임베디드 시스템 설계 및 실험 보고서

5 주차 실험_Poling 방식을 이용한 UART 통신 및 Clock control

분반 : 001 분반

교수님 : 정 상화 교수님

조교님 : 유 동화 조교님

실험일 : 2019-09-30

제출일 : 2019-10-07

00. 목차

7 조

장 수현

박 창조

임 다영

이 힘찬

01. 실험 목적

- Poling 방식을 이용한 UART 통신의 이해
- Clock control 을 이용하여 System clock 으로 목적 clock 을 생성

02. 실험 과제

2.1 주 과제

UART 통신을 이용하여 Putty 로 문자열 출력

2.2 세부 과제

- 개발 환경 구축
- DS-5 에서 프로젝트 생성 및 설정
- DB 파일, 라이브러리, scatter 파일, flashclear 파일을 프로젝트 폴더 안으로 복사
- MCO 를 통해 오실로스코프를 사용하여 Clock 확인
- Reference Manual 을 참고하여 AFIO 설정 및 USART DIV 계산

03. 실험 준비

3.1 실험에 필요한 기초지식

3.1.1 Poling

- 특정 주기를 가지고 주기마다 처리를 위한 시그널이 들어왔는지 체크하는 방식
- 커널과 같은 Interrupt handler 가 필요하지 않다.
- Interrupt 에 비해 구현이 쉽지만 시스템의 리소스를 많이 차지하여 성능 저하의 원인이 되기도 한다.

3.1.2 Clock 과 Clock tree

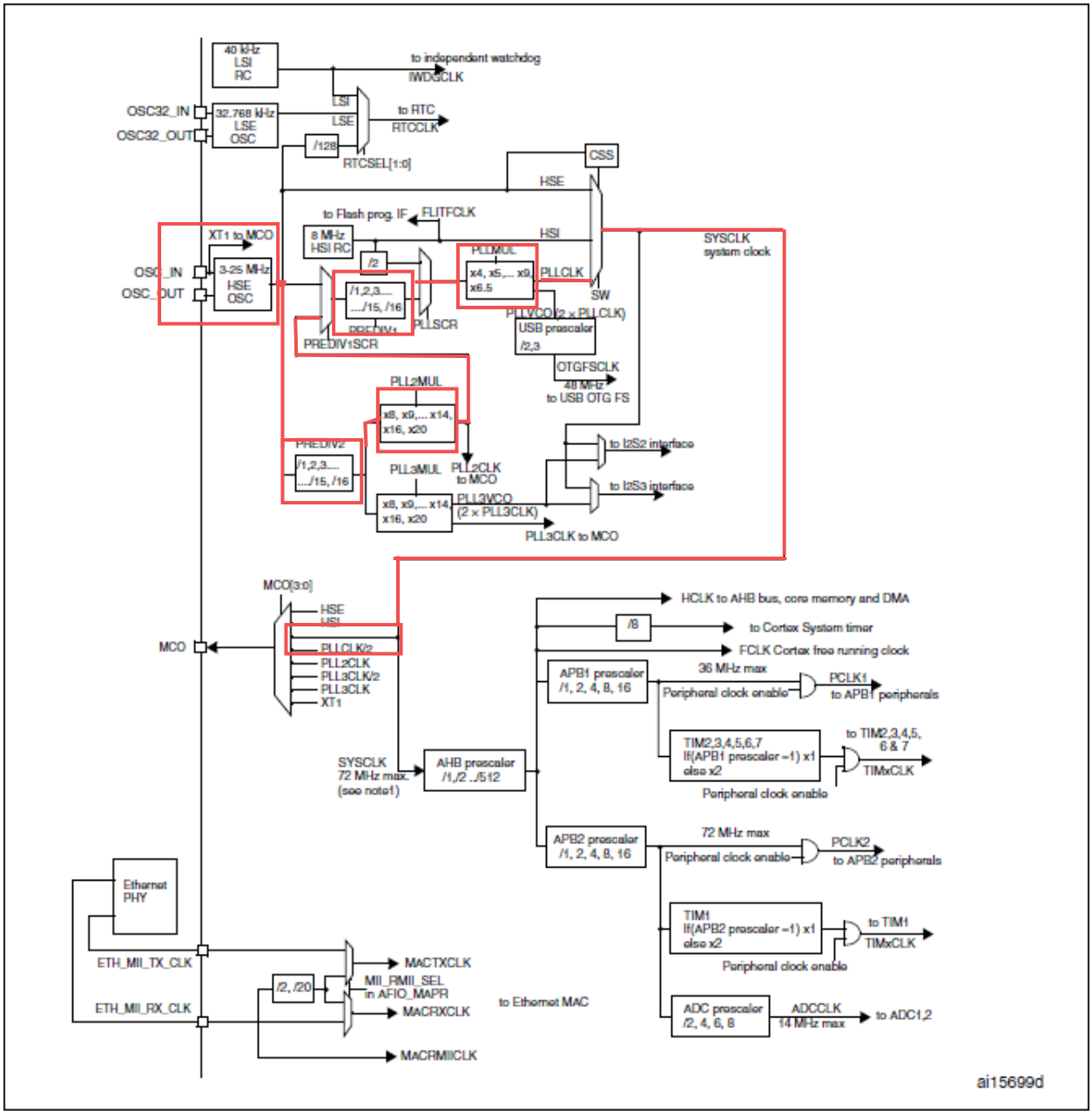
1. Clock
   - HSE : 외부에서 제공하는 Clock, 25Mhz 의 값을 가진다.
   - HSI : 내부에서 제공하는 Clock
   - PLL : Clock 을 증폭, 내부와 외부 Clock 선택 후 PLL 을 통해 주파수 조정

2. MCO(Microcontroller Clock Output)

- STM 내부에서 사용되는 clock 을 외부로 출력한다.
- MCO 핀으로 출력할 때 GPIO 최대 속도를 넘지 않도록 주의해야 한다.
- 오실로스코프와 연결하여 clock 설정이 정상적인지 확인할 수 있고 어떤 내부 clock 을 외부로 출력할지 결정한다.

3. Clock tree

- FCLK : CPU 에 사용
- HCLK : AHB BUS 에 사용, 고속 입출력 장치에 사용
- PCLK : APB BUS 에 사용, 저속 입출력 장치에 사용



3.1.3 UART/USART

1. UART

- 범용 비동기화 송수신기
- 병렬 데이터를 직렬 형식으로 전환하여 데이터를 전송하는 컴퓨터 하드웨어의 일종
- 시리얼 기반 통신으로 RS_232 를 통해 통신을 지원

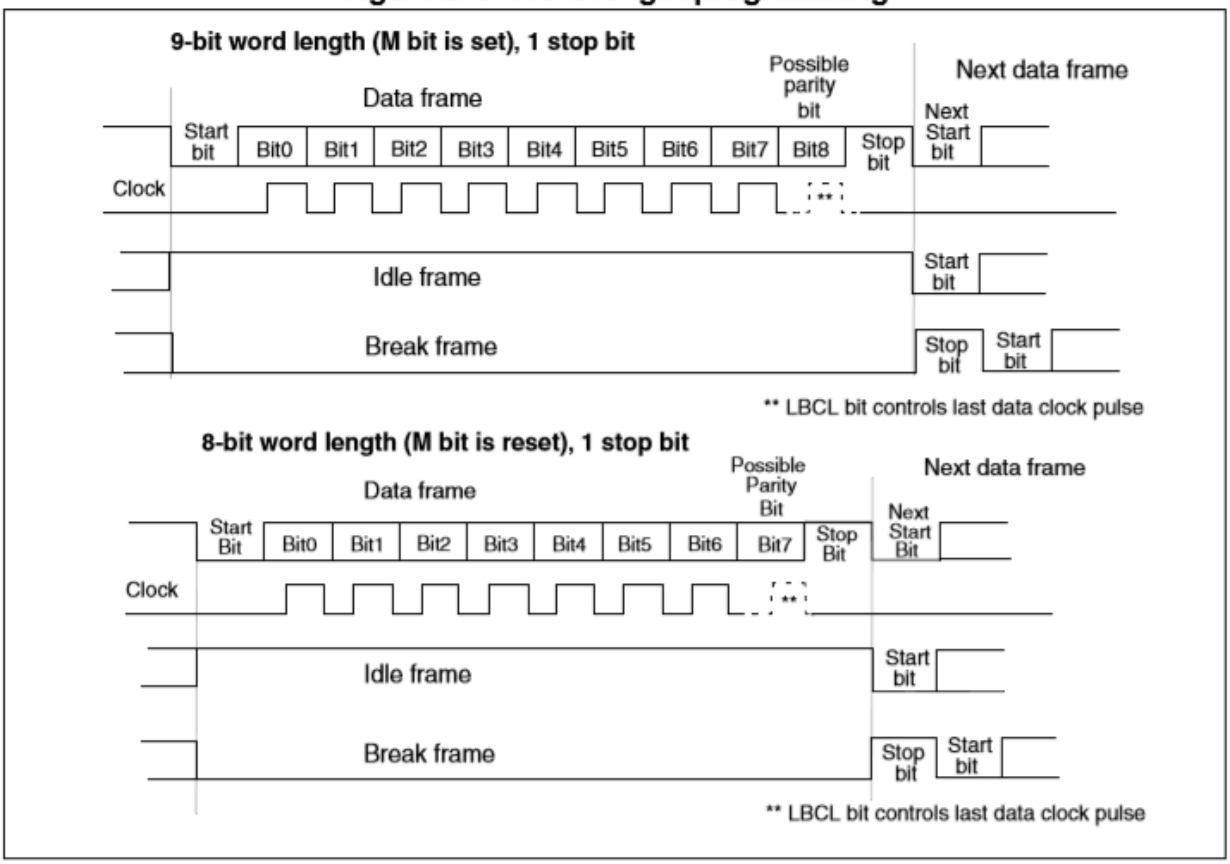
2. USART

- 범용 동기화 송수신기
- 동기화 통신까지 지원하는 UART

3. Data frame

- Start bits : 통신의 시작을 의미하는 것으로 0 으로 설정
- Data bits : 송/수신되는 데이터를 8-9bit 으로 나타낸다.
- Parity bits : 오류 검증을 위한 값으로 레지스터 설정에 따라 짝/홀/사용안함 으로 선택
- Stop bits : 통신 종료를 의미하는 것으로 레지스터 설정에 따라 비트 수가 나뉜다.
- Baud rate : 초당 얼마나 많은 심볼을 전송할 수 있는가를 표현, 초당 신호(signal)요소의 수

Figure 279. Word length programming



3.2 실험 진행 시 주의사항

- UART 를 쓰기 위해 GPIO(general purpose I/O)를 AFIO(Alternate function I/O)로 설정해야 한다.
- 목적 clock 을 만들기 위한 연산을 수행해야 한다.
- MCO 핀으로 출력할 때 GPIO 최대 속도를 넘지 않도록 주의해야 한다.
- &= ~()의 의미는 사용할 bit 만을 0 으로 초기화 하겠다는 의미이다.
- 정확한 실험결과를 얻기 위해 RCC 리셋을 습관화해야 한다.

3.3 환경 설정

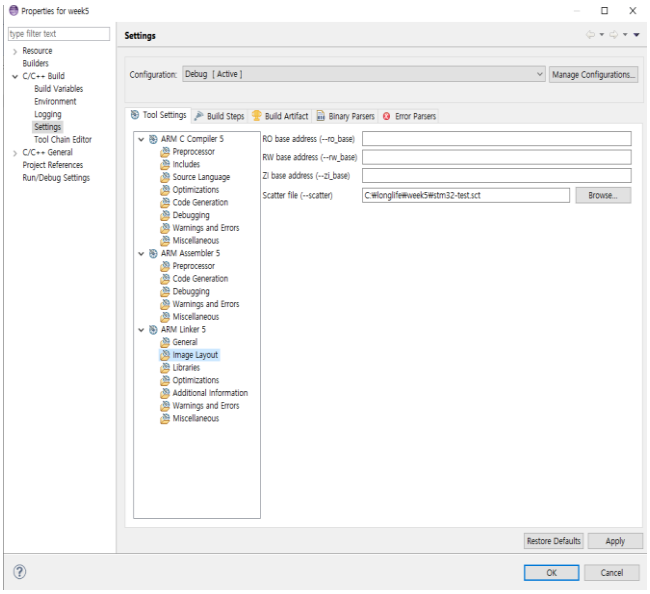
실험을 진행하기 전 아래의 프로젝트 생성 및 기본 환경 설정을 해주어야 한다.

3.2.1 프로젝트 생성

C 언어로 소스코드를 작성하기 때문에 C++프로젝트가 아닌 C 프로젝트를 생성해준다. Project type 은 Bare-metal Executable -〉 Empty Project 로, Toolchains 은 Arm Compiler 5 로 설정한다.

3.2.2 프로젝트 Properties-Settings

4 주차 실험과 동일하게 스캐터 파일을 이용한다.



3.2.3 보드 연결

보드 연결 시에는 반드시 연결 순서를 지키고, 규격에 맞는 전원선을 사용해야한다.

연결 순서 : 보드와 Dstream JTAG 연결 -〉 보드전원선 연결(보드 전원은 OFF) -〉 Dstream 전원 연결 및 ON -〉 Dstream Status LED 점등 확인 후 보드 전원 ON -〉 Dstream Target LED 점등 확인 후 DS-5 에서 'connect target'

3.2.4 데이터 베이스 설정

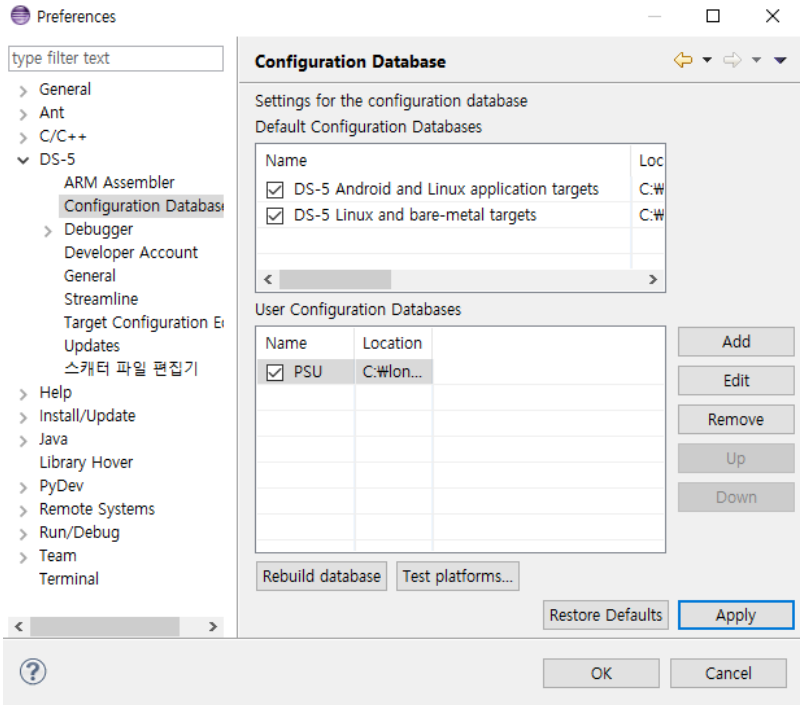
Dstream 를 USB 로 컴퓨터에 연결하고 Debug Hardware config 에서 보드를 connect 한다. 보드 연결 후에는 Auto Configure 를 클릭하여 설정을 진행하고, rvc 파일로 저장한다. rvc 파일 저장 경로에 한글이 들어가지 않도록 주의한다.

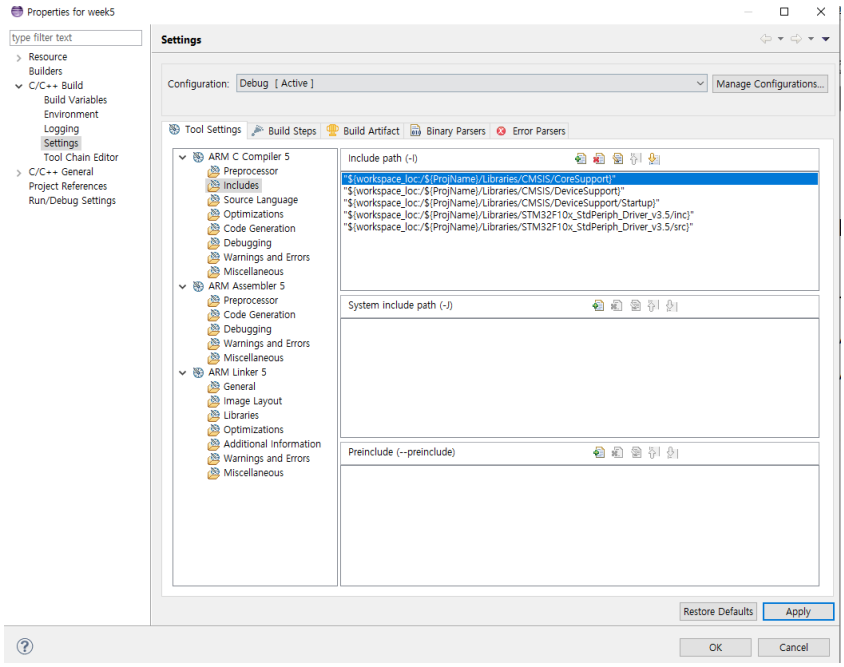
3.2.5 cdbimporter

DS-5 Command Prompt 에서 RVC 파일이 있는 폴더로 이동 후 아래와 같이 명령문을 작성한다.

3.2.6 데이터 베이스 등록 및 라이브러리 추가

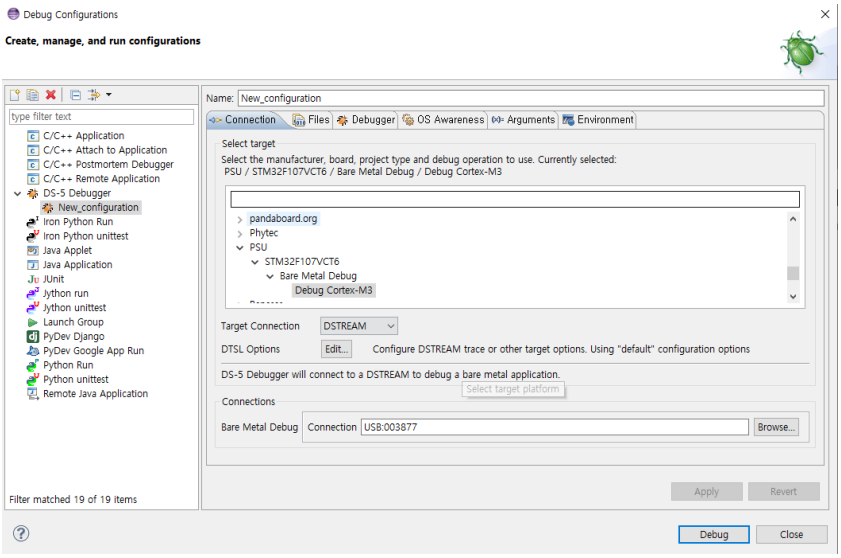
아래와 같이 데이터 베이스 등록 후 프로젝트 폴더에 라이브러리를 추가한다.

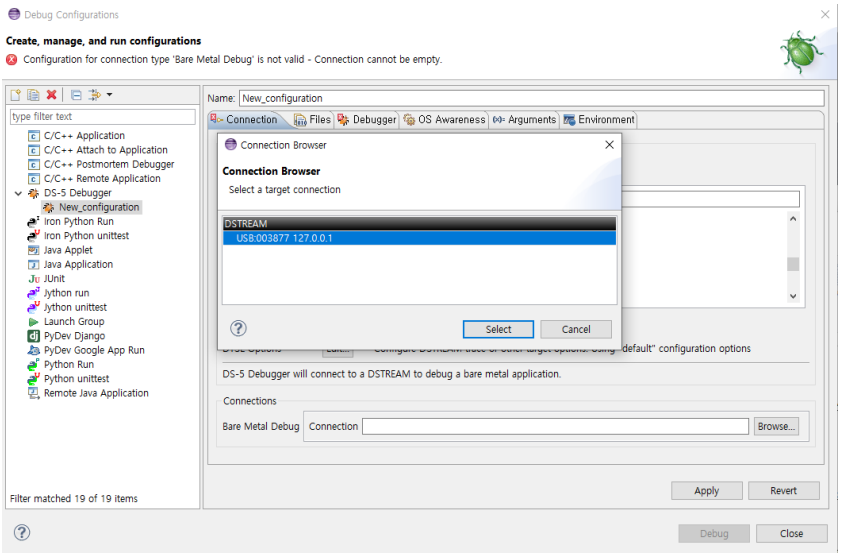


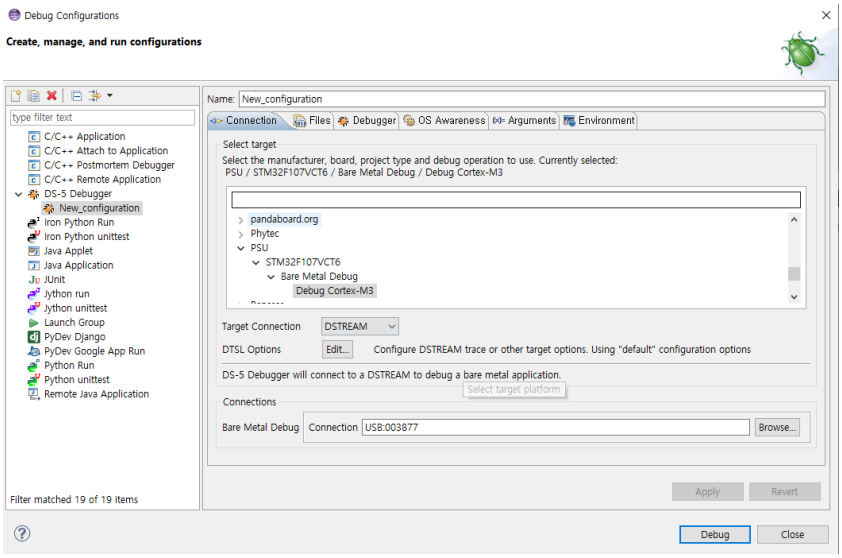


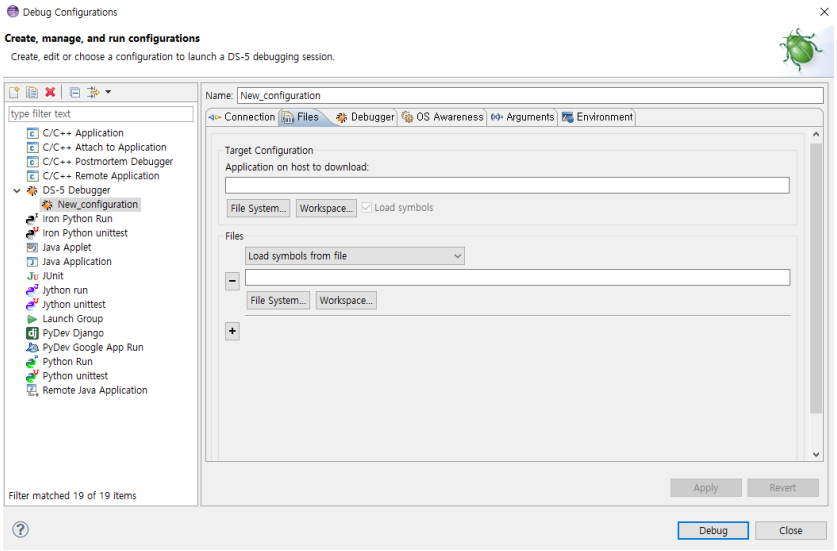
3.2.8 디버그 설정

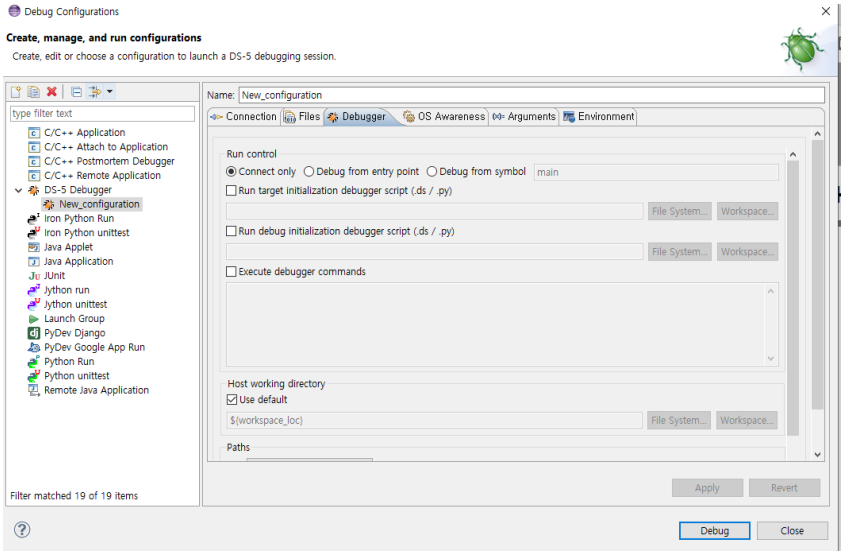
C 언어 소스파일을 만들어 빌드하고 디버그 설정을 한다. 4 주차와 동일한 환경으로 진행한다.






Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
DS-5 Debugger
New_configuration
Iron Python Run
Iron Python unittest
Java Applet
Java Application
JUnit
Jython run
Jython unittest
Launch Group
PyDev Django
PyDev Google App Run
Python Run
Python unittest
Remote Java Application
Filter matched 19 of 19 items
Name: New_configuration
Connection
Files
Debugger
OS Awareness
Arguments
Environment
Select target
Select the manufacturer, board, project type and debug operation to use. Currently selected:
PSU / STM32F107VCT6 / Bare Metal Debug / Debug Cortex-M3
pandaboard.org
Phytec
PSU
STM32F107VCT6
Bare Metal Debug
Debug Cortex-M3
Target Connection
DSTREAM
DTSL Options
Edit...
Configure DSTREAM trace or other target options. Using "default" configuration options
DS-5 Debugger will connect to a DSTREAM to debug a bare metal application.
Select target platform
Connections
Bare Metal Debug
Connection
USB:003877
Browse...
Apply
Revert
Debug
Close


Debug Configurations
Create, manage, and run configurations
Create, edit or choose a configuration to launch a DS-5 debugging session.
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
DS-5 Debugger
New_configuration
Iron Python Run
Iron Python unittest
Java Applet
Java Application
JUnit
Jython run
Jython unittest
Launch Group
PyDev Django
PyDev Google App Run
Python Run
Python unittest
Remote Java Application
Filter matched 19 of 19 items
Name: New_configuration
Connection
Files
Debugger
OS Awareness
Arguments
Environment
Target Configuration
Application on host to download:
File System...
Workspace...
Load symbols
Files
Load symbols from file
File System...
Workspace...
Apply
Revert
Debug
Close


Debug Configurations
Create, manage, and run configurations
Create, edit or choose a configuration to launch a DS-5 debugging session.
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
DS-5 Debugger
New_configuration
Iron Python Run
Iron Python unittest
Java Applet
Java Application
JUnit
Jython run
Jython unittest
Launch Group
PyDev Django
PyDev Google App Run
Python Run
Python unittest
Remote Java Application
Filter matched 19 of 19 items
Name: New_configuration
Connection
Files
Debugger
OS Awareness
Arguments
Environment
Run control
Connect only
Debug from entry point
Debug from symbol
main
Run target initialization debugger script (.ds / .py)
Run debug initialization debugger script (.ds / .py)
Execute debugger commands
Host working directory
Use default
${workspace_loc}
Paths
Apply
Revert
Debug
Close

04. 실험 및 과제 해결

4.1 System clock setting

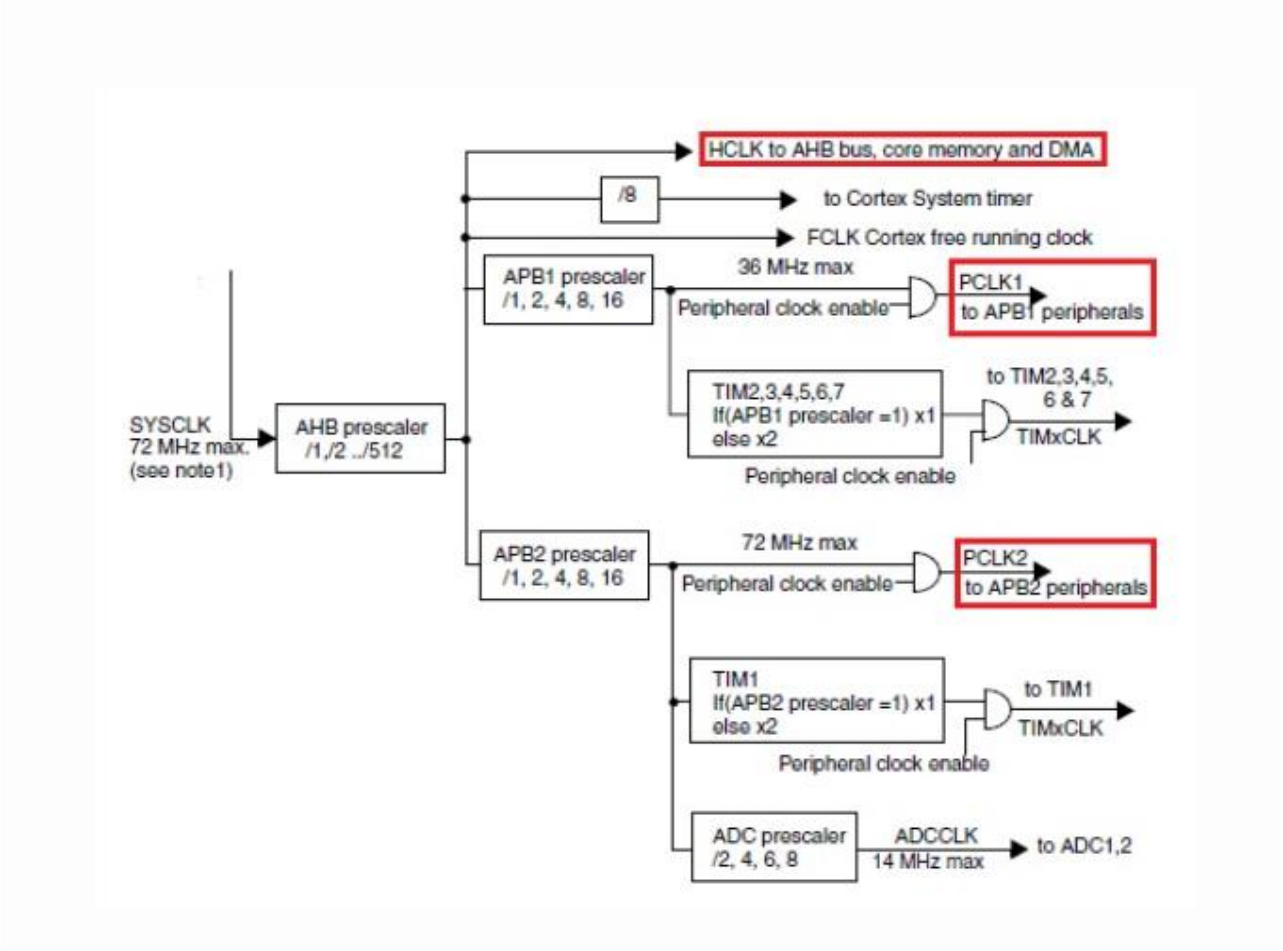
4.1.1 SYSCLK setting

보드의 Oscillator 에서 발생되는 25MHz 의 clock(HSE clock)과 보드의 내부 clock 인 8MHz 의 Clock(HSI clock)을 곱하거나 나누어서 최종 출력 clock 인 PLL 을 만들 수 있다.

실험에서 사용한 Clock 은 아래와 같다.

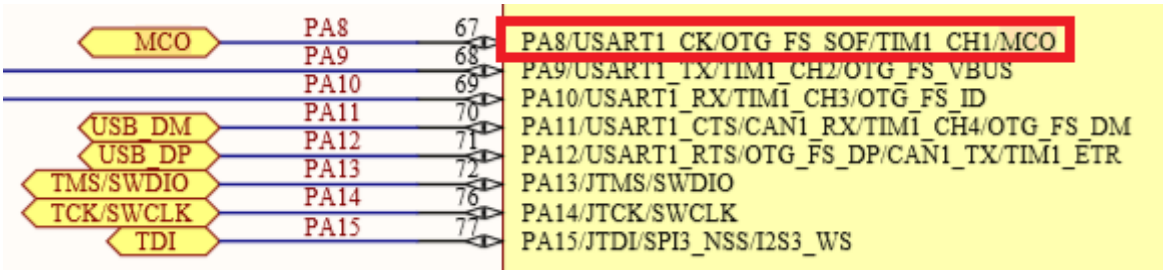
SYSTEM CLOCK	20MHz
HCLK	20MHz
PCLK2	10MHz

System clock 으로 20MHz 를 사용해야 하기 때문에, 이를 만들기 위해 25MHz clock 을 사용하여 25/5*10/10*4 = 20MHz 를 System clock 으로 지정한다.



이렇게 구한 SYSCLK 값을 사용하여 HCLK 와 PCLK2 의 값을 지정할 수 있다. HCLK 는 SYSCLK 과 동일한 20MHz 로 지정, PCLK2 는 SYSCLK 을 DIV2 한 값을 지정한다.

4.1.2 System Clock output 을 위한 MCO port setting



오실로스코프에 연결해 clock 을 확인하기 위하여 MCO port(PA8)를 output 으로 지정한다.

7.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCPRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 31:27 Reserved, must be kept at reset value.

Bits 26:24 **MCO:** Microcontroller clock output
Set and cleared by software.
0xx: No clock
100: System clock (SYSCLK) selected
101: HSI clock selected
110: HSE clock selected
111: PLL clock divided by 2 selected

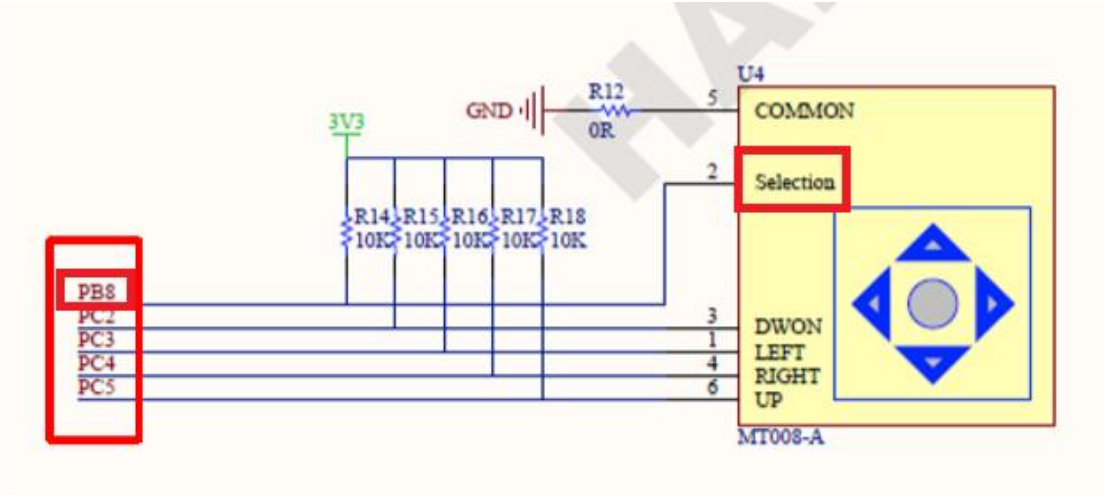
Note: This clock output may have some truncated cycles at startup or during MCO clock source switching.
When the System Clock is selected to output to the MCO pin, make sure that this clock does not exceed 50 MHz (the maximum IO speed).

Reference 의 내용에서 알 수 있듯 SYSCLK 을 MCO 로 내보내기 위해서 24-26 번째 비트에 각각 0 ,0,1 을 지정해 주어야 한다. (0x04000000)

4.2 GPIO(General purpose Input Output), USART setting

4.2.1 GPIO, USART Pin Setting

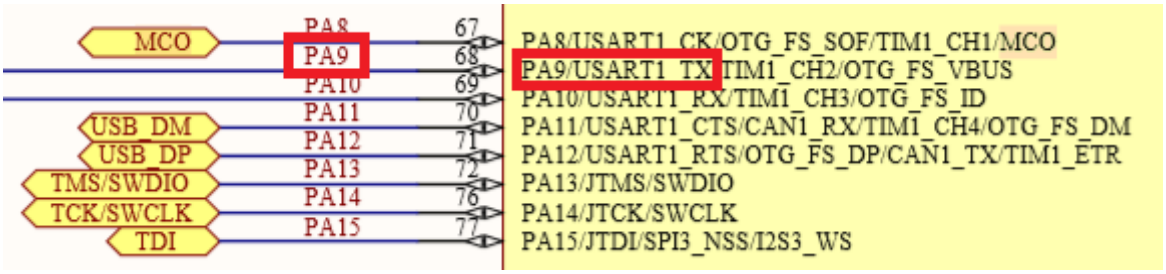
1. PB8(조이스틱 select) – Port8 (CRH), 보드 기준 input(mode 00 CNF 10)



Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

2. PA9(USART/TX)



보드의 select 버튼을 누르면 Putty 콘솔창에 문자열이 입력되도록 하는 것이 과제였으므로 RX 는 사용하지 않았다.

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

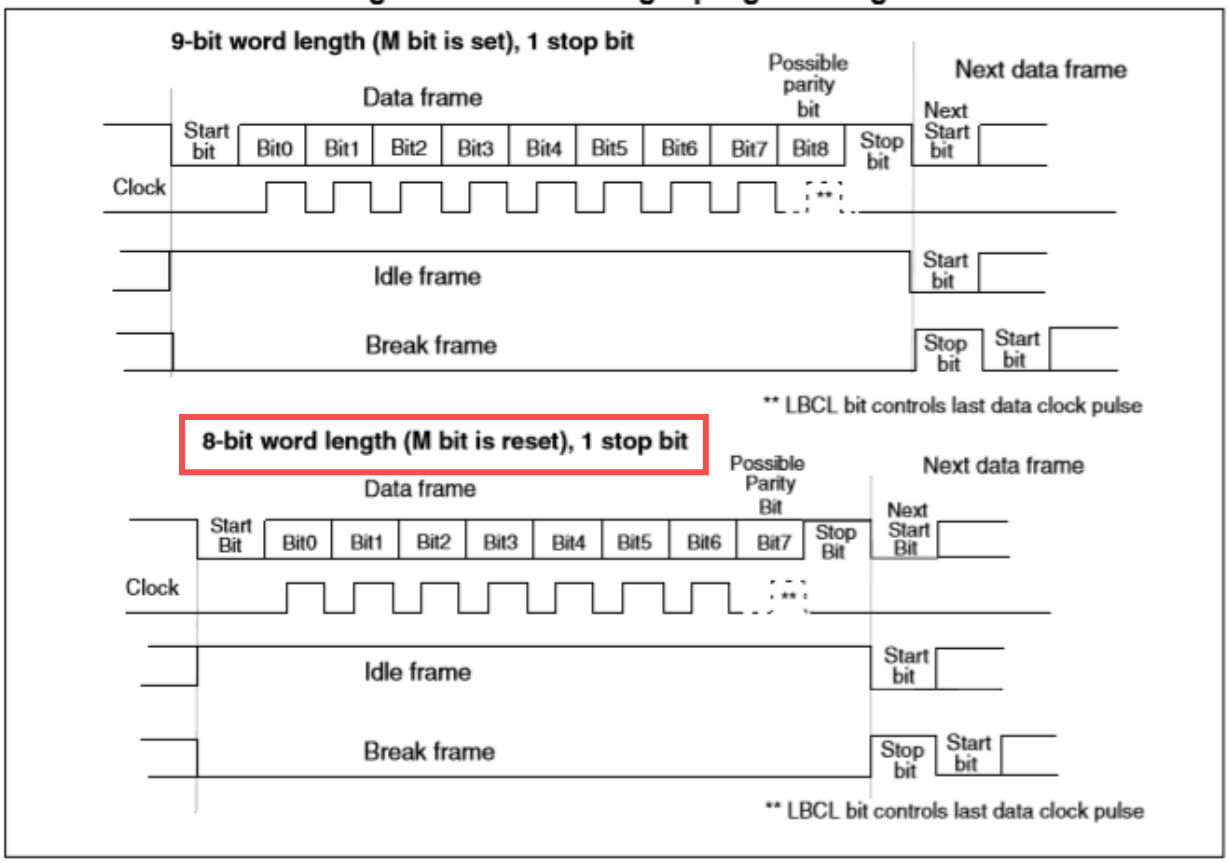
Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

UART 통신을 위해 GPIO 대신 AFIO(Alternate function I/O)를 사용해야 하므로 Mode 와 CNF 는 각각 10, 11 이다. (Mode = 0x00000030, CNF = 0x00000080)

4.3 USART Configuration

4.2.1 USART Init

Figure 279. Word length programming



1. Word length

   Word length 8 → 0 , Word length 9 → 1 인데, 실험에서는 8bit 를 사용하므로 기본값을 사용하였다.

2. Parity bit

   실험에서는 Parity bit = None 값을 사용했기 때문에 별도의 설정을 추가하지 않았다.

3. Stop bit

   The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

   1. ***1 stop bit***: This is the default value of number of stop bits.
   2. ***2 Stop bits***: This is supported by normal USART, single-wire and modem modes.
   3. ***0.5 stop bit***: To be used when receiving data in Smartcard mode.
   4. ***1.5 stop bits***: To be used when transmitting and receiving data in Smartcard mode.

   Stop bit 역시 default 값인 1bit 을 사용하였으므로 별도의 설정을 추가하지 않았다.

4.2.2 USART DIV (USART1 BRR) 계산

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

legend: f_{CK} - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

The baud counters are updated with the new value of the Baud registers after a write to USART_BRR. Hence the Baud rate register value should not be changed during communication.

실험에서 Tx/Rx baud 값이 115200 으로 주어졌고, reference 에 따르면 fck 는 PCKL2 의 clock 이므로 10MHz 이다. 위의 두 값을 이용해 구한 USART_DIV 값은 약 5.42 였다.

Example 2:

To program USARTDIV = 0d25.62

This leads to:

DIV_Fraction = 16*0d0.62 = 0d9.92

The nearest real number is 0d10 = 0xA

DIV_Mantissa = mantissa (0d25.620) = 0d25 = 0x19

Then, USART_BRR = 0x19A hence USARTDIV = 0d25.625

위의 예제에서 알 수 있듯 USART_BRR 에 들어갈 값은 ①USART_DIV 값의 fracrion 부분에 16 을 곱했을 때 가장 근접한 16 진수 값을 구하고, ②USART_DIV 값의 소수점을 제외한 부분을 16 진수로 바꾼 값을 구해 두 숫자(①, ②)를 조합하여 만들 수 있다.

USART_DIV 값이 5.42 였으므로,

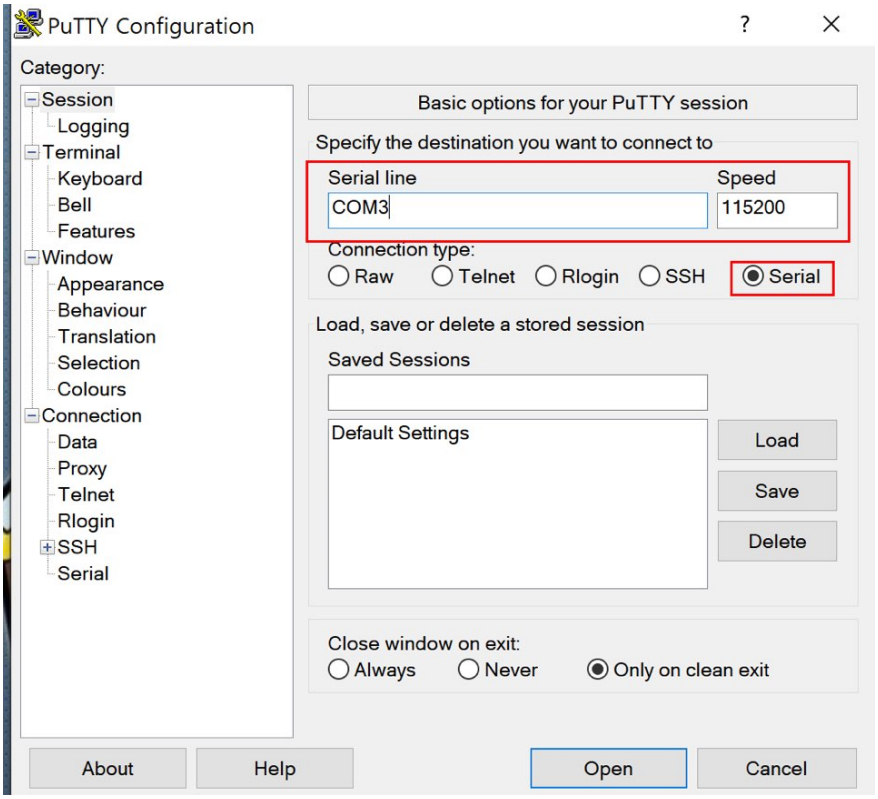
① 0.42*16 = 6.72

   7 에 근접하므로 16 진수로 변환해도 7 이다.

② 소수 부분을 제외하면 5 이므로, 16 진수로 변환해도 5 이다.

▶ 따라서 USART_BRR 에 들어갈 값은 0x57 임을 알 수 있다.

4.4 Putty setting



UART 통신을 사용하여 Putty 에 문자열을 출력하기 위해 Putty 를 다운로드하고 위와 같이 설정해준다. Serial Line 에는 보드와 연결된 COM3 port 를 설정해주고, Speed 에는 baud rate 인 115200 을 설정한다. Serial 통신을 할 것이기 때문에 Connection type 은 Serial 로 설정한다.

05. 실험 결과

5.1 소스코드

위의 내용을 바탕으로 작성한 소스코드는 아래와 같다.

```
#include "stm32f10x.h"
void SysInit(void) {
    /* Set HSION bit */
    /* Internal Clock Enable */
    RCC->CR |= (uint32_t)0x00000001; //HSION
    /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
    RCC->CFGR &= (uint32_t)0xF0FF0000;
    /* Reset HSEON, CSSON and PLLON bits */
    RCC->CR &= (uint32_t)0xFEF6FFFF;
    /* Reset HSEBYP bit */
    RCC->CR &= (uint32_t)0xFFFBFFFF;
    /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
    RCC->CFGR &= (uint32_t)0xFF80FFFF;
    /* Reset PLL2ON and PLL3ON bits */
    RCC->CR &= (uint32_t)0xEBFFFFFF;
    /* Disable all interrupts and clear pending bits  */
    RCC->CIR = 0x00FF0000;
    /* Reset CFGR2 register */
    RCC->CFGR2 = 0x00000000;
}
void SetSysClock(void) {
```

```
    volatile uint32_t StartUpCounter = 0, HSEStatus = 0;
    /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ---------------------------*/
    /* Enable HSE */
    RCC->CR |= ((uint32_t)RCC_CR_HSEON);
    /* Wait till HSE is ready and if Time out is reached exit */
    do {
        HSEStatus = RCC->CR & RCC_CR_HSERDY;
        StartUpCounter++;
    } while ((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));
    if ((RCC->CR & RCC_CR_HSERDY) != RESET) {
        HSEStatus = (uint32_t)0x01;
    }
    else {
        HSEStatus = (uint32_t)0x00;
    }
    if (HSEStatus == (uint32_t)0x01) {
        /* Enable Prefetch Buffer */
        FLASH->ACR |= FLASH_ACR_PRFTBE;
        /* Flash 0 wait state */
        FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
        FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_0;
//@TODO - 1 Set the clock
        /* HCLK = SYSCLK */
        RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
        /* PCLK2 = HCLK */
        RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV2;
        /* PCLK1 = HCLK */
        RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV1;
        /* Configure PLLs ------------------------------------------------------*/
        RCC->CFGR &= (uint32_t)~(RCC_CFGR_PLLXTPRE | RCC_CFGR_PLLSRC | RCC_CFGR_PLLMULL);
        RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLXTPRE_PREDIV1 | RCC_CFGR_PLLSRC_PREDIV1
|RCC_CFGR_PLLMULL4);
        //아래에서 구한 clock 에 4 를 곱한다.
        //5*4 = 20
        RCC->CFGR2 &= (uint32_t)~(RCC_CFGR2_PREDIV2 | RCC_CFGR2_PLL2MUL |
        RCC_CFGR2_PREDIV1 | RCC_CFGR2_PREDIV1SRC);
        RCC->CFGR2 |= (uint32_t)(RCC_CFGR2_PREDIV2_DIV5 | RCC_CFGR2_PLL2MUL10
|RCC_CFGR2_PREDIV1SRC_PLL2 | RCC_CFGR2_PREDIV1_DIV10);
        //25 /5 *10 /10 = 5
//@End of TODO - 1
        /* Enable PLL2 */
        RCC->CR |= RCC_CR_PLL2ON;
        /* Wait till PLL2 is ready */
        while ((RCC->CR & RCC_CR_PLL2RDY) == 0)
        {
        }
        /* Enable PLL */
        RCC->CR |= RCC_CR_PLLON;
        /* Wait till PLL is ready */
        while ((RCC->CR & RCC_CR_PLLRDY) == 0)
        {
        }
        /* Select PLL as system clock source */
        RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
        RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
        /* Wait till PLL is used as system clock source */
        while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08)
        {
        }
        /* Select System Clock as output of MCO */
//@TODO - 2 Set the MCO port for system clock output
        RCC->CFGR &= ~(uint32_t)RCC_CFGR_MCO;
        RCC->CFGR |= (uint32_t)RCC_CFGR_MCO_SYSCLK;
        //24-26 번째 bit 에 0(24),0(25),1(26) 지정하여 SYSCLK 을 MCO output 으로 설정
//@End of TODO - 2
    }
    else {
        /* If HSE fails to start-up, the application will have wrong clock
```

```
        configuration. User can add here some code to deal with this error */
    }
}
void RCC_Enable(void) {
//@TODO - 3 RCC Setting
    /*-------------------------- RCC Configuration -----------------------------*/
    /* GPIO RCC Enable  */
    /* UART Tx, Rx, MCO */
         RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPBEN;
    //Port A와 B RCC Enable
    /* USART RCC Enable */
         RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    //USART RCC Enable
}
void GPIO_Configuration(void) {
//@TODO - 4 GPIO Configuration
    /* Reset Port A CRH */
     GPIOA->CRH &= ~(GPIO_CRH_CNF8_0 | GPIO_CRH_CNF9_0);
    /* Reset Port B CRH */
     GPIOB->CRH &= ~(GPIO_CRH_CNF8_0);
    /* MCO Pin Configuration */
     GPIOA->CRH = (GPIO_CRH_MODE8 | GPIO_CRH_CNF8_1);
    /* USART Pin Configuration */
     GPIOA->CRH = (GPIO_CRH_MODE9 | GPIO_CRH_CNF9_1);
    /* Joy-stick SELECT Button Configuration */
     GPIOB->CRH |= 0x8;
}
void GPIO_Pin_InitSetting(void) {
//@TODO - 5 GPIO Set & Reset
    /* Enable MCO */
     GPIOA->BSRR |= 0x00000100;
    /* Joy-stick Button Reset */
     GPIOB->BRR  |= 0x00000100;
}
void UartInit(void) {
    /*-------------------------- USART CR1 Configuration -----------------------*/
    /* Clear M, PCE, PS, TE and RE bits */
    USART1->CR1 &= ~(uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR1_TE |
USART_CR1_RE);
    /* Configure the USART Word Length, Parity and mode ----------------------- */
    /* Set the M bits according to USART_WordLength value */
//@TODO - 6: WordLength : 8bit
    //8bit = 0, 9bit = 1인데 default인 8bit 사용할 것이기 때문에 별도의 설정 X
    /* Set PCE and PS bits according to USART_Parity value */
//@TODO - 7: Parity : None
    //None이라서 설정할 필요 없음
    /* Set TE and RE bits according to USART_Mode value */
//@TODO - 8: Enable Tx and Rx
    USART1->CR1 |= (uint32_t)(USART_CR1_TE | USART_CR1_RE);
    /*-------------------------- USART CR2 Configuration -----------------------*/
    /* Clear STOP[13:12] bits */
    USART1->CR2 &= ~(uint32_t)(USART_CR2_STOP);
    /* Configure the USART Stop Bits, Clock, CPOL, CPHA and LastBit ------------*/
    USART1->CR2 &= ~(uint32_t)(USART_CR2_CPHA | USART_CR2_CPOL | USART_CR2_CLKEN);
    /* Set STOP[13:12] bits according to USART_StopBits value */
//@TODO - 9: Stop bit : 1bit
    /*-------------------------- USART CR3 Configuration -----------------------*/
    /* Clear CTSE and RTSE bits */
    USART1->CR3 &= ~(uint32_t)(USART_CR3_CTSE | USART_CR3_RTSE);
    /* Configure the USART HFC -------------------------------------------------*/
    /* Set CTSE and RTSE bits according to USART_HardwareFlowControl value */
//@TODO - 10: CTS, RTS : disable
    //다 0이라서 따로 설정해줄 필요 없음
    /*-------------------------- USART BRR Configuration -----------------------*/
    /* Configure the USART Baud Rate -------------------------------------------*/
    /* Determine the integer part */
    /* Determine the fractional part */
```

```
//@TODO - 11: Calculate & configure BRR
     USART1->BRR |= 0x57;
    /*---------------------------- USART Enable ----------------------------------*/
    /* USART Enable Configuration */
//@TODO - 12: Enable UART
    USART1->CR1 |= USART_CR1_UE;
}
void delay(void){
    int i = 0;
    for(i=0;i<1000000;i++);
}
void SendData(int data) {
//    int i = 0;
    USART1->DR = data & 0xFF;
    delay();
}
int main() {
         int count = 0;
    SysInit();
    SetSysClock();
    RCC_Enable();
    GPIO_Configuration();
    UartInit();
    GPIO_Pin_InitSetting();
    //select 버튼을 누를 때마다 설정해둔 문자열이 출력됨
    //select 버튼을 눌렀을 때 IDR 이 변하는 값이 0x100 이기 때문에 0x100 사용
    while (1) {
                    if((GPIOB->IDR & 0x100) == 0 ) {
                             if(count%5==0) {
                               SendData('H');
                               SendData('e');
                               SendData('l');
                               SendData('l');
                               SendData('o');
                               SendData(' ');
                               SendData('T');
                               SendData('e');
                               SendData('a');
                               SendData('m');
                               SendData('0');
                               SendData('7');
                             }
                             else if( count%5 == 1){
                               SendData('L');
                               SendData('H');
                               SendData('C');
                             }
                             else if(count%5 == 2){
                               SendData('L');
                               SendData('D');
                               SendData('Y');
                             }
                             else if(count%5 == 3){
                               SendData('P');
                               SendData('C');
                               SendData('Z');
                             }
                             else if(count%5 == 4){
                               SendData('J');
                               SendData('S');
                               SendData('H');
                          }
                    count = count + 1;
                    delay();
                    }
    }
}
```

5.2 소스코드 해설

위의 소스코드는 ①보드의 Oscillator 에서 발생되는 clock 을 이용하여 목적 clock(SYSCLK)을 생성하고 오실로스코프에서 SYSCLK 이 정상적으로 출력되는지 확인한 후, ②UART 통신을 이용하여 joystick 의 select 입력을 받아 주어진 문자열을 putty 콘솔창에 출력하도록 하는 코드이다.

① 목적 clock 을 만들기 위해 기존의 25MHz clock 을 곱셈, 나눗셈 연산을 통해 20MHz 로 변환하고, 이를 MCO 로 출력하기 위해 MCO 로 SYSCLK 을 출력할 수 있도록 24-26 번째 비트를 0,0,1 로 지정해준다.

SYSCLK 을 이용하여 HCLK 와 PCLK2 를 설정하기 위해 연산을 수행한다. 실험에서 만들어야 할 값은 SYSCLK 20MHz, HCLK 20MHz, PCLK 10MHz 였으므로, HCLK 는 SYSCLK 과 동일한 값을 넣어주고, PCLK 를 구할 때에는 DIV2 연산을 수행한다.

② PA8, PA9 는 UART 통신을 사용하기 위해 GPIO 대신 Alternate function I/O 를 사용한다. Mode bit 와 CNF 를 각각 11, 10 으로 지정한다. (AFIO Output).

PB8 (조이스틱) 은 GPIO 를 사용한다.

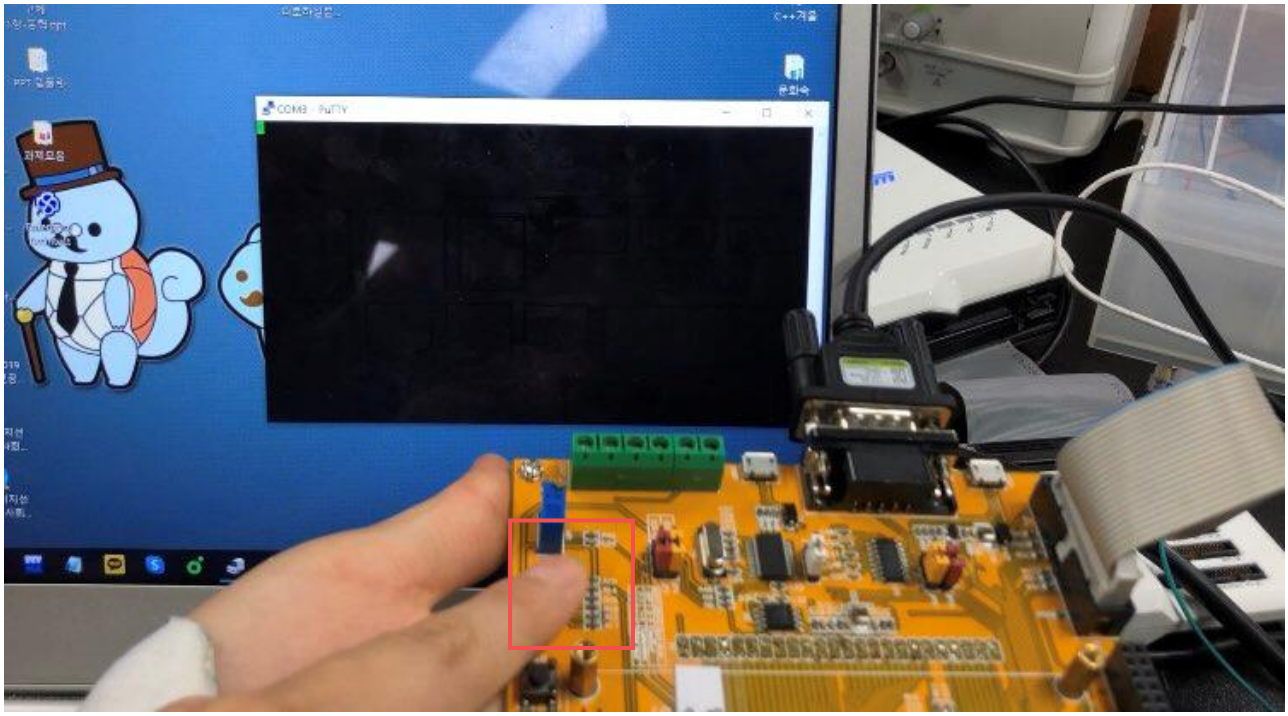
USART_BRR 에 들어가야 할 값은 reference 를 참고하여 계산한 값인 0x57 을 사용한다.

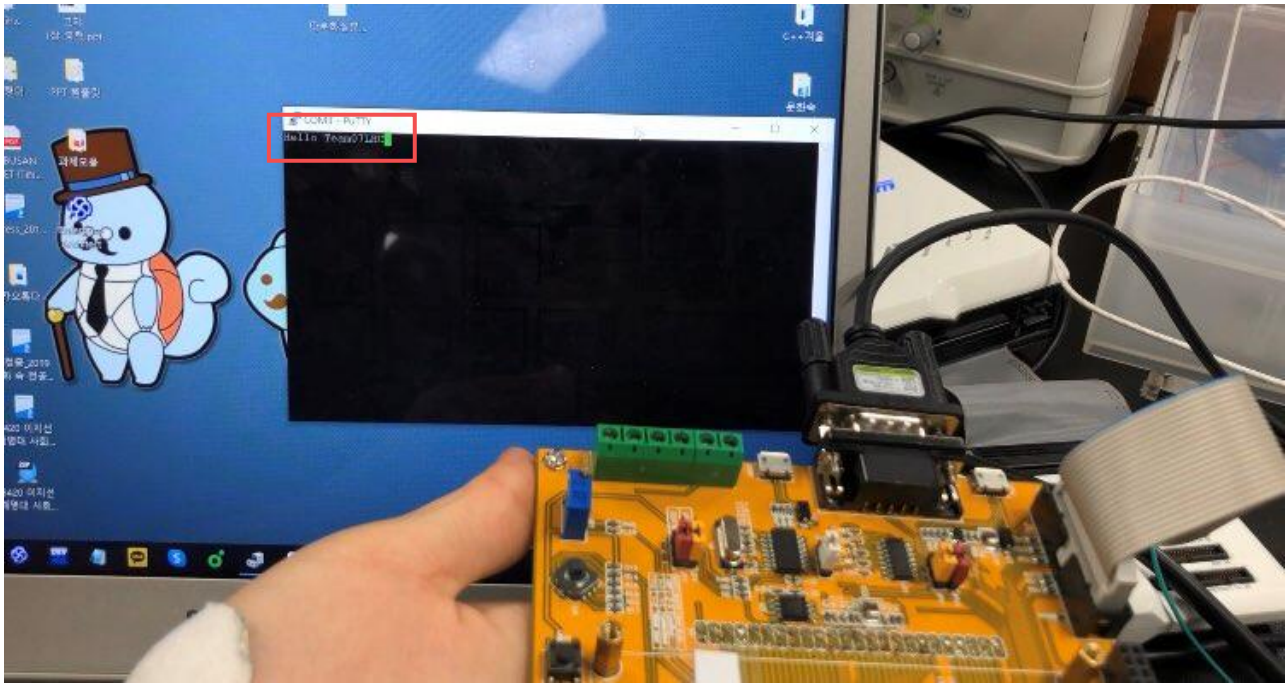
조이스틱의 select 입력을 받아 주어진 문자열을 UART 통신으로 Putty 에 출력한다. 이 때 4 주차와 같이 조이스틱의 select 입력 시 변하는 IDR 값은 0x100 이므로 조건문에 0x100 을 사용한다.

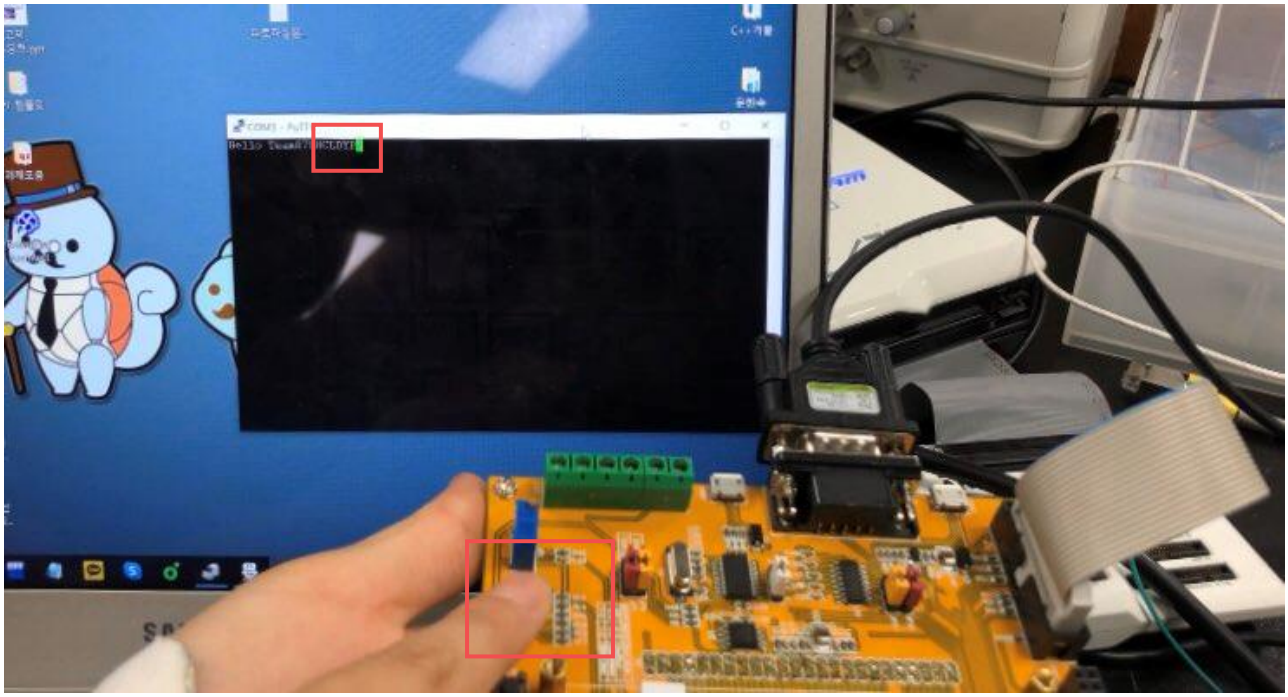
5.3 결과 확인 및 사진

해당 소스 코드대로 동작하는 프로그램은 다음과 같이 확인되었다.

- 조이스틱 select : 조이스틱을 select 버튼을 한 번 누를 때마다 설정해둔 문자열 출력

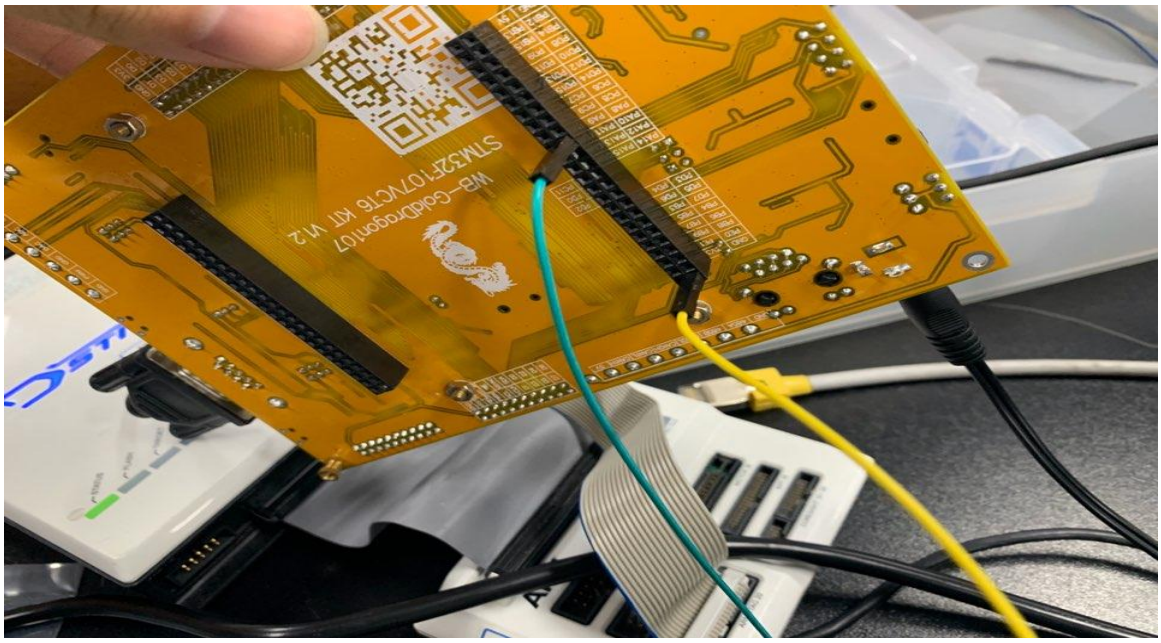


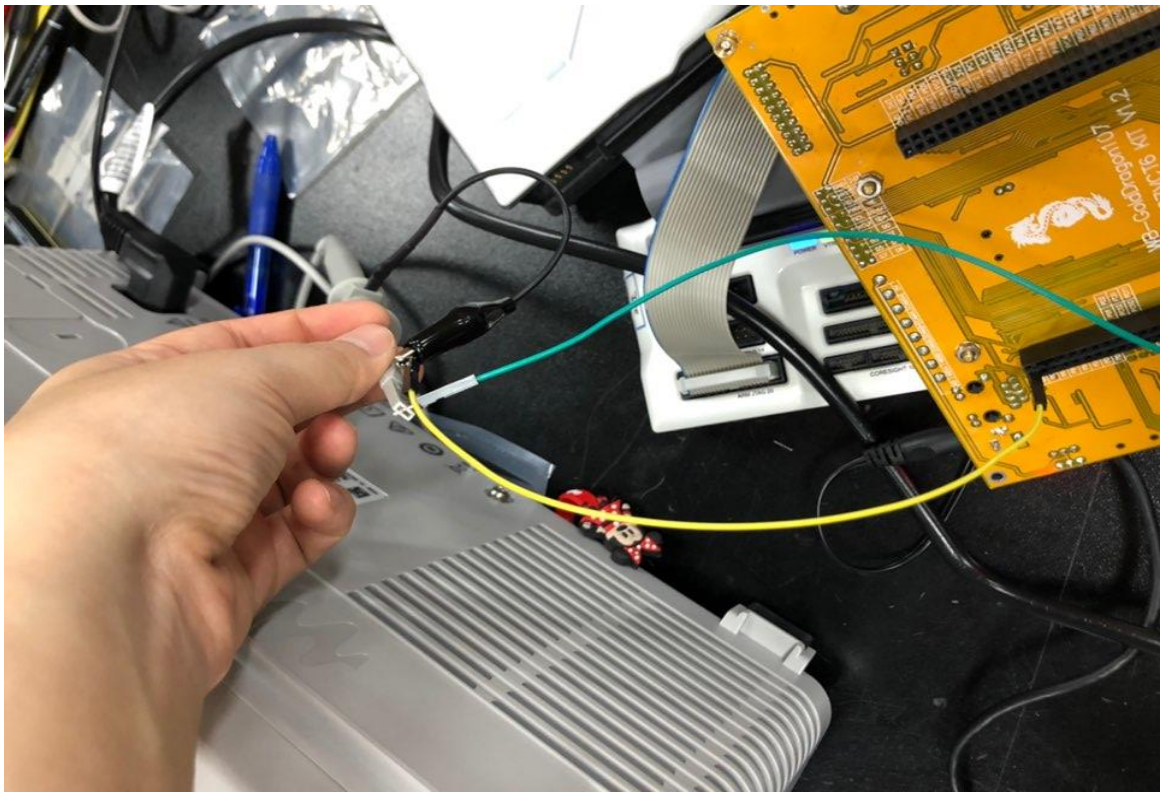


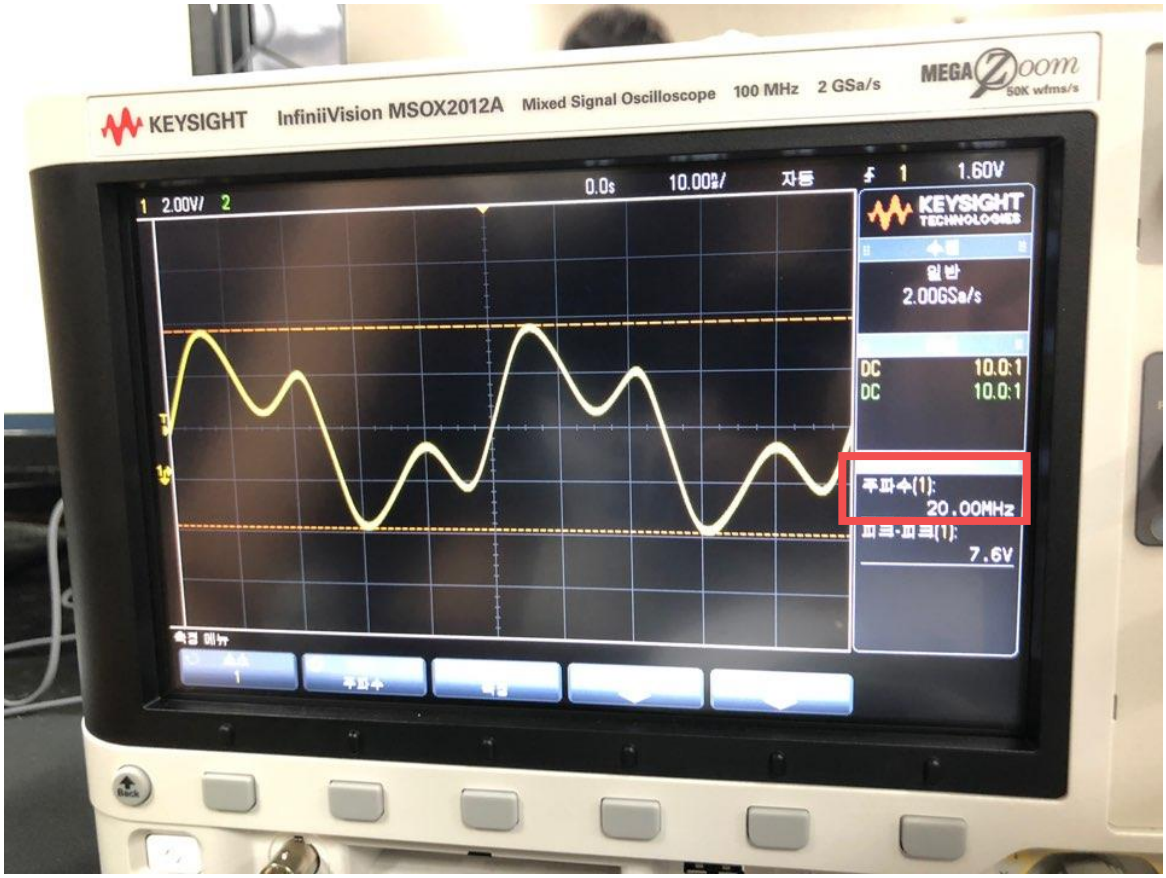


5.4 오실로스코프를 이용한 Clock 출력 확인

시스템 clock 인 20MHz 가 출력되는 것을 화면을 통해 확인할 수 있었다.







06. 결론

지금까지의 실험과는 다르게 &= ~() 연산을 통해 사용할 bit 만 초기화 하는 것이 익숙하지 않아 이해하기까지 꽤 많은 시간이 소요되었다. 또한 GPIO 를 AFIO 로 사용하는 방법을 찾기까지 많은 고생을 했는데, reference 를 자세히 읽어보니 Mode bit 와 CNF bit 만 바꾸면 되는 것이었다. 앞으로 새로운 개념을 듣고 실험에 적용할 때에 reference 를 좀 더 자세히 읽어보고 진행한다면 시간을 훨씬 단축할 수 있을 것이라는 생각이 들었다. 또한 목적 clock 을 만들어낸다는 개념을 어떻게 이해해야 할지 몰랐는데, 보드 내부에서 발생되는 clock 이 무엇이고, 바깥으로 나가야 하는 clock 이 무엇인지부터 먼저 생각하고 그 후에 clock 연산을 생각하니 보다 이해가 쉽게 되었다. 새로운 개념들이 많이 등장했던 실험이라 실험 과정이 순탄치 않았지만, 팀원들과 함께 문제를 하나씩 해결하다 보니 실험 도중에 깨닫는 것들도 많았고, 실험을 마무리하고 다시 전 과정을 살펴보니 고민했던 지점들이 이해가 되었다.