SPRINT 1

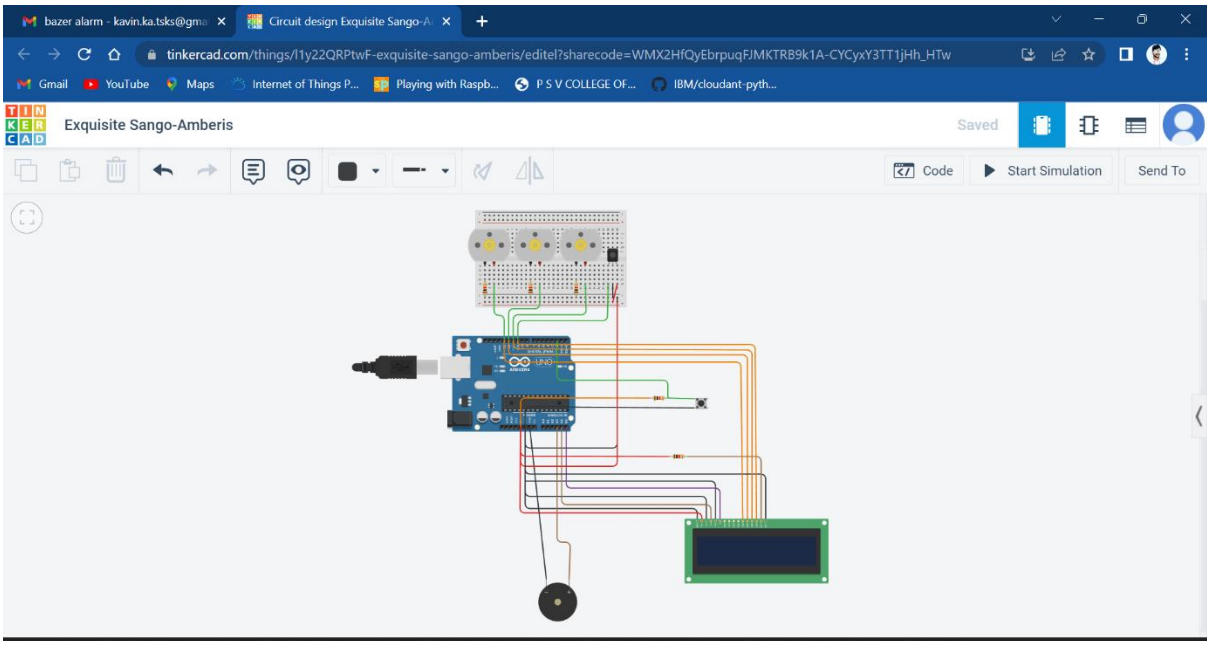
TEAM ID	**PNT2022TMID40888**
PROJECT TITLE	**Personal Assistance for seniors who are self-Reliant**

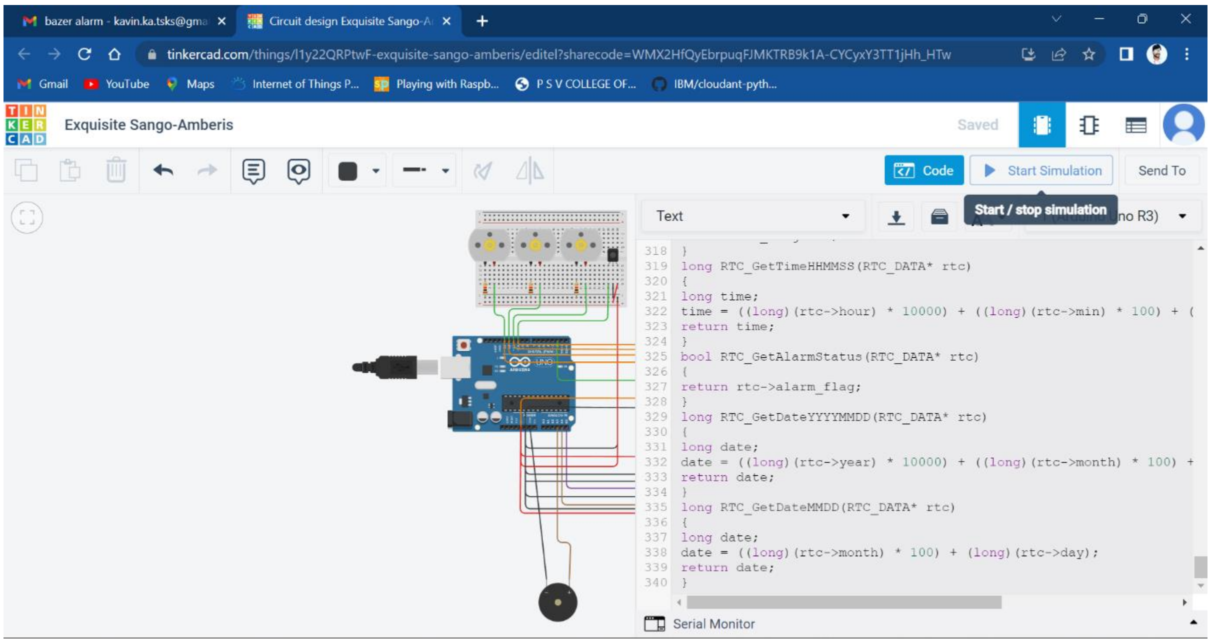
PROGRAM:

```
 #include <LiquidCrystal.h>
#include <stdio.h>
#define mSTATE_1 1
#define mSTATE_2 2
#define mSTATE_3 3
#define mSTATE_4 4
#define mSTATE_5 5
#define mSTATE_6 6
#define mSTATE_7 7
#define mSTATE_8 8
#define mSTATE_9 9
#define ALARM_SWITCH_PIN 2
#define BUZZER_PIN A3
typedef struct
{
bool ALARM_SWITCH;
bool RTC_ALARM;
unsigned long RTC_TIME;
unsigned long RTC_DATE;
char RTC_TIME_C[20];
char RTC_DATE_C[20];
}STATE_VAR;
typedef struct
{
bool alarm_enable;
bool alarm_flag;
int alarm_min;
int alarm_hour;
int tick;
int mls;
int sec;
int min;
int hour;
int month;
int day;
int year;
}RTC_DATA;
void get_input();
void fsm1(STATE_VAR *FSM_VAR);
void printLCDMesgFromStart(LiquidCrystal *lcd, const char* Message, unsigned int row, bool
lcdclear);
void Buzzer(bool ON, int Buzzer_PIN);
float readTemperature(int TempSensor_PIN);
void RTC(RTC_DATA* rtc);
void RTC_Init(RTC_DATA* rtc, int tick);
void RTC_SetTime(RTC_DATA* rtc, int hour, int min, int sec);
void RTC_SetDate(RTC_DATA* rtc, int year, int month, int day);
void RTC_SetAlarm(RTC_DATA* rtc, int hour, int min);
void RTC_EnableAlarm(RTC_DATA* rtc);
void RTC_DisableAlarm(RTC_DATA* rtc);
bool RTC_GetAlarmStatus(RTC_DATA* rtc);
long RTC_GetTimeHHMMSS(RTC_DATA* rtc);
long RTC_GetDateYYYYMMDD(RTC_DATA* rtc);
long RTC_GetDateMMDD(RTC_DATA* rtc);
STATE_VAR FSM1_VAR;
RTC_DATA rtc;
LiquidCrystal lcd(A4, A5, 13, 12, 11, 10);
void setup()
{
Serial.begin(9600);
pinMode(ALARM_SWITCH_PIN, INPUT);
lcd.begin(16, 2);
lcd.setCursor(0, 0);
RTC_Init(&rtc, 100);
RTC_SetTime(&rtc, 11, 00, 0);
RTC_SetDate(&rtc, 2020, 7, 7);
RTC_SetAlarm(&rtc, 11, 36);
RTC_EnableAlarm(&rtc);
}
void get_input()
{
String Date;
String Time;
FSM1_VAR.ALARM_SWITCH = digitalRead(ALARM_SWITCH_PIN);
FSM1_VAR.RTC_ALARM = RTC_GetAlarmStatus(&rtc);
FSM1_VAR.RTC_TIME = RTC_GetTimeHHMMSS(&rtc);
FSM1_VAR.RTC_DATE = RTC_GetDateYYYYMMDD(&rtc);
if(FSM1_VAR.ALARM_SWITCH == 0)
FSM1_VAR.ALARM_SWITCH = true;
else
FSM1_VAR.ALARM_SWITCH = false;
if(rtc.min < 10 && rtc.sec < 10)
Time = String(rtc.hour) + ':' + '0' + String(rtc.min) + ':' + '0' + String(rtc.sec);
else if (rtc.min >= 10 && rtc.sec < 10)
Time = String(rtc.hour) + ':' + String(rtc.min) + ':' + '0' + String(rtc.sec);
else if(rtc.min < 10 && rtc.sec >= 10)
Time = String(rtc.hour) + ':' + '0' + String(rtc.min) + ':' + String(rtc.sec);
else
Time = String(rtc.hour) + ':' + String(rtc.min) + ':' + String(rtc.sec);
Date = String(rtc.day) + '/' + String(rtc.month) + '/' + String(rtc.year);
strcpy(FSM1_VAR.RTC_TIME_C, Time.c_str());
strcpy(FSM1_VAR.RTC_DATE_C, Date.c_str());

}
void loop()
{
RTC(&rtc);
get_input();
fsm1(&FSM1_VAR);
delay(100);
}
void fsm1(STATE_VAR *FSM_VAR)
{
static int MACHINE_STATE;
static bool BUZZER;
bool ALARM_SWITCH = FSM_VAR->ALARM_SWITCH;
bool RTC_ALARM = FSM_VAR->RTC_ALARM;
long RTC_TIME = (FSM_VAR->RTC_TIME / 100);
long RTC_DATE = FSM_VAR->RTC_DATE;
Serial.print("DATE (YYYYMMDD): ");
Serial.print(RTC_DATE);
Serial.print(" TIME (HHMM): ");
Serial.println(RTC_TIME);
switch(MACHINE_STATE)
{
case mSTATE_1:
Serial.println("mSTATE_1");
if(!RTC_ALARM){
printLCDMesgFromStart(&lcd, FSM_VAR->RTC_DATE_C, 1, false);
printLCDMesgFromStart(&lcd, FSM_VAR->RTC_TIME_C, 2, false);
MACHINE_STATE = mSTATE_1 ;
}
else if(RTC_ALARM){
printLCDMesgFromStart(&lcd, "WARNING", 1, true);
printLCDMesgFromStart(&lcd, "ALARM!!", 2, false); BUZZER = true; MACHINE_STATE =
mSTATE_2;
}
else
{}
break;
case mSTATE_2:
Serial. println("mSTATE_2");
if(!ALARM_SWITCH){
MACHINE_STATE = mSTATE_2;
}
else if(ALARM_SWITCH){
RTC_DisableAlarm(&rtc); BUZZER = false; printLCDMesgFromStart(&lcd, FSM_VAR-
>RTC_DATE_C, 1, true);
printLCDMesgFromStart(&lcd, FSM_VAR->RTC_TIME_C, 2, false); MACHINE_STATE =
mSTATE_1;
}
else
{}
break;
default:
BUZZER = false;
MACHINE_STATE = mSTATE_1;
}
Buzzer(BUZZER, BUZZER_PIN);
}
void printLCDMesgFromStart(LiquidCrystal *lcd, const char* Message, unsigned int row, bool
lcdclear)
{
if(lcdclear)
{
lcd->clear();
}
switch(row)
{
case 1:
lcd->setCursor(0, 0);
break;
case 2:
lcd->setCursor(0, 1);
break;
default:
lcd->setCursor(0, 0);
}
if(Message != NULL)
{
lcd->print(Message);
}
}
void Buzzer(bool ON, int Buzzer_PIN)
{
static bool ON_STATE;
if (ON_STATE == false && ON == true)
{
ON_STATE = true;
tone(Buzzer_PIN, 2000);
}
else if (ON == false)
{
ON_STATE = false;
noTone(Buzzer_PIN);
}
else
{}
}
float readTemperature(int TempSensor_PIN)
{
float Temperature;
Temperature = (float) analogRead(TempSensor_PIN);
Temperature = (Temperature * 5.0) / 1024.0;
Temperature = Temperature - 0.5;
Temperature = Temperature * 100;

return Temperature;
}
void RTC(RTC_DATA* rtc)
{
static bool ALARM;
rtc->mls = rtc->mls + rtc->tick;
if(rtc->mls == 1000)
{
rtc->mls = 0;
rtc->sec++;
}
if(rtc->sec >= 60)
{
rtc->sec = 0;
rtc->min++;
}
if(rtc->min >= 60)
{
rtc->sec = 0;
rtc->min = 0;
rtc->hour++;
}
if(rtc->min < 0)
{
rtc->sec = 0;
rtc->min = 59;
rtc->hour--;
}
if(rtc->hour >= 24)
{
rtc->sec = 0;
rtc->min = 0;
rtc->hour = 0;
rtc->day++;
}
if(rtc->month != 2 && (rtc->month % 2) == 1)
{
if(rtc->day == 32)
{
rtc->day = 1;
rtc->month++;
}
if(rtc->day < 1)
{
rtc->day = 30;
rtc->month--;
}
}
if(rtc->month != 2 && (rtc->month % 2) == 0)
{
if(rtc->day == 31)
{rtc->day = 1;
rtc->month++;
}
if(rtc->day < 1)
{
rtc->day = 31;
rtc->month--;
}
}
if(rtc->month == 2)
{
if(rtc->day == 29)
{
rtc->day = 1;
rtc->month++;
}
}
if(rtc->month == 3)
{
if(rtc->day < 1)
{
rtc->day = 28;
rtc->month--;
}
}
if(rtc->hour < 0)
{
rtc->hour = 23;
}
if(rtc->month > 12)
{
rtc->month = 1;
rtc->day = 1;
rtc->year++;
}
if(rtc->alarm_enable == 1 && rtc->alarm_flag == 0)
{
if(rtc->min == rtc->alarm_min && rtc->hour == rtc->alarm_hour)
{
rtc->alarm_flag = 1;
}
}
}
void RTC_Init(RTC_DATA* rtc, int tick)
{
rtc->tick = tick;
rtc->alarm_flag = 0;
rtc->alarm_enable = 0;
}
void RTC_SetTime(RTC_DATA* rtc, int hour, int min, int sec)
{
rtc->sec = sec;
```

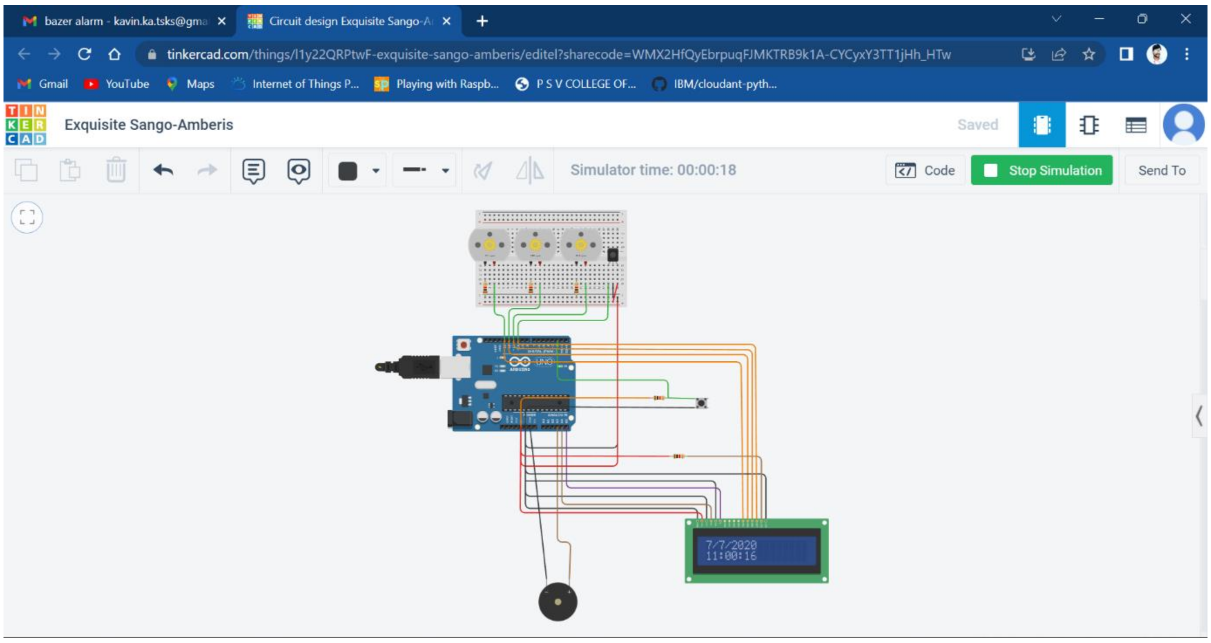
```
rtc->min = min;
rtc->hour = hour;
}
void RTC_SetDate(RTC_DATA* rtc, int year, int month, int day)
{
rtc->day = day;
rtc->month = month;
rtc->year = year;
}
void RTC_SetAlarm(RTC_DATA* rtc, int hour, int min)
{
rtc->alarm_min = min;
rtc->alarm_hour = hour;
}
void RTC_EnableAlarm(RTC_DATA* rtc)
{
rtc->alarm_enable = 1;
rtc->alarm_flag = 0;
}
void RTC_DisableAlarm(RTC_DATA* rtc)
{
rtc->alarm_enable = 0;
rtc->alarm_flag = 0;
}
long RTC_GetTimeHHMMSS(RTC_DATA* rtc)
{
long time;
time = ((long)(rtc->hour) * 10000) + ((long)(rtc->min) * 100) + (long)(rtc->sec);
return time;
}
bool RTC_GetAlarmStatus(RTC_DATA* rtc)
{
return rtc->alarm_flag;
}
long RTC_GetDateYYYYMMDD(RTC_DATA* rtc)
{
long date;
date = ((long)(rtc->year) * 10000) + ((long)(rtc->month) * 100) + (long)(rtc->day);
return date;
}
long RTC_GetDateMMDD(RTC_DATA* rtc)
{
long date;
date = ((long)(rtc->month) * 100) + (long)(rtc->day);
return date;
}
```

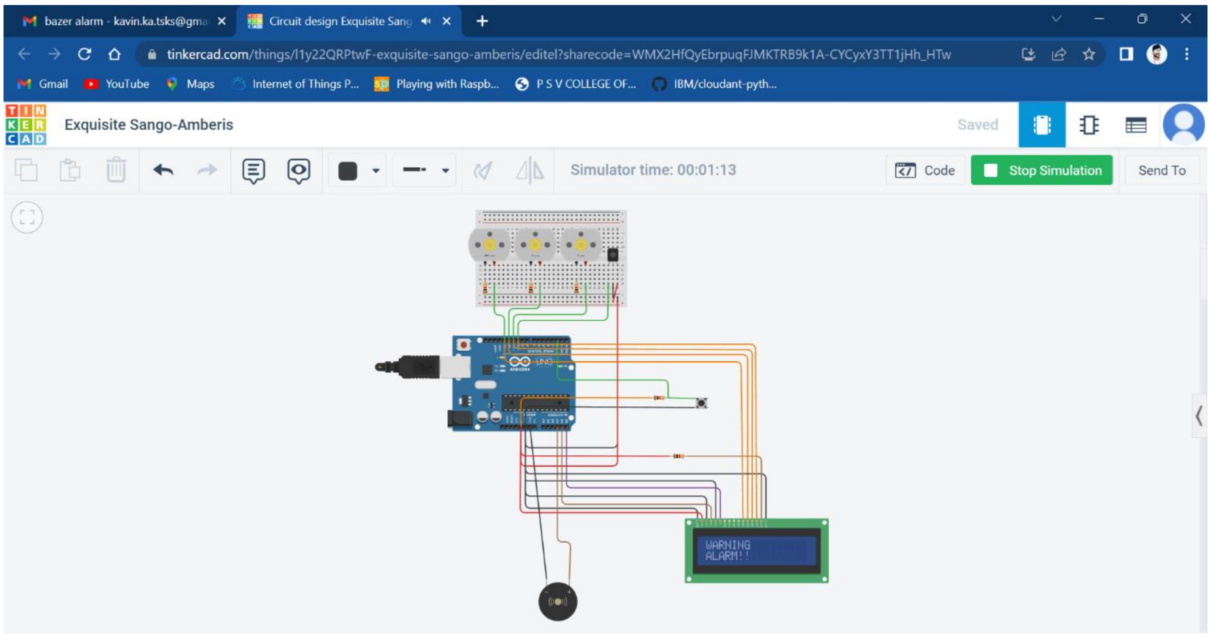
CIRCUIT DIAGRAM:


bazer alarm - kavin.ka.tsks@gma
Circuit design Exquisite Sango-A
tinkercad.com/things/l1y22QRPtwF-exquisite-sango-amberis/editel?sharecode=WMX2HfQyEbrpuqFJMKTRB9k1A-CYCyxY3TT1jHh_HTw
Gmail
YouTube
Maps
Internet of Things P...
Playing with Raspb...
P S V COLLEGE OF...
IBM/cloudant-pyth...
Exquisite Sango-Amberis
Saved
Code
Start Simulation
Send To



```
}
long RTC_GetTimeHHMMSS(RTC_DATA* rtc)
{
long time;
time = ((long)(rtc->hour) * 10000) + ((long)(rtc->min) * 100) + (
return time;
}
bool RTC_GetAlarmStatus(RTC_DATA* rtc)
{
return rtc->alarm_flag;
}
long RTC_GetDateYYYYMMDD(RTC_DATA* rtc)
{
long date;
date = ((long)(rtc->year) * 10000) + ((long)(rtc->month) * 100) +
return date;
}
long RTC_GetDateMMDD(RTC_DATA* rtc)
{
long date;
date = ((long)(rtc->month) * 100) + (long)(rtc->day);
return date;
}
```





CIRCUIT LINK:

https://www.tinkercad.com/things/l1y22QRPtwF-exquisite-sango-amberis/editel?sharecode=WMX2HfQyEbrpuqFJMKTRB9k1A-CYCyxY3TT1jHh_HTw

RESULT:

The circuit for the alarm has been designed and executed successfully.