Fitness Plug

Senior Project Report

CPE 461/462

Date: 6/5/2020

Author: Evan Ashley

Advisor(s): Dr. Hummel & Dr. Slivovsky

Abstract

My parents only allowed me to play videogames or watch television for a certain number of minutes or hours per day. By limiting screen time, they encouraged me to be active and find other things to do outside of television or videogames. The Fitness Plug aims to do both by converting time exercised to entertainment time; you can only watch television or play videogames for as long as you have exercised.

Table of Contents

Introduction

Stakeholders

The primary stakeholders of this project are parents who want their children to exercise more; and people who generally prioritize videogames, television, or other forms of entertainment over being active. Provided they have the Fitness Plug, people can have unlimited screen time so long as they are active and exercising.

Framed Insights and Opportunities

During the ideation process, I identified several necessary features for such a device. The device had to be adaptable to different types of exercise equipment, any involved devices would have to be connected to each other wirelessly, and the consumption device could be used independent of exercise. Supporting different types of exercise was important as this would not constrain the user to any one kind of exercise. In other words, by supporting a variety of sensor types, the device could identify when you were exercising and keep track of the duration, regardless of the type of exercise. The wireless component was important as this would not require you to be close to the outlet powering your entertainment system to be able to exercise. Tied with the previous feature, you can do any kind of exercise without being limited to the space itself; you could exercise outside or in a different room. Finally, the independence of the involved devices would enable a user to exercise for a period of time, and then consume entertainment, but not necessarily do the two at the same time. Effectively, the consumption device could be turned off during exercise, and store the duration of exercise (called “fitness minutes” from here forward) for use later.

Project Deliverables & Outcomes

The primary deliverable for this project is a device made of two components: a fitness device which keeps track of the time the user spends exercising, and a consumption device which spends the time exercised to power entertainment devices. Upon completing this project, I hope that this technology is explored more, and children are further encouraged to be active and exercise first, and to watch television or play videogames second.

Background

In a study by Dr. Leon Straker and Dr. Juliana Zabatiero entitled "Conflicting Guidelines on Young Children's Screen Time and Use of Digital Technology Create Policy and Practice Dilemmas", screen time in children helps in "enhancing learning, promoting children's digital skill set, engaging in STEM..., ensuring productive workforce membership, enabling competition in a globalized economy, and creating competence in social interaction." However, there are also "concerns about the effects on physical, cognitive, emotional, and social health, well-being, and development [which] include poor and sustained postures; ... limitation of time for learning opportunities, shortened attention spans, and fewer contexts for verbal interactions, problem-solving, and creativity; ... addiction, depression, and access to inappropriate content and advertising" [1].

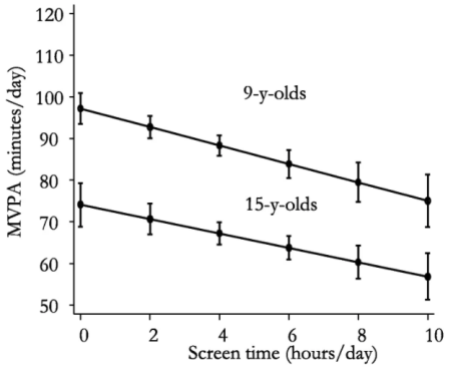

Figure 1 – Impact of screen time on moderate-to-vigorous physical activity (MVPA) [2]

In a study investigating the impact of sleep, screen time, school travel, and exercise on moderate-to-vigorous physical activity (MVPA) from 2018 published in the paper "Cross-Sectional and Prospective Associations between Sleep, Screen Time, Active School Travel, Sports/Exercise Participation and Physical Activity in Children and Adolescents", Knut Eirik Dalene and the other researchers found that "in 9- and 15-y-olds, [there were] inverse associations between screen time and MVPA, translating to 2.2 and 1.7 min/d less MVPA for each additional hour of screen time, respectively" [2]. We can see a graphical representation of these findings in Figure 1.

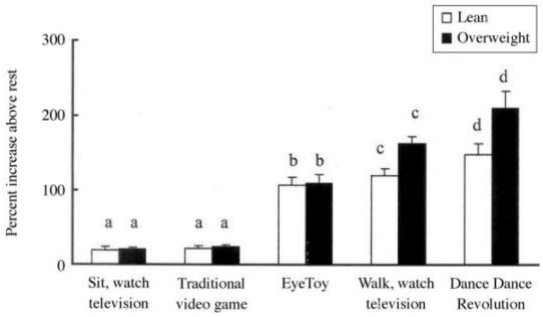

Figure 2 – Impact of activity during videogames and television on increase above rest [3]

In a study conducted by Dr. Lanningham-Foster and their team entitled "Energy Expenditure of Sedentary Screen Time Compared With Active Screen Time for Children", they found that "activity-promoting video games and treadmill television and computer use more than doubled energy expenditure, compared with the chair-based equivalents" [3]. As shown in Figure 2, even television coupled with walking on a treadmill motivated the participants in the study to increase activity.

There are inarguably negative aspects of screen time, however, screen time cannot be completely removed as technological competency is quite important in an increasingly digital world. By actively increasing physical activity during television or videogames, we can prevent the decrease in such activity generally associated with screen time. We know that television and videogames are great motivators for physical activity, as shown in the third study, so a device like the Fitness Plug would help to encourage such activities and associate them with the reward of digital entertainment.

Formal Project Definition

Customer Requirements

Going into this project, I had two main requirements: the devices couldn't be tethered by a wire as this would severely limit the number of exercises supported, and you had to be able to work out and gain fitness minutes without watching TV and consuming fitness minutes.

Engineering Requirements

Table 1 – Engineering requirements for the Fitness Plug

Spec #	Parameter	Target	Tolerance	Compliance
1	Connectivity	Bi-directional	N/A	I
2	Range	Devices can stay connected more than 10 meters apart	Max.	A, T
3	Accuracy	Fitness minutes are stored down to the second	Min.	I
4	Safety	Relay flips in less than a second	Min.	T, I
5		Accommodates grounding pins on the plugs of devices	N/A	I, S
6	Operation	Devices should connect quickly to one another	N/A	T, I
7	Usability	New sensors must be easily adapted to by the hardware and firmware	N/A	A, S

The options for compliance stand for: **a**nalysis, **t**esting, **i**nspection, and **s**imilarity.

End-User Personas

As shown in the Personas section of the Appendix, this device can be used for a number of reasons. The device can be used to actively discourage screen time over studying, or it can be used as a way to decrease inactivity. Considering Quinn's use case, his use of a computer which plugs into the wall enables the device to act as a 1:1 exercise to studying converter. In Judah's use case, however, using Reddit on a phone would produce a more disparate exercise to Reddit conversion, since the phone battery would charge at a quicker rate than the energy is consumed. In both cases the device would still limit screen time and encourage exercise, despite not both resulting in the same exercise to entertainment conversion.

Design

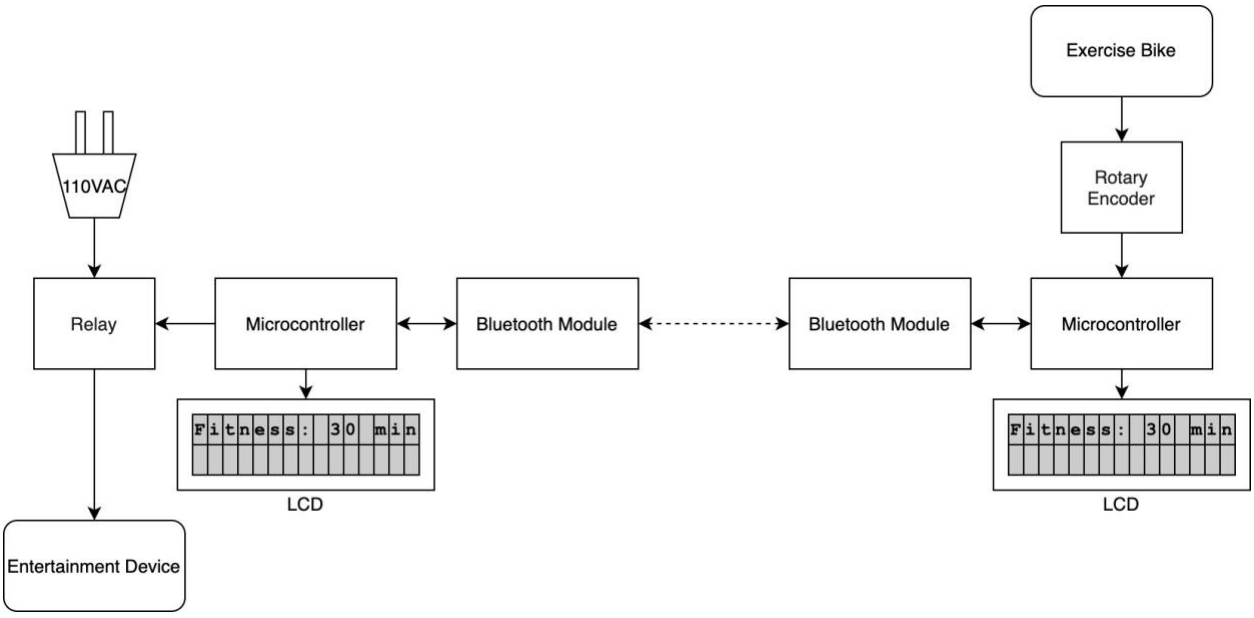

Figure 3 – Hardware overview for the Fitness Plug

At the start of the project I drew a block diagram, shown in Figure 3, roughly laying out how the Fitness Plug would work. This determined the features I needed to build into the project, and therefore what components I would have to purchase.

I knew that the devices needed to communicate wirelessly and at a relatively low range, so I would have to purchase microcontrollers that had WiFi or Bluetooth built in or buy external modules. Additionally, I knew the devices had to take up a small footprint, so devices like the Arduino Mega or MSP432 were out of the question. The Arduino Teensy would fit the size and BLE (Bluetooth Low Energy) constraint but wouldn't enable me to develop firmware in C since it uses the Arduino IDE. The ESP32-PICO-V4 development board, however, meets the size constraint, has Bluetooth and WiFi support (both in firmware and in hardware), and runs compiled C.

I was familiar with the Newhaven Displays and how to drive them so picking the LCD was somewhat straightforward. Additionally, with a limited number of pins and a lot of components connected to the ESP32, I wanted to make sure that a 4-bit mode of communication with the LCD was also supported in addition to the 8-bit mode.

Finally, I knew that I wanted to have some sort of control over the time stored on the devices, but I didn't want to buy an exercise bike. Using a Hall-effect sensor would not work since this would require mounting it along with magnets to some kind of rotating platform or crank. Whatever component I chose needed to be able to indicate rotation direction, so a quadrature encoder seemed to be a great option. I didn't need a particularly high resolution, so I was able to quickly acquire a cheap encoder from Amazon.

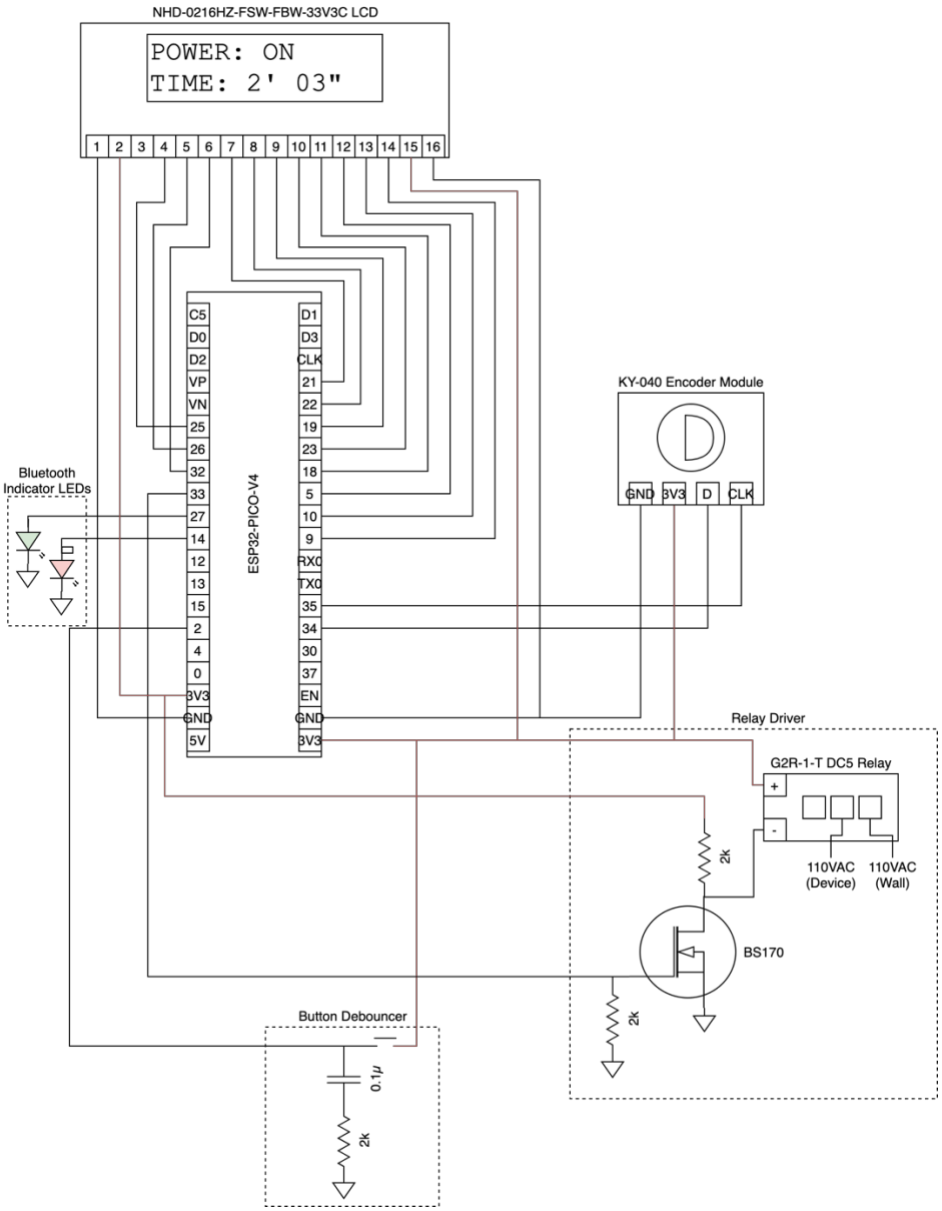

Figure 4 – Hardware schematic for the Fitness Plug devices

As shown in Figure 4, both the fitness and consumption devices support several peripherals, though only some of these are hooked up to each device, depending on what they are responsible for. The LCD shows the power state of the device, and the number of fitness minutes left in minutes and seconds. The relay driver circuit enables the ESP32 to control a relay with only 3.3V; this relay enables and disables the entertainment device (television, videogame console, phone charger, etc.). The button debouncer circuit prevents the microcontroller from picking up more than one edge when the button is pressed. Finally, the encoder controls the fitness minutes stored on the microcontroller in place of a sensor on an exercise bike or other exercise mechanism.

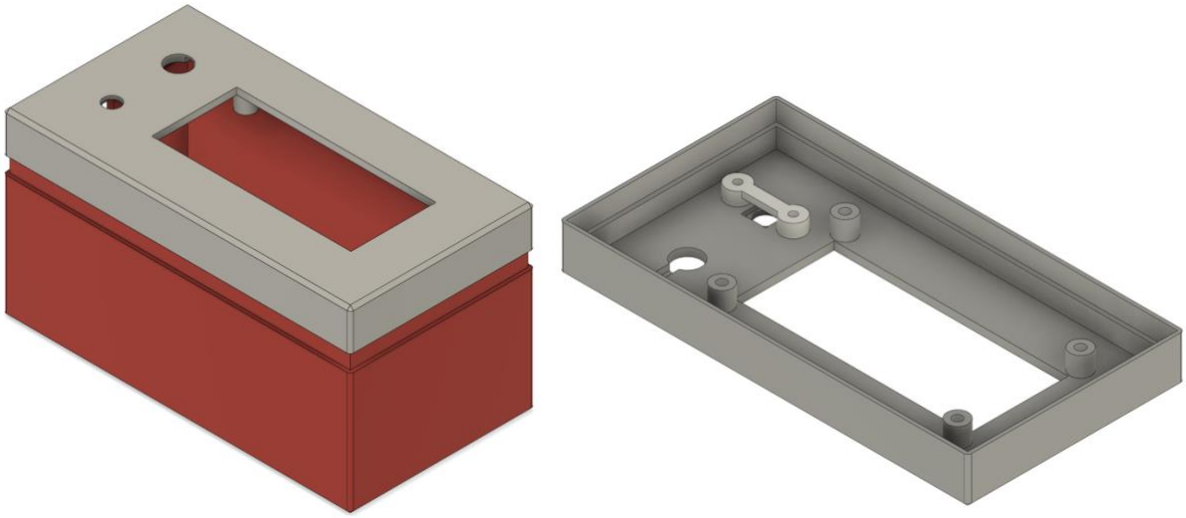

Figure 5 – Fitness Device CAD

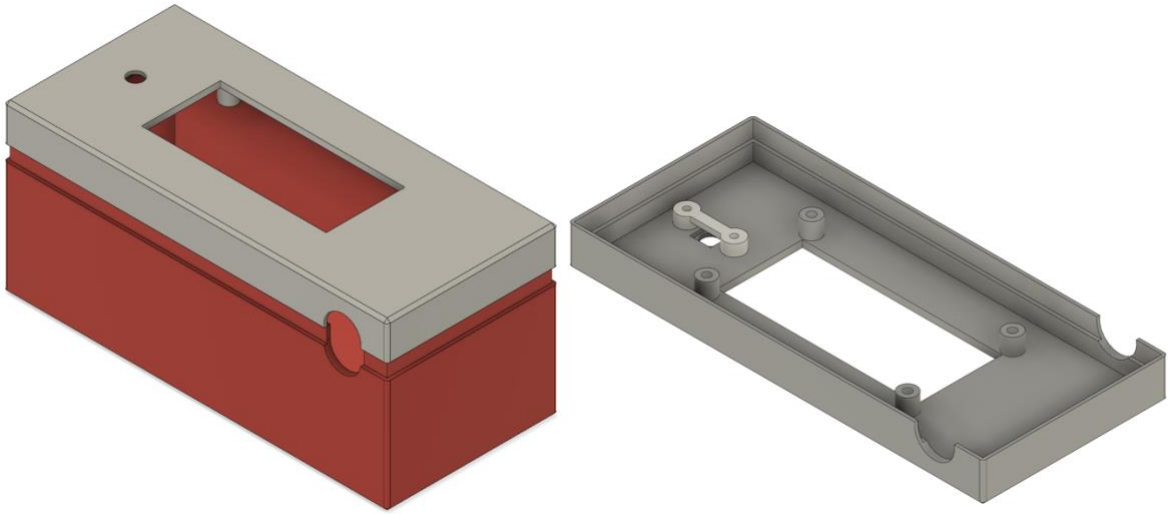

Figure 6 – Consumption Device CAD

The CAD models shown in Figures 5 and 6 were created primarily to remove visual clutter from the components, and not as part of an initial requirement for the project. Most of hardware and software development wrapped up in the first quarter, and while having breadboarded circuits for demo would enable me to quickly point at components and circuits involved, creating circuit boards and enclosures for the electronics seemed more professional. Additionally, having these boxes made it a bit easier to explain what each of the devices did, without the distraction of wires and scattered components on the breadboard.

Conclusion and Future Work

If development were to continue on this project, I would want to address a couple of outstanding issues. First, when the devices get disconnected, nothing restarts the connection sequence. Having some means of restarting the connection would make using the devices a lot easier as you would not have to worry about them becoming disconnected and consequently out of sync. Second, having mounts for Bluetooth connection lights would help to further debug any connection issues. Third, there is nothing to store the fitness minutes permanently across power cycles, so losing power on the consumption device would cause you to lose your fitness minutes. Fourth, the relay response time is not ideal; the relay flips state on the second, and not as soon as the consumption device is flipped from on to off, or off to on. As mentioned in the Engineering Requirements table (Table 1), this could be a safety hazard. Finally, having USB cables to power the microcontrollers is unideal for at least the fitness device. Having a battery for the fitness device would make it truly wireless and remove the need for a USB cable to power it. Powering the consumption device directly off of the wall plug would remove the need for two outlets to be used to power the consumption device.

Reflection

This project enabled me to take advantage of all of my knowledge and experience from my college career; it incorporated some circuits design and manufacturing, CAD modeling, software development and best practices, version control, and research and development. There were points in the project where I didn't think I was going to be able to get it working, but with a little bit of extra time spent, I was able to work through them.

The most notable challenge in this project was Bluetooth; I didn't know a lot about Bluetooth when this project began, and it was quite challenging to use anything other than the Bluetooth stack built into the Espressif toolchain. Throughout the whole project, it was quite apparent that they pushed for reliance on their API functions over manipulating registers to configure various communication peripherals, like Bluetooth, as everything from GPIO (General Purpose Input/Output) to the RTC (Real Time Clock) was configured via functions calls. While they had numerous examples for the Bluetooth functionality in particular, it was quite hard to determine which example I should base my code off of. It took several weeks for me to find one that was promising, and even longer to read through the code and determine if it was worth pursuing. Over spring break, one of them seemed quite appealing (the SPP Acceptor & SPP Initiator example), and I moved forward with it. Using this example over other ones enabled me to build out a networking module which could handle communications in either direction, making all of the wireless code much easier to develop.

The only other challenge I can think of was finding contiguous time in my schedule to work on the project; I found myself taking advantage of free time to make a ton of progress in anticipation of a loss of time later while working on projects in my other classes. Due to changes to my Spring Break plans, I was able to make up a lot of lost time on Bluetooth over Spring Break, eventually arriving at the solution I had at the end of the project.

Appendix

Bill of Materials

Table 2 – Bill of Materials for the Fitness Plug project

Material	Store	Quantity	Cost	Total Cost
ESP32-PICO-KIT	Digikey	2	$10.00	$20.00
G2R-1 DC3 (Relay)	Digikey	1	$5.22	$5.22
AK500/16-OE-5-0.5 (Male Plug)	Digikey	1	$2.52	$2.52
04530.73.01 (Female Plug)	Digikey	1	$5.53	$5.53
NHD-0216HZ-FSW-FBW-33V3C	Digikey	2	$12.15	$24.30
KY-040 (Rotary Encoder)	Amazon	1	$6.59	$6.59
Assorted Female Pin Headers	Amazon	1	$14.99	$14.99
Solderless Breadboard Jumper Wires	Amazon	2	$10.76	$21.52
			Total:	$100.67

As shown in Table 2, the total cost for this device came out to around $100. Considering spare parts and parts I had on hand already, the development of this senior project probably came to around $200 in total. If this device were to be marketed, I have no doubt in my mind that components could be purchased in bulk, and circuit boards could be designed removing a lot of unnecessary peripherals on the ESP32-PICO-KIT board to substantially reduce the price to something more affordable.

Personas

Figure 7 – Judah

Judah spends a lot of time on Reddit, but not a lot of time reading books. Since she has a lot of classes which require textbooks, her understanding of the material has departed from that of the class. Her parents and teachers want her to spend more time focusing on the material in the book, and less time on Reddit. Her parents thought that if there existed a device which limited her Reddit consumption, she would spend more time reading her textbooks.

Figure 8 - Quinn

Quinn spends most of his time studying trivia for his quiz bowl competitions. He doesn't, however, spend much time exercising. Since the trivia he does is on the tower computer which has a wall power supply, his parents saw the opportunity to reward his exercise with trivia studying. They couldn't, however, find a device that did just this, until now.

Figure 9 – Jay

Jay is quite active on Twitch, spending numerous hours per day streaming the videogames they play. However, they have neglected a lot of schoolwork, and don't spend a lot of time exercising. Since Jay's parents believe staying active is important for the brain, they decided that they needed some kind of device that would encourage exercising before videogames.

Code
main.c

```
#include <stdio.h>
#include <string.h>

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "lcd.h"
#include "relay.h"
#include "encoder.h"
#include "power.h"
#include "bluetooth.h"
#include "network.h"

#define FITNESS_DEVICE
// #define CONSUMPTION_DEVICE

#ifdef CONSUMPTION_DEVICE
   static void IRAM_ATTR flipRelay(void *args);
#endif
void bt_data_callback(uint8_t *data, int len);

uint32_t timePeriod = 2 * 60;

void app_main(void) {

   configureLCD();
   configureRelay();
   configureEncoder();
   configurePower();
   setPowered(0);

   resetDisplay();
   writeString("Power: OFF\n");
   writeString("Time: ");

   #ifdef FITNESS_DEVICE
      configureBluetooth(INITIATOR, bt_data_callback);
   #endif
   #ifdef CONSUMPTION_DEVICE
      configureBluetooth(ACCEPTOR, bt_data_callback);
      xTaskCreate(flipRelay, "flipRelay", 2048, NULL, 10, NULL);
   #endif

   char data[16] = {0};
   int32_t count;
   int powered = 0;
   while(1) {
      setCount(0);
```

```
      if (powered != isPowered()) {
         powered = isPowered();
         if (isConnected()) {
            writeConfigPacket(isPowered());
         }
         moveCursor(8, 1);
         writeString("          "); /* clear screen after "Power: " */
         moveCursor(8, 1);
         sprintf(data, isPowered() ? "ON\n" : "OFF\n");
         writeString(data);
         moveCursor(7, 2);
      }
      writeString("         "); /* clear screen after "Time: " */
      moveCursor(7, 2);
      sprintf(data, "%2d' %02d\"", timePeriod / 60, timePeriod % 60);
      writeString(data);

      if (isConnected()) {
         count = getCount();
         if (count) {
            timePeriod += count;
            writeDeltaPacket(count);
         }
      }

      if (!isPowered()) {
         switchRelay(0);
      }
   }
}

void bt_data_callback(uint8_t *data, int len) {
   Header header;
   DeltaPacket deltaPacket;
   SetPacket setPacket;
   ConfigPacket configPacket;

   stripHeader(data, len, &header);

   switch (header.flag) {
      case DELTA_PACKET:
         stripPacket(data, header.len, &deltaPacket);
         timePeriod += deltaPacket.delta;
         break;
      case SET_PACKET:
         stripPacket(data, header.len, &setPacket);
         timePeriod = setPacket.timePeriod;
         break;
      case CONFIG_PACKET:
         stripPacket(data, header.len, &configPacket);
         setPowered(configPacket.powered);
         break;
   }
```

```
}

static void IRAM_ATTR flipRelay(void *args) {
   const TickType_t xDelay = 1000 / portTICK_PERIOD_MS;
   while (1) {
      vTaskDelay(xDelay);
      if (isPowered()) {
         switchRelay(timePeriod);
         timePeriod = (timePeriod) ? timePeriod - 1 : timePeriod;
         if (isConnected()) {
            writeSetPacket(timePeriod);
         }
      } else {
         switchRelay(0);
      }
   }
}
```

gpio.h

```
#ifndef GPIO_H
#define GPIO_H

void setGPIO(gpio_num_t gpio_num, uint32_t level);

#endif
```

gpio.c

```
#include "driver/gpio.h"
#include "gpio.h"

void setGPIO(gpio_num_t gpio_num, uint32_t level) {
   if (gpio_set_level(gpio_num, level) != ESP_OK) {
      printf("Could not set GPIO[%d]\n", gpio_num);
   }
}
```

lcd.h

```
#ifndef LCD_H
#define LCD_H

#include <stdint.h>

#define RS_OUTPUT_PIN 25
#define RW_OUTPUT_PIN 26
#define E_OUTPUT_PIN 32
#define DB0_OUTPUT_PIN 21
#define DB1_OUTPUT_PIN 22
#define DB2_OUTPUT_PIN 19
```

```
#define DB3_OUTPUT_PIN 23
#define DB4_OUTPUT_PIN 18
#define DB5_OUTPUT_PIN 5
#define DB6_OUTPUT_PIN 10
#define DB7_OUTPUT_PIN 9
#define GPIO_OUTPUT_PIN_SEL_LCD ((1ULL << RS_OUTPUT_PIN)  |\
                                 (1ULL << RW_OUTPUT_PIN)  |\
                                 (1ULL << E_OUTPUT_PIN)   |\
                                 (1ULL << DB0_OUTPUT_PIN) |\
                                 (1ULL << DB1_OUTPUT_PIN) |\
                                 (1ULL << DB2_OUTPUT_PIN) |\
                                 (1ULL << DB3_OUTPUT_PIN) |\
                                 (1ULL << DB4_OUTPUT_PIN) |\
                                 (1ULL << DB5_OUTPUT_PIN) |\
                                 (1ULL << DB6_OUTPUT_PIN) |\
                                 (1ULL << DB7_OUTPUT_PIN))

void configureLCD(void);
void resetDisplay(void);
void moveCursor(uint8_t column, uint8_t row);
void writeString(char *str);

#endif
```

lcd.c

```
#include <stdio.h>

#include "driver/gpio.h"
#include "esp_timer.h"
#include "lcd.h"
#include "gpio.h"

void delay_us(uint32_t us) {
   int64_t now = esp_timer_get_time();
   while (esp_timer_get_time() - now < us);
}

/* pass db4-db8 in the lower nibble */
void writeToBus(uint8_t db) {
   setGPIO(DB0_OUTPUT_PIN, (db & 0x01));
   setGPIO(DB1_OUTPUT_PIN, (db & 0x02) >> 1);
   setGPIO(DB2_OUTPUT_PIN, (db & 0x04) >> 2);
   setGPIO(DB3_OUTPUT_PIN, (db & 0x08) >> 3);
   setGPIO(DB4_OUTPUT_PIN, (db & 0x10) >> 4);
   setGPIO(DB5_OUTPUT_PIN, (db & 0x20) >> 5);
   setGPIO(DB6_OUTPUT_PIN, (db & 0x40) >> 6);
   setGPIO(DB7_OUTPUT_PIN, (db & 0x80) >> 7);
}

void send(void) {
   setGPIO(E_OUTPUT_PIN, 1);
```

```
    delay_us(1);
    setGPIO(E_OUTPUT_PIN, 0);
}

void data(uint8_t data) {
    writeToBus(data);
    setGPIO(RS_OUTPUT_PIN, 1); /* data */
    setGPIO(RW_OUTPUT_PIN, 0); /* write */
    delay_us(40);
    send(); /* send data */
}

void command(uint8_t command) {
    writeToBus(command);
    setGPIO(RS_OUTPUT_PIN, 0); /* instruction */
    setGPIO(RW_OUTPUT_PIN, 0); /* write */
    delay_us(40);
    send(); /* send command */
}

void configureLCD(void) {
    gpio_config_t io_conf;
    //disable interrupt
    io_conf.intr_type = GPIO_PIN_INTR_DISABLE;
    //set as output mode
    io_conf.mode = GPIO_MODE_OUTPUT;
    //set output pins
    io_conf.pin_bit_mask = GPIO_OUTPUT_PIN_SEL_LCD;
    //disable pull-down mode
    io_conf.pull_down_en = 0;
    //disable pull-down mode
    io_conf.pull_up_en = 0;
    //configure GPIO
    gpio_config(&io_conf);

    /* wakeup */
    setGPIO(RS_OUTPUT_PIN, 0); /* instruction */
    setGPIO(RW_OUTPUT_PIN, 0); /* write */
    setGPIO(E_OUTPUT_PIN, 0);
    delay_us(50000);
    command(0x30);
    delay_us(5000);
    command(0x30);
    delay_us(160);
    command(0x30);
    delay_us(160);

    /* configure display */
    command(0x38); /* 8-bit, 2-line, 5x8 font */
    command(0x10); /* move cursor, move right */
    command(0x0C); /* display on, cursor off, blinking off */
    command(0x06); /* increment cursor, don't shift */
}
```

```
void resetDisplay(void) {
   /* clear and home */
   command(0x01);
   delay_us(1500);
   command(0x02);
   delay_us(1500);
}

void writeString(char *str) {
   uint32_t index = 0;
   while (str[index] != '\0') {
      if (str[index] == '\n') {
         command(0xC0); /* move down a line */
      } else {
         data(str[index]);
      }
      index++;
   }
}

void moveCursor(uint8_t column, uint8_t row) {
   command(0x80 + (0x40*(row - 1) + (column - 1)));
}
```

relay.h

```
#ifndef RELAY_H
#define RELAY_H

#include <stdint.h>

#define RELAY_OUTPUT_PIN 33
#define GPIO_OUTPUT_PIN_SEL_RELAY (1ULL << RELAY_OUTPUT_PIN)

void configureRelay(void);
void switchRelay(uint32_t state);

#endif
```

relay.c

```
#include "driver/gpio.h"
#include "relay.h"

void configureRelay(void) {
   gpio_config_t io_conf;
   //disable interrupt
   io_conf.intr_type = GPIO_PIN_INTR_DISABLE;
   //set as output mode
   io_conf.mode = GPIO_MODE_OUTPUT;
   //set output pins
```

```
    io_conf.pin_bit_mask = GPIO_OUTPUT_PIN_SEL_RELAY;
    //disable pull-down mode
    io_conf.pull_down_en = 0;
    //disable pull-down mode
    io_conf.pull_up_en = 0;
    //configure GPIO
    gpio_config(&io_conf);
}

void switchRelay(uint32_t state) {
    uint32_t level = state ? 1 : 0;
    if (gpio_set_level(RELAY_OUTPUT_PIN, level) != ESP_OK) {
        printf("Couldn't set GPIO[%d]\n", RELAY_OUTPUT_PIN);
    }
}
```

encoder.h

```
#ifndef ENCODER_H
#define ENCODER_H

#include <stdint.h>

#define PIN_A_INPUT_GPIO_PIN 35 (CLK)
#define PIN_B_INPUT_GPIO_PIN 34 (D)
#define GPIO_INPUT_PIN_SEL_ENCODER ((1ULL << PIN_A_INPUT_GPIO_PIN) |\
                                    (1ULL << PIN_B_INPUT_GPIO_PIN))

void configureEncoder(void);
int32_t getCount(void);
void setCount(int32_t newCount);

#endif
```

encoder.c

```
#include "encoder.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "driver/gpio.h"

static int32_t count = 0;
static xQueueHandle gpio_evt_queue = NULL;

static void IRAM_ATTR gpio_isr_handler(void* arg) {
    uint32_t gpio_num = (uint32_t) arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
}

static void IRAM_ATTR handleGPIOInterrupt(void *arg) {
    static uint32_t prevA = 0, pinA = 0, pinB = 0;
```

```
    int32_t direction;
    int io_num;
    while (1) {
        xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY);
        direction = 0;
        pinA = gpio_get_level(PIN_A_INPUT_GPIO_PIN);
        if (pinA != prevA) { /* check if something has changed */
            pinB = gpio_get_level(PIN_B_INPUT_GPIO_PIN);
            if (pinB != pinA) { /* 'a' moved first, rotating clockwise */
                direction = 1;
            } else { /* 'b' moved first, rotating counter-clockwise */
                direction = -1;
            }
        }
        prevA = pinA;
        count = count + direction;
    }
}

void configureEncoder(void) {

    gpio_config_t io_conf;

    //interrupt of rising edge
    io_conf.intr_type = GPIO_INTR_DISABLE;
    //bit mask of the pins, use GPIO4/5 here
    io_conf.pin_bit_mask = GPIO_INPUT_PIN_SEL_ENCODER;
    //set as input mode
    io_conf.mode = GPIO_MODE_INPUT;
    //disable pull-up mode
    io_conf.pull_up_en = 0;
    //enable pull-down mode
    io_conf.pull_down_en = 1;
    gpio_config(&io_conf);

    gpio_set_intr_type(PIN_A_INPUT_GPIO_PIN, GPIO_INTR_NEGEDGE);

    //create a queue to handle gpio event from isr
    gpio_evt_queue = xQueueCreate(10, sizeof(uint32_t));
    xTaskCreate(handleGPIOInterrupt, "handleGPIOInterrupt", 2048, NULL, 10,
NULL);

    //install gpio isr service
    gpio_install_isr_service(0);
    gpio_isr_handler_add(PIN_A_INPUT_GPIO_PIN, gpio_isr_handler, (void*)
PIN_A_INPUT_GPIO_PIN);
}

void setCount(int32_t newCount) {
    count = newCount;
}

int32_t getCount(void) {
```

```
    return count;
}
```

power.h

```
#ifndef POWER_H
#define POWER_H

#define POWER_PIN 2
#define GPIO_INPUT_PIN_SEL_POWER (1ULL << POWER_PIN)

void configurePower(void);
int isPowered(void);
void setPowered(int powered);

#endif
```

power.c

```
#include <stdio.h>

#include "power.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "driver/gpio.h"

int powered = 0;
static xQueueHandle gpio_evt_queue = NULL;

static void IRAM_ATTR power_gpio_isr_handler(void* arg) {
    uint32_t gpio_num = (uint32_t) arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
}

static void IRAM_ATTR handlePowerGPIOInterrupt(void *arg) {
    int io_num;
    while (1) {
        xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY);
        powered ^= gpio_get_level(POWER_PIN);
        printf(powered ? "True\n" : "False\n");
        xQueueReset(gpio_evt_queue);
    }
}

void configurePower(void) {

    gpio_config_t io_conf;

    //interrupt of rising edge
    io_conf.intr_type = GPIO_INTR_DISABLE;
    //bit mask of the pins, use GPIO4/5 here
```

```
    io_conf.pin_bit_mask = GPIO_INPUT_PIN_SEL_POWER;
    //set as input mode
    io_conf.mode = GPIO_MODE_INPUT;
    //disable pull-up mode
    io_conf.pull_up_en = 0;
    //enable pull-down mode
    io_conf.pull_down_en = 1;
    gpio_config(&io_conf);

    gpio_set_intr_type(POWER_PIN, GPIO_INTR_POSEDGE);

    //create a queue to handle gpio event from isr
    gpio_evt_queue = xQueueCreate(10, sizeof(uint32_t));
    xTaskCreate(handlePowerGPIOInterrupt, "handlePowerGPIOInterrupt", 2048,
NULL, 10, NULL);

    //install gpio isr service
    // gpio_install_isr_service(0);
    gpio_isr_handler_add(POWER_PIN, power_gpio_isr_handler, (void*)
POWER_PIN);
}

void setPowered(int powered_) {
    powered = powered_;
}

int isPowered(void) {
    return powered;
}
```

bluetooth.h

```
#ifndef BT_COMMON_H
#define BT_COMMON_H

#include "esp_spp_api.h"
#include "esp_gap_bt_api.h"

#define SPP_DATA_LEN ESP_SPP_MAX_MTU

#define CONNECTED_PIN 27
#define CONNECTION_FAILURE_PIN 14
#define GPIO_OUTPUT_PIN_SEL_BT ((1ULL << CONNECTED_PIN) |\
                                (1ULL << CONNECTION_FAILURE_PIN))


typedef void (*spp_data_rcv_cb_t)(uint8_t *data, int len);

extern int handle;
extern spp_data_rcv_cb_t dataCallback;

typedef enum {
```

```
        INITIATOR,
        ACCEPTOR
} spp_t;

void configureBluetooth(spp_t type, spp_data_rcv_cb_t cb);
void writeData(uint8_t *data, int len);
int isConnected(void);
void success(void);
void failure(void);
void translateGAPEvent(const char* tag, esp_bt_gap_cb_event_t event);
void translateSPPEvent(const char* tag, esp_spp_cb_event_t event);

#endif
```

bluetooth.c

```
#include <stdio.h>
#include "esp_spp_api.h"
#include "esp_gap_bt_api.h"
#include "esp_log.h"
#include "driver/gpio.h"

#include "gpio.h"
#include "bluetooth.h"
#include "spp_acceptor.h"
#include "spp_initiator.h"

int handle; // externally available
spp_data_rcv_cb_t dataCallback; // externally available

int connected;

void configureBluetooth(spp_t type, spp_data_rcv_cb_t cb) {

        gpio_config_t io_conf;
        //disable interrupt
        io_conf.intr_type = GPIO_PIN_INTR_DISABLE;
        //set as output mode
        io_conf.mode = GPIO_MODE_OUTPUT;
        //set output pins
        io_conf.pin_bit_mask = GPIO_OUTPUT_PIN_SEL_BT;
        //disable pull-down mode
        io_conf.pull_down_en = 0;
        //disable pull-down mode
        io_conf.pull_up_en = 0;
        //configure GPIO
        gpio_config(&io_conf);

        failure();

        dataCallback = cb;
```

```
        switch (type) {
          case INITIATOR:
                initSppInitiator();
                break;
          case ACCEPTOR:
                initSppAcceptor();
                break;
        }
}

void success(void) {
        connected = 1;
        setGPIO(CONNECTED_PIN, 1);
        setGPIO(CONNECTION_FAILURE_PIN, 0);
}

void failure(void) {
        connected = 0;
        setGPIO(CONNECTED_PIN, 0);
        setGPIO(CONNECTION_FAILURE_PIN, 1);
}

int isConnected(void) {
        return connected;
}

void writeData(uint8_t *data, int len) {
        if (len > SPP_DATA_LEN) {
                printf("Cannot write %d bytes via SPP\n", len);
                return;
        }

        if (esp_spp_write(handle, len, data) != ESP_OK) {
                printf("esp_spp_write() failed.");
        }
}

void translateGAPEvent(const char* tag, esp_bt_gap_cb_event_t event) {

        switch (event) {
                case ESP_BT_GAP_DISC_RES_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_DISC_RES_EVT");
                        break;
                case ESP_BT_GAP_DISC_STATE_CHANGED_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_DISC_STATE_CHANGED_EVT");
                        break;
                case ESP_BT_GAP_RMT_SRVCS_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_RMT_SRVCS_EVT");
                        break;
                case ESP_BT_GAP_RMT_SRVC_REC_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_RMT_SRVC_REC_EVT");
                        break;
                case ESP_BT_GAP_AUTH_CMPL_EVT:
```

```
                        ESP_LOGI(tag, "ESP_BT_GAP_AUTH_CMPL_EVT");
                        break;
                case ESP_BT_GAP_PIN_REQ_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_PIN_REQ_EVT");
                        break;
                case ESP_BT_GAP_CFM_REQ_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_CFM_REQ_EVT");
                        break;
                case ESP_BT_GAP_KEY_NOTIF_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_KEY_NOTIF_EVT");
                        break;
                        break;
                case ESP_BT_GAP_READ_RSSI_DELTA_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_READ_RSSI_DELTA_EVT");
                        break;
                case ESP_BT_GAP_CONFIG_EIR_DATA_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_CONFIG_EIR_DATA_EVT");
                        break;
                case ESP_BT_GAP_SET_AFH_CHANNELS_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_SET_AFH_CHANNELS_EVT");
                        break;
                case ESP_BT_GAP_READ_REMOTE_NAME_EVT:
                        ESP_LOGI(tag, "ESP_BT_GAP_READ_REMOTE_NAME_EVT");
                        break;
                case ESP_BT_GAP_EVT_MAX:
                        ESP_LOGI(tag, "ESP_BT_GAP_EVT_MAX");
                        break;
                default:
                        break;
        }
}

void translateSPPEvent(const char* tag, esp_spp_cb_event_t event) {
        switch (event) {
                case ESP_SPP_INIT_EVT:
                        ESP_LOGI(tag, "ESP_SPP_INIT_EVT");
                        break;
                case ESP_SPP_DISCOVERY_COMP_EVT:
                        ESP_LOGI(tag, "ESP_SPP_DISCOVERY_COMP_EVT");
                        break;
                case ESP_SPP_OPEN_EVT:
                        ESP_LOGI(tag, "ESP_SPP_OPEN_EVT");
                        break;
                case ESP_SPP_CLOSE_EVT:
                        ESP_LOGI(tag, "ESP_SPP_CLOSE_EVT");
                        break;
                case ESP_SPP_START_EVT:
                        ESP_LOGI(tag, "ESP_SPP_START_EVT");
                        break;
                case ESP_SPP_CL_INIT_EVT:
                        ESP_LOGI(tag, "ESP_SPP_CL_INIT_EVT");
                        break;
                case ESP_SPP_DATA_IND_EVT:
```

```
                        ESP_LOGI(tag, "ESP_SPP_DATA_IND_EVT");
                        break;
                case ESP_SPP_CONG_EVT:
                        ESP_LOGI(tag, "ESP_SPP_CONG_EVT");
                        break;
                case ESP_SPP_WRITE_EVT:
                        ESP_LOGI(tag, "ESP_SPP_WRITE_EVT");
                        break;
                case ESP_SPP_SRV_OPEN_EVT:
                        ESP_LOGI(tag, "ESP_SPP_SRV_OPEN_EVT");
                        break;
                default:
                        break;
        }
}
```

spp_initiator.h

```
#ifndef SPP_INITIATOR_H
#define SPP_INITIATOR_H

void initSppInitiator(void);

#endif
```

spp_initiator.c

```
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>

#include "nvs.h"
#include "nvs_flash.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_bt.h"
#include "esp_bt_main.h"
#include "esp_gap_bt_api.h"
#include "esp_bt_device.h"
#include "esp_spp_api.h"
#include "time.h"
#include "sys/time.h"

#include "bluetooth.h"

#define SPP_TAG "SPP_INITIATOR"
#define DEVICE_NAME "ESP_SPP_INITIATOR"

#define SPP_DATA_LEN ESP_SPP_MAX_MTU
```

```
static const esp_spp_mode_t esp_spp_mode = ESP_SPP_MODE_CB;

static const esp_spp_sec_t sec_mask = ESP_SPP_SEC_AUTHENTICATE;
static const esp_spp_role_t role_master = ESP_SPP_ROLE_MASTER;

static esp_bd_addr_t peer_bd_addr;
static uint8_t peer_bdname_len;
static char peer_bdname[ESP_BT_GAP_MAX_BDNAME_LEN + 1];
static const char remote_device_name[] = "ESP_SPP_ACCEPTOR";
static const esp_bt_inq_mode_t inq_mode = ESP_BT_INQ_MODE_GENERAL_INQUIRY;
static const uint8_t inq_len = 30;
static const uint8_t inq_num_rsps = 0;

int handle; // provided by bluetooth.h
spp_data_rcv_cb_t dataCallback; // provided by bluetooth.h

static bool get_name_from_eir(uint8_t *eir, char *bdname, uint8_t
*bdname_len) {
    uint8_t *rmt_bdname = NULL;
    uint8_t rmt_bdname_len = 0;

    if (!eir) {
        return false;
    }

    rmt_bdname = esp_bt_gap_resolve_eir_data(eir,
ESP_BT_EIR_TYPE_CMPL_LOCAL_NAME, &rmt_bdname_len);
    if (!rmt_bdname) {
        rmt_bdname = esp_bt_gap_resolve_eir_data(eir,
ESP_BT_EIR_TYPE_SHORT_LOCAL_NAME, &rmt_bdname_len);
    }

    if (rmt_bdname) {
        if (rmt_bdname_len > ESP_BT_GAP_MAX_BDNAME_LEN) {
            rmt_bdname_len = ESP_BT_GAP_MAX_BDNAME_LEN;
        }

        if (bdname) {
            memcpy(bdname, rmt_bdname, rmt_bdname_len);
            bdname[rmt_bdname_len] = '\0';
        }
        if (bdname_len) {
            *bdname_len = rmt_bdname_len;
        }
        return true;
    }

    return false;
}

static void esp_spp_cb(esp_spp_cb_event_t event, esp_spp_cb_param_t *param) {
    switch (event) {
        case ESP_SPP_INIT_EVT: // SPP is initiated
```

```
            ESP_LOGI(SPP_TAG, "ESP_SPP_INIT_EVT");
            esp_bt_dev_set_device_name(DEVICE_NAME);
            esp_bt_gap_set_scan_mode(ESP_BT_CONNECTABLE,
ESP_BT_GENERAL_DISCOVERABLE);
            esp_bt_gap_start_discovery(inq_mode, inq_len, inq_num_rsps);
            break;
        case ESP_SPP_OPEN_EVT: // SPP client connection open
            ESP_LOGI(SPP_TAG, "ESP_SPP_OPEN_EVT");
            memcpy(&handle, &param->srv_open.handle, sizeof(handle));
            success();
            break;
        case ESP_SPP_CLOSE_EVT: // SPP client connection open
            ESP_LOGI(SPP_TAG, "ESP_SPP_CLOSE_EVT");
            failure();
            break;
        case ESP_SPP_DISCOVERY_COMP_EVT: // SPP discovery complete
            ESP_LOGI(SPP_TAG, "ESP_SPP_DISCOVERY_COMP_EVT status=%d
scn_num=%d", param->disc_comp.status, param->disc_comp.scn_num);
            if (param->disc_comp.status == ESP_SPP_SUCCESS) {
                esp_spp_connect(sec_mask, role_master, param-
>disc_comp.scn[0], peer_bd_addr);
            } else {
                failure();
            }
            break;
        case ESP_SPP_WRITE_EVT: // SPP write operation completes (only for
ESP_SPP_MODE_CB)
            ESP_LOGI(SPP_TAG, "ESP_SPP_WRITE_EVT len=%d cong=%d", param-
>write.len, param->write.cong);
            break;
        case ESP_SPP_DATA_IND_EVT: // when SPP connection received data (only
for ESP_SPP_MODE_CB)
            ESP_LOGI(SPP_TAG, "ESP_SPP_DATA_IND_EVT len=%d", param-
>data_ind.len);
            dataCallback(param->data_ind.data, param->data_ind.len);
            // received data, store in queue
            // data = param->data_ind.data
            // len = param->data_ind.len
            break;
        default:
            translateSPPEvent(SPP_TAG, event);
            break;
    }
}

static void esp_bt_gap_cb(esp_bt_gap_cb_event_t event, esp_bt_gap_cb_param_t
*param) {
    switch(event) {
        case ESP_BT_GAP_DISC_RES_EVT: // device discovery result event
            ESP_LOGI(SPP_TAG, "ESP_BT_GAP_DISC_RES_EVT");
            esp_log_buffer_hex(SPP_TAG, param->disc_res.bda,
ESP_BD_ADDR_LEN);
            for (int i = 0; i < param->disc_res.num_prop; i++){
```

```
                if (param->disc_res.prop[i].type == ESP_BT_GAP_DEV_PROP_EIR
                    && get_name_from_eir(param->disc_res.prop[i].val,
peer_bdname, &peer_bdname_len)){
                    esp_log_buffer_char(SPP_TAG, peer_bdname,
peer_bdname_len);
                    if (strlen(remote_device_name) == peer_bdname_len
                        && strncmp(peer_bdname, remote_device_name,
peer_bdname_len) == 0) {
                        memcpy(peer_bd_addr, param->disc_res.bda,
ESP_BD_ADDR_LEN);
                        esp_spp_start_discovery(peer_bd_addr);
                        esp_bt_gap_cancel_discovery();
                    }
                }
            }
            break;
        case ESP_BT_GAP_DISC_STATE_CHANGED_EVT: // discovery state changed
event
            ESP_LOGI(SPP_TAG, "ESP_BT_GAP_DISC_STATE_CHANGED_EVT");
            break;
        case ESP_BT_GAP_AUTH_CMPL_EVT: // AUTH complete event
            if (param->auth_cmpl.stat == ESP_BT_STATUS_SUCCESS) {
                ESP_LOGI(SPP_TAG, "authentication success: %s", param-
>auth_cmpl.device_name);
                esp_log_buffer_hex(SPP_TAG, param->auth_cmpl.bda,
ESP_BD_ADDR_LEN);
            } else {
                ESP_LOGE(SPP_TAG, "authentication failed, status:%d", param-
>auth_cmpl.stat);
            }
            break;
        default:
            translateGAPEvent(SPP_TAG, event);
            break;
    }
}

void initSppInitiator(void) {

    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK( ret );

    ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_BLE));

    esp_bt_controller_config_t bt_cfg = BT_CONTROLLER_INIT_CONFIG_DEFAULT();
    if ((ret = esp_bt_controller_init(&bt_cfg)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize controller failed: %s\n", __func__,
esp_err_to_name(ret));
```

```
        return;
    }

    if ((ret = esp_bt_controller_enable(ESP_BT_MODE_BTDM)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluedroid_init()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize bluedroid failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluedroid_enable()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable bluedroid failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_gap_register_callback(esp_bt_gap_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s gap register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_spp_register_callback(esp_spp_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s spp register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_spp_init(esp_spp_mode)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s spp init failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }
}
```

spp_acceptor.h

```
#ifndef SPP_ACCEPTOR_H
#define SPP_ACCEPTOR_H

void initSppAcceptor(void);

#endif
```

spp_acceptor.c

```
#include <stdint.h>
#include <string.h>
#include <stdio.h>

#include "nvs.h"
#include "nvs_flash.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_bt.h"
#include "esp_bt_main.h"
#include "esp_gap_bt_api.h"
#include "esp_bt_device.h"
#include "esp_spp_api.h"
#include "time.h"
#include "sys/time.h"

#include "bluetooth.h"

#define SPP_TAG "SPP_ACCEPTOR"
#define SPP_SERVER_NAME "SPP_SERVER"
#define DEVICE_NAME "ESP_SPP_ACCEPTOR"

static const esp_spp_mode_t esp_spp_mode = ESP_SPP_MODE_CB;

static const esp_spp_sec_t sec_mask = ESP_SPP_SEC_AUTHENTICATE;
static const esp_spp_role_t role_slave = ESP_SPP_ROLE_SLAVE;

int handle; // provided by bluetooth.h
spp_data_rcv_cb_t dataCallback; // provided by bluetooth.h

static void esp_spp_cb(esp_spp_cb_event_t event, esp_spp_cb_param_t *param) {
    switch (event) {
        case ESP_SPP_INIT_EVT: // when SPP is initiated
            ESP_LOGI(SPP_TAG, "ESP_SPP_INIT_EVT");
            esp_bt_dev_set_device_name(DEVICE_NAME);
            esp_bt_gap_set_scan_mode(ESP_BT_CONNECTABLE,
ESP_BT_GENERAL_DISCOVERABLE);
            esp_spp_start_srv(sec_mask,role_slave, 0, SPP_SERVER_NAME);
            break;
        case ESP_SPP_WRITE_EVT: // SPP write operation completes (only for
ESP_SPP_MODE_CB)
            ESP_LOGI(SPP_TAG, "ESP_SPP_WRITE_EVT len=%d cong=%d", param-
>write.len, param->write.cong);
            break;
        case ESP_SPP_DATA_IND_EVT: // when SPP connection received data (only
for ESP_SPP_MODE_CB)
            ESP_LOGI(SPP_TAG, "ESP_SPP_DATA_IND_EVT len=%d", param-
>data_ind.len);
            dataCallback(param->data_ind.data, param->data_ind.len);
            // received data, store in queue
            // data = param->data_ind.data
            // len = param->data_ind.len
```

```
            break;
        case ESP_SPP_SRV_OPEN_EVT:
            ESP_LOGI(SPP_TAG, "ESP_SPP_SRV_OPEN_EVT");
            memcpy(&handle, &param->srv_open.handle, sizeof(handle));
            success();
            break;
        case ESP_SPP_CLOSE_EVT:
            ESP_LOGI(SPP_TAG, "ESP_SPP_CLOSE_EVT");
            failure();
            break;
        default:
            translateSPPEvent(SPP_TAG, event);
            break;
    }
}

void esp_bt_gap_cb(esp_bt_gap_cb_event_t event, esp_bt_gap_cb_param_t *param)
{
    switch (event) {
        default:
            translateGAPEvent(SPP_TAG, event);
            break;
    }
    return;
}

void initSppAcceptor(void) {

    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK( ret );
    ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_BLE));

    esp_bt_controller_config_t bt_cfg = BT_CONTROLLER_INIT_CONFIG_DEFAULT();
    if ((ret = esp_bt_controller_init(&bt_cfg)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_controller_enable(ESP_BT_MODE_BTDM)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluedroid_init()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize bluedroid failed: %s\n", __func__,
esp_err_to_name(ret));
```

```
        return;
    }

    if ((ret = esp_bluedroid_enable()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable bluedroid failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_gap_register_callback(esp_bt_gap_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s gap register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_spp_register_callback(esp_spp_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s spp register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_spp_init(esp_spp_mode)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s spp init failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }
}
```

network.h

```
#ifndef NETWORK_H
#define NETWORK_H

#define SET_PACKET 0x01
#define DELTA_PACKET 0x02
#define CONFIG_PACKET 0x03

typedef struct __attribute__((__packed__)) {
        uint32_t len;
        uint8_t flag;
} Header;

typedef struct __attribute__((__packed__)) {
        int32_t delta;
} DeltaPacket;

typedef struct __attribute__((__packed__)) {
        uint32_t timePeriod;
} SetPacket;

typedef struct __attribute__((__packed__)) {
        int powered;
```

```
} ConfigPacket;

void writeDeltaPacket(int32_t delta);
void writeSetPacket(uint32_t timePeriod);
void writeConfigPacket(int powered);
void stripHeader(uint8_t *data, int len, Header *header);
void stripPacket(uint8_t *data, int len, void *packet);

#endif
```

network.c

```
#include <string.h>

#include "bluetooth.h"
#include "network.h"

void sendPacket(Header *header, void *packet, int len) {
        uint8_t data[SPP_DATA_LEN] = {0};
        int offset = 0;

        memcpy(data, (uint8_t *) header, sizeof(Header));
        offset += sizeof(Header);

        memcpy(data+offset, (uint8_t *) packet, len);

        writeData(data, sizeof(Header) + len);
}

void writeDeltaPacket(int32_t delta) {
        Header header;
        header.len = sizeof(Header) + sizeof(DeltaPacket);
        header.flag = DELTA_PACKET;

        DeltaPacket packet;
        packet.delta = delta;

        sendPacket(&header, &packet, sizeof(DeltaPacket));
}

void writeSetPacket(uint32_t timePeriod) {
        Header header;
        header.len = sizeof(Header) + sizeof(SetPacket);
        header.flag = SET_PACKET;

        SetPacket packet;
        packet.timePeriod = timePeriod;

        sendPacket(&header, &packet, sizeof(SetPacket));
}

void writeConfigPacket(int powered) {
```

```
        Header header;
        header.len = sizeof(Header) + sizeof(ConfigPacket);
        header.flag = CONFIG_PACKET;

        ConfigPacket packet;
        packet.powered = powered;

        sendPacket(&header, &packet, sizeof(ConfigPacket));
}

void stripHeader(uint8_t *data, int len, Header *header) {
        memcpy(header, (Header *) data, len);
        memcpy(data, data+sizeof(Header), len-sizeof(Header));
}

void stripPacket(uint8_t *data, int len, void *packet) {
        memcpy(packet, (void *) data, len);
}
```

Bibliography

[1] Straker, Leon, et al. “Conflicting Guidelines on Young Children’s Screen Time and Use of Digital Technology Create Policy and Practice Dilemmas.” *The Journal of Pediatrics*, vol. 202, 2018, pp. 300–303.

[2] Dalene, Knut Eirik, et al. “Cross-Sectional and Prospective Associations between Sleep, Screen Time, Active School Travel, Sports/Exercise Participation and Physical Activity in Children and Adolescents.” *BMC Public Health*, vol. 18, no. 1, July 2018.

[3] Lanningham-Foster, L., et al. “Energy Expenditure of Sedentary Screen Time Compared With Active Screen Time for Children.” *Pediatrics*, vol. 118, no. 6, 2006.