      }
      else if (logic_gate_result == NAND_gate)
      {
        lcd.print(" NAND Gate   ");
        delay(2000);
      }
      else if (logic_gate_result == OR_gate)
      {
        lcd.print(" OR Gate   ");
        delay(2000);
      }
      else if (logic_gate_result == EX_OR_gate)
      {
        lcd.print(" EXOR Gate   ");
        delay(2000);
      }
      else if (logic_gate_result == EX_NOR_gate)
      {
        lcd.print(" EXNOR Gate   ");
        delay(2000);
      }
      else if (logic_gate_result == INV_gate)
      {
        lcd.print(" NOT Gate   ");
        delay(2000);
      }

    }
    else
    {
      lcd.setCursor(0, 1);
      lcd.print("Found Nothing!   ");
      digitalWrite(test_fail_led, HIGH);
      digitalWrite(test_ok_led, LOW);
      delay(2000);
```

```
    }
  }




}

void find_gate()
{
  look_for_INV_gate();
  disp_result();
  if (result[0] == 1 && result[1] == 0)logic_gate_result =
INV_gate;
  else
  {
    look_for_NOR_gate();
    disp_result();
    if (result[0] == 1 && result[1] == 0 && result[2] == 0 &&
result[3] == 1)logic_gate_result = NOR_gate;
    else
    {
      look_for_basic_gate();
      disp_result();
      delay(2000);
      if (result[0] == 0 && result[1] == 0 && result[2] == 0 &&
result[3] == 1)logic_gate_result = AND_gate;
      else if (result[0] == 1 && result[1] == 1 && result[2] ==
1 && result[3] == 0)logic_gate_result = NAND_gate;
      else if (result[0] == 0 && result[1] == 1 && result[2] ==
1 && result[3] == 1)logic_gate_result = OR_gate;
      else if (result[0] == 0 && result[1] == 1 && result[2] ==
1 && result[3] == 0)logic_gate_result = EX_OR_gate;
      else if (result[0] == 1 && result[1] == 0 && result[2] ==
0 && result[3] == 1)logic_gate_result = EX_NOR_gate;
    }
  }

}//

void look_for_INV_gate()
{
  pinMode(pin1, OUTPUT);
  pinMode(pin2, INPUT);
  pinMode(pin3, OUTPUT);
  digitalWrite(pin3, LOW);
  for (int k = 0; k < 4; k++)
```

```
  {
    result[k] = 0;
  }
  for (int k = 0; k < 4; k++)
  {
    digitalWrite(test_fail_led, !digitalRead(test_fail_led));
    if (k == 0)
    {
      digitalWrite(pin1, LOW);
      delay(100);
      result[k] = digitalRead(pin2);
    }
    if (k == 1)
    {
      digitalWrite(pin1, HIGH);
      delay(100);
      result[k] = digitalRead(pin2);
    }
  }
}

void look_for_NOR_gate()
{
  pinMode(pin1, INPUT);
  pinMode(pin2, OUTPUT);
  pinMode(pin3, OUTPUT);
  delay(100);
  for (int k = 0; k < 4; k++)
  {
    result[k] = 0;
  }
  for (int k = 0; k < 4; k++)
  {
    digitalWrite(test_fail_led, !digitalRead(test_fail_led));
    if (k == 0)
    {
      digitalWrite(pin3, LOW);
      digitalWrite(pin2, LOW);
      delay(100);
      result[k] = digitalRead(pin1);
    }
    if (k == 1)
    {
      digitalWrite(pin3, HIGH);
      digitalWrite(pin2, LOW);
      delay(100);
      result[k] = digitalRead(pin1);
```

```
    }
    if (k == 2)
    {
      digitalWrite(pin3, LOW);
      digitalWrite(pin2, HIGH);
      delay(100);
      result[k] = digitalRead(pin1);
    }
    if (k == 3)
    {
      digitalWrite(pin3, HIGH);
      digitalWrite(pin2, HIGH);
      delay(100);
      result[k] = digitalRead(pin1);
    }
  }
}

void disp_result()
{
  lcd.setCursor(0, 1);
  lcd.print(result[0]);
  lcd.print(result[1]);
  lcd.print(result[2]);
  lcd.print(result[3]);
  delay(2000);
}

void look_for_basic_gate()
{
  pinMode(pin1, OUTPUT);
  pinMode(pin2, OUTPUT);
  pinMode(pin3, INPUT);
  for (int k = 0; k < 4; k++)
  {
    result[k] = 0;
  }
  for (int k = 0; k < 4; k++)
  {
    digitalWrite(test_fail_led, !digitalRead(test_fail_led));
    if (k == 0)
    {
      digitalWrite(pin1, LOW);
      digitalWrite(pin2, LOW);
      delay(100);
      result[k] = digitalRead(pin3);
    }
```

```
    if (k == 1)
    {
      digitalWrite(pin1, HIGH);
      digitalWrite(pin2, LOW);
      delay(100);
      result[k] = digitalRead(pin3);
    }
    if (k == 2)
    {
      digitalWrite(pin1, LOW);
      digitalWrite(pin2, HIGH);
      delay(100);
      result[k] = digitalRead(pin3);
    }
    if (k == 3)
    {
      digitalWrite(pin1, HIGH);
      digitalWrite(pin2, HIGH);
      delay(100);
      result[k] = digitalRead(pin3);
    }
  }
}



//end
```


```
<?xml version='2.0' encoding='UTF-8' standalone='yes'?>
<WORKSPACE>
<FRAME activewindow="1">
<PLACEMENT>2c0000000200000003000000000083ffff0083fffffffffffffffff
fffab0000000d000000d506000003040000</PLACEMENT>
<WINDOW type="default" module="STARTUP"/>
<WINDOW type="default" module="ISIS"/>
</FRAME>
<MODULE name="VSMDEBUG">
<PWI/>
</MODULE>
</WORKSPACE>
```