Yarmouk Private University

The Faculty of Informatics and Communication Engineering

Software Engineering and Communication Engineering Departments

Applied Project

9/7/2013

SMART LOCK

Android device for controlling and monitoring purpose

Supervised by: Eng. Nouar Al-Dahoul

BY: Wael Hamada, Rashad Abdulla, Anas Hamdan

DEDICATIONS

This Research is dedicated to our families in general and specifically our parents, who provided us with endless support, encouragement, love and who have the most credit behind our success.

عَنِ النَّبِيِّ صلى الله عليه وسلم ، قَالَ:

(مَنْ سَلَكَ طَرِيقًا يَلْتَمِسُ فِيهِ عِلْمًا ، سَهَّلَ اللهُ لَهُ طَرِيقًا إِلَى الْجَنَّةِ).

ACKNOWLEDGEMENTS

Our thanks go to our Supervisor: **Eng.Nouar Al-Dahoul**, who has guided us through this Project.

ABSTRACT:

The life has changed and humanity hobbits did since everything is about future and technology to make a better world, a world where everything in controlled by smart gadgets and the main gadgets now a days are smart phones, for that the developers around the world are trying to inject smart phones in every single moment of our life since those phones are in our hands around the day so we must join this explosion with a small application to control a house door using any smart phone around with a simple steps. And objective of this project is to build a system to control and monitor devices remotely by using Bluetooth, SMS or GSM techniques that are embedded on android device.

The system consists of a sender part that includes user mobile with android OS and receiver part that includes the host android mobile connected to microcontroller that is connected with devices.

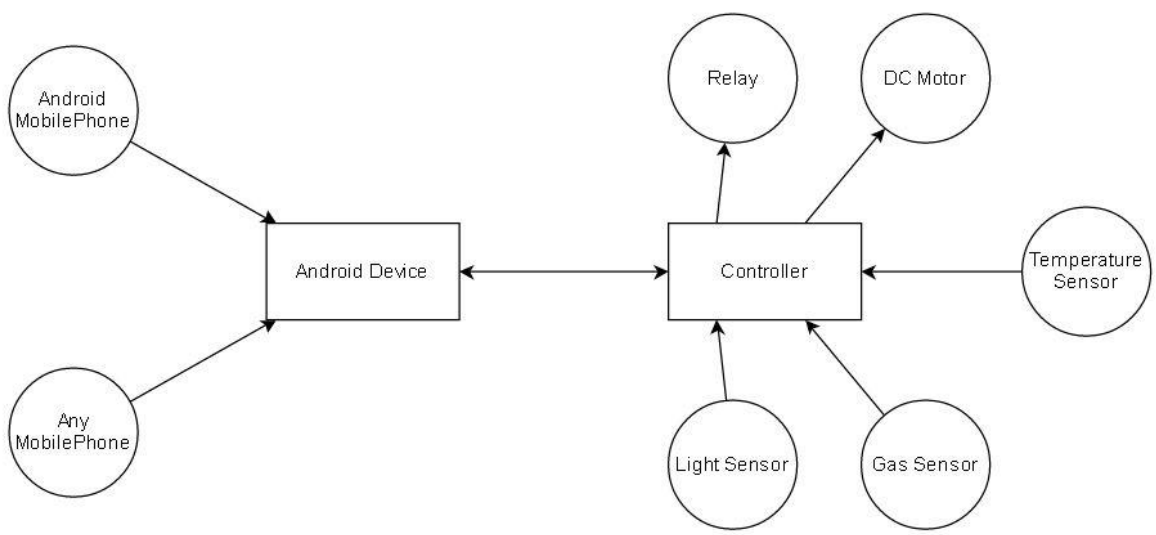


Figure 1

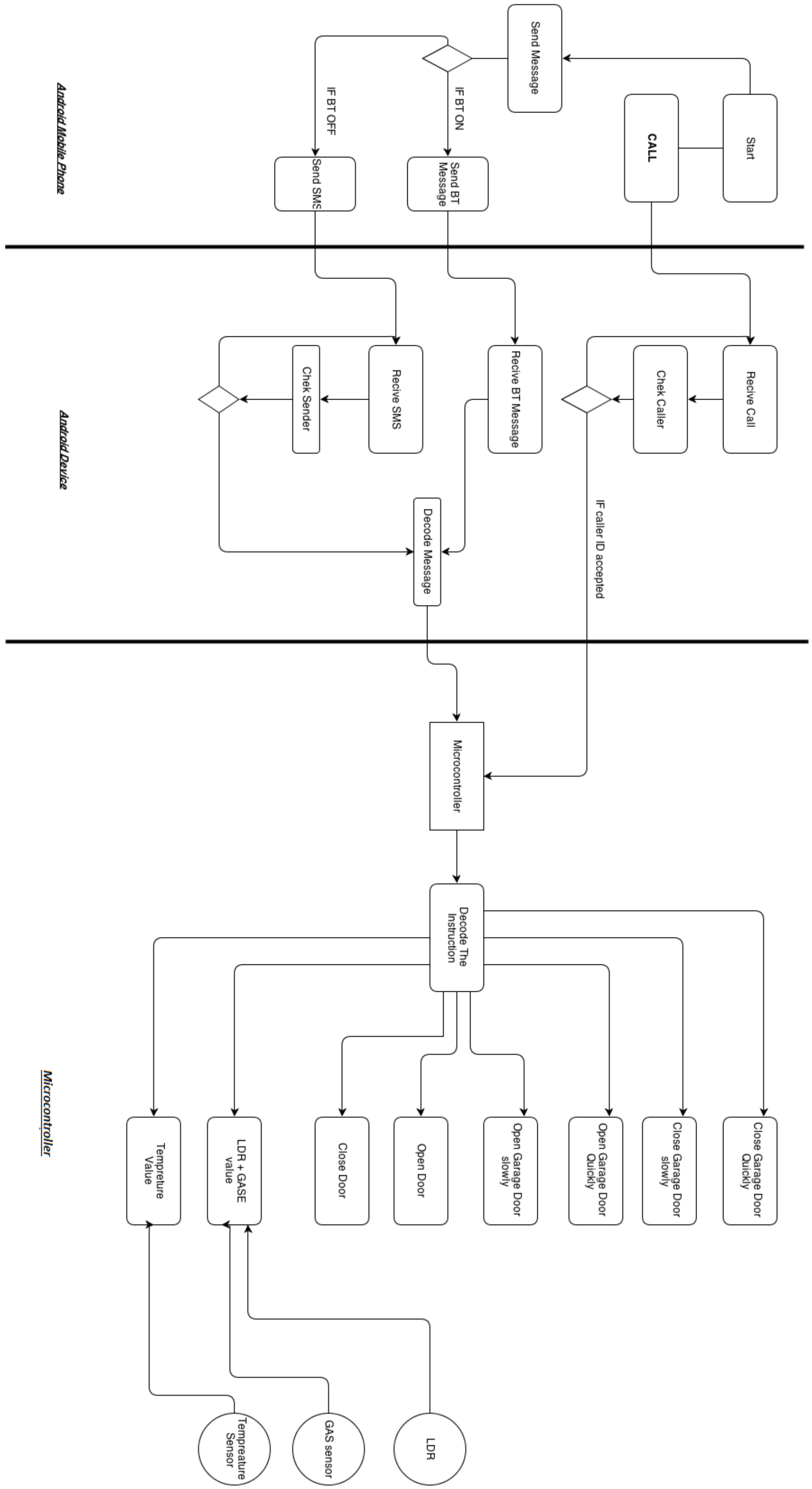

Android Mobile Phone
Start
CALL
Send Message
IF BT ON
IF BT OFF
Send BT Message
Send SMS
Android Device
Recive Call
Chek Caller
Recive BT Message
Recive SMS
Chek Sender
IF caller ID accepted
Decode Message
Microcontroller
Microcontroller
Decode The Instruction
Close Garage Door Quickly
Close Garage Door slowly
Open Garage Door Quickly
Open Garage Door slowly
Open Door
Close Door
LDR + GASE value
Tempreture Value
LDR
GAS sensor
Tempreature Sensor

Table of Contents:

Chapter 1: Hardware components 16
Microcontroller 16
Adapter 18
Max 232 19
RS232 Pin out 20
LDR 21
USB-to-Serial Cable 22
Gas Sensor 23
Relay 24
DC motor 25
LM336 27
H-bridge 28
Serial Communication 30
USART Registers 33
Analog to Digital conversion 42
ADC Register 44
Sensor Calibration 49

Chapter 2: Schematics and simulation 52

Chapter 3: Practical work 62
Using H-Term 63
Using an android device 69

Chapter 4: Software Components 73
Android overview 73
Required device feature 74
Usecase Diagrams 78
Sequence Diagrams 82
Application Development 85

Chapter 5: Android User interface 92
Application overview 92
Requirements 92
Usecase Diagrams 96
Sequence Diagrams 104
The mechanism of action within the system 110

Future Works 113

References 114

Table of Figures

Figure 1 (Block Diagram) 4

Figure 2 (Atmega16) 16

Figure 3 (Adapter) 18

Figure 4 (MAX232) 19

Figure 5 (RS232 Pin Out) 20

Figure 6 (Table of RS232 Pin Out) 21

Figure 7 (LDR Curve) 21

Figure 8 (LDR) 21

Figure 9 (USB to serial cable) 22

Figure 10 (Gas Sensor) 23

Figure 11 (Gas Sensor Circuit) 24

Figure 12 (Relay) 25

Figure 13 (Motor block diagram) 25

Figure 14 (Motor) 26

Figure 15 (LM335) 27

Figure 16 (H-bridge) 28

Figure 17 (Full bridge operation) 28

Figure 18 (PWM control of a full bridge) 29

Figure 19 (Frame) 30

Figure 20 (Simplex) 31

Figure 21 (Half-duplex) 31

Figure 22 (Full-duplex) 31

Figure 23 (USART Pin) 32

Figure 24 (Frame equation) 32

Figure 25 (UCSRA) 33

Figure 26 (UCSRB) 33

Figure 27 (Parity bits) 34

Figure 28 (UCSZ table) 35

Figure 29 (UBRRH + UBRRL) 36

Figure 30 (UBRR) 36

Figure 31 (MAX 232 Circuit 2) 38

Figure 32 (MAX 232 General Schismatic) 39

Figure 33 (Baud rate table) 40

Figure 34 (USART of AVR Microcontroller) 41

Figure 35 (ADC) 42

Figure 36 (ADC pins) 43

Figure 37 (ADC clock frequency) 43

Figure 38 (Channels) 44

Figure 39 (ADMUX) 44

Figure 40 (Reference voltage selection) 45

Figure 41 (ADC pin reference) 45

Figure 42 (Input channel and gain selection) 46

Figure 43 (ADCSRA register) 46

Figure 44 (ADC-PreScaler) 47

Figure 45 (Sensor Calibration) 49

Figure 46 (Full ADC) 51

Figure 47 (Motor and PWM) 52

Figure 48 (Start motor) 53

Figure 49 (Select baud rate) 54

Figure 50 (Speed motor right) 55

Figure 51 (Slow motor right) 56

Figure 52 (Slow motor left) 57

Figure 53 (Speed motor left) 58

Figure 54 (Stop motor) 59

Figure 55 (Temperature Sensor) 60

Figure 56 (Relay) 61

Figure 57 (Test board) 62

Figure 58 (Send 1.1) 63

Figure 59 (Send 1.2) 63

Figure 60 (Send 1.3) 64

Figure 61 (Send 1.4) 64

Figure 62 (Send 2) 65

Figure 63 (Send 3) 65

Figure 64 (Send 4) 66

Figure 65 (Send 5) 66

Figure 66 (Send 6) 67

Figure 67 (Send 7) 67

Figure 68 (Send 8) 68

Figure 69 (Send 9) 68

Figure 70 (Send 2 from Android device) 69

Figure 71 (Send 3 from Android device) 70

Figure 72 (Send 4 from Android device) 70

Figure 73 (Send 5 from Android device) 71

Figure 74 (Send 6 from Android device) 71

Figure 75 (Send 7 from Android device) 72

Figure 76 (Send 8 from Android device) 72

Figure 77 (Android logo) 73

Figure 78 (USB-OTG) 75

Figure 79 (Bluetooth logo) 76

Figure 80 (GSM logo) 77

Figure 81 (Start Sequence Diagram) 82

Figure 82 (Incoming Bluetooth Sequence Diagram) 83

Figure 83 (Incoming SMS message Sequence Diagram) 83

Figure 84 (Incoming voice call Sequence Diagram) 84

Figure 85 (Android logo) 93

Figure 86 (Start Sequence Diagram) 104

Figure 87 (Show LDR + Gas Sequence Diagram) 104

Figure 88 (Close garage quickly Sequence Diagram) 105

Figure 89 (Close garage slowly Sequence Diagram) **105**

Figure 90 (Open garage quickly Sequence Diagram) **106**

Figure 91 (Open garage slowly Sequence Diagram) **106**

Figure 92 (Open door Sequence Diagram) **107**

Figure 93 (Close door Sequence Diagram) **107**

Figure 94 (Temperature Sequence Diagram) **108**

Figure 95 (Call Sequence Diagram) **108**

Introduction:

Home automation refers to the use of computer and information technology to control home appliances and features (such as windows or lighting). Systems can range from simple remote control of lighting through to complex computer/micro-controller based networks with varying degrees of intelligence and automation. Home automation is adopted for reasons of ease, security and energy efficiency
Control and integration of security systems and also the potential for central locking of all perimeter doors and windows

Security systems can include motion sensors that will detect any kind of unauthorized movement and notify the user through the security system or via cell phone. This category also includes control and distribution of security cameras.

COMMUNICATION ENGINEERING SECTION

Chapter1

Hardware components:

In this chapter will discuss the basic elements of the project and its future.

The project is made of 12 Components:

1. **Microcontroller**:
   A microcontroller often serves as the “brain” of a mechatronic system. Like a mini, self-contained computer, it can be programmed to interact with both the hardware of the system and the user. Even the most basic microcontroller can perform simple math operations, control digital outputs, and monitor digital inputs. As the computer industry has evolved, so has the technology associated with microcontrollers. Newer microcontrollers are much faster, have more memory, and have a host of input and output features that dwarf the ability of earlier models. Most modern controllers have **Analog-to-digital** converters, high-speed timers and counters, interrupt capabilities, outputs that can be **pulse-width modulated**, **serial communication** ports, etc.

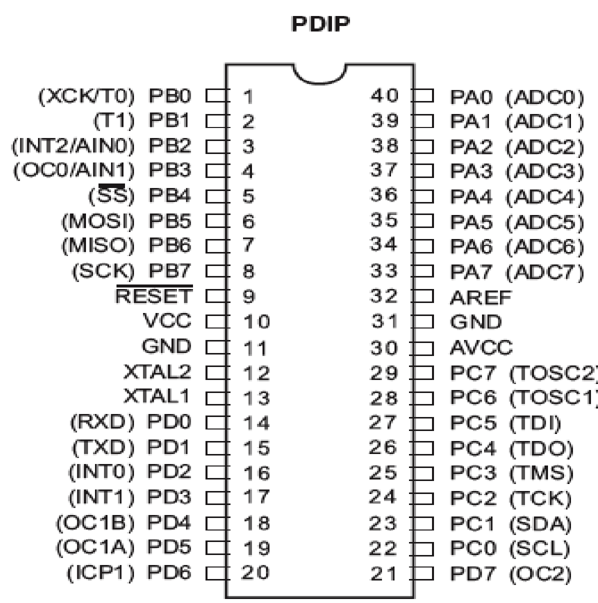


Figure 2

The AVR core combines a rich instruction set with 32 general purpose working registers.
All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU),
Allowing two independent registers to be accessed in one single instruction executed in one clock cycle.
The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The AVR ATmega16 provides the following features:

- 16K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities,
- 512 bytes EEPROM,
- 1K byte SRAM,
- 32 general purpose I/O lines,
- 32 general purpose working registers,
- A JTAG interface for Boundary scan,
- On-chip Debugging support and programming,
- Three flexible Timer/Counters with compare modes,
- Internal and External Interrupts,
- A serial programmable USART,
- A byte oriented Two-wire Serial Interface,
- An 8-channel, 10-bit ADC with optional differential input stage with programmable
- Gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator,
- An SPI serial port, and six software selectable power saving modes.

The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM,

Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, Disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.
The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping.
This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run

2. Adapter :

Figure 3

The adapter convert the AC 220 v to DC 5 v that working with this project exactly

3. Max232:

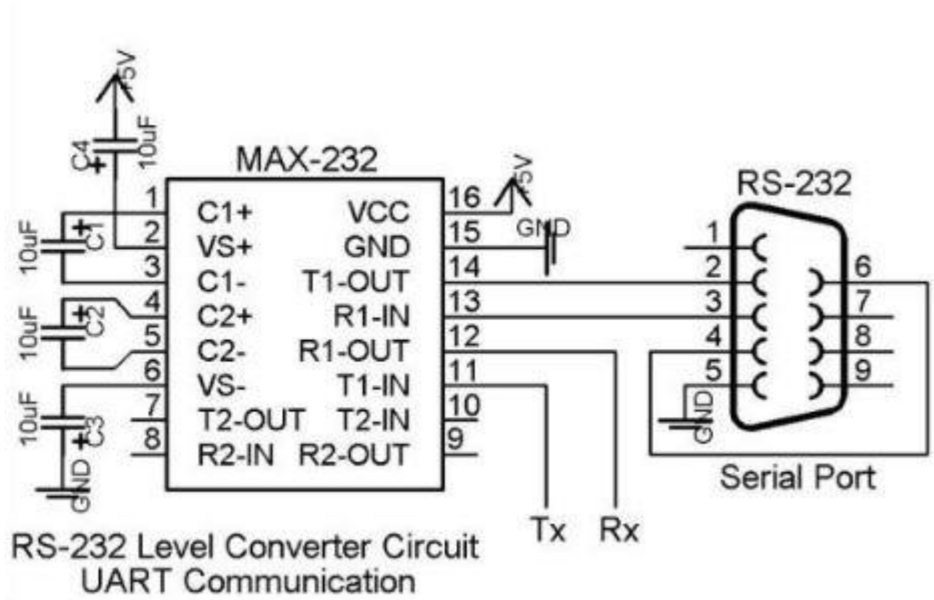


Figure 4

The MAX232 was the first IC which in one package contains the necessary drivers (two) and receivers (also two), to adapt the RS-232 signal voltage levels to TTL logic. It became popular, because it just needs one voltage (+5V) and generates the necessary RS-232 voltage levels (approx. -10V and +10V) internally. This greatly simplified the design of circuitry.

4. RS232 PINOUT :

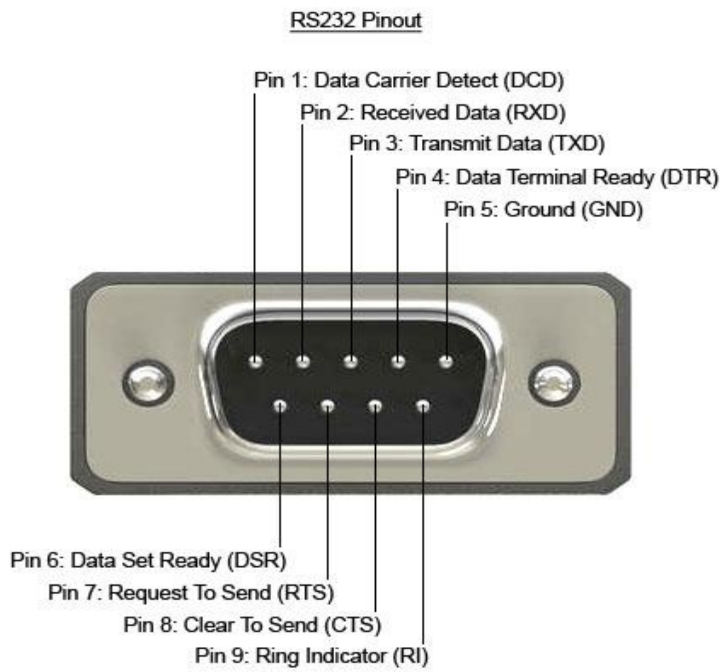


Figure 5

RS-232 (also called serial, COM port) is a common interface and most of PC nowadays are still equipped with one or two serial interface (RS232C) connectors. This PC serial port interface is single ended (connects only two devices with each other), the data rate is less than 20 kbps. It's a voltage loop serial interface with full-duplex communication represented by voltage levels with respect to system ground. A common ground between the PC and the associated device is necessary

DE-9 Pin	Signal Name	Dir	Description	IDC internal (newer)*	IDC internal (older)*
1	DCD	←	Data Carrier Detect	1	1
2	RXD	←	Receive Data	2	3
3	TXD	→	Transmit Data	3	5
4	DTR	→	Data Terminal Ready	4	7
5	GND	—	System Ground	5	9
6	DSR	←	Data Set Ready	6	2
7	RTS	→	Request to Send	7	4
8	CTS	←	Clear to Send	8	6
9	RI	←	Ring Indicator	9	8

Figure 6

5. light-dependent resistor:

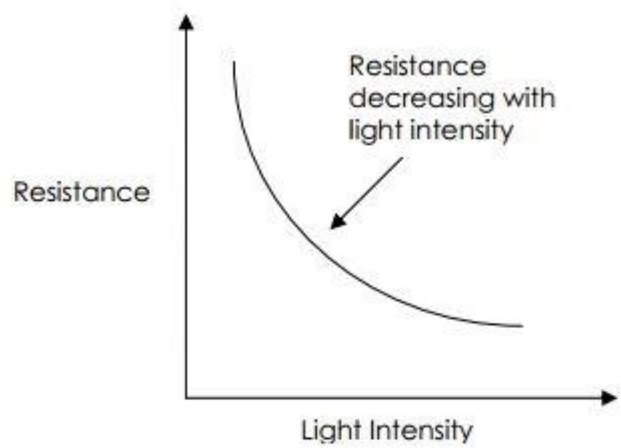


Figure 7

Alternatively called an LDR, photo resistor, photoconductor, or *photocell*, is a variable resistor whose value decreases with increasing incident light intensity. An LDR is made of a high-resistance semiconductor. If light falling on the device is of high enough frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electron (and its hole partner) conduct electricity, thereby lowering resistance. A photoelectric device can be either intrinsic or extrinsic. In intrinsic devices, the only available electrons are in the valence band, and hence the photon must have enough energy to excite the electron across the entire bandgap. Extrinsic


Figure 8

devices have impurities added, which have a ground state energy closer to the conduction band - since the electrons don't have as far to jump, lower energy photons (i.e. longer wavelengths and lower frequencies) are sufficient to trigger the device. Two of its earliest applications were as part of smoke and fire detection systems and camera light meters. Because cadmium sulfide cells are inexpensive and widely available, LDRs are still used in electronic devices that need light detection capability, such as security alarms, street lamps, and clock radios

6. USB to serial cable :

a. Used to connect serial devices such as modems to a PC or a laptop's USB port
b. Consists of a USB A type plug on one end and a male 9-pin serial connector on the other
c. Convert the serial RS232 to a USB.

Figure 9

7. Gas sensor :

Figure 10

The MQ series of gas sensors use a small heater inside with an electro-chemical sensor. They are sensitive for a range of gasses and are used indoors at room temperature. The output is an analog signal and can be read with an analog input of the Arduino.

The voltage for the internal heater is very important some sensors use 5V for the heater, others need 2V. The 2V can be created with a PWM signal, using analogWrite() and a transistor or logic-level mosfet.The heater may not be connected directly to an output-pin of the Arduino, since it uses too much current for that.

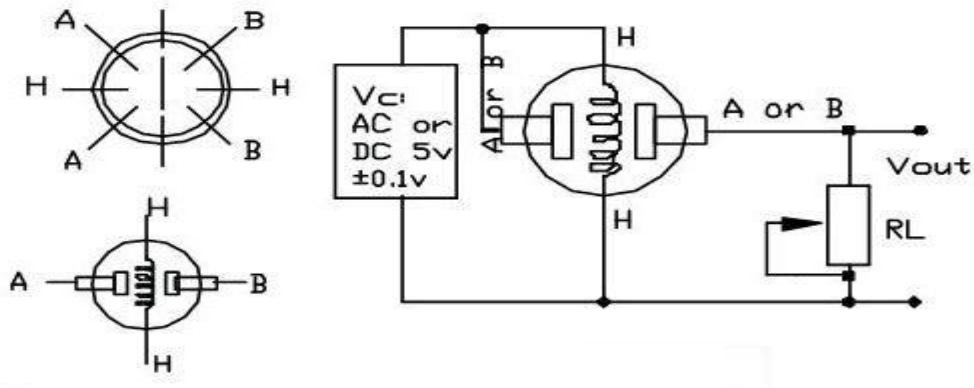


Figure11

8. **Relay :**

Relay is one of the most important electromechanical devices highly used in industrial applications specifically in automation. A relay is used for electronic to electrical interfacing i.e. it is used to switch on or off electrical circuits operating at high AC voltage using a low DC control voltage. A relay generally has two parts, a coil which operates at the rated DC voltage and a mechanically movable switch. The electronic and electrical circuits are electrically isolated but magnetically connected to each other, hence any fault on either side does not affects the other side.

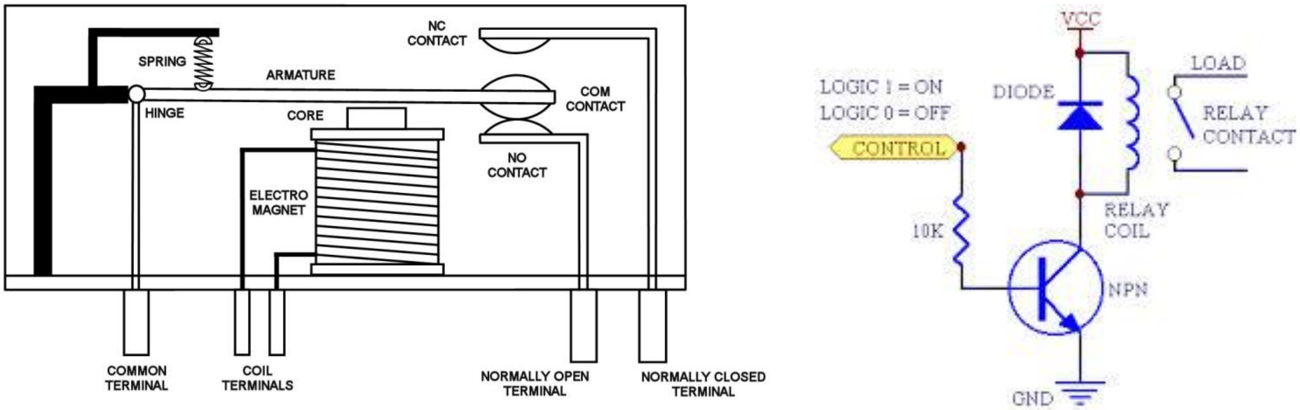


Figure 12

9. DC Motors :

A DC motor works by converting electric power into mechanical work. This is accomplished by forcing current through a coil and producing a magnetic field that spins the motor.

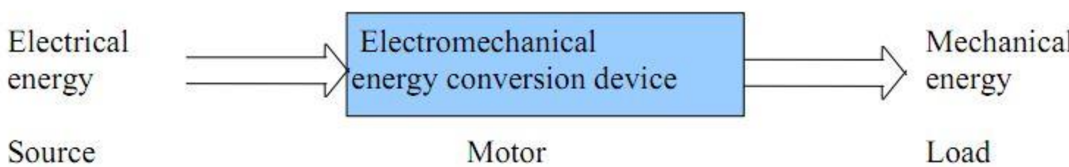


Figure13

Figure 14

A D.C. motor consists of a rectangular coil made of insulated copper wire wound on a soft iron core. This coil wound on the soft iron core forms the armature. The coil is mounted on an axle and is placed between the cylindrical concave poles of a magnet.

When the coil is powered, a magnetic field is generated around the armature. The left side of the armature is pushed away from the left magnet and drawn towards the right, causing rotation.

10. Lm335 :

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM135 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K. With less than 1Ω dynamic impedance the device operates over a current range of 400 μA to 5 mA with virtually no change in performance. When calibrated at 25°C the LM135 has typically less than 1°C error over a 100°C temperature range. Unlike other sensors the LM135 has a linear output.

Application for the LM135 include almost any type of temperature sensing over a −55°C to 150°C temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy.

The LM135 operates over a −55°C to 150°C temperature range while the LM235 operates over a −40°C to 125°C temperature range. The LM335 operates from −40°C to 100°C. The LM135/LM235/LM335 are available packaged in hermetic TO transistor packages while the LM335 is also available in plastic TO-92 packages.

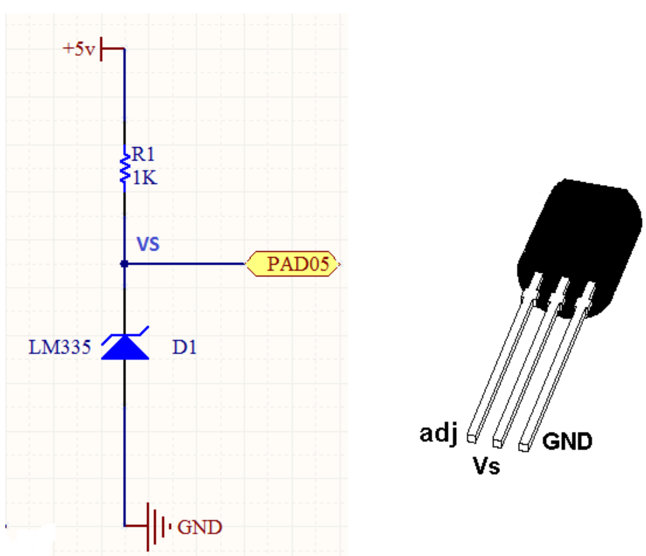


Figure 15

11. H-Bridge:

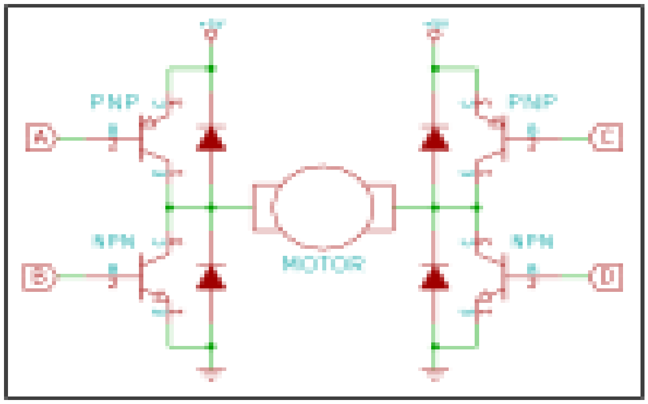


Figure 16

The H-bridge is a circuit used in electronic control of high current devices, particularly where the device polarity may be reversed, e.g. DC motors. The name comes from the fact that the circuit typically looks like a letter "H".

Full bridge operation:

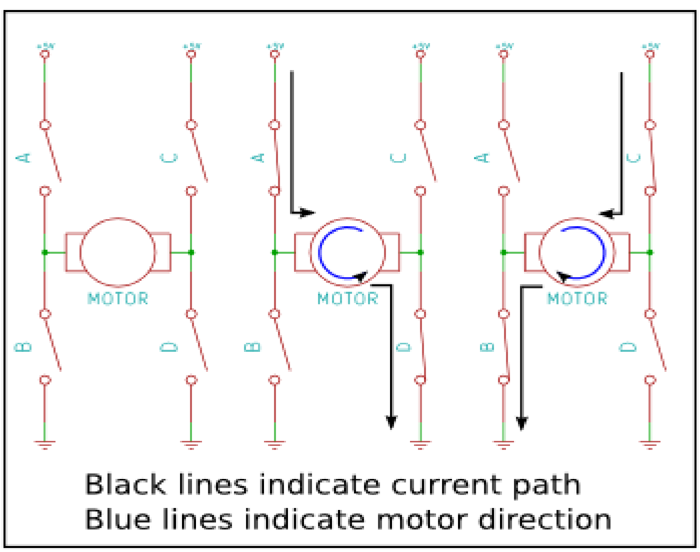


Figure 17

The circuit shown has four switches and a motor. To apply a voltage across the motor a pair of diagonally opposite switches need to be turned on. Depending on which pair of switches are turned on the motor will turn one way or another. If both the top or both the bottom switches are turned on the motor will have no voltage difference across it so it won't move at all. If the top and bottom switches on one side are turned on together by accident we will short out the supply, so you need to be careful if you're building your own H-bridge.

PWM Control of a full bridge:

PWM is a method of digitally controlling an output with a variable equivalent voltage. Essentially if you take the average of the signal over time then it has a varying analog level, however in the short term it is digital. This makes it easy to generate and efficient as transistors are most efficient when on or off rather than partially conducting. Most modern microcontrollers have the ability to generate PWM built in, including the atmega16 .

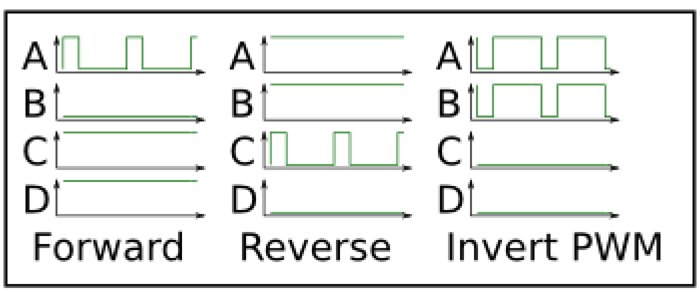


Figure 18

To control a plain H-bridge you need 4 signals to control the 4 transistors, to control it with PWM you need two PWM signals and two plain digital signals. You could theoretically use just the two PWM outputs, however you need to be careful about the polarity of the signal compared to the transistor you're driving. Some microcontrollers include a full bridge driver which drives 4 outputs and you can hook up to the H-bridge. However it still needs 4 control lines. If you can drive both your PNP and NPN transistors from the same logic line you can connect the PWM to both A and B inputs and a direction control to C and D. In this configuration however you have to invert the duty-cycle when you are running in reverse because it's the low part of the duty cycle that turns the motor on.

12. Serial communication (Data receive) using AVR Microcontroller (ATmega16) USART:

Data can be exchanged using parallel or serial techniques. Setup for parallel data transfer is not cost effective but is a very fast method of communication. Serial communication is cost effective because it requires only a single line of connection but on the other hand is a slow process in comparison to parallel communication. This article explains serial communication of AVR microcontroller (ATmega16) with PC. The data is transmitted from the controller using RS232 standard and displayed on the PC using Hyper Terminal.

There are two methods for serial data communication (i) Synchronous and (ii) Asynchronous communication.

In synchronous communication the frame consists of data bits while in asynchronous communication the total number of bits in a frame may be more than the data bits.

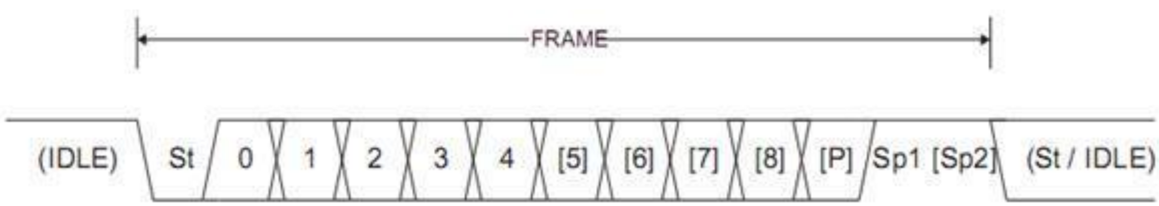


Figure 19

There are three ways in which serial communication can be done

i. Simplex: Transmission is done in one direction.

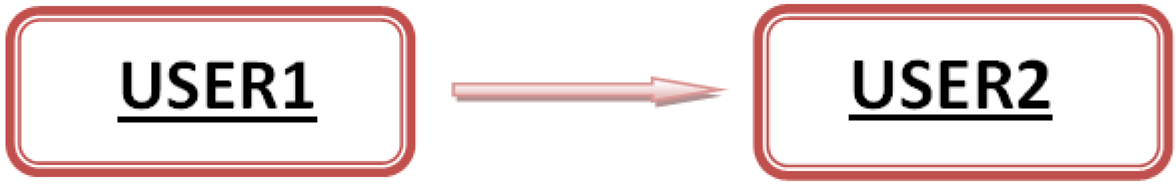


Figure 20

ii. Half duplex: Transmission can be done in both the direction but one side at a time.

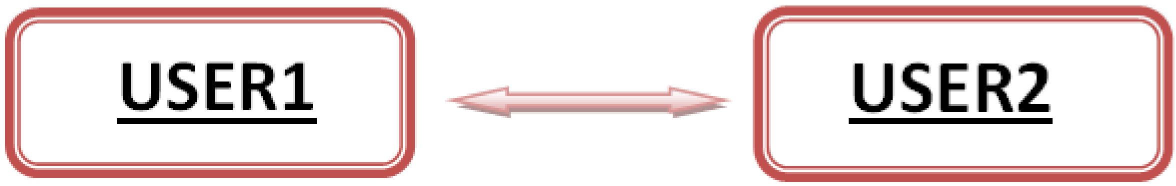


Figure 21

iii. Full duplex: Transmission can be done in both the direction simultaneously.

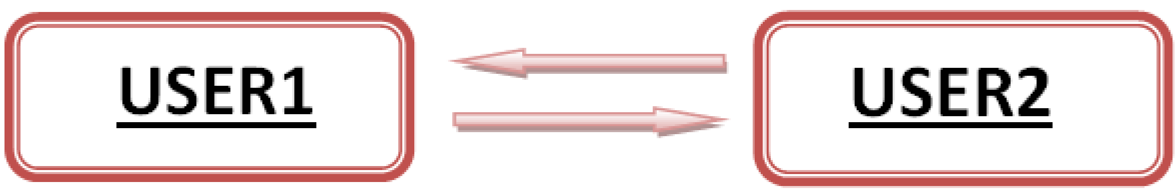


Figure 22

SERIAL USART (universal synchronous asynchronous receiver and transmission/ transmitter):

Serial USART provides **full-duplex** communication between the transmitter and receiver. Atmega16 is equipped with independent hardware for serial USART communication. Pin-14 (RXD) and Pin-15 (TXD) provide receive and transmit interface to the microcontroller.

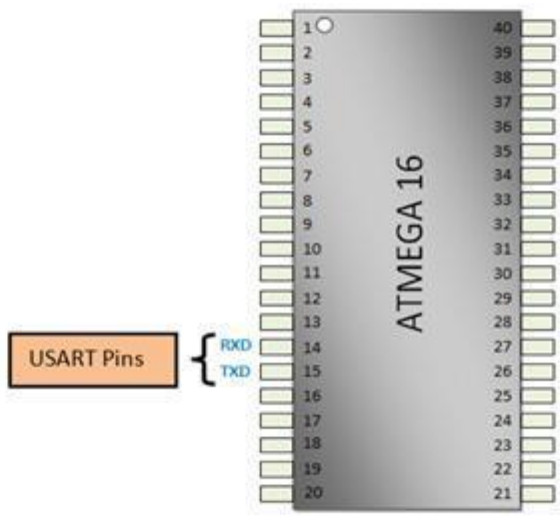


Figure 23

Atmega16 USART provides asynchronous mode of communication and do not have a dedicated clock line between the transmitting and receiving end. The synchronization is achieved by properly setting the baud rate, start and stop bits in a transmission sequence.

Start bit and stop bit: These bits are use to synchronize the data frame. Start bit is one single low bit and is always given at the starting of the frame, indicating the next bits are data bits. Stop bit can be one or two high bits at the end of frame, indicating the completion of frame.

One frame= start bit + data bits + stop bit.

Figure 24

USART Registers

To use the USART of Atmega16, certain registers need to be configured.
UCSR: USART control and status register. It's is basically divided into three parts **UCSRA**, **UCSRB** and **UCSRC**. These registers are basically used to configure the USART.
UBRR: USART Baud Rate Registers. Basically use to set the baud rate of USART
UDR: USART data register

i. UCSRA: (USART Control and Status Register A):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
0	0	1	0	0	0	0	0

Figure 25

RXC (USART Receive Complete): RXC flag is set to 1 if unread data exists in receive buffer, and set to 0 if receive buffer is empty.
TXC (USART Transmit complete): TXC flag is set to 1 when data is completely transmitted to Transmit shift register and no data is present in the buffer register UDR.
UDRE (USART Data Register Empty): This flag is set to logic 1 when the transmit buffer is empty, indicating it is ready to receive new data. UDRE bit is cleared by writing to the UDR register.

ii. UCSRB: (USART Control and Status Register B)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
0	0	0	0	0	0	0	0

Figure 26

RXCIE: RX Complete Interrupt Enable,
When 1 -> RX complete interrupt is enabled.
When 0 -> RX complete interrupt is disabled.

TXCIE: TX Complete Interrupt Enable,
When 1 -> TX complete interrupt is enabled
When 0-> TX complete interrupt is disabled

UDRIE: USART Data Register Empty Interrupt Enable,
When 1 -> UDRE flag interrupt is enabled.
When 0 -> UDRE flag interrupt is disabled.

RXEN: Receiver Enabled,
When 1 -> USART Receiver is enabled.
When 0 -> USART Receiver is disabled.

TXEN: Transmitter Enabled,
When 1 -> USART Transmitter is enabled.
When 0 -> USART Transmitter is disabled.

iii. UCSRC: (USART Control and Status Register C)

The transmitter and receiver are configured with the same data features as configured in this register for proper data transmission.

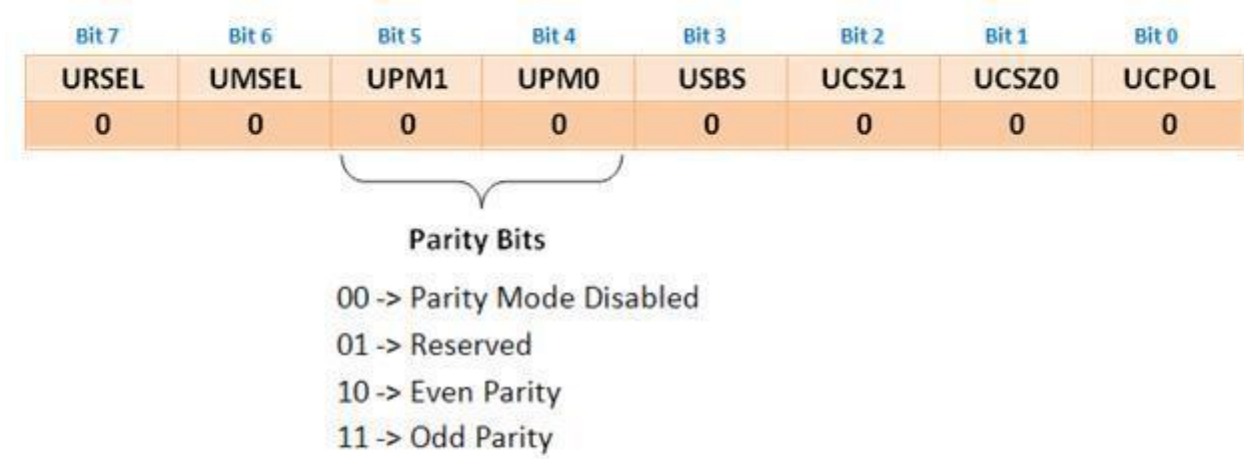


Figure 27

URSEL: USART Register select. This bit must be set due to sharing of I/O location by UBRRH and UCSRC
UMSEL: USART Mode Select,
When 1 -> Synchronous Operation
When 0 -> Asynchronous Operation

UPM[0:1]: USART Parity Mode, Parity mode selection bits.
USBS: USART Stop Select Bit,
When 0-> 1 Stop Bit
When 1 -> 2 Stop Bits

UCSZ[0:1]: The UCSZ[1:0] bits combined with the UCSZ2 bit in UCSRB sets size of data frame i.e., the number of data bits. The table shows the bit combinations with respective character size.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Figure 28

iv. UBRRH & UBRRL (USART Baud Rate Registers)

	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
UBRRH	URSEL	-	-	-	UBRR11	UBRR10	UBRR9	UBRR8
UBRRL	UBRR7	UBRR6	UBRR5	UBRR4	UBRR3	UBRR2	UBRR1	UBRR0

Figure 29

The UBRRH register shares the same I/O address with the UCSRC register, The differentiation is done on the basis of value of URSEL bit.
When URSEL=0; write operation is done on UBRRH register.
When URSEL=1; write operation is done on UCSRC register.
The UBRRH and UBRRL register together stores the 12-bit value of baud rate, UBRRH contains the 4 most significant bits and UBRRL contains the other 8 least significant bits. Baud rates of the transmitting and receiving bodies must match for successful communication to take place.

UBRR register value is calculated by the following formula:

$$UBRR = ((\text{System Clock Freq})/(16 \times \text{Baud Rate})) - 1$$

$$UBRR = \frac{\text{XTAL Frequency}}{16 \times \text{Baud Rate}} - 1$$

Example: XTAL Freq = 12 MHz

Baud Rate = 9600 bps

UBRR = (12000000 / (16 * 9600)) - 1

= 77 (Decimal approx)

= 4D (Hex equivalent)

Figure 30

AVR/PC Serial Communication via MAX232:

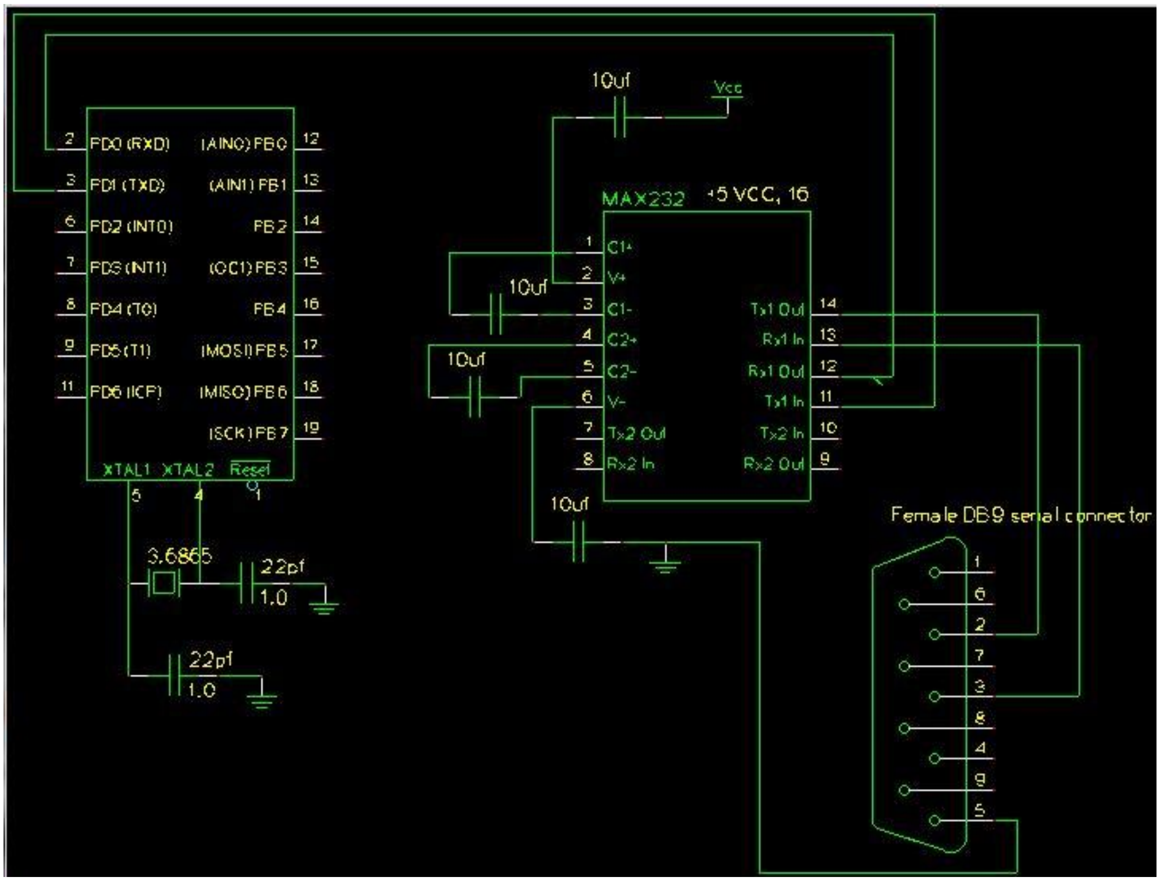


Figure 31

14 ➔ 3 , 13 ➔2
12 ➔ rx , 11 ➔ tx

Here's a more general schematic:

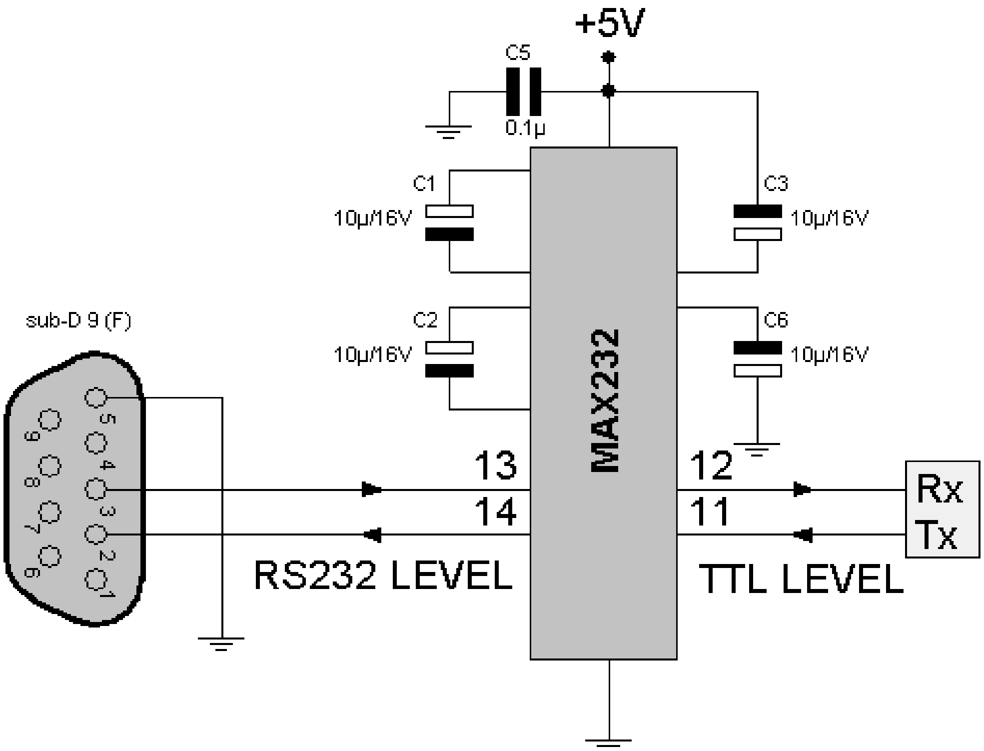


Figure 32

Determining the speed of serial communication is important.

$$UBRR = \frac{f_{osc}}{16 \times \text{Baud Rate}} - 1$$

The datasheet of your microcontroller may list a lot of common crystal frequencies, bandwidths, and their appropriate UBRR values.

		Baud Rate						
Multiple	Frequency (MHz)	2400	4800	9600	19200	38400	57600	115200
1	1.8432	47	23	11	5	2	1	0
2	3.6864	95	47	23	11	5	3	1
3	5.5296	143	71	35	17	8	5	2
4	7.3728	191	95	47	23	11	7	3
5	9.2160	239	119	59	29	14	9	4
6	11.0592	287	143	71	35	17	11	5
7	12.9024	335	167	83	41	20	13	6
8	14.7456	383	191	95	47	23	15	7
9	16.5888	431	215	107	53	26	17	8

Figure 33

Using the USART of AVR Microcontrollers: Reading and Writing Data

We need to generate two functions:

USARTReadChar() : To read the data (char) from the USART buffer.
USARTWriteChar(): To write a given data (char) to the USART.

USART of AVR Microcontrollers:

The USART of the AVR is connected to the CPU by the following six registers.

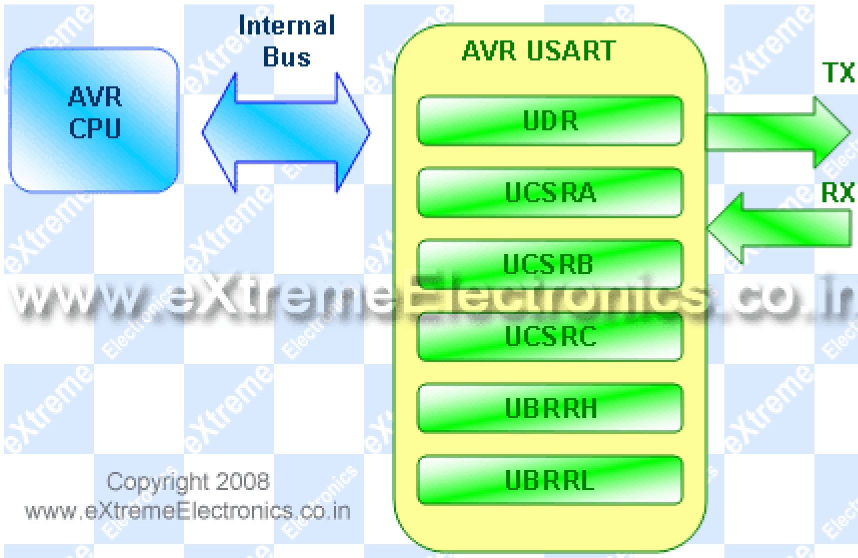


Figure 34

13. Analog to Digital Conversion

Most real world data is analog. Whether it be temperature, pressure, voltage, etc., their variation is always analog in nature. For example, the temperature inside a boiler is around 800°C. During its light-up, the temperature never approaches directly to 800°C. If the ambient temperature is 400°C, it will start increasing gradually to 450°C, 500°C and thus reaches 800°C over a period of time. This is an analog data.

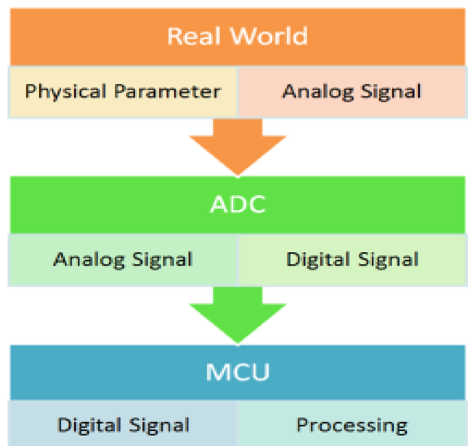


Figure 35

Signal Acquisition Process:
Now, we must process the data that we have received. But analog signal processing is quite inefficient in terms of accuracy, speed and desired output. Hence, we convert them to digital form using an Analog to Digital Converter (ADC).

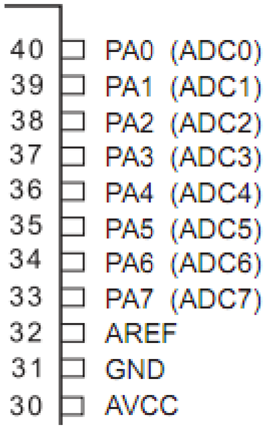


Figure 36

The ADC of the AVR:

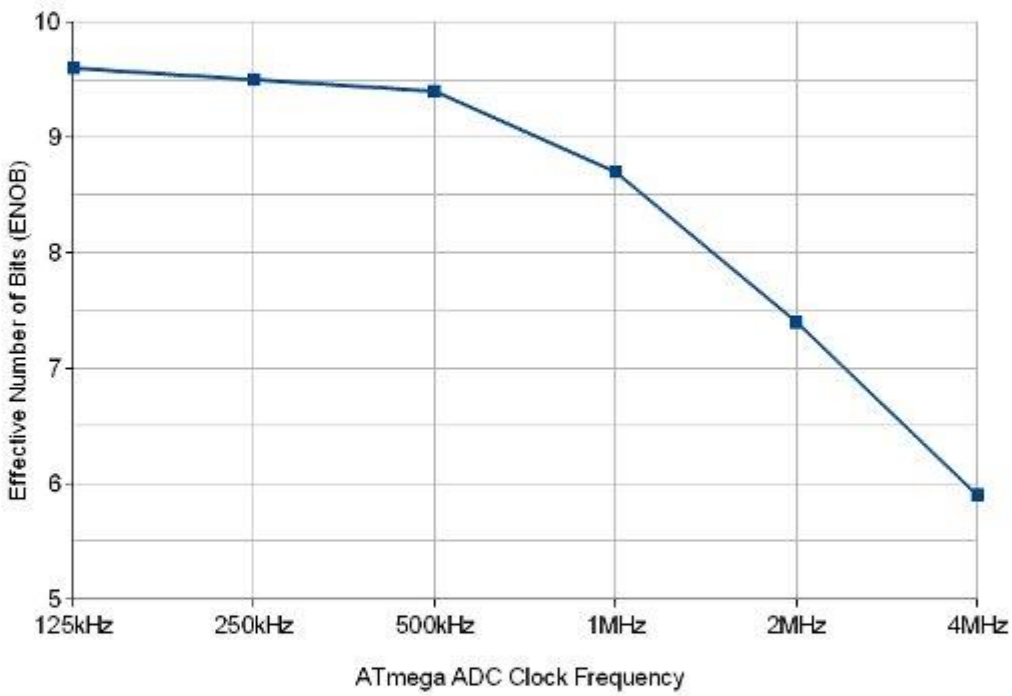


Figure 37

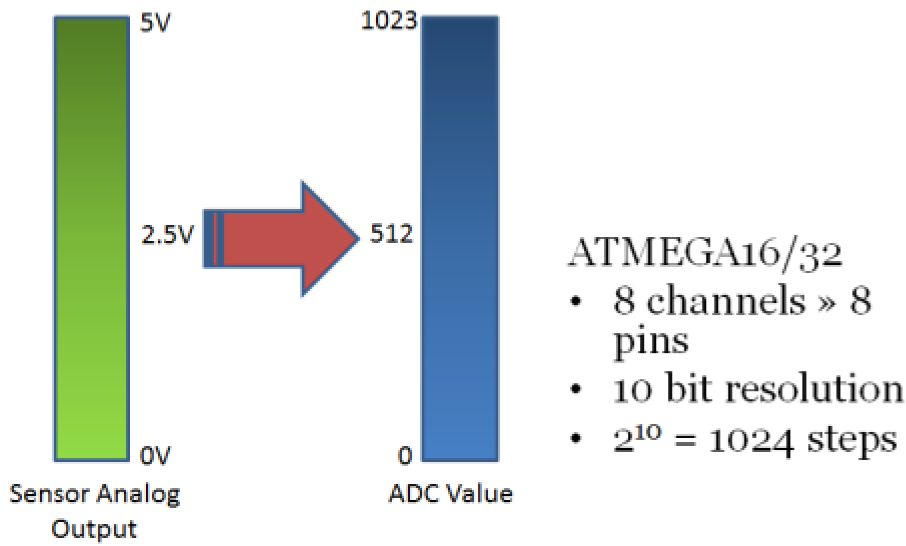


Figure 38

Suppose we use a 5V reference. In this case, any analog value in between 0 and 5V is converted into its equivalent ADC value as shown above. The 0-5V range is divided into 2^10 = 1024 steps. Thus, a 0V input will give an ADC output of 0, 5V input will give an ADC output of 1023, whereas a 2.5V input will give an ADC output of around 512. This is the basic concept of ADC.

ADC Registers:

ADMUX – ADC Multiplexer Selection Register

The ADMUX register is as follows.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 39

The bits that are highlighted are of interest to us. In any case, we will discuss all the bits one by one.

Bits 7:6 – REFS1:0 – Reference Selection Bits – These bits are used to choose the reference voltage. The following combinations are used.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Figure 40

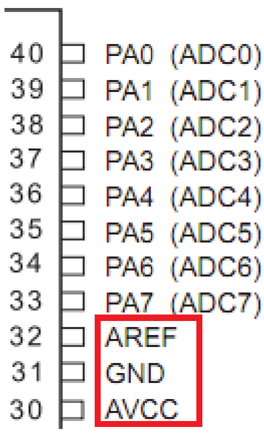


Figure 41

The ADC needs a reference voltage to work upon. For this we have a three pins AREF, AVCC and GND. We can supply our own reference voltage across AREF and GND. For this, **choose the first option**. Apart from this case, you can either connect a capacitor across AREF pin or ground it to prevent from noise, or you may choose to leave it unconnected. If you want to use the VCC (+5V), **choose the second option**. Or else, **choose the last option** for internal Vref.
Let's choose the second option for Vcc = 5V.
Bit 5 – ADLAR – ADC Left Adjust Result – Make it '1' to Left Adjust the ADC Result. We will discuss about this a bit later.
Bits 4:0 – MUX4:0 – Analog Channel and Gain Selection Bits – There are 8 ADC channels (PA0…PA7). Which one do we choose? Choose any one! It doesn't matter. How to choose? You can choose it by setting these bits. Since there are 5 bits, it consists of 2^5 = 32 different conditions as follows. However, we are concerned only with the first 8 conditions. Initially, all the bits are set to zero.

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Figure 42

Thus, to initialize ADMUX, we write
ADMUX = (1<<REFS0);

ADCSRA – ADC Control and Status Register A

The ADCSRA register is as follows.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 43

The bits that are highlighted are of interest to us. In any case, we will discuss all the bits one by one.
Bit 7 – ADEN – ADC Enable – As the name says, it enables the ADC feature. Unless this is enabled, ADC operations cannot take place across PORTA i.e. PORTA will behave as GPIO pins.
Bit 6 – ADSC – ADC Start Conversion – Write this to ’1′ before starting any conversion. This 1 is written as long as the conversion is in progress, after which it returns to zero. Normally it takes 13 ADC clock pulses for this operation. But when you call it for the first time, it takes 25 as it performs the initialization together with it.
Bit 5 – ADATE – ADC Auto Trigger Enable – Setting it to ’1′ enables auto-triggering of ADC. ADC is triggered automatically at every rising edge of clock pulse. View the SFIOR register for more details.
Bit 4 – ADIF – ADC Interrupt Flag – Whenever a conversion is finished and the registers are updated, this bit is set to ’1′ automatically. Thus, this is used to check whether the conversion is complete or not.
Bit 3 – ADIE – ADC Interrupt Enable – When this bit is set to ’1′, the ADC interrupt is enabled. This is used in the case of interrupt-driven ADC.
Bits 2:0 – ADPS2:0 – ADC Pre-scaler Select Bits – The pre-scaler (division factor between XTAL frequency and the ADC clock frequency) is determined by selecting the proper combination from the following.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Figure 44

Assuming XTAL frequency of 16MHz and the frequency range of 50 kHz-200 kHz, we choose a pre-scaler of 128.
Thus, F_ADC = 16M/128 = 125 kHz.
Thus, we initialize ADCSRA as follows.

```
ADCSRA = (1<<ADEN)| (1<<ADPS2)| (1<<ADPS1)| (1<<ADPS0);
// prescaler = 128
```

Sensor Calibration:

Calibration means linking your real world data with the virtual data. In the problem statement given earlier, I have mentioned that the LED should glow if the light intensity reduces. But **when** should it start to glow? The MCU/code doesn't know by itself. You get the readings from the sensor continuously in between 0 and 1023. So, the question is **how do we know that below 'such and such' level the LED should glow?**

This is achieved by calibration. You need to physically set this value. What you do is that you run the sensor for all the lighting conditions. You have the ADC values for all these levels. Now, you need to physically see and check the conditions yourself and then apply a threshold. Below this threshold, the light intensity goes sufficiently down enough for the LED to glow.

The potentiometer connected in the circuit is also for the same reason. Now, by the basic knowledge of electronics, you could easily say that upon changing the pot value the ADC value changes. Thus, for various reasons (like poor lighting conditions, you are unable to distinguish between bright and dark conditions, etc), you can vary the pot to get desired results.

Microcontroller understands only digital language. However, the inputs available from the environment to the microcontroller are mostly analog in nature, i.e., they vary continuously with time. In order to understand the inputs by the digital processor, a device called Analog to Digital Converter (ADC) is used. As the name suggests this peripheral gathers the analog information supplied from the environment and converts it to the controller understandable digital format, microcontroller then processes the information and provides the desired result at the output end.

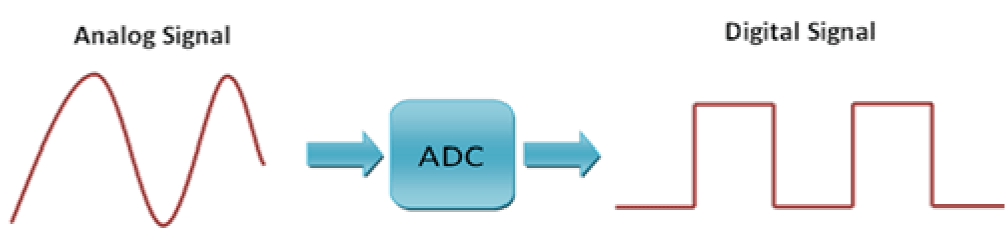


Figure 45

This is why I have given the two thresholds (RTHRES anf LTHRES) in the beginning of the code.
So, this is all with the ADC. I hope you enjoyed reading this. **Please post the comments below for any suggestion, doubt, clarification, etc.**
ATmega16 has an inbuilt 10 bit, 8-channel ADC system. Some of the basic features of Armega16 ADC are:

- 8 Channels.
- 10-bit Resolution.
- Input voltage range of 0 to Vcc.
- Selectable 2.56V of internal Reference voltage source.
- AREF pin for External Reference voltage.
- ADC Conversion Complete Interrupt.

ADC channels in Atmega16 are multiplexed with PORTA and use the common pins (pin33 to pin40) with PORTA. ADC system of Atmega16 microcontroller consists of following pins:

i. ADC0-ADC7: 8 Channels from Pin 40 to Pin 33 of Atmega16 ADC peripheral.

ii. AREF: Pin32 of Atmega16 microcontroller, the voltage on AREF pin acts as the reference voltage for ADC conversion, reference voltage is always less than or equal to the supply voltage, i.e., Vcc.

iii. AVCC: Pin30, this pin is the supply voltage pin for using PORTA and the ADC; AVCC pin must be connected to Vcc (microcontroller supply voltage) to use PORTA and ADC.

Note: External reference voltage source can be used at AREF pin. However, Atmega16 also has internal reference voltage options of 2.56V and Vref = Vcc.

The figure below shows the pin configuration for ADC system of Atmega16 microcontroller.

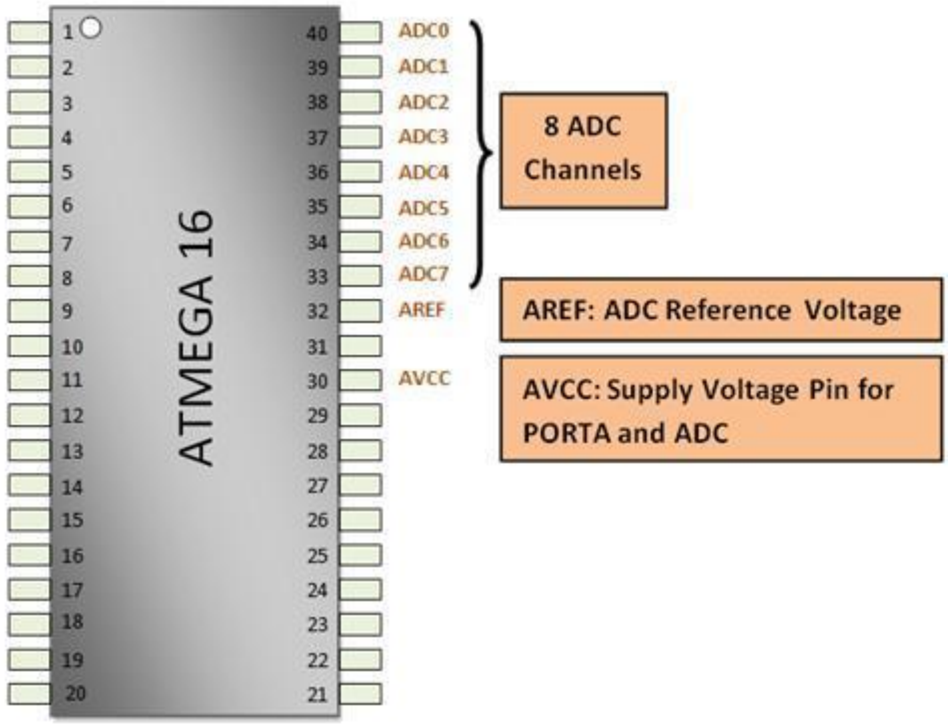

ATMEGA 16
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
ADC6
ADC7
AREF
AVCC
8 ADC Channels
AREF: ADC Reference Voltage
AVCC: Supply Voltage Pin for PORTA and ADC

Figure 46

Chapter2

Schematic and simulation

1. Motor and PWM :

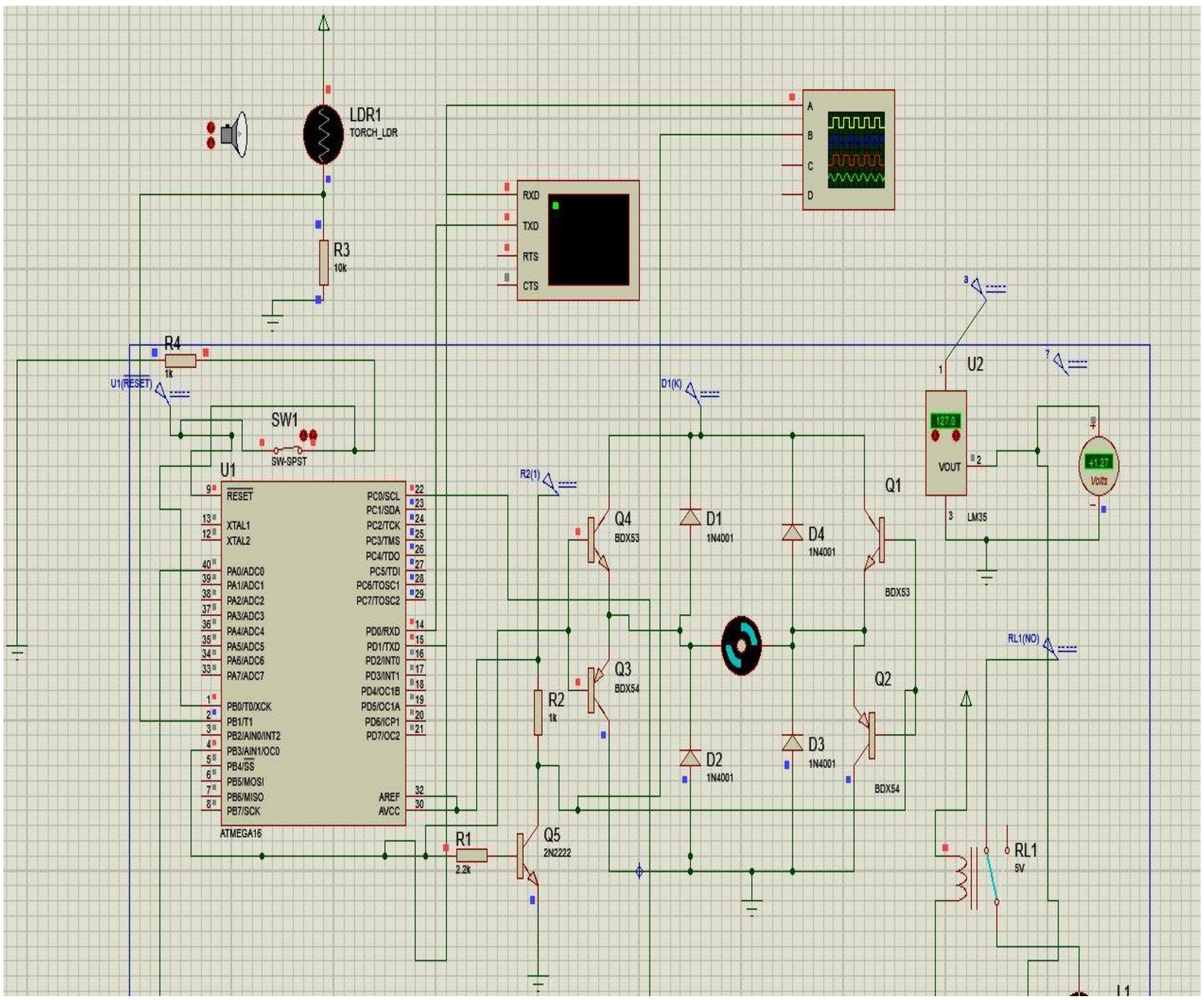


Figure 47

a. When proteus is start:

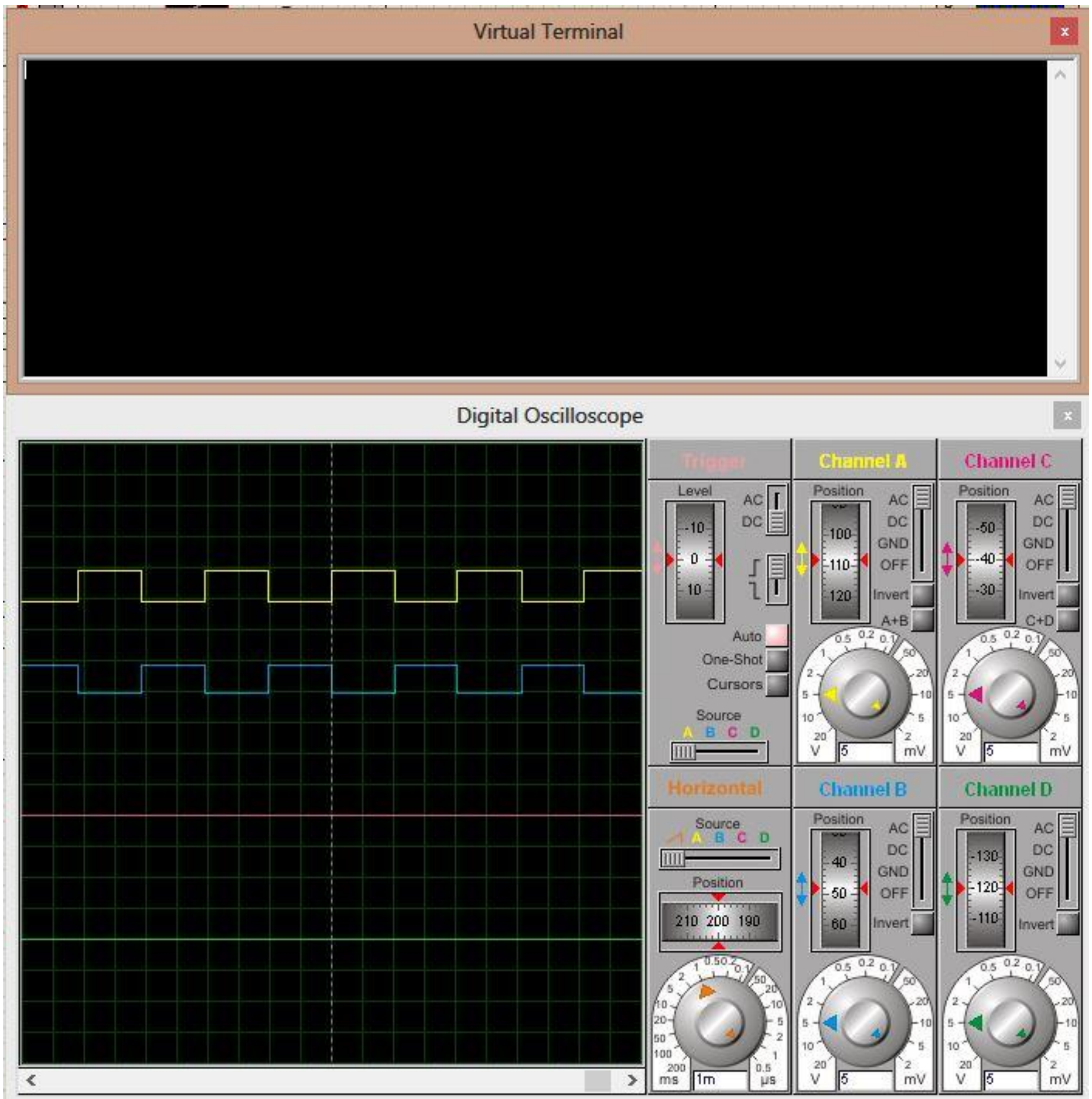


Figure 48

b. Then select tha boud rate :

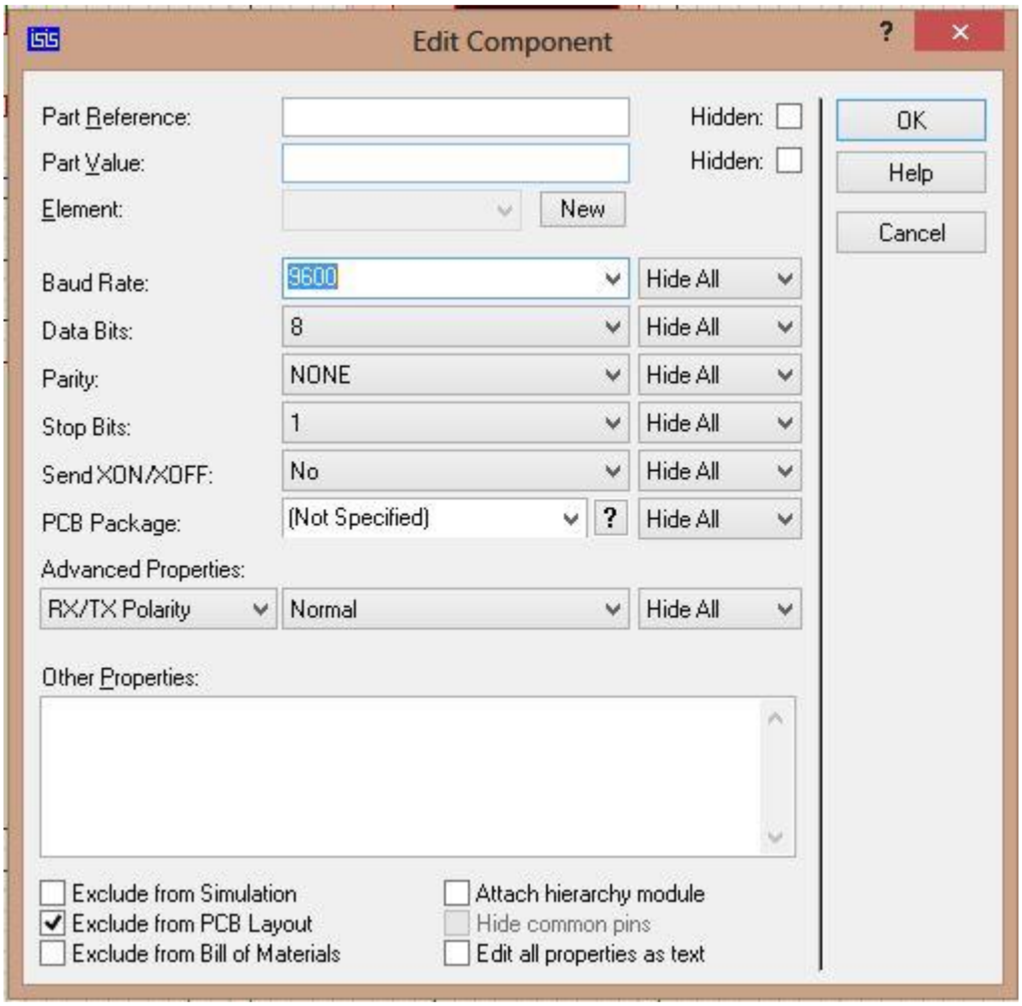


Figure 49

c. When press 2 :

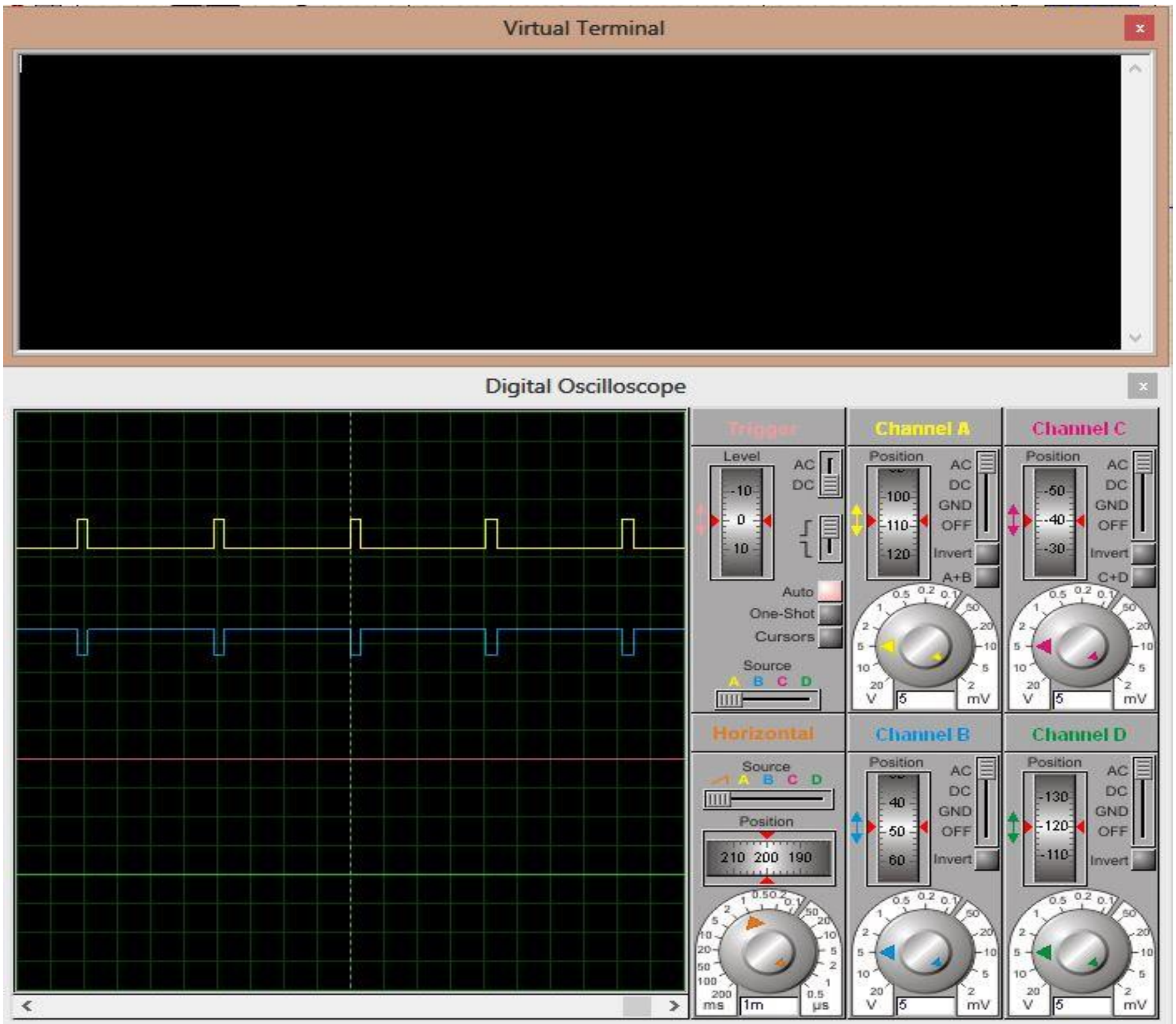


Figure 50

d. When press 3 :

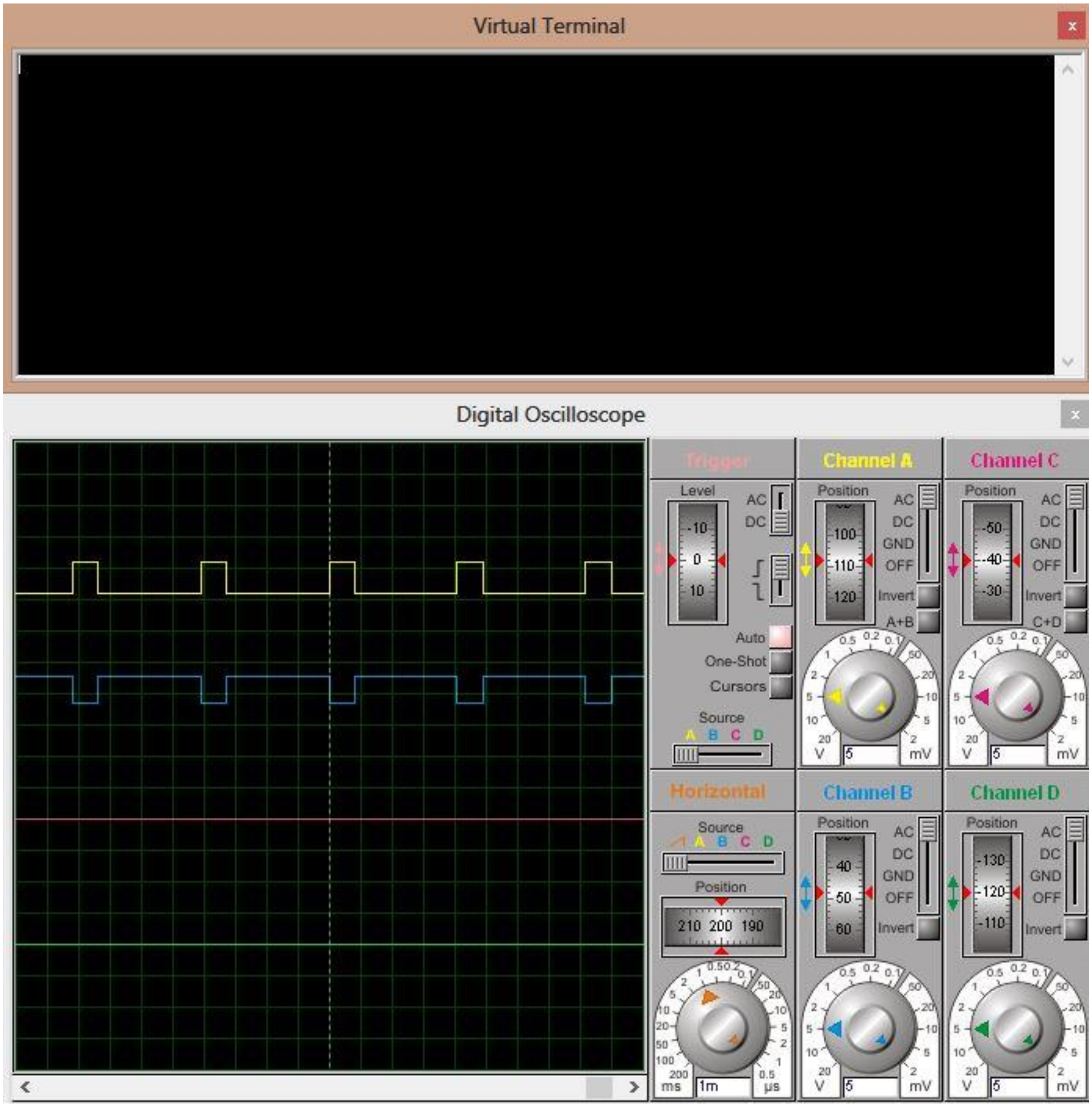


Figure 51

e. When press 4 :

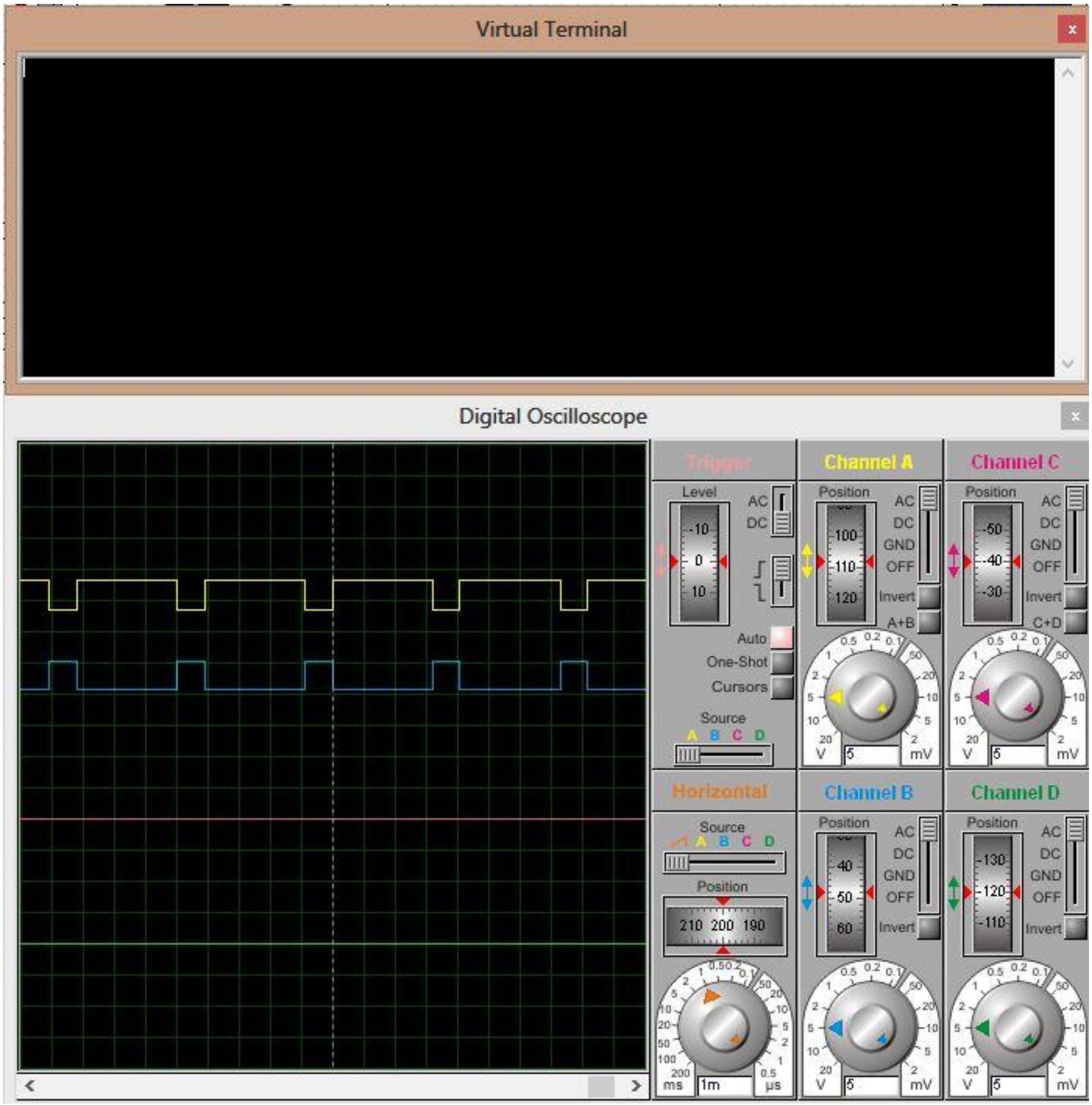


Figure 52

f. When press 5 :

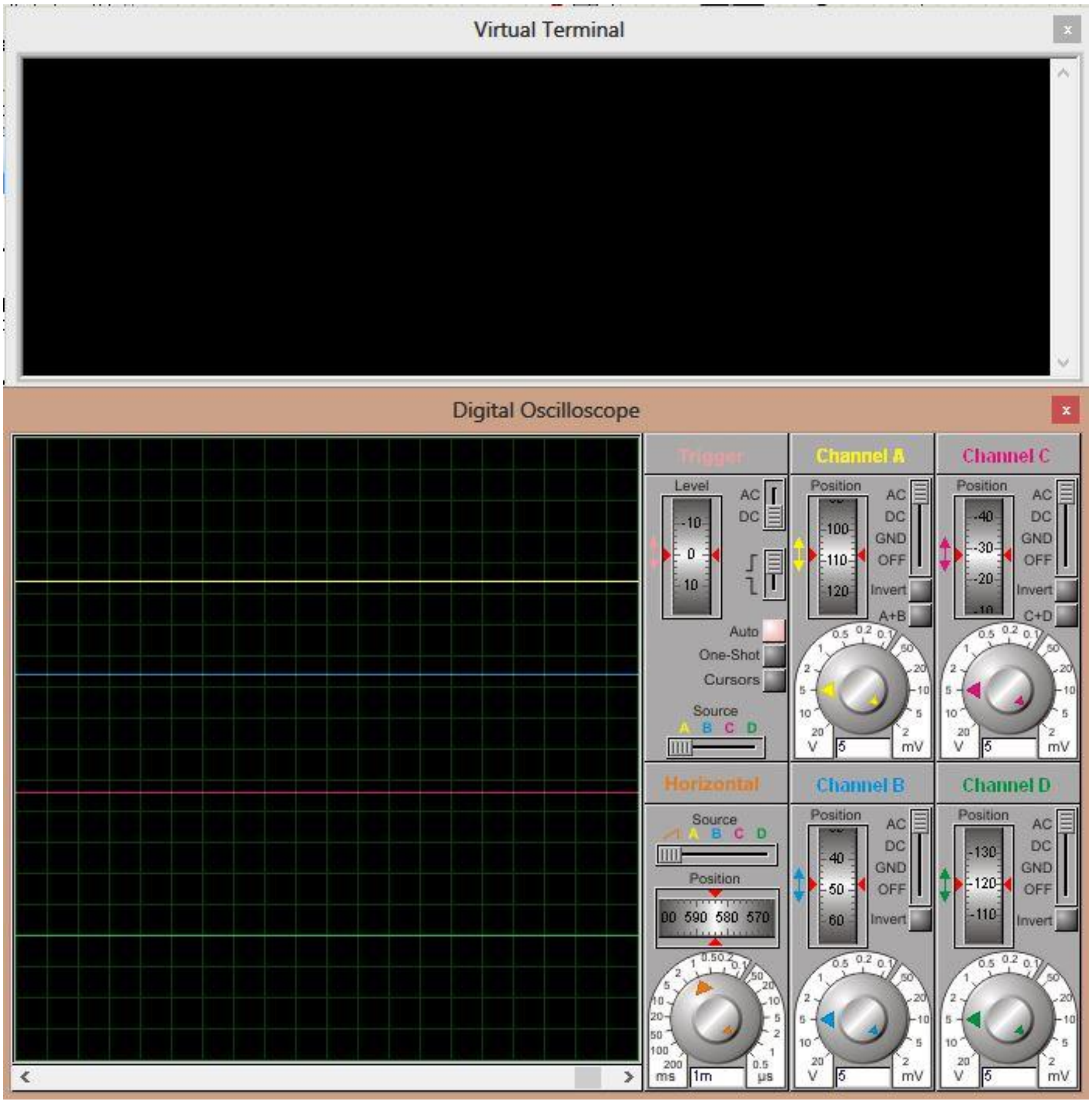


Figure 53

g. When press 6 :

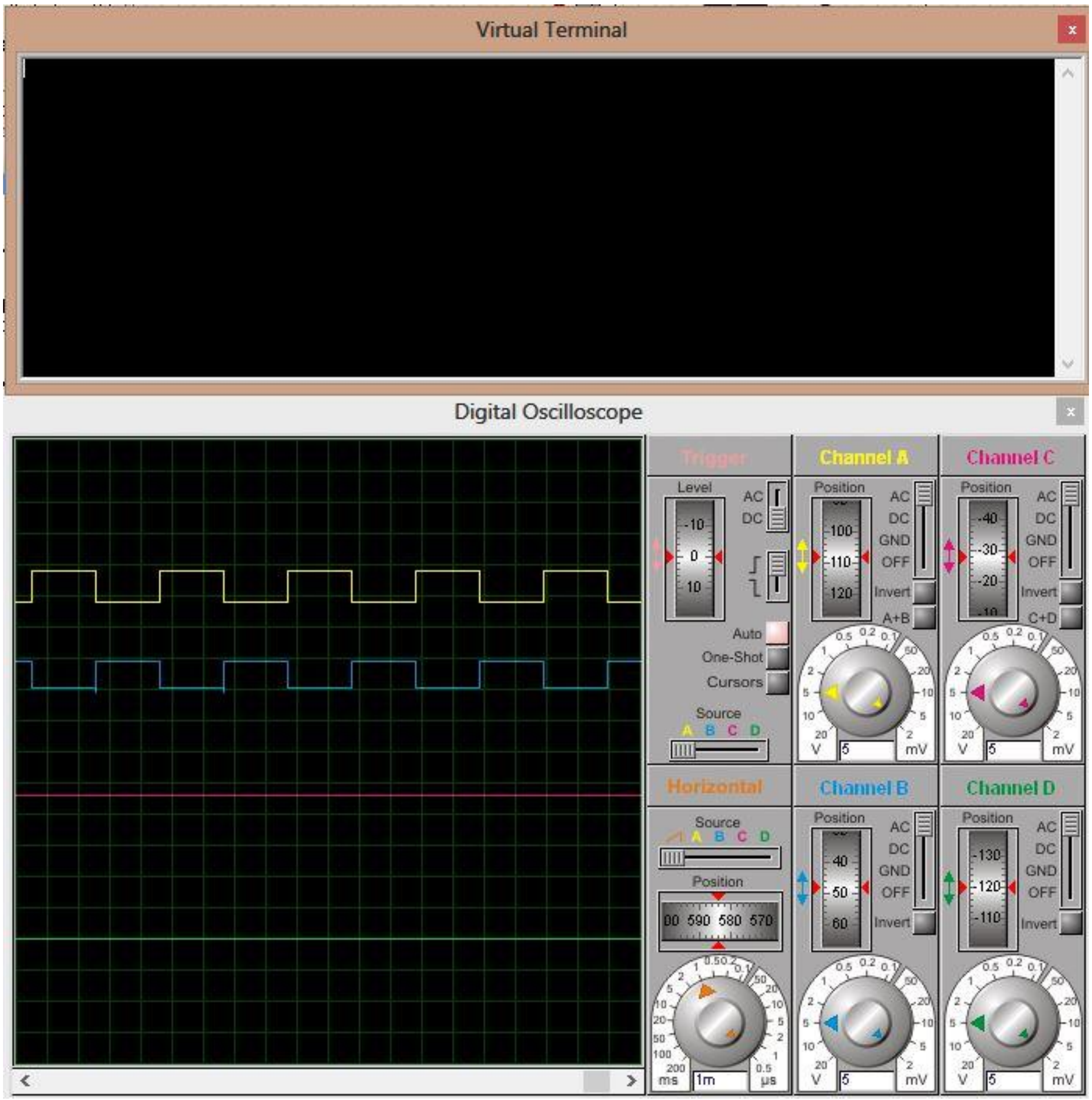


Figure 54

2. Temperature sensor (lm335) :

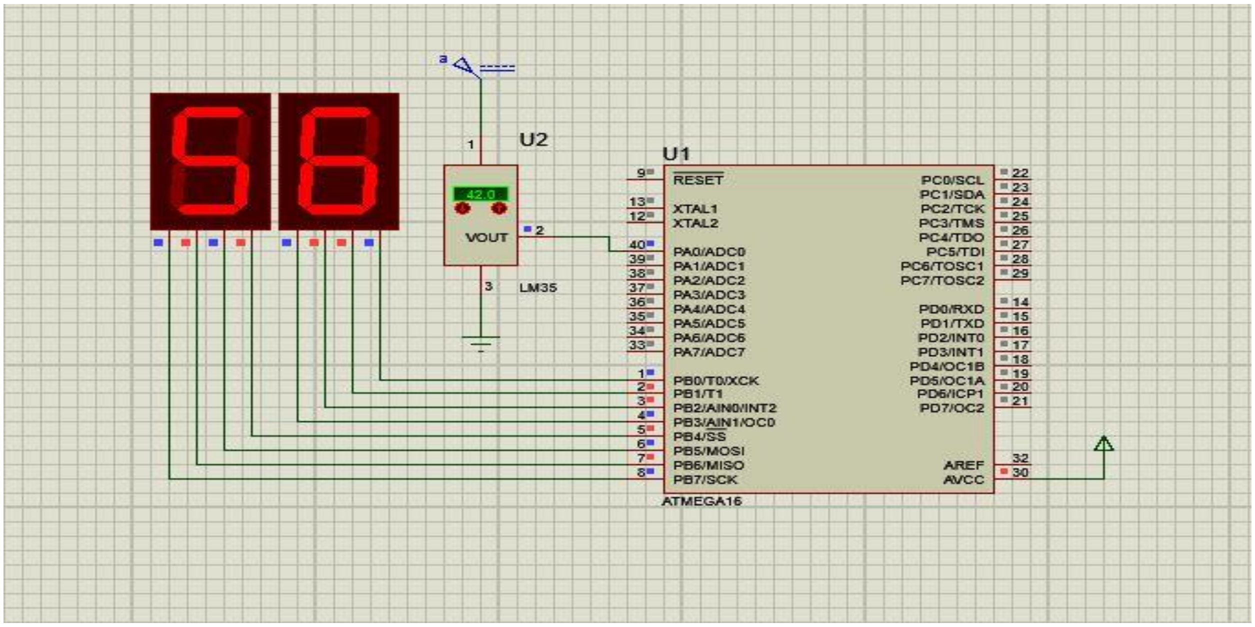


Figure 55

3. Relay :

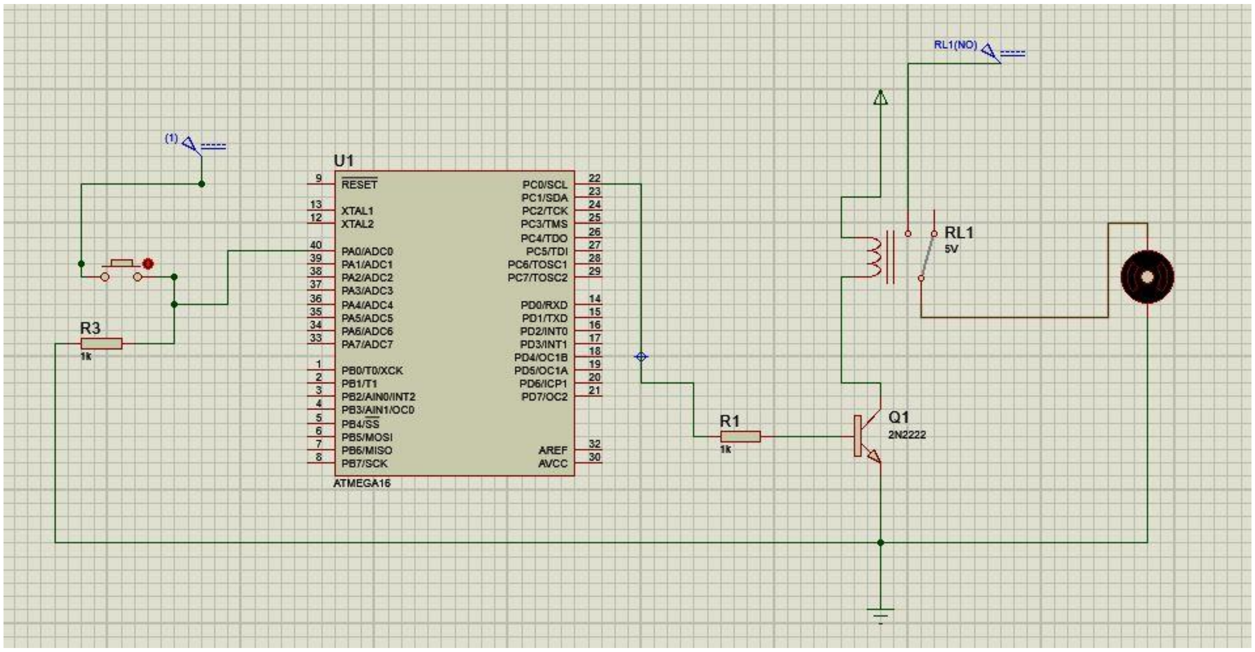


Figure 56

Chapter 3

Practical Work

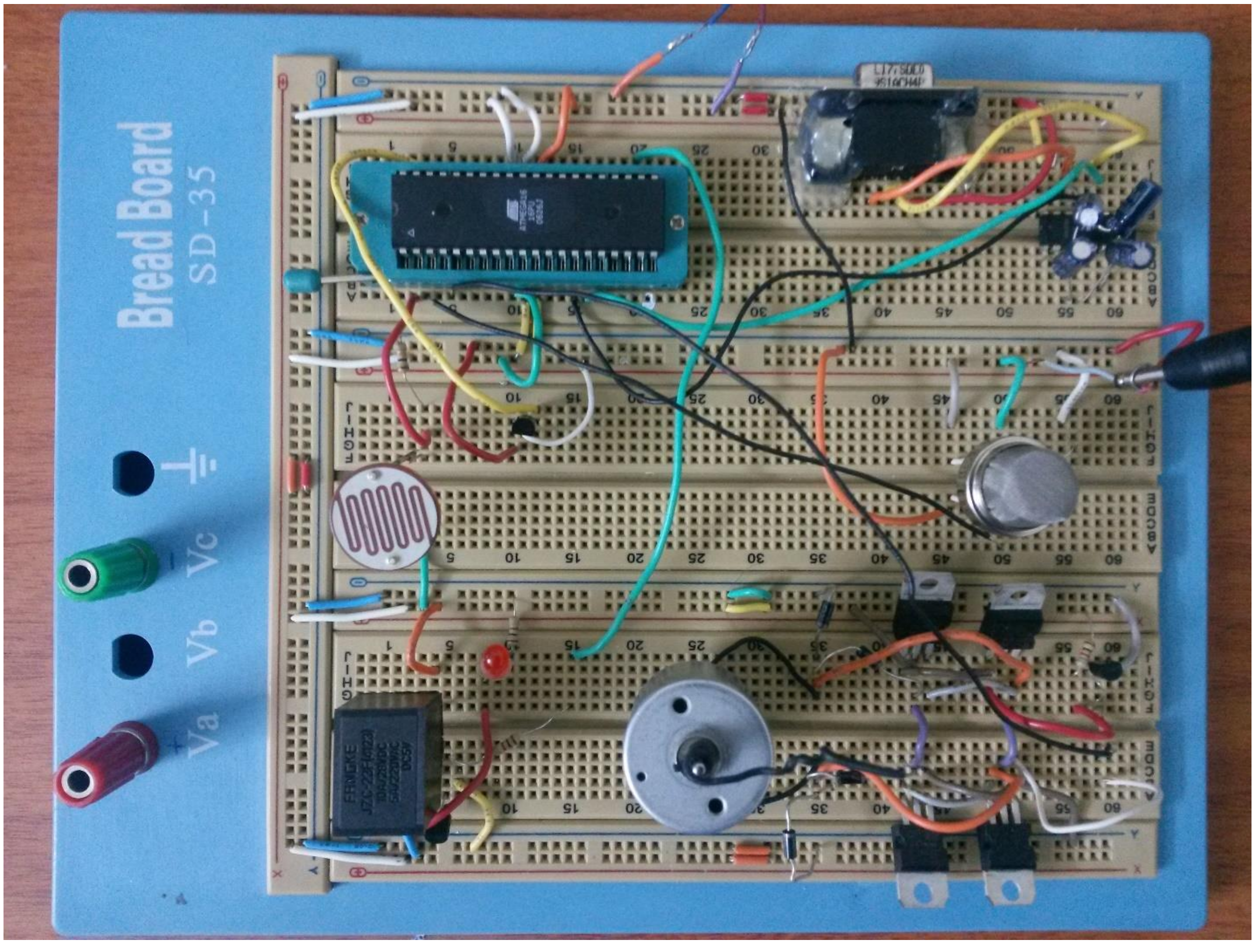


Figure 57

Using Hterm windows application for monitoring serial ports

1. The 1st situation:

   when we send “1” through Serial port the microcontroller replays:

   a. if the LDR not covered and there is no Gas on the Gas Sensor

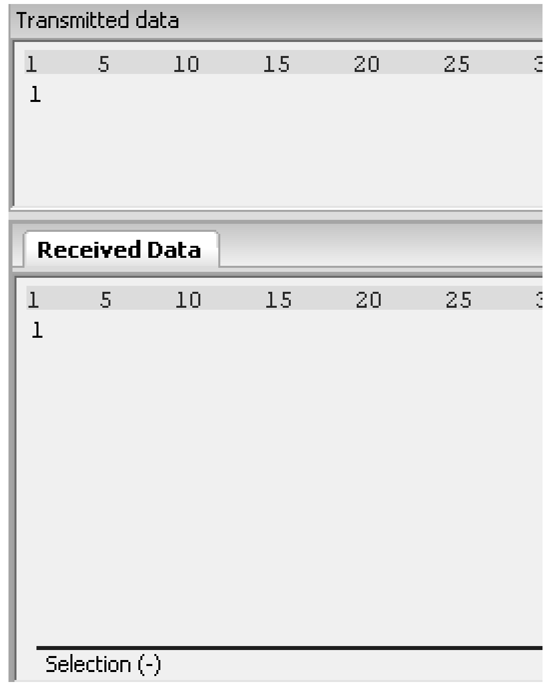


Figure 58

   b. if the LDR is covered and there is no Gas on the Gas sensor

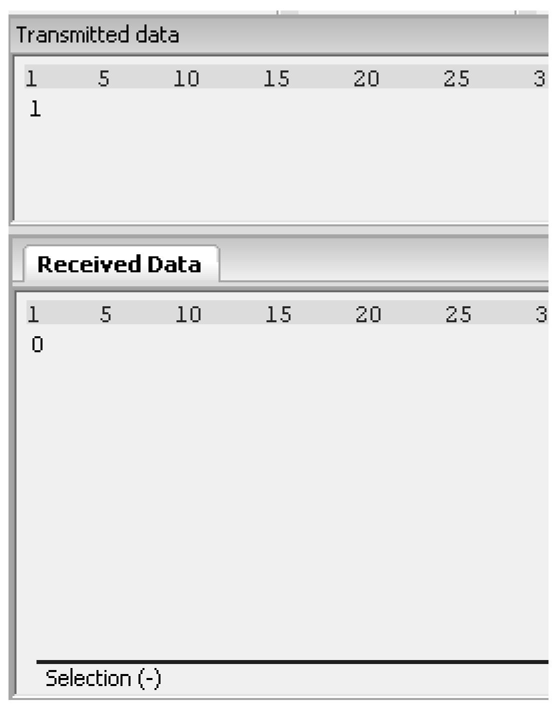


Figure 59

c. if the LDR is covered and there is some Gas on the Gas Sensor

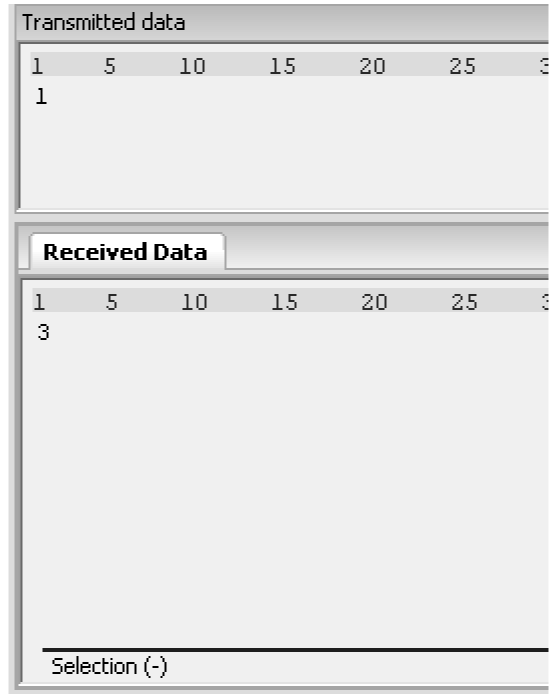


Figure 60

d. if the LDR is not covered and there is some Gas on the Gas Sensor

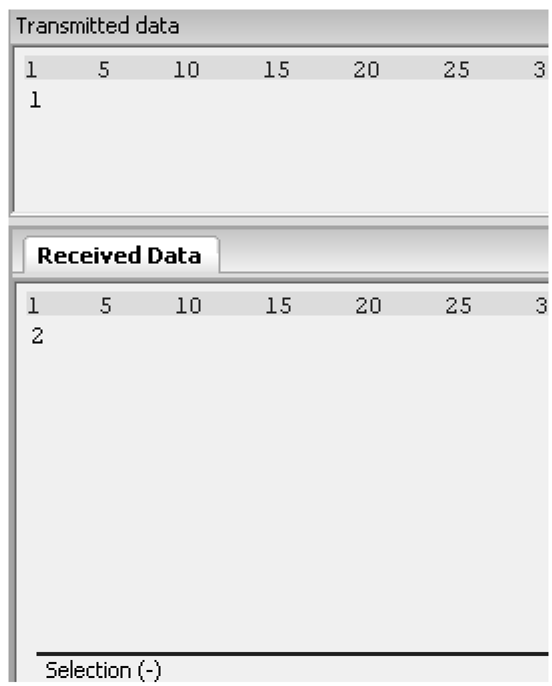
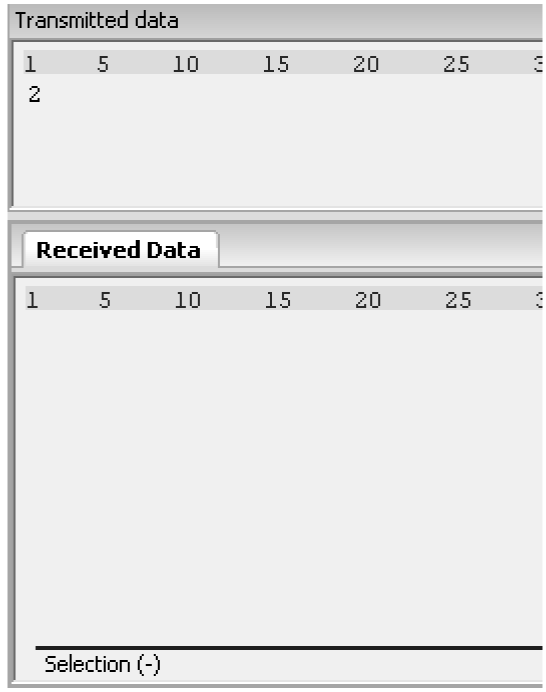


Figure 61

2. The 2nd Situation:

When we send “2” through Serial port the microcontroller moves the motor CW with a high speed:



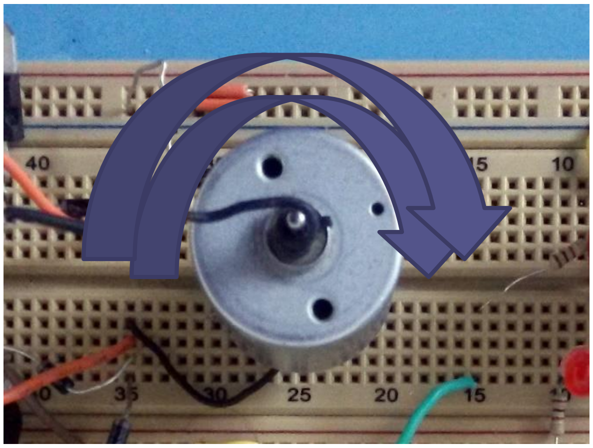
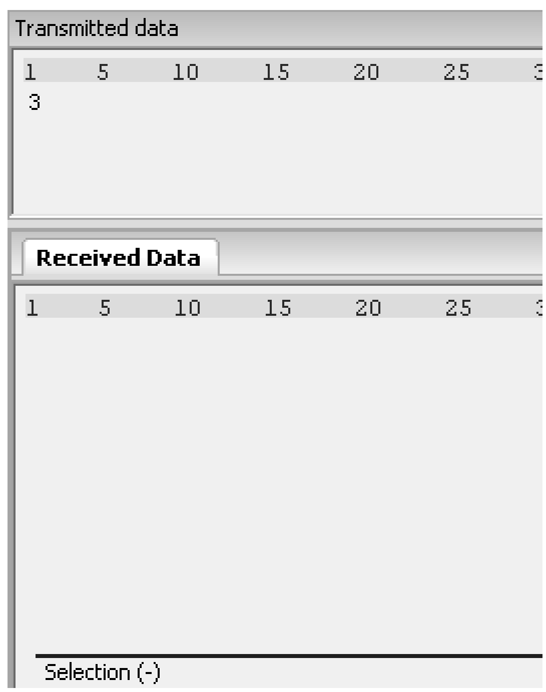


Figure 62

3. The 3rd Situation:

When we send “3” through Serial port the microcontroller moves the motor CW with a low speed:



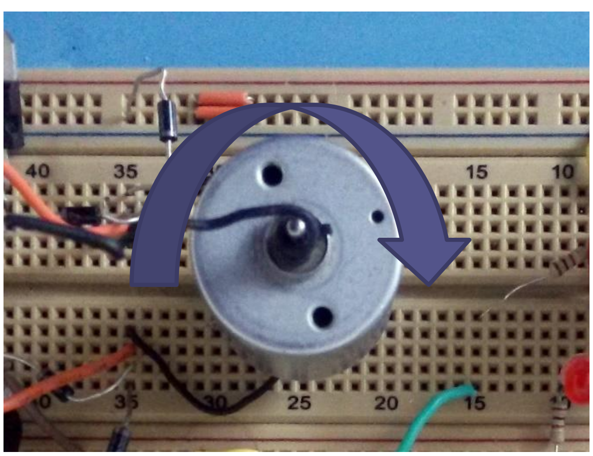
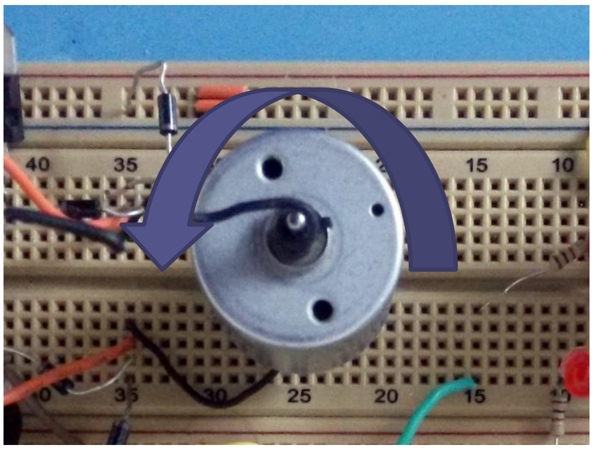


Figure 63

4. ## The 4th Situation:

When we send “4” through Serial port the microcontroller moves the motor CCW with a low speed:



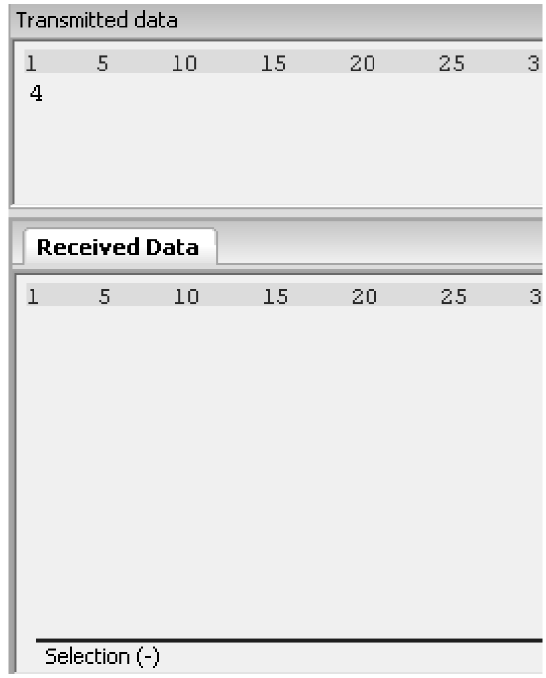
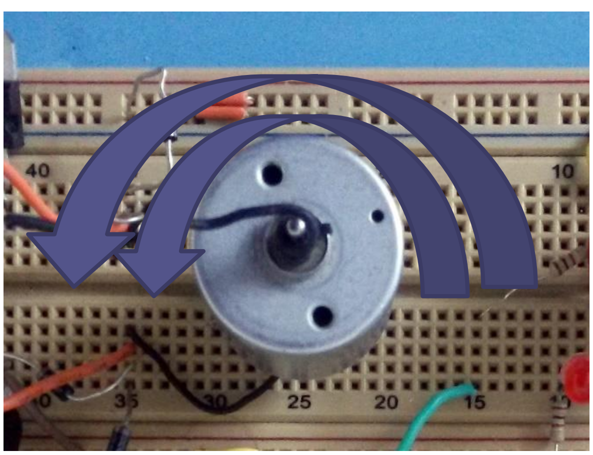


Figure 64

5. ## The 5th Situation:

When we send “5” through Serial port the microcontroller moves the motor CCW with a high speed:



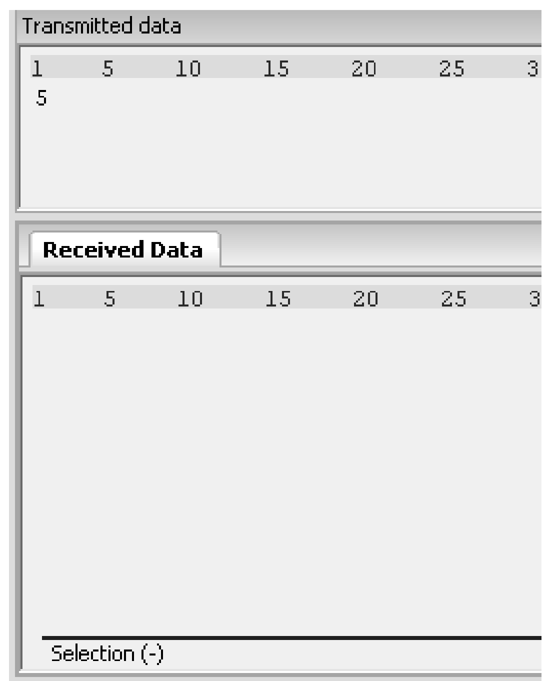


Figure 65

6. The 6th Situation:

When we send "6" through Serial port the microcontroller Stops the motor:

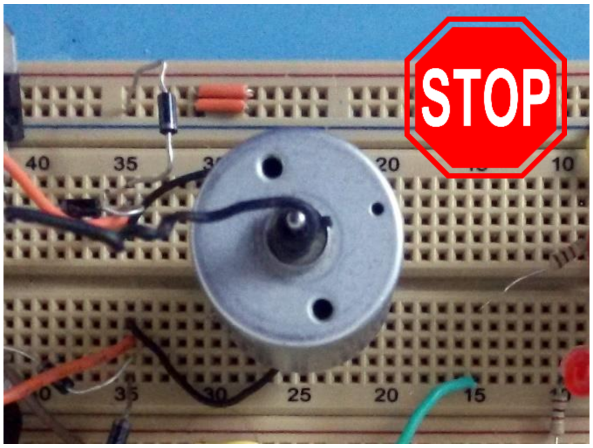
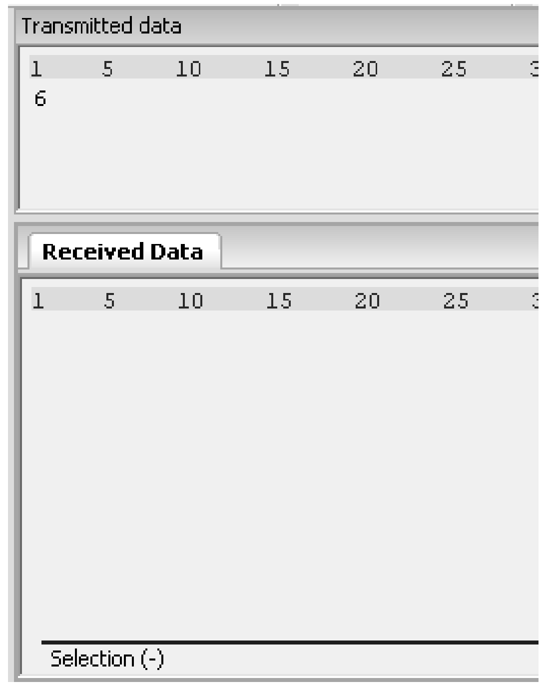


Figure 66



7. The 7th Situation:

When we send "7" through Serial port the microcontroller turns on the Relay:

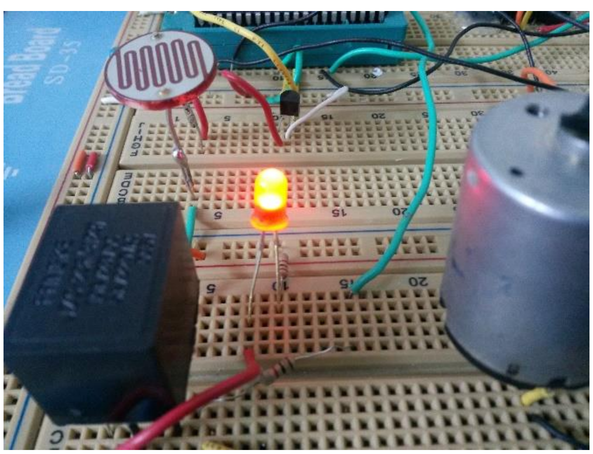
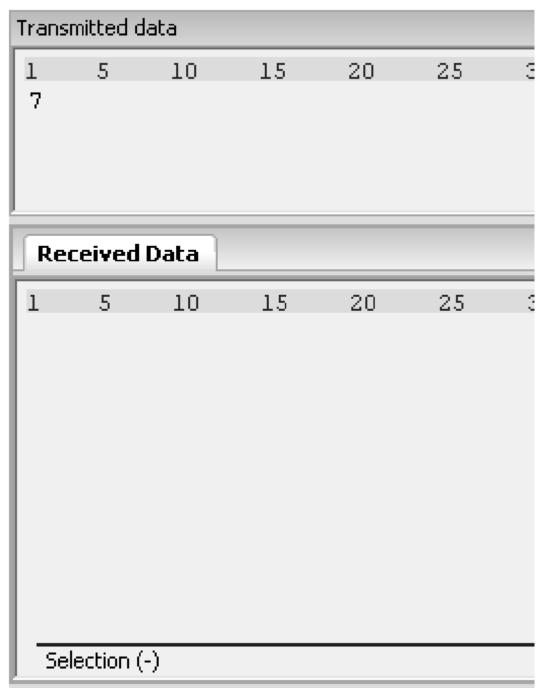


Figure 67



8. The 8th Situation:

When we send "8" through Serial port the microcontroller turns off the Relay:

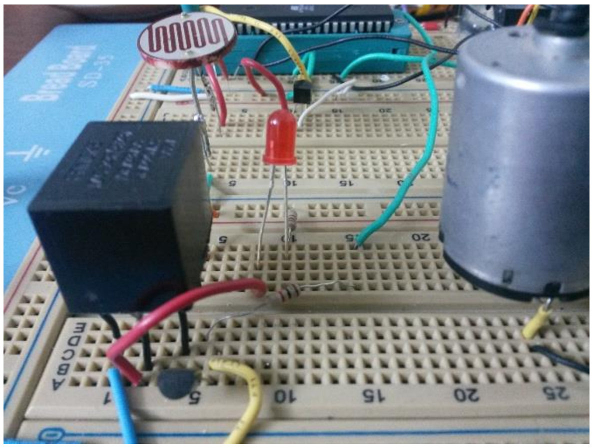
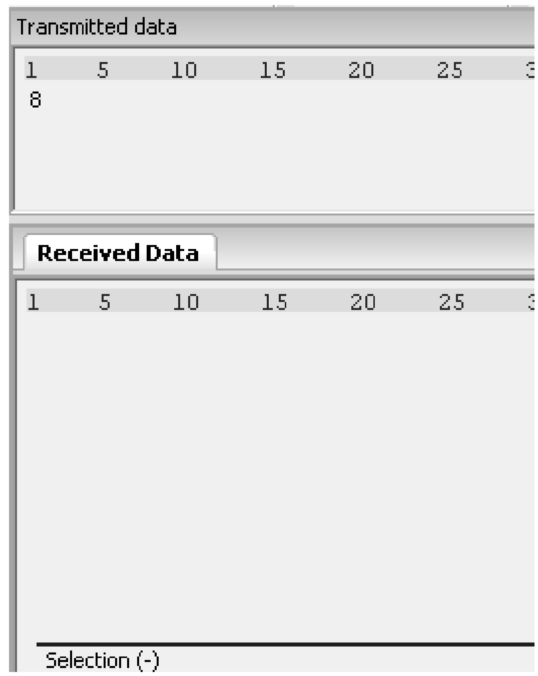


Figure 68



9. The 9th Situation:

When we send "9" through Serial port the microcontroller forwards the value of the temperature sensor:

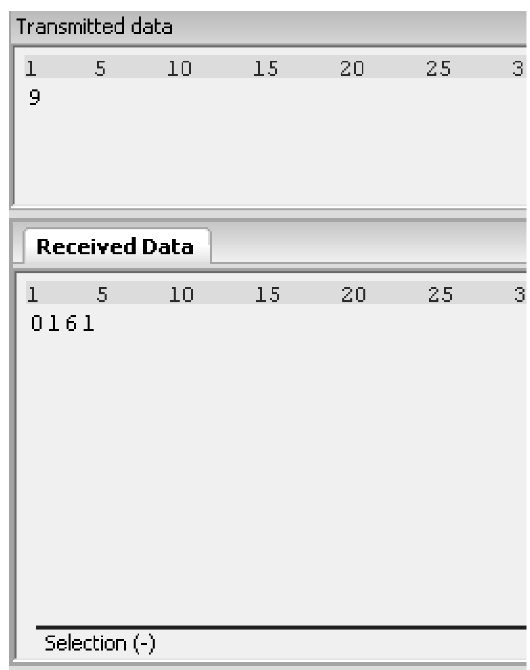


Figure 69

Using an android device with a Custom application Developed by Wael hamada

1. The 2nd Situation:

When we send “2” through Serial port the microcontroller moves the motor CW with a high speed:

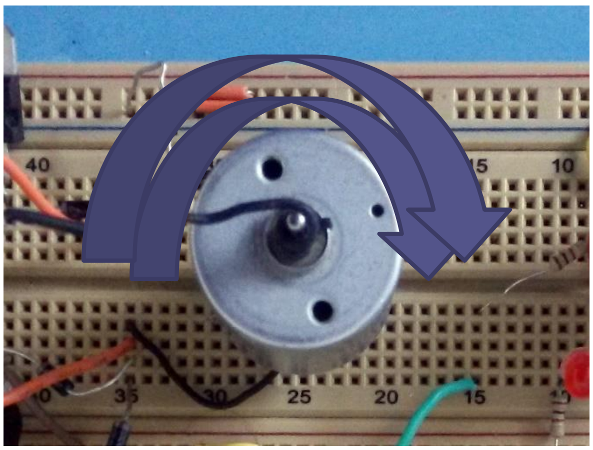
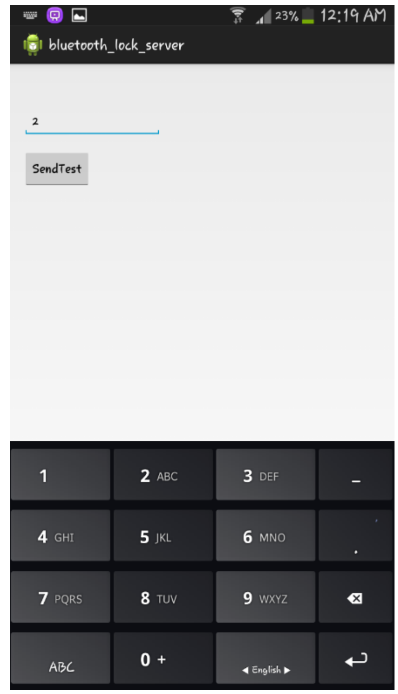
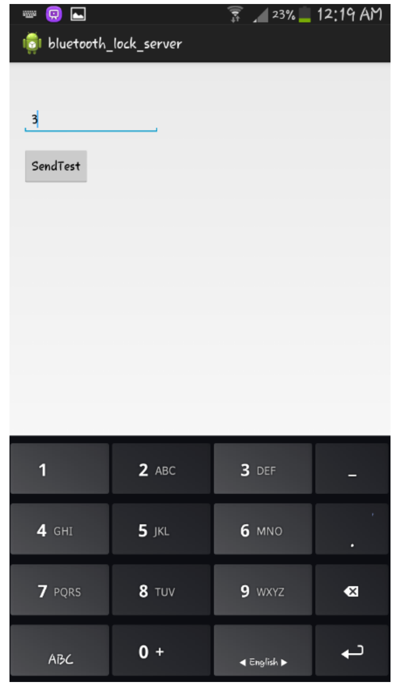


Figure 70



2. The 3rd Situation:

When we send "3" through Serial port the microcontroller moves the motor CW with a low speed:



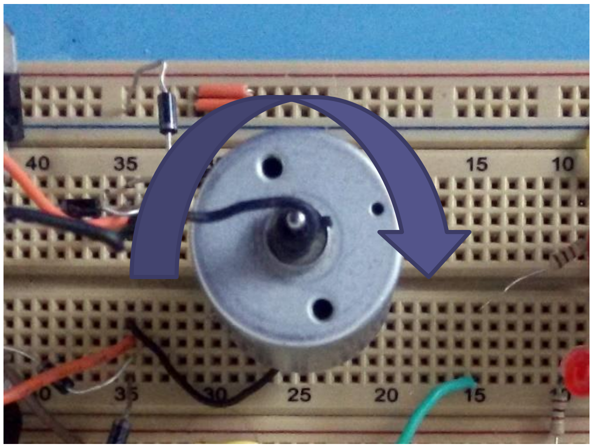
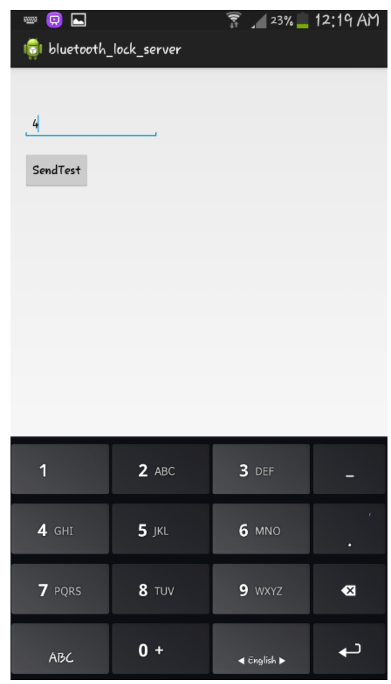


Figure 71

3. The 4th Situation:

When we send "4" through Serial port the microcontroller moves the motor CCW with a low speed:



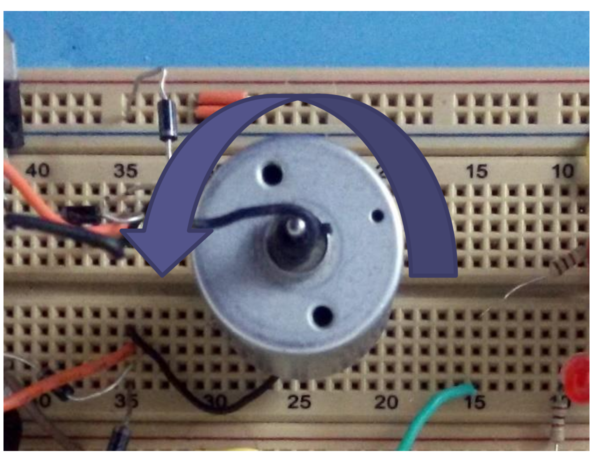
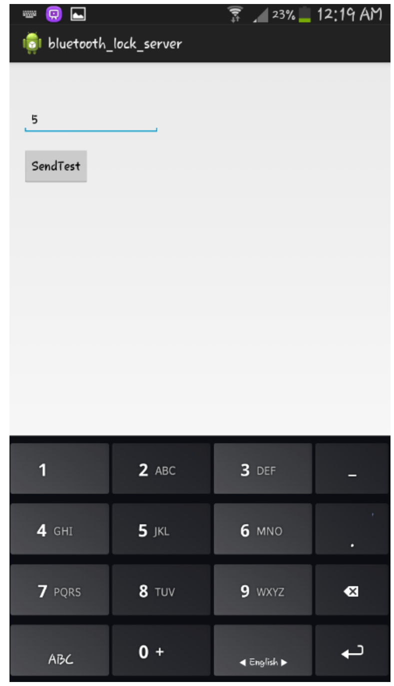


Figure 72

4. The 5th Situation:

When we send "5" through Serial port the microcontroller moves the motor CCW with a high speed:



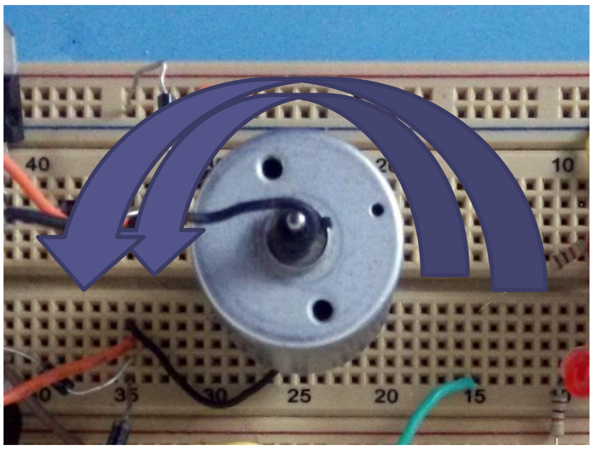
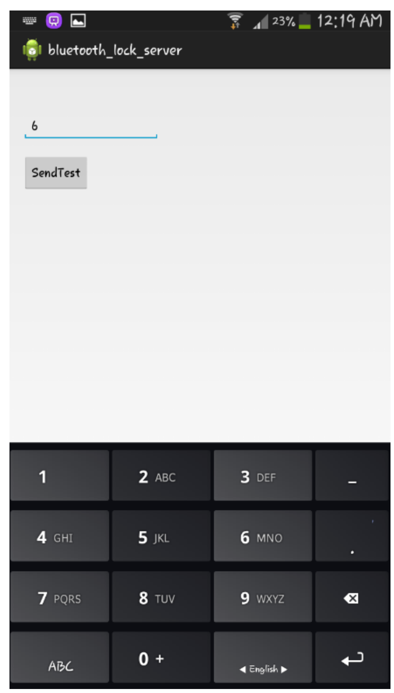


Figure 73

5. The 6th Situation:

When we send "6" through Serial port the microcontroller Stops the motor:



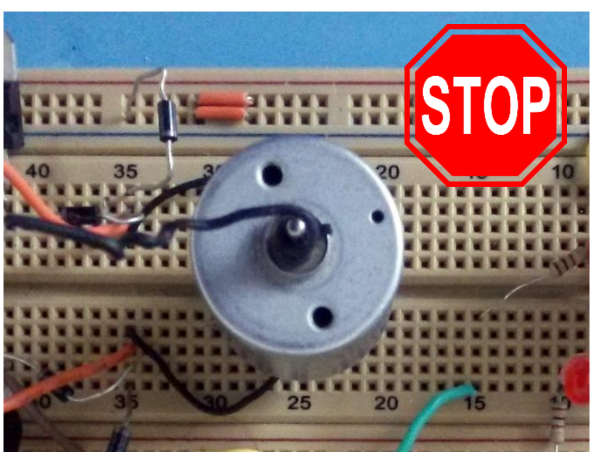
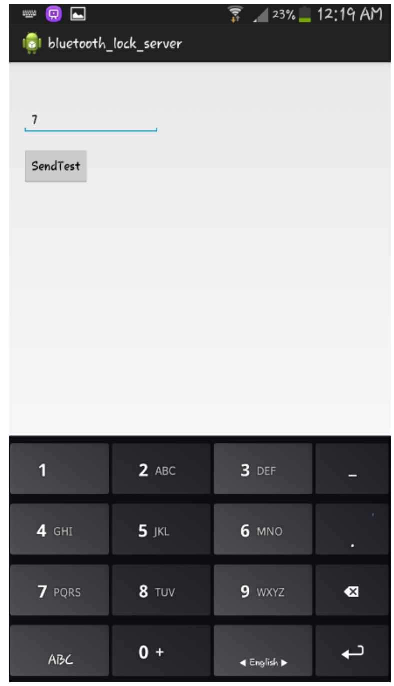


Figure 74

6. The 7th Situation:

When we send “7” through Serial port the microcontroller turns on the Relay:



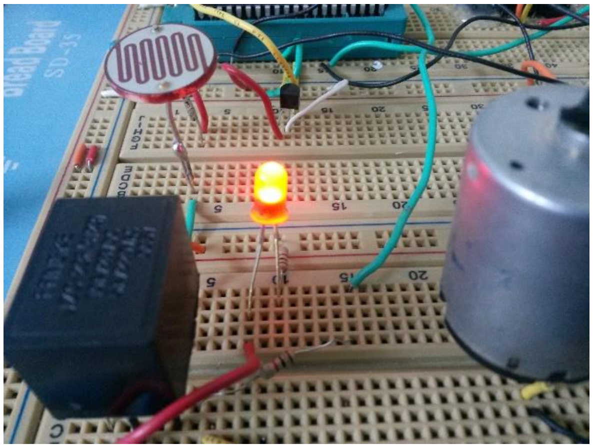
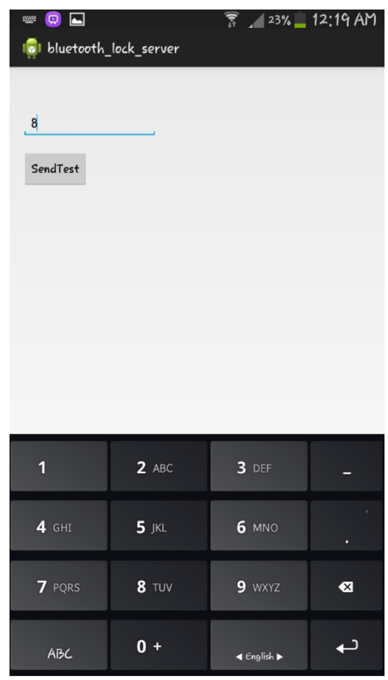


Figure 75

7. The 8th Situation:

When we send “8” through Serial port the microcontroller turns off the Relay:



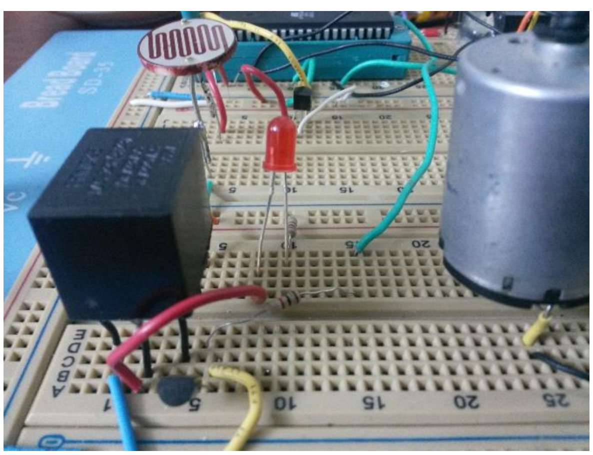


Figure 76

SOFTWARE ENGINEERING SECTION:

Chapter 4

1), Android Overview:

Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).

Figure 77

Google wanted Android to be open and free; hence, most of the Android code was released under the open source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others. This simple development model makes Android very attractive and has thus piqued the interest of many vendors. This has been especially true for companies affected by the phenomenon of Apple’s iPhone, a hugely successful product that revolutionized the smartphone industry. Such companies include Motorola and Sony Ericsson, which for many years have been developing their own mobile operating systems. When the iPhone was launched, many of these manufacturers had to scramble to find new ways of revitalizing their products. These manufacturers see Android as a solution — they will continue to design their own hardware and use Android as the operating system that powers it.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain. Device manufacturers therefore see Android as their best hope to challenge the onslaught of the iPhone, which already commands a large base of applications.

2), Required device features:

- USB On-The-Go
- Bluetooth connectivity
- GSM (2G) enabled (Optional)

USB On-The-Go in brief:

Often abbreviated USB OTG or just OTG, is a specification that allows USB devices such as digital audio players or mobile phones to act as a host, allowing other USB devices like a USB flash drive, mouse, or keyboard to be attached to them. Unlike conventional USB systems, USB OTG systems can drop the hosting role and act as normal USB devices when attached to another host. This can be used to allow a mobile phone to act as host for a flash drive and read its contents, downloading music for instance, but then act as a flash drive when plugged into a host computer and allow the host to read data from the device. However, USB-OTG uses three new communication protocols, Attach Detection Protocol (ADP), Session Request Protocol (SRP) and Host Negotiation Protocol (HNP).

Figure 78

Bluetooth in brief:

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength radio transmissions in the ISM band from 2400–2480 MHz) from fixed and mobile devices, creating personal area networks (PANs) with high levels of security. Created by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables. It can connect several devices, overcoming problems of synchronization.

Bluetooth is managed by the Bluetooth Special Interest Group, which has more than 18,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics. Bluetooth was standardized as IEEE 802.15.1, but the standard is no longer maintained. The SIG oversees the development of the specification, manages the qualification program, and protects the trademarks. To be marketed as a Bluetooth device, it must be qualified to standards defined by the SIG. A network of patents is required to implement the technology, which is licensed only for that qualifying device.

Figure 79

GSM (2G):

Second generation 2G cellular telecom networks were commercially launched on the GSM standard in Finland by Radiolinja (now part of Elisa Oyj) in 1991 Three primary benefits of 2G networks over their predecessors were that phone conversations were digitally encrypted; 2G systems were significantly more efficient on the spectrum allowing for far greater mobile phone penetration levels; and 2G introduced data services for mobile, starting with SMS text messages. 2G network allows for much greater penetration intensity. 2G technologies enabled the various mobile phone networks to provide the services such as text messages, picture messages and MMS (multimedia messages). All text messages sent over 2G are digitally encrypted, allowing for the transfer of data in such a way that only the intended receiver can receive and read it.

Figure 80

3), Use Case Diagram:

Use Case Name:	Start	Author: Wael hamada
Use Case ID:	UCID1	
Priority:	High	
Primary System Actor:	User	
Prerequisite:	The system connected to the controller via USB-to-Serial converter cable	
Description:	This use case initialize the system components and ports	
Trigger:	This use case initiated when the user power on the system	
Flow of events:	Actor Action	System Response
	1: the user turn on the system	2: turn on android OS 3: configure USB ports 4: establish the connection to the Controller 5: turn on Bluetooth connectivity and set adapter to Discoverable 6: start listening for incoming Bluetooth connection 7: start listening for incoming SMS message 8: start listening for incoming voice call

Use Case Name:	Incoming Bluetooth connection	Author: Wael hamada
Use Case ID:	UCID2	
Priority:	High	
Primary System Actor:	Mobile phone with Bluetooth connectivity and system application installed on it	
Prerequisite:	UCID1	
Description:	This Use case checks the incoming Bluetooth connection	
Trigger:	This use case initiated when a mobile phone request a connection to the system via Bluetooth	
Flow of events:	Actor Action	System Response
	1:mobile phone request a Bluetooth connection	
		2:the system checks the request
		3:the system start the connection session if the device is one of the paired devices
		4:the system forwards the command to the controller

Use Case Name:	Incoming SMS message	Author: Wael hamada
Use Case ID:	UCID3	
Priority:	High	
Primary System Actor:	Mobile phone with GSM connectivity	
Prerequisite:	UCID1	
Description:	This use case checks the incoming SMS message	
Trigger:	This use case initiated when a SMS message received	
Flow of events:	Actor Action	System Response
	1:a command sent in SMS message to from a mobile phone	
		2:the system checks the SMS sender number and command
		3:the system forwards the command to the controller

Use Case Name:	Incoming voice call	Author: Wael hamada
Use Case ID:	UCID4	
Priority:	High	
Primary System Actor:	Mobile phone with GSM connectivity	
Prerequisite:	UCID1	
Description:	This use case checks the incoming voice call	
Trigger:	This use case initiated when a voice call received	
Flow of events:	Actor Action	System Response
	1:calling the system via a regular voice call	
		2:the system checks the incoming voice call callers ID
		3: the system sends a command to the controller

4), Sequence Diagrams:

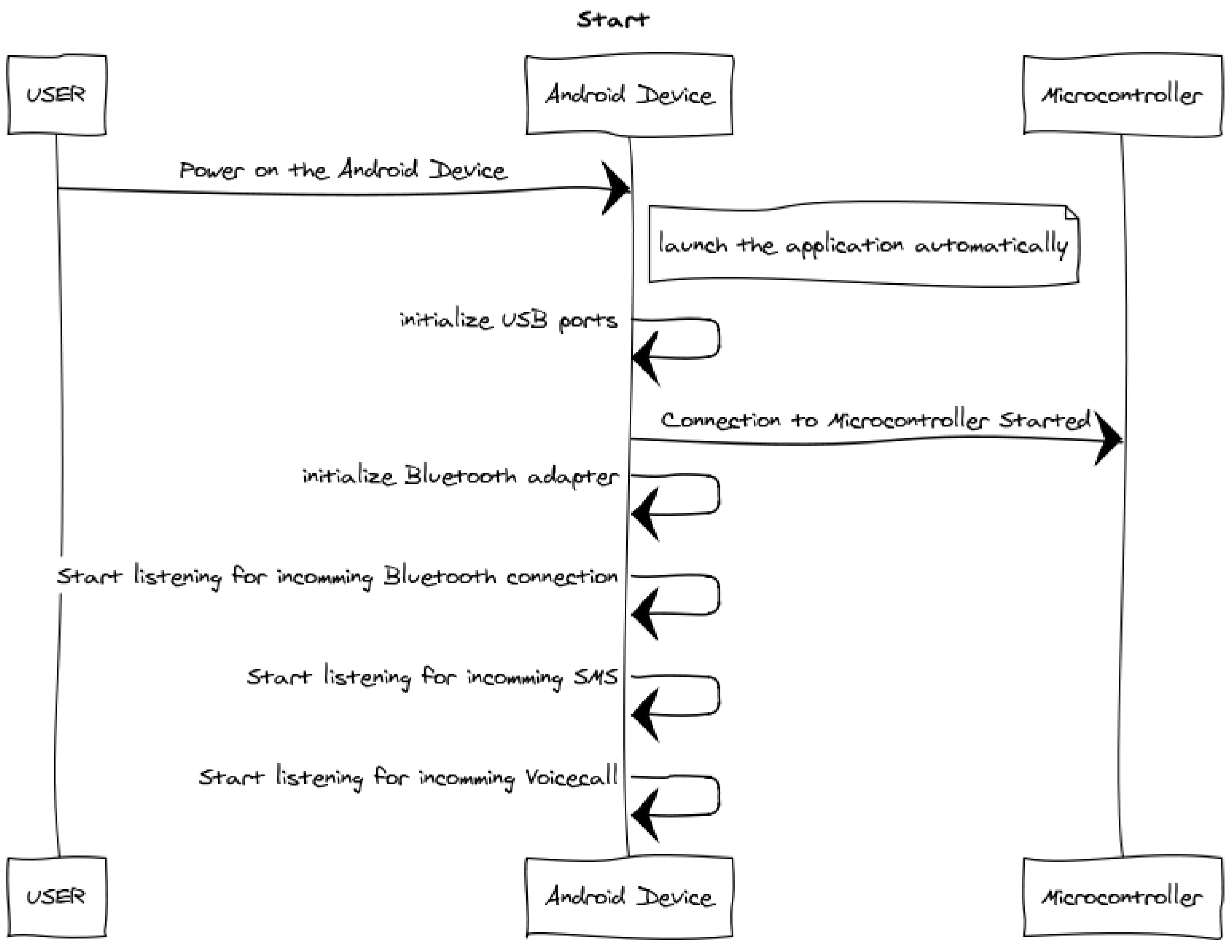


Figure 81

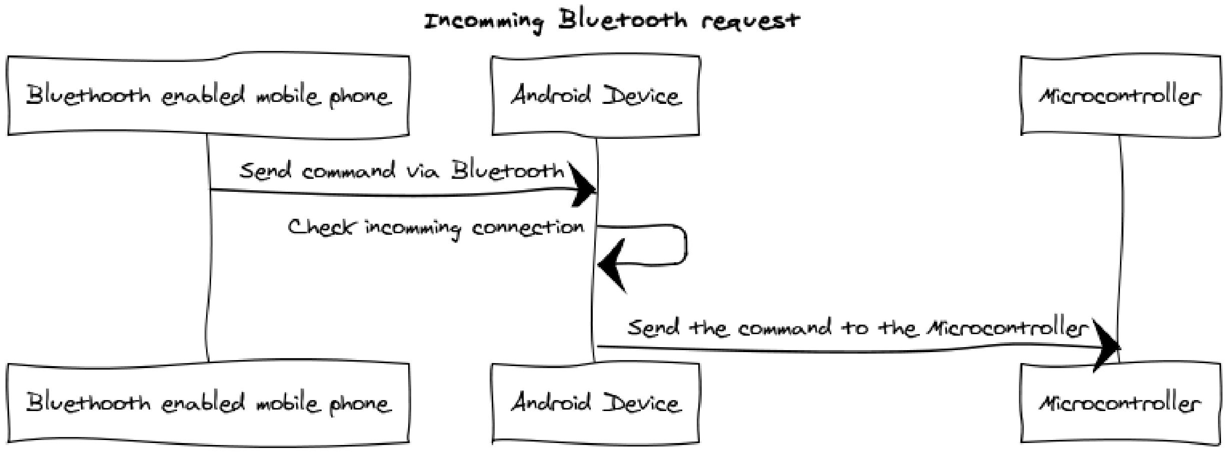


Figure 82

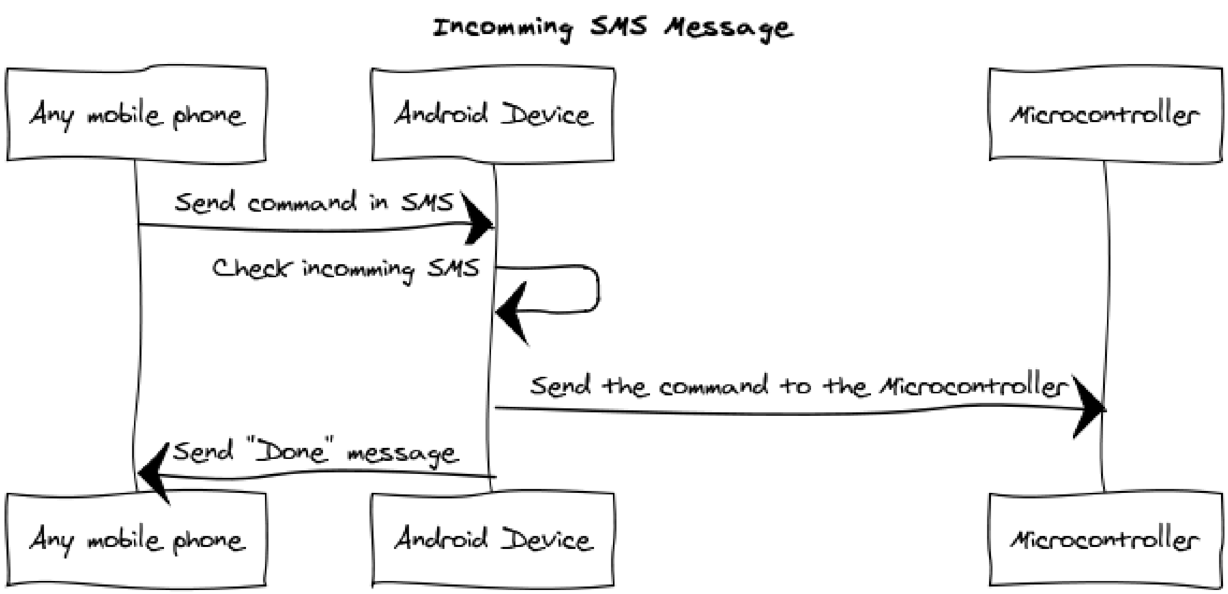


Figure 83

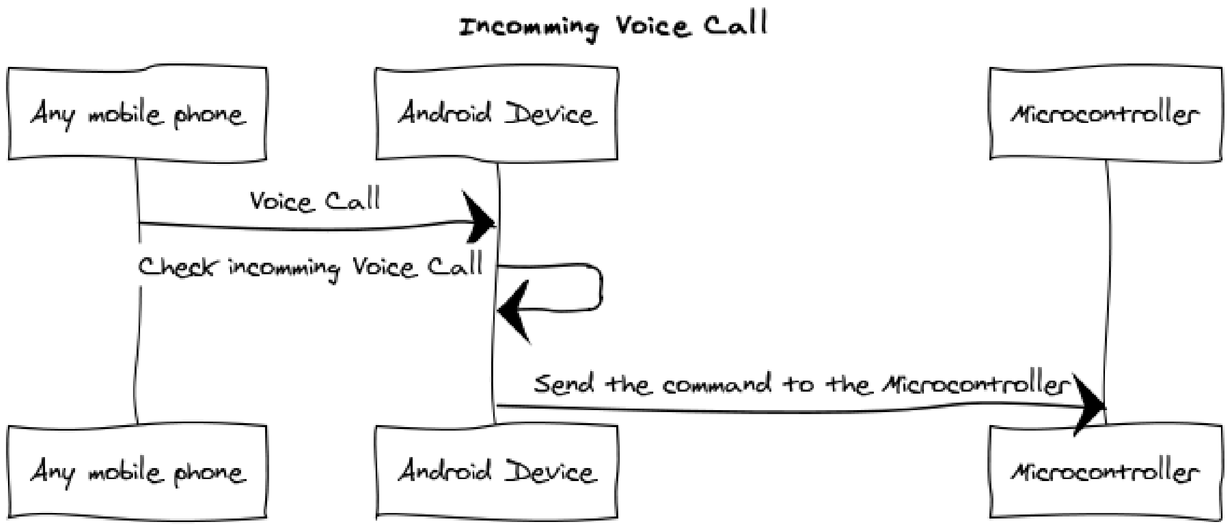

Incomming Voice Call
Any mobile phone
Android Device
Microcontroller
Voice Call
Check incomming Voice Call
Send the command to the Microcontroller
Any mobile phone
Android Device
Microcontroller

Figure 84

5), Application development:

The android project we are going to write is going to have to do a few things:

1), open a Bluetooth connection.

2), Listen for incoming data.

3), Send data to AtMega16A through USB-to-Serial cable.

4), Listen for incoming SMS.

5), Close the connection.

But before we can do any of that we need to take care of a few pesky little details. First, we need to pair the two android devices. You can do this by opening the application drawer on both devices then going to settings. From there open Wireless and network. Then Bluetooth settings.

Next we need to tell android that we will be working with Bluetooth by adding those two elements to the <manifest> tag inside AndroidManifest.XML file:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />
```

Now that we have that stuff out of the way we can get on with opening the Bluetooth connection. To get started we need a BluetoothAdapter reference from Android. We can get that by calling

```
BluetoothAdapter.getDefaultAdapter();
```

The return value of this will be null if the device does not have Bluetooth capabilities. With the adapter you can check to see if Bluetooth is enabled on the device and request that if not with this code:

```
if(!mBluetoothAdapter.isEnabled())
{
Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBluetooth, 0);
}
```

Now that we have the Bluetooth adapter and we know that it's turned on we can get a reference to out Client's Bluetooth device with this code:

```
Set pairedDevices = mBluetoothAdapter.getBondedDevices();
if(pairedDevices.size() > 0)
{
for(BluetoothDevice device : pairedDevices)
{
if(device.getName().equals("Clientbluetoothlock")) //Note, you will need
to change this to match the name of your device
{
mmDevice = device;
break;}}
```

Armed with the Bluetooth device reference we can now connect to it using this code:

```
UUID uuid = UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66"); //insecure UUID

mmSocket = mmDevice.createRfcommSocketToServiceRecord(uuid);

mmSocket.connect();

mmOutputStream = mmSocket.getOutputStream();

mmInputStream = mmSocket.getInputStream();
```

With this we have one way connection established .let start implementing our serial connection stuff.

First we need to make sure that the android API we are using is Level 12 or higher. Because The USB host APIs are not present on earlier API levels by adding the following element to the <manifest> tag in the AndroidManifest.xml file:

```
<uses-sdk android:minSdkVersion="12"/>
```

And Because not all Android-powered devices are guaranteed to support the USB host APIs, include a <uses-feature> element that declares that your application uses the android.hardware.usb.host feature by adding the following element:

```
<uses-feature android:name="android.hardware.usb.host"/>
```

We need the device to notify of an attached USB device, so we need to specify an <intent-filter> and <meta-data> element pair for the android.hardware.usb.action.USB_DEVICE_ATTACHED intent in our main activity by adding the following element to the <application><activity> tag:

```
        <intent-filter>
<action
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHE
D" />
        </intent-filter>
<meta-data
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHE
D"
                android:resource="@xml/device_filter" />
```

However but in this case we should create an XML file called “device_filter” to specify the VendorID, ProductID for our serial RS232 to USB converter device:

```
<resources>
   <usb-device vendor-id="1234" product-id="5678" />
</resources>
```

Now to communicate with the USB Device we need to check all the connected USB Devices while our application is running, so we use the getDeviceList(); method to get a hash map of all the USB devices that are connected. The hash map is keyed by the USB device’s name :

```
        UsbManager manager = (UsbManager)
getSystemService(Context.USB_SERVICE);
        HashMap<String, UsbDevice> deviceList =
manager.getDeviceList();
```

To select a device from the hashmap we obtain an iterator from the hash map and check each device one by one:

```
Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
```

Before we can communicate with the USB device we must first have a permission from the user, to obtain permission first create a broadcast receiver. This receiver listens for the intent that gets broadcast when you call requestPermission(). The call to requestPermission() displays a dialog to the user asking for permission to connect to the device.

```
final String ACTION_USB_PERMISSION
="com.android.example.USB_PERMISSION";
PendingIntent mPermissionIntent ;
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);

public final BroadcastReceiver mUsbReceiver = new
BroadcastReceiver() {

public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action)) {
        synchronized (this) {
    UsbDevice device =
(UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);

if
(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTE
D, false)) {
if(device != null){
//call method to set up device communication
Toast.makeText(getApplicationContext(),"Permission
Granted",Toast.LENGTH_SHORT).show();
}
}
     else {
```

```
        Toast.makeText(getApplicationContext(),"permission denied for
        device " + device, Toast.LENGTH_SHORT).show();}}}}};
```

Then after we have our permission accepted we can go back to our devices iterator to try connecting to the devices connected via USB cable to our device:

```
try {
        while(deviceIterator.hasNext()){
                UsbDevice device1 = deviceIterator.next();
```

Then we will need to obtain the Endpoints and the Interfaces we've got for our USB device so we can use the method getInterface(0); which returns a UsbInterface Object then we call getEndpoint(1); on that interface to get the first endpoint after that we have to open a connection to that device an claim that interface and that endpoint with that connection before we can transfer some data through the USB cable:

```
                UsbInterface intf = device1.getInterface(0);
                        UsbEndpoint endpoint = intf.getEndpoint(1);
                        UsbDeviceConnection connection =
        manager.openDevice(device1);
                connection.claimInterface(intf, forceClaim);
        connection.controlTransfer(0x40, 0x03, 0x4138, 0, null, 0, 0);
```

now we are ready to transfer data through that connection by calling bulkTransfer();

```
connection.bulkTransfer(endpoint, bytes, bytes.length, TIMEOUT);
```

note: data must be byte[] type so we use "byte[] bytes=commandToSerial.getBytes();"

With this we are able to transfer any command to the AtMega16A via USB-OTG cable converted to RS232 port

Now we need to detect any SMS message received and check if it have a command, to do that we must add the SMS receiver permission to the manifest file:

<uses-permission android:name="android.permission.RECEIVE_SMS" />

to receive SMS we obtain a class called SMSReceiver that extends BroadcastReceiver to notify us when a SMS received which receive the SMS and returns the SMS body to the mainactivity using a broadcastIntent:

```
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
                        broadcastIntent.putExtra("sms", str);
                        arg0.sendBroadcast(broadcastIntent);
```

"arg0 is the context"

In the mainactivity we create a BroadcastReceiver to Receive the Intent send from SMSReceiver class and then get the body sent with it an manage to recognize the message and send the specify command to the microcontroller.

Then we need to detect the incoming calls to send a command to the microcontroller, for that the program first must have the permission to access the incoming calls so we add the following element to the <manifest> file:
<uses-permission
android:name="android.permission.READ_PHONE_STATE"/>

Then we decelerated a PhoneStatelistener to notify the program when a change happened to the current PhoneState:

```
telephonyManager.listen(new PhoneStateListener() {
                        public void onCallStateChanged (){…}
```

Chapter 5

1) Application overview

The following application uses the Android Bluetooth APIs to construct a

Simple peer-to-peer messaging between two paired Bluetooth devices to control the door lock and the garage lock.

And connects to the Server by phone call or by SMS to control the door lock And the garage lock.

2) Required for this application

- Android Phone
- Android Application

About the Android phone:

An Android phone is a smartphone running on Google's open-source Android operating-system, And every android phone support Bluetooth and that is important in this project becauseBluetooth is also one of the ways to transfer information between elements of this projectAndroid's share of the global smartphone market, led by Samsung products, was 64% in March 2013.In July 2013 there were 11,868 different models of Android device, scores of screen sizes and eight OS versions simultaneously in use. The operating system's success has made it a target for patent litigation as part of the so-called "smartphone wars" between technology companies. As of May 2013, a total of 900 million Android devices have been activated and 48 billion apps have been installed from the Google Play store.

Figure 85

About the Android Application:

There's no other software quite like Android. Google engineered Android, and Google’s own apps run best on it. And with millions of apps, games, songs, and videos on Google Play, Android is great for fun, and for getting things done.

Mobile software application developed for use on devices powered by Google's Android platform. Android apps are available in the Google Play Store (formerly known as the Android Market), and the apps can run on Android smartphones, tablets, Google TV and other devices

A feature that enables Google Android-based smartphones, tablets and similar mobile devices to share content with other near-field communication-capable devices by simply touching the devices together and pressing a button on the device sending the content. The Android Beam feature first appeared in the v4.1 “Ice Cream Sandwich” release of Google’s Android mobile operating system, which became available in late 2011.

- The interface of this application It should contain a
    1. Button to Show LDR + GAS sensor
    2. Button to close garage door quickly
    3. Button to close garage door slowly
    4. Button to open garage door quickly
    5. Button to open garage door slowly
    6. Button to stop garage door
    7. Button to Open house Door
    8. Button to show Temperature Value
    9. Button to make a call with the server

3), Use cases of this Application

Use Case Name:	Start	Author: Anas Hamdan
Use Case ID:	BTID1	
Priority:	High	
Primary System Actor:	User	
Prerequisite:	-------	
Description:	This use case initialize the Application Interface	
Trigger:	This use case initiated when the user Click on the Application Icon	
Flow of events:	Actor Action	System Response
	1: the user click on the Application icon	2: initialize the Application Interface

Use Case Name:	LDR+GAS	Author: Anas Hamdan
Use Case ID:	BTID2	
Priority:	High	
Primary System Actor:	User	
Prerequisite:	BTID1	
Description:	This use case show the LDR + Gas value	
Trigger:	This use case initiated when the user Click on LDR + GAS Button	
Flow of events:	Actor Action	System Response
	1: the user click on the LDR + GAS Button	2: Send number 1 as a message to the server 3.show the LDR + GAS value

Use Case Name:	close garage quickly	Author: Anas Hamdan
Use Case ID:	BTID3	
Priority:	High	
Primary System Actor:	User	
Prerequisite:	BTID1	
Description:	This use case close garage door quickly	
Trigger:	This use case initiated when the user Click on close garage door quickly Button	
Flow of events:	Actor Action	System Response
	1: the user click on the close garage door quickly Button	
		2: Send number 2 as a message to the server

Use Case Name:	close Garage slowly	Author: Anas Hamdan
Use Case ID:	UCID4	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case close garage slowly	
Trigger:	This use case initiated when a user click on close garage slowly	
Flow of events:	Actor Action	System Response
	1: user click on open close garage slowly Bluetooth Button	
		2. Send number 3 as a message to the server

Use Case Name:	open Garage slowly	Author: Anas Hamdan
Use Case ID:	UCID5	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case open Garage door slowly	
Trigger:	This use case initiated when a user click on open Garage door slowly Button	
Flow of events:	Actor Action	System Response
	1: user click on open Garage door slowly Button	
		2. Send number 4 as a message to the server

Use Case Name:	open Garage quickly	Author: Anas Hamdan
Use Case ID:	UCID6	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case open Garage door quickly	
Trigger:	This use case initiated when a user click on open Garage door quickly Button	
Flow of events:	Actor Action	System Response
	1: user click on open Garage door quickly Button	
		2. Send number 5 as a message to the server

Use Case Name:	Stop Garage Door	Author: Anas Hamdan
Use Case ID:	UCID7	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case Stop Garage Door	
Trigger:	This use case initiated when a user click on Stop Garage Door Button	
Flow of events:	Actor Action	System Response
	1: user click on Stop Garage Door Button	
		2: Send number 6 as a message to the server

Use Case Name:	open Door	Author: Anas Hamdan
Use Case ID:	UCID8	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case send a message to open house door	
Trigger:	This use case initiated when a user click on open house door	
Flow of events:	Actor Action	System Response
	1: user click on open house door Button	
		2: Send number 7 as a message to the server

Use Case Name:	Close Door	Author: Anas Hamdan
Use Case ID:	UCID9	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case send a message to Close house door	
Trigger:	This use case initiated when a user click on Close house door	
Flow of events:	Actor Action	System Response
	1: user click on Close house door Button	
		2: Send number 8 as a message to the server

Use Case Name:		Author: Anas Hamdan
Use Case ID:	BTID10	
Priority:	High	
Primary System Actor:	User	
Prerequisite:	BTID1	
Description:	This use case show the temperature value	
Trigger:	This use case initiated when the user Click on temperature Button	
Flow of events:	Actor Action	System Response
	1: the user click on the temperature Button	
		2. Send number 9 as a message to the server 2: show The temperature Value

Use Case Name:	Open Door By call	Author: Anas Hamdan
Use Case ID:	UCID11	
Priority:	High	
Primary System Actor:	user	
Prerequisite:	UCID1	
Description:	This use case call the server to open the door	
Trigger:	This use case initiated when a user click on Open Door By call Button	
Flow of events:	Actor Action	System Response
	1: user click on Open Door By call	
		2:the system call the server

4), Sequence diagram of each Usecase

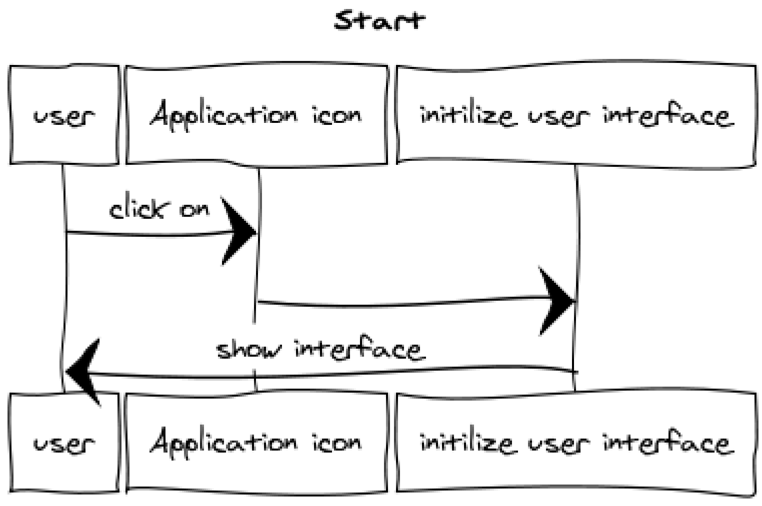


Figure 86

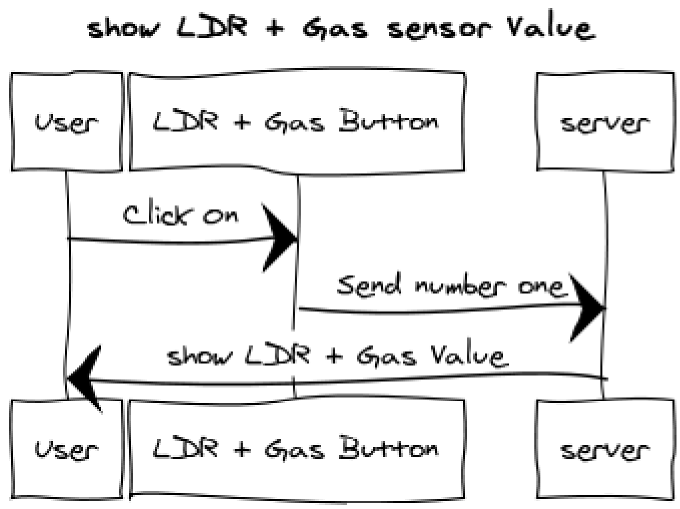


Figure 87

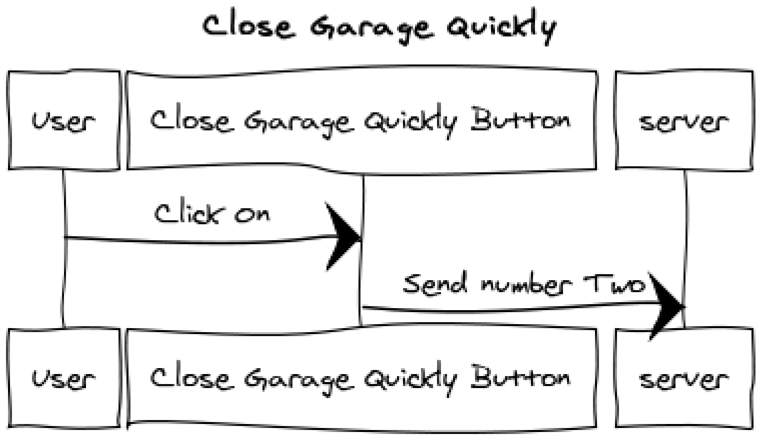


Figure 88

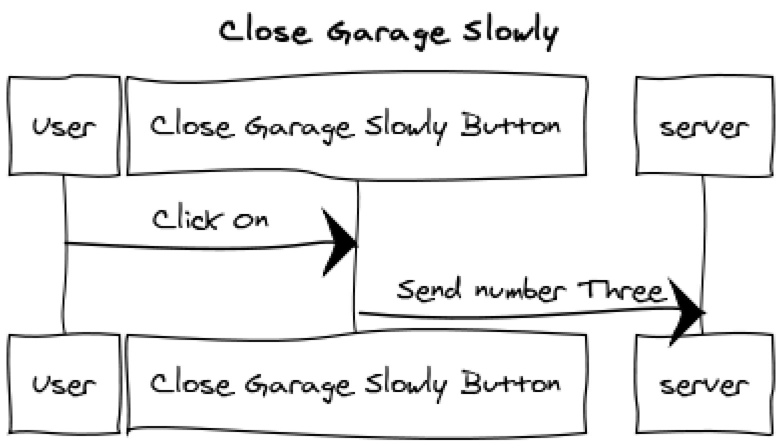


Figure 89

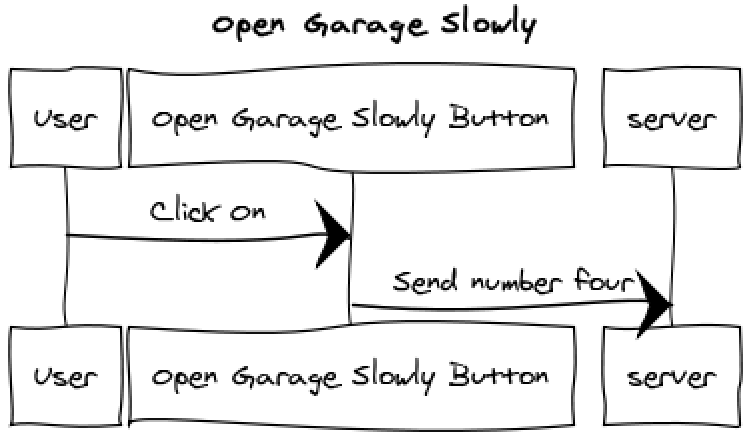


Figure 90

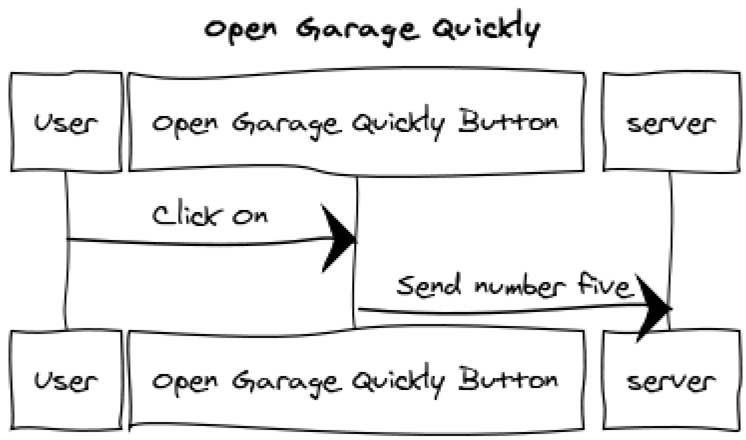


Figure 91

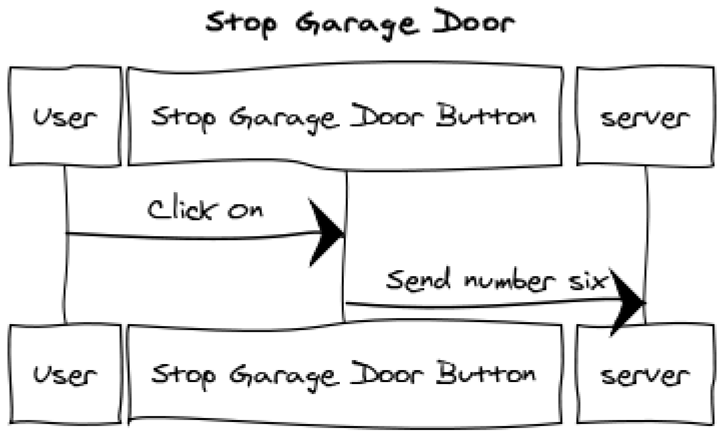


Figure 92

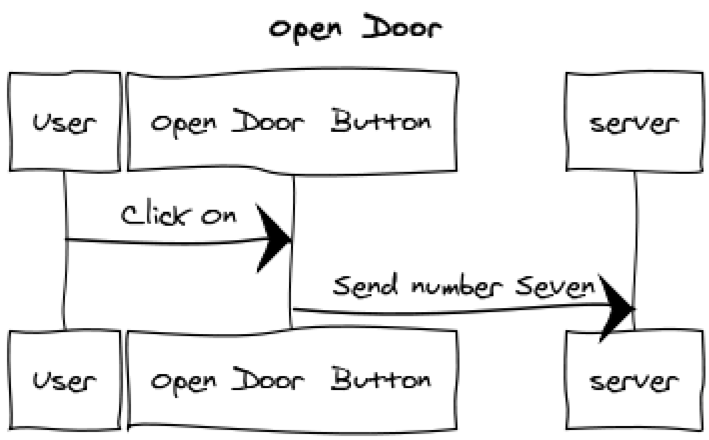


Figure 93

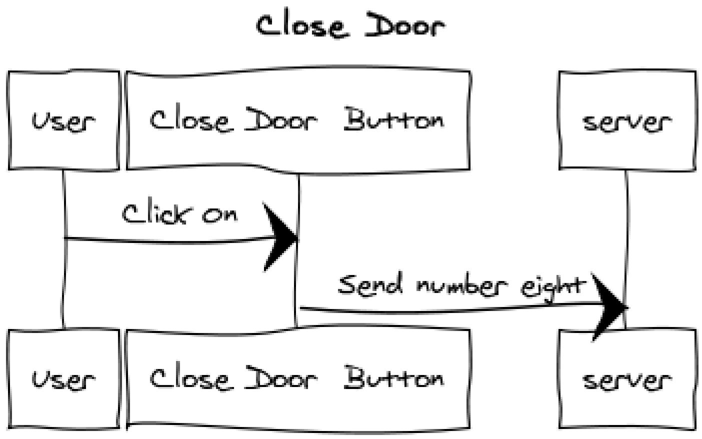


Figure 93

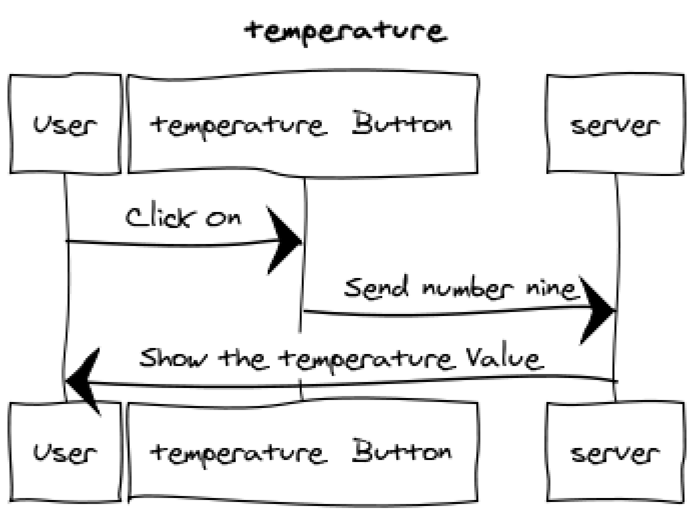


Figure 94

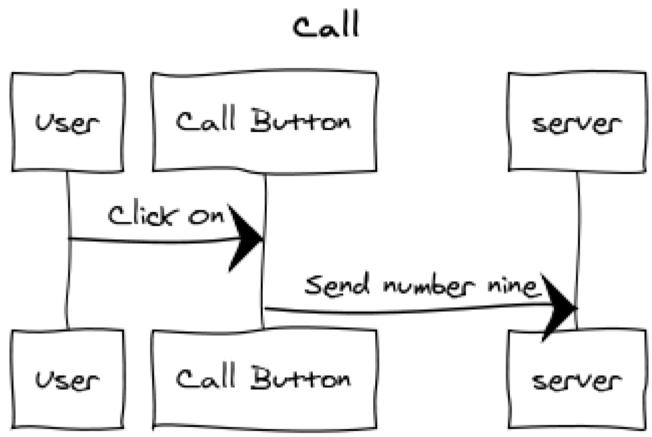

Call
User
Call Button
server
Click on
Send number nine
User
Call Button
server

Figure 95

5), the mechanism of action within the system

There are three ways to connect to the server through either Bluetooth or messages or phone call

1. Connect via Bluetooth
   a. But before we can do any of that we need to take care of a few pesky little details. First, we need to pair the two android devices. You can do this by opening the application drawer on both devices then going to settings. From there open Wireless and network. Then Bluetooth settings. Next we need to tell android that we will be working with Bluetooth by adding those two elements to the <manifest> tag inside AndroidManifest.XML file: <uses-permission android:name="android.permission.BLUETOOTH"/><uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

   b. Fill in the configureBluetooth stub to get access to the local Bluetooth Adapter and store it in a field variable.
      Take this opportunity to create a field variable for a Bluetooth Socket.
      This will be used to store either the server or client communications socket once a channel has been established.
      Should also define a UUID to identify your application when connections are being established.
      Private UUID uuid = UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666");
      bluetooth = BluetoothAdapter.getDefaultAdapter();

c. Create the client-side connection code. By performing discovery and displaying each of the possible devices, will provide a means for the client device to search for the listening server.

```
remoteDevice=intent.getParcelableExtra(BluetoothDevice.EXTRA
_DEVIC);
if (bluetooth.getBondedDevices().contains(remoteDevice)) {
foundDevices.add(remoteDevice);
registerReceiver(discoveryResult,
newIntentFilter(BluetoothDevice.ACTION_FOUND));
if (!bluetooth.isDiscovering()) {
bluetooth.startDiscovery();
```

d. Create an asynchronous listener that monitors the Bluetooth Socket for incoming messages.
   Start by creating a new MessagePoster class that implements Runnable. It should
   accept two parameters, a Text View and a message string. The received message
   should be inserted into the Text View parameter

```
private class MessagePoster implements Runnable
 {
            privateTextViewtextView;
            private String message;
            publicMessagePoster(TextViewtextView, String
   message)
            {this.textView = textView;
            this.message = message;}
   public void run()
            {
   textView.setText(message);
            }
 }
```

2. Connect via SMS:

```
try {
SmsManager.getDefault().sendTextMessage(phoneNumber
, null, "Hello
SMS!", null, null);
 } catch (Exception e) {
 AlertDialog.Builder alertDialogBuilder = new
 AlertDialog.Builder(this);
  AlertDialog dialog = alertDialogBuilder.create();

  dialog.setMessage(e.getMessage());

  dialog.show();
```

3. Connect via Call:

```
IntentcallIntent=newIntent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:0999745772"));
startActivity(callIntent);
```

Future Works:

- **Add a GPS capability.**
- **Add NFC capability to auto unlock the door with a single touch.**
- **Use the web to control the system from anywhere.**
- **Build a control application for IOS.**

References:

inc., G. (n.d.). *Android Telephony APIs*. Retrieved from http://developer.android.com/: http://developer.android.com/reference/android/telephony/package-summary.html

inc., G. (n.d.). *Bluetooth Connectivity*. Retrieved from http://developer.android.com/: http://developer.android.com/guide/topics/connectivity/bluetooth.html

inc., G. (n.d.). *USB Connectivity*. Retrieved from http://developer.android.com/: http://developer.android.com/guide/topics/connectivity/usb/accessory.html