TI DSP, MCU 및 Xilinx Zynq FPGA   
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)

[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생 – 문한나

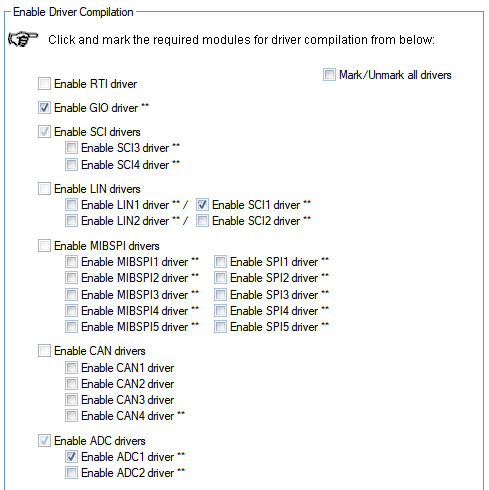
[mhn97@naver.com](mailto:mhn97@naver.com)

학생 – 장성환

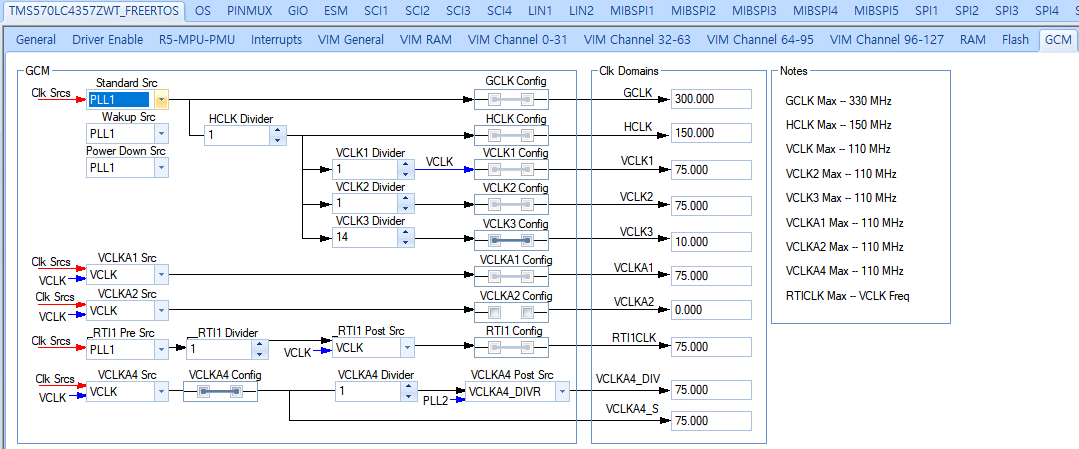
[redmk1025@gmail.com](mailto:redmk1025@gmail.com)

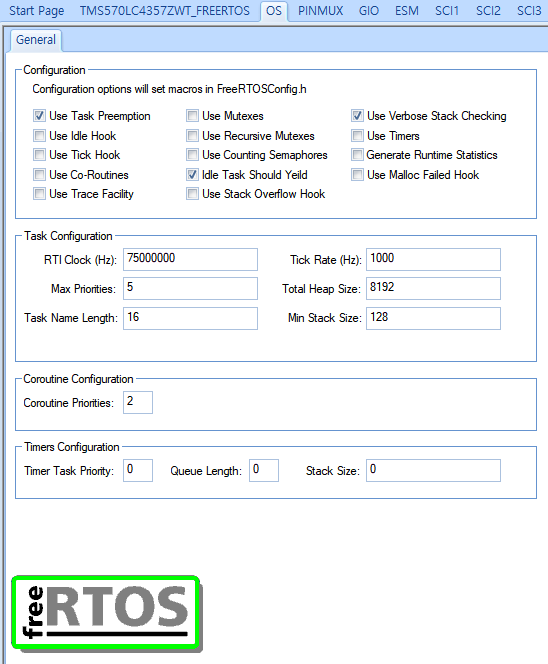
조도센서를 활용한 수동주행

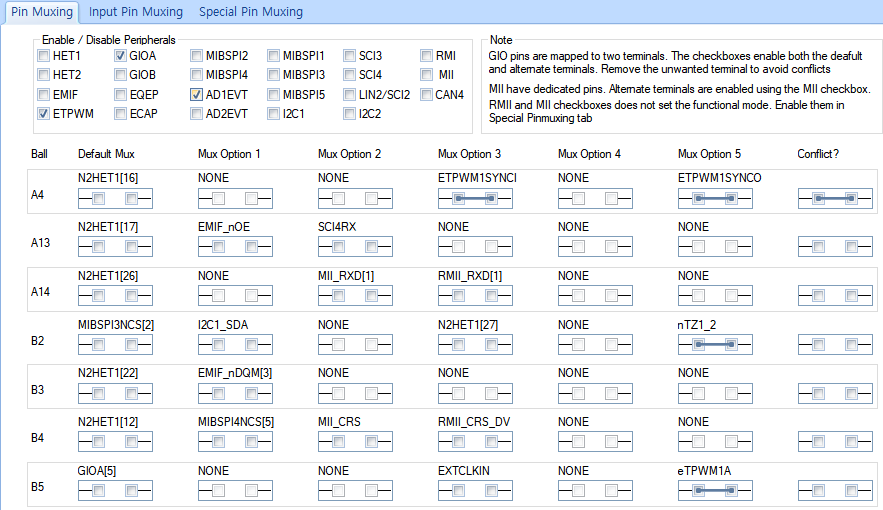
**HCG 설정**

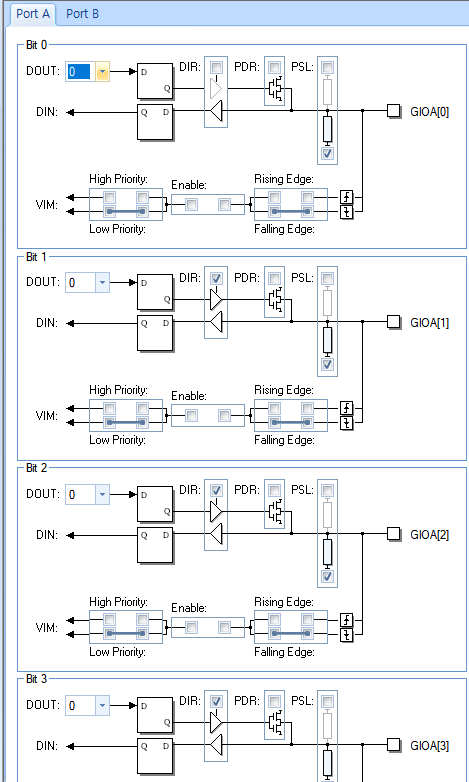
****

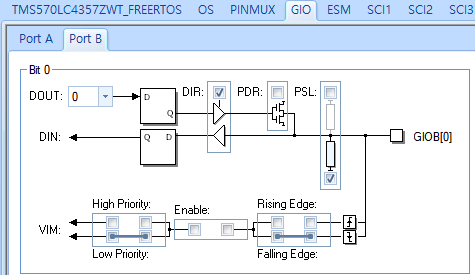
****

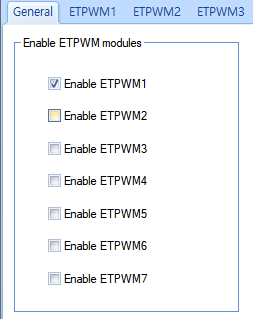
****

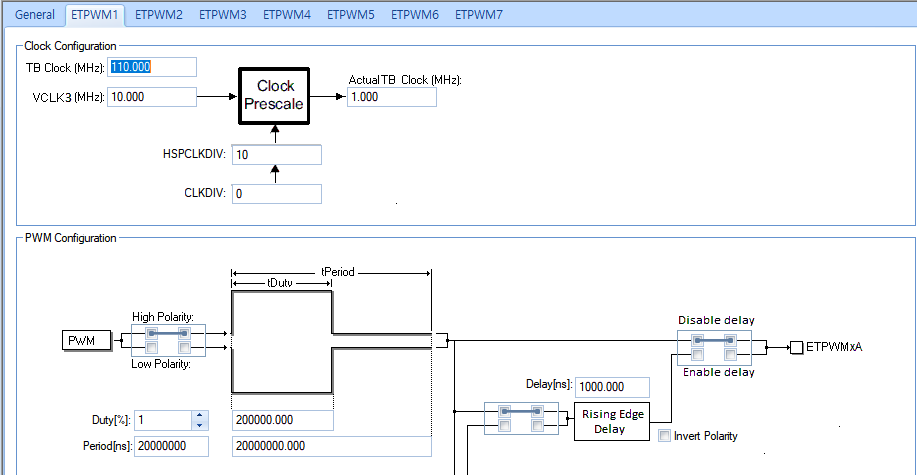
****

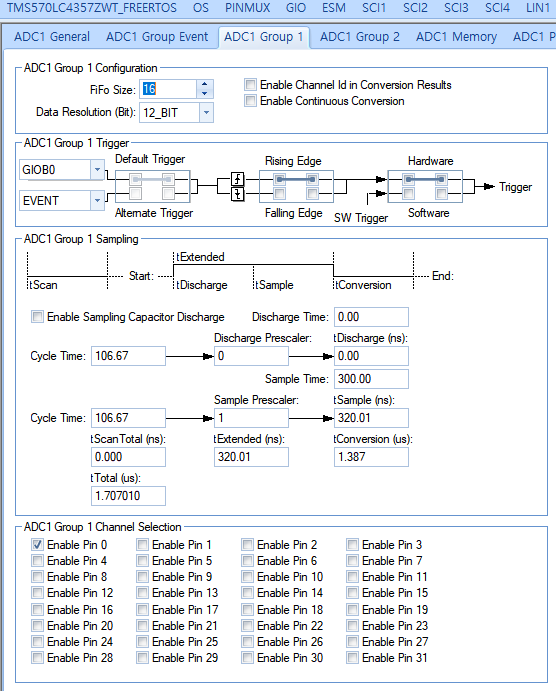
****

****

****

****

****

****

**CCS 코드**

/\* Include Files \*/

**#include** <include/FreeRTOS.h>

**#include** <include/FreeRTOSConfig.h>

**#include** <include/HL\_adc.h>

**#include** <include/HL\_etpwm.h>

**#include** <include/HL\_gio.h>

**#include** <include/HL\_hal\_stdtypes.h>

**#include** <include/HL\_reg\_etpwm.h>

**#include** <include/HL\_reg\_gio.h>

**#include** <include/HL\_reg\_sci.h>

**#include** <include/HL\_sci.h>

**#include** <include/os\_mpu\_wrappers.h>

**#include** <include/os\_portmacro.h>

**#include** <include/os\_projdefs.h>

**#include** <include/os\_queue.h>

**#include** <include/os\_semphr.h>

**#include** <include/os\_task.h>

**#include** <stdlib.h>

**#include** <string.h>

xTaskHandle xTask1Handle;

xTaskHandle xTask2Handle;

xTaskHandle xTask3Handle;

xTaskHandle xTask4Handle;

xTaskHandle xTask5Handle;

QueueHandle\_t mutex = NULL;

**void** **vTask1**(**void**\* pvParameters);

**void** **vTask2**(**void**\* pvParameters);

**void** **vTask3**(**void**\* pvParameters);

**void** **vTask4**(**void**\* pvParameters);

**void** **vTask5**(**void**\* pvParameters);

**#define** MAX 10

uint32 fir[10] = { 0, };

uint32 ave = 0;

uint8 x[32] = { 0, };

**char** flag = 0;

**int** i;

adcData\_t data;

uint8 msg[32] = { 0, };

**void** **send\_data**(sciBASE\_t \*sci, uint8\* byte, uint32 length)

{

**int** i;

**for** (i = 0; i < length; i++)

sciSendByte(sciREG1, byte[i]);

sciSendByte(sciREG1, '\r');

sciSendByte(sciREG1, '\n');

}

**void** **delay**(uint32 delay)

{

**int** i;

**for** (i = 0; i < delay; i++)

;

}

**int** **main**(**void**)

{

gioInit();

sciInit();

adcInit();

etpwmInit();

etpwmStartTBCLK();

gioSetDirection(gioPORTA, 0xE);

gioSetPort(gioPORTA, 0xffffffff);

gioSetBit(gioPORTA, 0, 0);

gioSetBit(gioPORTA, 1, 1);

gioSetBit(gioPORTA, 2, 1);

gioSetBit(gioPORTA, 3, 1);

gioSetBit(gioPORTB, 0, 0);

adcStartConversion(adcREG1, adcGROUP1);

etpwmREG1->CMPA = 1500; // 정지

vSemaphoreCreateBinary(mutex)

**if** (xTaskCreate(vTask1, "Task1", configMINIMAL\_STACK\_SIZE, NULL, 1,

&xTask1Handle) != pdTRUE)

{

**while** (1)

;

}

**if** (xTaskCreate(vTask2, "Task2", configMINIMAL\_STACK\_SIZE, NULL, 1,

&xTask2Handle) != pdTRUE)

{

**while** (1)

;

}

**if** (xTaskCreate(vTask3, "Task3", configMINIMAL\_STACK\_SIZE, NULL, 1,

&xTask3Handle) != pdTRUE)

{

**while** (1)

;

}

**if** (xTaskCreate(vTask4, "Task4", configMINIMAL\_STACK\_SIZE, NULL, 1,

&xTask3Handle) != pdTRUE)

{

**while** (1)

;

}

**if** (xTaskCreate(vTask5, "Task5", configMINIMAL\_STACK\_SIZE, NULL, 1,

&xTask3Handle) != pdTRUE)

{

**while** (1)

;

}

vTaskStartScheduler();

**while** (1)

;

**return** 0;

}

**void** **vTask1**(**void** \*pbParameters)

{

**while** (1)

{

//delay(10000000);

**if** (flag == 1)

{

**if** (xSemaphoreTake(mutex, ( TickType\_t ) 10) == pdTRUE)

{

**for** (i = 0; i < 10; i++)

{

gioSetBit(gioPORTB, 0, 1);

**while** (adcIsConversionComplete(adcREG1, adcGROUP1) == 0)

;

adcGetData(adcREG1, adcGROUP1, &data);

fir[i] = data.value;

gioSetBit(gioPORTB, 0, 0);

}

//필터 생성

ave = (fir[0] + fir[1] + fir[2] + fir[3] + fir[4] + fir[5]

+ fir[6] + fir[7] + fir[8] + fir[9]) / MAX;

//필터값 출력

**ltoa**(ave, x);

send\_data(sciREG1, x, **strlen**(x));

gioSetBit(gioPORTB, 0, 1);

**while** (adcIsConversionComplete(adcREG1, adcGROUP1) == 0)

;

adcGetData(adcREG1, adcGROUP1, &data);

**ltoa**(data.value, x);

//현재값 출력

send\_data(sciREG1, x, **strlen**(x));

gioSetBit(gioPORTB, 0, 0);

flag = 5;

//vTaskDelay(portMAX\_DELAY);

xSemaphoreGive(mutex);

vTaskDelay(10);

}

**else**

{ //키를 받아오지 못하는 경우에 실행 코드를 작성

//flag = 1;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

}

}

}

**void** **vTask2**(**void** \*pbParameters)

{

**while** (1)

{

**if** (flag == 2)

{

**if** (xSemaphoreTake(mutex, ( TickType\_t ) 10) == pdTRUE)

{

//현재값

gioSetBit(gioPORTB, 0, 1);

**while** (adcIsConversionComplete(adcREG1, adcGROUP1) == 0)

;

adcGetData(adcREG1, adcGROUP1, &data);

**ltoa**(data.value, x);

//현재값 출력

send\_data(sciREG1, x, **strlen**(x));

gioSetBit(gioPORTB, 0, 0);

**if** ((data.value - ave) > 0 && (data.value - ave) <= 200)

{

flag = 3;

}

**else** **if** ((ave - data.value) > 200)

{

flag = 4;

}

//vTaskDelay(portMAX\_DELAY);

xSemaphoreGive(mutex);

vTaskDelay(10);

}

**else**

{ //키를 받아오지 못하는 경우에 실행 코드를 작성

//flag = 1;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

}

}

}

**void** **vTask3**(**void** \*pbParameters)

{

**while** (1)

{

**if** (flag == 3)

{

**if** (xSemaphoreTake(mutex, ( TickType\_t ) 10) == pdTRUE)

{

etpwmREG1->CMPA -= 1;

flag = 0;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

**else**

{ //키를 받아오지 못하는 경우에 실행 코드를 작성

//flag = 1;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

}

**else** **if** (flag == 4)

{

**if** (xSemaphoreTake(mutex, ( TickType\_t ) 10) == pdTRUE)

{

etpwmREG1->CMPA -= 2;

flag = 0;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

**else**

{ //키를 받아오지 못하는 경우에 실행 코드를 작성

//flag = 1;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

}

}

}

**void** **vTask4**(**void** \*pbParameters)

{

**while** (1)

{

**if** (flag == 5)

{

**if** (xSemaphoreTake(mutex, ( TickType\_t ) 10) == pdTRUE)

{

etpwmREG1->CMPA += 5;

**if** ((etpwmREG1->CMPA) >= 1500)

{

etpwmREG1->CMPA = 1500;

}

flag = 0;

xSemaphoreGive(mutex);

vTaskDelay(10);

}

**else**

{

xSemaphoreGive(mutex);

vTaskDelay(10);

}

}

}

}

**void** **vTask5**(**void** \*pbParameters)

{

**while** (1)

{

**if** (flag == 0)

{

**if** (xSemaphoreTake(mutex, ( TickType\_t ) 10) == pdTRUE)

{

**if** (gioGetBit(gioPORTA, 0) == 0)

{

flag = 1;

vTaskDelay(10);

}

**else** **if** (gioGetBit(gioPORTA, 0) == 1)

{

flag = 2;

vTaskDelay(10);

}

xSemaphoreGive(mutex);

vTaskDelay(10);

}

**else**

{

xSemaphoreGive(mutex);

vTaskDelay(10);

}

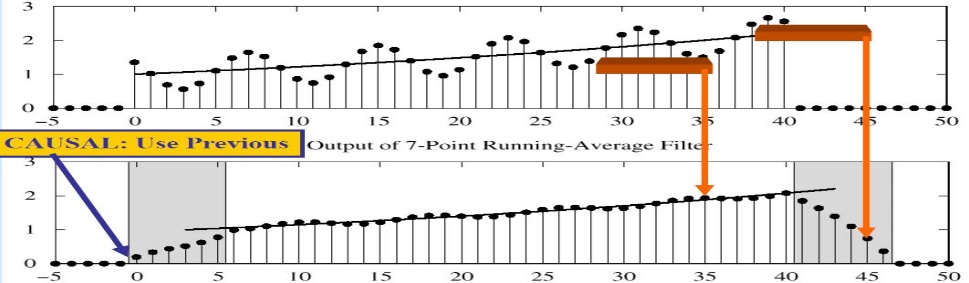
}

}

}

ADC 의 조도 센서는 주변환경의 영향을 많이 받는다. 밝은 곳으로 가면 기본 ADC 값이 높은 상태가 되어 버리고 어두운 곳으로 가면 기본 ADC 값이 낮은 상태가 된다.

최대한 ACC 페달 주변에 빛의 영향을 받지 않도록 밀폐 시켰지만, 장소에 따라 값의 변화가 커서 어느 ADC 값의 기준점을 가지고 알고리즘을 쓸 수 없었다.

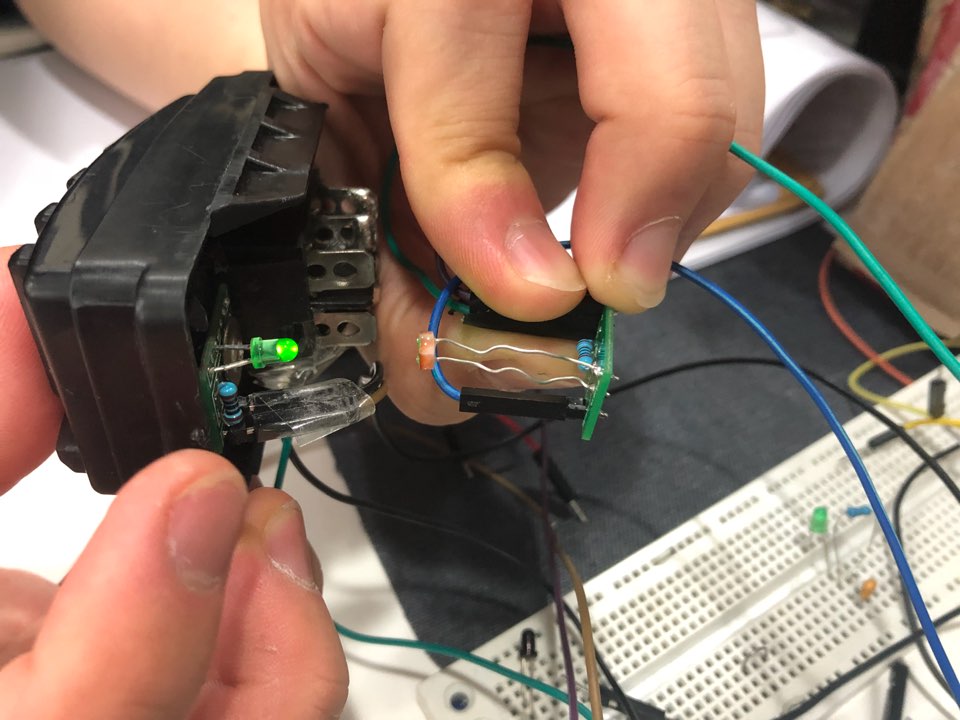
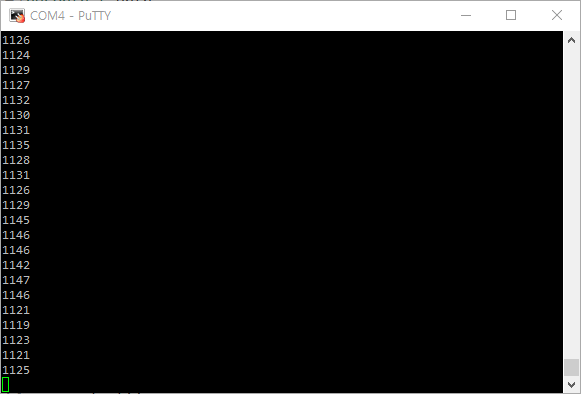


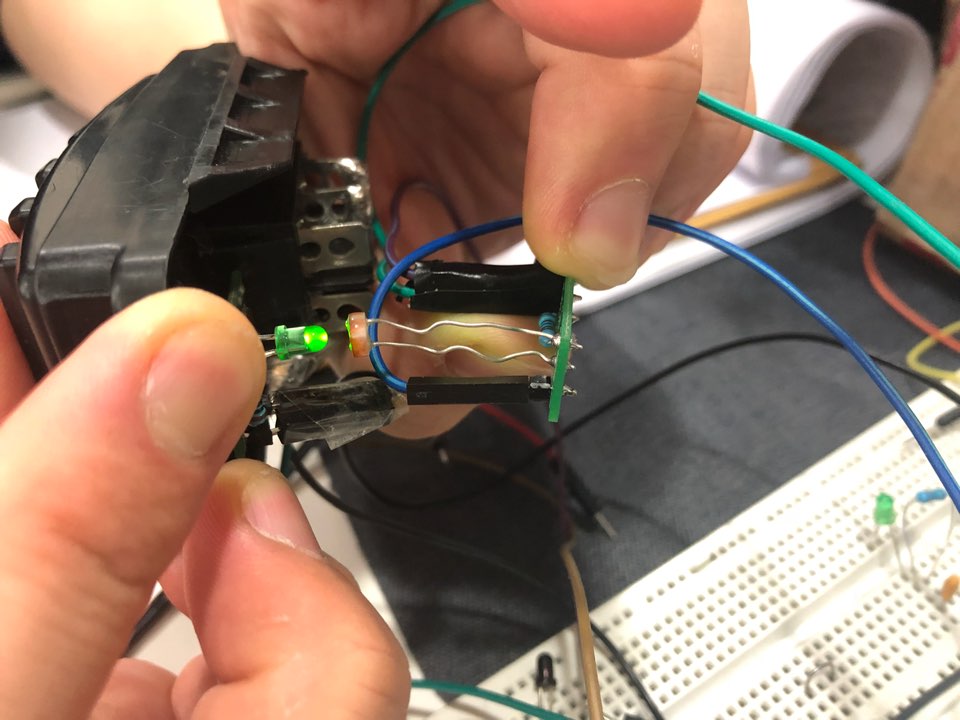
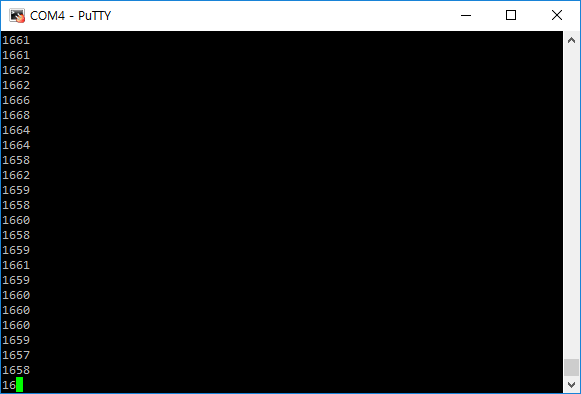
<FIR 필터>

위 그림처럼 노이즈가 있는 상황에서 피드백이 없는 시스템의 경우 FIR 필터를 사용하면 좋은 결과를 얻을 수 있다.

FIR 필터를 사용하여 현재 상태의 ADC 값을 저장해 두고, 현재 상태의 값에서 변동이 생기는 크기에 따라서 주행 가속도가 결정되는 방식을 사용하니 안정적으로 동작하였다.

**결과**

** **

** **