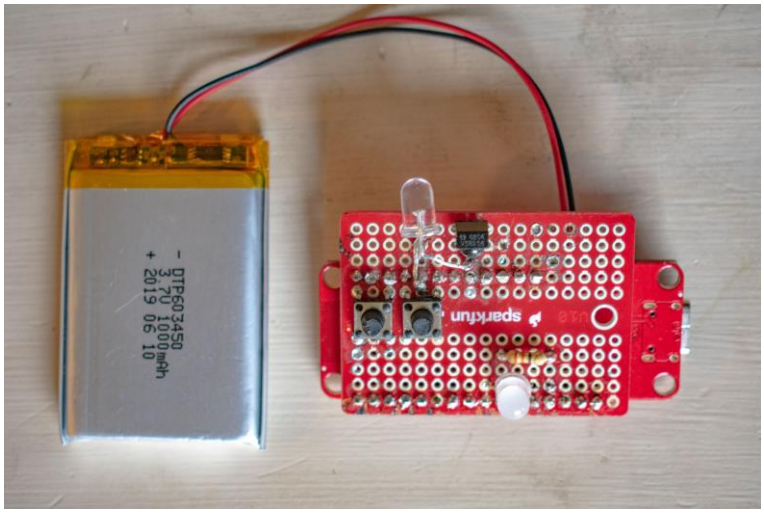
Fall 2019 Final Project

Merced College
CPSC-42
Computer Architecture and Organization



Sparkfun Artemis Thing Plus Camera Remote and Intervalometer

Kendall Helms

December 3, 2019

Abstract

This report details the assembly and programming of a trainable infrared remote with intervalometer functionality. This remote can be used with any camera which can be triggered by an infrared remote. It was built using Sparkfun's Artemis Thing Plus board, which is based on the Ambiq Micro Apollo3 Blue controller chip.

Table of Contents

1. Introduction

This project was undertaken to exhibit three main functionalities:

- Trigger a camera to take a photo via infrared signaling
- Function as an intervalometer, taking a set number of photos over a set period of time
- To be capable of being trained to output any camera's infrared signal trigger

While these three functions are not so impressive in effect, due in part to the choice of hardware, their implementation in software is fairly complicated, and difficult to sequence.

2. Building the Project

2.1 Parts:

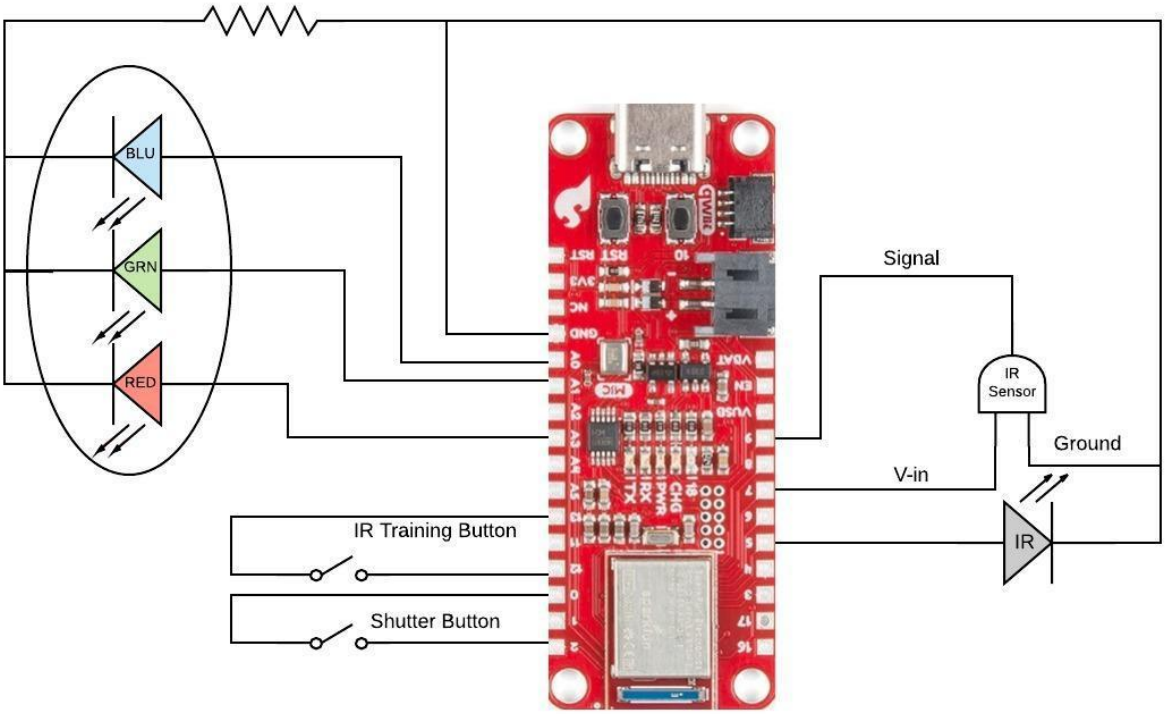
The project uses a fairly short and simple hardware parts list:

1. Sparkfun Artemis Thing Plus
2. Solderable mini breadboard
3. Solderable male pin headers
4. Infrared sensor
5. Infrared LED
6. 2 x Buttons
7. RGB LED
8. 1 x resistor
9. 1 x Jumper wire

2.2 Assembly:

In order to create a small form factor for the finished product, the components for this project are soldered directly onto the mini breadboard, which is then soldered to the Artemis Thing Plus in a sandwich configuration, using the pin headers listed above. The jumper wire is used to ground both the infrared sensor and infrared LED.

2.2.1 Circuit Diagram



3. Programming

The choice of the Artemis Thing Plus over an Arduino was made to accommodate future features which will utilize Bluetooth communication, as well as its small form factor. This choice did turn out to add an extra layer of difficulty to the project, as there is as of yet no available code library for implementing infrared communication for this board. Being that the Ambiq Micro Apollo3 controller chip used on this board differs from the ATmega chips used on Arduinos, the commonly used IRremote library, written by Ken Shirrif, is unfortunately incompatible. This meant that in order to make the project function as desired, two main programming goals needed to be accomplished:

- Implement functionality to read, parse, and store incoming infrared signals
- Implement functionality to output the stored infrared signals with proper timing through the infrared LED

Every feature of the project relies on the completion of these two goals.

3.1 Infrared Signal Training

Step one to creating an infrared remote, is to have stored a signal to output. The project stores this signal code through a set of functions in a namespace titled “recv.” The functions in recv use interrupts to store the number, length, and on-state of infrared signal pulses received by the infrared sensor. They then detect the repeating sequence of the signal and store that to be used for output. *The flow chart on Appendix i describes the sequence of these operations.*

The heart of this feature is the two interrupt service routines (ISRs), which are triggered by the rising and falling of the output of the IR sensor. *The recv::hi ISR implementation can be found on Appendix ii.*

This function is modified from code found on Ben Ripley's blog, in which he describes ways to read a PWM signal using an Arduino. It is called when the output of the infrared sensor rises from low to high. It records the time since the previous interrupt, and stores this information into an array named "raw_code." A nearly identical function does the same when the sensor output falls from high to low. These two functions ping-pong back and forth until raw_code has been filled.

3.2 Infrared Signal Output

Outputting the signal stored by the previous set of functions is made particularly difficult by this choice of controller. The sticking point is that most infrared communication is done at a 38 kHz duty cycle. This means that when it is on, the infrared LED will blink on and off at 38 kHz. Without use of the popular IRremote library, instructing the controller chip to do this requires programming its internal timers through Ambiq Micro's API. Conveniently, in order to make this system accessible to Arduino users, Sparkfun made it possible to program this chip using Ambiq API functions within the Arduino environment.

The functions written for this purpose initialize the proper timer for the infrared LED's pin, and set it to clock at 187.5 KHz. It then sets an interrupt to create a 5-pulse duty cycle. This means that the LED will blink once for every 5 pulses of the timer's clock. 187.5 kHz divided by 5 works out to 37.5 kHz, which turns out to be near enough in practice to 38 kHz. *The timer setup implementation is shown on Appendix iii.*

This code was largely appropriated from an example found within the Ambiq API, but was altered to be used in this context.

Now with the duty cycle implemented, the timer is pulsed on and off with delays between in accordance with the signal which was recorded by the infrared signal training function in the previous section. This is shown below.

3.2.2 Shutter Function

```
451  //
452  //  < Key >
453  //    Calls @pwm::start, @pwm::stop and @delay in order to output
454  //    the code stored in @key_code through the IR LED
455  //
456  void sht::key ()
457  {
458    for (int i = 1; i < key_code_size; i++){
459      !key_code[i][1] ? pwm::start() : pwm::stop();
460
461      sht::delay(key_code[i][0]);
462    }
463
464    interval::shutter = true;
465    sht::rpt();
466  }
467
```

This and the signal training functions accomplish the two main goals listed in the introduction to the programming section.

3.3 Intervalometer

An intervalometer is a piece of photography equipment which can trigger a camera to take a set number of photos over a set period of time. For example, if an intervalometer were set to take 5 photos over 5 minutes, it would trigger the camera once every minute until the 5 minutes were up.

This is useful for time-lapse photography/videography, which shows very slow movement sped up so that it can be seen by us.

This is simple to implement. Two functions, triggered by button interrupts count the number of photos to be taken, and the time for which the intervalometer will be set. After these processes are complete, the photo interval is calculated (photos divided by time), and the intervalometer starts taking photos.

It works by calling the shutter function shown above, and then waiting for the length of the interval. It then decrements the number of remaining photos, and calls itself recursively. The intervalometer implementation can be found on Appendix iv.

3.4 Two-Button Interface Handling

The final challenge of this project is brought about by another hardware decision. The use of two buttons to accomplish a fairly wide variety of functions requires careful sequencing and debouncing in order to prevent an unwanted function from being triggered.

The desired behavior is for one button to trigger the infrared signal training with a tap, and the other to trigger the shutter with a tap or hold. A tap of both buttons together should initialize the intervalometer, which repurposes the buttons for the shot count functions and the timer count functions. Either button should also be able to interrupt and cancel ongoing signal training and intervalometer functions.

Most of this behavior is implemented in the main loop using a button debounce function and a series of if / else if statements. The debounce function simply takes a button pin number as an input and returns true for a good input or false for a bad one.

The sequence of if /else if statements is important. Specifically, if the both-button-press input is not listed first in the loop, one of the other two inputs will often be mistakenly detected. When this statement is listed before the others, it prevents the possibility of any others being reached, even if they would be evaluated true. *The main loop code can be found on Appendix v.*

4. Future Ideas

To start, this project will need a power switch and a case in order to be practical for everyday use.

It should also be able to indicate battery charge level using the RGB LED indicator when the device's battery is being charged.

It will also be important to implement a way to store trained signals to non-volatile memory so that the device will not need to be retrained every time it is powered on.

This project will at some point have Bluetooth functionality which will allow all of its functions to be triggered from within a mobile phone application. This will improve ease of use, and allow other functions to be added from within the phone app. One such function might be to trigger the camera to shoot a photo based on the phone's geolocation in order to take action shots from an unattended camera.

5. Final Thoughts

In undertaking this project, I very much underestimated the difficulty I would have in implementing its most basic functionalities. The uninformed choice in hardware I made in going with the Artemis Thing Plus became a major setback in terms of time spent and difficulty. The benefit of this is that I have learned quite a bit about microcontrollers thanks to my mistake. This project has tugged away at the veil of simplicity that the Arduino brings to microcontroller programming, and made it clear how much more there is to learn to be able to create the kind of consumer electronics we use every day.

References

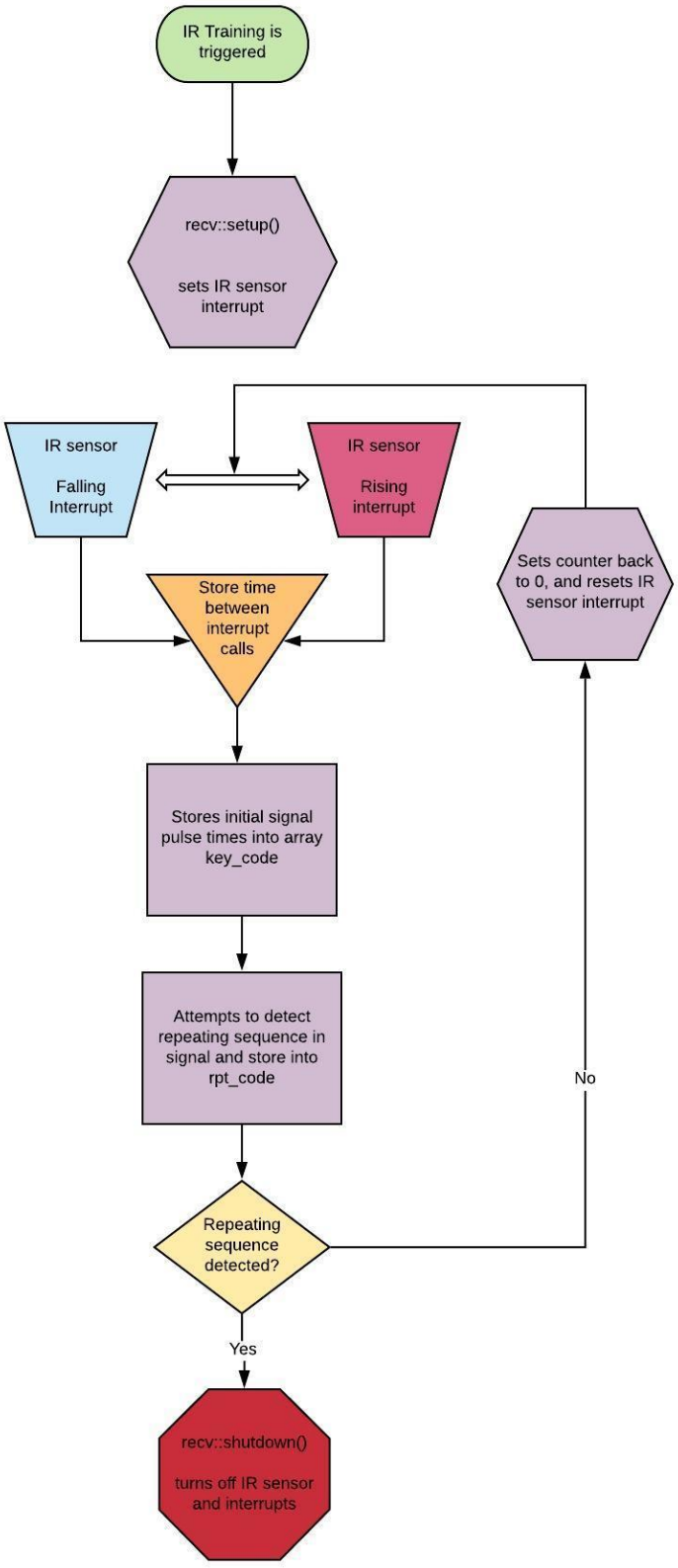
Ambiq Micro, Inc. Apollo3 Blue Datasheet. Ambiq Micro. Retrieved from https://support.ambiqmicro.com/hc/en-us/articles/360032906492-Apollo3-Blue-Plus-MCU-Datasheet

Ripley, B. (2014) Three Ways To Read A PWM Signal With Arduino. Benripley. Retrieved from https://www.benripley.com/diy/arduino/three-ways-to-read-a-pwm-signal-with-arduino/

Shirriff, K. (2012). Arduino IRremote. GitHub. Retrieved from https://z3t0https://learn.sparkfun.com/tutorials/ir-communication/all.github.io/Arduino-IRremote/

Sparkfun. IR Communication. Sparkfun. Retrieved from https://learn.sparkfun.com/tutorials/ir-communication/all

3.1.1 IR Signal Training Flow Chart



3.1.2 recv::hi ISR

```
252 //
253 //  << Hi >> ~Interrupt Service Routine~
254 //    Called by IR sensor rising interrupt.
255 //
256 //    Records time since previous IR sensor falling interrupt into current
257 //    index of the @raw_code array
258 //
259 //    Assigns Falling interrupt to IR sensor.
260 //
261 //    Deassigns interrupts from IR sensor, and calls @fill_key or @fill_rpt
262 //    when @raw_code is full.
263 //
264 void recv::hi()
265 {
266   attachInterrupt(digitalPinToInterrupt(IR_RECV_IN), recv::lo, FALLING);
267
268   recv_stop = micros() - recv_start;
269   recv_start = micros();
270
271   if(recv_count < raw_code_size) {
272     if(recv_stop > 0){
273       raw_code[recv_count][0] = recv_stop;
274       raw_code[recv_count][1] = 0;
275
276       recv_count++;
277     }
278   }
279
280   else {
281     detachInterrupt(digitalPinToInterrupt(IR_RECV_IN));
282
283     if(!key_code_full) recv::fill_key(); //Calls @fill_key if key is empty
284
285     else recv::fill_rpt();               //Calls @fill_rpt if key is full
286   }
287 }
```

3.2.1 PWM Timer Setup Code

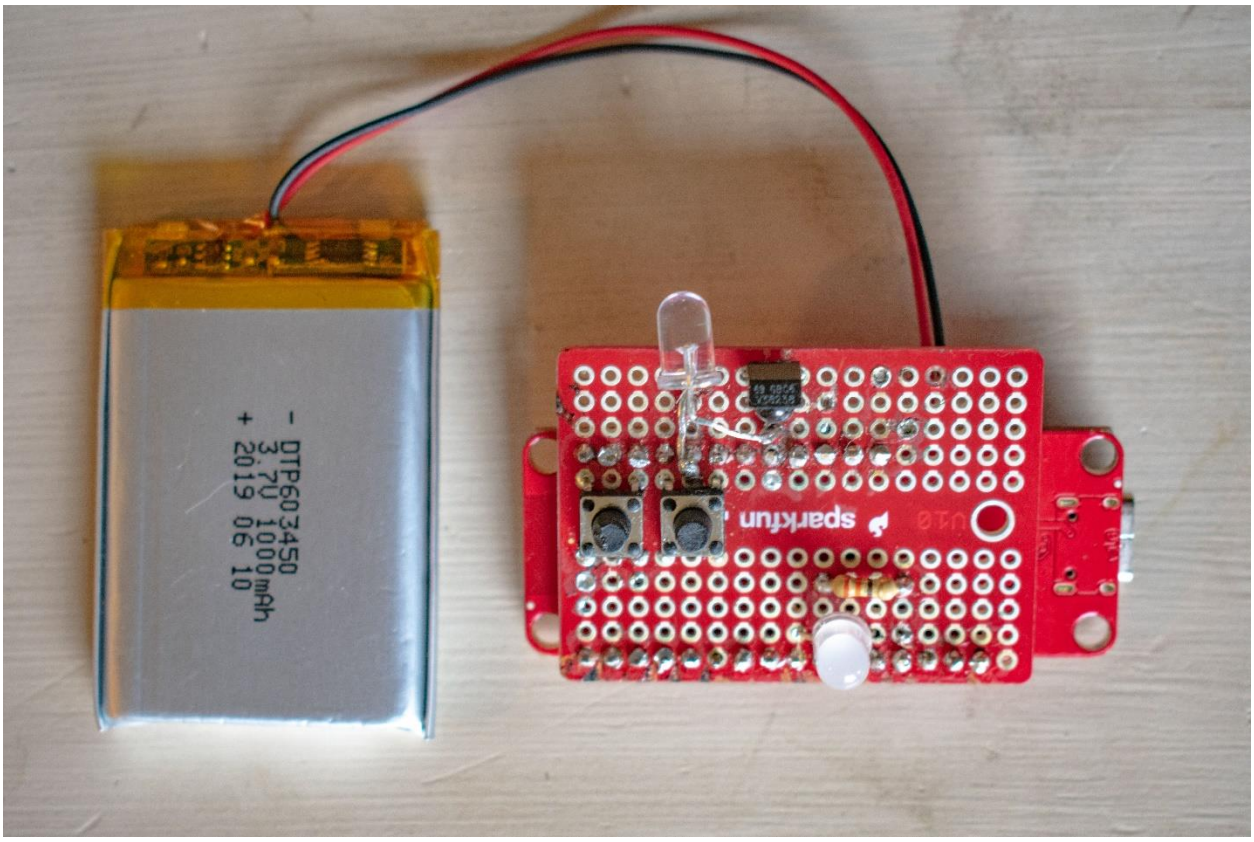
```
8  //
9  //  < Timer_setup >
10 //    Initializes timer and sets interrupts for PWM
11 //
12 void pwm::timer_setup ()
13 {
14   am_hal_clkgen_control(AM_HAL_CLKGEN_CONTROL_SYSCLK_MAX, 0);
15
16   am_bsp_low_power_init();
17
18   am_hal_clkgen_control(AM_HAL_CLKGEN_CONTROL_XTAL_START, 0);
19
20   //
21   // Configure the output pin.
22   //
23   am_hal_ctimer_output_config(IR_PWM_TIMER,
24                               IR_PWM_TIMER_SEG,
25                               IR_LED,
26                               AM_HAL_CTIMER_OUTPUT_NORMAL,
27                               AM_HAL_GPIO_PIN_DRIVESTRENGTH_12MA);
28   //
29   //Configure a timer to drive the LED.
30   //
31   am_hal_ctimer_config_single(IR_PWM_TIMER,
32                               IR_PWM_TIMER_SEG,
33                              (AM_HAL_CTIMER_FN_PWM_REPEAT    |
34                               PWM_CLK                        |
35                               AM_HAL_CTIMER_INT_ENABLE) );
36
37   //
38   // Set up initial timer periods.
39   //
40   am_hal_ctimer_period_set(IR_PWM_TIMER,
41                            IR_PWM_TIMER_SEG,
42                            PERIOD, ON_TIME);
43
44   //
45   // Enable interrupts for the Timer we are using on this board.
46   //
47   am_hal_ctimer_int_enable(IR_PWM_TIMER_INT);
48   NVIC_EnableIRQ(CTIMER_IRQn);
49   am_hal_interrupt_master_enable();
50 }
```

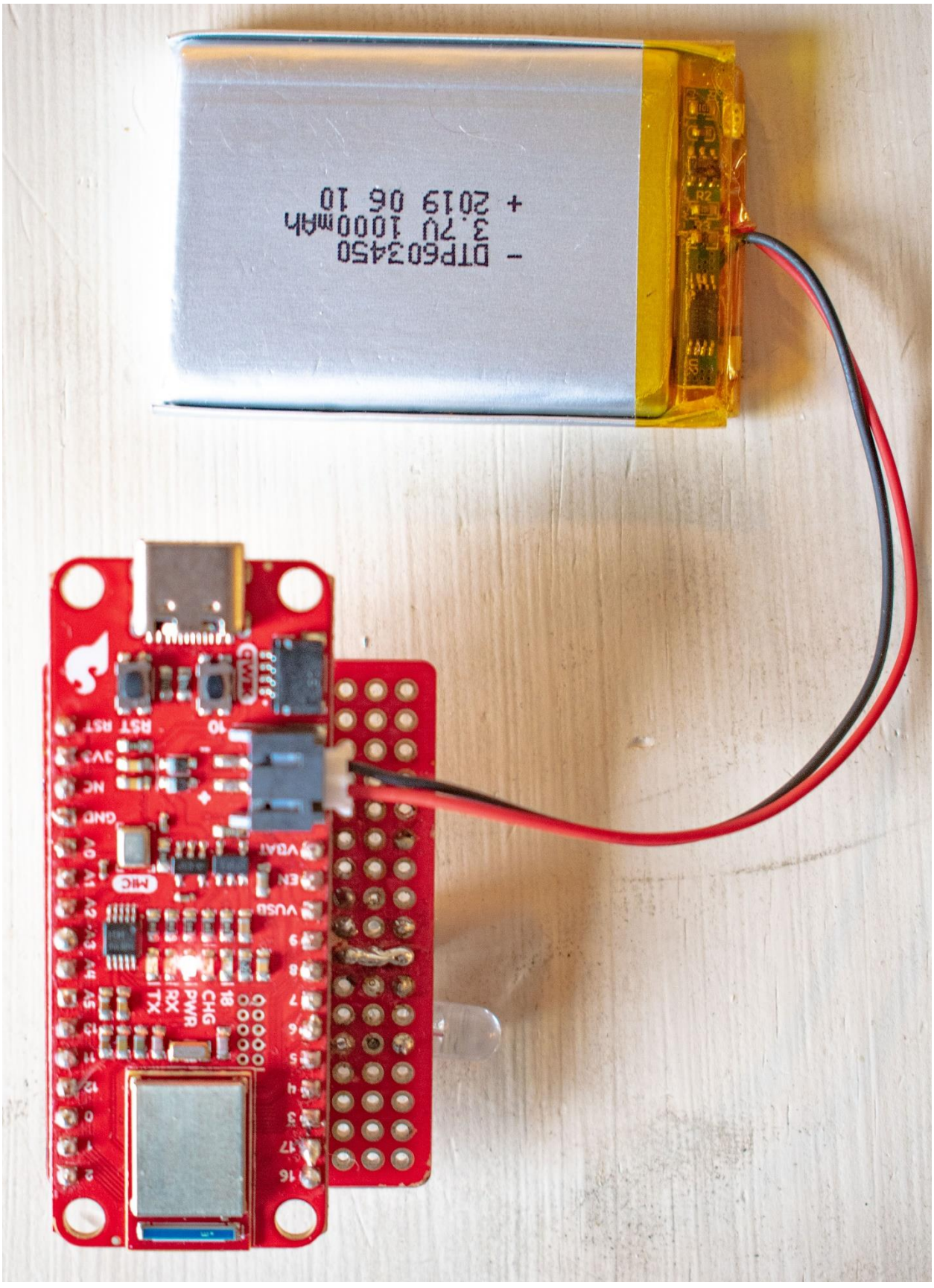
3.3.1 Intervalometer Implementation

```
636 //
637 //  < Shoot >
638 //    If @shots is greater than 0:
639 //      waits length of @interval in seconds, then calls @sht::key and decrements
640 //      @shots.
641 //
642 //      Calls itself recursively
643 //
644 //    Else:
645 //      Calls @shutdown
646 //
647 void interval::shoot(){
648   if(shots){
649     int wait_start = seconds();
650     int wait_current = 0;
651
652     while((wait_current = seconds() - wait_start) < interval);
653
654     rgb::interval_count_shots();
655
656     if(shots){
657       interval::shots--;
658       interval::shutter = true;
659       sht::key();
660     }
661
662     Serial.print("Yep \n");
663
664     interval::shoot();
665   }
666
667   else interval::shutdown();
668 }
```

3.4.1 Main Loop

```
void loop() {
  bool all_off = !interval::on && !sht::on && !recv::on;

  bool rec_sw_on = !button::debounce(SHT_SW) && button::debounce(REC_SW);
  bool sht_sw_on = !button::debounce(REC_SW) && button::debounce(SHT_SW);
  bool both_sw_on = button::debounce(SHT_SW) && button::debounce(REC_SW);

  if(all_off && both_sw_on)                    //REC_SW + SHT_SW Tap
    interval::on = interval::setup();          //  -> Start Intervalometer Setup

  else if(interval::setup_complete)            //Intervalometer Setup Complete
    interval::shoot();                         //  -> Fire Intervalometer

  else if(all_off && rec_sw_on)                //REC_SW Tap -> Begin IR Recording
    recv::on = recv::setup();

  else if (recv::on)                           //IR Recording Active -> Record LED Blink
    rgb::recv_wait_blink();

  else if(all_off && sht_sw_on)                //SHT_SW Tap -> Fire IR LED
    sht::setup();

  else if (all_off){                           //No Active Functions -> Sleep

    attachInterrupt(digitalPinToInterrupt(REC_SW), button::wakeup_isr, RISING);
    attachInterrupt(digitalPinToInterrupt(SHT_SW), button::wakeup_isr, RISING);

    am_hal_sysctrl_sleep(AM_HAL_SYSCTRL_SLEEP_DEEP);
  }
}
```


- DTP603450
3.7V 1000mAh
+ 2019 06 10
sparkfun


- DTP603450
3.7V 1000mAh
+ 2019 06 10
RST
3V3
NC
GND
A0
A1
A2
A3
A4
A5
SCK
MOSI
MISO
RX
TX
MIC
VBAT
EN
VUSB
PWR
CHG