ROAD ACCIDENT PREVENTION UNIT (R.A.P.U)
(A Prototyping Approach to Mitigate an Omnipresent Threat)

By Dibakar Barua, Pranshu Jain, Jitesh Gupta and
Prof. Dhananjay V. Gadre
Dept. of Electronics and Communication

Netaji Subhas Institute of Technology
Azad Hind Fauj Marg, Sector 3, Dwarka, New Delhi -78
Contact: dibakar.barua92@gmail.com

Abstract

Road accidents claim a staggeringly high number of lives every year. From drunk driving, rash driving and driver distraction to visual impairment, over speeding and over-crowding of vehicles, the majority of road accidents occur because of some fault or the other of the driver/occupants of the vehicle. According to the report on "Road Accidents in India, 2011" by the Ministry of Transport and Highways, Government of India, approximately every 11th person out of 100,000 died in a road accident and further, every 37th person was injured in one, making it an alarming situation for a completely unnecessary cause of death.
The above survey also concluded that in 77.5 percent of the cases, the driver of the vehicle was at fault. The situation makes it a necessity to target the root cause of road accidents in order to avoid them. While car manufacturers include a system for avoiding damages to the driver and the vehicle, no real steps have been taken to actually avoid accidents. "Road Accident Prevention Unit" is a step forward in this stead. This design monitors the driver's state using multiple sensors and looks for triggers that can cause accidents, such as alcohol in the driver's breath and driver fatigue or distraction. When an alert situation is detected, the system informs the driver and tries to alert him. If the driver does not respond within a stipulated time, the system turns on a distress signal outside the vehicle to inform nearby drivers and sends a text message to the driver's next of kin about the situation. A marketable design would also shut down power to the vehicle, thus providing maximum probability for avoiding road accidents and extending a crucial window for preventive and mitigation measures to be taken.

1. Introduction

According to the study conducted by the Ministry of Transport and Highways, only 9 percent of the accidents observed were attributed to material causes such as faults in the road, weather conditions, vehicular defects etc. and a meager 3.7 percent of the accidents were caused when a cyclist or pedestrian was at fault. Further, 60 percent of the driver-caused road accidents were attributed to over speeding, 16.7 percent of these were due to alcohol or drug consumption and lastly 23.6 percent were caused due to driver fatigue or overcrowding of vehicles. These clearly bring to light the gravity of the situation and the enormous responsibility of vehicle drivers towards causing road accidents. "R.A.P.U" is motivated by the desire to curb such incidents which are caused due to a moment of madness or complete irresponsibility of the driver as such situations are easily avoidable. A life lost in a road accident is unforeseen and absolutely unnecessary, making the addressing of the situation a complete must. The technical content of R.A.P.U's design is inspired by the module suggested in "Accident Prevention Using Eye Blinking and Head Movement", published in the International Journal for Computer Applications® (IJCA) [1]. Our goal was to make the suggested design simpler to implement, low cost, low power, easily installable and extendable. R.A.P.U has been dealt with in an entry level manner

and has been tested for premium effectiveness.

1.1 Technical Background

This issue has previously been dealt with in quite detail in “Accident Prevention Using Eye Blinking and Head Movement”, published in the International Journal for Computer Applications® (IJCA) [1].
The author suggested the use of spontaneous video capture and image processing to analyze the driver’s state. However the key factors inhibiting the driver’s ability i.e. eye blinking and head tilting can be easily analyzed using low cost sensors such as accelerometers and IR sensors, thus obviating the need for expensive equipment as well as advanced processing platforms. Including these in the “Road Accident Prevention Unit” has made the design a low cost, low power implementation which can be easily installed and changed. The accident prevention and mitigation steps involve a Distress Signal visualization outside the vehicle using a Display Board, alarm intimation for the driver and a text message sent to the next of kin with the GPS coordinates of the driver upon non-attendance of the situation.

1.2 What does R.A.P.U do?

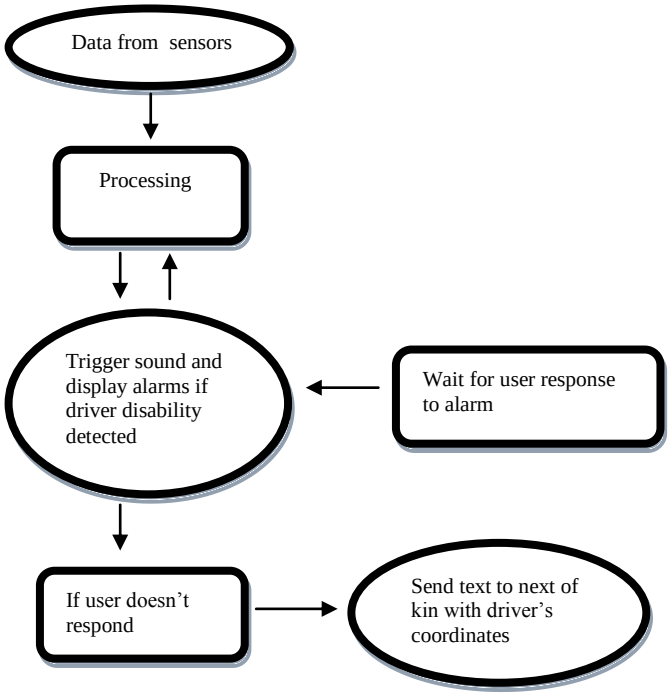


Figure 1: R.A.P.U, a bird’s eye view

Figure 1 shows a top level block diagram of the entire system. The key responsibility of the design is to constantly fetch data from the sensors regarding the driver’s ability to drive safely. If the system renders the driver incapable to do so, appropriate steps are taken to notify the driver’s nearby vehicles and the driver’s next of kin of the situation so that suitable prevention and mitigation steps can be taken. The design has been made in a modular approach in such a way that it can be easily installed in any vehicle. The modules can be individually installed as per the need of the user. Overall the system is a low cost and low power solution to the issue which has enormous potential for adaptation and expansion to be made road-worthy.

1.3 Organization of the report

The paper describes the technical aspects of R.A.P.U in detail. The top level hardware and software modules are described in Section 2 along with the approach taken to implement the system.

Section 3 deals with the Hardware and Software design of the system extensively. The various analog components in the front as well as back end are explained. The software algorithms and interfacing with the various modules of the system are also explained.

Section 4 details the conclusions and results of the tests that were carried out on R.A.P.U. Tests were carried out on multiple subjects in different environments and the results of the trials have been suitably drafted. Software trials of the design have also been explained.

The future scope of R.A.P.U, which is essentially its strong point, and its limitations and issues have also been dealt with.

The report ends with the details of the PCB designs and the software model that have been used in the system, provided in Appendix A and B.

2. Proposed Solution

2.1 Hardware Components

The hardware model of R.A.P.U has been broadly divided into the following modules:

a. Sensor Interface proto-shield for attaching all peripherals to microcontroller unit.
b. Sensor and Alarm Platform which has all the sensors and a speaker unit to be mounted on the driver.
c. Display Unit to be mounted outside the vehicle.
d. Amplification and filtering Unit for sending enlarged Alarm Sound to the speaker's Sensor framework.
e. LCD unit for visual interface to the system.
f. Communication Unit for using the GSM and GPS modules to send a text message.
g. Power Supplies for powering the entire device and the GSM module.

A few assumptions have been made with regard to the hardware implementation of R.A.P.U:

a. The IR sensor has been calibrated for the ambient setting before using the device.
b. The Sensor unit does not cause any interference with the driver's ability.
c. The Display Shield is visible to the drivers nearby.
d. The slight delay between Alarm Trigger and message being received by the next of kin is inconsequential and of no significant value.
e. The power supplies designed can be easily modified to draw power from the vehicle.

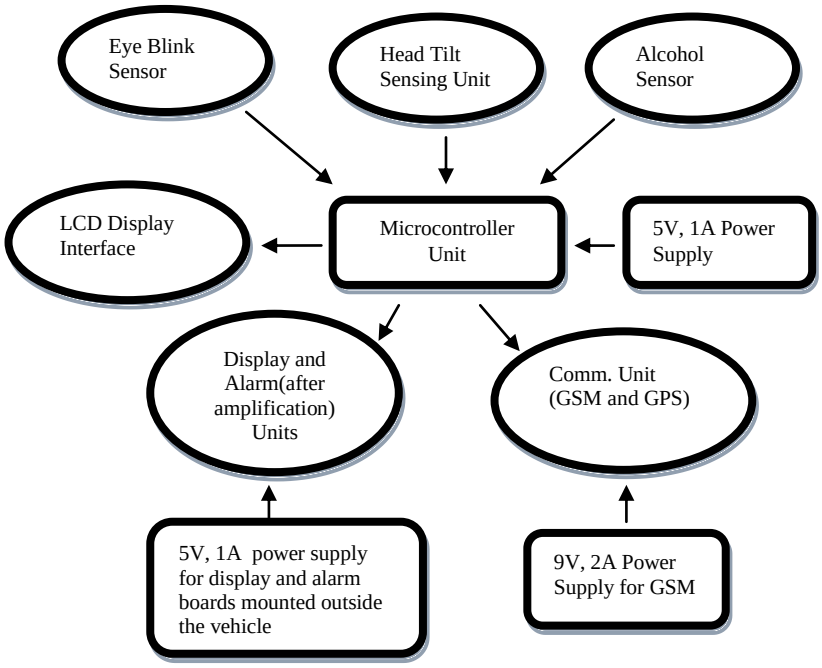


Figure 2: Hardware Model

There are certain constraints that R.A.P.U's hardware design inflicts upon the user. Firstly the user has to wear the sensor unit in order to use the system which is an added responsibility along with the vehicle. The Communication Unit has a fixed contact number programmed into it to which the message will be sent. Also, the GSM unit uses a larger power supply separate from the system.

The analog components of the system are involved in the backend of the hardware design. The Infra Red sensor utilizes an LM358 op-amp for amplification of the signal so that it can be calibrated to adjust to ambient light settings. The Speaker unit uses an Amplification Unit which uses an audio grade LM386 op amp for amplification and filtering. The primary power supply uses an LM723 Voltage Regulator by Texas Instruments to provide a stable 5V, 1A power supply.

2.2 Software Components

The Software model of the design consists of the following:

1. Display of system status on the Device Liquid Crystal Display.
2. Auto Calibration of Accelerometer to adjust to driver position upon System Reset.
3. Reading the sensor data in a polled fashion.
4. Triggering the Alcohol Alarm if alcohol is detected in driver's breath.
5. Triggering the Fatigue Alarm if Head Tilt or Eye Blinks are detected.
6. Triggering the Communication Unit to send a text message with GPS coordinates as final steps.

3. Implementation

3.1 Hardware Implementation

The various Hardware modules used were:

a. Sensor Unit: comprising of a MQ2 alcohol and combustible gas sensor, an MMA7760 Low-G accelerometer and an Infra Red Sensor Breakout Board consisting of an LM358 Op Amp by Texas Instruments and a potentiometer for calibration.

b. Alarm Device consisting of a Display Board made using Seven Segment Displays and a 5V relay and a Speaker Unit consisting of two 4 Ohm Speakers, a amplification and filtering unit based on the audio grade LM386 Op Amp by Texas Instruments and a 3.5 mm jack based Headphone unit.

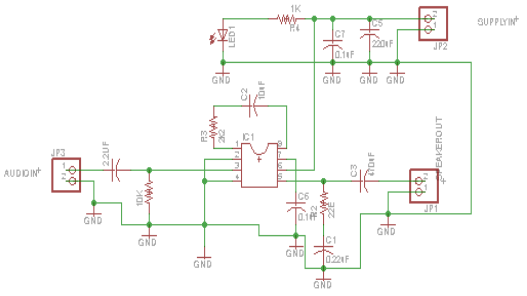


Figure 3: Schematic for Sound Amplifier

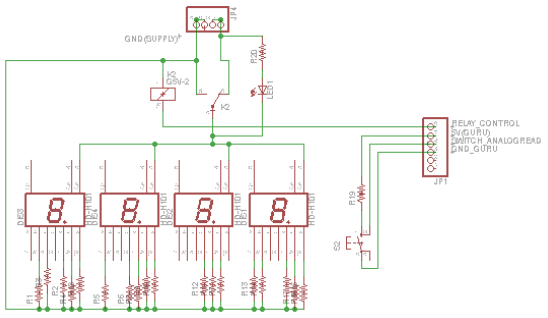


Figure 4: Schematic for Display

c. LCD Unit consisting of an HD44780 based 16X2 Liquid Crystal Display as a System Interface.

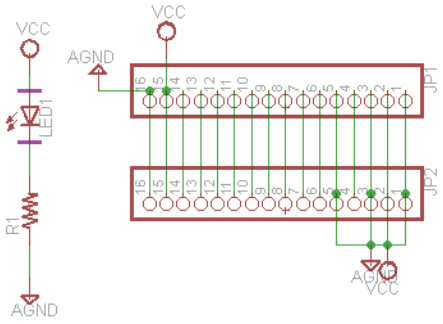


Figure 5: Schematic for LCD Board

d. Communication Unit consisting of a GNS TC6000-GN GPS Unit by Texas Instruments, a SIMCOM GSM Module along with a 9V, 2A Power Supply and an AT Mega 328 based microcontroller board.

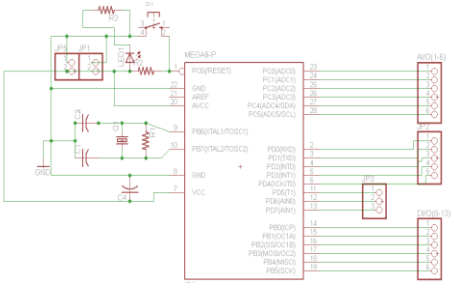


Figure 6: Schematic for Communication Unit

e. A 5V, 1A Power Supply based on the LM723 voltage regulator by Texas Instruments.

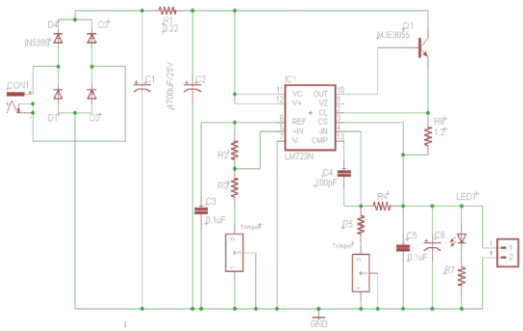


Figure 7: Schematic for Power Supply

f. The Microcontroller Unit comprising of the Stellaris Guru Evaluation Kit by Texas Instruments based on LM3S608 MCU and a proto-shield for peripheral interfacing.

The various integrated circuits that have been used are:

a. LM358 Op Amp: has been used in the IR Sensor Board for amplification of the sensed voltage for adjustment using a potentiometer. This serves as a calibration for adjusting to the ambient lighting. This IC was particularly chosen due to its single 5V supply requirement which was easily provided by our microcontroller.

b. LM386 Op Amp: has been used in the Audio Amplifier circuitry. This IC has been chosen due to its audio grade properties and single supply requirement which helps provide a distinguished sound output.

c. LM723 Voltage Regulator: has been used in the 5V, 1A Primary Power Supply. This has been used due to its ready customizability by changing the output voltage by simply changing the value of a potentiometer.

d. LM3S608 Stellaris Cortex M3 Microcontroller: has been used in the Stellaris Guru Evaluation Kit as the primary processor. The Stellaris Family has been chosen due its fast processing capability, Low Power working and sufficient memory provisions which adhere appropriately to the system design.

e. AT Mega 328 Microcontroller: has been used in the Communication Module for easy interfacing of the GSM and GPS kits which require a 5V-GND digital communication protocol.

3.2 Software Implementation

The software model follows a modular approach consisting of the following components:

a. The LCD Display Library written exclusively for the Stellaris Cortex-M3 Peripheral Driver Library
b. The Auto Calibration of the Accelerometer Reference upon System Reset.
c. Sensor reads done in a polling fashion.
d. The Alarm State for fatigued state which can be escape within a period of 10 seconds using a button which is serviced as a hardware interrupt.
e. The Speaker and Relay Control using the Stellaris Driver Library achieved by configuration of the GPIO output current set to 8mA.

The software model is best described by the following framework diagram:

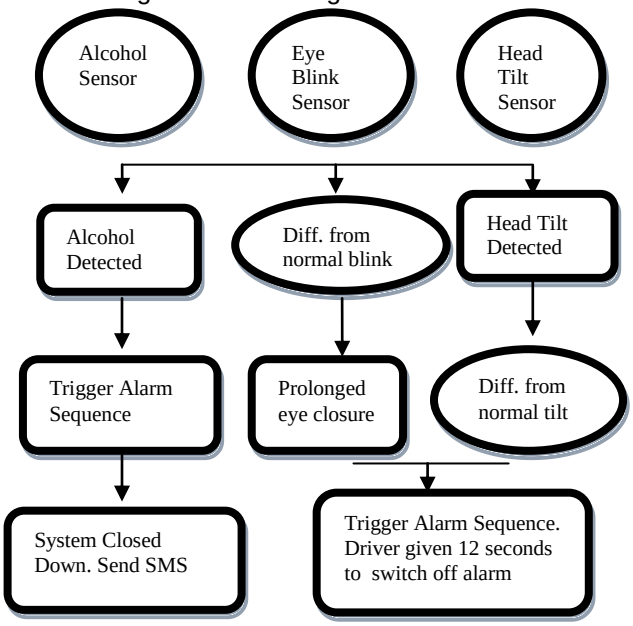


Figure 8: Software Framework Overview

As shown, the sensors are read in a polled fashion with the alcohol sensor readings checked first. Even one trigger in this case locks the system down and sends the SMS. The eye blinks and head tilts are differentiated from arbitrary head movements and eye blinks by re-reading of the sensors upon detection.
The device uses an Auto Calibration Algorithm upon System reset to learn the the driver's head's reference coordinates as per his seating configuration. The deviations from these values are considered for detecting his head movement.

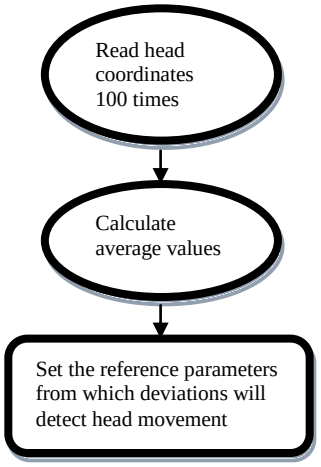


Figure 9: Auto Calibration Procedure for Driver's Head Configuration

The Alarm and Display Board of the system use a 5V relay which is controlled by the 3.3V GPIO output of the microcontroller using a software-modified pad current of 8mA. The Push Button which allows the driver to escape the Alarm Sequence is configured as a Hardware Interrupt Controlled GPIO Input for immediate redressal and to ensure the driver only has to press it once at any time once the Alarm is triggered.

4. Results

The system devised is a Low Power and Easily Installable Modular device. The hardware components were tested for maximum Power Consumption. R.A.P.U has an absolute maximum current rating of 38mA when all the GPIO Outputs are loading the processor working at a voltage of 3.3V providing an ultra low power treatment.

The device achieved a user-independent compatibility for eye blink and head tilt detection through rigorous testing to determine the most appropriate comparison counters.

1. **Eye Blink Detection**: The I.R Sensor used for Eye Blink Detection uses an I.R transmitter for throwing I.R light at the user's eye. The open eye absorbs this light while the closed eye reflects it, thus allowing us to determine the eye state. This mechanism is adjustable using a potentiometer to ensure that the ambient lighting does not affect the device working. This was proved to be a better alternative than Image Processing as camera readings are light dependent and the processing requirements are also less in R.A.P.U.

The device uses a re-reading mechanism to ensure that stray blinks to not trigger the system. It was determined that a closed reading of 12 values out of 15 resembled an eye that was closed for approximately 2-3 seconds in our detection scheme.

2. Head Tilt Detection: The device measures deviations from the reference head coordinates, which are a learned upon System Reset, to determine head movement. A significant deviation triggers a re-reading mechanism to fully ascertain the head movement to ensure that stray movements such as on vehicle turns, music etc. are not detected.

5. Conclusions

R.A.P.U is an easy-to-install system that can be implemented in most situations. However if the driver actually fails to wear the Sensor Framework Unit, the device cannot be used. Also there is always the issue of adjustment of the calibration potentiometer in the I.R Sensor to ensure the ambient light does not interfere with the sensing process.

R.A.P.U's true potential lays in the future implementations which are possible to make

the system a complete accident detection and prevention unit.

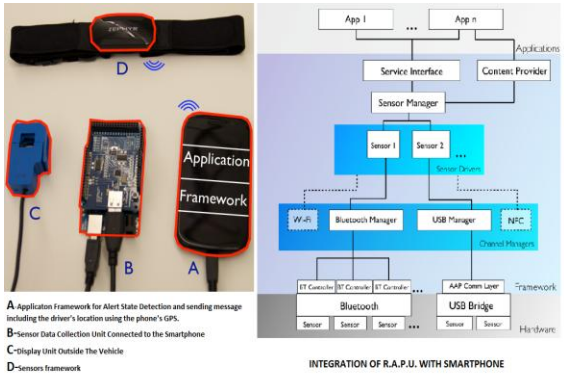
1. **Automatic braking system** (to slow down the speed of car) or lock down of car (in case of alcoholic state) can also be implemented to improve the efficiency of RAPU.

2. To monitor the **pulse rate** of the driver to observe any sudden change so that any situation of panic can also be pre-detected.

3. The sensor framework can be made a **Wireless** node using a secondary processor and sensors to achieve a complete modular system.

4. Provisions can be made for detecting **Rash Driving** using a large number of sensors and suitable sensing algorithms.

5. R.A.P.U As Mobile Based Device



INTEGRATION OF R.A.P.U. WITH SMARTPHONE

R.A.P.U will be a complete and ubiquitous system if the device is configured as a Mobile Based Framework which configures itself to the user's smartphone using Bluetooth. Not only will this obviate the need for expensive GSM and GPS Units but also allow the device to use the mobile's connectivity to achieve instant reactivity to Trigger Situations. Android applications for the User End and the Traffic Controller/ Police Booth End will allow the device to upload the details of an accident-prone scenario directly to the corresponding servers to achieve immediate addressal of such a scenario.

References

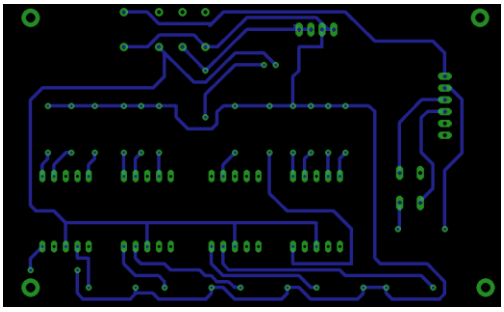
1. Accident Prevention "Using Eye Blinking and Head Movement", by Abhi R. Varma, Chetna Bharti and Seema V. Arote, Proceedings published in International Journal of Computer Applications® (IJCA), Emerging Trends in Computer Science and Information Technology - 2012(ETCSIT2012).

2. "Head Mounted Input Device using MEMS Sensors", by Anbarasu V (Research scholar, Sathyabama University) and Dr. T. Ravi (Professor & Head, Dept of CSE, KCG College of Technology), Proceedings Published in Indian Journal of Computer Science and Engineering (IJCSE)

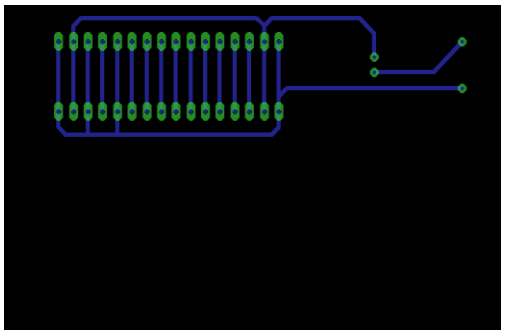
3. "Accident Avoidance and Prevention on Highways" S.P Bhumkar, V.V Deotare, R.V Babar , Singhad Institute of Technology, Lonavala, Processidings published in International Journal of Engineering Trends and Technology (IJETT Vol. 3, Issue 2, 2012)

Appendix A : PCB DESIGN

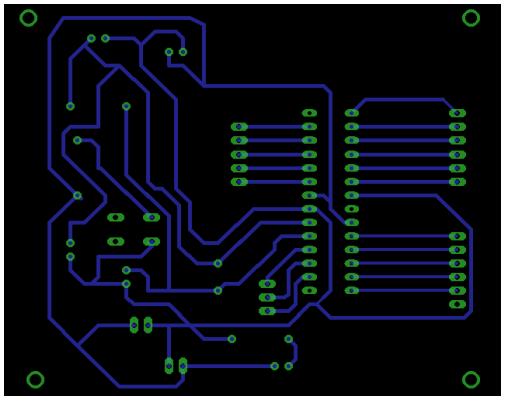
The various PCB's designed for the device were :



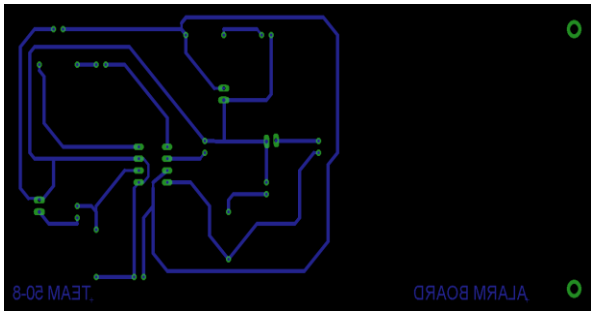
1 . Board for the External Display Unit



2 . Board for the LCD Interface Unit



3. Board for the Communication Unit



4 . Board for the Alarm's Audio Ampllifier

APPENDIX B: SOFTWARE PROGRAM

```
/* Centre for Electronic Design and Technology, NSIT, New Delhi
   Dibakar Barua
   Pranshu Jain
   Jitesh Gupta
*/

/* Defines the base address of the memories and peripherals */
#include "inc/hw_memmap.h"

/* Defines the common types and macros */
#include "inc/hw_types.h"

/* Defines and Macros for GPIO API */
#include "driverlib/gpio.h"

/* Prototypes for the system control driver */
#include "driverlib/sysctl.h"

/* Header files for various peripherals */

#include "driverlib/adc.h"

#include "driverlib/uart.h"

#include "driverlib/interrupt.h"

#include "utils/uartstdio.h"

/* Prototypes for the SysTick driver. */
#include "driverlib/systick.h"

/*  Macros that define the interrupt assignment on Stellaris. */
#include "inc/hw_ints.h"


/*macros for GPIO digital bases*/

#define digital2base GPIO_PORTB_BASE
#define digital3base GPIO_PORTB_BASE
#define digital4base GPIO_PORTB_BASE
#define digital5base GPIO_PORTB_BASE
#define digital6base GPIO_PORTB_BASE
#define digital7base GPIO_PORTB_BASE
#define digital8base GPIO_PORTC_BASE
#define digital9base GPIO_PORTD_BASE
#define digital10base GPIO_PORTA_BASE
#define digital11base GPIO_PORTA_BASE
#define digital12base GPIO_PORTA_BASE
#define digital13base GPIO_PORTA_BASE

/*macros for GPIO digital pins*/

#define digital2pin GPIO_PIN_5
#define digital3pin GPIO_PIN_4
#define digital4pin GPIO_PIN_1
#define digital5pin GPIO_PIN_0
#define digital6pin GPIO_PIN_6
#define digital7pin GPIO_PIN_7
#define digital8pin GPIO_PIN_4
#define digital9pin GPIO_PIN_0
#define digital10pin GPIO_PIN_3
#define digital11pin GPIO_PIN_5
#define digital12pin GPIO_PIN_4
#define digital13pin GPIO_PIN_2

/*macros for digital relay control*/
#define relaycontrolbase digital8base
#define relaycontrolpin                 digital8pin

/*macros for digital speaker control*/
#define speakerbase digital9base
#define speakerpin digital9pin

/*LCD Pins connection
LCD                                     Guru
D4                                      D2(PB5)
D5                                      D3(PB4)
D6                                      D4(PB1)
D7                                      D5(PB0)
E                                       D6(PB6)
```

```
RS                                  D7(PB7)
R/W                                 GND
*/

/*Variables*/

//Store Sensor Values
unsigned long temp[8];

//for checking eye blinks/eyes being closed
unsigned int blinkcounter = 0;

//loop variables
unsigned long i = 0;
unsigned long count = 0;
unsigned long count1 = 0;
unsigned long count2 = 0;
unsigned long counter = 0;

//Variables to Store sensor Values
unsigned long eyesense = 0;
unsigned long alcoholsense = 0;
unsigned long butval = 0;
unsigned long xval = 0;
unsigned long zval = 0;
unsigned int deltax = 0;
unsigned int deltaz = 0;

//switch interrupt counter
volatile unsigned long g_ulcount = 0;

// macros for LCD Commands
#define CLEARSCREEN 0x1
#define GOTO_FIRSTROW_THIRDCOLUMN 0x82
#define GOTO_FIRSTROW_FIFTHCOLUMN 0x84
#define GOTO_SECONDROW_THIRDCOLUMN 0xC3
#define GOTO_SECONDROW_FIRSTCOLUMN 0xC0

/* Delay Function */
void delay(unsigned long ulSeconds)
{
   /* Loop while there are more seconds to wait. */
                                    while(ulSeconds--)
   {
      /* Wait until the SysTick value is less than 1000. */
      while(SysTickValueGet() > 1000)
      {
      }

      /* Wait until the SysTick value is greater than
1000. */
      while(SysTickValueGet() < 1000)
      {
      }
   }
}

/* Function definitions for the LCD */

void LCD_Command(char ch)
{
int y=2;
 while(y>0)
{
GPIOPinWrite(GPIO_PORTB_BASE,
GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_6);
if(y==2)
{

GPIOPinWrite(GPIO_PORTB_BASE,      GPIO_PIN_0|
GPIO_PIN_1|      GPIO_PIN_4|      GPIO_PIN_5,
0x00000000|((ch&(1<<7))>>7)|((ch&(1<<6))>>5)|((ch&(
1<<5))>>1)|((ch&(1<<4))<<1));
}
if(y==1)

{GPIOPinWrite(GPIO_PORTB_BASE,      GPIO_PIN_0|
GPIO_PIN_1|      GPIO_PIN_4|      GPIO_PIN_5,
0x00000000|((ch&(1<<3))>>3)|((ch&(1<<2))>>1)|((ch&(
1<<1))<<3)|((ch&(1<<0))<<5));
}

GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
delay(2);
y--;
  }
}

void LCD_PrintStringOnLCD(char *ch)
{
int i=0; for(i=0; ch[i]!='\0';++i)
 {
int y=2;
while(y>0)
                                        {
GPIOPinWrite(GPIO_PORTB_BASE,
GPIO_PIN_6|GPIO_PIN_7,
GPIO_PIN_6|GPIO_PIN_7);
 if(y==2)
                                            {

GPIOPinWrite(GPIO_PORTB_BASE,   GPIO_PIN_0   |
GPIO_PIN_1      |      GPIO_PIN_4|GPIO_PIN_5,
0x00000000|((ch[i]&(1<<7))>>7)|((ch[i]&(1<<6))>>5)|((c
h[i]&(1<<5))>>1)|((ch[i]&(1<<4))<<1));
                                            }
if(y==1)
                                            {
GPIOPinWrite(GPIO_PORTB_BASE,   GPIO_PIN_0   |
GPIO_PIN_1      |      GPIO_PIN_4|GPIO_PIN_5,
0x00000000|((ch[i]&(1<<3))>>3)|((ch[i]&(1<<2))>>1)|((c
h[i]&(1<<1))<<3)|((ch[i]&(1<<0))<<5));
                                            }
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
delay(2);
y--;
  }
  }
}

void SetFourBitMode(char ch)
{
GPIOPinWrite(GPIO_PORTB_BASE,
GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_6);
GPIOPinWrite(GPIO_PORTB_BASE,      GPIO_PIN_0|
GPIO_PIN_1|      GPIO_PIN_4|      GPIO_PIN_5,
0x00000000|((ch&(1<<7))>>7)|((ch&(1<<6))>>5)|((ch&(
1<<5))>>1)|((ch&(1<<4))<<1));
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
delay(1);
}

void ConfigureLCD()
{
SetFourBitMode(0x30); //Eight bit mode


SetFourBitMode(0x30);   //Second attempt to get into
eight bit mode
```

```
SetFourBitMode(0x20);  //Set LCD in 4 bit mode


LCD_Command(0x2);  //Return home


LCD_Command(CLEARSCREEN);  //Clear display


LCD_Command(0x6);  //Cursor move right


LCD_Command(0xC);    //Display control: Display on, cursor off and cursor blink off

}

void LCD_PrintIntOnLCD(unsigned int var)
{
char str[5];
short i=0;
int temporary=10;
while(1) //Temp should ultimately store 10^(No. of digits in var-1)
{
if(var/temporary==0)
break;
else
temporary*=10;
}
temporary/=10;

while(temporary>0)
{
str[i]=var/temporary+48;
var%=temporary;
temporary/=10; i++;
}
str[i]='\0';
LCD_PrintStringOnLCD(str);
}

void InitConsole(void)
{
   /* Make the UART pins be peripheral controlled. */
GPIOPinTypeUART(GPIO_PORTA_BASE,
GPIO_PIN_0 | GPIO_PIN_1);

//Initialize UART
UARTConfigSetExpClk(UART0_BASE,          8000000,
115200,             (UART_CONFIG_WLEN_8             |
UART_CONFIG_STOP_ONE                    |
UART_CONFIG_PAR_NONE));
UARTEnable(UART0_BASE);

// Initialize the UART for console I/O.
UARTStdioInit(0);

}

/*making the speaker play an alarm bell with an infinitesimally small delay*/

void speakerplay(void)
{
for(count1=0; count1 <= 1000; count1++)
{
/* Make Pin Low */
GPIOPadConfigSet(speakerbase,              speakerpin,
GPIO_STRENGTH_8MA,
GPIO_PIN_TYPE_STD_WPU);
GPIOPinWrite(speakerbase, speakerpin, speakerpin);


SysCtlDelay(SysCtlClockGet()/5000);

/* Make Pin High */

GPIOPinWrite(speakerbase, speakerpin, 0);

/* Delay for a while */

SysCtlDelay(SysCtlClockGet()/5000);

}

SysCtlDelay(SysCtlClockGet()/50);
GPIOPinWrite(speakerbase, speakerpin, 0);

SysCtlDelay(SysCtlClockGet()/50);

}

/*trigger this alert condition if alcohol is detected. Driver cannot revert back to a previous state. System
shuts down. Message is Sent and alarm will constantly ring.
 */

void alcoholdetected(void)
{
     /*Send message using GPS*/

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("Alcohol");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("DetectedSTOP");
UARTprintf("Alcohol Detected. \n\nSTALL SYSTEM \n\n");
SysCtlDelay(SysCtlClockGet()/3);

while(1)
     {


GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

GPIOPinWrite(relaycontrolbase,          relaycontrolpin,
relaycontrolpin); //To signal help

for(i=0; i<=100; i++)
        {
        speakerplay();
        }
GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_8MA,
GPIO_PIN_TYPE_STD_WPU);
GPIOPinWrite(relaycontrolbase, relaycontrolpin, 0);

delay(2000);
```

```
    }
}
//ISR for Button Pressing

void Pin_Int(void)
{

GPIOPinIntClear(GPIO_PORTE_BASE, GPIO_PIN_0);
g_ulcount++;

}


void alarmState()
{
LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("ALARM");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("TRIGGERED");
UARTprintf("Alarm Triggered \n\n");

SysCtlDelay(SysCtlClockGet()/5);

//ENABLE THE BUTTON INTERRUPT ON PORT E

/* Set the clock for the GPIO Port C and E */

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

/* Set the type of the GPIO Pin */
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,
GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);
GPIOPinTypeGPIOInput(GPIO_PORTE_BASE,
GPIO_PIN_0);

/* GPIO Pins 5, 6, 7 on PORT C initialized to 1
* All LEDs off.
* */
GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7);

/*Register Interrupt to call Interrupt Handler*/
GPIOPortIntRegister(GPIO_PORTE_BASE, Pin_Int);

/*Clear interrupt register*/
GPIOPinIntClear(GPIO_PORTE_BASE, GPIO_PIN_0);

/*Configure GPIO pad with internal pull-up enabled*/
GPIOPadConfigSet(GPIO_PORTE_BASE,
GPIO_PIN_0,
GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WP
U);

/*Set interrupt triggering sequence*/
GPIOIntTypeSet(GPIO_PORTE_BASE, GPIO_PIN_0,
GPIO_FALLING_EDGE);

/*Enable interrupts on selected pin*/
GPIOPinIntEnable(GPIO_PORTE_BASE,
GPIO_PIN_0);

/*Enable interrupts on selected port*/
IntEnable(INT_GPIOE);

/* run a  loop 4 times to give driver time to stop alarm */

for(i=1; i<=4; i++)
        {

/*If button is not pressed then initiate alarm sequence */


LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;

LCD_PrintStringOnLCD("BUTTON NOT");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);

LCD_PrintStringOnLCD("PRESSED");

UARTprintf("Button Not Pressed \n\n");


/*Sound the alarm everytime the loop is entered */


GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU); //Sink Current is 2mA

GPIOPinWrite(relaycontrolbase,          relaycontrolpin,
relaycontrolpin);

for(count=0; count<=2; count++)
    {
    speakerplay();
    }


GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_8MA,
GPIO_PIN_TYPE_STD_WPU);

GPIOPinWrite(relaycontrolbase, relaycontrolpin, 0);


SysCtlDelay(SysCtlClockGet()/3);


/*Check if button was pressed*/
if(g_ulcount>=1)
    {
    g_ulcount=0;


GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0);

SysCtlDelay(SysCtlClockGet() / 12);

GPIOPinWrite(GPIO_PORTC_BASE,      GPIO_PIN_5,
GPIO_PIN_5);

SysCtlDelay(SysCtlClockGet() / 12);

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
```

```
LCD_PrintStringOnLCD("BUTTON");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);

LCD_PrintStringOnLCD("PRESSED");

UARTprintf("Button Pressed \n\n");

//DISABLE BUTTON INTERRUPT

/*Enable interrupts on selected pin*/

GPIOPinIntDisable(GPIO_PORTE_BASE,
GPIO_PIN_0);

/*Enable interrupts on selected port*/

IntDisable(INT_GPIOE);

break;
                                        }



        }

/* If the stipulated time elapses and the driver hasnt
pressed the button */
/* Enter and infinite loop and keep sounding the alarm */
/* SEND a GSM Text Message */



if(i >= 4)
        {

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
 LCD_PrintStringOnLCD("NO REACTION");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("SYSTEM STALL");
UARTprintf("No Reaction from Driver System Stalled
\n");

SysCtlDelay(SysCtlClockGet()/5);

/*Send message using GPS*/

//ENTER FUNCTION FOR GSM MESSAGE HERE


/*System will stall NOW */
while(1)
            {

GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);
GPIOPinWrite(relaycontrolbase,          relaycontrolpin,
relaycontrolpin);

for(count=0; count<=2; i++)
                {
                   speakerplay();
                }

GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_8MA,
GPIO_PIN_TYPE_STD_WPU);
GPIOPinWrite(relaycontrolbase, relaycontrolpin, 0);

delay(2000);

            }
        }

/* If the driver presses the button within the stipulated
time */
/* Escape the function */

else if (i < 4)
        {


LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;

LCD_PrintStringOnLCD("USER");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);

LCD_PrintStringOnLCD("ALERTED");

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("ALARM");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("ESCAPED");
UARTprintf("Alarm State Escaped \n");

/*Turn Display Off if alarm is escaped */

GPIOPadConfigSet(relaycontrolbase,    relaycontrolpin,
GPIO_STRENGTH_8MA,
GPIO_PIN_TYPE_STD_WPU);
GPIOPinWrite(relaycontrolbase, relaycontrolpin, 0);
SysCtlDelay(SysCtlClockGet()/2);

        }

}



int main(void)

{
/*Set the clocking to directly run from the crystal at
8MHz*/
SysCtlClockSet(SYSCTL_SYSDIV_1                  |
SYSCTL_USE_OSC    |    SYSCTL_OSC_MAIN    |
SYSCTL_XTAL_8MHZ);

/* Set the clock for the GPIO Ports */

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);


SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

/*Enable ADC Peripheral*/

SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

/* Set the type of the GPIO Pins as DIGITAL OUTPUTS
*/

GPIOPinTypeGPIOOutput(digital2base, digital2pin);
GPIOPinTypeGPIOOutput(digital3base, pin);
GPIOPinTypeGPIOOutput(digital4base, digital4pin);
GPIOPinTypeGPIOOutput(digital5base, digital5pin);
GPIOPinTypeGPIOOutput(digital6base, digital6pin);
GPIOPinTypeGPIOOutput(digital7base, digital7pin);
GPIOPinTypeGPIOOutput(digital8base, digital8pin);
GPIOPinTypeGPIOOutput(digital9base, digital9pin);
GPIOPinTypeGPIOOutput(digital10base, digital10pin);
GPIOPinTypeGPIOOutput(digital11base, digital11pin);
GPIOPinTypeGPIOOutput(digital12base, digital12pin);

        /*Initialize digital control pins*/

GPIOPadConfigSet(relaycontrolbase, relaycontrolpin,
GPIO_STRENGTH_8MA,
GPIO_PIN_TYPE_STD_WPU); // 8 mA sink current
required
GPIOPinWrite(relaycontrolbase, relaycontrolpin, 0);
//relay switch off inside main() function

GPIOPinWrite(speakerbase, speakerpin, 0); //speaker
off inside main() function

/* UART config */
InitConsole();

/* Enable is set low */
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);

/* Set up the period for the SysTick timer.  The SysTick
timer period will
be equal to 1ms.*/
SysTickPeriodSet(SysCtlClockGet()/1000);

/* Enable SysTick. */
SysTickEnable();

ConfigureLCD();

/*WELCOME SCREEN */


LCD_Command(GOTO_FIRSTROW_FIFTHCOLUMN);
LCD_PrintStringOnLCD("WELCOME");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("TO R.A.P.U");
UARTprintf("Welcome \n");
delay(3000);

/*Configure the ADC Sequence for the FIRST and
ONLY TIME*/

ADCSequenceConfigure(ADC0_BASE,                 0,
ADC_TRIGGER_PROCESSOR, 0);

/*Enable ADC sequence*/

ADCSequenceDisable(ADC0_BASE, 0);

ADCSequenceEnable(ADC0_BASE, 0);
 delay(10);

ADCIntClear(ADC0_BASE, 0);

 /*Clear ADC Interrupt*/

ADCIntClear(ADC0_BASE, 0);

/*Configure ADC Sample Collection from 4 Channels
READ ALL OTHER SENSORS*/

ADCSequenceStepConfigure(ADC0_BASE,     0,     0,
ADC_CTL_CH5);//button
ADCSequenceStepConfigure(ADC0_BASE,     0,     1,
ADC_CTL_CH0); //z axis
ADCSequenceStepConfigure(ADC0_BASE,     0,     2,
ADC_CTL_CH2); // x axis
ADCSequenceStepConfigure(ADC0_BASE,     0,     3,
ADC_CTL_CH4); // IR sensor
ADCSequenceStepConfigure(ADC0_BASE,     0,     4,
ADC_CTL_CH3 | ADC_CTL_IE | ADC_CTL_END);
//alcohol

/*

        According to above Sequence of steps
        temp[o] ---> button
        temp[1] ---> z axis
        temp[2] ---> x axis
        temp[3] ---> IR Sensor
        temp[4] ---> alcohol Sensor

*/

/* START calculating the MEAN VALUES for
acceleromter (AUTO CALIBRATION) */


LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("CALCULATING");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("REFERENCE");

unsigned long sumx = 0;
unsigned long sumz = 0;
unsigned int xref = 0;
unsigned int zref = 0;

SysCtlDelay(SysCtlClockGet()/5);

for(i=0; i<= 100; i++)
        {
```

```
SysCtlDelay(SysCtlClockGet()/50);

/*the sequence given below is used to READ the sensors
at any point from now on */
/* Just use the appropriate temp[i] values for analysis */

/* SEQUENCE STARTS */

/*Configure the ADC Sequence for the FIRST and ONLY TIME*/
ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);

/*Enable ADC sequence*/
ADCSequenceDisable(ADC0_BASE, 0);
ADCSequenceEnable(ADC0_BASE, 0);
delay(10);
ADCIntClear(ADC0_BASE, 0);

/*Clear ADC Interrupt*/
ADCIntClear(ADC0_BASE, 0);

/*Configure ADC Sample Collection from 4 Channels READ ALL OTHER SENSORS*/

ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH5);//button
ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH0); //z axis
ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH2); // x axis
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH4); // IR sensor
ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH3 | ADC_CTL_IE | ADC_CTL_END); //alcohol


ADCProcessorTrigger(ADC0_BASE, 0);
while(!ADCIntStatus(ADC0_BASE, 0, false))
        {
        }
ADCSequenceDataGet(ADC0_BASE, 0, temp);

/* SEQUENCE ENDS */

sumx = sumx + temp[2];
sumz = sumz + temp[1];
SysCtlDelay(SysCtlClockGet()/50);

    }

xref = sumx/101;
zref = sumz/101;

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("Xref = ");
LCD_PrintIntOnLCD(xref);
UARTprintf("Xref = %04d\n", xref);
delay(3000);

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("Yref = ");
LCD_PrintIntOnLCD(zref);
UARTprintf("Yref = %04d\n", zref);
delay(3000);

LCD_Command(CLEARSCREEN);

/*end of auto calibration */
while(1)
    {
/*Reinitiate blinkcounter for eyes CLOSED*/
blinkcounter = 0;

/*Reading just IR sensor values in a loop */

for(i=0; i<= 15; i++)
    {
/*Configure the ADC Sequence for the FIRST and ONLY TIME*/
ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);

/*Enable ADC sequence*/
ADCSequenceDisable(ADC0_BASE, 0);
ADCSequenceEnable(ADC0_BASE, 0);
delay(10);
ADCIntClear(ADC0_BASE, 0);

/*Clear ADC Interrupt*/
ADCIntClear(ADC0_BASE, 0);

/*Configure ADC Sample Collection from 4 Channels READ ALL OTHER SENSORS*/

ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH5);//button
ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH0); //z axis
ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH2); // x axis
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH4); // IR sensor
ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH3 | ADC_CTL_IE | ADC_CTL_END); //alcohol


ADCProcessorTrigger(ADC0_BASE, 0);

while(!ADCIntStatus(ADC0_BASE, 0, false))
        {
        }

ADCSequenceDataGet(ADC0_BASE, 0, temp);

if(temp[3] < 1000)
        {
        blinkcounter++;
        }

UARTprintf("IR value %04d\n", temp[3]);
LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("\ReadEyeState");
LCD_PrintIntOnLCD(temp[3]);
delay(1);

    }
```

```
/*Configure the ADC Sequence for the FIRST and ONLY TIME*/
ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 0);

/*Enable ADC sequence*/
ADCSequenceDisable(ADC0_BASE, 0);
ADCSequenceEnable(ADC0_BASE, 0);
delay(10);
ADCIntClear(ADC0_BASE, 0);

/*Clear ADC Interrupt*/
ADCIntClear(ADC0_BASE, 0);

/*Configure ADC Sample Collection from 4 Channels READ ALL OTHER SENSORS*/

ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH5);//button
ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH0); //z axis
ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH2); // x axis
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH4); // IR sensor
ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH3 | ADC_CTL_IE | ADC_CTL_END); //alcohol

/* Read ALL other sensors ONCE */

ADCProcessorTrigger(ADC0_BASE, 0);

while(!ADCIntStatus(ADC0_BASE, 0, false))
        {
        }

ADCSequenceDataGet(ADC0_BASE, 0, temp);

/*assign sensor values to sensor variables */
alcoholsense = temp[4];
xval = temp[2];
zval = temp[1];

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN);
LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN);
LCD_PrintStringOnLCD("AlcoholVal ");

LCD_Command(GOTO_SECONDROW_THIRDCOLUMN);
LCD_PrintIntOnLCD(alcoholsense);
UARTprintf("AlcoholVal = %04d \n\n", alcoholsense);
SysCtlDelay(SysCtlClockGet()/5);

/*calculate instantaneous deviation from reference values for head tilting */

deltax = abs(xref - xval);
deltaz = abs(zref - zval);

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN);
LCD_PrintStringOnLCD("Z DELTA ");

LCD_Command(GOTO_SECONDROW_THIRDCOLUMN);
LCD_PrintIntOnLCD(deltaz);
UARTprintf("Delta Z = %04d \n\n", deltaz);
SysCtlDelay(SysCtlClockGet()/7);


LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN);
LCD_PrintStringOnLCD("X DELTA ");

LCD_Command(GOTO_SECONDROW_THIRDCOLUMN);
LCD_PrintIntOnLCD(deltax);
UARTprintf("Delta X = %04d \n\n", deltax);
SysCtlDelay(SysCtlClockGet()/5);

//Check for Alcohol

if(alcoholsense > 1000)
        {
           alcoholdetected();
        }

/*Check for eyes being closed or blinking at a very fast rate or partially closed for long. */

if(blinkcounter >= 12)
        {

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN);
LCD_PrintStringOnLCD("Eyes");

LCD_Command(GOTO_SECONDROW_THIRDCOLUMN);
LCD_PrintStringOnLCD("Closed");
UARTprintf("Eye BLink Detected \n\n");
delay(2000);
/*Reinitiate blinkcounter for eyes CLOSED*/
blinkcounter = 0;
alarmState();

        }

/*Check for accelerometer alert state*/

/*Reinitiate accelerometer Counter */
        counter = 0;

/*If one of the values in head position differs from auto calibrated values, re-sensing is done */

//REREAD

if((deltax >= 70) || (deltaz >= 70))
        {

//REREAD THE HEAD TILT AND EYE BLINK SENSORS

blinkcounter = 0;

LCD_Command(CLEARSCREEN);
```

```
LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("TILT");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("DETECTED");
LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("RE READ");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("SENSORS");

for(count = 0; count <= 10; count++)
                {
/*Configure the ADC Sequence*/

ADCSequenceConfigure(ADC0_BASE,                 0,
ADC_TRIGGER_PROCESSOR, 0);

/*Enable ADC sequence*/
ADCSequenceDisable(ADC0_BASE, 0);
ADCSequenceEnable(ADC0_BASE, 0);
delay(10);
ADCIntClear(ADC0_BASE, 0);

/*Clear ADC Interrupt*/
ADCIntClear(ADC0_BASE, 0);

/*Configure ADC Sample Collection from 4 Channels
READ ALL OTHER SENSORS*/

ADCSequenceStepConfigure(ADC0_BASE,     0,     0,
ADC_CTL_CH5);//button
ADCSequenceStepConfigure(ADC0_BASE,     0,     1,
ADC_CTL_CH0); //z axis
ADCSequenceStepConfigure(ADC0_BASE,     0,     2,
ADC_CTL_CH2); // x axis
ADCSequenceStepConfigure(ADC0_BASE,     0,     3,
ADC_CTL_CH4); // IR sensor
ADCSequenceStepConfigure(ADC0_BASE,     0,     4,
ADC_CTL_CH3 | ADC_CTL_IE | ADC_CTL_END);
//alcohol



ADCProcessorTrigger(ADC0_BASE, 0);
while(!ADCIntStatus(ADC0_BASE, 0, false))
                {
                }

ADCSequenceDataGet(ADC0_BASE, 0, temp);

deltax = abs(xref - temp[2]);
deltaz = abs(zref - temp[1]);

if((deltax >= 70) || (deltaz >= 70))
                {
                counter++;
                SysCtlDelay(SysCtlClockGet()/5);
                }

if(temp[3] < 1000)
                {
                blinkcounter ++;
                }
              }
            }

/* Check For HEAD TILTED */

if (counter >= 7)
            {


LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("Head");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("Tilted");
UARTprintf("HEAD TILT DETECTED \n\n");
delay(800);
alarmState();

            }

/*Chech for Eyes Detected */

else if(blinkcounter >= 4)
            {

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
; LCD_PrintStringOnLCD("EYES");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);
LCD_PrintStringOnLCD("CLOSED");
UARTprintf("EYE BLINK DETECTED\n\n");
delay(1000);
alarmState();


            }

else

            {

LCD_Command(CLEARSCREEN);

LCD_Command(GOTO_FIRSTROW_THIRDCOLUMN)
;
LCD_PrintStringOnLCD("NO");

LCD_Command(GOTO_SECONDROW_THIRDCOLU
MN);

LCD_PrintStringOnLCD("ALARM");
UARTprintf("No alarm Detected \n\n");
delay(800);

            }

            }
}
```

APPENDIX C: Bill of Materials

	Component	Manufacturer	Cost per component	Quantity	Total cost of component	TI Supplied/ Purchased
1.	LM 723 Voltage Regulator	TI	Free	1	Free	TI
2.	LM3S608 MCU Unit on ARM Stellaris GURU	TI	Free	1	Free	TI
3.	LM358 op amp	TI	Free	1	Free	TI
4.	LM386 op amp	TI	Free	1	Free	TI
5.	CC4000 GPS Module	TI	Free	1	Free	TI
6.	MMA7760 Acceler-ometer	Freescale	Rs. 150	1	Rs. 150	Purchased
7.	IR leds	Generic	Rs. 20	4	Rs. 80	Purchased
8.	10K pot	Generic	Rs. 10	5	Rs. 50	Purchased
9.	SIMCOM GSM module	Robokits	Rs. 2000	1	Rs. 2000	Purchased
10.	DC adapters	Generic	Rs. 50	2	Rs. 50	Purchased
11.	Pinheads Jumpers LEDs Capacitors and other electronic components	Generic	Rs. 300	-	Rs. 300	Purchased
				Total cost of the project	Rs. 2580	