Hat Heart Arduino Arduino Tutorial

1. Preparation

- Environment setup: Refer to the Arduino IDE Getting Started Guide to install the IDE, and install the appropriate board manager and required libraries for your development board.
- Required libraries:
  - M5Unified
  - M5GFX
  - SparkFun_MAX3010x_Sensor
- Hardware used:
  - StickS3
  - Hat Heart

2. Notes

Pin Compatibility

Each host device has different pin assignments. Please refer to the Pin Compatibility Table in the product documentation and modify the example code according to your actual wiring.

3. Example

- This guide uses StickS3 as the controller with the Hat Heart module. The heart rate module communicates via I2C. Adjust the pin definitions in the code according to your wiring; when stacked, the I2C pins are `G0 (SCL)` and `G8 (SDA)`.

Notes

1. After placing your finger on the sensor, keep your finger stable and wait for several seconds to obtain accurate SpO2 and BPM (heart rate) values.
2. If using StickS3 as the main control device, due to hardware limitations, the heart rate module cannot be reset after pressing the reset button. The device needs to be powered off and restarted to reinitialize the heart rate module. Otherwise, the main control device will display a prompt indicating initialization failure.

cpp

```cpp
#include <M5Unified.h>
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"    // Heart rate detection algorithm
#include "spo2_algorithm.h" // SpO2 calculation algorithm

M5Canvas canvas = M5Canvas(&M5.Lcd);

MAX30105 Sensor;

#define Heart_SDA 8
#define Heart_SCL 0

// Length of sample buffer for SpO2 calculation
#define bufferLength   100

// Button A pin definition
const byte Button_A = 11;

// Buffers to store infrared (IR) and red light sensor data
uint32_t irBuffer[bufferLength];
uint32_t redBuffer[bufferLength];

// Button state and reset flag
int8_t ButtonState, flag_Reset;

// SpO2 and heart rate calculation results
int32_t spo2, heartRate, old_spo2;
int8_t validSPO2, validHeartRate;

// Heart rate averaging parameters
const byte RATE_SIZE = 5;     // Number of beats used for averaging
uint16_t rate_begin  = 0;     // Counter for initial averaging
uint16_t rates[RATE_SIZE];   // Circular buffer for BPM values
byte rateSpot = 0;            // Index for BPM buffer
float beatsPerMinute;         // Instantaneous BPM
int beatAvg;                  // Averaged BPM

// Failure counter (e.g., finger removed)
byte num_fail;

// Display waveform buffers (red & IR), width 320 samples
uint16_t line[2][320] = {0};

// Current write positions for waveform buffers
uint32_t red_pos = 0, ir_pos = 0;

// Max/min values for waveform scaling
uint16_t ir_max = 0, red_max = 0, ir_min = 0, red_min = 0;
uint16_t ir_last = 0, red_last = 0;
```

```
51
52  // Raw filtered values from last sample
53  uint16_t ir_last_raw = 0, red_last_raw = 0;
54
55  // Display-mapped waveform values
56  uint16_t ir_disdata, red_disdata;
57
58  // Low-pass filter coefficient (scaled by 256)
59  uint16_t Alpha = 0.3 * 256;
60
61  // Timing variables for beat detection
62  uint32_t t1, t2, last_beat, Program_freq;
63
64  /**
65   * @brief Interrupt callback for Button A
66   *        Sets reset flag when button is pressed
67   */
68  void callBack(void) {
69      ButtonState = digitalRead(Button_A); // Read button state
70      if (ButtonState == 0) flag_Reset = 1; // Set reset flag if button pressed
71      delay(10);                           // Simple debounce delay
72  }
73
74  void setup() {
75      M5.begin();
76      Serial.begin(115200);
77      pinMode(Button_A, INPUT);
78      Wire.begin(Heart_SDA, Heart_SCL);
79
80      M5.Lcd.setRotation(3);
81      M5.Lcd.setSwapBytes(false);
82      canvas.createSprite(240, 135);
83      canvas.setSwapBytes(true);
84      canvas.createSprite(240, 135);
85
86      // Initialize MAX30102/MAX30105 sensor
87      if (!Sensor.begin(Wire, I2C_SPEED_FAST)) {
88          M5.Lcd.print("Init Failed");
89          Serial.println(F("MAX30102 was not found. Please check wiring/power."));
90          while (1); // Halt program
91      }
92
93      // Prompt user to place finger on sensor
94      Serial.println("Place your index finger on the Sensor with steady pressure");
95
96      // Attach interrupt to Button A (falling edge)
97      attachInterrupt(Button_A, callBack, FALLING);
98
99      // Configure MAX30102 sensor with default settings
100     Sensor.setup();
101     Sensor.clearFIFO(); // Optional FIFO clear (commented out)
102 }
103
```

```
void loop() {
    uint16_t ir, red;

    // If reset flag is set, clear sensor FIFO
    if (flag_Reset) {
        Sensor.clearFIFO();
        delay(5);
        flag_Reset = 0;
    }

    // Main acquisition loop (runs until reset)
    while (flag_Reset == 0) {
        // Wait until sensor data is available
        while (Sensor.available() == false) {
            delay(10);
            Sensor.check();
        }

        // Continuous data reading loop
        while (1) {
            red = Sensor.getRed(); // Read red light value
            ir  = Sensor.getIR();  // Read infrared value

            // Check if finger is properly placed
            if ((ir > 1000) && (red > 1000)) {
                num_fail = 0;       // Reset failure counter
                t1 = millis();      // Timestamp before processing

                // Store samples into circular buffers
                redBuffer[(red_pos + 100) % 100] = red;
                irBuffer[(ir_pos + 100) % 100]   = ir;

                // Measure processing frequency
                t2 = millis();
                Program_freq++;

                // Heartbeat detection using IR signal
                if (checkForBeat(ir) == true) {
                    long delta =
                        millis() - last_beat - (t2 - t1) * (Program_freq - 1);
                    last_beat = millis();
                    Program_freq = 0;

                    // Calculate BPM
                    beatsPerMinute = 60 / (delta / 1000.0);

                    if (beatsPerMinute > 40 && beatsPerMinute < 180) {

                        if (rate_begin == 0) {
                            beatAvg = (int)beatsPerMinute;
                        }

                        rates[rateSpot++] = (uint16_t)beatsPerMinute;
```

```
                        rateSpot %= RATE_SIZE;

                        if (rate_begin < RATE_SIZE) rate_begin++;

                        int sum = 0;
                        for (byte i = 0; i < rate_begin; i++) {
                            sum += rates[i];
                        }
                        beatAvg = sum / rate_begin;
                    }
                }
            } else {
                num_fail++; // Increase failure count if finger not detected
            }

            // Apply low-pass filtering to waveform data
            line[0][(red_pos + 240) % 320] =
                (red_last_raw * (256 - Alpha) + red * Alpha) / 256;
            line[1][(ir_pos + 240) % 320] =
                (ir_last_raw * (256 - Alpha) + ir * Alpha) / 256;

            red_last_raw = line[0][(red_pos + 240) % 320];
            ir_last_raw  = line[1][(ir_pos + 240) % 320];

            // Advance waveform positions
            red_pos++;
            ir_pos++;

            // Exit loop if no more data or reset requested
            if ((Sensor.check() == false) || flag_Reset) break;
        }

        // Clear FIFO after processing batch
        Sensor.clearFIFO();

        // Find max and min values for waveform scaling
        for (int i = 0; i < 240; i++) {
            if (i == 0) {
                red_max = red_min = line[0][(red_pos + i) % 320];
                ir_max = ir_min = line[1][(ir_pos + i) % 320];
            } else {
                red_max = (line[0][(red_pos + i) % 320] > red_max)
                            ? line[0][(red_pos + i) % 320]
                            : red_max;
                red_min = (line[0][(red_pos + i) % 320] < red_min)
                            ? line[0][(red_pos + i) % 320]
                            : red_min;
                ir_max = (line[1][(ir_pos + i) % 320] > ir_max)
                            ? line[1][(ir_pos + i) % 320]
                            : ir_max;
                ir_min = (line[1][(ir_pos + i) % 320] < ir_min)
                            ? line[1][(ir_pos + i) % 320]
                            : ir_min;
```

```
            }
            if (flag_Reset) break;
        }

        // Clear display canvas
        canvas.fillRect(0, 0, 240, 135, BLACK);

        // Draw waveform lines
        for (int i = 0; i < 240; i++) {
            red_disdata =
                map(line[0][(red_pos + i) % 320], red_max, red_min, 0, 135);
            ir_disdata =
                map(line[1][(ir_pos + i) % 320], ir_max, ir_min, 0, 135);

            canvas.drawLine(i, red_last, i + 1, red_disdata, RED);
            canvas.drawLine(i, ir_last, i + 1, ir_disdata, BLUE);

            ir_last  = ir_disdata;
            red_last = red_disdata;

            if (flag_Reset) break;
        }

        // Save previous SpO2 value
        old_spo2 = spo2;

        // Calculate heart rate and SpO2 when enough samples are collected
        if (red_pos > 100)
            maxim_heart_rate_and_oxygen_saturation(
                irBuffer, bufferLength, redBuffer,
                &spo2, &validSPO2,
                &heartRate, &validHeartRate);

        // Keep old SpO2 if new value is invalid
        if (!validSPO2) spo2 = old_spo2;

        // Display text information
        canvas.setTextSize(1);

        canvas.setTextColor(RED);
        canvas.setCursor(5, 5);
        canvas.printf("red:%d,%d,%d", red_max, red_min, red_max - red_min);
        Serial.printf("red:%d,%d,%d, ", red_max, red_min, red_max - red_min);

        canvas.setTextColor(BLUE);
        canvas.setCursor(5, 15);
        canvas.printf("ir:%d,%d,%d", ir_max, ir_min, ir_max - ir_min);
        Serial.printf("ir:%d,%d,%d ", ir_max, ir_min, ir_max - ir_min);

        canvas.setCursor(5, 25);
        if (num_fail < 10) {
            canvas.setTextColor(GREEN);
            canvas.printf("spo2:%d,", spo2);
```

```
            canvas.setCursor(60, 25);
            canvas.printf(" BPM:%d", beatAvg);
            Serial.printf(" spo2:%d, BPM:%d\n", spo2, beatAvg);
        } else {
            canvas.setTextColor(RED);
            canvas.printf("No Finger!!");
            Serial.printf("No Finger!!\n");
        }

        // Push canvas to LCD
        canvas.pushSprite(0, 0);
    }
}
```

4. Heart Rate Detection

- After powering up, the screen displays the raw infrared and red light data from the MAX30102 sensor. Place your finger on the sensor area of the heart rate module and observe the changes in SpO2 and BPM (heart rate) values and waveforms on the screen. Adjust finger position and pressure to get stable readings.