IoT - MEDICAL VITAL SIGNS MONITOR

AUTHORS : ADAM BELL, CONOR DUNNE | ALT 4 EMBEDDED SYSTEMS

[S-01a] : INVESTIGATION

DESIGN STATEMENT

Our project aims to design and develop an effective embedded system capable of addressing a real-world problem by utilizing a microcontroller paired with one or more input sensors to collect data, analyzing and processing it to deliver a clear and meaningful conclusion to the end user whilst demonstrating effective ability to understand embedded systems.

SOLUTIONS

Idea #1 – Motion Sensor based burglar alarm

PROS:

- The problem is beneficial for the end user and can solve a real-world problem
- Logic is advanced but demonstrates capability of multiple embedded systems
- Easy to prototype and test in a classroom environment

CONS:

- An external sensor (such as ultrasonic or infrared) must be ordered from a supplier
- Requires a diverse tech stack to accomplish (MicroPython (Embedded system), Express.js (API) & HTML
- False triggers can be picked up such as dust, moving objects or pets

Idea #2 – Smart Temperature/Humidity monitor

PROS:

- Sensors are already built into Microbit, requiring no order of external sensors
- The project is beneficial for the end user and can be used in any environment (homes, schools, etc.)
- Logic flow and programming is very simple to perform, and data can be easily displayed on a frontend service. (Example: Thingspeak)

CONS:

- Temperature & Humidity can be inaccurate due to microbit sensors, leading to inconclusive results
- It can be difficult to test and gain conclusions from data due to the environment only being limited to one classroom
- The project offers limited challenges and room to display capabilities of embedded systems

Idea #3 – Heart Rate & Oxygen Saturation (SpO2) vital monitor

PROS:

- The problem is beneficial for the end user and can be treated as a portable biomedical device
- Logic is very advanced and demonstrates capability of multiple embedded systems
- Testing in the classroom is very easy as results are provided in real-time

CONS:

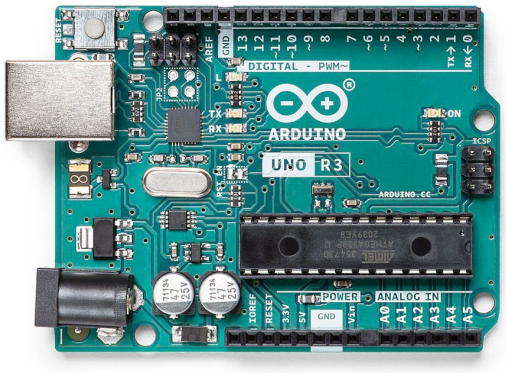
- An external sensor (such as MAX30102, or PulseSensor) needs to be ordered
- Requires a more capable embedded system to perform the needed math to determine SpO2
- Effective visualization of data to the end user adds increased logic complexity

[S-01b] : INVESTIGATION – PROJECT OUTLINE

PROJECT STATEMENT – WHY DID WE DO IT?

We chose to make a Heart Rate & Oxygen Saturation (SpO2) vital monitor because although the project requires a unique amount of complexity, we believe it is a great chance to get hands-on and practical usage of multiple embedded systems.

EQUIPMENT AND TECH STACK



(This includes technologies that couldn't be used to issues we encountered)

- Arduino Uno Rev3
- MAX30102 Sensor
- 0.96" Inch OLED Display Screen
- Arduino IDE (C++)
- Microbit:BBC & IoT:Bit (MicroPython)
- Male-Male, Female-Male Jumper wires
- Breadboard
- Express.js, HTML/CSS/JS, Thingspeak, & Vercel Serverless Deployments

PROJECT FEATURES

FEATURE	STATUS
Website & API to interpret data and form conclusions	No – Microbit didn't properly work with MAX30102, and Arduino does not have Wi-Fi to send data
Monitor Temperature, Heart BPM & SpO2	Yes – The library used in Arduino had built in functions to process raw data to effective results.
Display data through a physical embedded system	Yes – The library used in Arduino allowed to visualize data on an OLED screen
Use Gemini API (2.5f) or a local LLM to create personalized conclusions based on sex, weight, & height	No – This would've only worked if we used a website, plus local LLM requires CUDA for best performance
Use Thingspeak to visualize raw infrared data	No – Our final microcontroller choice (Arduino) did not have the ability to use Wi-Fi, and we also predicted that there may be latency/delay issues involved

At the start of the project, we originally planned to use IoT:Bit so we can get the MAX30102 data from the Microbit and send it to Thingspeak where we can visualize data but also create a website that can actively read data from Thingspeak to form more conclusions and advanced data visualizations. However, we could not pair the MAX30102 to the Microbit due to compatibility issues. We had made a backup plan to switch to Arduino in the event this occurs, and were able to develop a working solution after several failed attempts with the Arduino too (view s-06a / evaluation) for problems we encountered

TARGET AUDIENCE / WHO WOULD USE OUR PRODUCT?

The primary goal for us in this project is to provide a non-invasive method for measuring key-medical vitals that can be used by individuals of all ages, and all levels of digital literacy. The project is designed to be able to perform effectively in a wide range of situations such as fitness, healthcare, education, and home environments. So, this would allow us to have a very diverse target audience meaning anybody can use our device and receive conclusions that matter to the end user. We also believe that due to the very compact size of our project, you could use this on the go which makes it even more diverse for use cases for our target audience.

[S-02a] : PLAN

TEAM MEMBERS AND ROLES

ADAM BELL	CONOR DUNNE
Programmer – Develop the codebase to process results and display them with C++	Programmer – Attempt to develop the IoT:Bit code on Makecode to send data to Thingspeak
Graphics – Develop any needed graphics (flowcharts, diagrams etc.)	Logistics – Coordinating a list of resources required and ensuring we get them for every class
Project Documenter – Develops the project style, UI/UX and revises all details before submission	Records – Ensures we preserve images/videos/code to show project progression
Journal – Using Obsidian to log progression, ideas, etc. on a journal like document	Project Documenter – Develops the project document for submission
Researcher – Research development with Arduino, library documentation, and stackoverflow for issues related to C++	Researcher – Research possible solutions, IoT:Bit with MAX30102, and visualizing data with Thingspeak

RISK ASSESSMENT AND MITIGATION

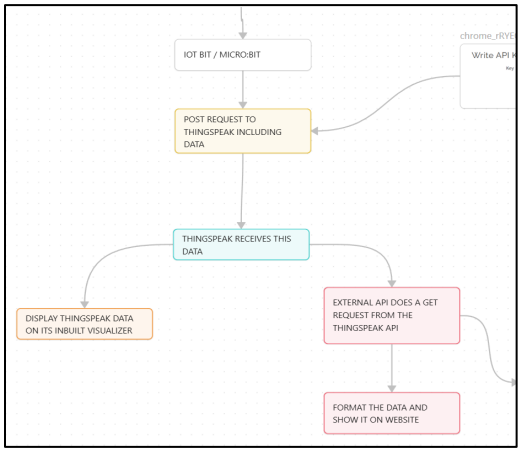
RISK	PLAN TO CIRCUMVET/MITIGATE
MAX30102 not working at all with the selected microcontroller	Multiple layers of backup (backup microcontrollers, backup max30102 sensor / different brand)
No scope for data visualization with the current setup	Multiple layers of backup (website, thingspeak, oled screen & console)
Physical wiring issues (such as voltage spikes, breadboard issues)	Planned backup of different wiring methods
Loss of data	Git version control + obsidian

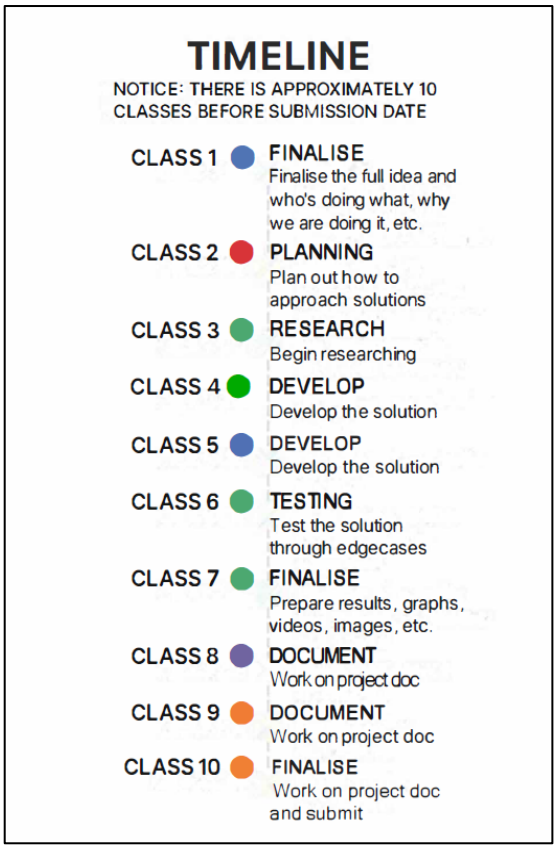
SOURCES LIST AND USAGE

SOURCE	USAGE
ChatGPT	Identify pros/cons of ideas
GitHub	Libraries and their documentation
YouTube	Tutorial on breadboards + IoT:Bit
Stackoverflow / various docs	Fix programming issues
OneNote	For flowcharts
Vercel	Serverless function docs

PREVIOUS PLANS ARCHIVE

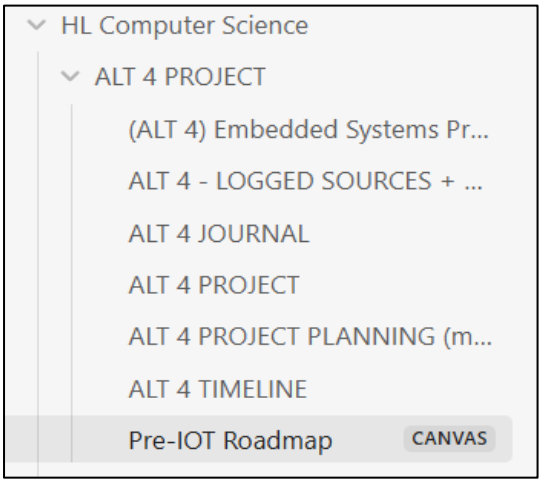
As we mentioned in “HOW DID WE PLAN?”, we used Obsidian which also allowed us to make rough flowcharts on the go. This was our first ever rough planned flowchart made in class two. It allowed us to quickly assess the flow of logic, and any potential issues we may encounter.





HOW DID WE PLAN?

We used Obsidian which is an advanced note taking app to plan everything, design rough flowcharts, and ensure we kept log of everything. We coordinated and communicated effectively using Teams. We followed a design thinking-based system which involves the CIDI process. (Clarify, Ideate, Develop & Implementation), Below is an image of how we organized our notes on Obsidian.



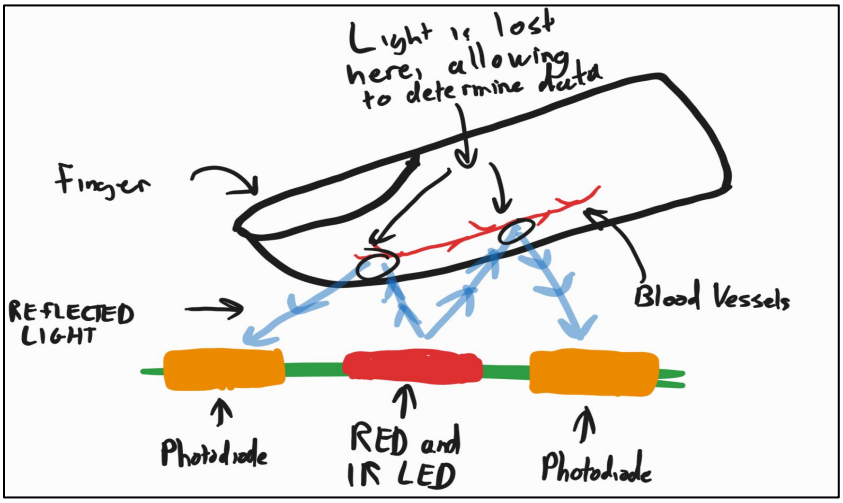
[S-03a] : DESIGN – MAX30102

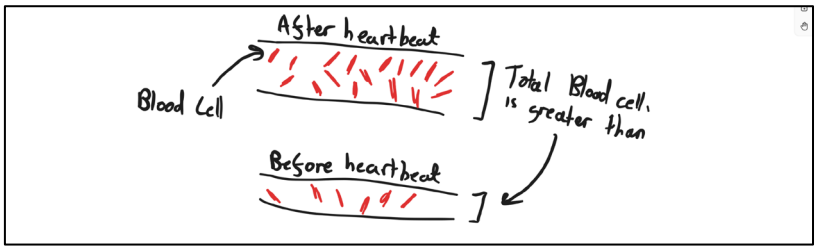
HOW DO WE MEASURE THE DATA?

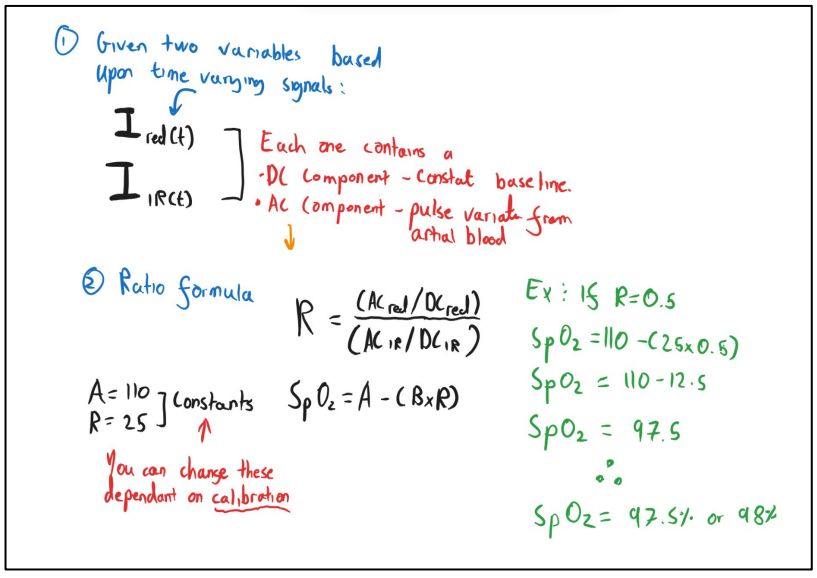
The method of measuring heartbeat, and SpO2 is called Photoplethysmography (PPG for short), which is a non-invasive method to measure results. It relies on the fact that blood absorbs light differently dependent on its volume and oxygen content. It involves two LEDs to illuminate the skin and is picked up by a photodiode.

HOW DID WE MEASURE THE HEARTBEAT?

When your heart beats, blood cells are flushed into your blood vessels which expands them and increases the overall amount of blood cells. The more blood cells in your vessels are equal to more absorption of light from the LEDs, hence returning less reflected light. In that sense, light reflection levels are equal to the end users heartbeat







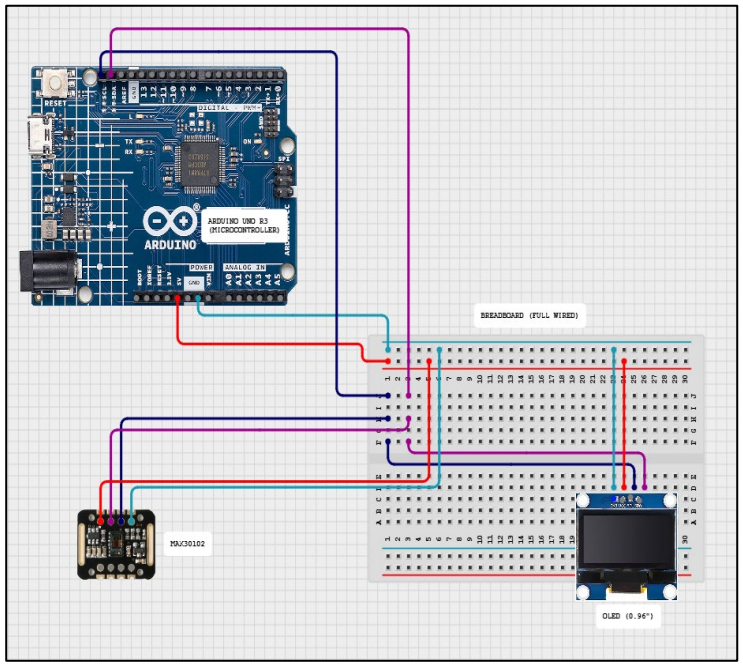
HOW DID WE MEASURE OXYGEN SATURATION?

Measuring oxygen saturation (SpO2) involves using math to equate the differences from two varying signals (the two LEDs), this is one of the limitations as the Microbit did not have the computational power to perform the math in real time. In medical terms, SpO2 is the estimate of how much hemoglobin (Hb) in the blood carrying oxygen (HbO2). Hemoglobin changes its color dependent on its oxygenated status, so oxygenated blood is more absorbed to infrared whilst deoxygenated is more absorbed to red light. On the image on the left is the full demonstration and breakdown of the math and how it works.

WHY IS THIS ALL IMPORTANT?

When designing this it is important that we know the exact science and math behind how everything works, such as the MAX30102 sensor. Knowing how things work the way they do allows us to actively work on the project without being “blind” in what we are attempting to accomplish. In the next page we will explain the flow of logic, how the wiring layout works and all relevant flowcharts.

[S-03b] : DESIGN – WIRING / FLOW OF LOGIC

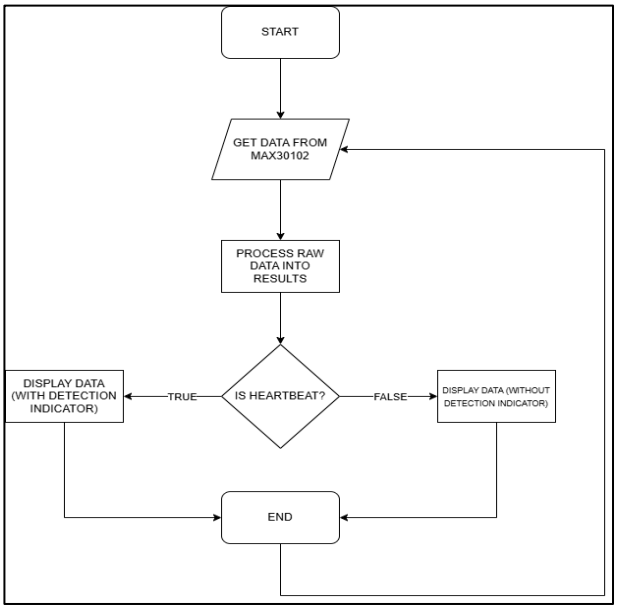


DATA TABLE / INPUTS

NAME	TYPE	PROCESS	OUTPUT	PURPOSE
MAX30102	ANALOUGE INPUT	Uses external library to process data	Heartbeat, SpO2, and Temperature	Measures vital data and process it
Arduino (Uno Rev3)	DIGITAL	Determines the state of the UI on the OLED	Signals to the OLED on what to display	Acts as a middleman between the sensor and OLED
OLED DISPLAY (0.96")	DIGITAL OUTPUT	Displays text from commands given by the Arduino	Visual display of vital readings in a nice UI style	Provides real time feedback to the end user

WIRING INFORMATION

The wiring bridges **GND** and **5V** to (- and +) rails on the breadboard, allowing for easy distribution to OLED + MAX30102. On COL1 we bridge **SCL** and on COL3 we bridge **SDA**. This allows us to communicate with the devices. Mixing isn't an issue because of memory addresses.



NAME	GROUP	PURPOSE
SDA	I2C BUS	Transfers data on I2C
SCL	I2C BUS	Clock to sync I2C
GND	POWER	Common ground
VCC	POWER	Supplies 5V power

SIMPLFIIED LOGIC FLOWCHART

On start signal: Like all libraries, sensors require an initialization call to begin `[ex: foo.init(params)]`

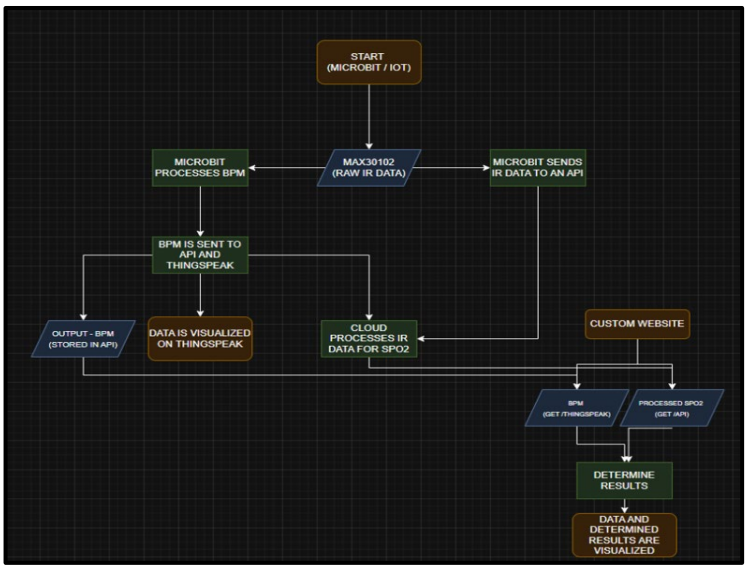
Process raw data into results: The library we use (which you will get to see in [s-04a] has functions to process our data into readable results.

Is Heartbeat? We use an if statement to determine if heartbeat is true and last beat was less than 400ms ago

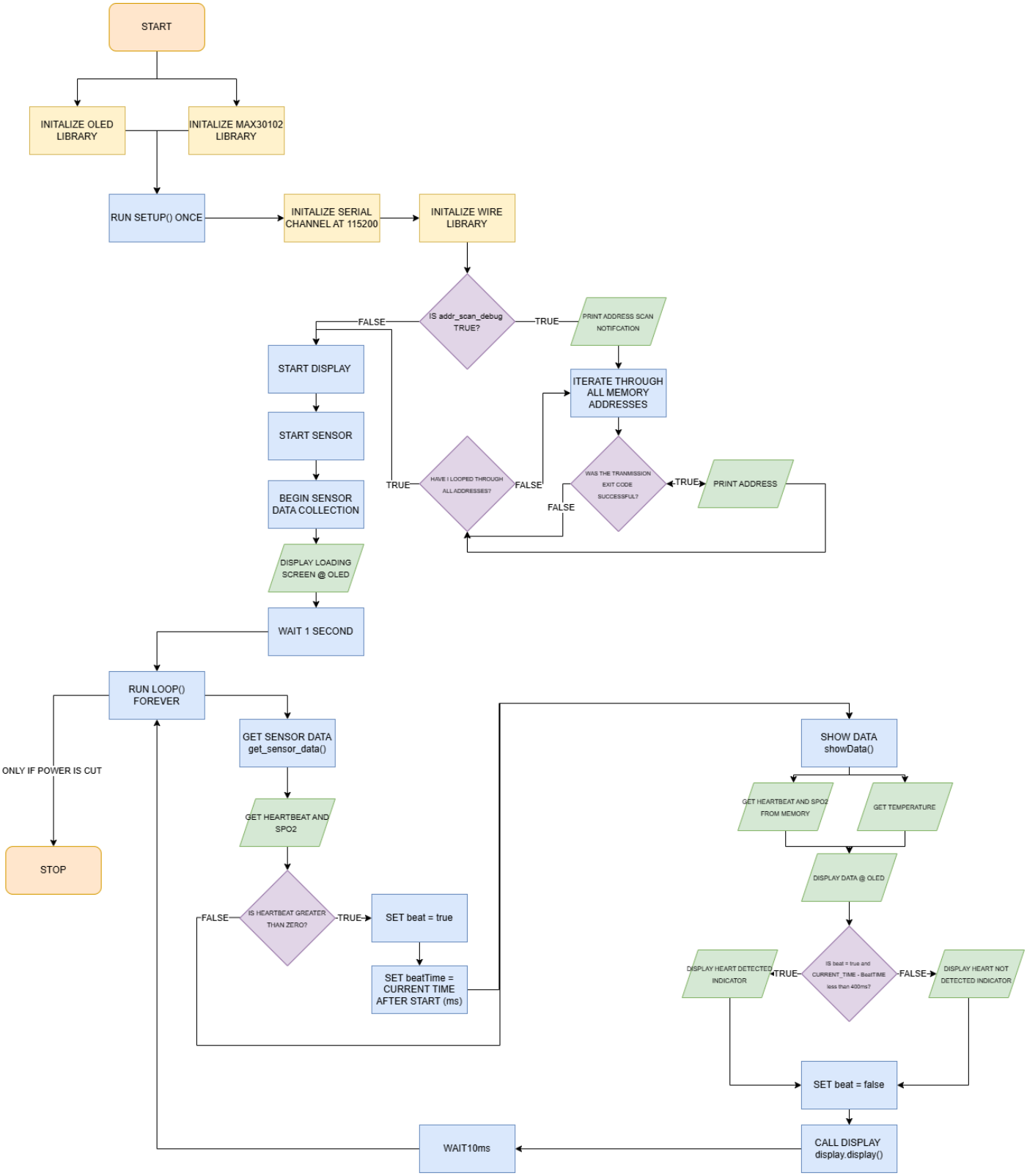
Stop: This all happens in 1 event, it will repeat again

PREVIOUS FLOWCHART (PROTOTYPE #1)

The design process constantly includes reiteration and that is something as a team we had to experience. (We will further reflect on this in evaluation). The idea was to allow the Microbit to process the BPM and send that including RAW data to Thingspeak. We would then use Vercel's serverless functions to compute SpO2 because it had the needed computational power that the Microbit did not have. We would then display all the data on a website which would also include AI processing. However, due to problems we could not make this flowchart into reality. View evaluation for problems we encountered.



[S-03c] : DESIGN – IN-DEPTH FLOWCHART

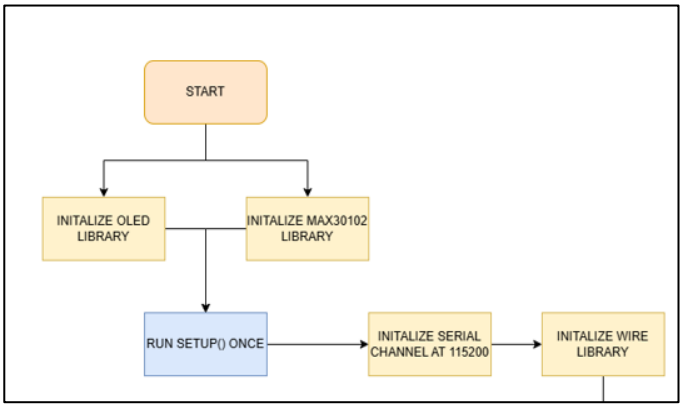


FLOWCHART NOTES

Arduino flashes its code and runs under `void loop()` forever unless interrupted by a soft-kill (button/command to stop it) or a hard-kill (power cut). The only way we thought of including the END is by implementing the hard kill. Secondly `get_sensor_data()` and `showData()` is to represent that the logic underneath it is part of a function. In the next page we will do further analysis of the flowchart.

[S-03d] : DESIGN – FLOWCHART ANALYSIS

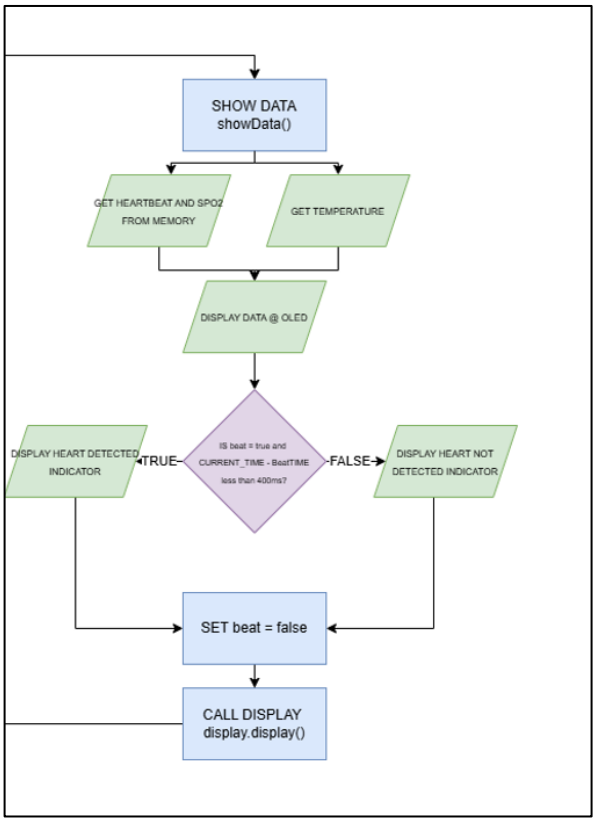
In this page we are only going to analyze some of the more "complex" portions of the flowchart so there is a further understanding of why things the way they are. We also included pseudo code to help you visualize how it would work in a real code situation. You can see how it works in real code in the [S-04] CREATE section



The first two process blocks occurs before `void setup()` is called, their main job is to create a "controller object" so that it can be called later on in our code. *[PSEUDO CODE BELOW]*

```
CREATE a display object from AdaFruit_SSD1306 USING:
    -   SCREEN_WDITH, SCREEN_HEIGHT, &Wire,
        clock_reset
CREATE A sensor object from DFRobot_BloodOxygen_S_I2C
USING:
    -   %WIRE and SENSOR ADDRESS
```

This is the most important portion of the code as it is the largest function and does the most (even has 34 lines). First of all we get the heartbeat and SpO2 data from memory which is very important. They are defined in memory by `get_sensor_data()`, we use a custom function to handle this because we want to `set beat = true` only if the heartbeat is greater than zero. It then returns `-1` if it cannot detect a valid heartbeat. We want to create a variable that tells us if there was a beat or not so we can display it on the frontend user interface whether it is detecting a heartbeat or not. View [S-03d] for UI/UX information. Lastly we also check whether last beat was less than 400ms ago, and if so and the beat is true then we fill the indicator, so it indicates to the end user that we have successfully detected a heartbeat. We use a very simple method to determine this by filling in a circle (if filled, then we know there is a heartbeat, otherwise if there is not we do not fill the circle leaving it empty). Display.display() draws everything onto screen.



WHY DO WE FOLLOW AN OOP (OBJECT ORINTENATED) METHOD?

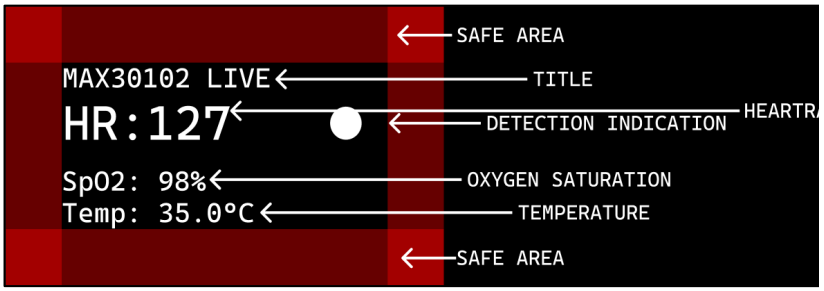
Although the code isn't a full OOP project, it has been designed so that we use functions in almost every step of it. In the flowchart you can see that we have taken use of two custom functions and two required functions by the Arduino. We made it this way because working in a "linear" method can become very clumpy and make the overall code look disorganized.

WHAT IS "addr_scan_debug" IN FIGURE 1 ON [S-03B]

This will be further explained in [S-04] create section, but here is a quick brief on what this actually does. It is a variable that can be changed (not during runtime) but if it is true, it will scan all memory addresses that may be linked on the I2C transmission system. If we get a exit code of 0 (success) then we print the address. We did this because we are aware that \our OLED and sensor may have a different address than what the library we used actually expects.

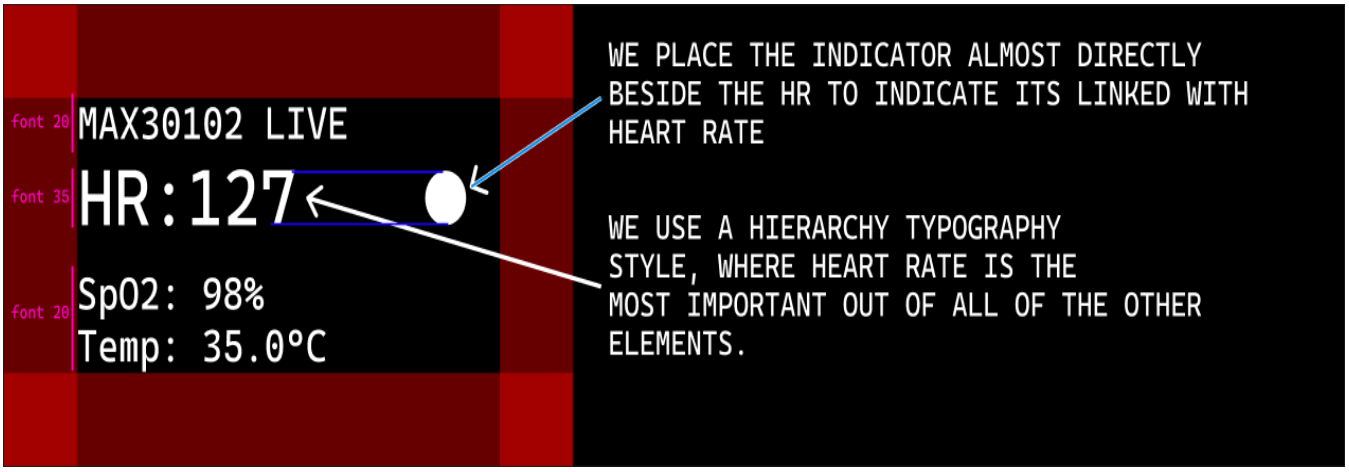
[S-03e] : DESIGN – UI/UX & UNIVERSAL DESIGN

Please note, all images (except for IRL demonstration image below) are made using Figma and may not be an accurate representation of direct physical dimensions and is only a near accurate representation of what is seen from the IRL point of view. Figure 4 : stripe.com/ie (05/11/2025)



USER INTERFACE OVERVIEW

Safe area is a portion of the OLED that is visible to the user without being obstructed. Detection indication as mentioned in [S-03c] is to let the end user know whether we have picked up their vitals or not, rest of the elements are outlined in the image.



TYPOGRAPHY AND POSITION REASONING (UI/UX)

In typography one of the best ways to convey importance or direct the user's attention is to use "visual hierarchy" which can be achieved through font weight, size, and even color. However, in our project we use font to show importance. This can also be seen by Stripe where they use a distinct color and font size to determine importance.

UNIVERSAL DESIGN REASONING

According to our research you cannot be colorblind to shades of white. We did originally plan to color code the heart rate element based on whether you were going to die or not. However, we knew this would make it inaccessible to people with color blindness, so we kept it white making it accessible. Secondly, we tried to avoid all usage of complex English words to make it somewhat multilingual to the end user. For example, SpO2 is a chemical name and is internationally recognized, the unit of measurement °C is also Internationally recognized, and lastly the detection indicator bypasses all barriers of language entirely.

[S-04a] : CREATE – EQUIPMENT / TECH OVERVIEW

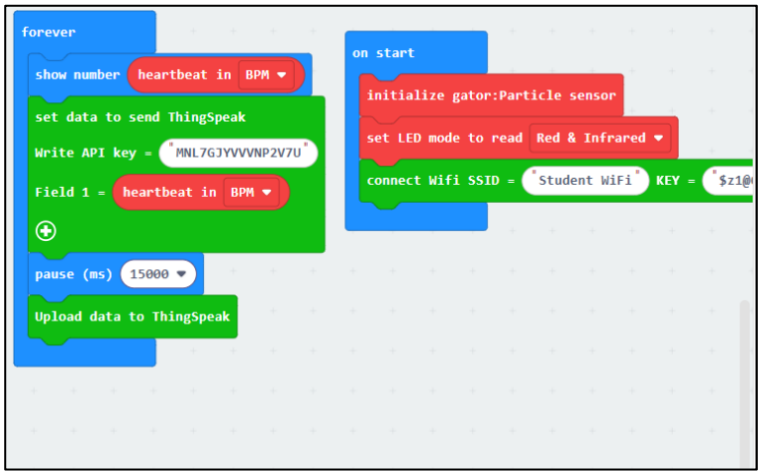
This also includes all the technologies we never got to use due to switching to different solution ideas, project limitations, etc.

NAME	TYPE	STATUS	PURPOSE
Arduino Uno Rev3	DEVICE	IN USE	Bridges communication between I2C devices and processes all analogue data into readable data
Microbit	DEVICE	NOT IN USE	Bridges communication between I2C devices, processes heartbeat and sends analogue data to IoT:Bit
IoT:Bit	ACCESSORY	NOT IN USE	Receives analogue data and processed heartbeat from the Microbit and sends data through Thingspeak
MAX30102	SENSOR	IN USE	Uses PPG to get analogue data to send by I2C
0.96in OLED	DISPLAY	IN USE	Receives I2C communication from a device to display information such as data on its screen
Microbit Breakout Board	ACCESSORY	NOT IN USE	Bridges more pins from the Microbit making connecting them to sensors, displays, etc. easier
Adafruit_SSD1306	LIBRARY	IN USE	Allows communicating to OLEDs with a SSD1306 driver from a device instead of blindly touching memory without memory safety controls
DFRobot_BloodOxygen_S	LIBRARY	IN USE	Allows communicating with the MAX30102 sensor and providing functions to perform the math to get SpO2, Temperature and Heartbeat
Adafruit_GFX	LIBRARY	IN USE	Provides the graphical functions and drawing functions, so commands can be sent to its dependent library (Adafruit_SSD1306)
gator:particle	LIBRARY	NOT IN USE	A MakeCode extension that processes analogue input from the MAX30102 and only output the heartbeat and raw analogue data.
Thingspeak	SERVICE	NOT IN USE	Visualizes data from the IoT:Bit on the internet using an API system
Vercel	SERVICE	NOT IN USE	Provides rapid deployments and serverless functions to the internet to make websites, APIs, etc.
Express.js	LIBRARY	NOT IN USE	Provides middleware communication between the frontend and backend to develop APIs, serve static files, etc.
Arduino IDE (C++)	PROGRAM	IN USE	Provides a developer environment to code and compile C++ and easily flash it onto the Arduino
Breadboard	ACCESSORY	IN USE	Used for easy prototyping and multiple connections on GND, 5V, SDA, SCL
Resistor	ACCESSORY	NOT IN USE	Manages voltage and prevents potential damage to circuits that can't handle certain voltage levels
LEDs	ACCESSORY	NOT IN USE	Used for testing to see if 5V and GND were properly working or not
iot-environment-kit	LIBRARY	NOT IN USE	A Makecode extension that allows for easy configuration and communication between the IoT:Bit and Microbit

[S-04b] : CREATE – PROTOTYPE #1

PROTOTYPE STATEMENT

The first ever prototype we had decided on this solution was to use a Microbit (paired with the IoT:Bit), our original plan was to process the heartbeat data using the gator:particle library, which we would then send the heartbeat and raw data to Thingspeak. (You can also look at figure three in [s-03a] for the flowchart)

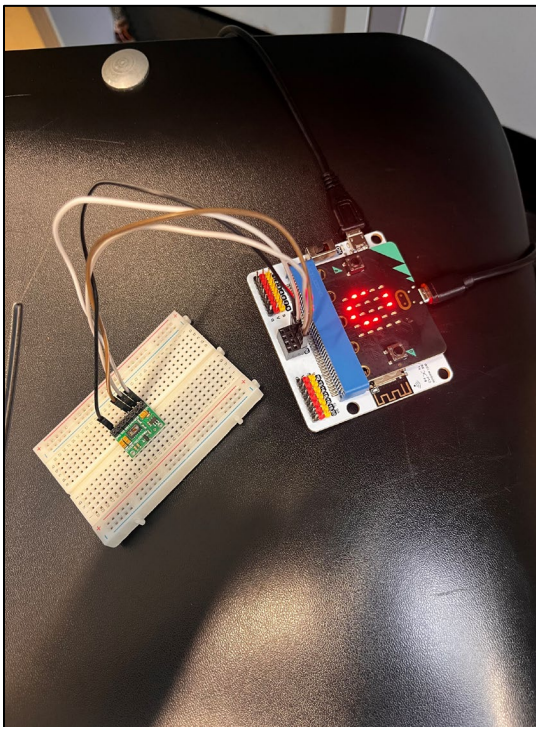


CODE ANALYSIS #1

This was our first ever code that we had implemented, the main goal was to initialize the MAX30102 (gator:particle) library and initialize the IoT:Bit to connect to the Student Wi-Fi. The IoT:Bit library had only supported API POST requests to Thingspeak (which was also another major limitation that we had to encounter). The API key had to be made from Thingspeak, which is a unique credential that allows us to send POST requests. (GET had a separate API key)

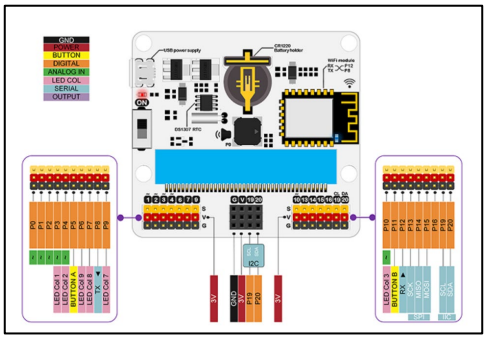
ISSUE / BUGS LIST OF PROTOTYPE #1

ISSUE	STATUS
MAX30102 was not powering on	Not resolved – No root cause could be found
Microbit could only process HR and not SpO2	Resolved – Thingspeak supports RAW signals
SDA and SCL could not be found on the Microbit	Resolved – IoT:Bit completely solved this issue



ISSUE INVESTIGATION / HYPOTHESIS

We hypothesized that issue #1 was occurring because there is so many different MAX30102 brands and it is possible that the only library we had found (gator:particle) just simply did not support our specific MAX30102. We had also tested it with our second MAX30102 and it did not work either. Power issues were ruled out because the working voltage is 3.3V minimum and 5V maximum, due to this major issue we had to move to Arduino, which also included its own issues during creation.



PROTOTYPE #1 – LAYOUT / WIRING

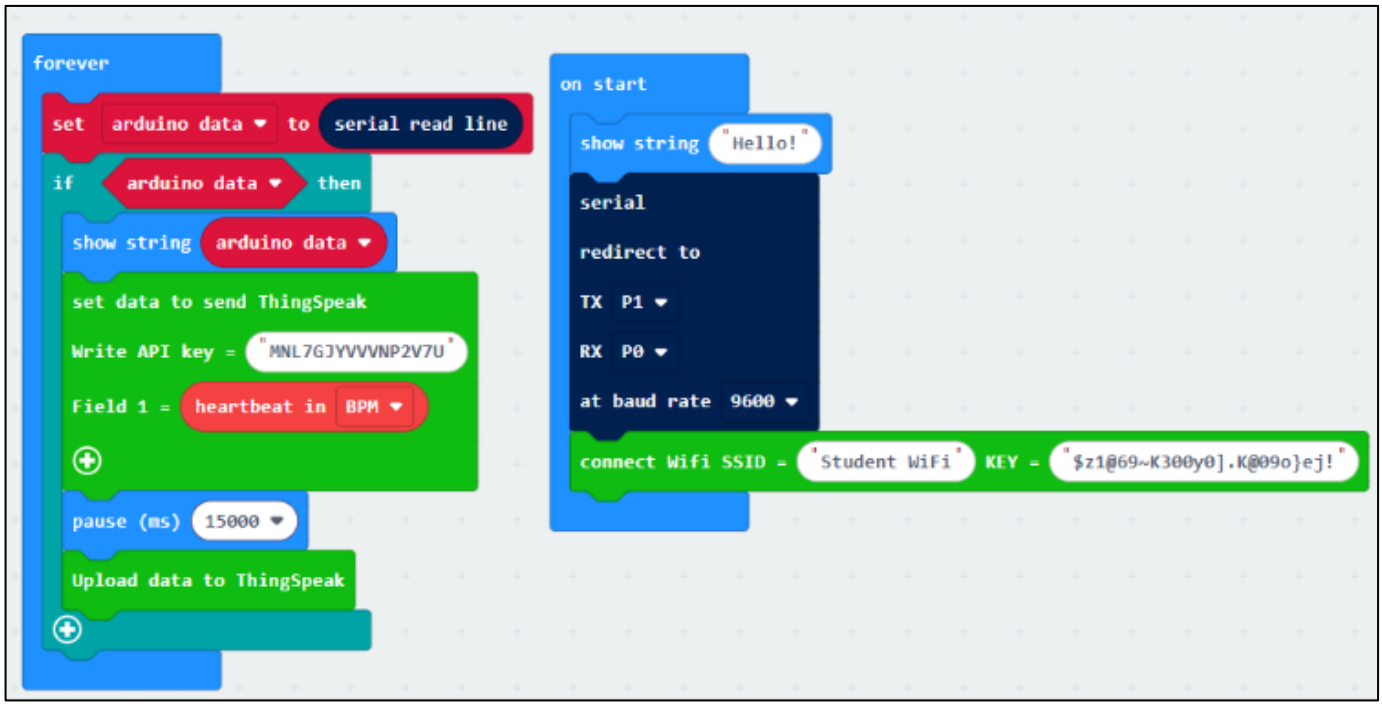
Prototype #1 and our final working design have a very similar wiring layout where we had GND, VCC, SDA and SCL, however it was much simpler as we did not have to work with any other accessory (such as the OLED). We had connected the wires on the IoT:Bit labelled as: G, V, 19, and 20. (where 19 was SCL and 20 was SDA). We also had to use a breadboard for stability on the MAX30102

[S-04c] : CREATE – FINAL DESIGN (MAKECODE)

Website: makecode.microbit.org | Second website: Thingspeak.mathworks.com

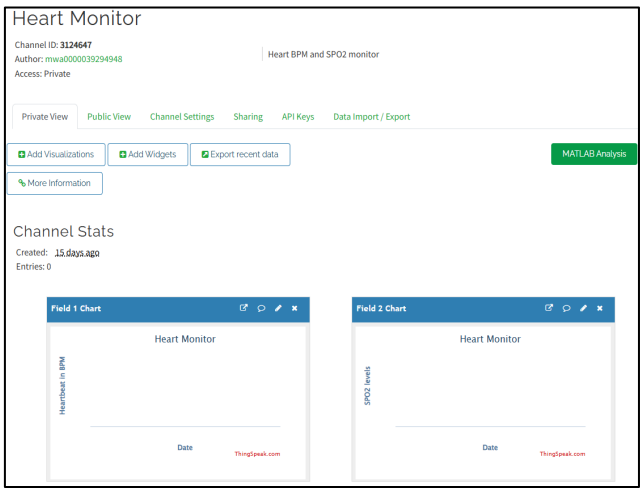
FINAL DESIGN FOR MAKECODE STATEMENT

From all the previous pages of this document you have begun to see the complexity that was involved in the final design, however, we never talked about the history behind it and some of the features that we never got to implement. In this page we will discuss the code we had made from Makecode tailored for the Arduino.



CODE ANALYSIS #2

This code was tailored to receive data from the Arduino using TX/RX ports (which were both available on both boards) and then upon receiving the data from the Arduino, the IoT:Bit would send over the data to Thingspeak where we would be able to visualize it and display it on a custom website. A key analysis in this figure is the if statement in the forever loop. This statement is used to only POST data through the Thingspeak API if there was any available data coming from the Arduino.



BUGS LIST OF FINAL DESIGN #1

ISSUE	STATUS
TX/RX was not being received by Microbit from Arduino	Not resolved – No root cause could be found

ISSUE INVESTIGATION

We had hypothesized that the RX/TX was not communicating because of a potential disruption between the Arduino and the school computer. RX/TX is also combined with USB connections (which the Arduino was connected to a computer with) at the time.

(The baud rates between the Arduino code and Makecode are different, but we had fixed that a few minutes after the image was taken, however it still should've showed signs of communication even if unreadable, if it was on a different baud rate

THINGSPEAK ON FINAL DESIGN

The image on the left is what we had originally planned to use to show data by IoT:Bit after receiving data from the Arduino. This is from Thingspeak.

[S-04d] : CREATE – FINAL DESIGN (C++ / ARDUINO)

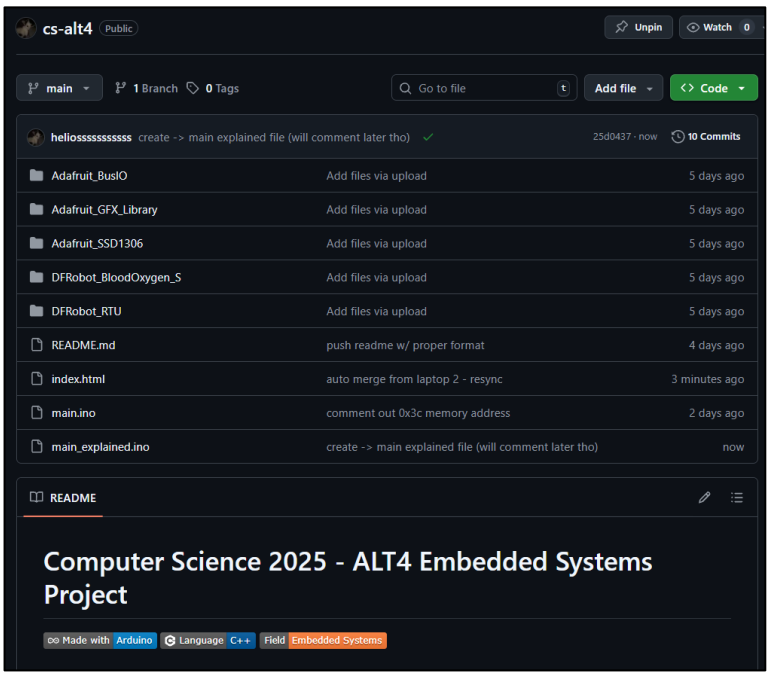
Please note, the following codebase is fully public and is published by Adam Bell on GitHub, the repository of the project also includes a fully comment/explained version and all the libraries. Repository URL: https://github.com/heliosssssssssss/cs-alt4

FINAL DESIGN FOR ARDUINO STATEMENT

When we were developing the C++ code for the Arduino, we encountered a lot of issues especially with the fact that we didn't want to actively work with memory. We would rather outsource memory control to a library (Adafruit_SSD1306, DFRobot_BloodOxygen_S). Below is how we created and laid out the repository.

GITHUB REPOSITORY

We used GitHub because of the familiarity with its source control which allowed us to easily preserve code and work with it during the create progress. In this section we will show you what each file is and why it is there. Firstly, the first five folders are all the libraries (or their dependencies), if you wish to try this exact setup you must copy all five over to \Documents\Arduino\... so you can import the libraries. main.ino is where the code is, and main_explained.ino is the commented version for further explanation. In the next page we will look through the code and compare with the flowchart from [s-03b]




```ino
#include <DFRobot_MAX30102.h>
DFRobot_MAX30102 particleSensor;

void setup() {
  Serial.begin(115200);

  while (!particleSensor.begin()) {
    Serial.println("MAX30102 was not found");
    delay(1000);
  }
  particleSensor.sensorConfiguration(
    0x1F,           // ledBrightness
    SAMPLEAVG_4,
    MODE_MULTILED,
    SAMPLERATE_400,
    PULSEWIDTH_411,
    ADCRANGE_4096
);
}

void loop() {
  Serial.print("R=");
  Serial.print(particleSensor.getRed());
  Serial.print(" IR=");
  Serial.print(particleSensor.getIR());
  Serial.println();
  delay(100);
}
```

OUR FIRST ATTEMPTS

The first few attempts were very rough, because we had to go through several different libraries all claiming they work with the MAX30102, which all had different ways of naming functions, processing data, etc. This meant we had to go through several lists of documentation and code examples and rule out every single library till we found a working one. What was interesting is that the working library is named under the same brand as the one in the first attempt but work differently. Eventually we were able to find a working library (which you will see in the next page). This was also before we even had thought on how to code about the OLED, as we wanted to make sure the MAX30102 works first, as that is the more important thing in the project.

BUGS LIST OF FINAL DESIGN #1

ISSUE	STATUS
0x6B LCD (backup of OLED) not working – Address @ L:13	Not resolved – Backlight requires its resistor to be soldered
0x3C OLED has screen jitter	Not resolved – No root cause found
SDA SCL Transmission sometimes bugs out and gets unwanted noise	Not resolved – No cause found, hypothesized as a wiring issue
Heartbeat (not SpO2 though) picks up -1 even with finger on the sensor	Not resolved – Issue is hardware related and is out of our control
MAX30102 doesn't work	Resolved – Proper library found

Please note this also includes issues that we have as part of our final design and will be further discussed in evaluation. This table represents issues that are present in the final code

[S-04e] : CREATE - FINAL CODE / IN-DEPTH ANALYSIS

Please note, the following codebase is fully public and is published by Adam Bell on GitHub, the repository of the project also includes a fully comment/explained version and all the libraries. Repository URL: https://github.com/heliosssssssssss/cs-alt4

There is a 4-minute rapid code explanation video that explains the codebase line by line: https://www.youtube.com/watch?v=gRYU9wE2Gk0

```
if (addr_scan_debug) {
  Serial.println("[notify] : address scan init")
  for (byte a = 1; a < 127; a++) {
    Wire.beginTransmission(a);
    if (Wire.endTransmission() == 0) Serial.print("0x"), Serial.println(a, HEX);
  }
}
```

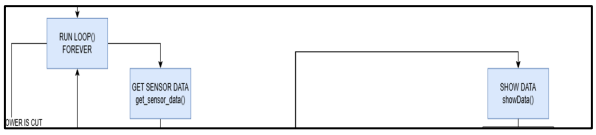
ANALYSIS – I2C SCANNER

Projects like this tend to have a lot of room for issues, due to that we implemented this I2C memory address scan, easily helping us identify where the problem/bug was occurring.

I2C SCAN STATE	CONCLUSION
I2C address found, and the device connected is not functional	The device is being detected by I2C, however the library communicating with the device is not functional
I2C address not found, and the device connected is not functional	The device has a wiring issue, common issues such as mixed SDA SCL, or loose wire
I2C address not found, and the device connected is functional	It may not even be using I2C communication and could be using RX/TX for example
Unknown I2C address found	Potential source causing unwanted noise

ANALYSIS – LOOP() STRUCT

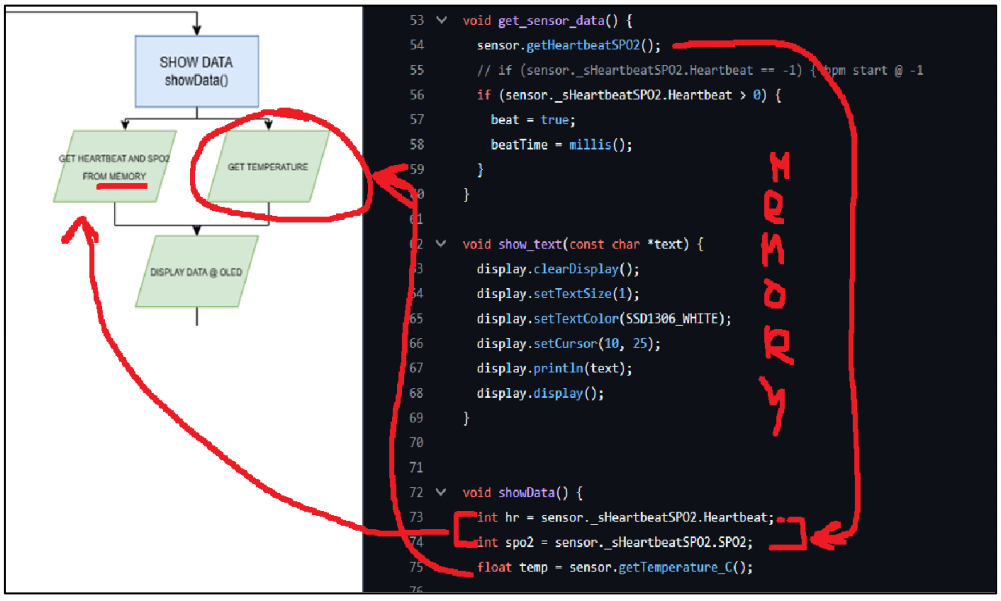
In the flowchart we specifically show that get_sensor_data() occurs first, because in the codebase we follow a linear path rather than performing any multithread or coroutines. This shows the flow of logic/flowchart in reality




```
void loop() { // constant
  get_sensor_data();
  showData();
  delay(refresh_cycle_delay);
}

void get_sensor_data() {
  sensor.getHeartbeatSPO2();
  // if (sensor._sHeartbeatSPO2.Heartbeat == -1) { bpm start @ -1
  if (sensor._sHeartbeatSPO2.Heartbeat > 0) {
    beat = true;
    beatTime = millis();
  }
}
```

ANALYSIS – SENSOR STORED VARIABLES FOR HR / SPO2 VITALS



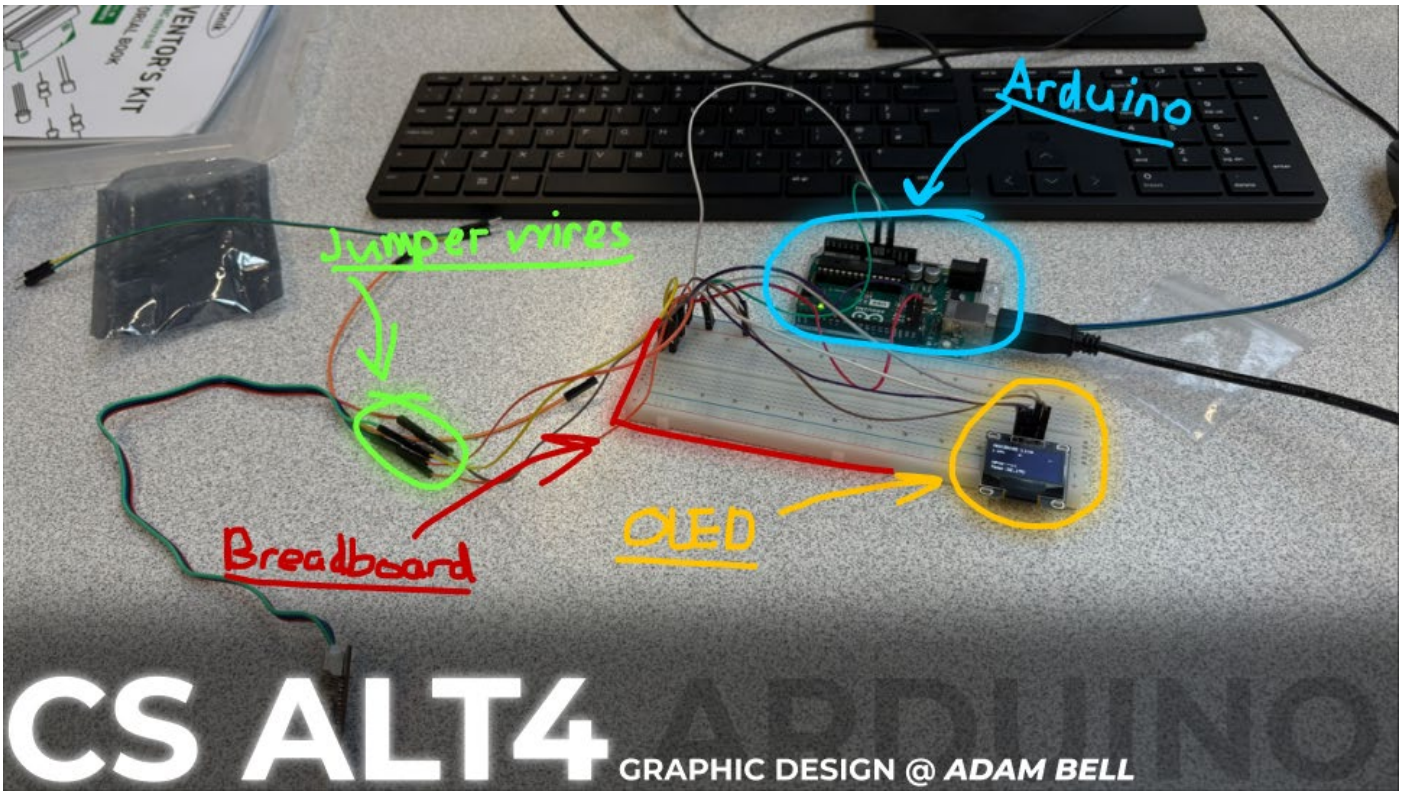
The only reason why we do not also get temperature in get_sensor_data() is purely because the function also determines the beat, and beatTime variables, so it would be a bit unnecessary to push L:75 into that function. We directly show this on the flowchart and show how it correlates with our code. The drawings indicate where everything goes.

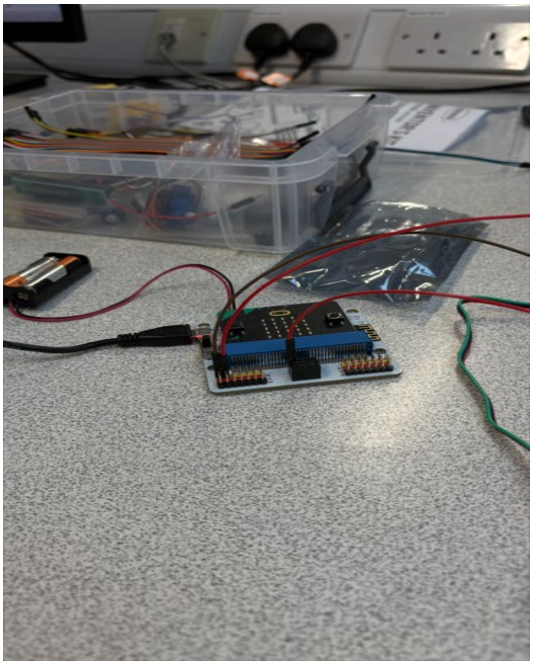
[S-04f] : CREATE – THE FINAL PRODUCT

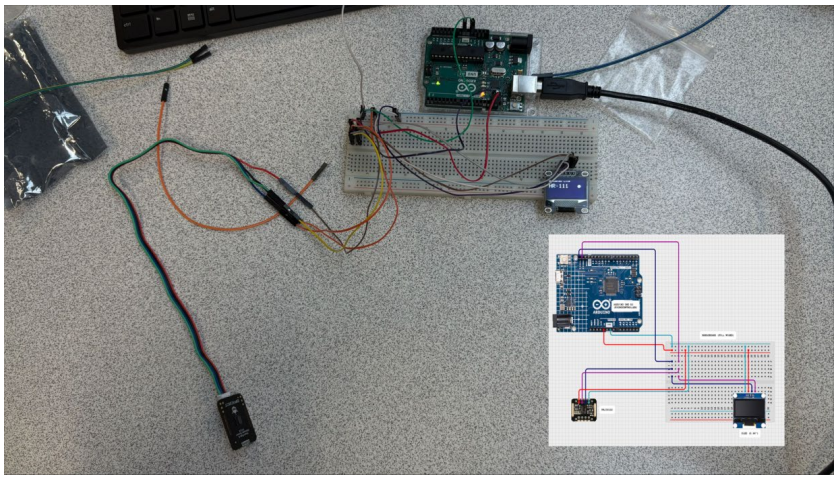
There is a short unlisted video demonstrating the final product in action with the OLED: https://www.youtube.com/shorts/W5-Y21DuUVw

FINAL DESIGN/RESULT STATEMENT

We had encountered many issues from the Microbit not working with the MAX30102, the IoT:Bit being limited to POST requests with Thingspeak, the Arduino and the several libraries that didn't worked with the MAX30102, the failed RX/TX connection line between the Arduino and the Microbit, and lastly the weird little issues we didn't resolve (such as the SDA SCL transmission getting unwanted noise), however what we did was accomplish our main goal after counteracting multiple problems. We will further reflect on all of this, in evaluation however this page is dedicated to showing some of the photos and videos. You can check out the demo video here: **https://www.youtube.com/shorts/W5-Y21DuUVw**







These are the photos we took during the final class when we finally got it working. We wanted to make the last slide of [S-04] full of images to show how some text and diagrams can be all made into reality.

Figure 1 - Attempt to connect RX/TX

[S-05a] : EVALUATE – TESTING & RESULTS ANALYSIS

(IC-N) = In-class normal, (WHO) = World Health Organization, (NHS) = United Kingdom National Health Service, (No data for HSE could be found)

TEST TABLE / EDGE CASES

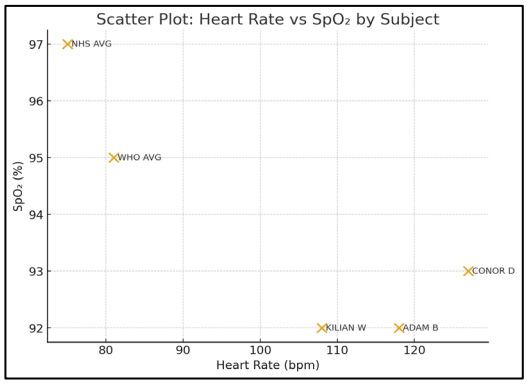
NO	TYPE	TEST DATA	REASON	EXPECTED OUTCOME	ACTUAL OUTCOME	STATUS
001	NORMAL	Finger placed normally on sensor	To confirm HR, SpO2, and Temp display correctly	OLED should display normal readings	OLED displayed normal readings with small latency delay	PASS
002	INCORRECT	No finger placed on sensor	To determine HR, SpO2 and Temp with no input	OLED should display -1 on readings and detected indicator should be outlined only	OLED displayed -1 and the detected indicator was outlined with small latency delay	PASS
003	EXTREME	Strong pressure on the sensor from finger	To simulate poor sensor conditions	OLED readings should remain normal or slightly fluctuate	OLED readings slightly dropped briefly but returned to normal within few seconds	PASS
004	BOUNDARY	Connect an extra 3.3V source to rail	To test how the Arduino and MAX30102 handle unexpected voltage	Arduino should be the first to shut down, then the MAX30102	Both devices turned off instantly and the Arduino required a reset	PASS
005	BOUNDARY	Slightly disconnect SDA during runtime	To test response to signal failure	Issues should occur but be able to resume instantly once reconnected	OLED elements bugged out and froze, reconnect did nothing and an Arduino reset was required	FAILED

TESTING CONCLUSIONS / FINAL STATEMENT FOR TESTING

From all 5 tests we had done on our project, we confirmed that 4/5 tests were able to successfully pass and the final test failed (being the boundary test). We predicted that the I2C should be able to resync with itself automatically once reconnected, however it had caused a weird visual bug with the OLED. To resolve this issue, we had to reset the Arduino using the reset button. However, besides the fact that we failed the last test, the other four tests did pass successfully, so we can further say that the product did work as intended in testing.

RESULTS ANALYSIS & RELIABILITY (16-17)

SUBJECT	HR	SPO2	TEMP	ENV
WHO	~81bpm	95%	36.5 °C	NORMAL
NHS	~75bpm	97%	37.0 °C	NORMAL
CONOR D	~127bpm	93%	31.0 °C	IC-N
ADAM B	~118bpm	92%	34.2 °C	IC-N
KILIAN W	~108bpm	92%	33.8 °C	IC-N



CONCLUSIONS / ANALYSIS

From the two graphs we can determine a few major and important conclusions which will be listed below:

- In-class results tend to lean more grouped together shown in the scatter plot, and there is a direct distinction of two groups. One being IC-N (bottom right), other one being NORMAL (top left).
- The SpO2, and temperature do not deviate as much compared to the HR seen in the scatter plot
- The average deviation between IC-N and NORMAL is 39.7bpm which could indicate a potential bias source in the MAX30102 or in the IC-N environment

[S-05b] : EVALUATE – PRODUCT ANALYSIS

DOES THE PRODUCT WORK AS INTENDED?

Yes and no. Our original intent was to design the product so that we could display data onto Thingspeak and our custom website using IoT:Bit, however due to implications we were not able to make that a possibility. However, we did meet our core intent which was to measure HR, SpO2, and Temperature using the MAX30102, regardless of what embedded system we worked with.

INTENT	STATUS
Display data through Thingspeak and our website	NO
Display data through any other method	YES
Receive data from the MAX30102	YES
Microbit + IoT:Bit as the embedded system	NO
Arduino as the embedded system	YES
Personalize data results on the website using AI	NO

DOES IT MEET THE BRIEF?

Yes, in accordance with the LCCS specifications, our product is actively using a microprocessor (Arduino) that is using external sensors (MAX30102) and is receiving input and outputting to an OLED, which is all part of an embedded system. It meets all four learning outcomes, and all were successfully applied to the project.

NO	LEARNING OUTCOME	STATUS / HOW IT WAS APPLIED IN THE PROJECT
3.11	Use and control digital inputs and outputs within an embedded system	The Arduino actively controls digital output to the OLED
3.12	Measure and store data returned from an analogue input	The Arduino processes raw analogue data into readable result and stores in memory (view [s-04e], figure 4 for data storing using variables)
3.13	Develop a program that utilises digital and analogue inputs	The Arduino processes digital input and analogue input (MAX30102, and OLED)
3.14	Design automated applications using embedded systems	The maths performed to calculate SpO2 is entirely automated and requires no human intervention to calculate the SpO2

POSSIBLE FEATURES TO ADD
WRIST STRAP (PROTECTIVE STRAP) for the MAX30102 for easy portability
ESP32 Microcontroller to broadcast data removing the need of jumper wires everywhere
iOS/Android app that can page emergency contacts if the HR or SpO2 of the end user enters critical states
NIR spectroscopy to monitor glucose, creating a niche audience for those with diabetes

WHAT OTHER FEATURES WOULD WE ADD?

There is a lot of other features we would want to add but we think one of the biggest focuses would be making the product more accessible and viable to the end user in all situations and environment, this would include making the product more compact and safer during different environments. We also wanted to extend the target audience using other non-invasive measurement techniques

WHAT CHALLENGES OCCURRED THAT AFFECTED OUR TIMELINE?

Every single team for the ALT4 project had been paired with three people, however as seen in [s-02a] we had only assigned roles to two members because the other team member was not actively present during the duration of the project timeline. This meant the work rate for the remaining two members had to exceed expectations to deliver results (which we did). Due to a critical circumstance that occurred with one of the team members, we were limited down to one member for a good portion of the classes before the midterm, causing issues of its own such as the MAX30102 not connecting with the Microbit. However, because we had planned backup plans in the event of major issues like this arising, we were easily able to switch towards a different embedded system strategy, therefor allowing us to complete a working product.

[S-05c] : EVALUATE – SELF-REFLECTION

WERE WE TOO AMBITIOUS OR NOT ENOUGH?

In our opinion we both agreed that you can never be "too ambitious", otherwise how would we be able to achieve our end goal? Had we not been ambitious and passionate enough, we would've called it a day once the MAX30102 did not work with the Microbit and say, "oh yeah, that is enough for a passing project report!". However, we do recognize that there is a difference between being ambitious and not accepting the facts. As we both further reflected, we realized that much time could have been saved had we done a more in-depth investigation with compatibility instead of following a "trial and error" approach with the MAX30102 connected to the Microbit.

WHAT WOULD WE DO DIFFERENTLY IF WE HAD TO DO ALT4 AGAIN?

We would've loved to make the process demonstrate further IoT abilities such as making a mobile application (or even just a website that can work responsively on all devices) that can monitor data in real time whilst using AI to determine personalized results. On the workflow side we would've wanted to approach our design more carefully so that we would not waste any time on any complications (as we would've predicted ahead because of planning). However, we both believe that nothing could replace the amount of fun and learning we had as a duo together. Sure, overcomplicating and innovating new ideas and performing the work differently would've produced more results but as we've mentioned we both think that at the end of the day we both had fun learning and making issues, because how can you learn if you don't make mistakes?

ARE THERE ANY KNOWN ISSUES/BUGS IN YOUR FINAL PRODUCT?

Yes, like all products we identified issues that we simply didn't have time or resources to solve.

ISSUE	STATUS
0x6B LCD (backup of OLED) not working – Address @ L:13	Not resolved – Backlight requires its resistor to be soldered
0x3C OLED has screen jitter	Not resolved – No root cause found
SDA SCL Transmission sometimes bugs out and gets unwanted noise	Not resolved – No cause found, hypothesized as a wiring issue
Heartbeat (not SpO2 though) picks up -1 even with finger on the sensor	Not resolved – Issue is hardware related and is out of our control
MAX30102 doesn't work	Resolved – Proper library found

ISSUE #1 – INFO/HYPOTHESIS

Our backup display (LCD) was not working because the board requires its resistor to be soldered. We were going to attempt to jump the connection however we would need a very small jumper wire. We hypothesized that the LCD has the resistor not soldered for safety reasons. The LCD did function correctly with the library we had found on the Arduino library registry, however it was virtually impossible (breaking rules of universal design) to even see the text because of no backlight present.

ISSUE #2 – INFO/HYPOTHESIS

The 0x3C primary OLED display we had used tends to have a very subtle jitter. It appears to be most noticeable when the "loading" screen occurs and switches to main display. We hypothesized that this may occur for one of the two reasons. #1 is that the screen simply just can't process a large amount of pixel change in such short time, hence causing a jitter. #2 It is possible that there was noise between the SDA or SCL rails which caused latency in receiving data to the screen, hence causing a subtle jitter.

ISSUE #3 - INFO/HYPOTHESIS

We noticed during I2C scanning that sometimes the SDA and SCL would receive unwanted noise sometimes, although small and probably doesn't affect much we knew it had to be documented. We hypothesize that this may (just like #2) just be a wiring issue with the SDA and SCL transmission rails.

ISSUE #4 - INFO/HYPOTHESIS

Sometimes there is a long delay in the MAX30102 to pick up the HR even after several seconds of the finger being placed onto the sensor. This would cause the display to state "-1" on the HR for the duration. We know that this isn't a matter of "delayed data" as the sensor within less than a second goes to "-1" upon the finger is no longer in contact. We hypothesized that this issue occurs because whilst the finger is not placed on the sensor, it adapts to the environmental lighting, however upon contact it must re-adapt before processing results to avoid unsolicited data that is not accurate. We believe this may be the root cause and is what's causing the slight delay.

[S-05d] : EVALUATE – PERSONAL STATEMENTS

WAS THERE ANY LESSONS LEARNT FROM ALT4?

Absolutely. There were both positive and negative lessons that we both will uniquely bring forward in the future in our careers and for the remainder of the Computer Science Course. First, we both successfully achieved the four learning outcomes in understanding and applying the knowledge further on to develop an innovative solution for our design statement. We both learnt a lot of mistakes we made down the road such as the MAX30102 incompatibility with the Microbit. However, these are mistakes that we've taken as learning opportunities. On a further level with regards to working as a team we learnt how to effectively coordinate and plan out solutions in a limited timeframe, whilst also under a "disadvantage" due to only having two team members. One of the biggest lessons we both decided that should be mentioned here is that we learnt how to use an embedded system and work with some of the coolest things such as breadboard, sensors etc.

WHAT LESSONS WILL YOU CARRY FORWARD? (ADAM)

When I first started designing I never thought the need for redundancy was ever important however this one time I decided to implement that methodology into the project (where we would have multiple layers of backup to help us achieve the solution in the event anything happens such as issues, or unforeseen events), and I am glad we decided to do that at the very start because had we not, we probably wouldn't have a working product to show for. I also got to learn the very delicate ecosystem of embedded systems and how single misplaced digits of memory can cause a sensor to not work, and how each sensor has its own respective libraries, and I really found that interesting especially with the fact it was on C++. These are some important lessons I will bring forward with my career where you cannot afford any mistakes. My most interesting lesson? It was probably the documentation part, looking at it know I never really delved into the fact that everything is documented to the smallest of details, and I think it is something I'm going to have to learn further for my future. Working with Conor was an absolute pleasure and was an amazing person to work with throughout the project and I am thankful for Mr. Allen's team selection process because I wouldn't rather anyone else for a project especially when we are one man short.

WHAT LESSONS WILL YOU CARRY FORWARD? (CONOR)

ALT4 has thought me so much. To start, this was my first time connecting an external device to a microprocessor. Coming from a background where I didn't code growing up, being completely new to this subject, this was very interesting to learn. I had never seen or heard of a breadboard before and ALT4 had me wiring a MAX30102 sensor to an Arduino through a breadboard and jump wires. While I had to learn to wire through various online videos, it was surprisingly easier than I had initially anticipated. While we didn't get to use the Micro:Bit and IoT:Bit in the end, this project helped me learn the capabilities and boundaries of the BBC Micro:Bit and to my surprise, its capabilities were astonishing for such a cheap piece of equipment. We used extensions and learning about the hundreds of available extensions and advance setting amazed me. It really baffled me how capable it is, and it has since influenced me to purchase a Micro:bit for my own personal use. I found the Micro:bit much easier to learn than the Adruino, which frankly I still don't fully understand. However, despite these capabilities the Micro:bit could not successfully connect to the MAX30102, so it does have its limits, especially when it comes to voltage. I think that ALT4 has enhanced my knowledge of coding and how different aspects work and it has undoubtedly enhanced my learning and had made me much more confident in the Computer Science classroom. Working with an experienced and extremely advanced coder in Adam also was very interesting and fun. He helped me to learn so much throughout the project while he also put his trust in me on numerous occasions. Working with him was an absolute pleasure and I really enjoyed it. The biggest lesson I have learned however is not to panic. While time was against us and we couldn't connect the Micro:Bit, we stayed resilient and ended up getting our project to successfully function.

/ THE END

WRITTEN BY *ADAM BELL*

CONOR DUNNE

THIS PROJECT TOOK 31 DAYS TO COMPLETE AND WAS DESIGNED BY TWO INDIVIDUALS.

CUSTOM GRAPHICS WERE DESIGNED BY ADAM BELL USING FIGMA, PYTHON, AND OBISIDIAN NOTES.

THANK YOU FOR READING OUR PROJECT REPORT, *HAVE A GOOD DAY*