layout: post author: ... date: 2016-09-30 17:48 +0900 tags:

comments: true

ARM DS-5

DStream 과 Cortex-M3 으로 하는 실험이다. DStream 엔 JTAG 라는 디버거가 있고, 플래시 다운로드가 있어서 Cortex-M3 이라는 cpu 로 전송이 가능하다. 또한 이 JTAG 를 통해서 실시간 디버깅이 가능하다.

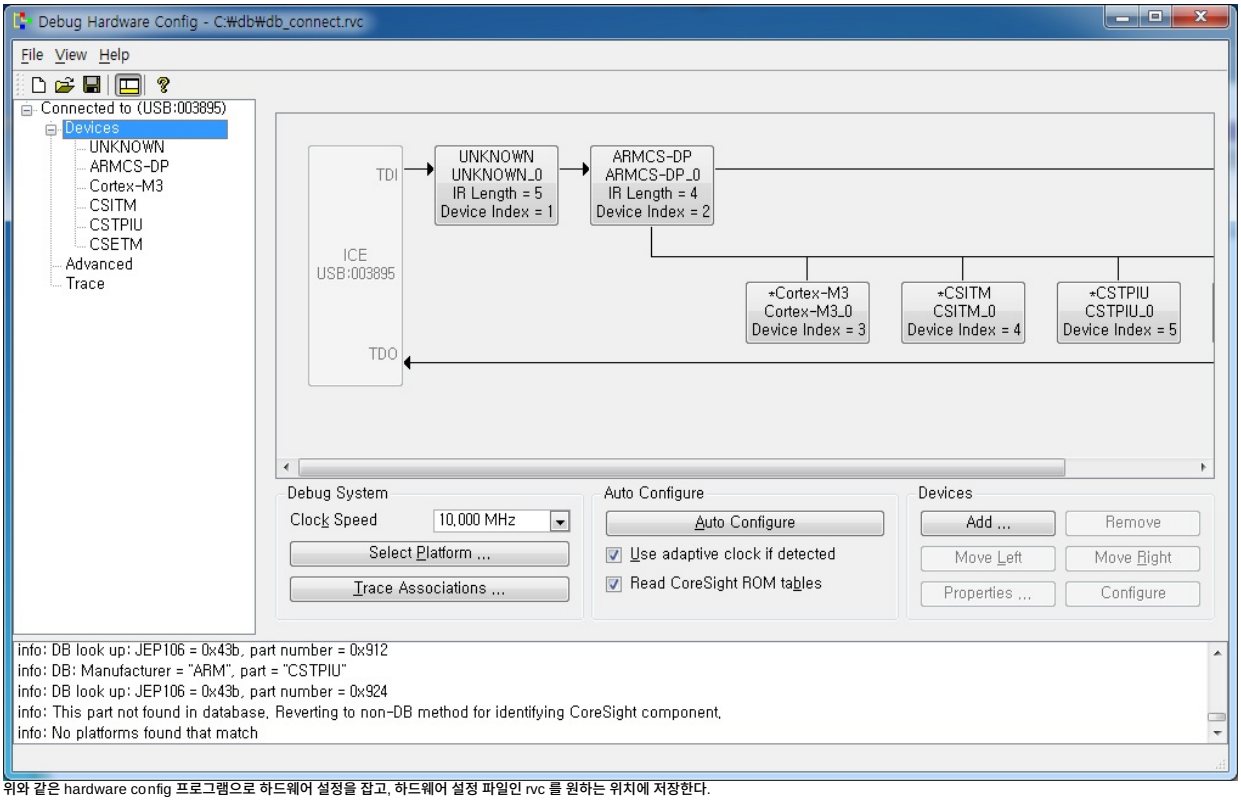
DStream 으로 Cortex-M3 에 axf 를 플래시 다운로더를 통해서 작업할 수 있다.

개요

Eclipse for DS-5 로 보드의 레지스터 값을 조작하는 프로그램을 플래시 다운로더로 보드의 cpu 로 전송시켜 LED를 제어한다.

세팅

Debug Hardware Config를 이용하여 host PC에 연결된 Dstream 목록 중에 해당 Dstream 항목을 선택한 뒤, Connect한다. Auto Configure버튼을 눌르면 Dstream 장비가 JTAG를 통해 프로세서 내부 Debug Logic의 각 컴포넌트에 대한 체인 정보를 읽어온다.



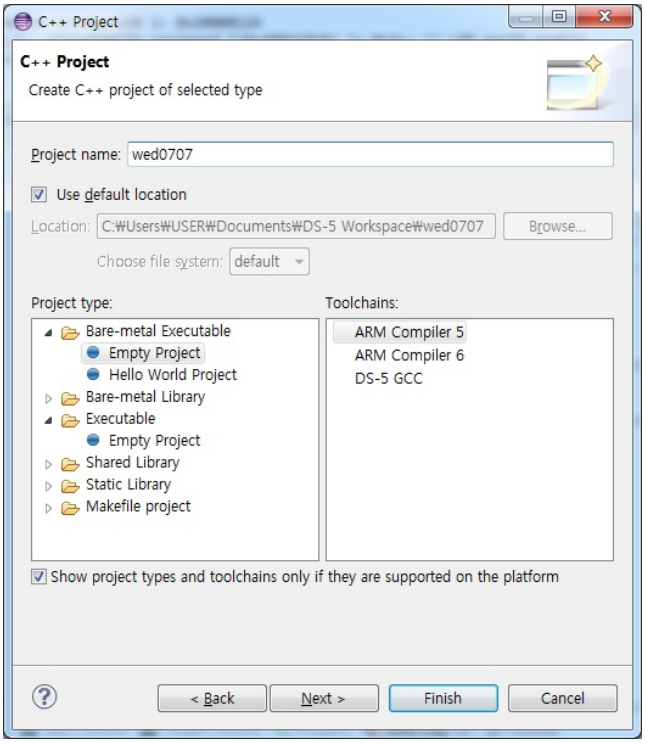
위와 같은 hardware config 프로그램으로 하드웨어 설정을 잡고, 하드웨어 설정 파일인 rvc 를 원하는 위치에 저장한다.

ex) c:\db\db_connect

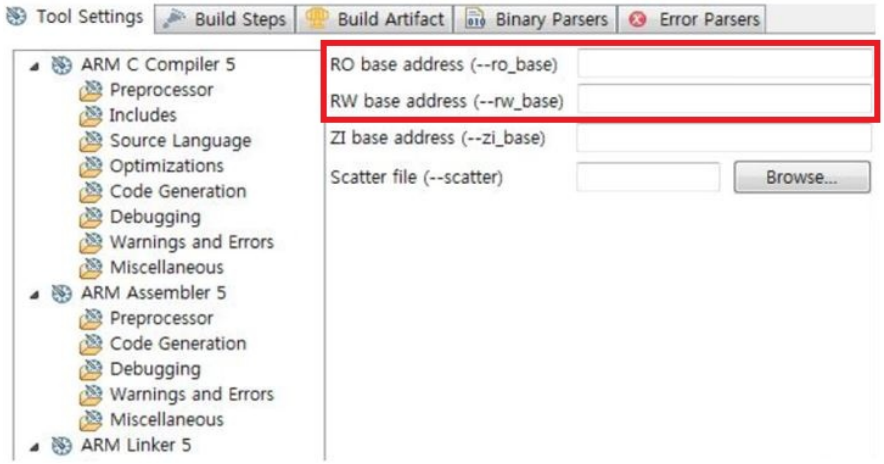
이제 Debugger 가 이 설정파일에 관한 디바이스를 import 하기 위해

```
cdbimporter.exe -t c:\db c:\db_connect
```

라는 명령어로 import 한다.



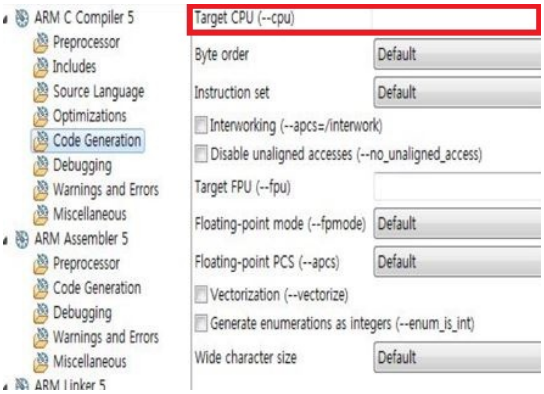
Eclipse for DS-5 를 이용하여 C++ Proejct 를 생성한다.



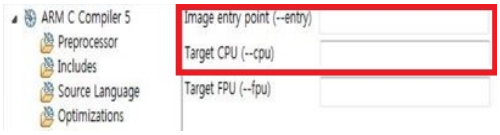
RO base address(--ro*base)와 RW base address(--rw* base)에 각각 SRAM (aliased by bit-banding)의 메모리 주소를 입력한다.

RO base address(--ro_base) = 0x2000 0000.

RW base address(--rw_base) = 0x2000 0000 ~ Ox2000 FFFF 중 임의의 주소

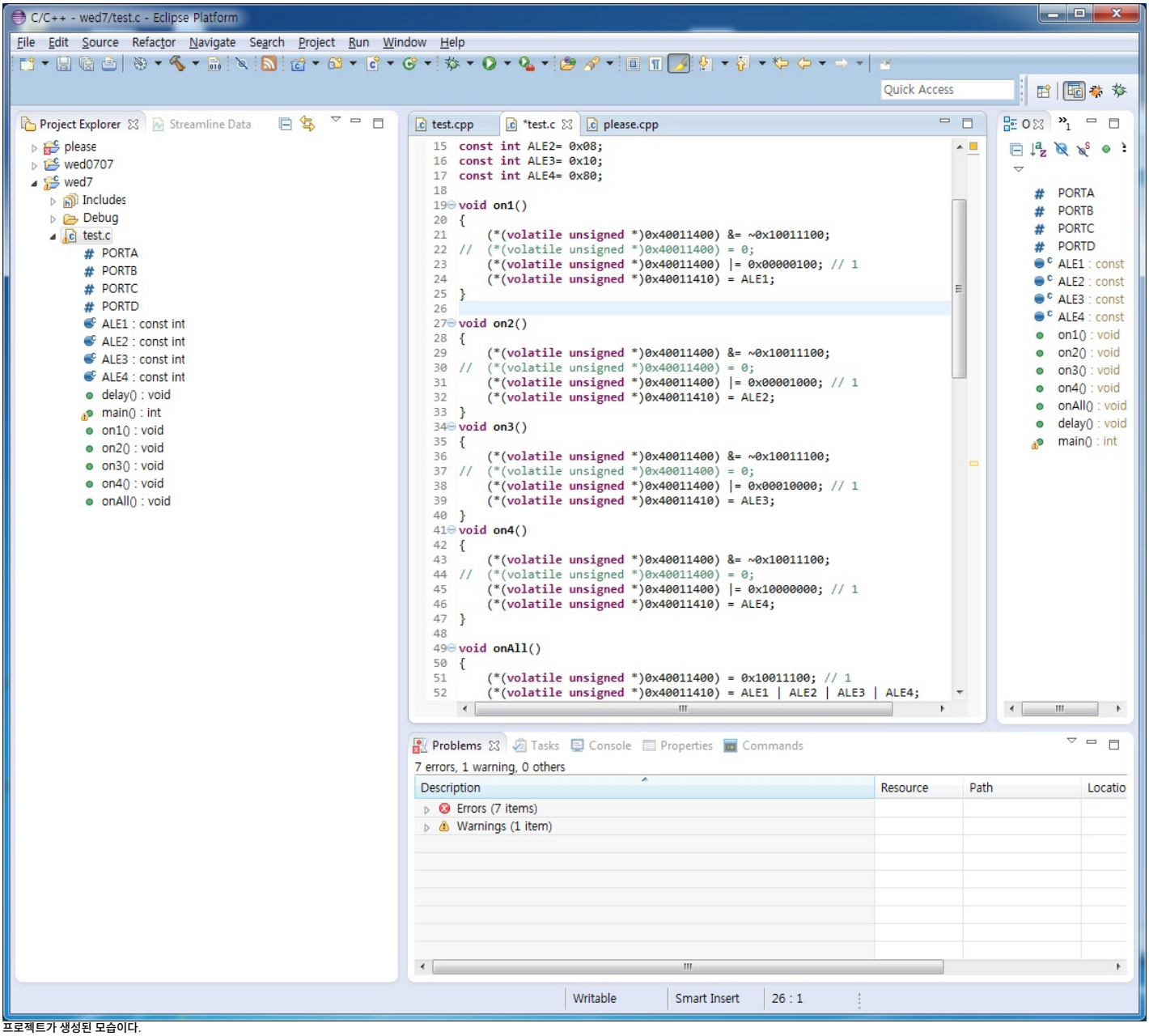


Target CPU(--cpu) 자리에 우리가 STM32 보드에서 사용할 CPU이름 Cortex-M3를 입력.



여기에도 Image entry point(--entry) = main Target CPU(--cpu) = Cortex-M3

값을 넣어준다.

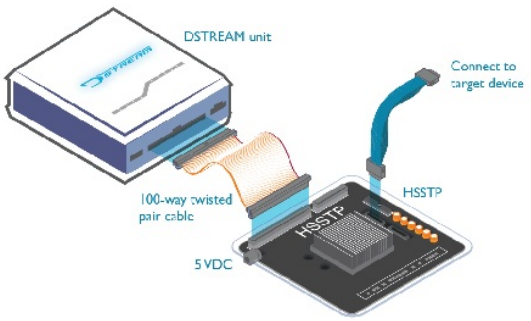


프로젝트가 생성된 모습이다.

아무것도 하지 않은채 다음 코드를 build 하게 되면 axf 파일이 Debug 하위 폴더에 생성이 된다.

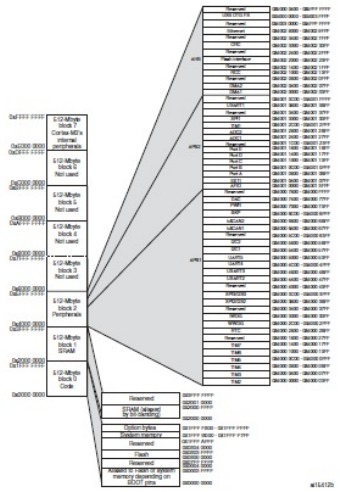
```
int main(){
  return 0;
}
```

Debug Configuration 에 들어가서 axf 파일을 지정한 뒤, Debug 를 누르면 플래시 다운로더가 axf 파일을 전송하면서 우리가 만든 프로그램이 ARM 보드 상에서 동작하게 된다.



구현

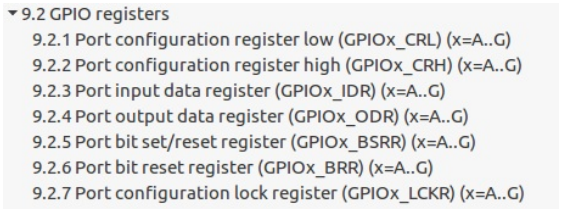
GPIO memory map



memory mapping에 사용될 memory map.

각 register 별로 memory address 가 나타나 있다.

GPIO Register 에 대해



그림을 보면 GPIO x Register 에 대해 나타나 있는데, 각 CRL 부터 LCKR 까지 GPIO_BASE 주소로 부터 offset 을 가진다.

```
//offset
CRL = 0x00
CRH = 0x04
IDR = 0x08
ODR = 0x0C
BSRR = 0x10
BRR = 0x14
LCKR = 0x18
```

앞으로 이 GPIO 의 원하는 목적에 따라 해당 register 를 set/unset 하면서 과제를 수행할 것이다.

이 register 변수를 제어하기 위해서 간단한 비트연산의 지식이 필요하다.

Bit 다루기

Bit 는 여기서 32bit 변수라고 가정한다. 그리고 bit 는 0 부터 시작한다고 생각한다.

Bit Set

bit set 은 | (OR) 연산을 통해 구현하게 된다.

```
BIT |= (1 << 3) // 3 번째 비트를 set
BIT |= (1 << n) // n 번째 비트를 set

// macro
#define set_bit(data, loc)      ((data) |= (0x1 << (loc)))
```

Bit Unset

bit unset 은 & 와 ~ 을 조합하여 구현하게 되는데 코드는 다음과 같다.

```
BIT &= ~(1 << 3) // 3번째 비트를 unset

/* 한 비트 클리어 */
#define clear_bit(data, loc)    ((data) &= ~(0x1 << (loc)))
```

Bit Check

bit check 방법은 다음과 같다.

```
if(data & (1 << 3)) // 3번째 bit 가 1인지 검사

/* 비트 검사 */
#define check_bit(data, loc)    ((data) & (0x1 << (loc)))
```

Port Enable

```
#define RCC_BASE                0x40021000
#define RCC_APB2ENR         REGISTER_32(RCC_BASE + 0x18 )
(*(volatile unsigned *)0x40021018) |= 0x3c; // A, B, C, D Port Enable
```

7.3.11 RCC register map

The following table gives the RCC register map and the reset values.

Table 18. RCC register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RCC_CR	Reserved						PLL RDY	PLL ON	Reserved				CSSON	HSEBYP	HSERDY	HSEON	HSICAL[7:0]								HSITRIM[4:0]					Reserved	HSIRDY	HSION
	Reset value							0	0					0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		1	1
0x04	RCC_CFGR	Reserved					MCO [2:0]			Reserved	USBPRE	PLLMUL[3:0]				PLLXTPRE	PLLSRC	ADC PRE [1:0]		PPRE2 [2:0]			PPRE1 [2:0]			HPRE[3:0]				SWS [1:0]		SW [1:0]	
	Reset value						0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	RCC_CIR	Reserved								CSSC	Reserved			PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved			PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	Reserved			PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF
	Reset value									0				0	0	0	0	0				0	0	0	0	0	0				0	0	0	0	0
0x0C	RCC_APB2RSTR	Reserved										TIM11RST	TIM10RST	TIM9RST	Reserved			ADC3RST	USART1RST	TIM8RST	SPI1RST	TIM1RST	ADC2RST	ADC1RST	IOPGRST	IOPFRST	IOPERST	IOPDRST	IOPCRST	IOPBRST	IOPARST	Reserved	AFIORST
	Reset value											0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
0x010	RCC_APB1RSTR	Reserved		DACRST	PWRRST	BKPRST	Reserved		CANRST	Reserved	USBRST	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	Reserved	SPI3RST	SPI2RST	Reserved		WWDGRST	Reserved		TIM14RST	TIM13RST	TIM12RST	TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST
	Reset value			0	0	0			0		0	0	0	0	0	0	0		0	0			0			0	0	0	0	0	0	0	0	0
0x14	RCC_AHBENR	Reserved																							SDIOEN	Reserved	FSMCEN	Reserved	CRCEN	Reserved	FLITFEN	Reserved	SRAMEN	DMA2AEN	DMA1AEN
	Reset value																								0		0		0		1		1	0	0
0x18	RCC_APB2ENR	Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved			ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Reserved	AFIOEN
	Reset value											0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0		0

RCC_APB2ENR 에서 PORT 들을 enable 시켜줘야 사용이 가능한데, 위에서 표를 보면 PORTAEN, PORTBEN, PORTCEN, PORTDEN 에 해당된다.

해당 비트를 2진수로 표현하면 0b0011 1100 인데 16진수로 표현하면 0x3C 가 된다. 그래서 0x40021018 주소에 0x3C 를 넣어주면 enable 이 된다.

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	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
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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= 0 .. 7)

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= 0 .. 7)

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.

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

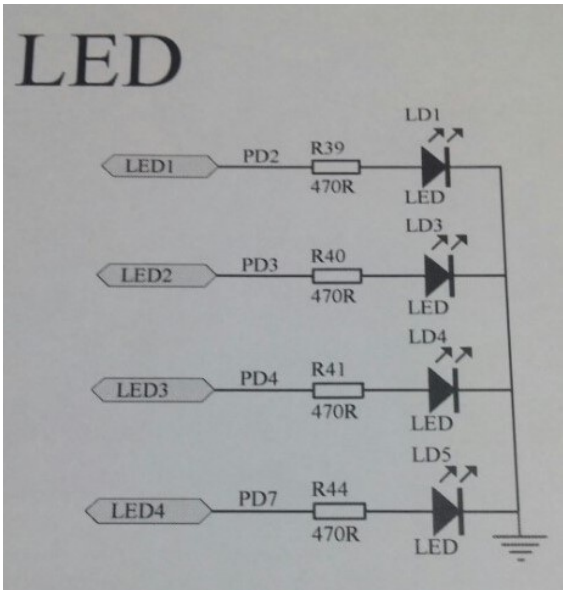
Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Set the corresponding ODRx bit

LED 에 불켜기



LED 는 기본적으로 PortD 를 사용하여 동작시킬 수 있다.

GPIOD*BSRR*, *GPIOD*CRL 두개의 레지스터에 값을 넣으면서 구현 가능하다.

```
LED1 : BS2
LED2 : BS3
LED3 : BS4
LED4 : BS7

0b10000000 : LED4
0b00010000 : LED3
0b00001000 : LED2
0b00000100 : LED1
```

LED1을 동작시키는 실제 코드는 다음과 같다.

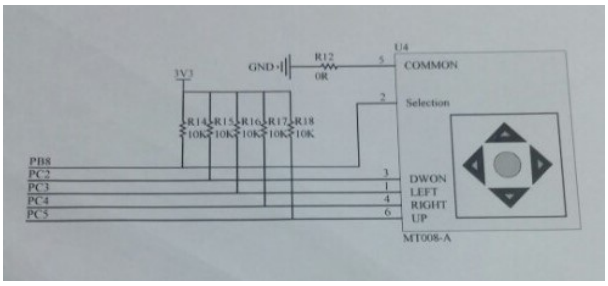
```
const int ALE1= 0x04;
```

```
const int ALE2= 0x08;
const int ALE3= 0x10;
const int ALE4= 0x80;

void on1()
{
  (*(volatile unsigned *)0x40011400) &= ~0x10011100; // all unset
  (*(volatile unsigned *)0x40011400) |= 0x00000100; // 1
  (*(volatile unsigned *)0x40011410) = ALE1;
}

int main(){
  (*(volatile unsigned *)0x40011404) = 0x44444444; // reset
  (*(volatile unsigned *)0x40011400) = 0x10011100; // 1
  (*(volatile unsigned *)0x40011008) = 0x00000000;
  return 0;
}
```

조이스틱 탐지



조이스틱은 PortC 를 이용해서 탐지가 가능하다.

GPIOC_IDR : input data register 를 이용해서 탐지한다.

GPIOC_IDR 의 주소는 0x40011008. 조이스틱을 상하좌우로 컨트롤 하면 bit 가 다음과 같이 바뀌게 된다.

```
0b00x0 0000
0b000x 0000
0b0000 x000
0b0000 0x00
```

조이스틱 입력이 들어오면 x 가 0으로 바뀐다. 어떻게 바뀌는지 알았으니, 코드로 구현하면 다음과 같다.

조이스틱이 눌러졌을 때 처리

```
if(~(*(volatile unsigned *)0x40011008) & 0x4)
  on1();
else if(~(*(volatile unsigned *)0x40011008) & 0x20)
  on4();
else if(~(*(volatile unsigned *)0x40011008) & 0x8)
  on2();
else if(~(*(volatile unsigned *)0x40011008) & 0x10)
  on3();
```

조이스틱이 눌러졌을 땐 PORTC_IDR 값의 0b000x 0000 0000.

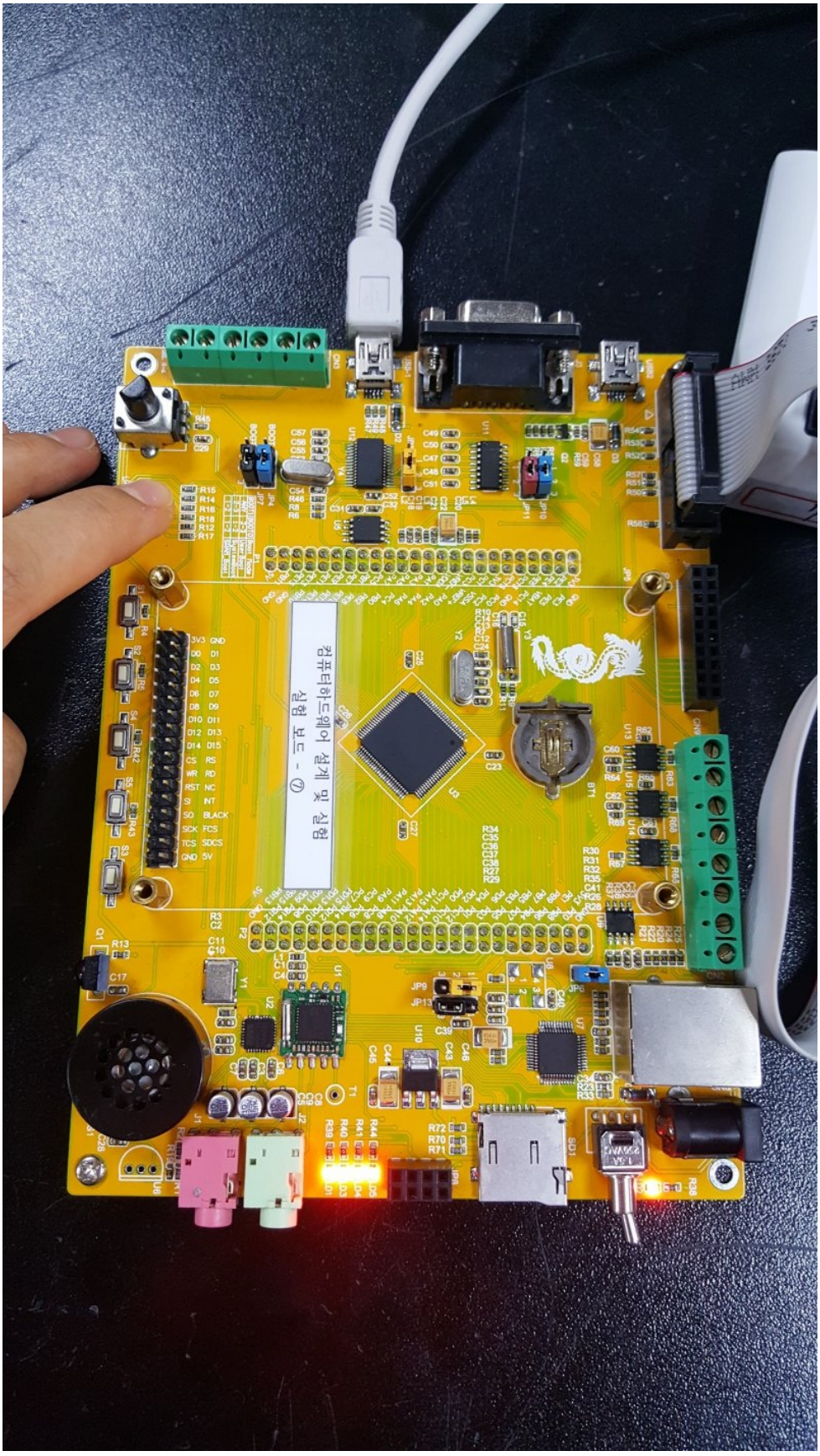
x부분의 값이 0으로 바뀌는데 마찬가지로 코드로 구현하면 다음과 같다.

LED에 불이 전부 들어온 경우

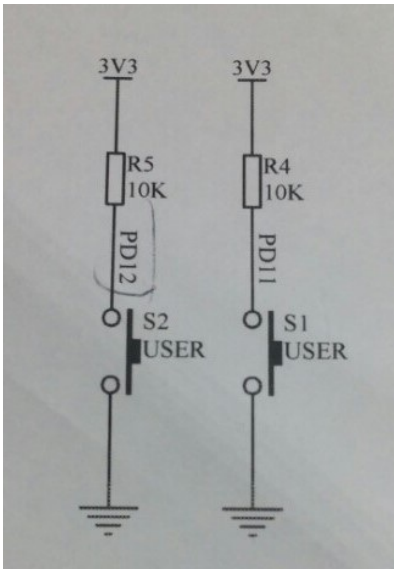
```
if(~(*(volatile unsigned *)0x40010C08) & 0x100)
  onAll();

void onAll()
{
  (*(volatile unsigned *)0x40011400) = 0x10011100; // 1
  (*(volatile unsigned *)0x40011410) = ALE1 | ALE2 | ALE3 | ALE4;
}
```

동작된 그림은 다음과 같다.



S1, S2 버튼 탐지




```
CRL = 0x00
CRH = 0x04
IDR = 0x08
ODR = 0x0C
BSRR = 0x10
BRR = 0x14
LCKR = 0x18
```

버튼은 PORTD 를 이용해서 탐지가 가능하다. 좀더 자세히 쓰면 PORTD*IDR 을 통해서 탐지가 가능하다. 좀 전 LED 를 제어할 때도 PORTD 를 썼는데, 입력을 받을 때도 같은 PORT 를 쓰는게 가능하다. 출력할 땐 PORTD*CNF, PORTD*BSRR 을 썼고, 입력을 받을 땐 PORTD*IDR 을 쓴다.

```
// S1 가 눌러졌을 땐
0b0000 x000 0000 0000
// S2 가 눌러졌을 땐
0b000x 0000 0000 0000
```

위의 x 인 부분이 해당 버튼이 눌러졌을 때 0으로 바뀐다. 이를 탐지하는 코드를 쓰면 다음과 같다.

```
while ( !(((*(volatile unsigned *)0x40011408) & 0x0800) >> 11) ) {
  on1(); delay();
  on2(); delay();
  on3(); delay();
  on4(); delay();
}
while ( !(((*(volatile unsigned *)0x40011408) & 0x1000) >> 12) ) {
  on4(); delay();
  on3(); delay();
  on2(); delay();
  on1(); delay();
}
```

결론

처음 다루는 ARM 보드라서 많이 헷갈렸고 어려웠다. 예비실험 조의 발표에서 조금 정보가 덜 주어진 것 같아 처음엔 조원 모두가 불만이었으나, 문제를 해결해 나가면서 우리가 다루는 플랫폼에 대해서 점점 익숙해졌다. 만약 모든 정보가 주어지고 거기에 맞춰서 그냥 따라하기만 했다면 해당 정보들이 머리 속에 들어가있지 않았을것 같다. 고생한만큼 큰 공부가 되었고, 앞으로 수행할 과제에 대해 스스로 해결하는 힘을 길렀다고 생각한다. 또한 처음이라서 memory 주소를 그대로 쓰면서 과제를 했는데, 다음부터는 미리 define 된 bit macro 와 stm32f macro 를 적극 활용해서 readability 를 높이는 방향으로 코딩을 할 것이다.
memory address를 통해 ARM 보드의 기능을 다루는 법을 알게 되었다.