CPE403 – Advanced Embedded Systems

Design Assignment 4

Name: Samuel McCormick

Email: samuel.mccormick@gmail.com

Github Repository link (root): https://github.com/brokenboredom/tech-muffin

Youtube Playlist link (root):
https://www.youtube.com/playlist?list=PLCSfNPhZUD_NZxU7Tsm_bMTwxiQqxqpBr

Follow the submission guideline to be awarded points for this Assignment.

Submit the following for all Assignments:

1. In the document, for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only.
2. Create a private Github repository with a random name (no CPE403/603/710, Lastname, Firstname). Place all labs under the root folder MSP432E4/CC1352, sub-folder named Assignment1, with one document and one video link file for each lab, place modified c files named as asng_taskxx.c.
3. If multiple c files or other libraries are used, create a folder asng1_t01 and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup_ccs.c and other include files, c) text file with youtube video links (see template).
5. Submit the doc file in canvas before the due date. The root folder of the github assignment directory should have the documentation and the text file with youtube video links.
6. Organize your youtube videos as playlist under the name “ADVEMBSYS”. The playlist should have the video sequence arranged as submission or due dates.
7. Only submit pdf documents. Do not forget to upload this document in the github repository and in the canvas submission portal.

I understand the Student Academic Misconduct Policy -
http://studentconduct.unlv.edu/misconduct/policy.html

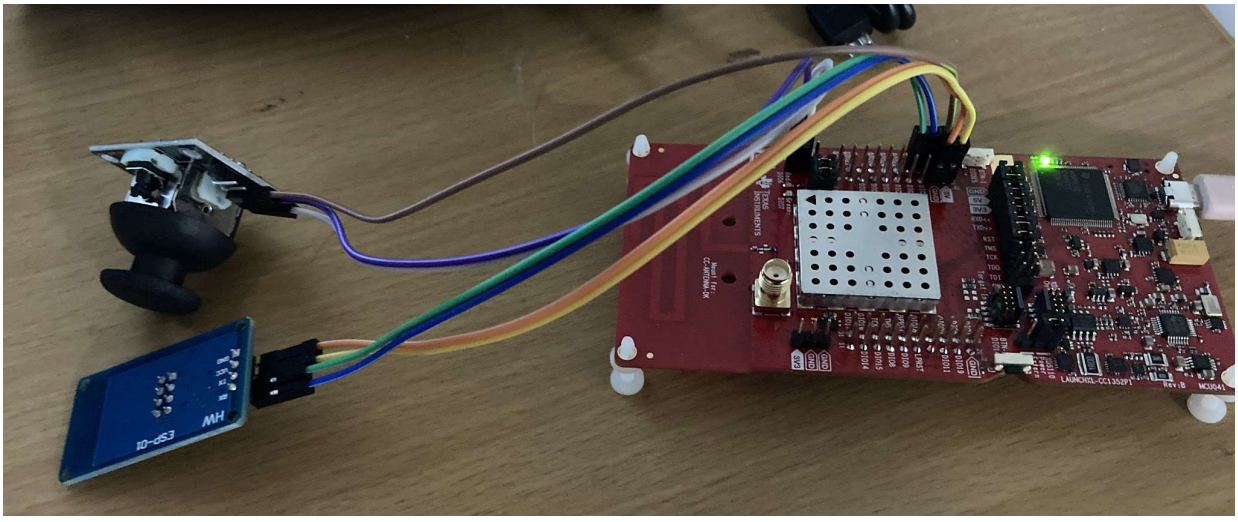
“This assignment submission is my own, original work”.

Samuel McCormick

Goal of this assignment is to transmit/receive a series of data from the CC1352 to a mobile or PC using BLE technology. The data to be transmitted include the following: 1) Both button status (one service - two characteristics), 2) LED turn on/off service (one service – two characteristics), and 3) ADC data from joystick (one service – two characteristics). The ADC data must be a data from the joystick (use both channels). Use the examples provided in the examples a) project_zero, simple_peripheral, and simple_peripheral_gatt_builder_preview. Also, review the BLE materials in CC13XX CC26XX SimpleLink Academy (especially the Bluetooth low energy Custom Profile) before completing the assignment. The data can be displayed or controlled using an open BLE scanner/app or using Web Bluetooth. All tasks working should be shown in the video.

Encountered Problems

During the course of this project my board became unresponsive with the iOS app, bTool and all other bluetooth connections. When running *project_zero* the board would advertise as Project Zero but the apps would see it as *Simple Peripheral* with a "0" icon and would not recognize any services or connect to the device. I tried Uniflash to restore the factory default but that didn't solve the problem. I tried overwriting the projects with new tutorial examples but nothing would connect to my board anymore. I followed the training and provided tutorial videos as best as I could without any way to actively test the board and as such I have no serviceable demo videos for this project.



CC1352P1 connected to UART (DIO12/13) and Joystick (DIO23).

Task 1

This task was sourced from the *project_zero* example which comes pre-loaded on the board and I believe no code needed changed in order to meet the project guidelines. Below are the header declarations for the button_service and the two characteristics and their attributes.

```
// Service UUID
#define BUTTON_SERVICE_SERV_UUID 0x1120
#define BUTTON_SERVICE_SERV_UUID_BASE128(uuid) 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0xB0, 0x00, 0x40, 0x51, 0x04, LO_UINT16(uuid), HI_UINT16(uuid), \
  0x00, 0xF0

// BUTTON0 Characteristic defines
#define BS_BUTTON0_ID             0
#define BS_BUTTON0_UUID           0x1121
#define BS_BUTTON0_UUID_BASE128(uuid) 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0xB0, 0x00, 0x40, 0x51, 0x04, LO_UINT16(uuid), HI_UINT16(uuid), 0x00, 0xF0
#define BS_BUTTON0_LEN            1
#define BS_BUTTON0_LEN_MIN        1

// BUTTON1 Characteristic defines
#define BS_BUTTON1_ID             1
#define BS_BUTTON1_UUID           0x1122
#define BS_BUTTON1_UUID_BASE128(uuid) 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0xB0, 0x00, 0x40, 0x51, 0x04, LO_UINT16(uuid), HI_UINT16(uuid), 0x00, 0xF0
#define BS_BUTTON1_LEN            1
#define BS_BUTTON1_LEN_MIN        1
```

Task 2

The LED turn on/off service was also present out-of-the-box on *project_zero* and was not altered. As seen below, the led_service contains once service made up of two characteristics, which are in-turn made up of attributes.

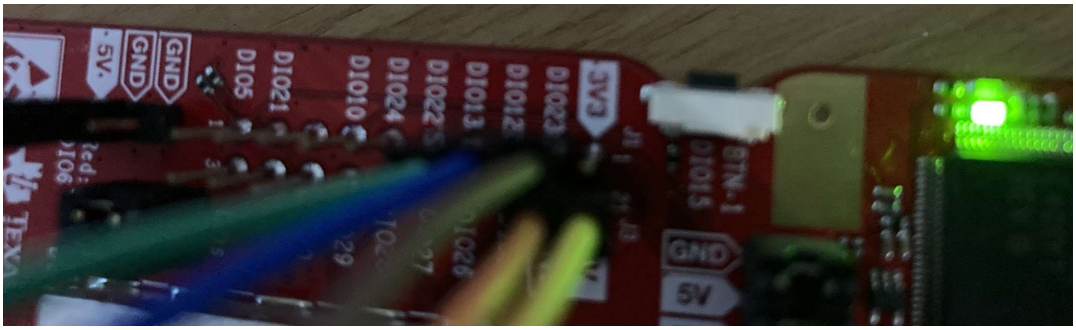
```
// Service UUID
#define LED_SERVICE_SERV_UUID 0x1110
#define LED_SERVICE_SERV_UUID_BASE128(uuid) 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
   0x00, 0xB0, 0x00, 0x40, 0x51, 0x04, LO_UINT16(uuid), HI_UINT16(uuid), 0x00, \
   0xF0

// LED0 Characteristic defines
#define LS_LED0_ID              0
#define LS_LED0_UUID            0x1111
#define LS_LED0_UUID_BASE128(uuid) 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
   0xB0, 0x00, 0x40, 0x51, 0x04, LO_UINT16(uuid), HI_UINT16(uuid), 0x00, 0xF0
#define LS_LED0_LEN             1
#define LS_LED0_LEN_MIN         1

// LED1 Characteristic defines
#define LS_LED1_ID              1
#define LS_LED1_UUID            0x1112
#define LS_LED1_UUID_BASE128(uuid) 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
   0xB0, 0x00, 0x40, 0x51, 0x04, LO_UINT16(uuid), HI_UINT16(uuid), 0x00, 0xF0
#define LS_LED1_LEN             1
#define LS_LED1_LEN_MIN         1
```

Task 3

ADC data was captured by altering the data_service from *project_zero* to include a notify permission in the *Data_Service_ReadAttrCB()* function. The code was also altered to initialize the ADC and capture data during that function. Where it was thereby converted to a string using *snprintf()* and fed to the data array.



Wires from Joystick and UART Tx/Rx connected to DIO23/12/13 respectively.

ADC initialization, conversion, and string formatting:

```
static bStatus_t Data_Service_ReadAttrCB(uint16_t connHandle,
...
   ADC_Handle adc;
   ADC_Params params;
   int_fast16_t res;
   ADC_init();
   ADC_Params_init(&params);
   adc = ADC_open(CONFIG_ADC_0, &params);
   if (adc == NULL)
   {
      while (1)
         ;
   }
...
else
   {
      *pLen = 5;                    //Data length
      char int_str[5];               // Data string
     res = ADC_convert(adc, &adcValue0);  //Run conversion
     if (res == ADC_STATUS_SUCCESS)      //Check for completion
     {
      sprintf(int_str, "%d", adcValue0);          //Convert int to string
      //printf("adcValue0: %s\n", int_str);       //Debug: Display adc string to console

     }
      *pLen = MIN(maxLen, valueLen - offset);     // Transmit as much as possible
      memcpy(pValue, int_str + offset, *pLen);    //Copy our string data array instead of a single char
      ADC_close(adc);                          //Close ADC
   }
```

Youtube Playlist: https://www.youtube.com/playlist?list=PLCSfNPhZUD_NZxU7Tsm_bMTwxiQqxqpBr