Mobile Health Monitoring Device

A

Project Work

Submitted in partial fulfillment of the Requirements for the Award degree

Of

Bachelor of Technology

In

Electronics and Communication Engineering

Submitted by

Name of Student	**Scholar Number**
S Hemanth	121114127
G Kranthi Kiran	121114099
S Mahi Kiran Reddy	121114095
Shiv Ram Abhishek Ch	121114093
Yaswanth Santha Hruday J	121114086

Under the Guidance of

Dr. Sangeeta Nakhate

April – 2016

ELECTRONICS & COMM. ENGINEERING DEPARTMENT
MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL

CANDIDATE'S DECLARATION

We hereby declare that the work, which is being presented in the U.G. project report entitled "**MOBILE HEALTH MONITORING DEVICE"**, which is being submitted to the **Electronics and Communication Engineering Department, MANIT Bhopal**, in the partial fulfilment of the requirements for the award of the Degree of **Bachelor of Technology** in the **Department of Electronics & Communication Engineering,** is a bonafide report of the work carried out by us**.** We have not submitted the matter embodied in the dissertation to any other university or institution for the award of any other degree or diploma.

Name	**Scholar No.**	**Signature**
S MAHI KIRAN REDDY	**121114095**	
S HEMANTH	**121114127**	
GOBIDESI KRANTHI KIRAN	**121114099**	
YASWANTH S HRUDAY J	**121114086**	
SHIV RAM ABHISHEK CH	**121114093**	

CERTIFICATE OF APPROVAL

It is certified that the work contained in this report titled **"MOBILE HEALTH MONITORING DEVICE"** is the original work done by above students and has been carried out under my supervision.

Supervisor's Name: **Dr. Sangeeta Nakhate**

Date :

CONTENTS

LIST OF FIGURES

ABSTRACT

Heart rate, or pulse, is one of the vital signs used to monitor basic functions of human body. Heart rate is the number of times one’s heart beats per minute. The heart rate can be measured by monitoring one's pulse using specialized medical devices such as an electrocardiograph (ECG), portable device e.g. wrist strap watch, or any other commercial heart rate monitors which normally consists of a chest strap with electrodes. Despite of its accuracy, somehow it is costly, involve many clinical settings and patient must be attended by medical experts for continuous monitoring. For a patient who was already diagnosed with fatal heart disease, their heart rate condition has to be monitored continuously. This project proposes an alert system that is able to monitor the heartbeat of a patient. From the various methods, after a great scrutiny, we came up with the method widely known as Photoplethysmography (PPG).

The constructed device can be used to find out the heart rate of a person and analyse readings using raspberry pi and LCD display. Any body part can be used to measure heart rate through the sensor of the device, and in this particular device we do it through fingertip.

Along with the heart beat sensor, a temperature sensor is also interfaced through the raspberry pi to display the surrounding temperature on the display.

This device is made using a microprocessor and interfacing it with sensors that fetch data at regular intervals. We build algorithms that collect raw data from all the sensors and then analyse the data. With the help of display, the user can check his status whenever he wants. Also in case of emergency, the device will send an alert to a predefined person.

INTRODUCTION

Most heart cases need continuous monitoring of the heart condition. Apart from indicating the soundness of the heart, heart rate helps in assessing the cardiovascular system. Analysis of heart rate helps in diagnosis and detection of coronary diseases. The normal range of heart rate among adults is 60-100 beats per minute (bpm). If heart rate is higher than normal the condition is known as tachycardia, in the opposite case it is known as bradycardia.

There are many devices available in the market to measure heart rate. The constructed device brings about some major changes compared to the available ones. Firstly, this is a cost effective option to measure heart rate. Secondly, the device provides an easy way to interface with personal computers and mobile phones. Thirdly, although there are a few analog devices to measure heart rate, the patient needs help from another person to do so. This device can be used without any need of assistance from others. Finally, the device is easy to use. The user only needs to place a fingertip on a sensor.

The signal from the sensor is sent to raspberry pi which counts the heart rate and sends it to the display. Coding of raspberry pi is done using python language. Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or java. Moreover python is the main programming language of raspberry pi.

Along with the heartbeat sensor, a temperature sensor is also interfaced to display the surrounding temperature. The temperature sensor used is LM35, a precision integrated-circuit temperature sensor, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ˚ Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1⁄4˚C at room temperature and ±3⁄4˚ Cover a full −55 to +150˚C temperature range. This temperature sensor could be used in such a way that it measures the temperature of the patient. The display used is JHD162A which is a 16*2 8 bit backlit LCD display. The heart beat and temperature sensed by the sensors are displayed on this display through the python code run through raspberry pi. Thus the heartbeat of the patient could be continuously monitored and in case of any abnormality, a message or an alert is sent to a particular predefined person. In addition to this the data of the vitals could be stored in cloud or precisely in mail.

RASPBERRY PI

Introduction

Our project comprises the interfacing of hardware components as well as software concepts such as coding and circuit simulations. And moreover, we require a device which can simultaneously handle Heart Beat measurement, Real Time Clock operation, Tepmerature measurement, Alert Sending etc. The device must also be able to back its data up in the cloud as well as send regular alerts in case of any abnormal fluctuations. To coordinate all these ideas, we need memory as well as USB ports. Hence, we chose Raspberry Pi instead of any ordinary microcontroller.

Raspberry Pi is a simple yet powerful single board computer. It has been developed in England by the Raspberry Pi foundation. Several generations of Raspberry Pi's have been released, wherein each consecutive generation is an advanced version of the previous generations in terms of storage memory, CPU speed, number of USB ports, compactness in size and weight. All models feature a Broadcom system on a chip (SOC) which include an ARM compatible CPU and an on chip graphics processing unit GPU (a VideoCore IV). CPU speed range from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256MB to 1GB RAMS. Secure Digital SD cards are used to store the operating system and program memory in either the SDHC or MicroSDHC sizes. Most boards have between 1 and 4 USB slots, HDMI and composite video output, and a 3.5mm phono jack for audio. Lower level output is provided by a number of GPIO pins which support common protocols like I2C. Some models have an RJ45 Ethernet port and the Pi 3 has on board WiFi 802.11n and Bluetooth.

For this project, we have used a Raspberry Pi 2 B with the following specifications.

- A 900 MHz quad-core ARM Cortex-A7 CPU
- Broadcom BCM2836 (SoC)
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)

- Micro SD card slot
- VideoCore IV 3D graphics core

Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, as well as Microsoft Windows 10. It utilizes a Broadcom BCM2836 System On Chip (SoC).

RASPBERRY PI LAYOUT

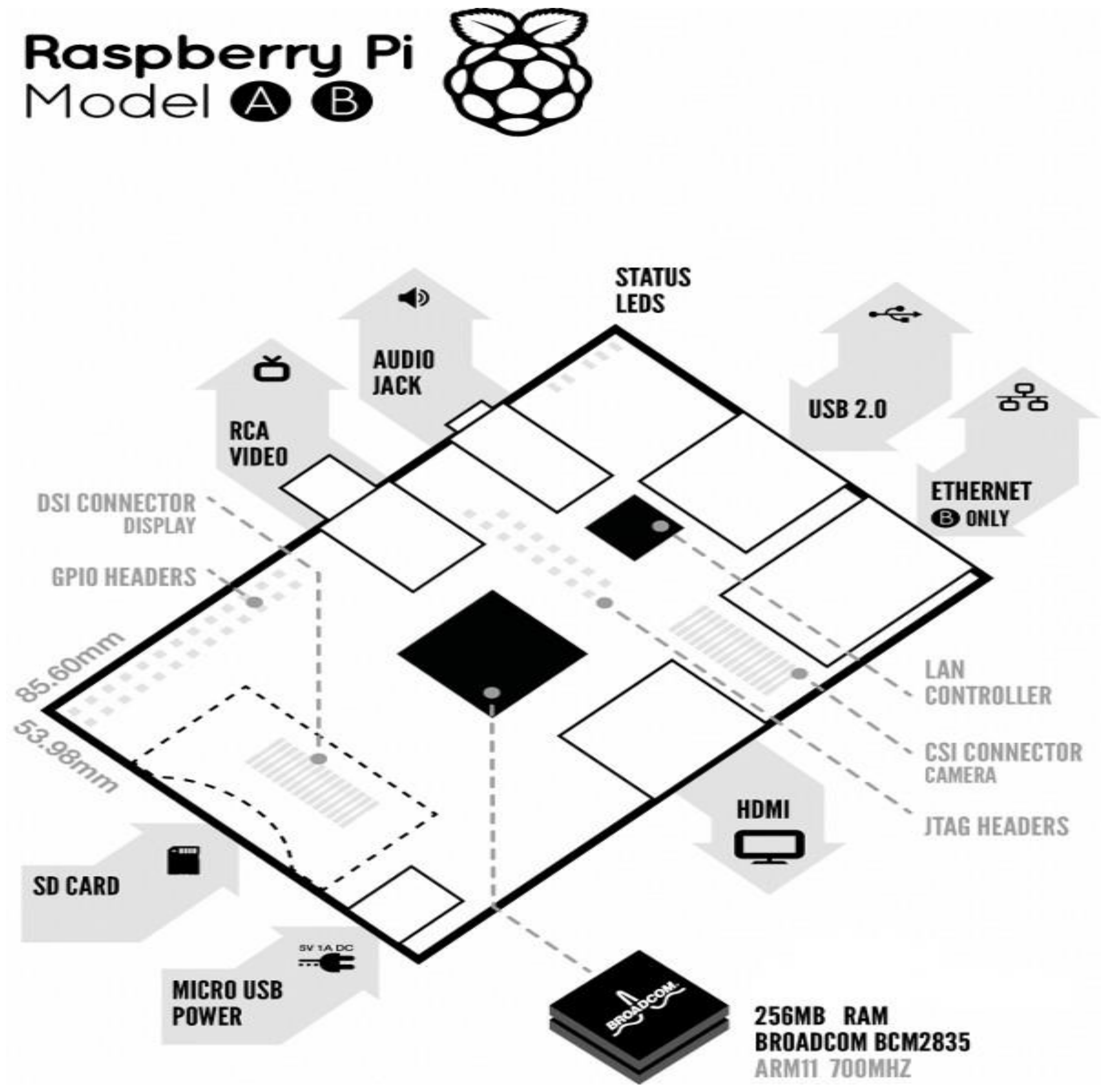


Figure 1: Raspberry Pi Layout

Heartbeat Sensor

INTRODUCTION

The major constituent of the project is a heartbeat sensor circuit. The circuit identifies the variations in blood volume in tissues using the process of Photoplethysmography, measures the variations and records them. The circuit diagram is given below.

Working Principle

This project is based on the principle of Photoplethysmography (PPG) which is a non-invasive method of measuring the variation in blood volume in tissues using a light source and a detector. Since the change in blood volume is synchronous to the heart beat, this technique can be used to calculate the heart rate. Transmittance and Reflectance are two basic types of photoplethysmography. For this project we used the reflectance PPG. For the transmittance PPG, a light source is emitted in to the tissue and a light detector is placed in the opposite side of the tissue to measure the resultant light. Because of the limited penetration depth of the light through organ tissue, the transmittance PPG is applicable to a restricted body part, such as the finger or the ear lobe. However, in the reflectance PPG, the light source and the light detector are both placed on the same side of a body part. The light is emitted into the tissue and the reflected light is measured by the detector. As the light doesn't have to penetrate the body, the reflectance PPG can be applied to any parts of human body. In either case, the detected light reflected from or transmitted through the body part will fluctuate according to the pulsating blood flow caused by the beating of the heart.

The PPG signal has two components, frequently referred to as AC and DC. The AC component is mainly caused by pulsating changes in arterial blood volume, which is synchronous with the heart beat. So, the AC component can be used as a source of heart rate information. This AC component is superimposed onto a large DC component that relates to the tissues and to the average blood volume. The DC component must be removed to measure the AC waveform with a high signal-to-noise ratio. Since the useful AC signal is only a very small portion of the whole signal, an effective amplification circuit is also required to extract desired information from it.

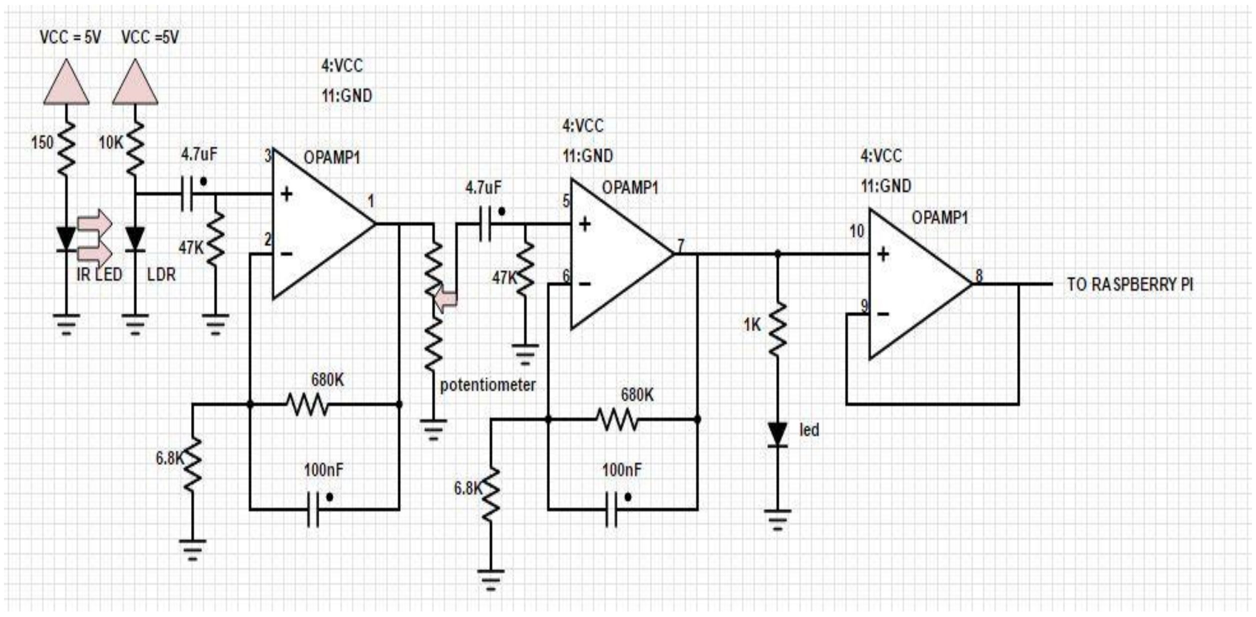


Figure 2: Heartbeat Sensor Circuit Diagram

<u>Working of circuit:</u>

The sensor used in this project is a reflective optical sensor with both the infrared light emitter and phototransistor placed side by side and are enclosed inside a leaded package so that there is minimum effect of surrounding visible light. Connecting the Enable pin to 5V will turn the IR emitter LED on and activate the sensor. A fingertip placed over the sensor will act as a reflector of the incident light. The amount of light reflected back from the fingertip is monitored by the phototransistor.

The output (V_{SENSOR}) from the sensor is a periodic physiological waveform attributed to small variations in the reflected IR light which is caused by the pulsating tissue blood volume inside the finger. The waveform is, therefore, synchronous with the heartbeat. This weak signal contains a lot of noise superimposed on the useful AC signal which is then thoroughly filtered a series of signal processing circuits via a Quad OP-AMP LM324N.

The signal is first passed through a RC high-pass filter (HPF) to get rid of the DC component. The cut-off frequency of the HPF is set to 0.7 Hz. The next stage is an active low-pass filter (LPF) that is made of an Op-Amp circuit. The gain and the cut-off frequency of the LPF are set to 101 and 2.34 Hz, respectively. Thus the combination of the HPF and LPF helps to remove unwanted DC signal and high frequency noise including 50 Hz mains interference, while amplifying the low amplitude pulse signal (AC component) 101 times. The output from the first signal conditioning stage goes to a similar HPF/LPF combination for further filtering and amplification. So, the total voltage gain achieved from the two cascaded stages is 101*101 = 10201. The two stages of filtering and amplification converts the input PPG signals to near Transistor –transistor Logic (TTL) pulses and they are synchronous with the heartbeat.

A 10K potentiometer is placed at the output of the first signal conditioning stage in case the total gain of the two stages is required to be less than 10201. An LED connected to the output of the second stage of signal conditioning will blink when a heartbeat is detected. The final stage of the instrumentation constitutes a simple non-inverting buffer to lower the output impedance. This is helpful if an ADC channel of a microcontroller is used to read the amplified PPG signal. The output is then connected to Digital Pin 22 of Raspberry pi. The microprocessor measures the frequency of the PPG signal and returns the value of heart rate in BPM. The frequency (f) of these pulses is related to the heart rate (BPM) as,

$$\text{Beats per minute (BPM)} = 60*f$$

Temperature Sensor

Introduction

Instead choosing digital temperature senor we chose analog temperature sensor with ADC because both are same cost but later one offers more advantages as we can connect more analog sensors. We know analog sensors are cheaper than digital one's, so cost can be reduced.

Analog temperature sensor used in this device is LM35 because these are precision integrated-circuit temperature devices with an output voltage linearly-proportional to the Centigrade temperature.

Sensor working:

There are two transistors in the centre of the drawing. One has ten times the emitter area of the other. This means it has one tenth of the current density, since the same current is going through both transistors. This causes a voltage across the resistor R1 that is proportional to the absolute temperature, and is almost linear across the range we care about. The "almost" part is taken care of by a special circuit that straightens out the slightly curved graph of voltage versus temperature.

The amplifier at the top ensures that the voltage at the base of the left transistor (Q1) is proportional to absolute temperature (PTAT) by comparing the output of the two transistors. The amplifier at the right converts absolute temperature (measured in Kelvin) into Celsius. The little circle with the "i" in it is a constant current source circuit. The two resistors are calibrated in the factory to produce accurate temperature sensor. The integrated circuit has many transistors in it -- two in the middle, some in each amplifier, some in the constant current source, and some in the curvature compensation circuit. All of that is fit into the tiny package with three leads.

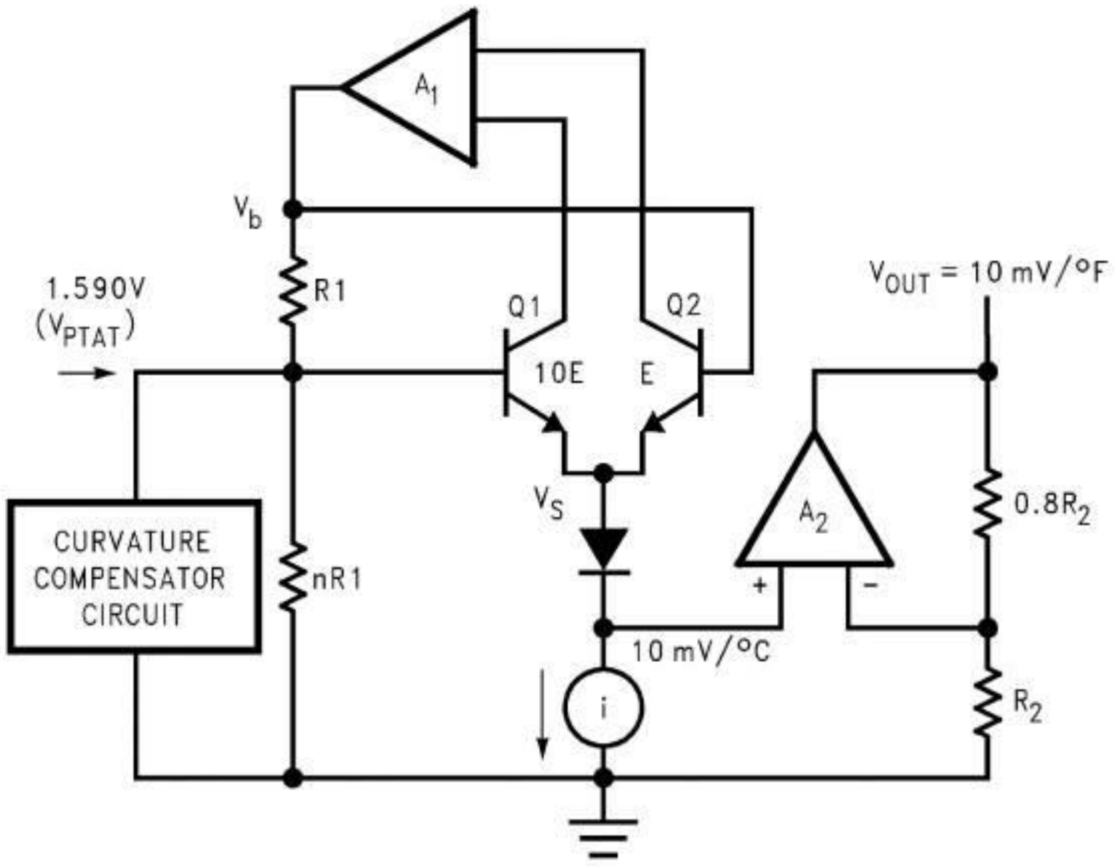


Figure 3: Circuit Diagram of Temperature Sensor

ADC using Atmega 8L

Introduction

Atmega 8L is low power AVR RISC based microcontroller. It provides the following features. 8 Kbytes of In System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1 Kbyte of SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two wire Serial Interface, a 6-channel ADC, a programmable Watchdog Timer with Internal Oscillator and an SPI serial port.

It is crucial to use digital values which can be easily recognized by the computing systems. The computing system understands this by converting them into binary numbers. So for transferring external continuous information (analog information) into a digital/computing system, we must convert them into digital values. This type of conversion is carried out by Analog to Digital Converter (ADC). The process of converting an analog value into digital value is known as Analog to Digital Conversion.

In this project the sole purpose of using Atmega 8L is using its inbuilt ADC for converting the analog temperature sensor output to a digital signal which can be further used in Raspberry pi. ADC of the Atmega 8L is multiplexed with Port C.

Principle:

The principle for ADC in AVR microcontrollers uses a technique known as successive approximation by comparing input voltage with half of the reference voltage generated internally. The comparison continues by dividing the voltage further down and updating each bit in ADC register by 1 if input voltage is higher than the reference voltage, 0 otherwise. This process lasts 10 times (for 10 bit ADC) and generates resulting binary output. This process is known as successive approximation.

Modes of Operation:

The ADC can be operated in single conversion mode and free running mode. In single conversion mode, the ADC does a single conversion and stops. But in free running mode the ADC is continuously converting, i.e. it does a conversion and then start the next conversion instantly after that. In the project we are using the ADC in the free running mode for continuously converting the analog temperature sensor value of the LM35 temperature.

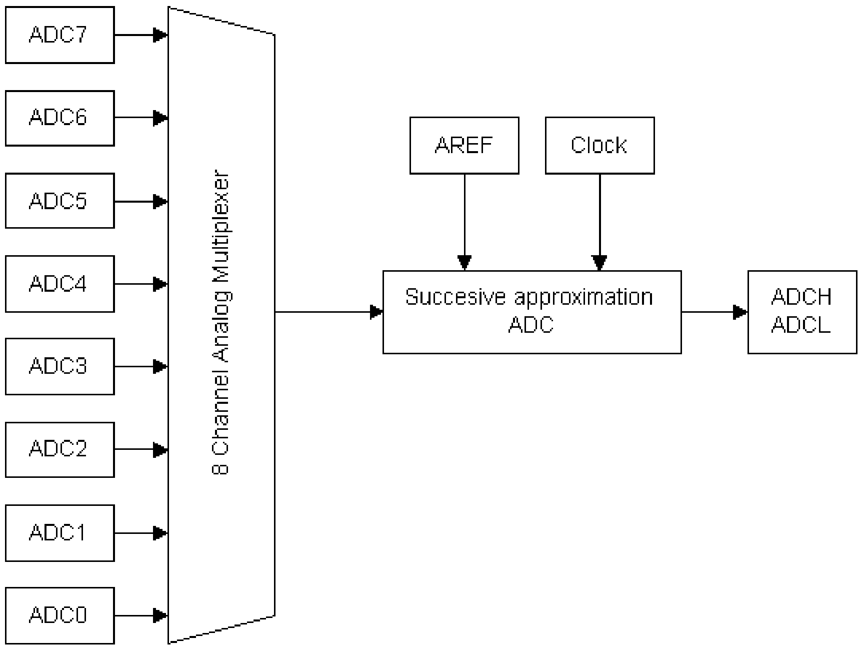


Figure 4: Circuit of ADC

ADC Prescaler

The ADC of the AVR converts analog signal into digital signal at some regular interval. This interval is determined by the clock frequency. In general, the ADC operates within a frequency range of 50 kHz to 200 kHz. But the CPU clock frequency is much higher (in the order of MHz). So to achieve it, frequency division must take place. The prescaler acts as this division factor. It produces desired frequency from the external higher frequency. You configure the division factor of the prescaler using the ADPS bits.

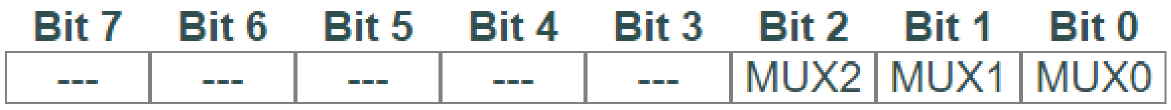
Registers

The ADC has only four registers.

1. ADC Multiplexer Selection Register – ADMUX
2. ADC Control and Status Register A – ADCSRA
3. The ADC Data Register – ADCL and ADCH

ADC Multiplexer Selection Register – ADMUX

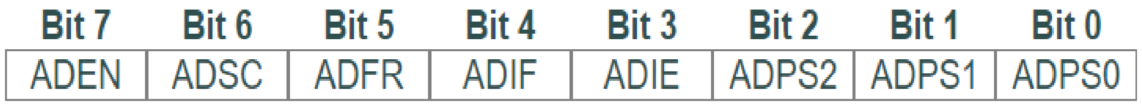
This register is used to select which of the 8 channel (between ADC0 to ADC7) will be the input



Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
---	---	---	---	---	MUX2	MUX1	MUX0

to the ADC. Since there are 8 possible inputs, only the 3 least significant bits of this register are used.

ADC Control and Status Register A – ADCSRA



Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN (ADC Enable) bit: Setting this bit enables the ADC. By clearing this bit to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress will terminate this conversion.

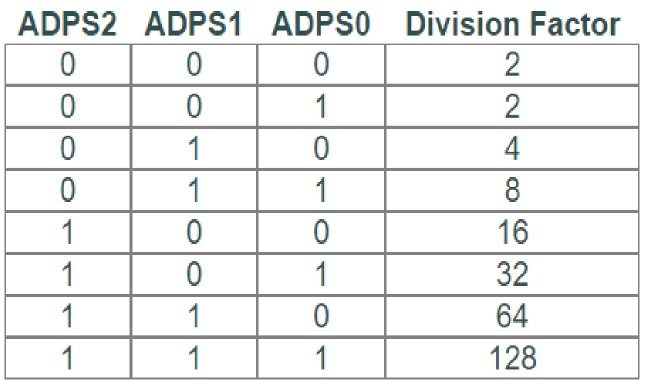
ADSC (ADC Start Conversion) bit: In Free Running Mode, you must set this bit to start the first conversion. The following conversions will be started automatically. In Single Conversion Mode, you must set it to start each conversion. This bit will be cleared by hardware when a normal conversion is completed. Remember that the first conversion after the ADC is enabled is an extended conversion. An extended conversion will not clear this bit after completion.

ADFR (ADC Free Running Select) bit: If you want to use the Free Running Mode, you must set this bit.

ADIF (ADC Interrupt Flag) bit: This bit is set when an ADC conversion is completed. If the ADIE bit is set and global interrupts are enabled, the ADC Conversion Complete interrupt is executed. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical 1 (!) to the flag. This has a nasty side effect: if you modify some other bit of ADCSR using the SBI or the CBI instruction, ADIF will be cleared if it has become set before the operation.

ADIE (ADC Interrupt Enable) bit: When the ADIE bit is set and global interrupts are enabled, the ADC interrupt is activated and the ADC interrupt routine is called when a conversion is completed. When cleared, the interrupt is disabled.

ADPS (ADC Prescaler Select) bits: These bits determine the division factor between the AVR clock frequency and the ADC clock frequency. The following table describe the setting of these bits:



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

The ADC Data Register – ADCL and ADCH

These registers hold the result of the last ADC conversion. ADCH holds the two most significant bits, and ADCL holds the remaining bits. When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, it is essential that both registers are read and that ADCL is read before ADCH.

UART (Universal Asynchronous Receiver/Transmitter)

Introduction

UART (Universal Asynchronous Receiver/Transmitter), from the name itself, it is clear that it is asynchronous i.e. the data bits are not synchronized with the clock pulses. A UART provides the computer with the interface necessary for communication with modems and other serial devices. However, unlike a UART, a USART offers the option of both synchronous mode and asynchronous mode.

In our project the output of the ADC from the Atmega 8L is sent to the Raspberry pi through UART protocol. The received serial data through UART at Raspberry pi is used to display the temperature of the sensor.

UART Pin Configuration

Now let's have a look at the hardware pins related to UART. The UART of the AVR occupies three hardware pins pins:

1. RxD: USART Receiver Pin (ATMega8L Pin 2)
2. TxD: USART Transmit Pin (ATMega8L Pin 3)

Working of UART

UART relies on a baud rate agreement between receiving and transmitting controllers. The receiving Raspberry pi will expect to receive bits into its RX port at specific baud rate, 9600 in our case, and the transmitting Atmega 8L will transmit bits at the same baud rate, 9600. There is no master clock between the two parties, only the agreement in baud rate. Therefore the receiving Raspberry pi will know the rate that bits will flow into its RX port.

Baud Rate Generation

The baud rate of UART is set using the 16-bit wide USART Baud Rate Register (UBRR)**.** Since AVR is an 8-bit microcontroller, every register should have a size of 8 bits. Hence, in this case, the 16-bit UBRR register is comprised of two 8-bit registers – UBRRH (high) and UBRRL (low).

Frame Formats

A frame refers to the entire data packet which is being sent/received during a communication. A typical frame for USART/RS232 is usually 10 bits long: 1 start bit, 8 data bits, and a stop bit.

Order of Bits

1. Start bit (Always low)
2. Data bits (LSB to MSB) (5-9 bits)
3. Parity bit (optional) (Can be odd or even)
4. Stop bit (1 or 2) (Always high)

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, a new frame can directly follow it, or the communication line can be set to an idle (high) state.

LCD DISPLAY: JHD 162A

JHD 162A is the 8 bit LCD display. It consists of 16 pins.

FEATURES

•Display construction……………16 Characters * 2 Lines

•Display type……………………… Positive Transflective

•Backlight………………………… LED (B/5.0V)

•Operating temperature…………… Indoor

•Driving voltage…………………… Single power

•Driving method……………………1/16 duty,1/5 bias

•Type………………………………… COB (Chip on Board)

•Number of data line…………………8-bit parallel

•Connector…………………………… Pin

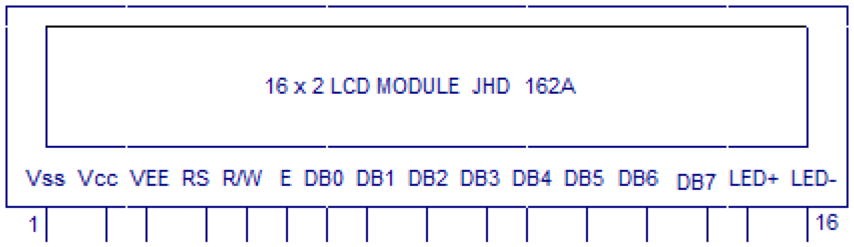


Figure 5: Pin Diagram of LCD

PCB

In this project, we have used a single layer PCB for the fabrication of circuits of Heartbeat sensor and Temperature sensor.

A Printed Circuit Board (PCB) mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from copper sheets laminated onto a non-conductive substrate. PCBs can be *single sided* (one copper layer), *double sided* (two copper layers) or *multi-layer* (outer and inner layers). Multi-layer PCBs allow for much higher component density. Advanced PCBs may contain components - capacitors, resistors or active devices - embedded in the substrate.

In our project we are using a simple single sided PCB's to design both the heart beat sensor and also the pressure sensor. The first step of the PCB design was to prepare the board layout from the circuit schematic. For this purpose we used PCB wizard 4.0. This software provides a comprehensive range of tools including schematic drawing, schematic capture, component placement, automatic routing and file generation for manufacturing. Drawing PCB's is an artwork. Some simple guideline should be followed while preparing the board layout.

- Try to design a layout which fits the whole circuit in a limited area.
- Place your components in a similar way as in your schematics. This makes it easier to troubleshoot your circuit if needed later on.
- Your components should all have labels with name and part value printed on the board. This makes it easier to solder the circuit board.
- Finally, before you send your board layout to PCB manufacturing, the layout should be checked thoroughly to rectify any errors made.

PCB DESIGN

The PCB designs for the respective project are attached here below for the better understanding and outlook of the pin connection of the respective components:

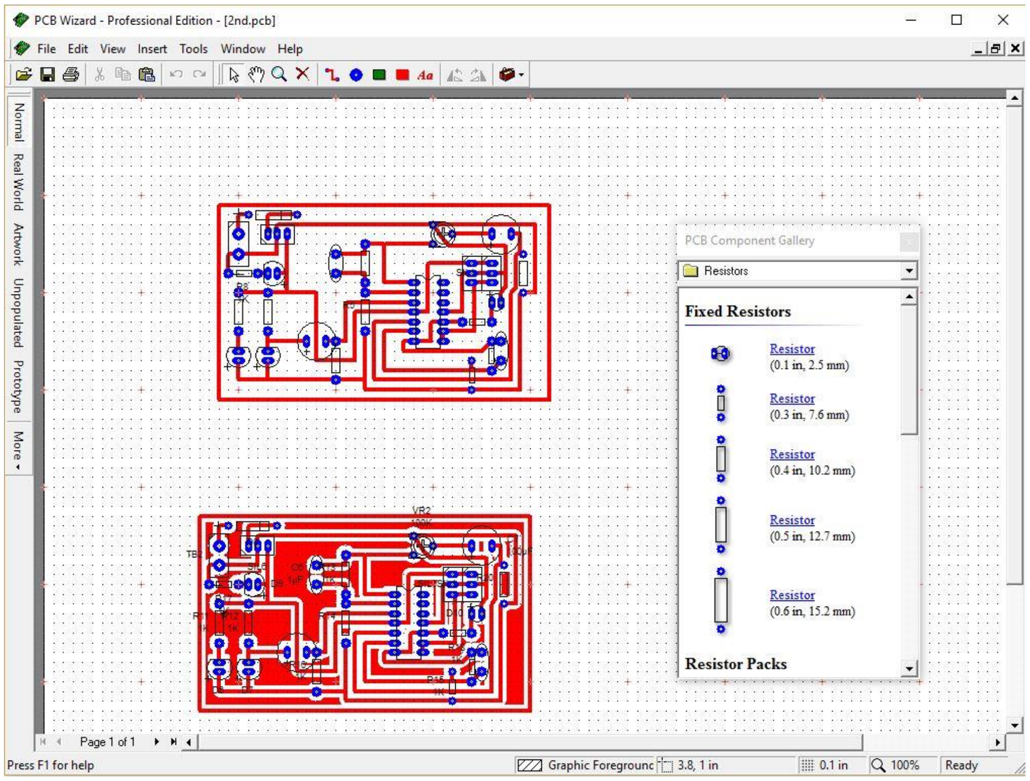

PCB Wizard - Professional Edition - [2nd.pcb]
File Edit View Insert Tools Window Help
Normal Real World Artwork Unpopulated Prototype More
PCB Component Gallery
Resistors
Fixed Resistors
Resistor (0.1 in, 2.5 mm)
Resistor (0.3 in, 7.6 mm)
Resistor (0.4 in, 10.2 mm)
Resistor (0.5 in, 12.7 mm)
Resistor (0.6 in, 15.2 mm)
Resistor Packs
Page 1 of 1
Press F1 for help
Graphic Foreground 3.8, 1 in 0.1 in 100% Ready

Figure 6: Heartbeat Sensor PCB Design

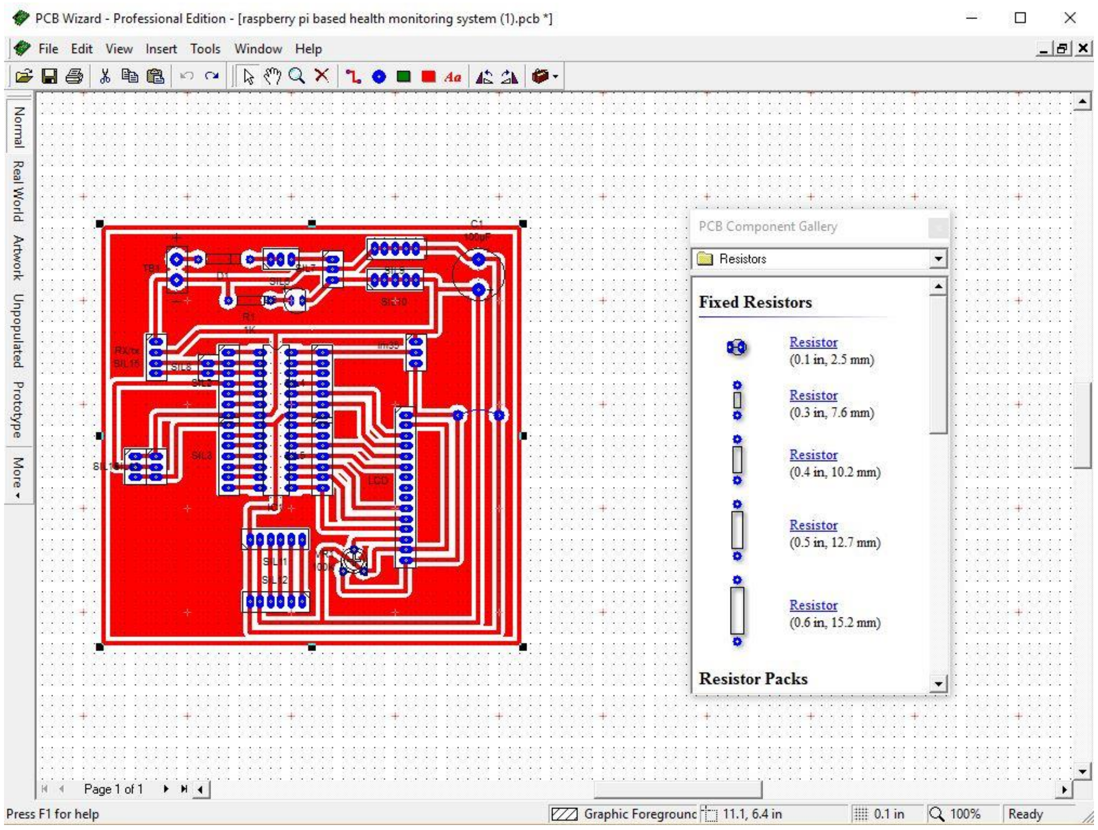

PCB Wizard - Professional Edition - [raspberry pi based health monitoring system (1).pcb *]
File Edit View Insert Tools Window Help
Normal Real World Artwork Unpopulated Prototype More
PCB Component Gallery
Resistors
Fixed Resistors
Resistor (0.1 in, 2.5 mm)
Resistor (0.3 in, 7.6 mm)
Resistor (0.4 in, 10.2 mm)
Resistor (0.5 in, 12.7 mm)
Resistor (0.6 in, 15.2 mm)
Resistor Packs
Page 1 of 1
Press F1 for help
Graphic Foreground 11.1, 6.4 in 0.1 in 100% Ready

Figure 7: PCB Design of ADC circuit

PCB Fabrication:

In the second step we take the print of the layout on a glossy paper using carbon ink. Now, cut the copper board according to the size of the layout using a cutter. Next, rub the copper side of PCB using an abrasive spongy scrubs. This removes the top oxide layer of copper as well as the photo resists layer. In the next step, we put the copper surface of the board on the printed layout. Ensure that the board is aligned correctly along the borders of the printed layout. And use tape to hold the board and the printed paper in the correct position.

Now, we iron it image side down to copper side. Heat up the electric iron to the maximum temperature. Here, the heat from the iron transfers the ink printed on the glossy paper to the copper plate. After ironing, place printed plate in luke warm water for around 10 minutes. Paper will dissolve, then remove paper gently. Remove the paper off by peeling it from a low angle.

Now, for etching the plate we dip the PCB into the etching solution (Ferric chloride solution, $FeCl3$) for approximately 30 mins. The $FeCl3$ reacts with the unmasked copper and removes the unwanted copper from the PCB. This process is called as Etching. Use pliers to take out the PCB and check if the entire unmasked area has been etched or not. In case it is not etched leave it for some more time in the solution. The reaction is given as:

$$Cu + FeCl3 = CuCl3 + Fe$$

Finally, drill holes using a PCB driller and solder all the components in their respective positions.

SENDING DATA TO CLOUD

Step I: First we need to install SMTP which is a light weight SMTP server used to send emails. Here SMTP stands for Simple Mail Transfer Protocol. To install SMTP first we need to update the system:

```
$ sudo apt-get install update
```

Then type the command for installing SSMTP:

```
$ sudo apt-get install ssmtp
```

Press “Y” if it prompts.

Step II: Next install 'mailutils' which is a set of libraries used to handle emails.

```
$ sudo apt-get install mailutils
```

Press “Y” if it prompts.

Step III: Now we need to edit the SSMTP configuration file. To open the SMTP type:

```
$ sudo nano /etc/ssmtp/ssmtp.conf
```

This will open a file with the following data already present.
Make sure that in the following lines:

```
-AuthUser=YourEmailID@gmail.com
-AuthPass=YourPassword
```

We should enter the email id and the correct password for the email id.
Save the file by pressing 'Control + X' followed by 'Y' and then 'press enter button'.

Step IV: To send the email, simply type the following command:

```
$ echo “Body” | mail -s “Mail Subject” ReceiverMailID@xyz.com
```

Step V: If we want to send the email after receiving the input from a sensor /switch then we will have to embed the command to send the email inside an 'if statement'.
But the command we have used is a shell command and it will not work directly in the python code. There is a library, which allows us to run a Shell Command inside a python code. The python code is attached in the Appendix 1.

For the emergency alert, that is when the heartbeat exceeds or changes to an abnormal value then message should be sent for this we found out an economical way by using a free EMAIL TO SMS GATEWAY named TEXT LOCAL and with this facility we can send direct sms alert to anyone by sending a simple email as explained below:

Step 1: Create your new email address that is your number followed by @sms.textlocal.in.
For Example: 8500xxxxxx@sms.textlocal.in

Step 2: Sending the mail to the respective Email-ID: hruday.95@gmail.com

The whole program is looped and it keeps on running until it is powered and the data keeps on getting accumulated into the drive of google and in case of any abnormality the text message will be sent to the respective person with readings of heart beat and temperate to alert the person for a very swift response.

FUTURE PROSPECT

- With the use of IOT we can make an automated responder or service that can react immediately to the situation and get the person to the nearest medical care available.
- As minimum technical or medical knowledge is needed to implement this device into practice, this device is user friendly device and can be used by anyone, anytime and at anywhere irrespective of the level of knowledge the person is having.
- Implementing this device in mobile phone would make it not only compatible but also popular as people would prefer this as they can inspect their health in reasonable price.
- We know how a doctor would study the heartbeat, which is done by counting the pulse of the person while the time is taken into account too, in that case human errors may occur. But in this we can avoid that and also heartbeat can be displayed on LCD.
- Interfacing a good display at reasonable cost for example a television screen or a computer screen which would thus help the user to get a better view and analysis of the output of the device.

CONCLUSION

Using this device the heart beat and the temperature of the patient is measured and is monitored continuously. The heartbeat sensor designed acts as a digital sensor and hence is directly interfaced with raspberry pi. Raspberry pi counts the heart beats per minute and displays it on the LCD screen. The data from the sensor is continuously sent to the mail and hence can be monitored continuously. In case of any abnormality an alert is sent to a predefined mobile number.

This device also contains temperature sensor which senses the surrounding temperature. The temperature sensor used is an analog device and hence we need an A/D converter to interface it with the raspberry pi. Using this ADC other analog sensors can also be interfaced. The temperature and heart beat readings obtained from the other devices available in the market are compared with the ones coming from the designed device and these readings are calibrated to achieve high precision and accuracy.

Our project is Micro Processor based heart beat monitoring on LCD is mainly intended to design a system, that can have access to internet so that alert can be sent when an abnormal value observed and respective care can be instantly taken for the patient who is being monitored**.** Using online carriers we achieved it. Thus the project has been successfully designed and tested.

APPENDIX

APPENDIX-1

Code for Sending Email from Raspberry pi in PYTHON:

```
import smtplib
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
from email.MIMEBase import MIMEBase
from email import encoders

fromaddr = "pihealthmonitor@gmail.com"
toaddr = "hruday.95@gmail.com"

msg = MIMEMultipart()

msg['From'] = fromaddr
msg['To'] = toaddr
msg['Subject'] = "SUBJECT"
data=987
data=str(data)

body = "TEXT YOU WANT TO SEND"+data

msg.attach(MIMEText(body, 'plain'))

server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login(fromaddr, "pimonitorhealth")
text = msg.as_string()
server.sendmail(fromaddr, toaddr, text)
server.quit()
```

APPENDIX-2

Code for Atmega 8L in using it as ADC:

```
#include <mega8.h>

#include <delay.h>
#include <stdlib.h>
// Alphanumeric LCD functions
#include <alcd.h>

// Standard Input/Output functions
#include <stdio.h>

// Voltage Reference: AVCC pin
#define ADC_VREF_TYPE ((0<<REFS1) | (1<<REFS0) | (0<<ADLAR))

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | ADC_VREF_TYPE;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Starting the AD conversion
ADCSRA|=(1<<ADSC);
// Waiting for the AD conversion to complete
while ((ADCSRA & (1<<ADIF))==0);
ADCSRA|=(1<<ADIF);
return ADCW;
}
void main(void)
{    int x;
float t,t1,t2;
char c[5];
char y;
```

```
// Input/output Ports i.e, Transmitter and Receiver initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
// Port C initialization
// Function: Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRC=(0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) |
(0<<PORTC1) | (0<<PORTC0);

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=(0<<CS02) | (0<<CS01) | (0<<CS00);
TCNT0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Disconnected
// OC1B output: Disconnected
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) |
(0<<WGM10);
TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (0<<CS12) | (0<<CS11) |
(0<<CS10);
TCNT1H=0x00;
TCNT1L=0x00;
```

```
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0<<AS2;
TCCR2=(0<<PWM2) | (0<<COM21) | (0<<COM20) | (0<<CTC2) | (0<<CS22) | (0<<CS21) | (0<<CS20);
TCNT2=0x00;
OCR2=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) | (0<<OCIE1B) | (0<<TOIE1) |
(0<<TOIE0);

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=(0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
```

```
UCSRA=(0<<RXC) | (0<<TXC) | (0<<UDRE) | (0<<FE) | (0<<DOR) | (0<<UPE) | (0<<U2X) |
(0<<MPCM);
UCSRB=(0<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (1<<RXEN) | (1<<TXEN) | (0<<UCSZ2) |
(0<<RXB8) | (0<<TXB8);
UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) | (1<<UCSZ1) |
(1<<UCSZ0) | (0<<UCPOL);
UBRRH=0x00;
UBRRL=0x33;



// ADC initialization
// ADC Clock frequency: 1000.000 kHz
// ADC Voltage Reference: AVCC pin
ADMUX=ADC_VREF_TYPE;
ADCSRA=(1<<ADEN) | (0<<ADSC) | (0<<ADFR) | (0<<ADIF) | (0<<ADIE) | (0<<ADPS2) |
(1<<ADPS1) | (1<<ADPS0);
SFIOR=(0<<ACME);

// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTB Bit 0
// RD - PORTB Bit 1
// EN - PORTB Bit 2
// D4 - PORTB Bit 4
// D5 - PORTB Bit 5
// D6 - PORTB Bit 6
// D7 - PORTB Bit 7
// Characters/line: 16
lcd_init(16);
 lcd_clear();
 lcd_putsf("Temperature");
 delay_ms(1000);
while (1)
```

```
    {
       y=getchar();
       if(y=='a')
       {
       x=read_adc(5);
       t=x*5.0;
       t1=t/1023;
       t2=t1*100;
       ftoa(t2,3,c);
       puts(c);

       }
       }
}
```

APPENDIX-3

Code for implementing the whole PROJECT in PYTHON:

```
import RPi.GPIO as GPIO

import time

import serial

import os

from array import *

import smtplib

from email.MIMEMultipart import MIMEMultipart

from email.MIMEText import MIMEText

from email.MIMEBase import MIMEBase

from email import encoders



count=0

temp= []

beat=0

temp2=0

tx= 0




#initiate Serial

ser = serial.Serial(

    port='/dev/ttyAMA0',

    baudrate = 9600,

    parity=serial.PARITY_NONE,

    stopbits=serial.STOPBITS_ONE,

    bytesize=serial.EIGHTBITS,

    timeout=1

   )

# Define GPIO to LCD mapping
```

```
LCD_RS = 40
LCD_E  = 38
LCD_D4 = 36
LCD_D5 = 32
LCD_D6 = 26
LCD_D7 = 24
Heart_out = 22

# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

def main():
  # Main program block

  GPIO.setwarnings(False)
  GPIO.setmode(GPIO.BOARD)       # Use BOARD PIN numbers
  GPIO.setup(LCD_E, GPIO.OUT)  # E
  GPIO.setup(LCD_RS, GPIO.OUT) # RS
  GPIO.setup(LCD_D4, GPIO.OUT) # DB4
  GPIO.setup(LCD_D5, GPIO.OUT) # DB5
  GPIO.setup(LCD_D6, GPIO.OUT) # DB6
  GPIO.setup(LCD_D7, GPIO.OUT) # DB7
  GPIO.setup(Heart_out, GPIO.IN) # HEART BEAT
  GPIO.setup(18,GPIO.IN)   #Reset
```

```
def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)

def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()

  # Low bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()

def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)
  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

if __name__ == '__main__':

  try:
    main()
  except KeyboardInterrupt:
    pass
 # finally:
 #   lcd_byte(0x01, LCD_CMD)
 #   lcd_string("Goodbye!",LCD_LINE_1)
    GPIO.cleanup()

# Initialise display

  lcd_init()
  while True:
    ser.write("a")

    for k in range(0,7):
      data=ser.read()
      if(data>=0):
	    temp.insert(k,data)



    temp1= "".join(temp)
    list([temp.pop() for z in range(len(temp))])
    print temp1
    temp2=float(temp1)
    temp2=temp2-6
    print temp2
 temp1=str(temp2)
 lcd_string("Temp-"+temp1,LCD_LINE_1)
 lcd_string("`C",LCD_LINE_1 +11)
 time.sleep(1)
 if(GPIO.input(18)==1):
         beat=0
         lcd_string("Heart beat-"+str(beat),LCD_LINE_2)
# lcd_byte(0x01,LCD_CMD)

 if(GPIO.input(22)==1):
  for j in range(0,301):
   if(GPIO.input(22)==1):
     beat+=1
     j+=1
     lcd_string("Heart beat-"+str(beat),LCD_LINE_2)
     lcd_string(str(j),LCD_LINE_1+14)
   time.sleep(0.2)

  f_beat=beat
  beatstr=str(beat)
  print beatstr
  import smtplib
  from email.MIMEMultipart import MIMEMultipart
  from email.MIMEText import MIMEText
  from email.MIMEBase import MIMEBase
  from email import encoders
  fromaddr = "pihealthmonitor@gmail.com"
  toaddr = "8500374878@sms.textlocal.in"
  msg = MIMEMultipart()
  msg['From'] = fromaddr
  msg['To'] = toaddr
  msg['Subject'] = "SUBJECT"
```

```
    body = "#%#YOUR HEART BEAT COUNT IS "+beatstr+" be careful and your temp is "+temp1+"'C
##"
    msg.attach(MIMEText(body, 'plain'))
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()
    server.login(fromaddr, "pimonitorhealth")
    text = msg.as_string()
    server.sendmail(fromaddr, toaddr, text)
    server.quit()
    beat=0
```

References

1. Jeong, In Cheol et al. (2006). “A new method to estimate arterial blood pressure using Photoplethysmographic signal”, Conf. Proc. IEEE Eng Med Biol Soc 1:4667-70

2. Warsuzarina Mat Jubadi and Siti Faridatul Aisyah Mohd Sahak (2009). "Heartbeat Monitoring Alert via SMS", Conf. Proc. IEEE Symposium on Industrial Electronics and Applications (ISIEA 2009), October 4-6, 2009, Kuala Lumpur, Malaysia.

3. Prajakta A. Pawar (2014). "Heart Rate Monitoring System using IR base Sensor & Arduino Uno ", IEEE 2014 conference on Health Industry and Business, 8-9 March, 2014, pg. 1-3., Indore, India.

4. M. Asaduzzaman Miah; Mir Hussain Kabir ; M. Siddiqur Rahman Tanveer ; M. A. H. Akhand (2015). "Continuous heart rate and body temperature monitoring system using Arduino UNO and Android device", IEEE 2015 2nd International Conference on Electrical Information and Communication Technology (EICT), 10-12 Dec. 2015, 183 – 188, India.

5. Prof. (Dr.) Yusuf Mulge and Poonam (2013). "Remote Temperature Monitoring Using LM35 sensor and Intimate Android user via C2DM Service", International Journal of Computer Science and Mobile Computing, IJCSMC, Vol. 2, Issue. 6, June 2013, pg.32 – 36.

6. LM 35 Data Sheet "LM35 Precision Centigrade Temperature Sensors". (http://www.ti.com/lit/ds/symlink/lm35.pdf)

7. DS3231 Data Sheet "DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal". (http://datasheets.maximintegrated.com/en/ds/DS3231.pdf)

8. S.Sara and Z.Bill (1999) "Photoplethysmograph". Retrieved on 11th October, 2008. (http://en.wikipedia.org/photoplethysmograph)