Hello, Joe here!

This code is also available at https://github.com/jmBSU/maskTest/blob/main/maskTest/src/main.cpp

This document will only explain the workings of the Arduino Code used in the MASK.

This document is for commit 8e77998 on April 15.

The numbers on the left indicate the line number of the code.

Programming Language: C/C++

Contents

Includes/ Constants/ Initialization/ Declaration

```
#include <spo2_algorithm.h>
#include <RTCZero.h> //for rtc
#include <Arduino.h>
#include <Wire.h> //For I2C
#include <SD.h>
#include <SPI.h> //SD CARD
#include <MAX30105.h>
#include <Adafruit_MLX90614.h>
```

These are used to include the libraries we need for this project. (Similar to import in Java) It's code that others wrote, so we don't have to reinvent anything.

Libraries included:

- SPO2 Algorithm
  - PPG based algorithm – looks at peaks of different reflections
- RTCZero
  - Real Time Clock – timer that counts in minutes/ seconds and not clock cycles
- Arduino
  - All Arduino specific pins like mapping to digital/ analog pins
- Wire
  - For I2C communication protocol – Needed for communication with SPO2 and Temp Sensor
- SD
  - To read and write to an SD card
- SPI
  - For SPI communication protocol – Needed for SD card communication
- MAX30105
  - Functions to start the SPO2 sensor and retrieve data
- Adafruit_MLX90614
  - Functions to start the Temp sensor and retrieve data

```
#define bluetooth Serial1 //Define bluetooth to serial1
#define usb Serial //usb to serial
```

Internally, the two UART lines are bounded to Serial and Serial1. This just renames them to easier to remember names.

```
18    const int CHIP_SELECT = 3;
19    const int MAX_SD_RETRY = 5;
```

Here are some constants. CHIP_SELECT is the CS pin on the SD card that is wired to pin 3 on the board.

MAX_SD_RETRY is the amount of times the SD card reader polls for a SD card before giving up.

```
21    bool SD_FOUND = false;
22    File logFile;
23    Adafruit_MLX90614 tempSensor = Adafruit_MLX90614();
24    MAX30105 spSensor;
```

The boolean SD_FOUND is flipped if there is a SD card found.

The rest initializes a variable to be of that data type.

```
26    const int32_t bufferLength = 100; //data length
27    const int quarterSize = floor(bufferLength/4.0); //Double to int calc
28    const int threeQuarterSize = floor(quarterSize * 3.0);
29    const int wakeTime = 1; //In minutes
30    const int sleepTime = wakeTime * 2;
31    const int btconnectPin = 1;
32    const int btdisconnectPin = 2;
```

More constants.

- bufferLength – how big the data array for SPO2 values are
- quarterSize – quarter of bufferLength rounded down
- threeQuarterSize – three quarters of bufferLength rounded down
- wakeTime – Minutes for the Mask to stay awake
- sleepTime – Minutes of the Mask to sleep which is at twice the wakeTime
- btconnectPin – Indicates which pin the connect interrupt is attached
- btdisconnectPin – Indicates which pin the disconnect interrupt is attached

---

```
35    uint32_t irBuffer[bufferLength]; //infrared LED sensor data
36    uint32_t redBuffer[bufferLength];
```

IR LED and Red LED data arrays

```
38  int32_t spo2; //SPO2 value
39  int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
40  int32_t heartRate; //heart rate value
41  int8_t validHeartRate; //indicator to show if the heart rate calculation is valid
42
43  int32_t tempspo2 = 0; //SPO2 value - temp
44  int32_t tempheartRate = 0; //heart rate value - temp
45  int32_t tempTemperature = 0; //heart rate value - temp
```

Lines 38 – 41 variables needed for the SPO2 algorithm to work. The algorithm will write to those every time it is called. Lines 43 – 45 will hold similar data values apart from the ones mentioned before. Lines 43 – 45 hold the final valid data value.

```
47  char strToPrint[50]; //String to hold data to print
48  RTCZero rtc;
49  bool rtcSleep = false;
50  bool btCon = false;
51  bool firstSleep = true;
```

- strToPrint – char array to hold a string to print
- RTCZero rtc – rtc of type RTCZero from RTCZero.h

The rest are Booleans that get flipped later based on what it does. rtcSleep is true when the system is asleep and false otherwise. btCon is true when there is a Bluetooth device connected and false otherwise. firstSleep is only for the initial sleep and is set false afterwards.

Function Declaration

```
53  void PrintData(void);
54  void PrintDataBluetooth(void);
55  void PrintDataUSB(void);
56  void standbyone(void); //sleep function
57  void btCNISR(void);//Bt connect
58  void btDSISR(void);//Bt disconnect
59  void rtcSleepISR(void);//Sleep to wake transition
60  void rtcWakeISR(void); //Wake to sleep transition
61  static void configGCLK6(void);
```

Declares all the functions in this code. More details on the functions below.

```
175  // Prints data
176  void PrintData(void){
177    logFile.print(strToPrint);
178  }
179
180  void PrintDataBluetooth(void){
181    bluetooth.print(strToPrint);
182  }
183
184  void PrintDataUSB(void){
185    usb.print(strToPrint);
186  }
```

PrintData, PrintDataBluetooth, PrintDataUSB

Prints data to mentioned destinations.

```
188   void standbyone(void){ //Needs interrupt
189     //From rocketscream/Low-Power with some tweaks
190     spSensor.shutDown();
191     SysTick->CTRL &= ~SysTick_CTRL_TICKINT_Msk;
192     SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
193     __DSB();
194     __WFI();
195     __WFE();
196     // Enable systick interrupt
197     SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;
198   }
```

Takes the system into sleep/ standby

- Line 190 – Turns off SPO2 Sensor
- Line 191 – Turns off SysTick interrupt
- Line 192 – Turns on Deep Sleep
- Line 193 – Force completion of outstanding memory operations
- Line 194 – Turns on Wait For Interrupt
- Line 195 – Turns on Wait For Event
- Line 197 – Turns on SysTick interrupt

```
240   //Straight from
241   //https://github.com/arduino-libraries/ArduinoLowPower
242   //This code allows EIC to run off of a generic clock
243   //due to the fact that sleep will shut off all clocks except generics.
244   static void configGCLK6(){
245     // enable EIC clock
246     GCLK->CLKCTRL.bit.CLKEN = 0; //disable GCLK module
247     while (GCLK->STATUS.bit.SYNCBUSY);
248
249     GCLK->CLKCTRL.reg = (uint16_t) (GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK6 | GCLK_CLKCTRL_ID( GCM_EIC )) ;
250     while (GCLK->STATUS.bit.SYNCBUSY);
251
252     GCLK->GENCTRL.reg = (GCLK_GENCTRL_GENEN | GCLK_GENCTRL_SRC_OSCULP32K | GCLK_GENCTRL_ID(6)); //source for GCL
253     while (GCLK->STATUS.reg & GCLK_STATUS_SYNCBUSY);
254
255     GCLK->GENCTRL.bit.RUNSTDBY = 1; //GCLK6 run standby
256     while (GCLK->STATUS.reg & GCLK_STATUS_SYNCBUSY);
257   }
```

In short, this changes the clock for the EIC to GCLK6. Sets the source for GCLK6. Lets GCLK6 run in the background.

ISR – Interrupt Service Routines – Called when there is an interrupt

```
200  void btCNISR(void){//Kills timer when bluetooth is connected
201    spSensor.wakeUp();
202    btCon = true;
203    rtcSleep = false;
204    rtc.disableAlarm();
205    rtc.detachInterrupt();
206   }
```

Wakes up the SPO2 Sensor, changes two Booleans, and stops the timer.

```
208  void btDSISR(void){//changes state so Bluetooth is disconnected
209    btCon = false;
210    rtcWakeISR();
211  }
```

Flips btCon Boolean when Bluetooth is disconnected and calls rtcWakeISR()

```
213  void rtcSleepISR(void){//If sleep, now woke
214    //Being awake stops the sleep timer
215    //and starts the wake timer
216    spSensor.wakeUp();
217    rtcSleep = false;
218    rtc.disableAlarm();
219    rtc.detachInterrupt();
220    rtc.begin(true); //Resets time
221
222    rtc.setAlarmMinutes(wakeTime);
223    rtc.enableAlarm(rtc.MATCH_MMSS);
224    rtc.attachInterrupt(rtcWakeISR);
225
226  }
```

Poor naming convention, I know. This is for the timer transition from sleep to wake. It turns the SPO2 sensor back on and stops the timer. The timer gets reset and turns back on with an interrupt that is called at wakeTime.

```
228  void rtcWakeISR(void){//If woke, now sleep
229    rtcSleep = true;
230    rtc.disableAlarm();
231    rtc.detachInterrupt();
232    rtc.begin(true); //Resets time
233
234    rtc.setAlarmMinutes(sleepTime);
235    rtc.enableAlarm(rtc.MATCH_MMSS);
236    rtc.attachInterrupt(rtcSleepISR);
237
238  }
```

This is for the timer transition from wake to sleep. Like rtcSleepISR(), it kills the timer and resets it. It starts the timer again but with an interrupt that is called at sleepTime.

Setup Code

```
void setup() {

  //Saves roughly 4mA total
  PM->CPUSEL.reg |= PM_CPUSEL_CPUDIV_DIV2; //48MHz to 24MHz
  PM->APBCMASK.reg &= ~PM_APBCMASK_ADC; //Shuts off ADC

  bluetooth.begin(9600);
  usb.begin(9600);


  //Tries to start SD card
  for(int i = 0; i < MAX_SD_RETRY; i++){
    if(SD.begin(CHIP_SELECT)){
      SD_FOUND = true;
      break;
    }
  }

  //If there is an SD card, it will run. Otherwise this will crash the system
  if(SD_FOUND){
    logFile = SD.open(F("LOG.TXT"), FILE_WRITE);
    logFile.println(F("NEW START"));
    logFile.close();
  }

  //Begins sleep
  rtc.begin(true); //Resets clock
  rtc.setAlarmMinutes(sleepTime); //Sets alarm to sleepTime min
  rtc.enableAlarm(rtc.MATCH_MMSS);
  rtc.attachInterrupt(rtcSleepISR);
  rtcSleep = true;

  //Creates external interrupts for Bluetooth
  pinMode(btconnectPin, INPUT_PULLUP);
  attachInterrupt(btconnectPin, btCNISR, RISING );
  pinMode(btdisconnectPin, INPUT_PULLUP);
  attachInterrupt(btdisconnectPin, btDSISR, FALLING );

  //Latches EIC to a generic clock - makes it work in sleep
  configGCLK6();
  //Makes one of the interrupts from bluetooth to wake up the device
  EIC->WAKEUP.reg |= (1 << g_APinDescription[btconnectPin].ulExtInt);

  //Sets up sensors
  tempSensor.begin();
  spSensor.begin();
  spSensor.setup(60,4,2,100,411,4096);
  for (byte i = 0 ; i < bufferLength ; i++)
  {
    redBuffer[i] = spSensor.getRed();
    irBuffer[i] = spSensor.getIR();
    spSensor.nextSample(); //We're finished with this sample so move to next sample

  }
  //calculate heart rate and SpO2 after first 100 samples (first 4 seconds of samples)
  maxim_heart_rate_and_oxygen_saturation( irBuffer, bufferLength,redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

}
```

- Line 67 – Decreases the clock speed by half
- Line 68 – Shuts of ADC
- Line 70 and 71 – Begins UART connection to both USB and Bluetooth
- Line 75 – 80 – Finds SD card and retries up to MAX_SD_RETRY
    - If found, makes SD_FOUND true
- Line 83 – 87 – If there is an SD card, open LOG.TXT, write “NEW START”, and close the file
- Line 90 – 94 – Starts a reset clock and makes it ready to go to sleep
- Line 97 – 100 – Creates external interrupts for Bluetooth connect and disconnect
    - Done by using an internal pullup on the pins
    - For connects, interrupt will match for a rising signal
    - For disconnects, interrupt will match for a falling signal
- Line 103 – Moves EIC to GCLK6
- Line 105 – Makes Bluetooth connect pin to work as a wakeup interrupt
- Line 108 – 110 – Sets up both sensors
    - Line 110 – SPO2 set up for Power 60, sampleAVG 4, ledMode 2, sampleRate 100, sampleWidth 411, ADC range 4096
- Line 111 – 119 – Gets first set of samples and runs it through the algorithm

Loop Code

```
  //Checks if either it is scheduled or bluetooth start
  if(!rtcSleep || btCon){
      //dumping the first quarter of samples in the memory and shift the last 75 sets of samples to the top
      for (byte i = quarterSize; i < bufferLength; i++)
      {
        redBuffer[i - quarterSize] = redBuffer[i];
        irBuffer[i - quarterSize] = irBuffer[i];
      }
      //take some sets of samples before calculating the heart rate.
      for (byte i = threeQuarterSize; i < bufferLength; i++)
      {
        redBuffer[i] = spSensor.getRed();
        irBuffer[i] = spSensor.getIR();
        spSensor.nextSample(); //We're finished with this sample so move to next sample
      }
      //After gathering new samples recalculate HR and SP02
    maxim_heart_rate_and_oxygen_saturation( irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);

    if(validSPO2 && validHeartRate){ //Prints new data when valid data
        tempspo2 = spo2;
        tempheartRate = heartRate;
        tempTemperature = tempSensor.readObjectTempF();
        sprintf(strToPrint, "Temp: %d \n HR: %d \n SPO2: %d \n", tempTemperature, tempheartRate, tempspo2);

    }else{ //Prints old data with new temp
        tempTemperature = tempSensor.readObjectTempF();
        sprintf(strToPrint, "Temp: %d \n HR: %d \n SPO2: %d \n", tempTemperature, tempheartRate, tempspo2);

    }

    if(SD_FOUND){
      logFile = SD.open(F("LOG.TXT"), FILE_WRITE);
      PrintData();
      logFile.close();
    }

    if(btCon){ //Prints to bluetooth if bluetooth is connected
      PrintDataBluetooth();
    }

    if(rtcSleep && !btCon){//Sleeps if timer ends or bluetooth disconnects
      standbyone();
    }
  }
}
```

- Line 124 – 126 – Makes the system sleep inside the loop code during first boot up
- Line 130 – Checks if timer-based start or Bluetooth start
- Line 132 – 136 – Dumps first quarter of data and shifts the rest
- Line 138 – 144 – Takes new samples for the last quarter of data
- Line 145 – Calls the algorithm
- Line 147 – 157 – Writes new data to the char array strToPrint when the algorithm comes back with valid data
- Line 159 – 163 – Writes strToPrint to the SD card's LOG.TXT (if there is an SD card)
- Line 165 – 167 – Writes strToPrint to Bluetooth if Bluetooth is connected
- Line 169 – 171 – If either the timer ends or the Bluetooth disconnects, the system goes to sleep.