|  |
| --- |
| **device1­\_arduino1.ino** |
| #include <Wire.h>  #include <SoftwareSerial.h>  #include <Adafruit\_GFX.h>  #include <Adafruit\_SSD1306.h>  #include <avr/sleep.h>  #include "MAX30102.h"  #include "Pulse.h"  #ifndef cbi  #define cbi(sfr, bit) (\_SFR\_BYTE(sfr) &= ~\_BV(bit))  #endif  #ifndef sbi  #define sbi(sfr, bit) (\_SFR\_BYTE(sfr) |= \_BV(bit))  #endif  #define SCREEN\_ADDRESS 0x3C  #define SPHYGMO\_ADDRESS 0x50  #define max\_measurement 3  #define id\_len 5  #define BEAT\_LED LED\_BUILTIN  #define select\_btn\_pin 2  #define change\_btn\_pin 3  #define tensimeter\_pin 9  char charArr[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0' };  //spo2\_table is approximated as  -45.060\*ratioAverage\* ratioAverage + 30.354 \*ratioAverage + 94.845 ;  const uint8\_t spo2\_table[184] PROGMEM = { 95, 95, 95, 96, 96, 96, 97, 97, 97, 97, 97, 98, 98, 98, 98, 98, 99, 99, 99, 99,                                            99, 99, 99, 99, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,                                            100, 100, 100, 100, 99, 99, 99, 99, 99, 99, 99, 99, 98, 98, 98, 98, 98, 98, 97, 97,                                            97, 97, 96, 96, 96, 96, 95, 95, 95, 94, 94, 94, 93, 93, 93, 92, 92, 92, 91, 91,                                            90, 90, 89, 89, 89, 88, 88, 87, 87, 86, 86, 85, 85, 84, 84, 83, 82, 82, 81, 81,                                            80, 80, 79, 78, 78, 77, 76, 76, 75, 74, 74, 73, 72, 72, 71, 70, 69, 69, 68, 67,                                            66, 66, 65, 64, 63, 62, 62, 61, 60, 59, 58, 57, 56, 56, 55, 54, 53, 52, 51, 50,                                            49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 31, 30, 29,                                            28, 27, 26, 25, 23, 22, 21, 20, 19, 17, 16, 15, 14, 12, 11, 10, 9, 7, 6, 5,                                            3, 2, 1 };  static const unsigned char PROGMEM logo2\_bmp[] = {    0x03, 0xC0, 0xF0, 0x06, 0x71, 0x8C, 0x0C, 0x1B,    0x06, 0x18, 0x0E, 0x02, 0x10, 0x0C, 0x03, 0x10,    0x04, 0x01, 0x10, 0x04, 0x01, 0x10, 0x40, 0x01,    0x10, 0x40, 0x01, 0x10, 0xC0, 0x03, 0x08, 0x88,    0x02, 0x08, 0xB8, 0x04, 0xFF, 0x37, 0x08, 0x01,    0x30, 0x18, 0x01, 0x90, 0x30, 0x00, 0xC0, 0x60,    0x00, 0x60, 0xC0, 0x00, 0x31, 0x80, 0x00, 0x1B,    0x00, 0x00, 0x0E, 0x00, 0x00, 0x04, 0x00  };  static const unsigned char PROGMEM logo3\_bmp[] = {    0x01, 0xF0, 0x0F, 0x80, 0x06, 0x1C, 0x38, 0x60, 0x18, 0x06, 0x60, 0x18, 0x10, 0x01, 0x80, 0x08,    0x20, 0x01, 0x80, 0x04, 0x40, 0x00, 0x00, 0x02, 0x40, 0x00, 0x00, 0x02, 0xC0, 0x00, 0x08, 0x03,    0x80, 0x00, 0x08, 0x01, 0x80, 0x00, 0x18, 0x01, 0x80, 0x00, 0x1C, 0x01, 0x80, 0x00, 0x14, 0x00,    0x80, 0x00, 0x14, 0x00, 0x80, 0x00, 0x14, 0x00, 0x40, 0x10, 0x12, 0x00, 0x40, 0x10, 0x12, 0x00,    0x7E, 0x1F, 0x23, 0xFE, 0x03, 0x31, 0xA0, 0x04, 0x01, 0xA0, 0xA0, 0x0C, 0x00, 0xA0, 0xA0, 0x08,    0x00, 0x60, 0xE0, 0x10, 0x00, 0x20, 0x60, 0x20, 0x06, 0x00, 0x40, 0x60, 0x03, 0x00, 0x40, 0xC0,    0x01, 0x80, 0x01, 0x80, 0x00, 0xC0, 0x03, 0x00, 0x00, 0x60, 0x06, 0x00, 0x00, 0x30, 0x0C, 0x00,    0x00, 0x08, 0x10, 0x00, 0x00, 0x06, 0x60, 0x00, 0x00, 0x03, 0xC0, 0x00, 0x00, 0x01, 0x80, 0x00  };  // data to receive  struct data\_1 {    int sys;    int dias;    int bpm;  };  union packet\_1 {    data\_1 value;    byte byteArray[sizeof(data\_1)];  };  packet\_1 sphygmo;  // data to send  struct data\_2 {    int spo2;    char id[id\_len];    char cmd;  };  union packet\_2 {    data\_2 value;    byte byteArray[sizeof(data\_2)];  };  packet\_2 command;  int sys\_avg;  int dia\_avg;  int pul\_avg;  int risk;  int beatAvg = 0;  int SPO2 = 0, SPO2f = 0;  int page;  int menu = 1;  bool led\_on = false;         // true jika led sedang menyala, false jika led mati  bool tensimeter\_on = false;  // true jika tensimeter sedang menyala, false jika tensimeter mati  bool repeat\_flag = false;    // true untuk melakukan pengulangan, false setelah melakukan pengulangan (sudah selesai)  bool on\_repeat = false;      // true jika sudah pernah melakukan pengulangan, false jika masih dalam pengukuran pertama  int repeat\_countdown = 300;  // countdown 5 menit untuk repeat  long now = 0;  long lastTime = 0, lastBeat = 0;  uint8\_t sleep\_counter = 0;  int measurement\_counter = 0;       // sudah berapa kali pengukuran  int measurement\_countdown = 60;    // countdown sebelum pengukuran berikutnya  long start\_measurement\_count = 0;  // variabel waktu millis  long last\_measurement\_count = 0;   // variabel waktu millis  bool msg\_sent = false;  Adafruit\_SSD1306 display(128, 64, &Wire, -1);  SoftwareSerial \_arduino2\_(10, 11);  MAX30102 sensor;  Pulse pulseIR;  Pulse pulseRed;  MAFilter bpm;    void setup() {    Serial.begin(9600);    \_arduino2\_.begin(9600);    pinMode(change\_btn\_pin, INPUT);    pinMode(select\_btn\_pin, INPUT);    pinMode(BEAT\_LED, OUTPUT);    pinMode(tensimeter\_pin, OUTPUT);    digitalWrite(tensimeter\_pin, HIGH);  // tensimeter off -> low trigger    /\* begin SSD1306 \*/    if (!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {      Serial.println(F("SSD1306 allocation failed"));      while (1)        ;    }    display.clearDisplay();    display.setTextSize(1);    display.setTextColor(WHITE);    display.setCursor(0, 28);    display.println(F("Welcome"));    display.display();    delay(2000);    /\* begin MAX30102 \*/    if (!sensor.begin())      page = 0;    else page = 2;    sensor.setup();    attachInterrupt(digitalPinToInterrupt(change\_btn\_pin), change\_isr, FALLING);    attachInterrupt(digitalPinToInterrupt(select\_btn\_pin), select\_isr, FALLING);    for (int i = 0; i < id\_len; i++) {      command.value.id[i] = charArr[random(35)];    }  }    void loop() {    now = millis();    /\* Read SPO2 and BPM continuously \*/    \_\_max30102\_\_();    /\* Waiting data from tensimeter \*/    if (\_arduino2\_.available()) {      \_arduino2\_.readBytes(sphygmo.byteArray, sizeof(sphygmo.byteArray));      sphygmo.value.sys = abs(sphygmo.value.sys);      sphygmo.value.dias = abs(sphygmo.value.dias);      sphygmo.value.bpm = abs(sphygmo.value.bpm);      sys\_avg += sphygmo.value.sys;      dia\_avg += sphygmo.value.dias;      pul\_avg += sphygmo.value.bpm;      tensimeter\_on = false;      measurement\_counter++;      digitalWrite(tensimeter\_pin, LOW);  // reset tensimeter setelah penggunaan      delay(300);      digitalWrite(tensimeter\_pin, HIGH);    }    /\* Measurement timer and counter \*/    if (((measurement\_counter > 0 || repeat\_flag) && measurement\_counter <= max\_measurement) && !tensimeter\_on) {      start\_measurement\_count = millis();                              // update millis      if (start\_measurement\_count - last\_measurement\_count >= 1000) {  // blok untuk mengurangi countdown        if (repeat\_flag) {                                             // kalau repeat flag aktif          repeat\_countdown--;                                          // kurangi countdown repeat 300 detik        } else {                                                       // kalau repeat flag tdk aktif          measurement\_countdown--;                                     // kurangi countdown measurement 60 detik        }        last\_measurement\_count = start\_measurement\_count;  // update last time      }      if (measurement\_countdown == 0 || repeat\_countdown == 0) {  // jika countdown measurement habis, nyalakan tensimeter        digitalWrite(tensimeter\_pin, LOW);                        // trigger tensimeter        delay(300);        digitalWrite(tensimeter\_pin, HIGH);        if (repeat\_flag) {      // jika countdown repeat habis,          repeat\_flag = false;  // pengulangan selesai          on\_repeat = true;     // sudah melakukan pengulangan        }        measurement\_countdown = 60;  // reset variabel        repeat\_countdown = 300;        tensimeter\_on = true;      }    }    /\* Display \*/    if (now - lastTime > 300) {      \_\_check\_condition\_\_();      \_\_ssd1306\_\_();    }  }    void change\_isr() {    if (page == 4) {      menu++;      if (menu > 3) menu = 1;    }  }  void select\_isr() {    if (page == 4) {      if (menu == 2) {        for (int i = 0; i < id\_len; i++) {          command.value.id[i] = charArr[random(35)];        }        // Serial.println(command.value.id);      } else {        if (menu == 1) {                                                     // jika menu send terpilih          if (!msg\_sent) {                                                   // jika belum pernah kirim pesan            command.value.spo2 = SPO2;                                       //            \_arduino2\_.write(command.byteArray, sizeof(command.byteArray));  // maka kirim pesan          } else {                                                           // jika sudah pernah kirim pesan,            msg\_sent = false;                                                // maka matikan flag          }          // Serial.print(command.value.cmd); Serial.print("\t");          // Serial.print(command.value.bpm); Serial.print("\t");          // Serial.print(command.value.spo2); Serial.print("\t");          // Serial.println("done");        }        if (on\_repeat) on\_repeat = false;        risk = 0;        page = 2;        menu = 1;      }    } else {      page = 4;    }  }  void go\_sleep() {    display.clearDisplay();    display.ssd1306\_command(SSD1306\_DISPLAYOFF);    delay(10);    sensor.off();    delay(10);    cbi(ADCSRA, ADEN);  // disable adc    delay(10);    pinMode(0, INPUT);    pinMode(2, INPUT);    set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN);    sleep\_mode();  // sleep until act\_button press    // cause reset    setup();  }  void \_\_max30102\_\_() {    sensor.check();    if (!sensor.available()) return;    uint32\_t irValue = sensor.getIR();    uint32\_t redValue = sensor.getRed();    sensor.nextSample();    if (irValue < 50000) {      page = (page == 4 ? 4 : (sleep\_counter <= 100 ? 1 : 3));      delay(100);      ++sleep\_counter;      if (sleep\_counter > 150) {        go\_sleep();        sleep\_counter = 0;      }    } else {      page = (page == 4 ? 4 : 2);      sleep\_counter = 0;      int16\_t IR\_signal, Red\_signal;      bool beatRed, beatIR;      // if (!filter\_for\_graph) {      IR\_signal = pulseIR.dc\_filter(irValue);      Red\_signal = pulseRed.dc\_filter(redValue);      beatRed = pulseRed.isBeat(pulseRed.ma\_filter(Red\_signal));      beatIR = pulseIR.isBeat(pulseIR.ma\_filter(IR\_signal));      // check IR or Red for heartbeat      if (beatIR) {        long btpm = 60000 / (now - lastBeat);        if (btpm > 0 && btpm < 200) beatAvg = bpm.filter((int16\_t)btpm);        lastBeat = now;        digitalWrite(BEAT\_LED, HIGH);        //        if (page == 2) {          display.clearDisplay();          display.drawBitmap(0, 0, logo3\_bmp, 32, 32, WHITE);          display.setTextSize(2);          display.setCursor(42, 15);          display.print(sphygmo.value.bpm);          display.setCursor(90, 15);          display.print(SPO2);          display.setTextSize(1);          display.setCursor(42, 3);          display.print(F("bpm"));          display.setCursor(90, 3);          display.print(F("%SpO2"));          display.setCursor(5, 41);          display.print(F("Systole : "));          display.print(sphygmo.value.sys);          display.setCursor(5, 55);          display.print(F("Diastole: "));          display.print(sphygmo.value.dias);          display.setCursor(115, 41);          display.print(measurement\_counter);          display.setCursor(98, 55);          if (repeat\_flag) {            display.print(repeat\_countdown / 60);            display.print(":");            if (repeat\_countdown % 60 < 10) {              display.print(0);            }            display.print(repeat\_countdown % 60);          } else {            if (measurement\_countdown < 10) display.setCursor(115, 55);            else display.setCursor(108, 55);            display.print(measurement\_countdown);          }          display.display();        }        //        led\_on = true;        // compute SpO2 ratio        long numerator = (pulseRed.avgAC() \* pulseIR.avgDC()) / 256;        long denominator = (pulseRed.avgDC() \* pulseIR.avgAC()) / 256;        int RX100 = (denominator > 0) ? (numerator \* 100) / denominator : 999;        // using formula        SPO2f = (10400 - RX100 \* 17 + 50) / 100;        // from table        if ((RX100 >= 0) && (RX100 < 184))          SPO2 = pgm\_read\_byte\_near(&spo2\_table[RX100]);      }    }    if (led\_on && (now - lastBeat) > 25) {      digitalWrite(BEAT\_LED, LOW);      led\_on = false;    }  }  void \_\_ssd1306\_\_() {    display.clearDisplay();    switch (page) {      case 0:        display.setTextSize(1);        display.setCursor(0, 0);        display.println(F("Device not found!"));        display.display();        while (1)          ;        break;      case 1:        display.setTextSize(1);        display.setCursor(0, 28);        display.println(F("Place your finger!"));        break;      case 2:        display.drawBitmap(5, 5, logo2\_bmp, 24, 21, WHITE);        display.setTextSize(2);        display.setCursor(42, 15);        display.print(sphygmo.value.bpm);        display.setCursor(90, 15);        display.print(SPO2);        display.setTextSize(1);        display.setCursor(42, 3);        display.print(F("bpm"));        display.setCursor(90, 3);        display.print(F("%SpO2"));        display.setCursor(5, 41);        display.print(F("Systole : "));        display.setCursor(65, 41);        display.print(sphygmo.value.sys);        display.setCursor(5, 55);        display.print(F("Diastole: "));        display.setCursor(65, 55);        display.print(sphygmo.value.dias);        display.setCursor(115, 41);        display.print(measurement\_counter);        display.setCursor(98, 55);        if (repeat\_flag) {          display.print(repeat\_countdown / 60);          display.print(":");          if (repeat\_countdown % 60 < 10) {            display.print(0);          }          display.print(repeat\_countdown % 60);        } else {          if (measurement\_countdown < 10) display.setCursor(115, 55);          else display.setCursor(108, 55);          display.print(measurement\_countdown);        }        break;      case 3:        display.setTextSize(1);        display.setCursor(0, 28);        display.println(F("OFF IN"));        display.write(10 - (sleep\_counter - 50) / 10 + '0');        display.write(' s');        break;      case 4:  // condition interface        display.setTextSize(1);        display.setCursor(0, 0);        display.print(F("ID    : "));        display.println(command.value.id);        display.print(F("Status: "));        if (risk == 0) {          display.println(F("Normal [0]"));  // pesan pada LCD          command.value.cmd = 'n';           // normal          repeat\_flag = false;        } else {          if (risk == 1) {            display.println(F("Normal [1]"));  // pesan pada LCD            command.value.cmd = 'N';           // normal, dengan peringatan          } else if (risk == 2) {            display.println(F("Waspada [2]"));  // pesan pada LCD            command.value.cmd = 'W';            // warning          } else if (risk > 2) {            display.println("Bahaya [" + String(risk) + "]");  // pesan pada LCD            command.value.cmd = 'D';                           // danger          }          if (!on\_repeat) {                                                  // jika tidak dalam keadaan repeat (belum pernah melakukan repeat),            repeat\_flag = true;                                              // maka lakukan repeat          } else if (!msg\_sent) {                                            // jika sudah pernah melakukan repeat, dan belum pernah kirim pesan            command.value.spo2 = SPO2;                                       //            \_arduino2\_.write(command.byteArray, sizeof(command.byteArray));  // kirim pesan otomatis            msg\_sent = true;                                                 // flag pesan terkirim          }        }        display.setCursor(0, 32);        if (menu == 1) display.println(F("> Send"));        else display.println(F("  Send"));        if (menu == 2) display.println(F("> Change ID"));        else display.println(F("  Change ID"));        if (menu == 3) display.println(F("> Back"));        else display.println(F("  Back"));        break;    }    display.display();    lastTime = now;  }  void \_\_check\_condition\_\_() {    /\* Condition checking     \* risk = 0: normal     \* risk = 1: warn     \* risk = 2: danger     \*/    if (measurement\_counter == max\_measurement) {  // cek kondisi jika jumlah pengukuran sudah mencapai batas maksimal      sys\_avg /= max\_measurement;      dia\_avg /= max\_measurement;      pul\_avg /= max\_measurement;      if (SPO2 < 95) risk += 2;      if ((sys\_avg < 90) || (sys\_avg > 180)) risk += 2;      else if (((sys\_avg >= 90) && (sys\_avg <= 99)) || ((sys\_avg >= 140) && (sys\_avg <= 180))) risk += 1;      if ((dia\_avg < 50) || (dia\_avg > 100)) risk += 2;      else if ((dia\_avg >= 90) && (dia\_avg <= 100)) risk += 1;      if ((pul\_avg < 40) || (pul\_avg > 120)) risk += 2;      else if (((pul\_avg >= 40) && (pul\_avg <= 50)) || ((pul\_avg >= 100) && (pul\_avg <= 120))) risk += 1;      page = 4;      measurement\_counter = 0;      sys\_avg = 0;      dia\_avg = 0;      pul\_avg = 0;    }  } |

|  |
| --- |
| **device1\_arduino2.ino** |
| /\*   \* Merah (pin 1) -> 3.3V   \* Putih (pin 2) -> SCL   \* Kuning (pin 3) -> SDA   \* Hitam (pin 4) -> GND   \*/  #include <Wire.h>  #include <SoftwareSerial.h>  #define device1\_number "081328431180"  #define device2\_number "081328431160"  #define doctor\_number "082131929286"  #define id\_len 5  SoftwareSerial \_arduino1\_(10, 11);  SoftwareSerial SIM900A(8, 9);  // data to send  struct data\_1 {    int sys;    int dias;    int bpm;  };  union packet\_1 {    data\_1 value;    byte byteArray[sizeof(data\_1)];  };  packet\_1 sphygmo;  // data to receive  struct data\_2 {    int spo2;    char id[id\_len];    char cmd;  };  union packet\_2 {    data\_2 value;    byte byteArray[sizeof(data\_2)];  };  packet\_2 command;  bool bPrint = 0;  int count, countT;  char buff[30];  bool data\_available;  long lastTime;    void setup() {    Serial.begin(9600);    \_arduino1\_.begin(9600);    SIM900A.begin(2400);    pinMode(2, INPUT\_PULLUP);    delay(500);    SIM900A.println("AT");    updateSerial();    SIM900A.println("AT+CSQ");    updateSerial();    SIM900A.println("AT+CCID");    updateSerial();    SIM900A.println("AT+CREG?");    updateSerial();    Wire.begin(0x50);    Wire.onReceive(receiveEvent);  }    void loop() {    // Serial.println(digitalRead(2));    \_arduino1\_.listen();    while (\_arduino1\_.available()) {      \_arduino1\_.readBytes(command.byteArray, sizeof(command.byteArray));      Serial.print(command.value.cmd);      Serial.print("\t");      Serial.print(command.value.spo2);      Serial.print("\t");      Serial.println("done");      if (digitalRead(2) == 0) {        if (command.value.cmd = 'D') {          send\_sms(device2\_number, "Bahaya");          delay(5000);          send\_sms(doctor\_number, "Bahaya");          // call(doctor\_number);        } else if (command.value.cmd = 'W') {          send\_sms(device2\_number, "Waspada");          delay(5000);          send\_sms(doctor\_number, "Waspada");          // call(doctor\_number);        } else if (command.value.cmd = 'N') {          send\_sms(device2\_number, "Normal dengan peringatan");          delay(5000);          send\_sms(doctor\_number, "Normal dengan peringatan");          // call(doctor\_number);        } else if (command.value.cmd != 'n') {          send\_sms(device2\_number, "Normal");          delay(5000);          send\_sms(doctor\_number, "Normal");          // call(doctor\_number);        }      }      http\_post();  // kirim data ke website jika ada data masuk dari arduino 1    }    if (millis() - lastTime > 1000) {      if (data\_available) {        // sphygmo.value.dias = 0.8 \* sphygmo.value.dias + 8.4;        \_arduino1\_.listen();        \_arduino1\_.write(sphygmo.byteArray, sizeof(sphygmo.byteArray));      }      data\_available = 0;      lastTime = millis();    }  }    void send\_sms(String number, String message) {    SIM900A.listen();    SIM900A.println("AT");    updateSerial();    SIM900A.println("AT+CMGF=1");    updateSerial();    SIM900A.print("AT+CMGS=\"" + number + "\"\r");    updateSerial();    SIM900A.print("REPORT\n\nStatus: " + message + "\nSpO2 = " + String(command.value.spo2) + "\nSys = " + String(sphygmo.value.sys) + "\nDias = " + String(sphygmo.value.dias) + "\nRate = " + String(sphygmo.value.bpm));    updateSerial();    SIM900A.write(26);    delay(1000);  }  void call(String number) {    SIM900A.listen();    SIM900A.println("ATD" + number + ";");    updateSerial();    delay(20000);    SIM900A.println("ATH");    updateSerial();  }  void updateSerial() {    SIM900A.listen();    delay(500);    while (Serial.available()) {      SIM900A.write(Serial.read());    }    while (SIM900A.available()) {      Serial.write(SIM900A.read());    }  }  void receiveEvent(int howMany) {    while (0 < Wire.available()) {      char c = Wire.read();      if (countT < 4) {        if (c == 'A') {          countT = 1;        }        if (c == '9') {          countT++;        }        if (c == '1') {          countT++;        }        if (c == '0') {          countT++;        }      } else if (countT == 4) {        Serial.write(c);        if (count == 0) {          sphygmo.value.sys = c;        } else if (count == 1) {          sphygmo.value.dias = c;        } else {          sphygmo.value.bpm = c;        }        count++;        if (count == 3) {          // sprintf(buff, "sys:%d, dias:%d, bpm:%d", sphygmo.value.sys, sphygmo.value.dias, sphygmo.value.bpm);          sprintf(buff, "sys:%d, dias:%d", sphygmo.value.sys, sphygmo.value.dias);          Serial.println(buff);          countT = 0;          count = 0;          data\_available = 1;        }      }    }  }  void http\_post() {    String sendtoserver;    sendtoserver += "nama=";    sendtoserver += command.value.id;    sendtoserver += "&tanggallahir=";    sendtoserver += "0000-00-00";  // yyyy-mm-dd    sendtoserver += "&paritas=";    sendtoserver += "-";    sendtoserver += "&sistol=";    sendtoserver += sphygmo.value.sys;    sendtoserver += "&diastol=";    sendtoserver += sphygmo.value.dias;    sendtoserver += "&nadi=";    sendtoserver += sphygmo.value.bpm;    sendtoserver += "&saturasioksigen=";    sendtoserver += command.value.spo2;    SIM900A.println("AT");    delay(500);    SIM900A.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");  // Connection type GPRS    delay(2000);    updateSerial();    SIM900A.println("AT+SAPBR=3,1,\"APN\",\"internet\"");  // APN of the provider    delay(3000);    updateSerial();    SIM900A.println("AT+SAPBR=1,1");  // Open GPRS context    delay(3000);    updateSerial();    SIM900A.println("AT+SAPBR=2,1");  // Query the GPRS context    delay(3000);    updateSerial();    SIM900A.println("AT+HTTPINIT");  // Initialize HTTP service    delay(3000);    updateSerial();    SIM900A.println("AT+HTTPPARA=\"CID\",1");  // Set parameters for HTTP session    delay(3000);    updateSerial();    SIM900A.println("AT+HTTPPARA=\"URL\",\"http://vitalsign.sogydevelop.com/datakirim\"");  // Server address, PAKAI "http", TIDAK SUPPORT "https"    delay(5000);    updateSerial();    SIM900A.println("AT+HTTPPARA=\"CONTENT\",\"application/x-www-form-urlencoded\"");    delay(5000);    updateSerial();    SIM900A.println("AT+HTTPPARA=\"USERDATA\",\"Authorization: Bearer YWRtaW4xMjM0NToxMjM0NTY3OA==\"");  // Bearer token    delay(5000);    updateSerial();    SIM900A.println("AT+HTTPDATA=" + String(sendtoserver.length()) + ",100000");  // POST data of certain size with maximum latency time of 10seconds for inputting the data    Serial.println(sendtoserver);    delay(5000);    updateSerial();    SIM900A.println(sendtoserver);  // Data to be sent    delay(5000);    updateSerial();    SIM900A.println("AT+HTTPACTION=1");  // Start POST session    delay(5000);    updateSerial();    SIM900A.println("AT+HTTPREAD");    delay(3000);    updateSerial();    SIM900A.println("AT+HTTPTERM");  // Terminate HTTP service    delay(3000);    updateSerial();    SIM900A.println("AT+SAPBR=0,1");  // Close GPRS context    delay(3000);    updateSerial();    delay(2000);  } |

|  |
| --- |
| **device2.ino** |
| /\* DEVICE 2       |  Arduino  |  SIM900A  |     |-----------|-----------|     |  D3 (RX)  |   U-TX    |     |  D2 (TX)  |   U-RX    |    \*/  #define gsm\_rx 4  #define gsm\_tx 3  #define buz\_rst 2  #define buz\_pin 5  // #define buz\_pin LED\_BUILTIN  #include <SoftwareSerial.h>  String msg;  bool msg\_received = false;  SoftwareSerial SIM900A(gsm\_tx, gsm\_rx);  void setup() {    Serial.begin(9600);    SIM900A.begin(2400);    pinMode(buz\_pin, OUTPUT);    pinMode(buz\_rst, INPUT\_PULLUP);    digitalWrite(buz\_pin, LOW);    delay(500);    SIM900A.println("AT");    updateSerial();    SIM900A.println("AT+CSQ");    updateSerial();    SIM900A.println("AT+CCID");    updateSerial();    SIM900A.println("AT+CREG?");    updateSerial();    /\* RECEIVE SMS \*/    SIM900A.println("AT+CMGF=1");    updateSerial();    SIM900A.println("AT+CNMI=1,2,0,0,0");    updateSerial();    attachInterrupt(digitalPinToInterrupt(buz\_rst), buzzer\_reset, FALLING);  }  void loop() {    updateSerial();    while (msg\_received) {      buzzer();      if (!msg\_received) {        break;      }    }  }  void buzzer\_reset() {    msg\_received = false;    msg = "";  }  void buzzer() {    digitalWrite(buz\_pin, HIGH);    delay(1000);    digitalWrite(buz\_pin, LOW);    delay(1000);  }  void updateSerial() {    delay(500);    while (Serial.available()) {      SIM900A.write(Serial.read());    }    while (SIM900A.available()) {      // Serial.write(Serial.read());      msg = SIM900A.readString();      Serial.print(msg);      if (msg.indexOf("REPORT") >= 0) {        msg\_received = true;      }    }  } |

|  |
| --- |
| **MAX30102.cpp** |
| #include "Wire.h"  #include "MAX30102.h"  static const uint8\_t MAX\_30102\_ID = 0x15;  MAX30102::MAX30102() {    // Constructor  }  boolean MAX30102::begin(uint8\_t i2caddr) {    \_i2caddr = i2caddr;    if (readRegister8(REG\_PART\_ID) != MAX\_30102\_ID)  return false;    return true;  }  void MAX30102::setup() {     writeRegister8(REG\_MODE\_CONFIG,0x40); //reset     delay(500);     writeRegister8(REG\_FIFO\_WR\_PTR,0x00);//FIFO\_WR\_PTR[4:0]     writeRegister8(REG\_OVF\_COUNTER,0x00);//OVF\_COUNTER[4:0]     writeRegister8(REG\_FIFO\_RD\_PTR,0x00); //FIFO\_RD\_PTR[4:0]     writeRegister8(REG\_FIFO\_CONFIG,0x4f); //sample avg = 4, fifo rollover=false, fifo almost full = 17     writeRegister8(REG\_MODE\_CONFIG,0x03); //0x02 for Red only, 0x03 for SpO2 mode 0x07 multimode LED     writeRegister8(REG\_SPO2\_CONFIG,0x27); // SPO2\_ADC=4096nA, SPO2 sample rate(100Hz), pulseWidth (411uS)     writeRegister8(REG\_LED1\_PA,0x17); //Choose value for ~ 6mA for LED1 (IR)     writeRegister8(REG\_LED2\_PA,0x17); // Choose value for ~ 6mA for LED2 (Red)     writeRegister8(REG\_PILOT\_PA,0x1F); // Choose value for ~ 6mA for Pilot LED  }  //Tell caller how many samples are available  uint8\_t MAX30102::available(void) {    int8\_t numberOfSamples = sense.head - sense.tail;    if (numberOfSamples < 0) numberOfSamples += STORAGE\_SIZE;    return (numberOfSamples);  }  //Report the next Red value in the FIFO  uint32\_t MAX30102::getRed(void) {    return (sense.red[sense.tail]);  }  //Report the next IR value in the FIFO  uint32\_t MAX30102::getIR(void) {    return (sense.IR[sense.tail]);  }  //Advance the tail  void MAX30102::nextSample(void) {    if(available()) {      sense.tail++;      sense.tail %= STORAGE\_SIZE; //Wrap condition    }  }  // check sensor for new samples and upload if available  uint16\_t MAX30102::check(void) {    byte readPointer = readRegister8(REG\_FIFO\_RD\_PTR);    byte writePointer = readRegister8(REG\_FIFO\_WR\_PTR);    int numberOfSamples = 0;    if (readPointer != writePointer) {      //Calculate the number of readings we need to get from sensor      numberOfSamples = writePointer - readPointer;      if (numberOfSamples < 0) numberOfSamples += 32; //Wrap condition      int bytesLeftToRead = numberOfSamples \* 6; //3 bytes each for Red and IR      Wire.beginTransmission(\_i2caddr);      Wire.write(REG\_FIFO\_DATA);      Wire.endTransmission();      bytesLeftToRead = bytesLeftToRead<=32? bytesLeftToRead : 32;      Wire.requestFrom((uint8\_t)\_i2caddr, (uint8\_t)bytesLeftToRead);      while (bytesLeftToRead > 0) {          sense.head++; //Advance the head of the storage struct          sense.head %= STORAGE\_SIZE; //Wrap condition          sense.IR[sense.head] = readFIFOSample();          //Burst read three more bytes - IR          sense.red[sense.head] = readFIFOSample();          bytesLeftToRead -= 6;      }      Wire.endTransmission();    }    return (numberOfSamples);  }    //  // Low-level I2C Communication  //  uint8\_t MAX30102::readRegister8(uint8\_t reg) {      uint8\_t value;      Wire.beginTransmission(\_i2caddr);      Wire.write((uint8\_t)reg);      Wire.endTransmission();      Wire.requestFrom(\_i2caddr, (byte)1);      value = Wire.read();      Wire.endTransmission();      return value;  }  uint32\_t MAX30102::readFIFOSample() {      byte temp[4];      uint32\_t temp32;      temp[3] = 0;      temp[2] = Wire.read();      temp[1] = Wire.read();      temp[0] = Wire.read();      memcpy(&temp32, temp, 4);      return temp32 & 0x3FFFF;  }  void MAX30102::writeRegister8(uint8\_t reg, uint8\_t value) {    Wire.beginTransmission(\_i2caddr);    Wire.write(reg);    Wire.write(value);    Wire.endTransmission();  } |

|  |
| --- |
| **MAX30102.h** |
| #include <arduino.h>  #define MAX30105\_ADDRESS 0x57  //register addresses  #define REG\_INTR\_STATUS\_1 0x00  #define REG\_INTR\_STATUS\_2 0x01  #define REG\_INTR\_ENABLE\_1 0x02  #define REG\_INTR\_ENABLE\_2 0x03  #define REG\_FIFO\_WR\_PTR 0x04  #define REG\_OVF\_COUNTER 0x05  #define REG\_FIFO\_RD\_PTR 0x06  #define REG\_FIFO\_DATA 0x07  #define REG\_FIFO\_CONFIG 0x08  #define REG\_MODE\_CONFIG 0x09  #define REG\_SPO2\_CONFIG 0x0A  #define REG\_LED1\_PA 0x0C  #define REG\_LED2\_PA 0x0D  #define REG\_PILOT\_PA 0x10  #define REG\_MULTI\_LED\_CTRL1 0x11  #define REG\_MULTI\_LED\_CTRL2 0x12  #define REG\_TEMP\_INTR 0x1F  #define REG\_TEMP\_FRAC 0x20  #define REG\_TEMP\_CONFIG 0x21  #define REG\_PROX\_INT\_THRESH 0x30  #define REG\_REV\_ID 0xFE  #define REG\_PART\_ID 0xFF  #define STORAGE\_SIZE 3 // buffer size in samples  class MAX30102{   public:      MAX30102(void);        boolean begin(uint8\_t i2caddr = MAX30105\_ADDRESS);        // Setup the IC with      // powerLevel = 0x1F,  sampleAverage = 4,  Mode = Red and IR,      // sampleRate = 100,  pulseWidth = 411,  adcRange = 4096      void setup();      void off() {writeRegister8(REG\_MODE\_CONFIG,0x80);}        //FIFO Reading      uint16\_t check(void);     //Checks for new data and fills FIFO      uint8\_t  available(void); //returns number of samples  available (head - tail)      void     nextSample(void);//Advances the tail of the sense array      uint32\_t getRed(void);    //Returns the FIFO sample pointed to by tail      uint32\_t getIR(void);     //Returns the FIFO sample pointed to by tail        // Low-level I2C communication      uint8\_t  readRegister8(uint8\_t reg);      uint32\_t readFIFOSample(void);      void     writeRegister8(uint8\_t reg, uint8\_t value);     private:      uint8\_t \_i2caddr;      struct {        uint32\_t red[STORAGE\_SIZE];        uint32\_t IR[STORAGE\_SIZE];        byte head;        byte tail;      } sense; //Circular buffer of readings from the sensor  }; |

|  |
| --- |
| **Pulse.cpp** |
| #include "Pulse.h"  Pulse::Pulse(){      cycle\_max = 20;      cycle\_min = -20;      positive = false;      prev\_sig = 0;      amplitude\_avg\_total = 0;  }  //  Returns true if a beat is detected  bool Pulse::isBeat(int16\_t signal) {    bool beat = false;    //while positive slope record maximum    if (positive && (signal > prev\_sig)) cycle\_max = signal;    //while negative slope record minimum    if (!positive && (signal < prev\_sig)) cycle\_min = signal;    //  positive to negative i.e peak so declare beat    if (positive && (signal < prev\_sig)) {      int amplitude = cycle\_max - cycle\_min;      if (amplitude > 20 && amplitude < 3000) {        beat = true;        amplitude\_avg\_total += (amplitude - amplitude\_avg\_total/4);      }      cycle\_min = 0; positive = false;    }    //negative to positive i.e valley bottom    if (!positive && (signal > prev\_sig)) {       cycle\_max= 0; positive = true;    }    prev\_sig = signal; // save signal    return beat;  } |

|  |
| --- |
| **Pulse.h** |
| #include "Arduino.h"  #define NSAMPLE 24  //Exponential Moving Average DC removal filter alpha = 1/NSAMPLE  #define NSLOT   4   //Moving Average Filter over NSLOT values.  class MAFilter {  public:      MAFilter() {nextslot = 0;}        int16\_t filter(int16\_t value) {          buffer[nextslot] = value;          nextslot = (nextslot+1) % NSLOT;          int16\_t total = 0;          for(int i=0; i<NSLOT; ++i) total += buffer[i];          return total/NSLOT;      }    private:      int16\_t buffer[NSLOT];      uint8\_t nextslot;  };  class DCFilter {  public:      DCFilter(void) {sample\_avg\_total = 0;}      //remove dc from sample      int16\_t  filter(int32\_t sample) {          sample\_avg\_total += (sample - sample\_avg\_total/NSAMPLE);          return (int16\_t)(sample - sample\_avg\_total/NSAMPLE);      }      // return average dc      int32\_t avgDC() {return sample\_avg\_total/NSAMPLE;}    private:      int32\_t sample\_avg\_total;  };  class Pulse {  public:      Pulse(void);      //remove DC      int16\_t dc\_filter(int32\_t sample) { return dc.filter(sample);}      //low pass moving average filter      int16\_t ma\_filter(int16\_t sample) { return ma.filter(sample);}      //return true when beat detected      bool isBeat(int16\_t signal);      //return average DC      int32\_t avgDC() {return dc.avgDC();}      //return average AC      int16\_t avgAC() {return amplitude\_avg\_total/4;}    private:      DCFilter dc;      MAFilter ma;      int16\_t amplitude\_avg\_total;      int16\_t cycle\_max;      int16\_t cycle\_min;      bool positive;      int16\_t prev\_sig;  }; |