**RUNNING WATCH**

5/12/2022

Embedded Systems Design

Final Project Report  
Kate Kilpatrick, Aaron Mendoza, Seth Shively

A hand holding a circuit board

Description automatically generated with low confidence

**Project Summary**

This device is a simple running watch that outputs the heart rate, distance, and time to the face of the watch which is an LCD screen on an Atmel SAM4L8 Xplained Pro board.

This device used two sensors; one to measure heart rate and one to measure position. Both sensors communicated to the Atmel board via I2C and was converted from bytes into usable data through software written in C using Atmel Studio. The Atmel board itself allows the user to toggle between what the LCD displays by reading input from an external button. Two more buttons were connected to the board and used to start, stop, and pause a run. The main design lies within the software, where GPIO interrupts determine which state the watch is in to change what data is displayed and read by the Atmel board.

The overall design of the watch is to be used for portability and to keep track of the current statistics of a run. The wrist strap is used to keep all components compact. The watch is battery operated for it to function for multiple uses and portability.

**Project Objectives**

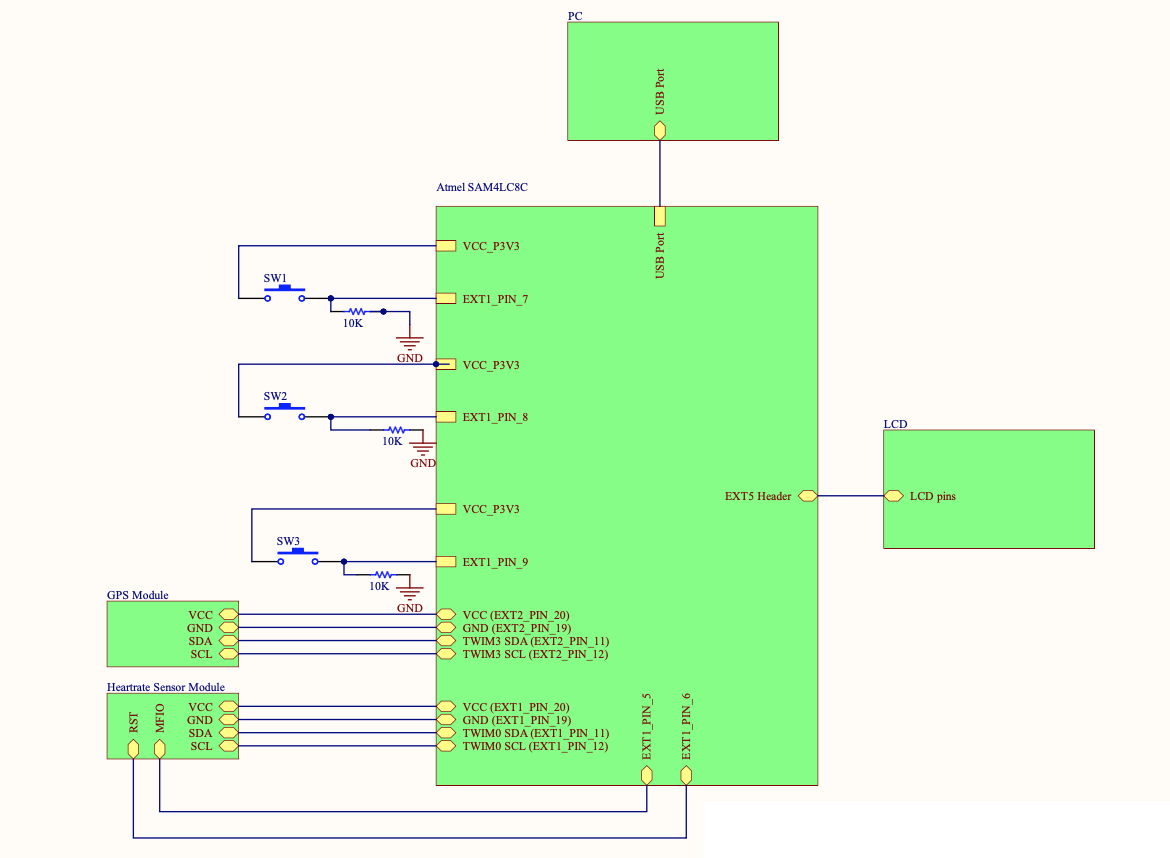
1. Collect data: The sensors measure and communicate the heart rate data and GPS data.
2. Calculate appropriate data: The data is converted from bytes into usable data such as beats per minute for heart rate and latitude/longitude from the GPS.
3. Display: The LCD screen displays data such as time, current heart rate, and distance in miles.
4. Portable: The device will be strapped onto the user and battery operated to be easily reused in any location.

**Selected Hardware Platform**

The SAM4L8 Xplained Pro – ATSAM4LC8C was chosen as the hardware platform because of its short bootup time, I2C capabilities, built-in LCD screen, and ability to use GPIO interrupts which allows switching between states quickly. These capabilities make this board the best fit for the project objectives.

**Hardware Design Description**

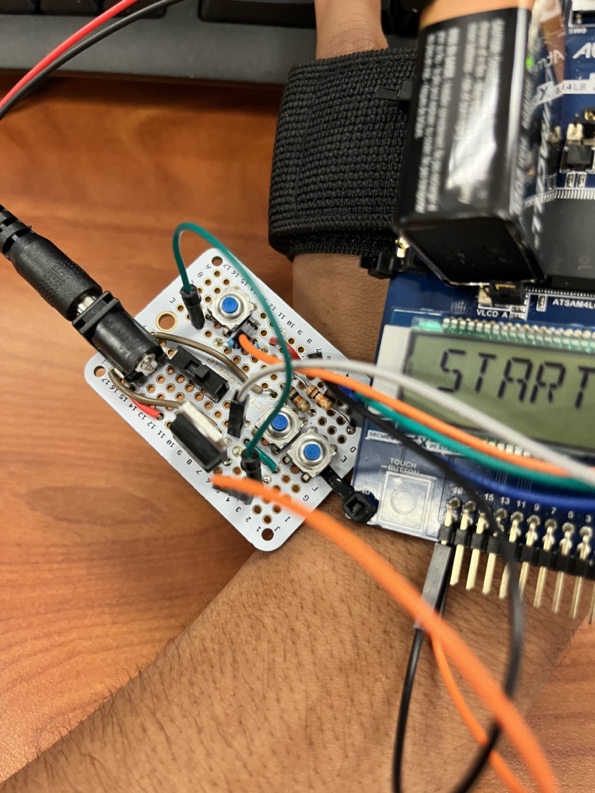
Figure 1 below shows the hardware design of the running watch.

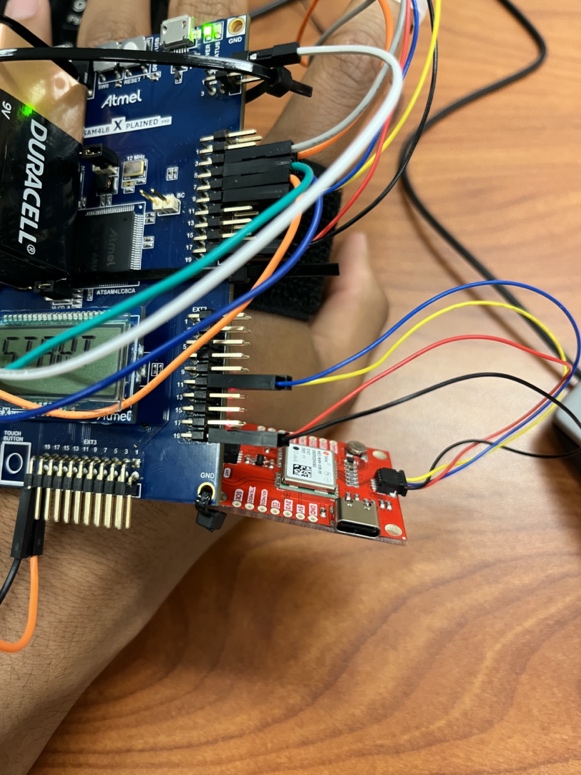


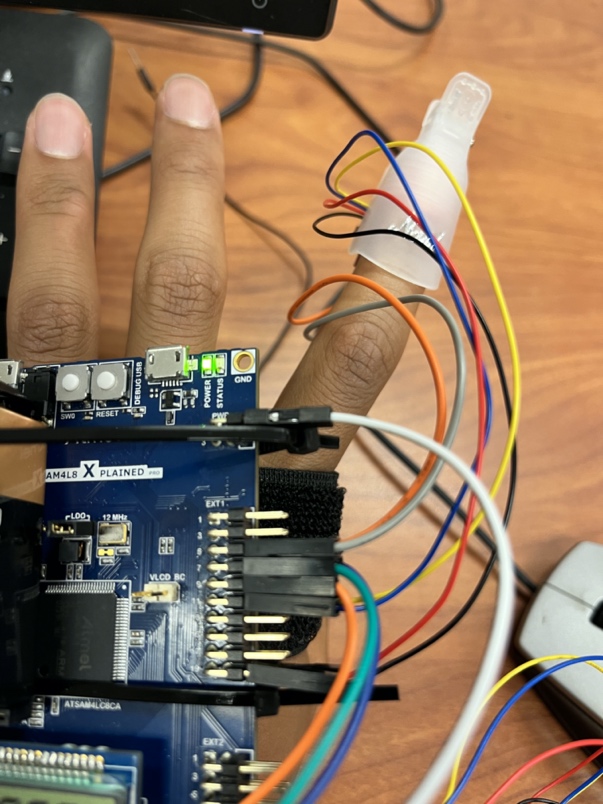
*Figure 1: Watch Schematic*

There are two sensors used in this project: the SparkFun Pulse Oximeter and Heart Rate Sensor and the SparkFun GPS Breakout NEO-M9N which provide heart rate data and location data respectively. Both sensors were powered by 3.3V from extension header pins on the Atmel board, with the heart rate sensor on the Extension Header 1 pins (11, 12, 19, 20) and the GPS module on the Extension Header 2 pins (11, 12, 19, 20).

To power the Atmel board, a 9V battery was used along with a male DC plug connected to a barrel jack that was soldered onto a protoboard to work with a voltage regulator that output 5V to the board. A switch controlled whether or not the board received power. All buttons were soldered onto the protoboard and were all connected to pins on extension header 1.

  
*Figure 2: Soldered Circuit with Three Buttons*

  
*Figure 3: Heart Rate Sensor*

  
*Figure 4: Finger Clip*

A brief summary of each hardware component is included below, along with part specifications:

1. *SAM4L8 Xplained Pro – ATSAM4LC8C*
   * Microcontroller supported by the Atmel Studio software for programming and implementation
   * Built-in GPIO pins
   * Multiple SDA/SDL pins for reading in sensor data
   * Powered by 5V external source
   * Supplies 3.3V
2. *Heart Rate Sensor – SparkFun Pulse Oximeter and Heart Rate Sensor, MAX30101 and MAX32664*

* Measures heart rate and pulse oximetry
* 3.3V VCC
* 1” x 0.5” dimension
* Utilizes two chips from MAXIM integrated
  + MAX30101 – Pulse oximetery and heart rate module to sense data
  + MAX32664 – Biometric sensor hub that handles all algorithmic calculations with a Cortex M4 processor

1. *GPS Sensor – SparkFun GPS Breakout NEO-M9N*
   * Measures latitude and longitude with approximately 1.5 meter accuracy
   * Communicates via I2C
   * 3.3V VCC and approximately 31mA consumption
   * 92 channel u-blox M9 engine GNSS receiver
     + Allows it to receive signals from multiple satellite constellations including GPS,GLONASS, Galilieo, and BeiDou
2. *Voltage Regulator - L7805CV*

* Voltage regulator that can take input up to 24V and output up to 24V
* Used to power Atmel board from a 9V battery

1. *Switch*

* Used to power Atmel board

1. *Resistors— 10K Resistors*

* Resistors are connected from the buttons to ground
* Resistors help with preventing any power surging to the ground of the Atmel board

1. *External Buttons*

* Used to toggle between watch states

*8. Protoboard*

* Holds all soldered connections

*9. Velcro Arm Strap*

* Holds all components via zip ties

*10. Qwiic connectors*

* Used to easily access power, ground, SDA, and SCL pins for each sensor

**Selected Software Platform**

Atmel Studio was the selected software platform. The programming language used in this project was C. This platform was chosen because it allows the Atmel board functionality to be easily modified using ASF modules and uploading a program to the board is efficient using a USB connector. Atmel Studio also comes with a built-in debugger that was used to find problems when toggling between states.

For additional debugging, a Digilent Analog Discovery 2 was used along with Waveforms to observe I2C communications. PuTTy was also used to observe output.

**Software Design**

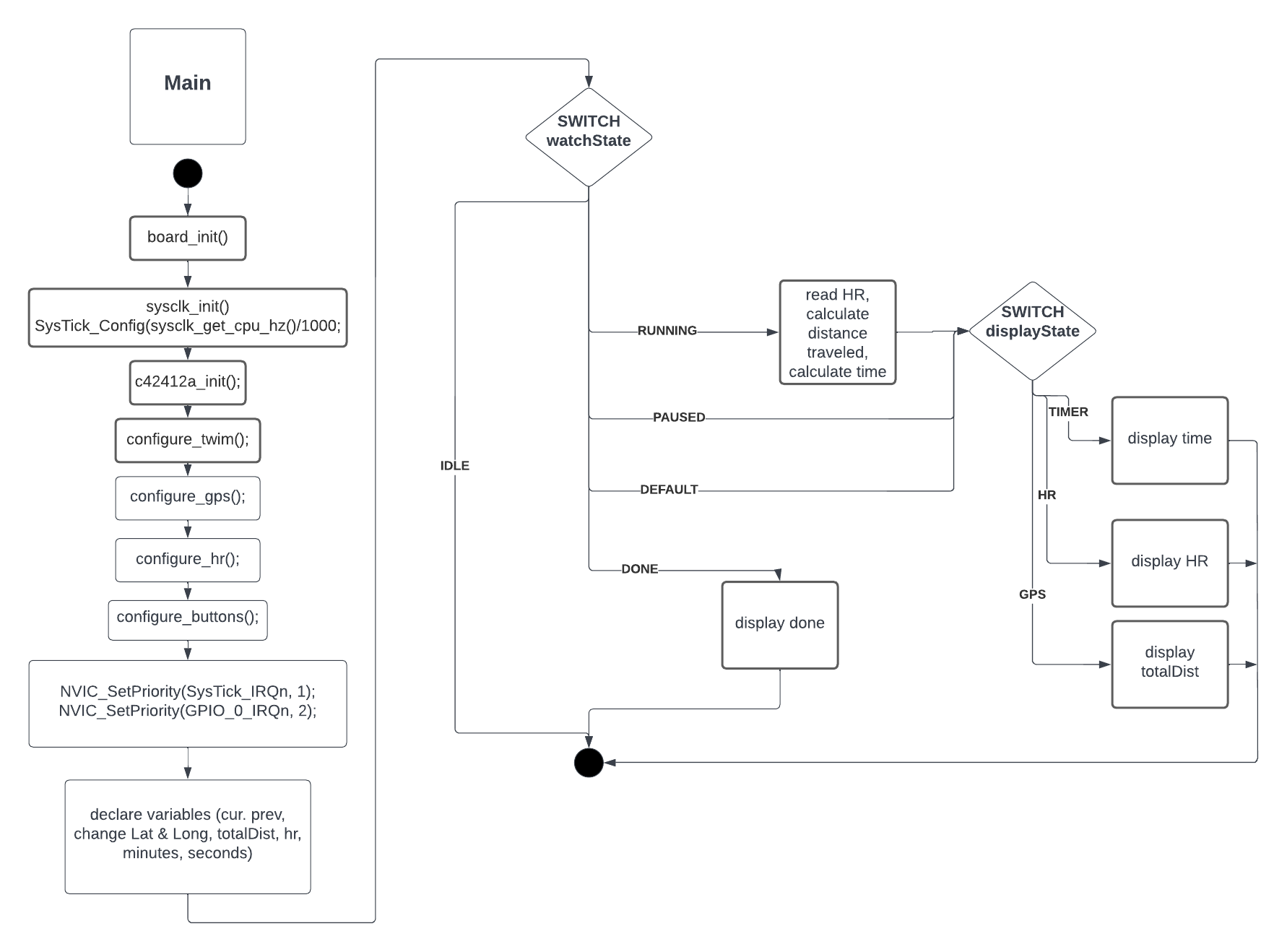
*Overall*

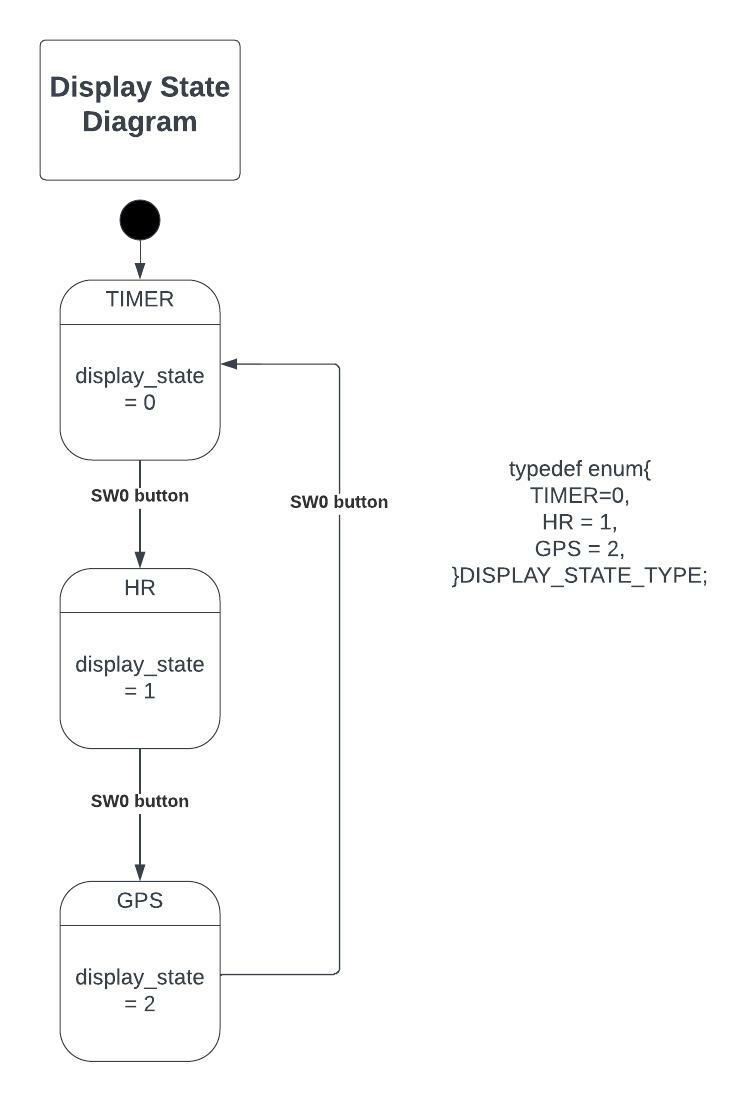
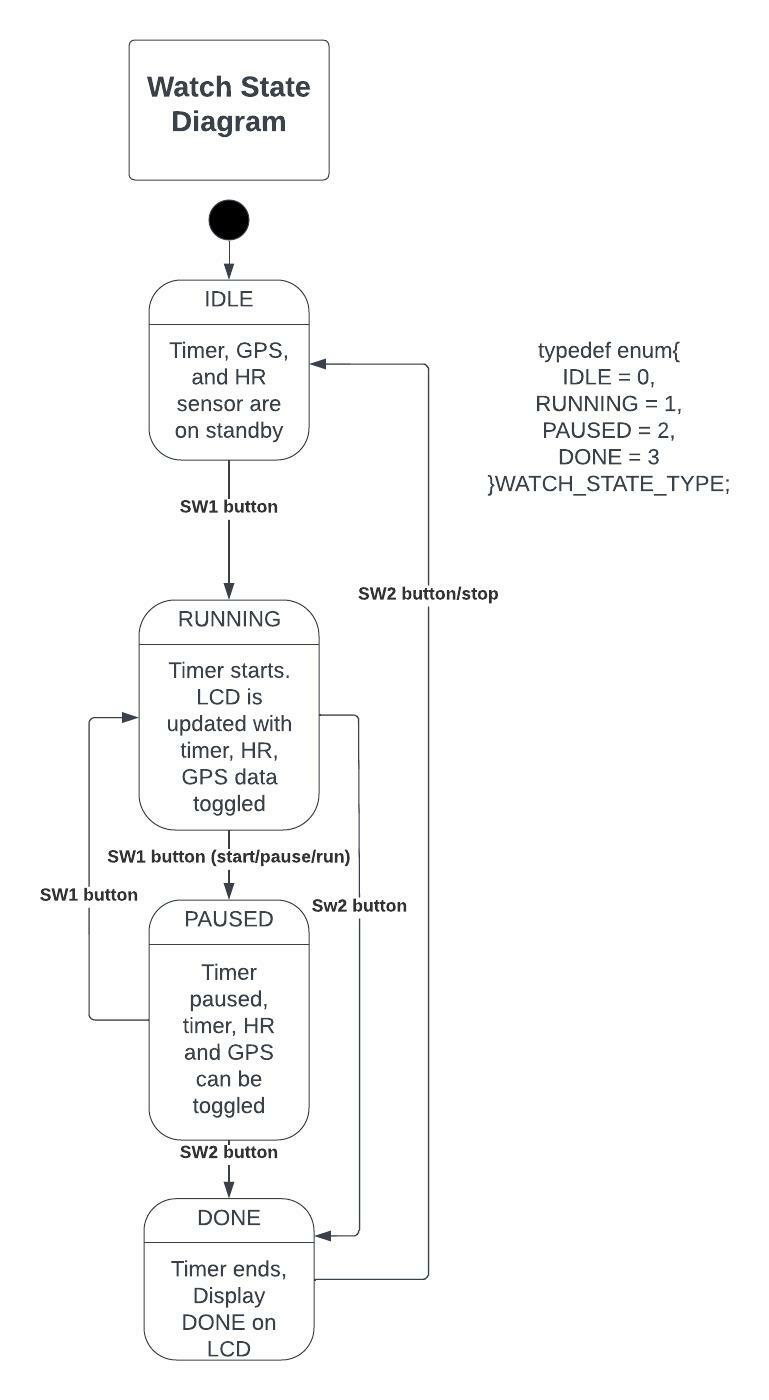
The Atmel project should be made specifically for the SAM4LC8C board. It includes multiple ASF modules that allow key functionalities of the board such as a time delay, I2C communications, and LCD display. There is more information on the specific ASF modules needed under the section **Software Setup Procedure**.

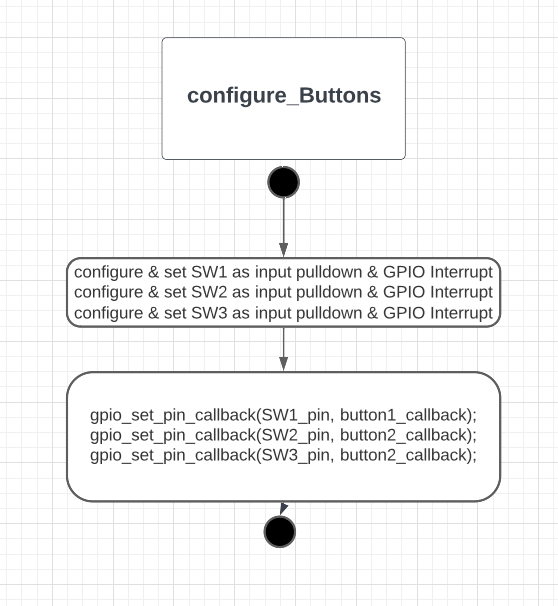
The main driver initializes the Atmel board to work properly with the sensors, buttons, and LCD display. It configures the buttons by calling predefined GPIO functions and initializes the sensors by calling functions from separate header files. It also allows users to print to PuTTy if needed. This driver has switch case statements for various states that the watch toggles between based on user input.

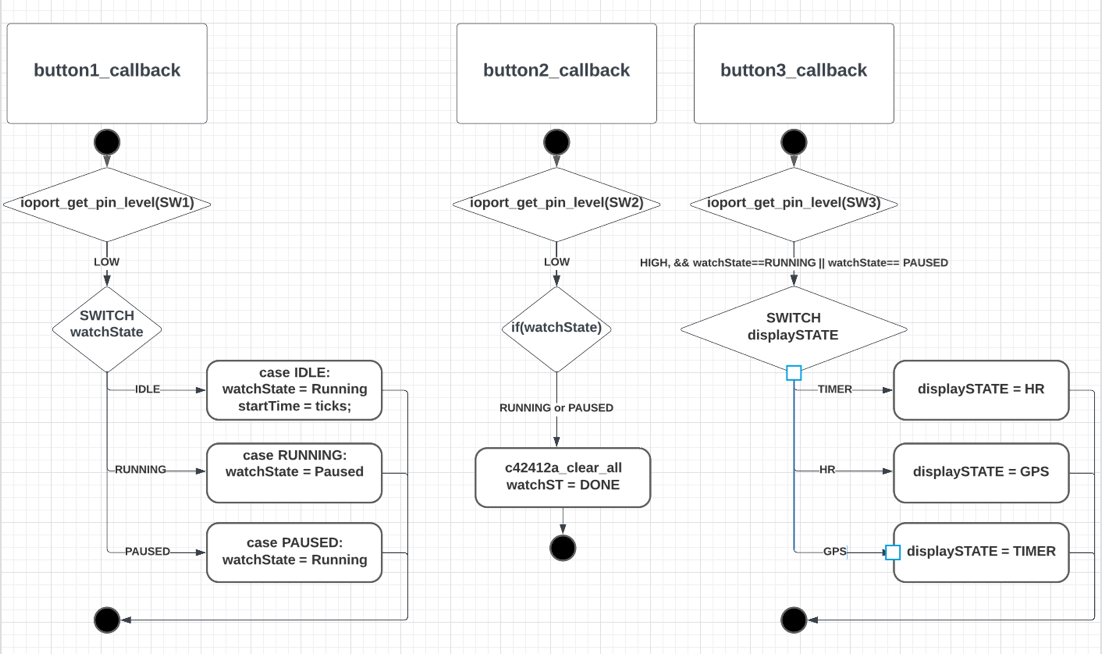
There are two main states that determine what the watch should be doing: watch state and display state. These states are controlled by GPIO interrupts from three buttons, two of which are external to the board while the last one is the SW0 button. The watch state changes between IDLE, RUNNING, PAUSED, and DONE. One external button (SW1) starts a workout (IDLE to RUNNING) or ends a workout (RUNNING to DONE or PAUSED to DONE) while the other button (SW2) pauses or continues a workout in progress (RUNNING to PAUSED or PAUSED to RUNNING). The display state is controlled by the SW0 button. The LCD screen can display the current heart rate in beats per minute, the total distance ran, and the current time. These states determine whether or not the Atmel board is reading from the sensors.

*State Machine Activity Diagrams*

*Figure 5: Main Activity Diagram*

*Figure 6: Watch and Display State Diagrams*

  
*Figure 7: Configure Buttons Activity Diagram*

  
*Figure 8: Callback Activity Diagram*

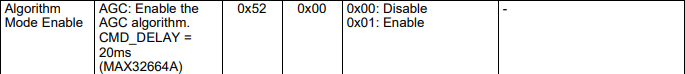
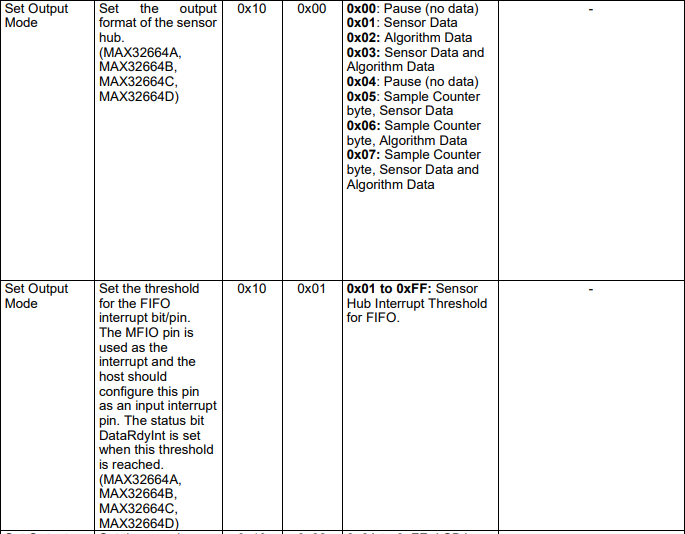
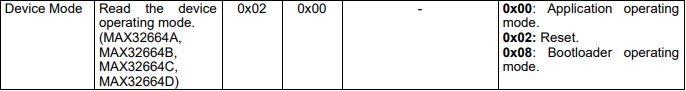
*I2C Communications*

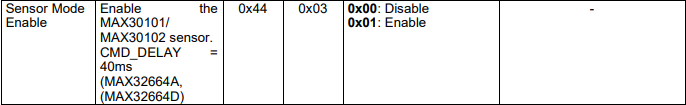
The heart rate sensor and the GPS module communicate via I2C. The functions required to do this lie within the utilities\_gps and utilites\_hr header files. These functions transmit and receive bytes that determine which sensor the Atmel board is talking to along with actual bytes to perform certain tasks such as initialization. These files are included in main to be used at a high level. Within the Atmel board, we used the TWIM0 module for the heart rate sensor messages and the TWIM3 module for the GPS sensor messages.

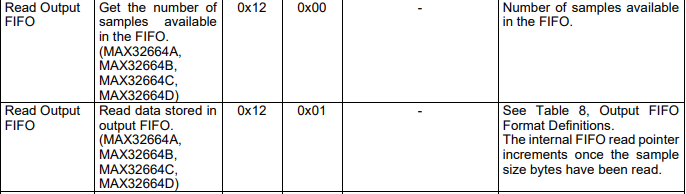
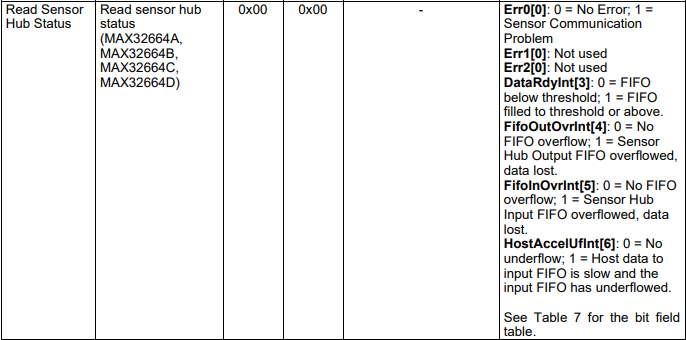
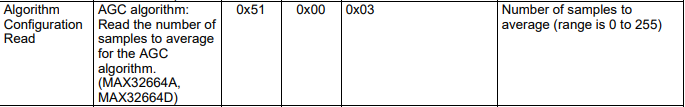
*Heart Rate Sensor ICD*

To ensure that the device is in application mode, the MFIO and RST pins on the sensor must have specific high/low values output to those pins via the Atmel board. More information on this can be found on page 17 of the [MAX32664 User Guide Version 3](https://pdfserv.maximintegrated.com/en/an/ug6806.pdf) or within **Software Setup Procedure**.

Each command for the heart rate sensor requires a write and a read. Each write has two bytes minimum: the family byte and the index byte. If necessary for the specific command, data bytes follow the index byte. The number of bytes read vary for each command. In between each read and write, there may be a specific delay (CMD\_DELAY) that is necessary to ensure correct functionality. If the command from the master is only writing to the sensor, then a status byte is returned. The meaning of these status bytes is shown on page 19 of the [MAX32664 User Guide Version 3](https://pdfserv.maximintegrated.com/en/an/ug6806.pdf). The 7-bit address for the sensor is 0x55. Below are the commands needed to initialize and read from the sensor, where the first hex number is the family byte, the second hex number is the index byte, the third hex number is the data byte, and the fourth hex number is what is returned by the sensor.





*Figure 9: Heart Rate ICD*

*GPS Module ICD*

FORMAT EXPLANATION

Single read message is of the following form:  
START, 0x85 [42, RD], B1, B2, B3, …, Bn , STOP  
0x85 is the combined 7-bit device address (0x42) and read bit (1). Bytes B1 through Bn are read by the master from the slave device.

Single write message is of the following form:  
START, 0x84 [42, WR], B1, B2, B3, …, Bn , STOP  
0x84 is the combined 7-bit device address (0x42) and write bit (0). Bytes B1 through Bn are written from the master to the slave device.

Write-read combination message is of the following format:  
START, 0x84 [42, WR], Breg, RESTART, 85 [42, RD], B1, B2, B3, …, Bn , STOP  
After START, one byte Breg is written by the master to the slave device, which contains the address of a specific register from which to read next. After RESTART, the master reads bytes B1 through Bn from the register at address Breg

MESSAGES SENT AND RECEIVED BY GPS UNIT:

*Initialization:*  
START, 0x84 [0x42, WR], STOP (write command with no contents, necessary for some reason)

Write the following 9 bytes to start initialization  
START, 0x84 [0x42, WR], 0xB5, 0x62, 0x06, 0x00, 0x01, 0x00, 0x00, 0x07, 0x21, STOP

Check how many data bytes are available, this value is found in the FD register. Should be equal to 0x26  
START, 0x84 [0x42, WR], 0xFD, RESTART, 0x85 [0x42, RD], 0x00, 0x26, STOP

Read 38 bytes, the value of these bytes is not used, however reading them seems to still be necessary for the unit to function properly  
START, 0x85 [0x42, RD], [32 bytes read], STOP  
START, 0x85 [0x42, RD], [6 bytes read], STOP

*Reading data:*

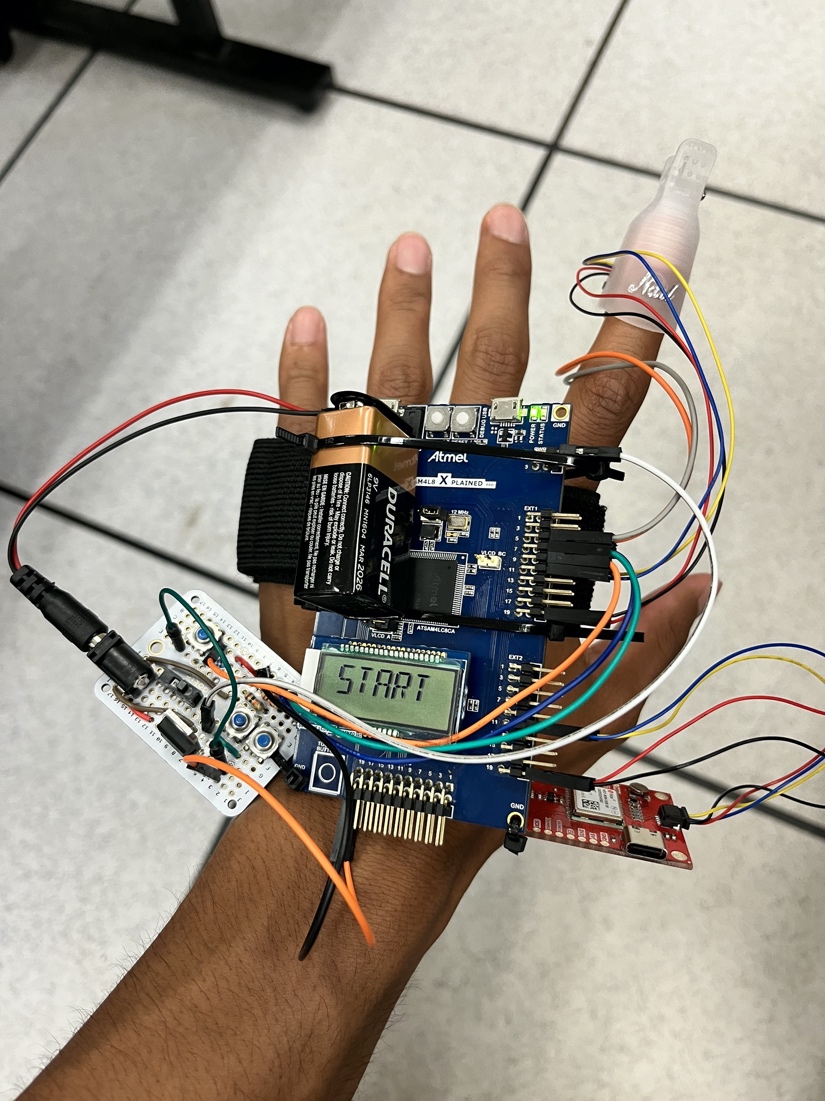
Write the following 8 bytes to tell unit to start collecting data:  
START, 0x84 [0x42, WR], 0xB5, 0x62, 0x01, 0x07, 0x00, 0x00, 0x08, 0x19, STOP

Run the following command to check how many bytes are available, this value is found in the register at address FD. Repeat this command until the second byte read is equal to 0x64 (100 decimal) or a timeout of 1 second is reached.  
START, 0x84 [0x42, WR], 0xFD, RESTART, 0x85 [0x42, RD], [2 bytes read], STOP

Run the following four read commands to read the full contents of the data available, equal to 100 bytes  
START, 0x85 [0x42, RD], [32 bytes read], STOP  
START, 0x85 [0x42, RD], [32 bytes read], STOP  
START, 0x85 [0x42, RD], [32 bytes read], STOP  
START, 0x85 [0x42, RD], [4 bytes read], STOP  
The 100 bytes read above contain all of the PVT data, from which the latitude and longitude can be extracted. If all bytes are concatenated into an array in the order in which they are read, longitude is represented by the four bytes starting at position 30, and latitude is the four bytes starting at position 34.

**Test Results**

The project successfully reads data from both the heart rate and GPS sensor. The interrupts from both the external buttons are able to correctly transition between states for full functionality of the watch. The timer within the watch is accurate to give the user more insight into their workout. On the watch itself, the LCD display shows current data as designed. For power, the 9V battery was able to be used to drive the Atmel board through the voltage regulator. Therefore, all project objectives were met.

  
*Figure 10: Overall Watch Picture*

**Lessons Learned**

1. *Unnecessary usage of FreeRTOS:* Initially, our design implemented FreeRTOS, a real time operating system that can be used with the Atmel board. Rather than having the buttons interrupt, the original idea was to have multiple tasks that continuously ran and switched between each other. This proved to be a timing issue. Each of the different group members worked on their own individually task, but when we started implementing the tasks together, the timing for each of the tasks had conflicts. The GPS sensor would take an unpredictable amount of time on each run. We learned that using a real time scheduler is only practical if the time of each task is known and that switching between the tasks at hard deadlines is important.
2. *Documentation can be misleading:* Although the datasheet for the heart rate sensor seemed to have everything needed for I2C capabilities, some commands that were included in examples were simply not necessary. Completely relying on the accuracy of documents can lead to wasted time.
3. *Reverse engineering can save time:* Utilizing the Digilent Analog Discovery 2 along with Waveforms to observe I2C communication with the sensors allowed us to see the necessary bytes the master must send. Rather than looking at the extensive datasheet of the GPS module, programming the example Arduino library onto an Arduino Uno board and spying on the bytes made translating it into code efficient and easy.
4. *Unit testing is important:* Within each step of the project, unit testing was crucial to make sure each part was working correctly. This is important in both software and hardware. For software, it makes sure that each function produces the right behavior. For hardware, it ensures that the correct voltages are being output. Without unit testing, debugging and solving issues is harder because the source of the problem is more difficult to find.
5. *Separate TWIM Modules can make a difference:* When using the same TWIM module for both sensors, incorrect messages were sent. We believe this is due to hidden pullup resistors on the sensors that pull the SDA/SCL lines high at times that interfere with other sensors. By moving one sensor to a separate TWIM module on the board, we were able to use the battery as a power source rather than rely on plugging it into a PC with Atmel Studio installed.

**Hardware Setup Procedure**

1. Follow the schematic shown in Figure 1 and solder all necessary components onto the protoboard.
2. Tape or zip tie HR sensor to inside of nail clip.
3. Connect the sensors to the correct Atmel board pins on the schematic using the Qwiic connectors. Note that yellow is SCL and blue is SDA.
4. Zip tie the protoboard to the bottom of the Atmel board using the holes on each board.
5. Zip tie the Atmel board to the arm band.

**Software Setup Procedure**

*Creating the Project*

1. Select the SAM4LC8C when creating the project on Atmel studio.
2. Include the following ASF modules by using the ASF wizard:
   1. Generic board support
   2. System Clock Control
   3. IOPORT – General purpose I/O service
   4. Standard serial I/O
   5. GPIO – General-Purpose Input/Output
   6. TWI – Two-Wire Master Interface
   7. WDT – Watchdog Timer
   8. sam.components.display.c42412a
3. Enable com port on conf\_board.h.
   1. #define CONF\_BOARD\_COM\_PORT
4. Download conf\_clock.h or manually uncomment the necessary variables.
5. Download main.c.
6. Download utilities.h.
7. Download common.h.

*Heart Rate Sensor Code*

1. Enable TWIM functionality within Atmel Studio by adding the TWI master interface ASF module and defining the desired TWIM module in conf\_board.h.
2. Download utilities\_hr.h.
3. Configure MFIO and RST pins as inputs.
   1. Define what pins on the external headers will be used for MFIO and RST.
   2. Set the RST to low and MFIO pin to high for 10 ms.
   3. After 10 ms, set the RST pin to high.
   4. Wait for 1 second.
   5. The board should be in application mode after these commands.
4. Create a TWIM config structure to be used to hold transmitted and received data.
5. Call necessary functions within utilites\_hr.h that are described in the heart rate sensor ICD.

*GPS Sensor Code*

1. Enable TWIM functionality within Atmel Studio by adding the TWI master interface ASF module and defining the desired TWIM module in conf\_board.h.
2. Create a TWIM config structure to be used to hold transmitted and received data.
3. Call necessary functions within utilites\_gps.h that are described in the GPS ICD.

**Device Usage Procedure**

1. Place the device on your desired forearm and put finger into heart rate clip.
2. Flip switch on protoboard to ON to power the board.
3. Reset the watch if necessary by pressing the RESET button on the watch, which is located beside the SW0 button.
4. Wait until the device will displays “START.”
5. To begin a workout, press the START/STOP button.
6. To pause a workout, press the PAUSE/CONTINUE button.
7. While a workout is in progress, press the SW0 button on the watch to toggle between what data is being displayed.
8. To end a workout, press the START/STOP button.
9. To start another run, press the reset button.
10. Flip switch on protoboard to OFF to power the board.