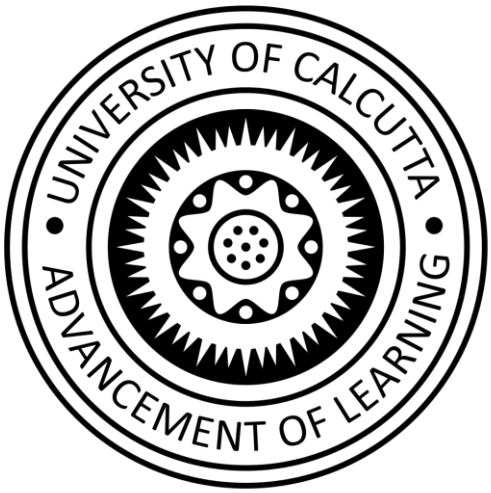


IOT PROJECT

HEALTH CARE CENTRE

GROUP-1

PRIYANKA GHOSH	91/MCA/160007
SANJIT KARMAKAR	91/MCA/160009
SOUMIK BOSE	91/MCA/160012
SANJOY DEY	91/MCA/160014

ABSTRACT

Internet of Things (IoT) is a computing process, where each physical object is equipped with sensors, microcontrollers and transceivers for empowering communication and is built with suitable protocol stacks which help them interacting with each other and communicating with the users. In IoT based healthcare, diverse distributed devices aggregate, analyse and communicate real time medical information to the cloud, thus making it possible to collect, store and analyse the large amount of data in several new forms and activate context based alarms. This novel information acquisition paradigm allows continuous and ubiquitous medical information access from any connected device over the Internet. As each one of the devices used in IoT are limited in battery power, it is optimal to minimise the power consumption to enhance the life of the healthcare system. This work explains the implementation of an IoT based In-hospital healthcare system using ZigBee mesh protocol. The healthcare system implementation can periodically monitor the physiological parameters of the In-hospital patients. Thus, IoT empowered devices simultaneously enhance the quality of care with regular monitoring and reduce the cost of care and actively engage in data collection and analysis of the same.

INTRODUCTION

Internet of Things means things interact with the Internet by employing sensors, microcontrollers and transceivers for empowering communication and is built with suitable protocol stacks which help them interacting with each other and communicating with the users, thus becoming the constitutive part of the Internet. Nowadays, Internet is impacting the several aspects of the potential user's everyday life. By keeping these things in view, several applications are developed based on IoT in which every physical object is connected to the Internet by employing sensor devices. The dependency of healthcare on IoT is increasing day by day to enhance the access to care, strengthen the quality of care and finally reduce the cost of care. Depending on an individual's unique biological, behavioural and cultural characteristics, the combined practice of wellbeing, healthcare and patient support is defined as personalised healthcare. This empowers each and every individual by following the basic healthcare principle of "the suitable care for the right person at the right time", which leads to more desirable results and improvement in satisfaction thus making healthcare cost effective. An efficient healthcare service should deal with prevention, early pathology detection and homecare instead of the high-priced clinical care. IoT ensures the personalisation of healthcare services by maintaining digital identity for each patient. Due to nonavailabilty of ready to access healthcare systems, many health problems have been getting undetected in conventional healthcare systems. But pervasive, non-invasive, powerful IoT based systems have been helpful in monitoring and analysing the patient data easily. In IoT based healthcare, various distributed devices gather, analyse and pass real time medical information to the cloud, thus making it possible to collect, store and analyse the big data streams in several new forms and activate context dependent alarms. This innovative data acquisition paradigm allows continuous and ubiquitous medical device access from any connected device over the Internet.

HARDWARE DETAILS

Arduino:

Overview:

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.

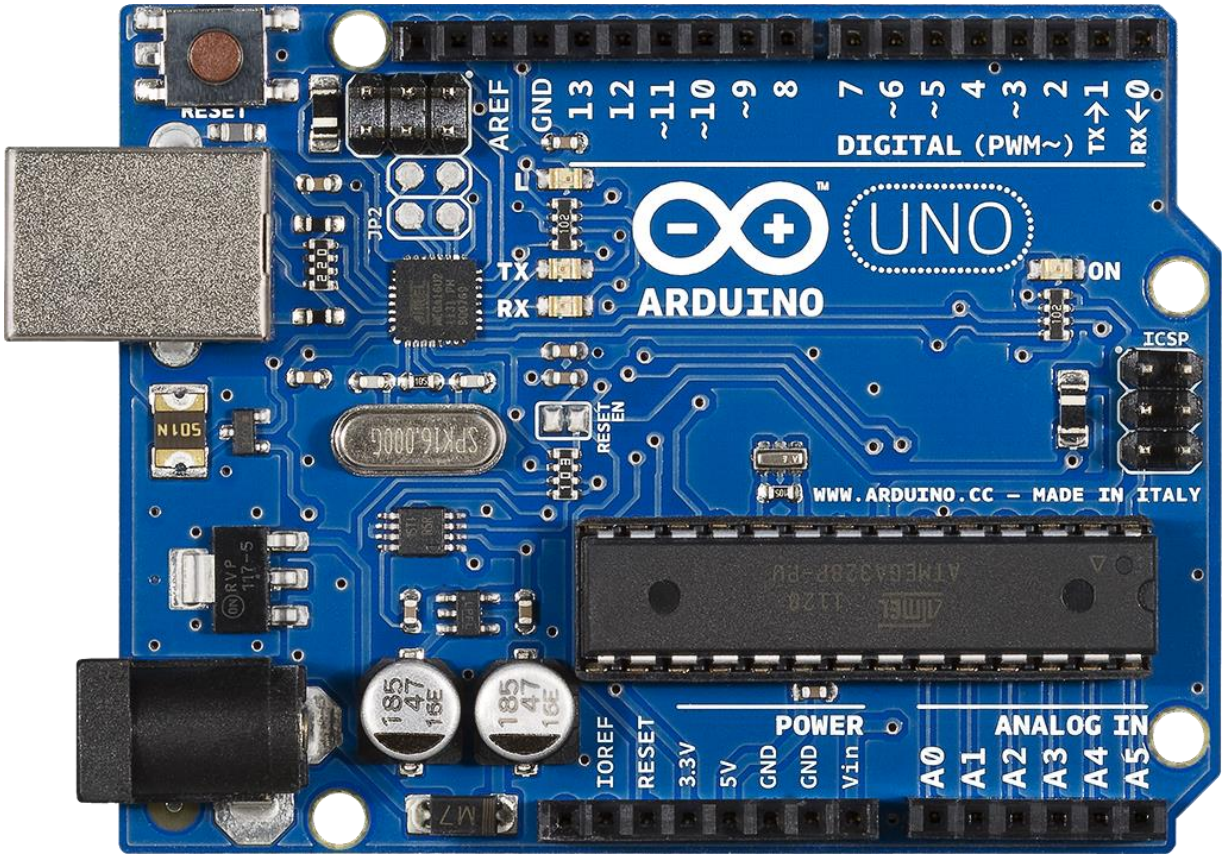
Revision 3 of the board has the following new features:

1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.

Stronger RESET circuit.

Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.



Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schematic.pdf

Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- 5V.This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

- GND. Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

• External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

• PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.

• SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

• LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

• TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

• AREF. Reference voltage for the analog inputs. Used with analogReference().

• Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.

- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following halfsecond or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

DS18B20

Programmable Resolution

1-Wire Digital Thermometer

FEATURES

Unique 1-Wire interface requires only one port pin for communication

Multidrop capability simplifies distributed temperature sensing applications

Requires no external components

Can be powered from data line. Power supply range is 3.0V to 5.5V

Zero standby power required

Measures temperatures from -55°C to +125°C. Fahrenheit equivalent is -67°F to +257°F

0.5°C accuracy from -10°C to +85°C

Thermometer resolution is programmable from 9 to 12 bits

Converts 12-bit temperature to digital word in 750 ms (max.)

User-definable, nonvolatile temperature alarm settings

Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)

Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN DESCRIPTION

GND - Ground

DQ - Data In/Out

VDD - Power Supply Voltage

NC - No Connect

DESCRIPTION

The DS18B20 Digital Thermometer provides 9 to 12-bit (configurable) temperature readings which indicate the temperature of the device.

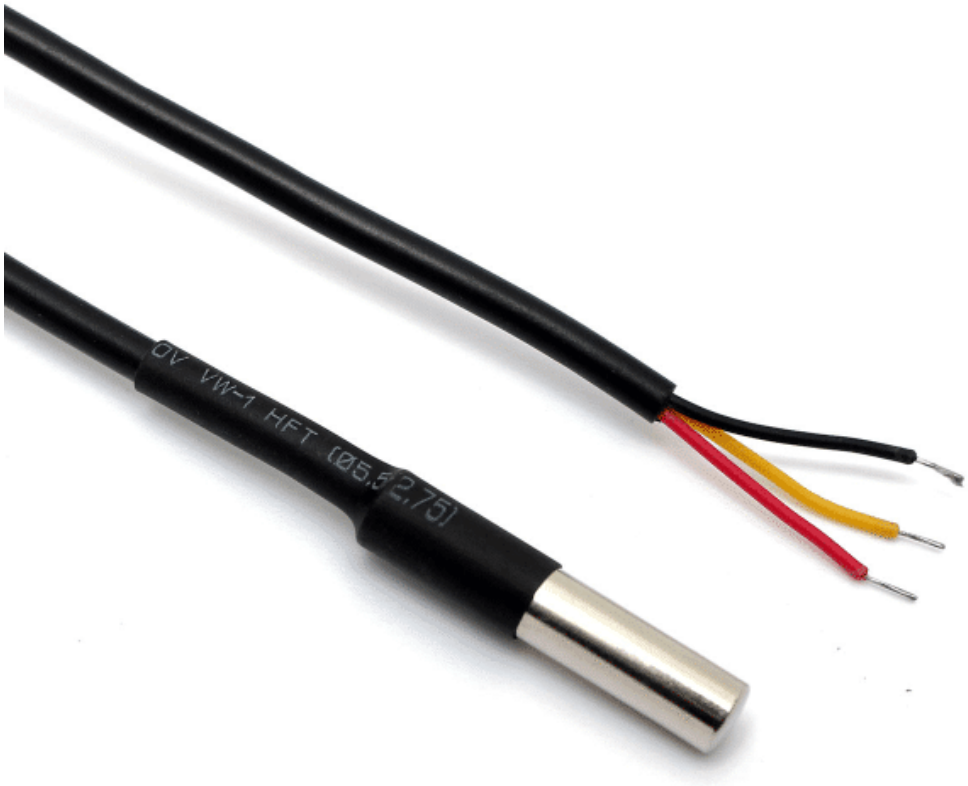
Information is sent to/from the DS18B20 over a 1-Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS18B20. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS18B20 contains a unique silicon serial number, multiple DS18B20s can exist on the same 1-Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls, sensing temperatures inside buildings, equipment or machinery, and process monitoring and control.

OVERVIEW

The block diagram of Figure 1 shows the major components of the DS18B20. The DS18B20 has four main data components: 1) 64-bit lasered ROM, 2) temperature sensor, 3) nonvolatile temperature alarm triggers TH and TL, and 4) a configuration register. The device derives its power from the 1-Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1-Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS18B20 may also be powered from an external 3 volt - 5.5 volt supply.

Communication to the DS18B20 is via a 1-Wire port. With the 1-Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64-bit lasered ROM portion of each device and can single out a specific device if many are present on the 1-Wire line as well as indicate to the bus master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands. One control function command instructs the DS18B20 to perform a temperature measurement. The result of this measurement will be placed in the DS18B20's scratch-pad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of 1 byte EEPROM each. If the alarm search command is not applied to the DS18B20, these registers may be used as general purpose user memory. The scratchpad also contains a configuration byte to set the desired resolution of the temperature to digital conversion. Writing TH, TL, and the configuration byte is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.



OPERATION - MEASURING TEMPERATURE

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the DS18B20 is configurable (9, 10, 11, or 12 bits), with 12-bit readings the factory default state. This equates to a temperature resolution of 0.5°C, 0.25°C, 0.125°C, or 0.0625°C. Following the issuance of the Convert T [44h] command, a temperature conversion is performed and the thermal data is stored in the scratchpad memory in a 16-bit, sign-extended two's complement format. The temperature information can be retrieved over the 1-Wire interface by issuing a Read Scratchpad [BEh] command once the conversion has been performed. The data is transferred over the 1-Wire bus, LSB first. The MSB of the temperature register contains the "sign" (S) bit, denoting whether the temperature is positive or negative.

Overview

PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensors range. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason they are commonly found in appliances and gadgets used in homes or businesses.

They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors.

PIRs are basically made of a pyroelectric sensor (which you can see below as the round metal can with a rectangular crystal in the center), which can detect levels of infrared radiation. Everything emits some low level radiation, and the hotter something is, the more radiation is emitted. The sensor in a motion detector is actually split in two halves. The reason for that is that we are looking to detect motion (change) not average IR levels. The two halves are wired up so that they cancel each other out. If one half sees more or less IR radiation than the other, the output will swing high or low Along with the pyroelectic sensor is a bunch of supporting circuitry, resistors and capacitors. It seems that most small hobbyist sensors use the BISS0001 ("Micro Power PIR Motion Detector IC"), undoubtedly a very inexpensive chip. This chip takes the output of the sensor and does some minor processing on it to emit a digital output pulse from the analog sensor.

How PIRs Work

PIR sensors are more complicated than many of the other sensors explained in these tutorials (like photocells, FSRs and tilt switches) because there are multiple variables that affect the sensors input and output. To begin explaining how a basic sensor works, we'll use this rather nice diagram

The PIR sensor itself has two slots in it, each slot is made of a special material that is sensitive to IR. The lens used here is not really doing much and so we see that the two slots can 'see' out past some distance (basically the sensitivity of the sensor). When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one half of the PIR sensor, which causes a *positive differential* change between the two halves. When the warm body leaves the sensing area, the reverse happens, whereby the sensor generates a negative differential change. These change pulses are what is detected.

The PIR Sensor

The IR sensor itself is housed in a hermetically sealed metal can to improve noise/temperature/humidity immunity. There is a window made of IR- transmissive material (typically coated silicon since that is very easy to come by) that protects the sensing element. Behind the window are the two balanced sensors.

Connecting to a PIR

Most PIR modules have a 3-pin connection at the side or bottom. The pinout may vary between modules so triple-check the pinout! It's often silkscreened on right next to the connection (at least, ours is!) One pin will be ground, another will be signal and the final one will be power. Power is usually 3-5VDC input but may be as high as 12V. Sometimes larger modules don't have direct output and instead just operate a relay in which case there is ground, power and the two switch connections.

The output of some relays may be 'open collector' - that means it requires a pullup resistor. If you're not getting a variable output be sure to try attaching a 10K pullup between the signal and power pins.

An easy way of prototyping with PIR sensors is to connect it to a breadboard since the connection port is 0.1" spacing. Some PIRs come with header on them already, the one's from adafruit have a straight 3-pin header on them for connecting a cable.

Using a PIR w/Arduino

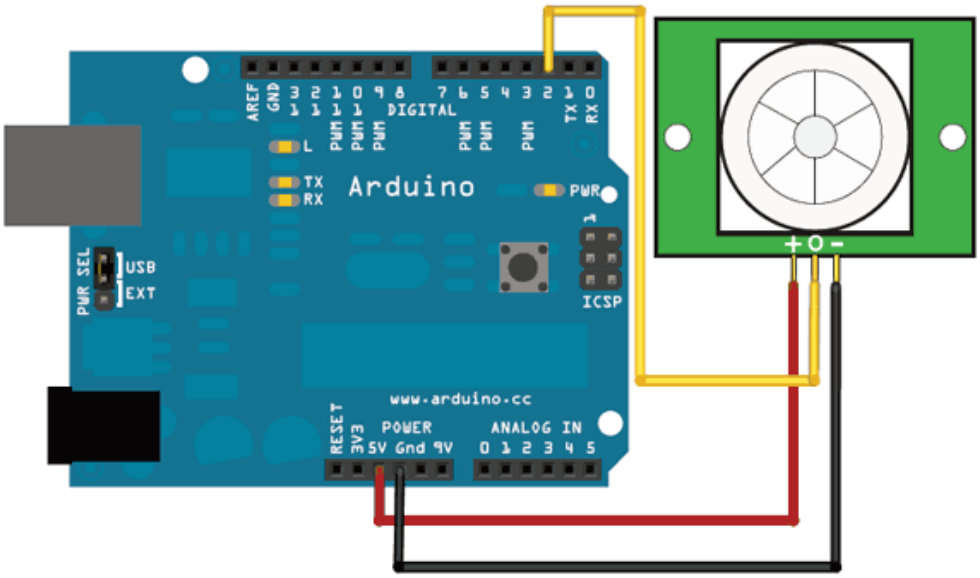
Reading PIR Sensors

The code is very simple, and is basically just keeps track of whether the input to pin 2 is high or low. It also tracks the *state* of the pin, so that it prints out a message when motion has started and stopped.

Connecting PIR sensors to a microcontroller is really simple. The PIR acts as a digital output, it can be high voltage or low voltage, so all you need to do is listen for the pin to flip high (detected) or low (not detected) by listening on a digital input on your Arduino

Its likely that you'll want reriggering, so be sure to put the jumper in the H position!

Power the PIR with 5V and connect ground to ground. Then connect the output to a digital pin. In this example we'll use pin 2.



Finger measuring heartbeat module

This project uses bright infrared (IR) LED and a phototransistor to detect the pulse of the finger, a red LED flashes with each pulse. Pulse monitor works as follows: The LED is the light side of the finger, and phototransistor on the other side of the finger, phototransistor used to obtain the flux emitted, when the blood pressure pulse by the finger when the resistance of the photo transistor will be slightly changed. The project's schematic circuit as shown, We chose a very high resistance resistor R1, because most of the light through the finger is absorbed, it is desirable that the phototransistor is sensitive enough. Resistance can be selected by experiment to get the best results. The most important is to keep the shield stray light into the phototransistor. For home lighting that is particularly important because the lights at home mostly based 50HZ or 60HZ fluctuate, so faint heartbeat will add considerable noise.

When running the program the measured values are printed. To get a real heartbeat from this could be challenging.

Connecting to Arduino

- Sensor pin S connect to Arduino pin Analoog 0 / A0
- Sensor pin + (middle pin) connect to Arduino pin 5+
- Sensor pin - connect to Arduino pin GND

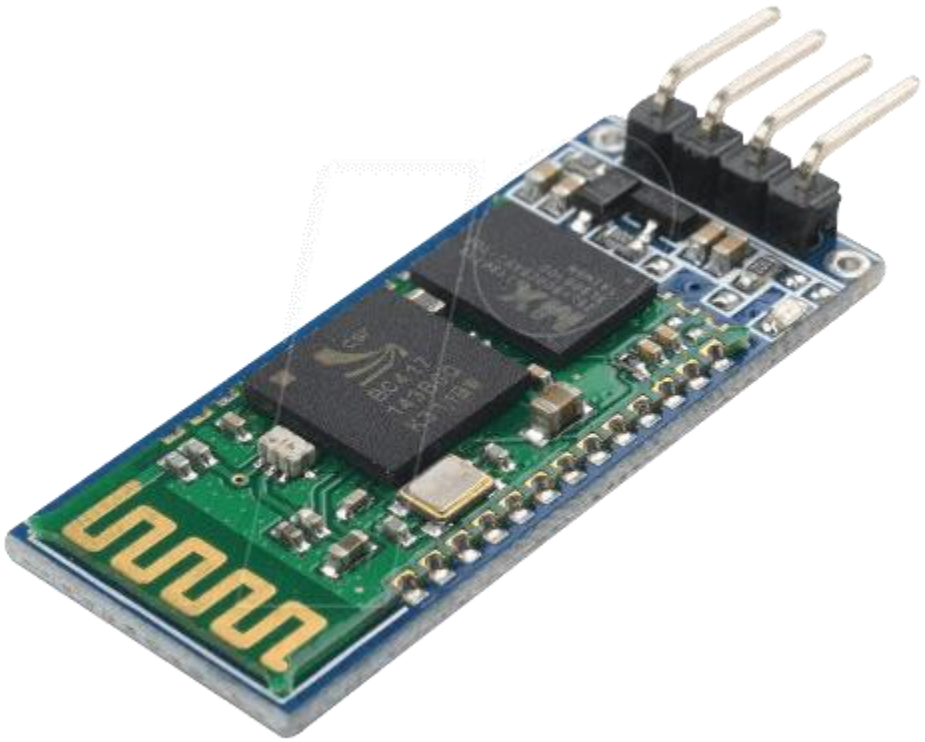
HC-05 Bluetooth Module

Introduction

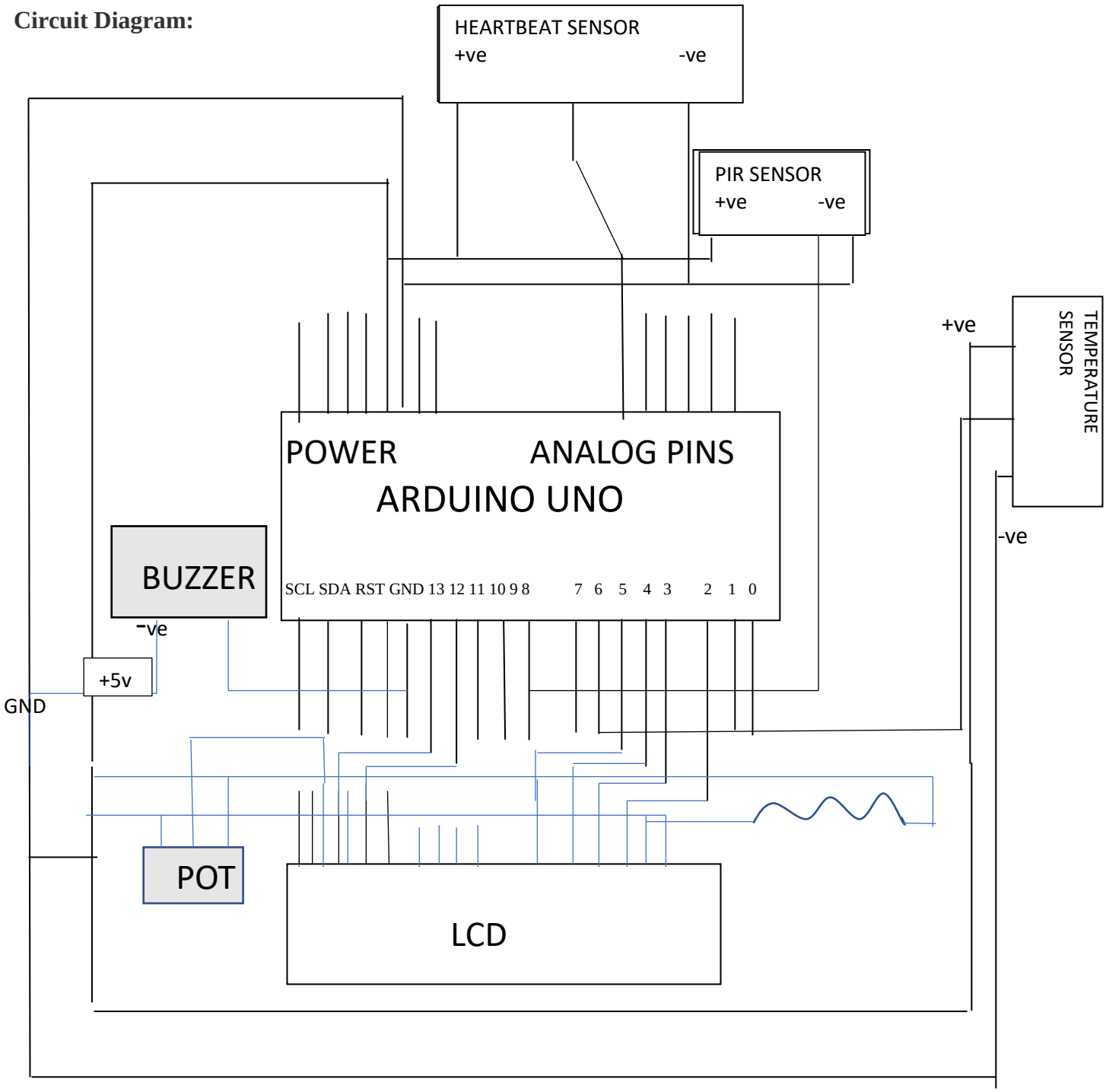
HC-05 Bluetooth Module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Its communication is via serial communication which makes an easy way to interface with controller or PC. HC-05 Bluetooth module provides switching mode between master and slave mode which means it able to use neither receiving nor transmitting data.

Specification:

- Model: HC-05
- Input Voltage: DC 5V
- Communication Method: Serial Communication
- Master and slave mode can be switched



Circuit Diagram:



CODING:

Code for Temperature Monitoring :

```
 #include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);
#define ONE_WIRE_BUS 6
OneWire oneWire(ONE_WIRE_BUS);

DallasTemperature sensors(&oneWire);

float Celcius=0;
float Fahrenheit=0;

void setup(void)
{
  Serial.begin(9600);
  sensors.begin();
  lcd.begin(16,2);
 lcd.setCursor(2,0);                 //Set LCD cursor position (column 2, row 0)
  }

void loop(void)
{
  sensors.requestTemperatures();
   delay(1000);
  Celcius=sensors.getTempCByIndex(0);
  Fahrenheit=sensors.toFahrenheit(Celcius);
  lcd.clear() ;
     lcd.setCursor(0, 0);              // Set LCD cursor position (column 0, row 0)
     lcd.print("Temp: ");
  Serial.print("C: ");
  lcd.print(Celcius);             //Print the temperature in celsius
  lcd.print(" C ,");

  Serial.print("F:");
  lcd.setCursor(0, 1);               // Set LCD cursor position (column 0, row 1)
  lcd.println(Fahrenheit);            //Print the temperature in Fahrenheit
  lcd.print(" F");
  delay(1000);
  }
```

Code for Motion Detection:

```
int buzpin=13;
int PIRpin=8;
int pirState = LOW;
int val=0;
int photocellPin=0;
int photocellReading;
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

void setup()
{
  pinMode(buzpin,OUTPUT);
  pinMode(PIRpin,INPUT);
  noTone(buzpin);              //Initially buzzer remains off
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.setCursor(2,0);             //Set LCD cursor position (column 2, row 0)
  lcd.print("P.I.R. Motion");
  lcd.setCursor(0,1);             //Set LCD cursor position (column 0, row 1)
  lcd.println("and Light Sensor");
  delay(2000);
  lcd.clear();
    lcd.setCursor(0,0);             //Set LCD cursor position (column 0, row 0)
    lcd.print("Processing data");
    delay(2000);
  lcd.clear();
    lcd.setCursor(3,0);              //Set LCD cursor position (column 3, row 0)
    Serial.println("NO Motion");
    lcd.print("NO Motion! ");          //Prints "No motion"
    lcd.setCursor(3,1);              //Set LCD cursor position (column 3, row 1)
    Serial.println("Waiting!");
      lcd.println("Waiting!");
}

void loop(){
  val = digitalRead(PIRpin);         //Taking input
  if (val == HIGH) {           // check if the input is HIGH
    delay(150);

    if (pirState == LOW) {
       tone(buzpin,100);          //buzzer starts ringing
      Serial.println("Motion detected!");
      lcd.clear() ;
      lcd.setCursor(0, 0);              // Set LCD cursor position (column 0, row 0)
      lcd.print("Motion Detected!");       //Prints "Motion Detected
       pirState = HIGH;
         delay(2000) ;
    }
```

```
  } else {
    noTone(buzpin);                  //buzzerstops ringing
    // display no motion screen saver
    scrollScreenSaver() ;
      if (pirState == HIGH){
      // There's no motion !

      // change to no motion detected
      pirState = LOW;
      }
  }
}

void scrollScreenSaver() {

  // autoscroll https://www.arduino.cc/en/Tutorial/LiquidCrystalAutoscroll
  lcd.clear() ;
    lcd.setCursor(15, 0);        // Set LCD cursor position (column 15, row 0)
    Serial.print("No Motion ");
  lcd.print("No Motion ");
  delay(500);
  lcd.setCursor(15, 1);        //Set LCD cursor position (column 15, row 1)
  Serial.println("Waiting !");
   lcd.println("Waiting !");
        // scroll 7 positions (display length - string length) to the left to move it back to center:

   for (int positionCounter = 0; positionCounter < 22; positionCounter++) {
                                                                          // scroll one position left:
    lcd.scrollDisplayLeft();
    // wait a bit:
    delay(150);

   }
}
```

Code for Heartbeat Monitoring:

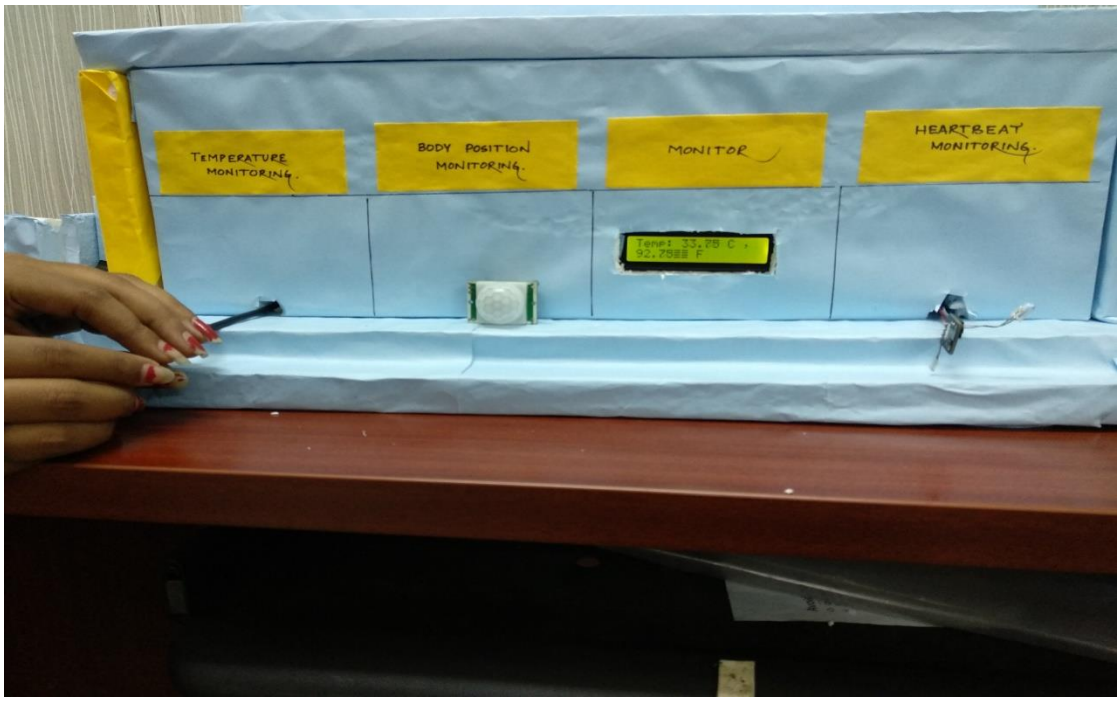
```
   int reading = 0;
   float BPM = 0.0;
  #include <LiquidCrystal.h>
   LiquidCrystal lcd(12,11,5,4,3,2);
   void setup(){
     Serial.begin(9600);
     lcd.begin(16,2);
   }
   void loop(){
      reading = analogRead(A0);
```

```
    BPM = (int)(60/(reading/1000.0));      //Change the analog reading into BPM
    if(BPM>150){
     lcd.setCursor(0,0);                   //Set LCD cursor position (column 0, row 0)
     lcd.print("Abnormal");
     Serial.println("Abnormal");
     }
     else{
      lcd.setCursor(0,0);                  //Set LCD cursor position (column 0, row 0)
      lcd.print("Normal");
      Serial.println("Normal");
     lcd.setCursor(0,1);                   //Set LCD cursor position (column 0, row 1)
     lcd.print("Heartbeat:");              //Print the heartbeat in BPM
    lcd.println(BPM);
     }
    delay(1000);
    lcd.clear();

    Serial.print("Heartbeat:");
    Serial.println(BPM);

    Serial.flush();
  }
```

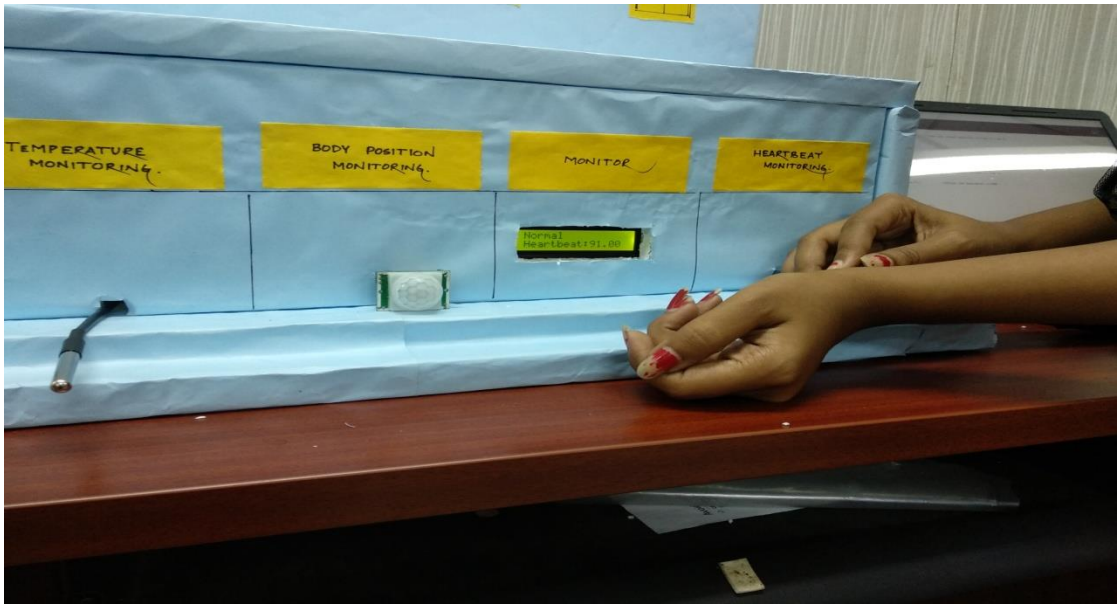
Code for Connecting Bluetooth Module:

```
char data = 0;                                  //Variable for storing received data
void setup()
{
   Serial.begin(9600);                         //Sets the baud for serial data transmission
   pinMode(13, OUTPUT);                        //Sets digital pin 13 as output pin
}
void loop()
{
  if(Serial.available() > 0)                   // Send data only when you receive data:
  {
     data = Serial.read();                     //Read the incoming data & store into data
     Serial.print(data);                       //Print Value inside data in Serial monitor
     Serial.print("\n");
     if(data == '1')                           // Checks whether value of data is equal to 1
       digitalWrite(13, HIGH);                 //If value is 1 then LED turns ON
     else if(data == '0')                      //  Checks whether value of data is equal to 0
       digitalWrite(13, LOW);                  //If value is 0 then LED turns OFF
  }
}
```

OUTPUTS:

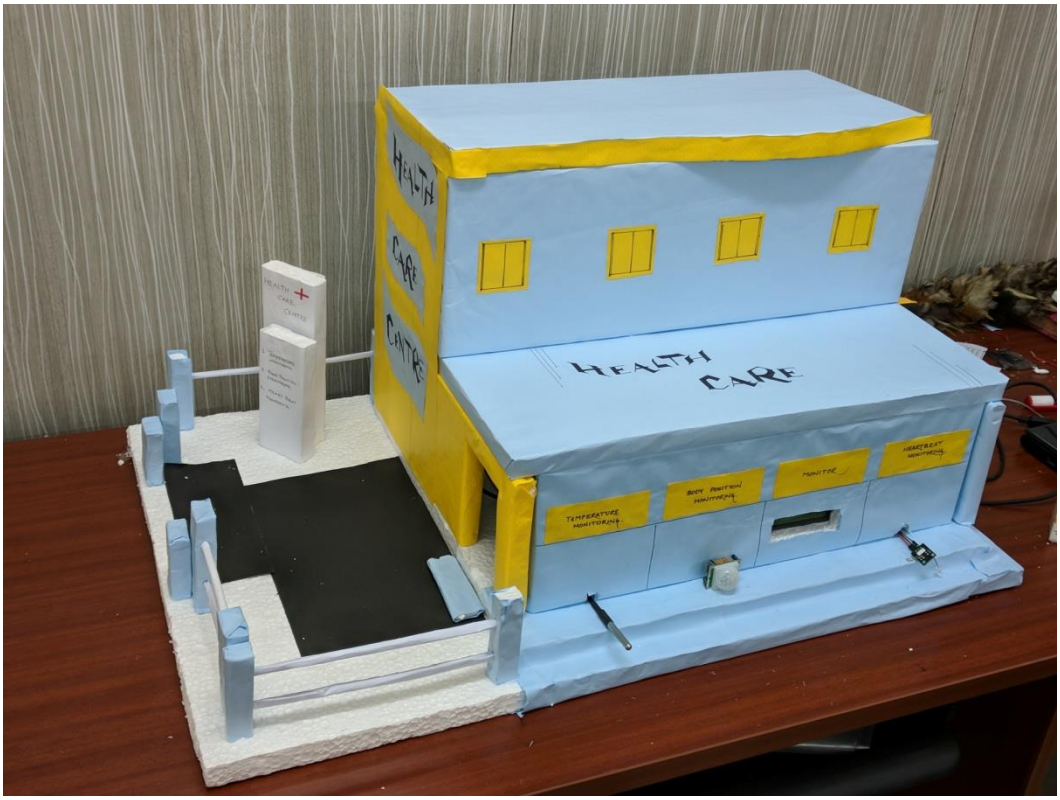


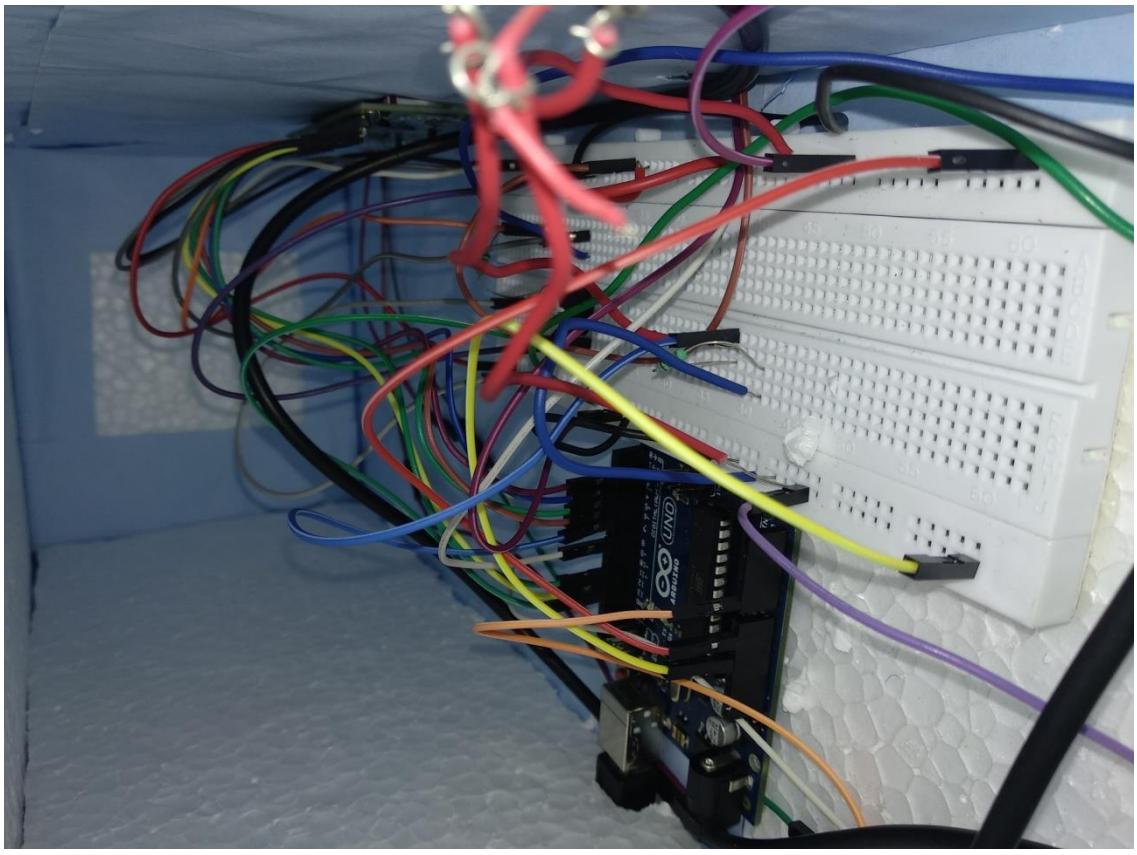
Output of Tempareture sensor



Output for Heartbeat Monitoring

MODEL PICTURES:


HEALTH
CARE
CENTRE
HEALTH
CARE
TEMPERATURE
MONITORING
BODY POSITION
MONITORING
PANIC/FIRE
HEARTBEAT
MONITORING



Precautions using ardunio

Hardware precautions

->Make sure that you have selected the right board that you have connected to your system check it by going in the Tools-Boards-select board.
->Do not put your Ardunio on a metal surface it will damage your board by conducting small currents beneath it.
->Do not apply voltage greater than 12volts to your board.
->Apply only direct current. Ardunio works only at DC(direct current).
->Do not remove the usb conector from PC when sketch is uploading.

Software precautions.

->when using serial communication digital pin 0 TX & 1 RX can not be used simultaneously.
->Any sensor connected to board returns values from 0 to 1023 and analog write takes only 255 so we have to convert it to max 255 range for analog Write.
->Using analog.read() uses analog INPUT pins only A0,A1,A2,A3,A4,A5.
->Using analog.write() uses analog WRITE pins only 3,5,6,9,10,11 (these pins can also be used as digital read and write)
->LiquidCrystal library uses .print(); to print string.
->println(); is print next statement from next line.
->where as Serial communication uses Serial.println(); statement