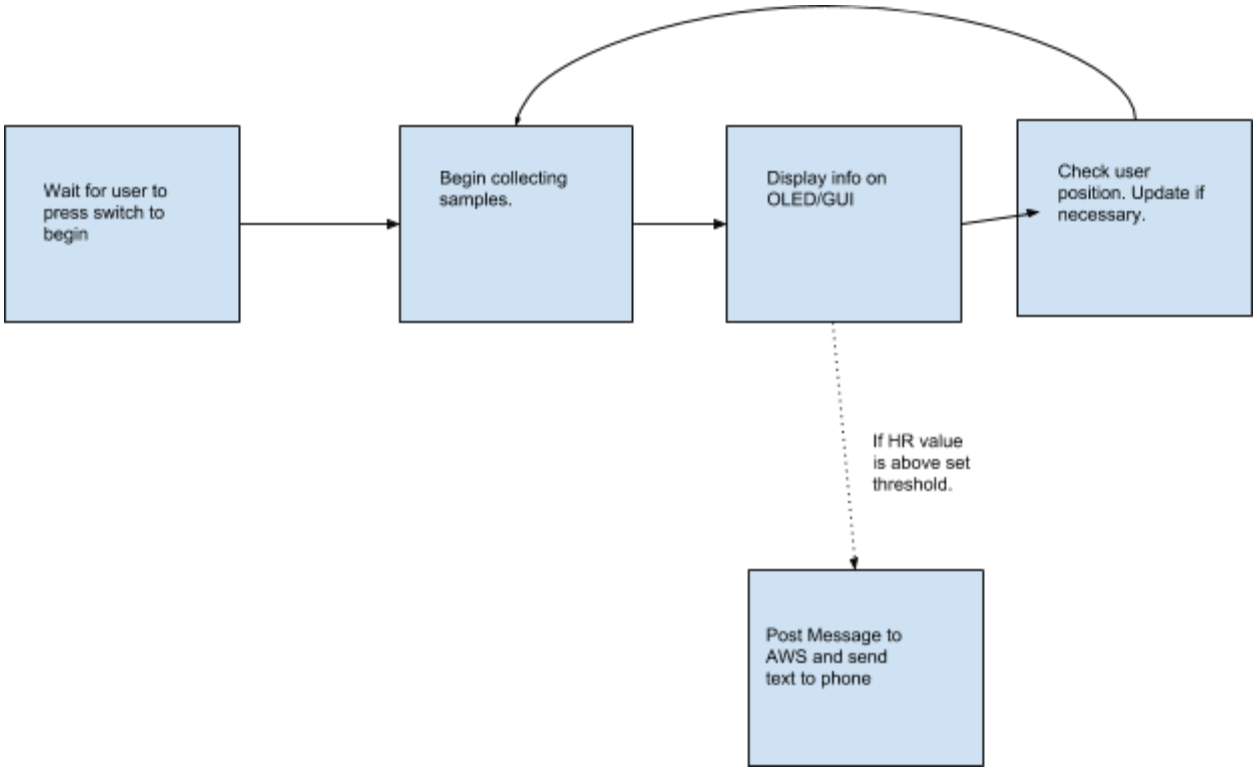
Minh Truong , Jose Torres-Vargas

Lab 6: Heart rate sensor with MAXREFDES117

Project Overview

For our Lab 6 project, we decided to build a heart sensor that calculates the user's heart rate whenever the finger is on the sensor. The data is then displayed on the OLED display. Apart from obtaining the heart rate we use the accelerometer on the launchpad to determine whether the patient is standing or sitting. Apart from just displaying the information on the OLED, the heart rate(HR) data is sent via UART to a computer so that the data can then be graphed on the computer via some python GUI(All processing is done on the launchpad). Finally, the board is connected to the internet as well in order to send the user a message whenever the HR exceeds a set threshold. These modules are explained in more detail below.



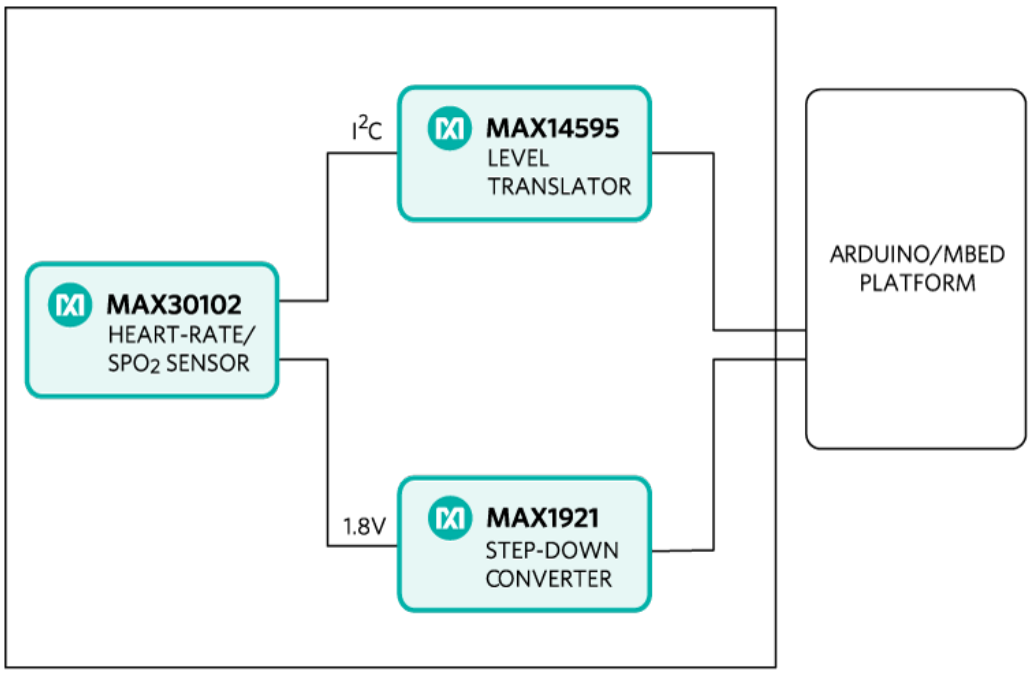
Modules

The system consists of the following modules, and their interaction are explained below:

- HR Sensor Module (connect with system via I2C)
- OLED Module (connect with system via SPI)
- Position Module (connect with system via I2C)
- Internet Module (connect with system via HTTP RestFUL / GPIO)
- GUI Module (connect with system via UART)

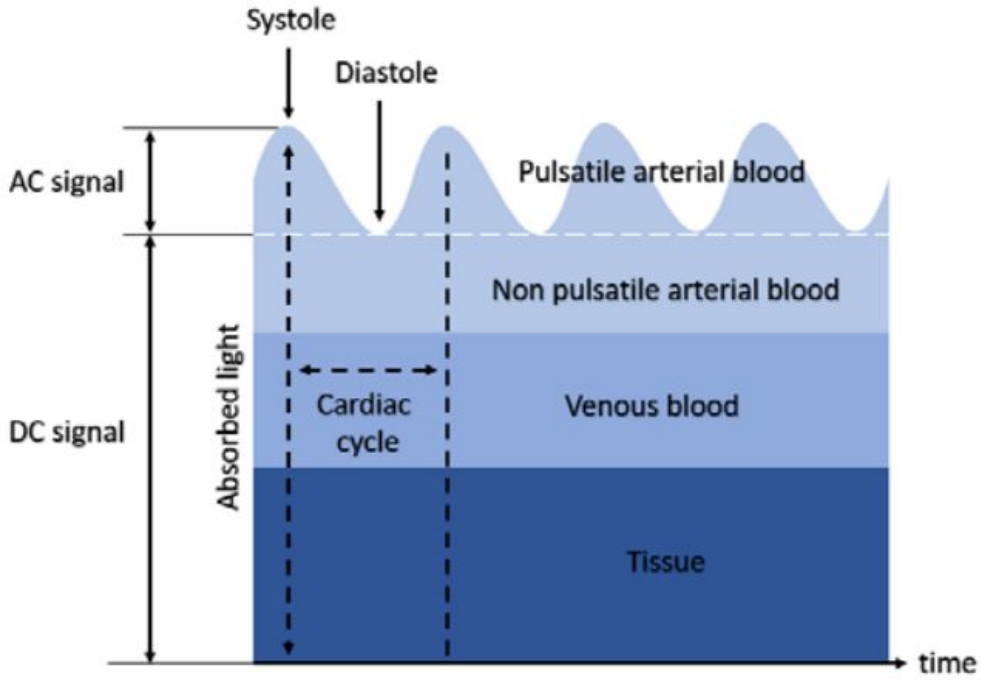
HR Sensor

The MAXREFDES117 is a compact package that contains 3 ICs illustrated below:



The MAX30102 is the main module of the sensor, it consists of 2 LED (red and IR) as well as a photosensor. All features of the module are devised with these 3 components. The other 2 ICs are used for power/communication management. They are passive modules. The sensor can be used with any embedded system. As discussed later, most of their I2C communication are developed for Arduino system, and we had to manually port the code to CCS for the CC3200 Launchpad.

The heart rate measurement is performed on the Launchpad after the sensor collect raw data by shining red/IR light onto fleshy surface like earlobe or fingertip. The raw data is in form of intensity level of both light source and it is typically around the order of 10e3 when there is no reflection and 10e6 when there is. For the purpose of measuring heart rate, we only use the red LED intensity value. The raw data has 2 componentes: DC and AC. The DC components are composed of reflected intensity of non-pulsating part of the body like venous blood and tissue. The AC components is attributed to pulsatile arterial blood, which has a direct connection to instantaneous heart rate. The following illustration show the theoretical AC and DC component over time:



From here, we can calculate the instantaneous heart rate by measuring the time between 2 peaks. The following pseudo code explains further:

```
findHeartRate():
      raw = collectData()
      DC = getDCComponent(raw)
      AC = raw - DC
      peak1, peak2 = calculatePeak(AC)
      return peak2 - peak1
```

The full implementation is available in algorithm.c . Calculating the DC component is tractable, we first sample a first few timestep to find the average of the light intensity, that average is situated in between the peak and the valley values of the AC component, by subtracting this average from raw, we can get the mean level at 0. To calculate the amount of time passes between each peak, we first calculate a threshold value. Because the raw data is very noisy, we need to devise a threshold value, which treat different data points to be the same level if their difference is within that threshold. This in effect smoothen out our signal, removing the high frequency component. This threshold is calculated by sample the signal for a very small amount of time. After that, we find the peaks by taking the derivative of the signal. These peaks will be situated at points where the slope is 0. In reality, we take the difference between a small time step to see if the signal goes up or down overall with time.

OLED Module

Once the HR value was obtained it was displayed on the OLED. In order to do so, the (int) value would first have to be converted into a string as the included Adafruit library only had a function to draw chars. In order to do so, the sprintf() function was used. Once the number has been converted into a string the drawChar() function is invoked to draw each individual character on the OLED screen. Before a new value is displayed the previous one is erased by drawing a small black rectangle over the old value.

```
void displayHeartInfo(int hr, int ox){
    int sb = 0;
    int xcur = 20, ycur = 40; //cursor values\\
    //buffer to hold converted number
    char* numBuf = malloc(30 * sizeof(char));
    SPIEnable(GSPI_BASE);
    fillRect(20, 40, 110, 17, BLACK);//erase old value
    sprintf(numBuf, "%d", hr);//convert num to string
    while(numBuf[sb] != '\0'){
        drawChar(xcur, ycur, numBuf[sb], WHITE, BLACK, 2);
        xcur += 12;//update x cursor
        sb++;
    }
    delay(5);
    SPIDisable(GSPI_BASE);
    free(numBuf);
}
```

Position Module

This module is based off and very similar to the concepts used in Lab 2. Using I2C we communicate with the onboard accelerometer. Assuming the user has the device, cc3200 Launchpad, attached to their arm we calculated the values of the X and Y position while the user was in different positions. In the end, we decided to simply use the Y values. We assume that when the user is standing their arm is hanging straight down and when they are sitting their arm is resting on a table or parallel on some flat surface. We then obtained values while the user was in either position. In our case when y>= 14 or y <= -14 we determine that the user is standing. The same is done for the seating position, but we obtained a different range through testing. The below function is in charge of determining user position.

```
void userPosition(){

    short int raw_x, raw_y;
    int conf= 0;
    enum state st; //
    enum state prevState;
    int done = 0;
    while(done == 0){
        //get gyro. value
        collect_raw(&raw_x, &raw_y); //function that gets values
from acc.

        if((raw_y >= 14) || (raw_y <= -14)){
            st = Standing;
        }
        else if((raw_y >= -3) && (raw_y <= 3)){
            st = Sitting;
        }
        else{
            ;
        }

        if(conf == 0){
            prevState = st;
            conf++;
        }
        else if(conf == 10){
            conf = 0;
            oledPrintState(prevState);
            done = 1;
        }
        else if(prevState == st){
            conf++;
        }
        else{
            conf = 0;
        }
    }
```

```
}
```

We included a confidence level value as well in order to ensure that the user is either standing or sitting.

```
void collect_raw(short int * raw_x, short int * raw_y) {
    unsigned char * x;
    unsigned char * y;
    short int zero = 0;
    short int holder;

    // read x coordinate
    x = readCord(0);
    holder = x[0] | zero;
    holder = holder << 8;
    holder /= 1000;
    *raw_x = holder;

    // read y coordinate
    y = readCord(1);
    holder = y[0] | zero;
    holder = holder << 8;
    holder /= 1000;
    *raw_y = holder;
}
This function simply prints the state to the OLED
void oledPrintState(enum state st){
    char* sit = "sitting";
    char* stand = "standing";
    int i = 0;
    int xc = 20;
    SPIEnable(GSPI_BASE);
    //Erasing previous values
    fillRect(20, 110, 110, 17, BLACK);
    //Message("Drawing State \r\n");
    if(st == Standing){
        for(i = 0; i < 8; i++){
            drawChar(xc,110, stand[i], WHITE, BLACK, 2);
            xc += 12;
        }
```

```
    }
    else{
        for(i = 0; i < 7; i++){
            drawChar(xc,110, sit[i], WHITE, BLACK, 2);
            xc += 12;
        }
    }
    SPIDisable(GSPI_BASE);
}
```

Internet Module

The internet module is practically Lab 5. The same certificates from Lab 5 are used since we're connecting to the same place. The only difference is that it's sending a warning message and the post is only triggered by a specific event. In this case, a GPIO pin write. When working on Lab 6 we initially had all the modules implemented separately. The intent was to combine everything in the end so we only had one executable(main). For the most part, this was achieved however when attempting to merge the internet module to the rest of the modules we encountered a problem. The program would build, but when trying to debug the target could not be found. Since the internet module and everything else worked separately and due to time limitations, we decided to keep them separate. In other words, one launchpad was dedicated to just handling the internet module while the other one did everything else. The launchpad that had the internet module was polling a GPIO pin waiting for it to be set to high. When the second launchpad detected that the HR exceeded the threshold it would write to its output pin which was connected to the other board pin that was polling. The GPIO pin was used to signal the other board telling it to post.

GUI Module

The GUI module is implemented in Python and it works on any Python-enabled computer with a UART port. Essentially, the Launchpad sends over the instantaneous heart rate using UART. On the other side, a Visdom module is implemented so that the GUI can be displayed on a webpage. Because the data is in floating point format, which is 4 byte, both sides (Launchpad and computer) needs to parse/recombine each number over UART, which only handle 1 byte at a time.

Difficulties

As mentioned previously one of the problems we ran into was towards when we tried to merge everything. Since we were running out of time our solution was to keep the internet module separate and have it flashed on another launchpad.

Another difficulty we ran into from the very beginning was simply getting the HR sensor to work. The sensor we used, MAXREFDES117, was designed to work with an Arduino or MBED platform. The included firmware was therefore not designed to work with the Launchpad out the box. It was necessary to port it so it could work with the CC3200. The problem was that the datasheet didn't have as much information as we would have liked, compared to the data sheets we've used in class, so getting the sensor to work was a challenge in itself. We fix this by translating Arduino I2C API to that of the CC3200.

Citations

In order to get the heart sensor to work, we used the included firmware that was provided to us by the manufacturer. Some of it had to be ported in order to work with our CC3200 since the included code is intended for an Arduino platform.

- Algorithm.cpp
- max30102.h/cpp

MAXREFDES117 Firmware