

Interrupt Programming

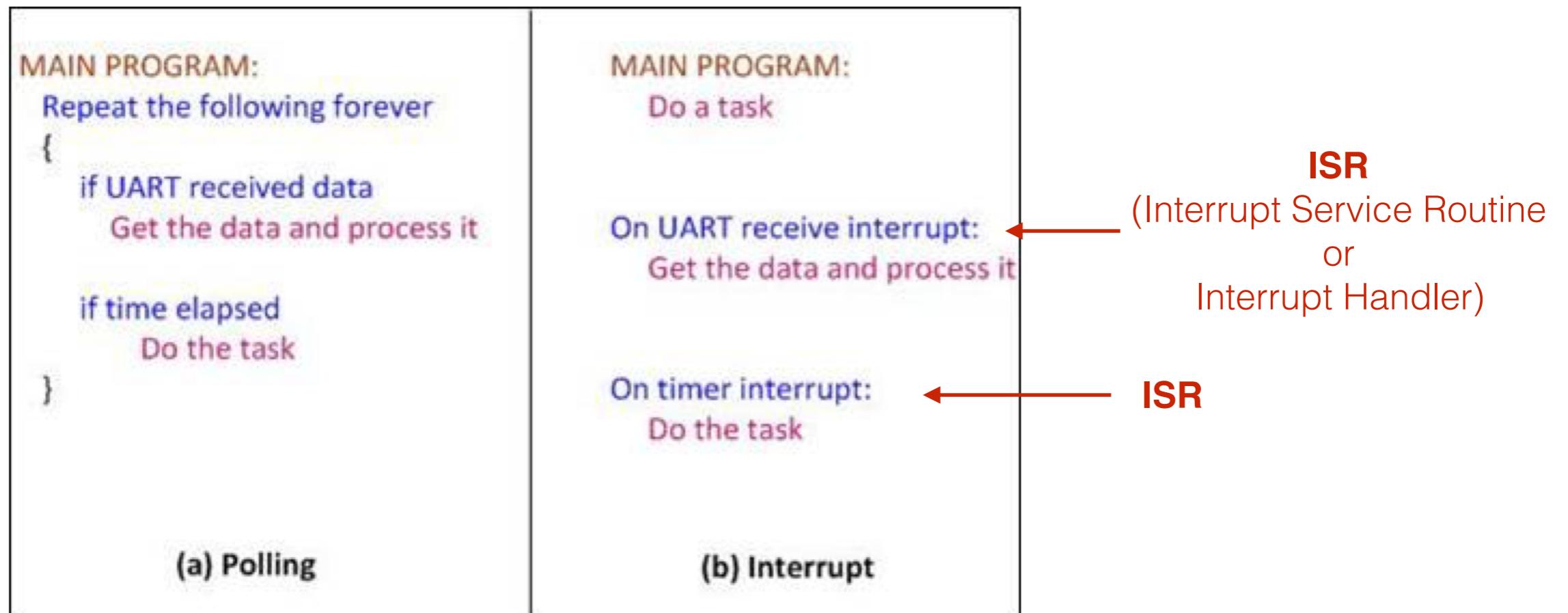
Lesson 06

Interrupts and Exceptions in ARM Cortex-M

인터럽트 (Interrupt)

- 현재 실행 중인 프로그램과는 별개로 비동기적으로 발생하는 이벤트(event)
- 인터럽트가 발생하면 CPU는 실행중인 프로그램을 중단하고 인터럽트 핸들러 (interrupt handler) 혹은 인터럽트 서비스 루틴 (interrupt service routine: ISR)이라고 불리는 특별한 코드를 실행
- ISR의 실행이 종료되면 일반적으로 원래 실행중이던 프로그램으로 복귀

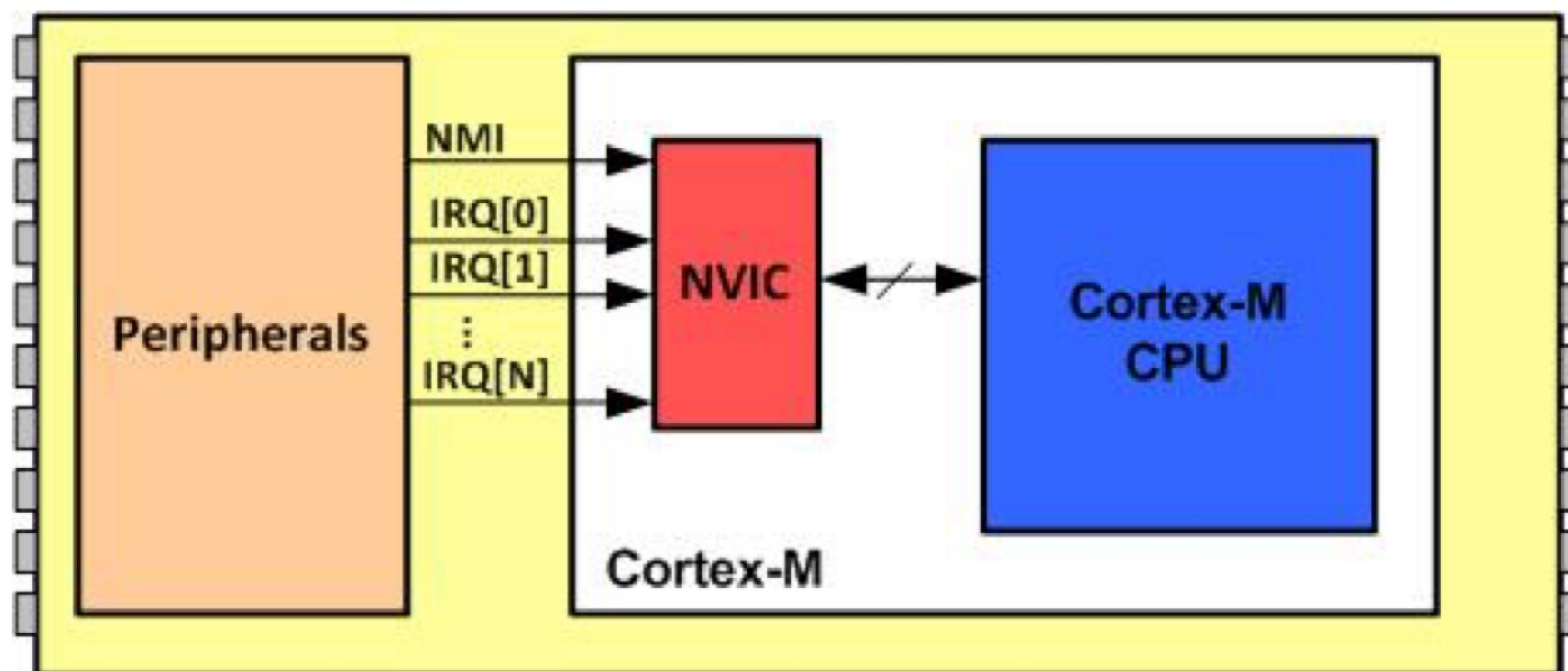
Interrupts vs. Polling



Some questions ?

- ◉ 인터럽트 발생시 어떤 핸들러를 실행해야 하는지 그리고 그 핸들러는 어디에 있는지 어떻게 알수 있나?
- ◉ 인터럽트 처리 후 어떻게 원래 수행하던 코드로 돌아올 수 있나?
- ◉ 인터럽트를 처리하는 도중에 다른 인터럽트가 발생하면 ?
- ◉ **Critical section**에 있는 도중에 인터럽트가 발생하면 ?

- Cortex-M4에서 인터럽트는 **Nested Vectored Interrupt Controller (NVIC)**이라고 불리는 장치에 의해 제어된다 (Classical ARM 혹은 Cortex-A와 Cortex-R 시리즈에서는 다른 방식으로 처리됨)
- 외부 장치로부터의 인터럽트를 허용하려면 NVIC에서 해당 장치의 인터럽트를 **enable**하고 우선순위(priority)를 지정해야 한다.



Cortex-M has on-chip interrupt controller called
NVIC (Nested Vector Interrupt Controller)

Interrupt Vector Table

- 인터럽트 핸들러의 주소를 저장하는 테이블
- 메모리의 최하위 1024 바이트에 위치
- 255개의 인터럽트에 대한 핸들러의 주소를 저장 (**0번지에는 SP의 초기값이 저장**)

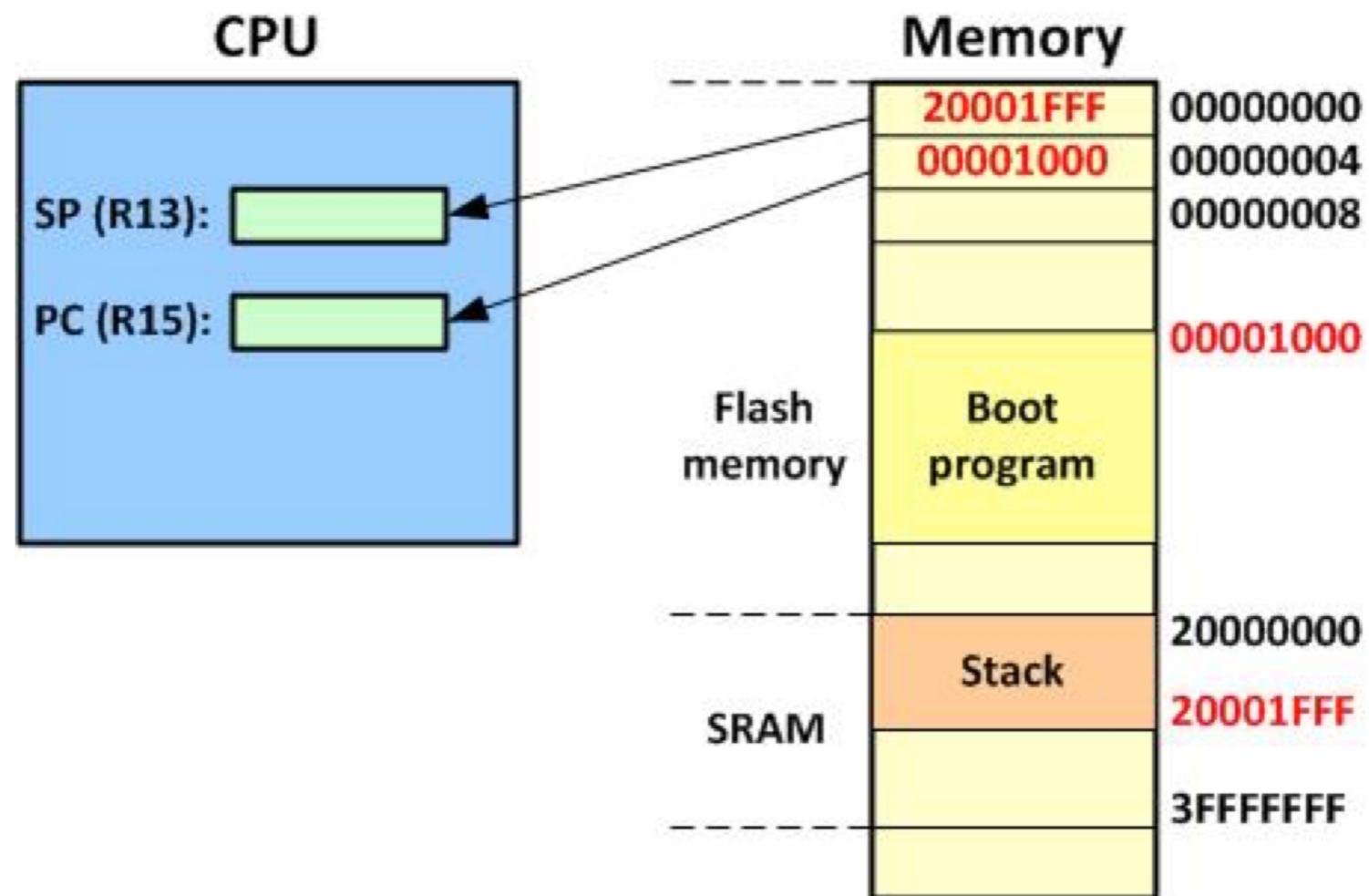
Interrupt Vector Table

Interrupt #	Interrupt	Memory Location (Hex)
	<i>Stack Pointer initial value</i>	0x00000000
1	Reset	0x00000004
2	NMI	0x00000008
3	Hard Fault	0x0000000C
4	Memory Management Fault	0x00000010
5	Bus Fault	0x00000014
6	Usage Fault (undefined instructions, divide by zero, unaligned memory access,...)	0x00000018
7	Reserved	0x0000001C
8	Reserved	0x00000020
9	Reserved	0x00000024
10	Reserved	0x00000028
11	SVCall	0x0000002C
12	Debug Monitor	0x00000030
13	Reserved	0x00000034
14	PendSV	0x00000038
15	SysTick	0x0000003C
16	IRQ for peripherals	0x00000040
17	IRQ for peripherals	0x00000044
...
255	IRQ for peripherals	0x000003FC

predefined
by Cortex-M

defined
by SOC chip

Predefined Interrupts: Reset Interrupt



Reset Interrupt의 처리

Predefined Interrupts

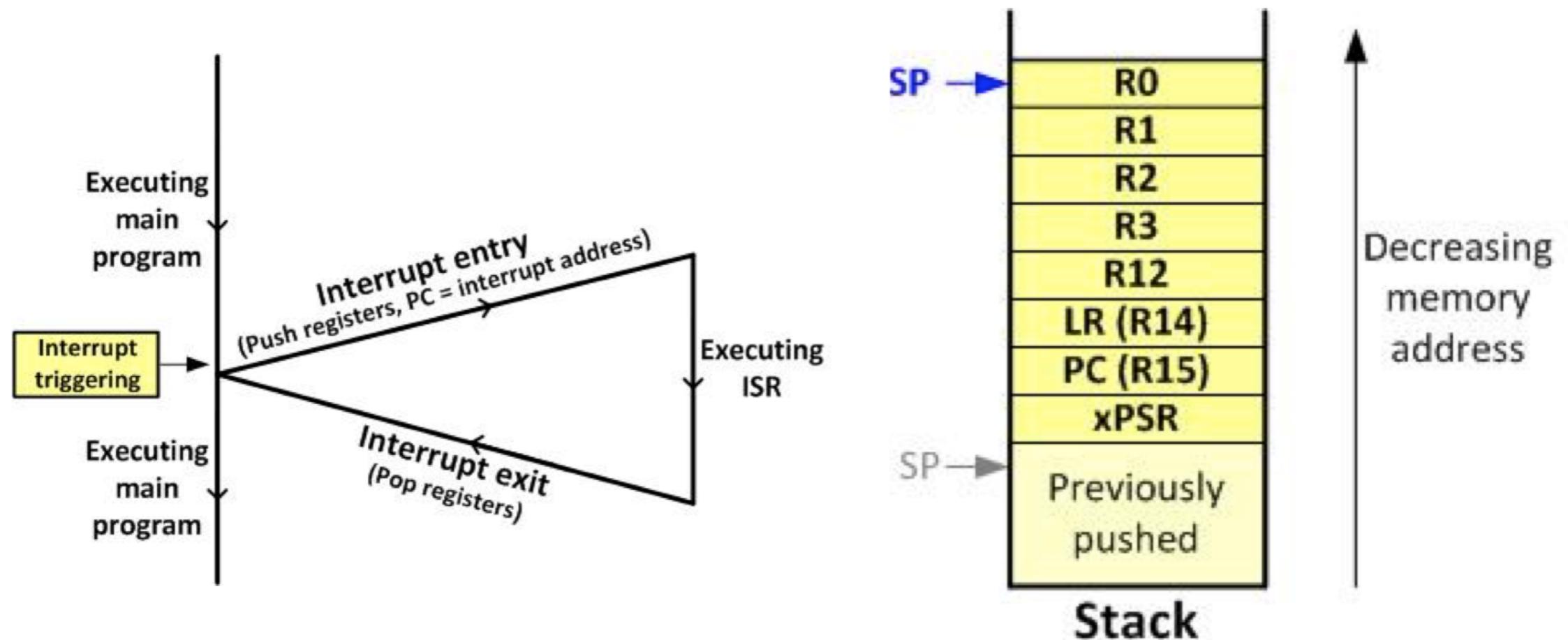
- ⦿ **Reset**
- ⦿ **Non-Maskable Interrupt (NMI)**
- ⦿ **Exceptions (Faults)**
 - ⦿ usage fault - divide by zero, unaligned memory access, undefined instruction
 - ⦿ bus fault
 - ⦿ hard fault - writing to clock-disabled register
 - ⦿ memory management fault - writing to read-only region, page fault
- ⦿ **SVCall (Supervisor call) and PendSV (pendable service call)**
 - ⦿ software interrupt
 - ⦿ used to gain a privileged access mode
- ⦿ **Debug Monitor**
 - ⦿ used to perform single-stepping by debugger
- ⦿ **SysTick**
 - ⦿ for multitasking OS

IRQ Peripheral Interrupts

- ⦿ **INT16 ~ INT255**
- ⦿ **Timer, ADC, Serial COM, 외부 하드웨어 인터럽트 등 다양한 주변 장치들에 할당 (SOC 칩마다 다름)**

Context Switch

- 인터넷트가 발생되면 **CPSR, PC, LR, R12, R3, R2, R1, R0** 레지스터의 값이 스택에 자동으로 저장됨
- ISR이 그 밖의 다른 레지스터를 사용할 경우 추가로 스택에 저장하고 종료 직전에 복원해야 한다.



ARM Cortex-M에서 인터럽트의 처리

1. **CPSR, PC, LR, R12, R3, R2, R1, R0** 레지스터의 값이 순서대로 스택에 push된다.
2. **Floating point coprocessor**의 레지스터 값들이 스택에 저장된다. (만약 lazy stacking이 enable되어 있으면 실제 값은 저장되지 않고 SP만 증가함)
3. CPU는 **Handler Mode**로 바뀌고 LR에는 31~5번째 비트가 모두 1인 주소가 저장된다.
4. **INT number**에 4를 곱한 값을 **PC**에 load하여 **ISR**로 분기한다.
5. **ISR**이 종료되면 **LR** 레지스터의 값을 보고 **Handler Mode**임을 인식한다. 스택에 저장해두었던 레지스터의 값을 복원하고 원래 실행하던 프로그램으로 복귀한다.

인터럽트 우선순위 (Interrupt Priority)

Interrupt #	Interrupt	Priority Level
0	<i>Stack Pointer initial value</i>	
1	Reset	-3 Highest
2	NMI	-2
3	Hard Fault	-1
4	Memory Management Fault	Programmable
5	Bus Fault	Programmable
6	Usage Fault (undefined instructions, divide by zero, unaligned memory access,...)	Programmable
7	Reserved	Programmable
8	Reserved	Programmable
9	Reserved	Programmable
10	Reserved	Programmable
11	SVCall	Programmable
12	Debug Monitor	Programmable
13	Reserved	Programmable
14	PendSV	Programmable
15	SysTick	Programmable
16	IRQ for peripherals	Programmable
17	IRQ for peripherals	Programmable
...	...	Programmable
255	IRQ for peripherals	Programmable

고정

설정가능
(0 ~ 7)

숫자가 작을 수록
높은 우선순위

인터럽트 우선순위 (Interrupt Priority)

- 동시에 여러 개의 인터럽트가 발생하면 가장 높은 우선순위를 가진 인터럽트가 먼저 처리되고 다른 인터럽트들은 대기해야 함
- 중첩된 인터럽트 (Nested Interrupt)
 - ISR을 실행하는 도중에 더 높은 우선순위의 인터럽트가 발생하면 실행중인 ISR을 중지하고 새로운 (높은 우선순위의) ISR이 실행됨

ARM Cortex-M Processor Modes

- ⦿ **ARM Cortex-M은 2가지 실행 모드를 가짐**
 - ⦿ Thread (Application) mode
 - ⦿ Handler (Exception) mode
- ⦿ **ARM Cortex-M이 처음 시작되어 reset 핸들러를 실행하고 빠져나오면 자동으로 Thread 모드가 됨**
- ⦿ **인터럽트가 발생하면 Handler 모드로 변경되고, ISR로부터 return되면 다시 Thread 모드로 돌아옴**
- ⦿ **LR 레지스터의 값으로 Handler 모드인지 Thread 모드인지 판단**

I/O Port Interrupt Programming

TM4C123H6MP.h: IRQ 할당

```
typedef enum {
/* ----- Cortex-M4 Processor Exceptions Numbers ----- */
    Reset_IRQn           = -15, /*!< 1 Reset Vector */
    NonMaskableInt_IRQn  = -14, /*!< 2 Non maskable Interrupt */
    HardFault_IRQn       = -13, /*!< 3 Hard Fault, all classes of Fault */
    MemoryManagement_IRQn= -12, /*!< 4 Memory Management, MPU mismatch */
    BusFault_IRQn        = -11, /*!< 5 Bus Fault, Memory Access Fault */
    UsageFault_IRQn      = -10, /*!< 6 Usage Fault, Undef Instruction */
    SVCall_IRQn          = -5,  /*!< 11 System Call via SVC instruction */
    DebugMonitor_IRQn    = -4,  /*!< 12 Debug Monitor */
    PendSV_IRQn          = -2,  /*!< 14 Pendable request */
    SysTick_IRQn         = -1,  /*!< 15 System Tick Timer */
/* ----- TM4C1232H6PM Specific Interrupt Numbers ----- */
    GPIOA_IRQn           = 0,   /*!< 0 GPIOA */
    GPIOB_IRQn           = 1,   /*!< 1 GPIOB */
    GPIOC_IRQn           = 2,   /*!< 2 GPIOC */
    GPIOD_IRQn           = 3,   /*!< 3 GPIOD */
    GPIOE_IRQn           = 4,   /*!< 4 GPIOE */
    UART0_IRQn           = 5,   /*!< 5 UART0 */
    UART1_IRQn           = 6,   /*!< 6 UART1 */
    SSI0_IRQn            = 7,   /*!< 7 SSI0 */
    I2C0_IRQn            = 8,   /*!< 8 I2C0 */}
```

TM4C123H6MP.h: IRQ 할당

ADC0SS0_IRQn	= 14,	/*!< 14 ADC0SS0 */
ADC0SS1_IRQn	= 15,	/*!< 15 ADC0SS1 */
ADC0SS2_IRQn	= 16,	/*!< 16 ADC0SS2 */
ADC0SS3_IRQn	= 17,	/*!< 17 ADC0SS3 */
WATCHDOG0_IRQn	= 18,	/*!< 18 WATCHDOG0 */
TIMER0A_IRQn	= 19,	/*!< 19 TIMER0A */
TIMER0B_IRQn	= 20,	/*!< 20 TIMER0B */
TIMER1A_IRQn	= 21,	/*!< 21 TIMER1A */
TIMER1B_IRQn	= 22,	/*!< 22 TIMER1B */
TIMER2A_IRQn	= 23,	/*!< 23 TIMER2A */
TIMER2B_IRQn	= 24,	/*!< 24 TIMER2B */
COMP0_IRQn	= 25,	/*!< 25 COMP0 */
COMP1_IRQn	= 26,	/*!< 26 COMP1 */
SYSCTL_IRQn	= 28,	/*!< 28 SYSCTL */
FLASH_CTRL_IRQn	= 29,	/*!< 29 FLASH_CTRL */
GPIOF_IRQn	= 30,	/*!< 30 GPIOF */
GPIOG_IRQn	= 31,	/*!< 31 GPIOG */
UART2_IRQn	= 33,	/*!< 33 UART2 */

} IRQn_Type;

GPIO Port F의 모든 핀에게 1개의 IRQ가 할당되어 있음
따라서 어떤 핀인지를 알아내는 것은 ISR이 해야할 일

PORTRF 인터럽트 레지스터

- 시스템이 시작될 때 모든 인터럽트는 기본적으로 **disable**됨
- 인터럽트를 **enable**하기 위한 3단계:

1. Enable the interrupt **for a specific peripheral module**
2. Enable the interrupts **at the NVIC module**
3. Enable the interrupt **globally**

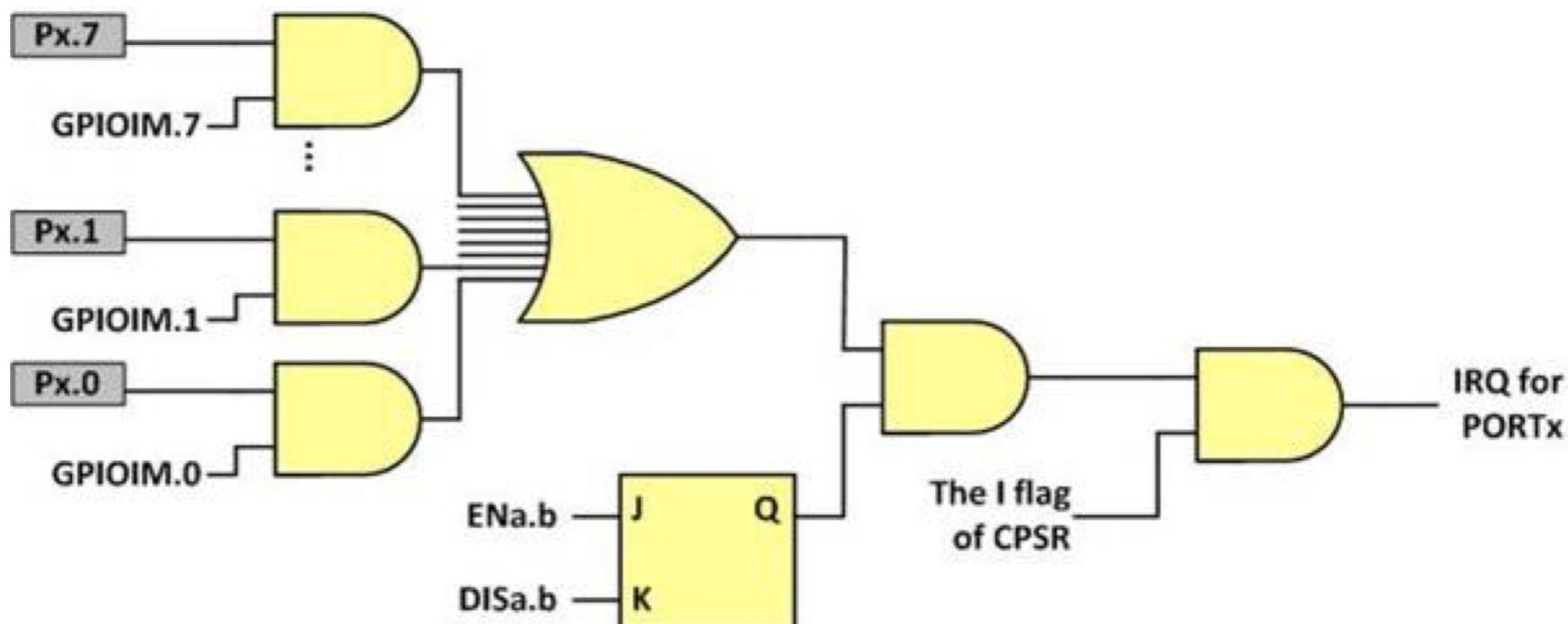


Figure 6-8: Interrupt enabling with all 3 levels

제1단계: GPIOIM

- I/O Port의 인터럽트를 enable하기 위해서 **GPIO Interrupt Mask (GPIOIM)** 레지스터를 설정해야 함

GPIOIM:	D31	D8	D7	D6	D5	D4	D3	D2	D1	D0	0x410
		Reserved		IME Px.7	IME Px.6	IME Px.5	IME Px.4	IME Px.3	IME Px.2	IME Px.1	IME Px.0	

Note: D0 to D7 are used to enable/disable the interrupt for pins 0 to 7 of the port.
1: Enable interrupt
0: Disable interrupt (mask the interrupt)

Figure 6-9: GPIO Interrupt Mask (GPIOIM)

GPIOF->IM |= 0x1

PF0핀의 인터럽트를 enable

제2단계: NVIC Registers

EN0 (ISER[0]):	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0x100															

Figure 6-10: Interrupts 0–31 Set Enable (EN0)

EN1 (ISER[1]):	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
	0x104															

Figure 6-11: Interrupts 32–63 Set Enable (EN1)

EN2 (ISER[2]):	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
	0x108															

Figure 6-12: Interrupts 64–95 Set Enable (EN2)

⋮
⋮

```
NVIC->ISER[0] |= 0x40000000 /* To enable INT30 */
```

제2단계: NVIC Registers

인터럽트를 enable하려면
Interrupt Set Enable (ISER)에 써야한다.



```
NVIC->ISER[0] |= 0x40000000  
/* enable INT30 */
```

혹은

```
void NVIC_EnableIRQ(30);
```

혹은

```
void NVIC_EnableIRQ(GPIOF_IRQn);
```



defined in core_cm4.h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0x100

Figure 6-10: Interrupts 0–31 Set Enable (EN0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

0x104

Figure 6-11: Interrupts 32–63 Set Enable (EN1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64

0x108

Figure 6-12: Interrupts 64–95 Set Enable (EN2)

⋮

Disable Interrupts

인터럽트를 disable하려면
Interrupt Clear Enable (ICER)에 써야한다.

NVIC->ICER[0] |= (1<<5)
/* disable UART0 Interrupt */

혹은

void NVIC_DisableIRQ(UART0_IRQn);

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
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

Figure 6-14: Interrupts 0–31 Clear Enable (DIS0)

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
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Figure 6-15: Interrupts 32–63 Clear Enable (DIS1)

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
95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64

•
•
•

제3단계: Global Interrupt Enable/Disable

- critical한 작업을 하는 동안 하나의 instruction으로 모든 인터럽트를 mask하기 위해서 제공되는 기능
- CMSIS의 pseudo-function

```
__enable_irq();      /* Enable interrupt Globally */  
__disable_irq();    /* Disable interrupt Globally */
```

Summary: Enable the Interrupts for PF0

```
GPIOF->IM |= 0x11          /* unmask interrupts for PF4 and PF0 */  
NVIC->ISER[0] = 0x40000000; /*enable INT30 (bit 30 of ISER[0] */  
_enable_irq();           /* global enable IRQs */
```

- ⦿ **NVIC 레지스터**
- ⦿ 각각의 **IRQ number**에 대해서 **1byte**로 우선순위 지정
- ⦿ 가장 왼쪽, 즉 상위 3비트가 **0~7까지의 우선순위를 지정**

```
NVIC->IP[ INTERRUPT_NUMBER ] |= PRIORITY << 5;  
NVIC->IP[30] |= 3 << 5; /* set priority level 3 */
```

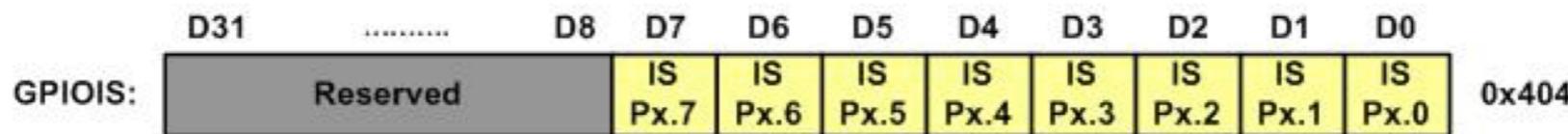
Interrupt Trigger Point

⦿ 5가지 중에 선택

1. low-level trigger
2. high-level trigger
3. rising-edge-trigger
4. falling-edge trigger
5. both edge (rising and falling) trigger

Interrupt Trigger Point

- 먼저 GPIO Interrupt Sense (GPIOIS) 레지스터로 level인지 edge인지 선택



Note: D0 to D7 are used to choose between edge and level sense for pins 0 to 7 of the port.

0: Edge-sensitive

1: Level-sensitive

Figure 6-19: GPIO Interrupt Sense (GPIOIS)

- GPIO Interrupt Event (GPIOIEV) 레지스터로 low인지 high인지, 혹은 falling인지 rising인지 선택



Note: D0 to D7 are used to choose between edge and level sense for pins 0 to 7 of the port.

0: A falling edge or a Low level on the corresponding pin triggers an interrupt

1: A rising edge or a High level on the corresponding pin triggers an interrupt

Interrupt Trigger Point

IS.n (interrupt sense)	IEV.n (Interrupt Event)	
0	0	Falling edge
0	1	Rising edge
1	0	Low level
1	1	High level

```
GPIOF->IS &= ~0x11 /* mask bit 4, 0, edge sensitive */
```

```
GPIOF->IBE &= ~0x11
```

← GPIO Interrupt Both Edges (GPIOIBE)를 clear해주지
않으면 무조건 both edge trigger가 됨

```
GPIOF->IEV &= ~0x11 /* falling edge trigger */
```

Toggle Red LED

```
/* Toggle red LED on PF1 continuously. Upon pressing either SW1 or SW2, the green  
LED of PF3 should toggle for three times.*/
```

```
#include "TM4C123GH6PM.h"
```

```
void delayMs(int n);
```

```
int main(void)
```

```
{
```

```
    SYSCTL->RCGCGPIO |= 0x20; /* enable clock to PORTF */
```

```
    /* PORTF0 has special function, need to unlock to modify */
```

```
    GPIOF->LOCK = 0x4C4F434B; /* unlock commit register */
```

```
    GPIOF->CR = 0x01; /* make PORTF0 configurable */
```

```
    GPIOF->LOCK = 0; /* lock commit register */ ← Commit Control
```

```
    /* configure PORTF for switch input and LED output */
```

```
    GPIOF->DIR &= ~0x11; /* make PORTF0 and 4 input for switch */
```

```
    GPIOF->DIR |= 0x0E; /* make PORTF3, 2, 1 output for LEDs */
```

```
    GPIOF->DEN |= 0x1F; /* make PORTF4-0 digital pins */
```

```
    GPIOF->PUR |= 0x11; /* enable pull up for PORTF0 and 4 */
```



Toggle Red LED

```
/* configure PORTF4, 0 for falling edge trigger interrupt */
GPIOF->IS  &= ~0x11;          /* make bit 4, 0 edge sensitive */
GPIOF->IBE &= ~0x11;          /* trigger is controlled by IEV */
GPIOF->IEV &= ~0x11;          /* falling edge trigger */
GPIOF->ICR |= 0x11;          /* clear any prior interrupt */
GPIOF->IM  |= 0x11;          /* unmask interrupt */

/* enable interrupt in NVIC and set priority to 3 */
NVIC->IP[30] = 3 << 5;      /* set interrupt priority to 3 */
NVIC->ISER[0] |= 0x40000000; /* enable IRQ30 (D30 of ISER[0]) */

__enable_irq(); /* global enable IRQs */

/* toggle the red LED (PF1) continuously */
while(1)
{
    GPIOF->DATA |= 0x02;
    delayMs(500);
    GPIOF->DATA &= ~0x02;
    delayMs(500);
}
}
```

Toggle Red LED

```
/* SW1 is connected to PF4 pin, SW2 is connected to PF0. */
/* Both of them trigger PORTF interrupt */
void GPIOF_Handler(void)
{
    int i;
    volatile int readback;
    /* toggle green LED (PF3) three times */
    for (i = 0; i < 3; i++)
    {
        GPIOF->DATA |= 0x08;
        delayMs(500);
        GPIOF->DATA &= ~0x08;
        delayMs(500);
    }
    GPIOF->ICR |= 0x11;           /* clear the interrupt flag before return */
    readback = GPIOF->ICR;        /* a read to force clearing of interrupt flag */
}

/* delay n milliseconds (16 MHz CPU clock) */
void delayMs(int n)
{
    int i, j;
    for(i = 0 ; i < n; i++)
        for(j = 0; j < 3180; j++)
    {} /* do nothing for 1 ms */
}
```

ISR에서 return하기 전에 interrupt flag를 clear해주어야 한다. GPIO Interrupt Clear Register에 1을 써주면 clear된다. 하지만 이 write는 buffered되므로 즉시 효과를 발휘하지 않는다. 그래서 flag를 한 번 read해준다.



TM4C123H6MP.h의 수정

```
/* PORTF0 has special function, need to unlock to modify */
GPIOF->LOCK = 0x4C4F434B; /* unlock commit register */
GPIOF->CR = 0x01;          /* make PORTF0 configurable */
GPIOF->LOCK = 0;           /* lock commit register */
```

GPIO Commit Control

- 특정 GPIO 핀들은 특별한 기능으로 할당되어 있음 (PC[3:0]은 JTAG/SWD, PD7과 PF0는 NMI)
- 이 핀들에 대해서 GPIOAFSEL, GPIOPUR, GPIODEN 등의 설정은 보호되어 있음
- 이를 해제하려면 GPIOLOCK을 unlock하고 GPIOCR을 set해야 함

GPIOA_Type 구조체에서

`__I uint32_t CR; /*!< GPIO Commit`

을 찾아서

`__IO uint32_t CR; /*!< GPIO Commit`

로 수정한다.

Startup Code에서

```
#include <stdint.h>

void ResetISR(void);
static void NmiISR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);
```

```
extern void GPIOF_Handler(void);
```

•

GPIOF 인터럽트 핸들러의
prototype을 추가해준다.

Startup Code에서

```
__attribute__ ((section(".intvecs")))
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)pui32Stack + sizeof(pui32Stack)),
                                // The initial stack pointer
    ResetISR,
                                // The reset handler
    NmiSR,
                                // The NMI handler
    FaultISR,
                                // The hard fault handler
    IntDefaultHandler,
                                // The MPU fault handler
    IntDefaultHandler,
                                // The bus fault handler
    IntDefaultHandler,
                                // The usage fault handler
    0,
                                // Reserved
    0,
                                // Reserved
    0,
                                // Reserved
    0,
                                // Reserved
    :
    :
    IntDefaultHandler,
                                // Analog Comparator 0
    IntDefaultHandler,
                                // Analog Comparator 1
    IntDefaultHandler,
                                // Analog Comparator 2
    IntDefaultHandler,
                                // System Control (PLL, OSC, B0)
    IntDefaultHandler,
                                // FLASH Control
    GPIOF_Handler,           // GPIO Port F
    IntDefaultHandler,
                                // GPIO Port G
    IntDefaultHandler,
                                // GPIO Port H
    IntDefaultHandler,
                                // UART2 Rx and Tx
    IntDefaultHandler,          // SSI1 Rx and Tx
```

30번 IRQ의 위치에 핸들러의 주소를 넣는다. 이제 실행해본다.

- ⓐ 앞의 **GPIOF_Handler**에서는 어떤 핀에서 인터럽트가 발생했는지 구분하지 않는다.
- ⓑ **GPIOF_Handler**를 수정하여 **SW1**을 눌렀는지 **SW2**를 눌렀는지 구분해서 다른 **LED**가 점멸되도록 해보자.
- ⓒ 어떤 핀에서 인터럽트가 발생했는지 알기 위해서 **GPIO Masked Interrupt Status (GPIO MIS)** 레지스터를 검사한다.

입터럽트 핀의 분별: GPIOF MIS

```
/* SW1 is connected to PF4 pin, SW2 is connected to PF0. Both of them trigger PORTF
interrupt */
void GPIOF_Handler(void)
{
    int i;
    volatile int readback;

    while (GPIOF->MIS != 0)
    {
        if (GPIOF->MIS & 0x10) /* is it SW1(PF4)? */
        {
            /* GPIOF4 pin interrupt */
            /* toggle green LED (PF3) three times */
            for (i = 0; i < 3; i++)
            {
                GPIOF->DATA |= 0x08;
                delayMs(500);
                GPIOF->DATA &= ~0x08;
                delayMs(500);
            }
            GPIOF->ICR |= 0x10;          /* clear the interrupt flag */
            readback = GPIOF->ICR;      /* a read to force clearing of
                                         interrupt flag */
        }
    }
}
```

Each of PORTF pin interrupt sets a pin in the GPIOF MIS register. The ISR can poll the bits of GPIOF MIS to find out which pin is requesting interrupt.

입터럽트 핀의 분별: GPIOF MIS

```
else if (GPIOF->MIS & 0x01) /* then it must be SW2(PF0) */
{
    /* GPIOF0 pin interrupt */
    /* toggle blue LED (PF2) three times */
    for (i = 0; i < 3; i++)
    {
        GPIOF->DATA |= 0x04;
        delayMs(500);
        GPIOF->DATA &= ~0x04;
        delayMs(500);
    }
    GPIOF->ICR |= 0x01;      /* clear the interrupt flag */
    readback = GPIOF->ICR;  /* a read to force clearing of
                                interrupt flag */
}
else
{
    /* We should never get here. */
    /* But if we do, clear all pending interrupts. */
    GPIOF->ICR = GPIOF->MIS;
    readback = GPIOF->ICR;  /* a read to force clearing of
                                interrupt flag */
}
}
```

- ◎ SW1 버튼을 누를 때와 뗄 때 인터럽트가 발생하여 (즉, **both edge**에 인터럽트가 발생하여), 누를 때 red LED가 켜지고 뗄 때 red LED가 꺼지도록 하라.
Both edge 인터럽트를 사용하려면 **GPIO Interrupt Both Edges (GPIOIBE)** 레지스터의 해당 비트를 **set**해야 한다.

Summary

```
typedef struct {  
    ...  
  
    __IO uint32_t IS;          /* GPIO Interrupt Sense: Level 혹은 Edge 선택 */  
    __IO uint32_t IBE;         /* GPIO Interrupt Both Edges */  
    __IO uint32_t IEV;         /* GPIO Interrupt Event: Low, High, Falling, or Rising ? */  
    __IO uint32_t IM;          /* GPIO Interrupt Mask: Enable/Disable Interrupt */  
    __IO uint32_t RIS;         /* GPIO Raw Interrupt Status: 인터럽트가 mask되어있든 안되어  
                               있든 상관없이 인터럽트가 발생하면 set된다. */  
    __IO uint32_t MIS;         /* GPIO Masked Interrupt Status: IM & RIS */  
    __O uint32_t ICR;          /* GPIO Interrupt Clear: Clear prior interrupt */  
    ...  
} GPIOA_Type;
```

UART Serial Port Interrupt Programming

Enable UART Interrupt

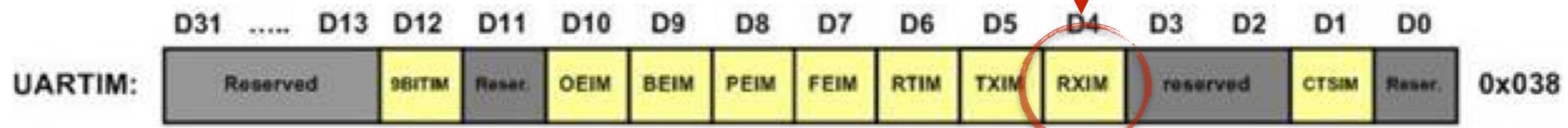
- ❸ 3장에서는 **UARTFR** 레지스터의 **TXFF**와 **RXFE** 플래그를 **polling**함으로써 송수신을 하였다.
- ❹ 여기에서는 인터럽트 기반으로 동작하도록 해보자.

UART 레지스터들

```
typedef struct {  
    ...  
    __IO uint32_t FR;           /* UART Flag */  
    __I  uint32_t RESERVED1;  
    __IO uint32_t ILPR;         /* UART IrDA Low-Power Register */  
    __IO uint32_t IBRD;         /* UART Integer Baud-Rate Divisor */  
    __IO uint32_t FBRD;         /* UART Fractional Baud-Rate Divisor */  
    __IO uint32_t LCRH;         /* UART Line Control */  
    __IO uint32_t CTL;          /* UART Control */  
    __IO uint32_t IFLS;         /* UART Interrupt FIFO Level Select */  
    __IO uint32_t IM;           /* UART Interrupt Mask */  
    __IO uint32_t RIS;          /* UART Raw Interrupt Status */  
    __IO uint32_t MIS;          /* UART Masked Interrupt Status */  
    __O  uint32_t ICR;          /* UART Interrupt Clear */  
    ...  
} UART0_Type;
```

Enable UART Interrupt

수신 인터럽트 enable



bit	Name	Description
1	CTSIM	1: an interrupt is made when the CTSRIS bit of UARTRIS is set, 0: masked (disabled)
4	RXIM	1: an interrupt is made when the RXRIS bit of UARTRIS is set, 0: masked (disabled)
5	TXIM	1: an interrupt is made when the TXRIS bit of UARTRIS is set, 0: masked (disabled)
6	RTIM	1: an interrupt is made when the RTSRIS bit of UARTRIS is set, 0: masked (disabled)
7	FEIM	1: an interrupt is made when the FERIS bit of UARTRIS is set, 0: masked (disabled)
8	PEIM	1: an interrupt is made when the PERIS bit of UARTRIS is set, 0: masked (disabled)
9	BEIM	1: an interrupt is made when the BERIS bit of UARTRIS is set, 0: masked (disabled)
10	OEIM	1: an interrupt is made when the OERIS bit of UARTRIS is set, 0: masked (disabled)
12	9BITIM	1: an interrupt is made when the 9BITRIS bit of UARTRIS is set, 0: masked (disabled)

Receiving Data

- **IRQ5 is assigned to UART0**
- **Enable the receive interrupt in UART0:**

```
UART0->IM |= 0x0010; /* enable RX interrupt */  
NVIC->ISER[0] |= 0x00000020; /* enable IRQ5 */  
__enable_irq();
```

Using UART0 Interrupt

```
#include "tm4c123gh6pm.h"

int main(void)
{
    SYSCTL->RCGCUART |= 1;      /* provide clock to UART0 */
    SYSCTL->RCGCGPIO |= 1;      /* enable clock to PORTA */

    SYSCTL->RCGCGPIO |= 0x20;   /* enable clock to PORTF */

    /* UART0 initialization */
    UART0->CTL = 0;            /* disable UART0 */
    UART0->IBRD = 104;          /* 16MHz/16=1MHz, 1MHz/104=9600 baud rate */
    UART0->FBRD = 11;           /* fraction part, see Example 4-4 */
    UART0->CC = 0;              /* use system clock */
    UART0->LCRH = 0x60;         /* 8-bit, no parity, 1-stop bit, no FIFO */

    UART0->IM |= 0x0010;        /* enable RX interrupt */

    UART0->CTL = 0x301;         /* enable UART0, TXE, RXE */
}
```

UART0를 사용하기 위해서는
UART0와 GPIOA 포트에
클럭을 제공해야 한다,

LED를 점멸하기 위해서
GPIOF에 clock을 제공

3장의 코드에서 추가된 부분.
인터럽트를 enable한다.

Using UART0 Interrupt

```
/* UART0 TX0 and RX0 use PA0 and PA1. Set them up. */
GPIOA->DEN = 0x03;          /* Make PA0 and PA1 as digital */
GPIOA->AFSEL = 0x03;        /* Use PA0,PA1 alternate function */
GPIOA->PCTL = 0x11;         /* configure PA0 and PA1 for UART */

GPIOF->DIR = 0x0E;          /* 핀 1~3을 출력으로 설정 */
GPIOF->DEN = 0x0E;          /* 핀 1~3을 digital enable */
GPIOF->DATA = 0;

/* enable interrupt in NVIC and set priority to 6 */
NVIC->IP[5] = 3 << 5;      /* set interrupt priority to 3 */
NVIC->ISER[0] |= 0x00000020; /* enable IRQ5 for UART0 */

__enable_irq(); /* global enable IRQs */

for(;;)
{
}

}
```

UART0_Handler

GPIOF_Handler와 마찬가지로 startup code를 수정한다.

```
void UART0_Handler(void)
{
    volatile int readback;
    char c;

    if ( UART0->MIS & 0x0010 )
    {
        c = UART0->DR;          /* read the received data */
        GPIOF->DATA = c << 1;   /* shift left and write it to LEDs */
        UART0->ICR = 0x0010;     /* clear Rx interrupt flag */
        readback = UART0->ICR;  /* a read to force clearing of interrupt flag */
    }
    else
    {
        /* should not get here. But if it does, */
        UART0->ICR = UART0->MIS; /* clear all interrupt flags */
        readback = UART0->ICR;
    }
}
```

Masked Interrupt Status (MIS) 레지스터의 D4비트를 검사

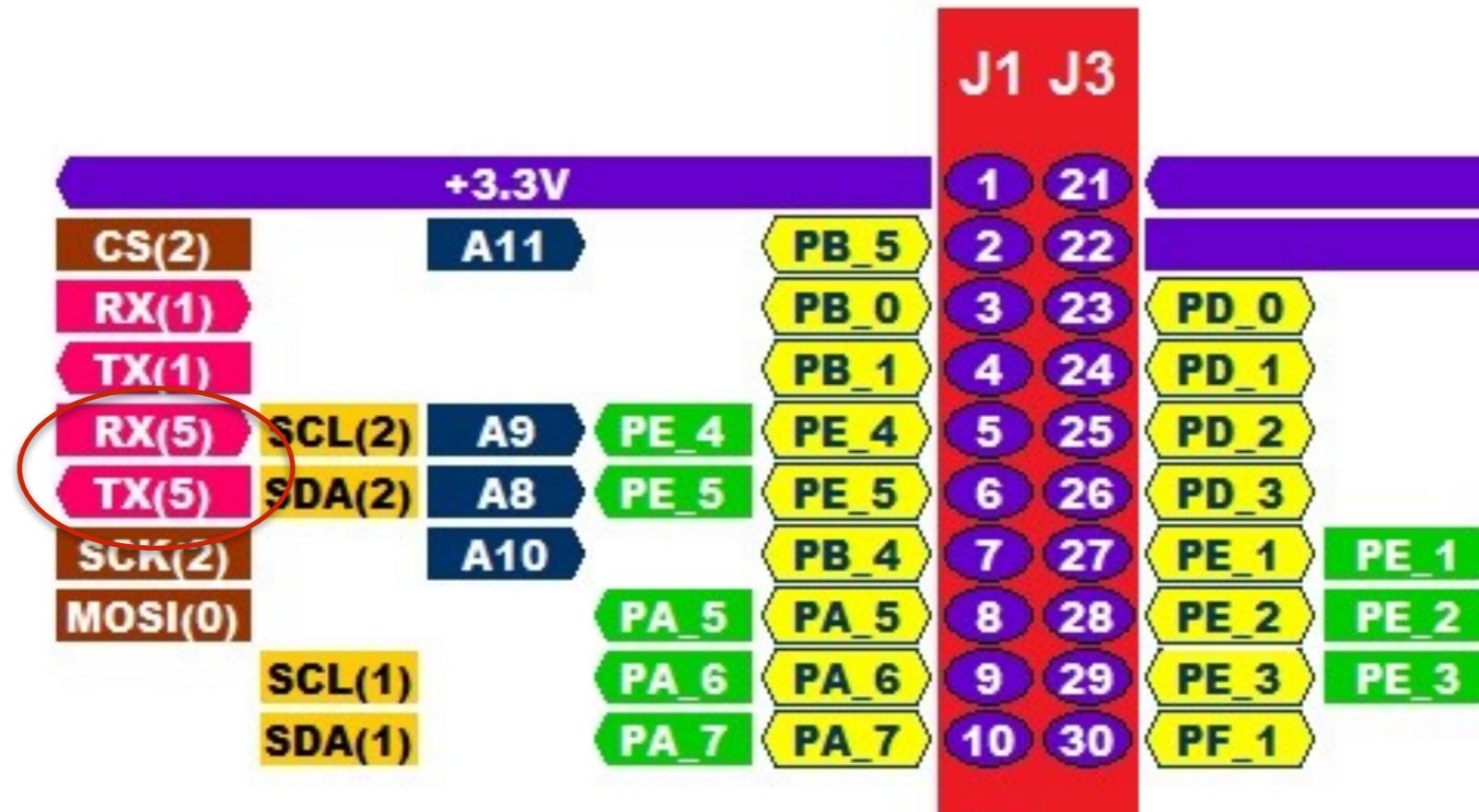
실습과제

- 3장에서 구현한 **serial.c** 프로그램을 인터럽트 기반으로 수정하도록 수정하라.

UART0과 UART5를 사용하여 노트북과 통신하기

- Serial-To-USB 변환 케이블을 이용하여 **UART5**를 노트북과 연결한다.
UART0로 노트북으로부터 받은 메시지를 **UART5**로 다시 노트북으로 전송하는
프로그램을 작성하라.
- Launchpad**가 두 장치간에 **relay**의 역할을 하는 셈이다. 적절한 변환 케이블이
있으면 한쪽을 **Arduino, GPS** 장치 등으로 대체할 수도 있다.

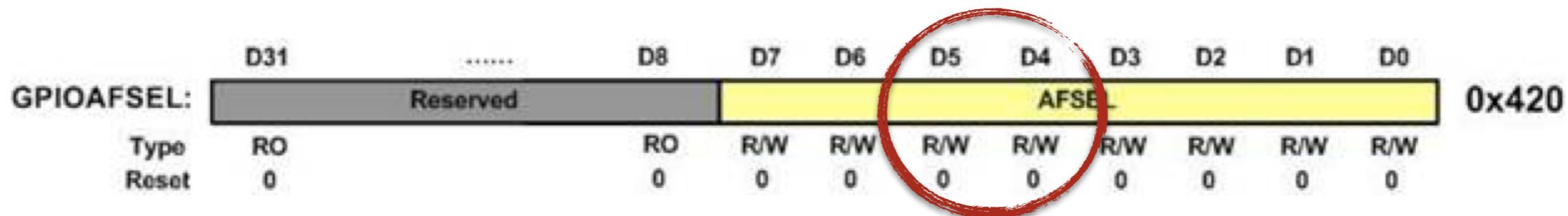
UART5의 사용



- UART5는 PE_4, PE_5번 핀을 사용한다.

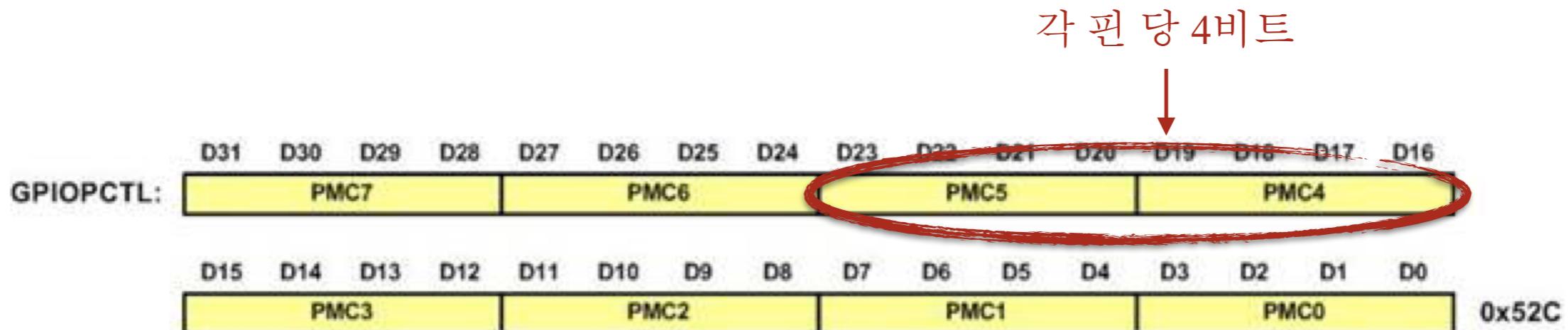
PE5번과 PE5번 핀을 UART5로 사용하기 위해서는

- GPIO Alternate Function을 사용하기 위해 먼저 GPIOAFSEL 레지스터에서 alternate function을 enable하고



$$\text{GPIOE-} \rightarrow \text{AFSEL} = 0x30;$$

- GPIOPCTL 레지스터에서 어떤 alternate function을 사용할지 지정해야함



GPIOPCTL 레지스터

PE4와 PE5를 UART5 Rx와 Tx로 사용하려면 0001을
GPIOPCTL의 D16~D19와 D17~D20에 각각 저장
GPIOE->PCTL = 0x00110000;



Pin	Digital Function (GPIOPCTL PMCx)															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
17	PA0	U0Rx							CAN1Rx							
18	PA1	U0Tx							CAN1Tx							
45	PB0	U1Rx						T2CCPO								
46	PB1	U1Tx						T2CCP1								
16	PC4	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCPO	U1RTS							
15	PC5	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS							
14	PC6	U3Rx					PhB1	WT1CCPO	USB0EPEN							
13	PC7	U3Tx						WT1CCP1	USB0PFLT							
43	PD4	U6Rx						WT4CCPO								
44	PD5	U6Tx						WT4CCP1								
53	PD6	U2Rx		M0FAULT0		phA0	WT5CCPO									
10	PD7	U2Tx					phB0	WT5CCP1	NMI							
9	PE0	U7Rx								USB0EPEN						
8	PE1	U7Tx								USB0PFLT						
59	PE4	U5Rx		I2C2SCL	M0PWM4	M1PWM2			CAN0Rx							
60	PE5	U5Tx		I2C2SDA	M0PWM5	M1PWM3			CAN0Tx							
28	PF0	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	PhA0	TOCCPO	NMI							
29	PF1	U1CTS	SSI1Tx			M1PWM5	PhB0	TO CCP1								

```
#include "tm4c123gh6pm.h"
#include “queue.h”
#include “serial.h”

int main(void)
{
    init_UART0();
    init_UART5();
    __enable_irq(); /* global enable IRQs */

    while(1) {
        if (!isEmpty()) {
            char ch = dequeue(); ← FIFO에서 dequeue한다.
            UART5Tx(ch);
        }
    }
}
```

serial.h

```
#ifndef SERIAL_H
#define SERIAL_H

#include "TM4C123GH6PM.h"

void init_UART0(void);
void init_UART5(void);
void UART5Tx(char c);
#endif
```

```
#include "serial.h"
void init_UART0()
{
    SYSCTL->RCGCUART |= (1<<0);
    SYSCTL->RCGCGPIO |= (1<<0);
    GPIOA->AFSEL = (1<<1)|(1<<0);
    GPIOA->PCTL = (1<<0)|(1<<4);
    GPIOA->DEN = (1<<0)|(1<<1);
    UART0->CTL &= ~(1<<0);

    UART0->IBRD = 104;
    UART0->FBRD = 11;

    UART0->LCRH = (0x3<<5)|(1<<4); // 8-bit, no parity, 1-stop bit, FIFO enable
    UART0->CC = 0x0;
    UART0->IM |= 0x0010; /* enable RX interrupt */

    UART0->CTL = (1<<0)|(1<<8)|(1<<9);
    /* enable interrupt in NVIC and set priority to 6 */
    NVIC->IP[5] = 3 << 5; /* set interrupt priority to 4 */
    NVIC->ISER[0] |= 0x00000020; /* enable IRQ6 for UART0 */
}
```

```
void init_UART5() {
    SYSCTL->RCGCUART |= 0x20; /* provide clock to UART5 */
    SYSCTL->RCGCGPIO |= 0x10; /* Enable clock to PORTE */
    /* UART5 initialization */
    UART5->CTL = 0;
    UART5->IBRD = 104;
    UART5->FBRD = 11;
    UART5->CC = 0;
    UART5->LCRH = 0x60;
    UART5->CTL = 0x301;      /* enable UART0, TXE, RXE */

    GPIOE->DEN = 0x30;
    GPIOE->AMSEL = 0;
    GPIOE->AFSEL = 0x30;
    GPIOE->PCTL = 0x00110000; /* configure PE5, PE4 for UART5 */
}
```

```

void UART5Tx(char c)
{
    while((UART5->FR & 0x20) != 0); /* wait until Tx buffer not full */
    UART5->DR = c;                  /* before giving it another byte */
}

void UART0_Handler(void)
{
    volatile int readback;

    char c;
    if (UART0->MIS & 0x0010) {
        c = UART0->DR;                      /* read the received data */
        enqueue(c);                         ← FIFO에 enqueue한다.
        UART0->ICR = 0x0010;                  /* clear Rx interrupt flag */
        readback = UART0->ICR;                /* a read to force clearing of interrupt flag */
    }
    else {
        /* should not get here. But if it does, */
        UART0->ICR = UART0->MIS; /* clear all interrupt flags */
        readback = UART0->ICR;    /* a read to force clearing of interrupt flag */
    }
}

```

queue.h

```
#ifndef QUEUE_H_
#define QUEUE_H_

int enqueue(char ch);
int dequeue();
int isEmpty();

#endif /* QUEUE_H_ */
```

queue.c

```
#define QUEUE_SIZE 100
char queue[QUEUE_SIZE]; // circular array
int front = 0, rear = 0, size = 0;

int enqueue(char ch) {
    if (size == QUEUE_SIZE) return 0;
    rear = (rear+1)%QUEUE_SIZE;
    queue[rear] = ch;
    size++;
    return 1;
}

int dequeue() {
    if (size == 0) return -1;
    char result = queue[front];
    front = (front+1)%QUEUE_SIZE;
    size--;
    return result;
}

int isEmpty() {
    return size==0;
}
```

Startup Code에서

- UART0 인터럽트 핸들러를 **interrupt vector table**에 추가한다.

USB to TTL Serial Cable



The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at the end of the 36" cable are four wire - red power, black ground, white RX into USB port, and green TX out of the USB port. The power pin provides the 5V @ 500mA direct from the USB port and the RX/TX pins are 3.3V level for interfacing with the most common 3.3V logic level chipsets.

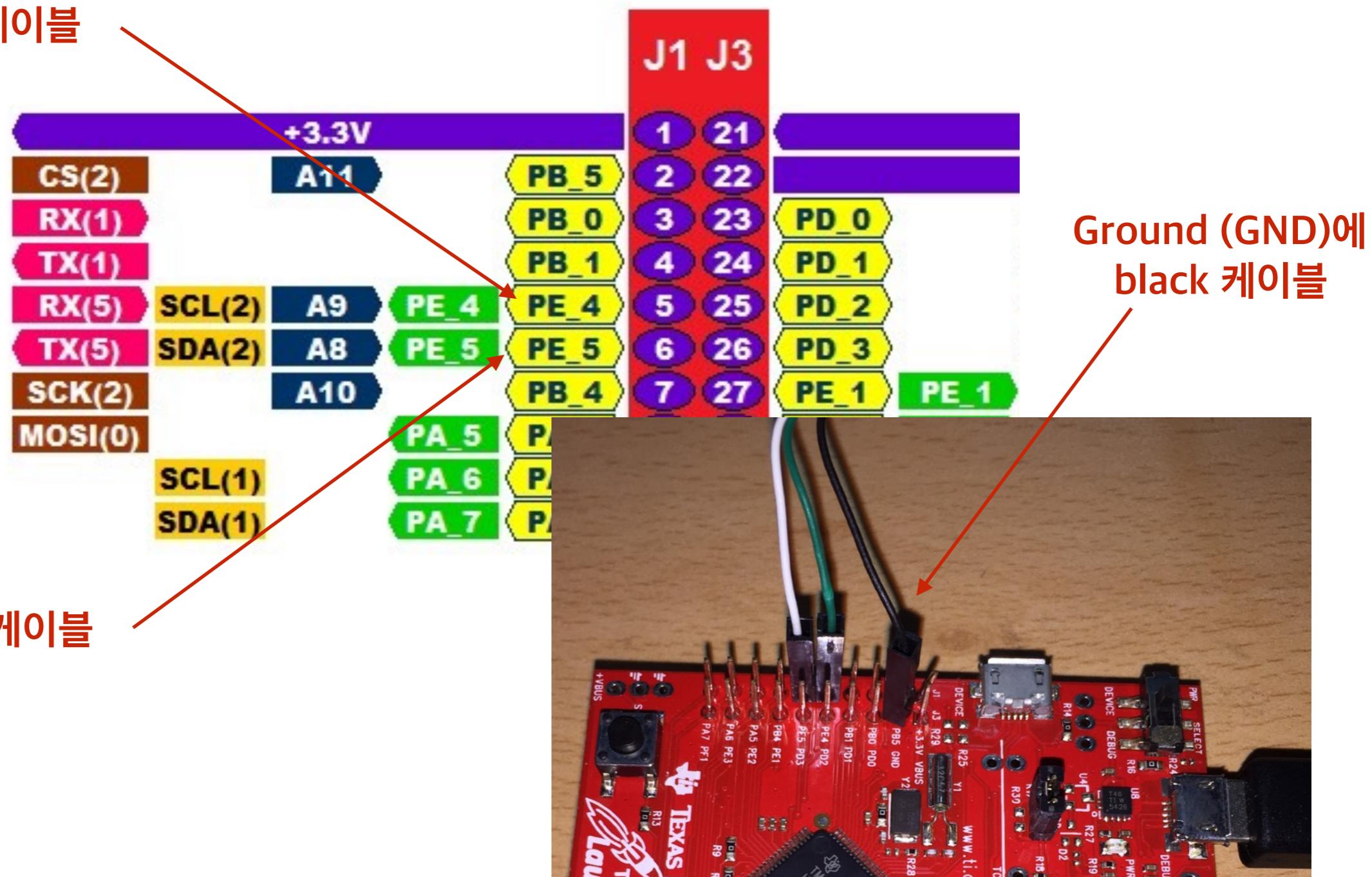
USB-To-TTL Serial 케이블이 필요

- <https://www.adafruit.com/product/954>
- 드라이버 설치 필요: **Windows or Mac**

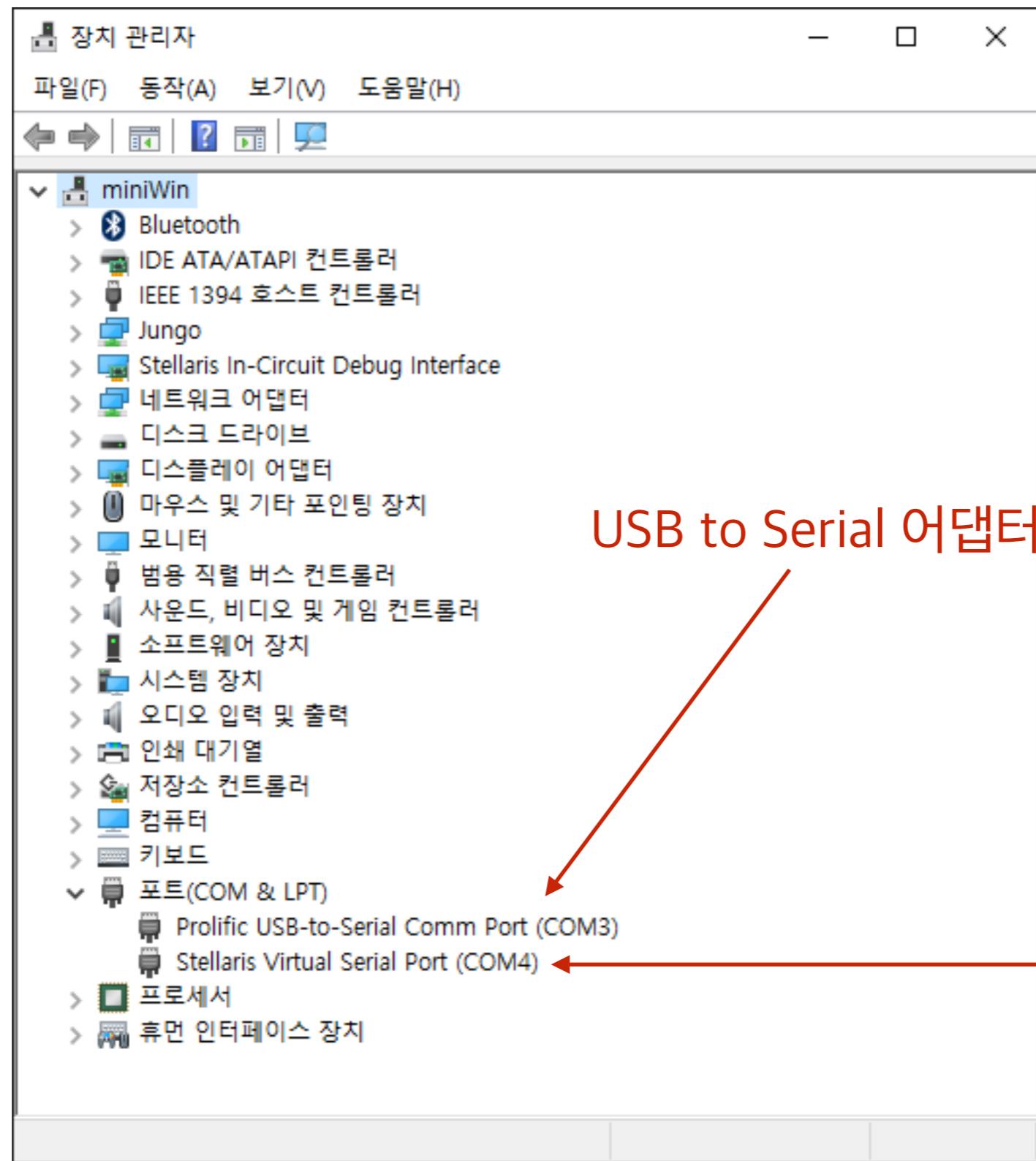
Windows의 경우 드라이버 설치
불필요할 수도 있음
(p.65 처럼 COM포트 2개가 잡힌다면 드라이버 설치 불필요)

USB to TTL Serial Cable 연결

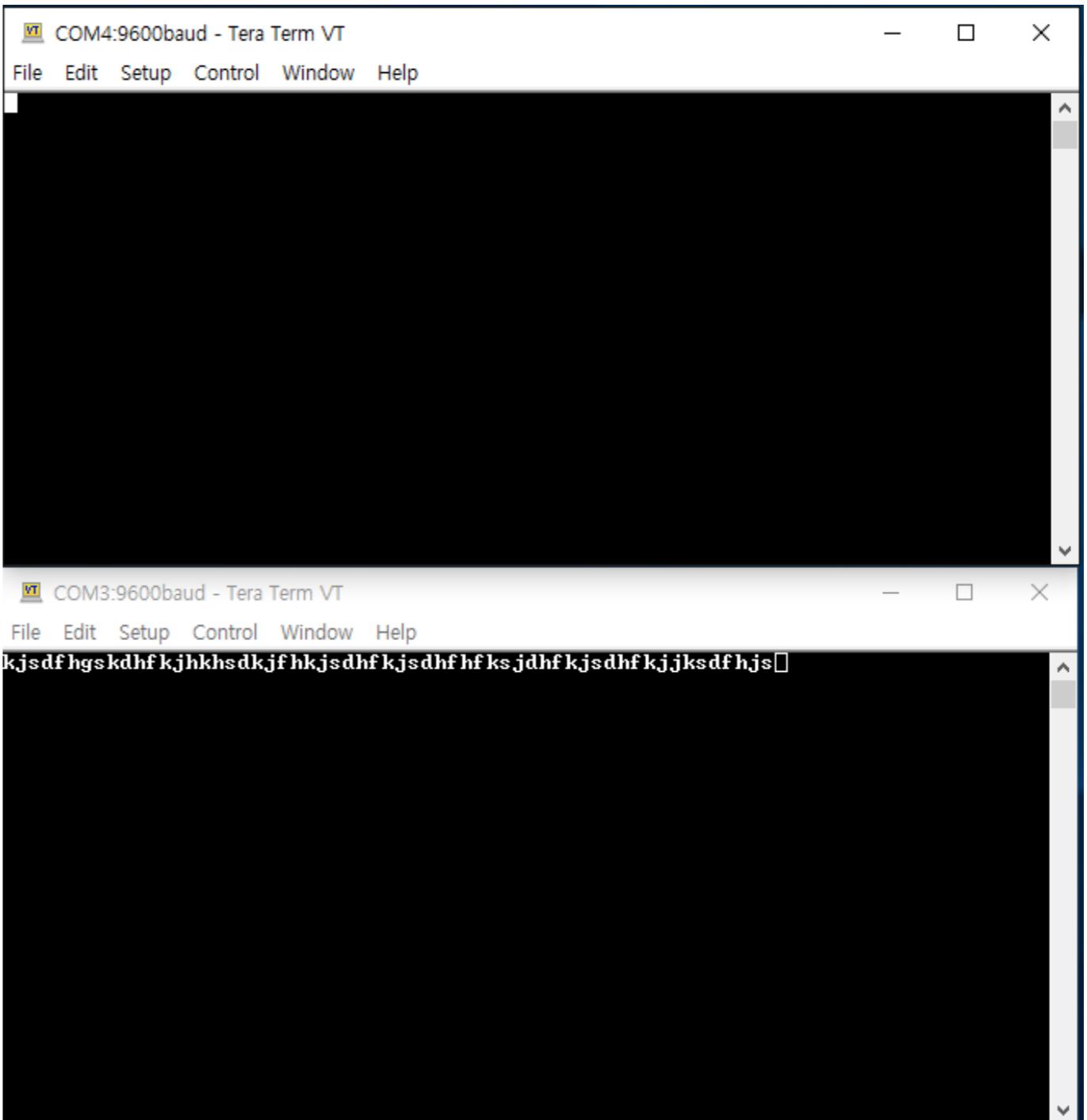
PE4에 green 케이블



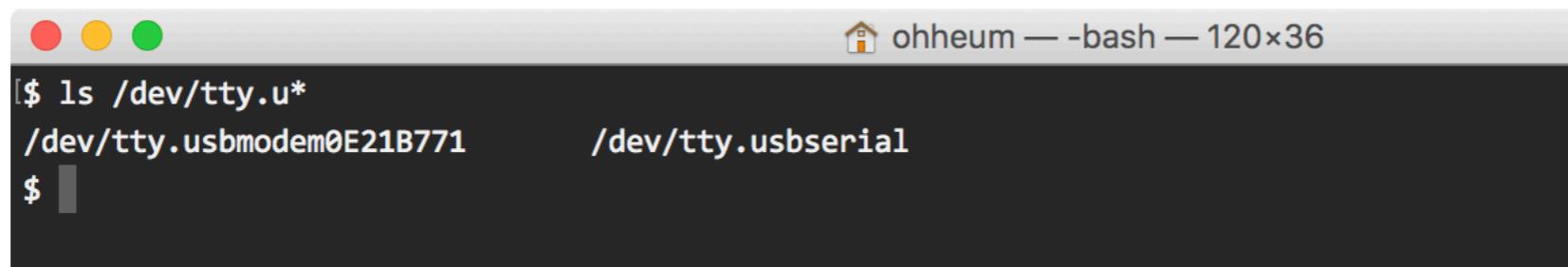
2개의 Serial Com 포트 확인



- ❶ TeraTerm을 2개 실행하여 2개의 Com 포트에 각각 접속한다.
- ❷ UART0쪽의 터미널에 입력 한 내용이 그대로 UART5를 거쳐 반대쪽 터미널로 출력됨을 확인한다.

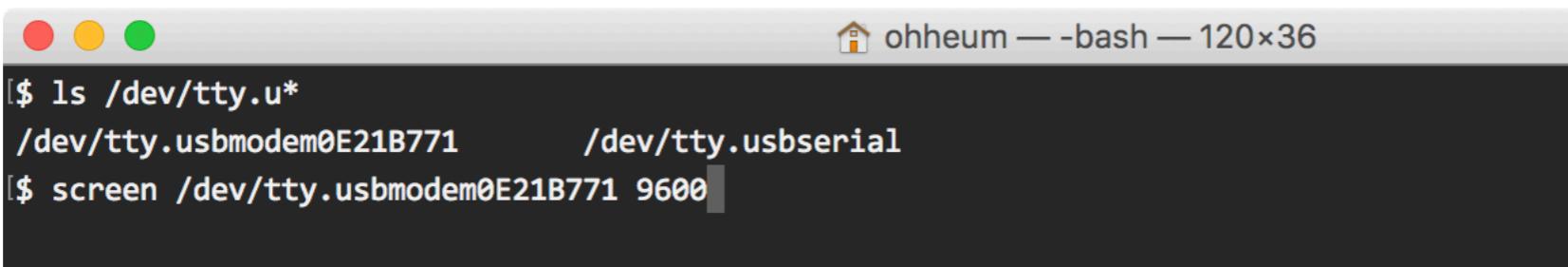


- ④ 2개의 serial tty 디바이스가 생성된다.



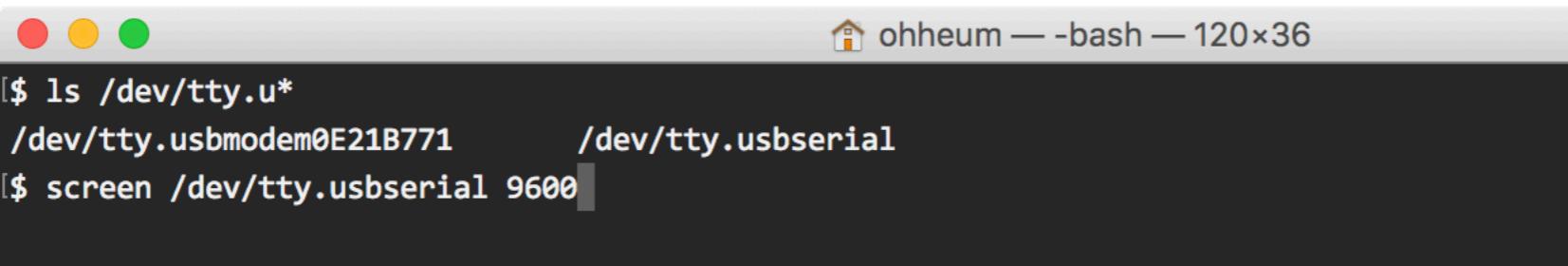
```
ohheum — -bash — 120x36
[$ ls /dev/tty.u*
/dev/tty.usbmodem0E21B771      /dev/tty.usbserial
$
```

- ④ screen 명령으로 Launchpad에 접속한다.



```
ohheum — -bash — 120x36
[$ ls /dev/tty.u*
/dev/tty.usbmodem0E21B771      /dev/tty.usbserial
[$ screen /dev/tty.usbmodem0E21B771 9600
```

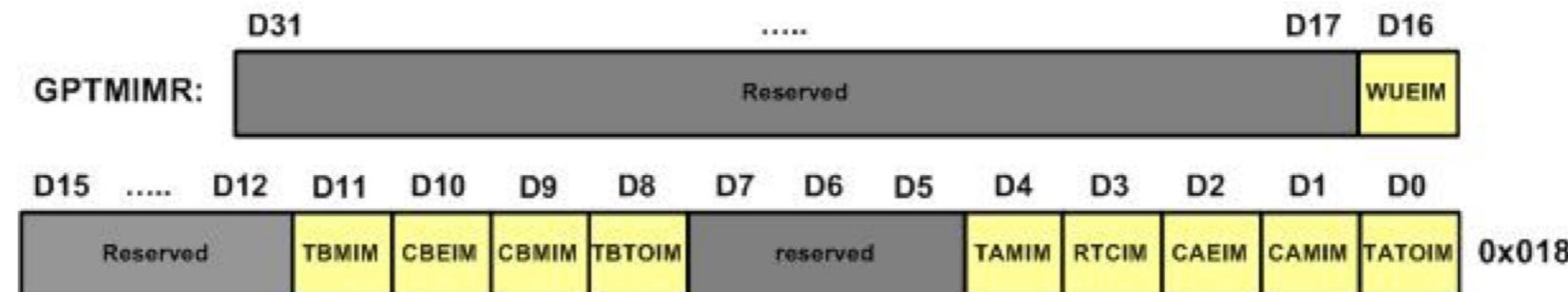
- ④ 새로운 terminal 창을 열고 USB-To_Serial 어댑터에 접속한다.



```
ohheum — -bash — 120x36
[$ ls /dev/tty.u*
/dev/tty.usbmodem0E21B771      /dev/tty.usbserial
[$ screen /dev/tty.usbserial 9600
```

Timer Interrupt Programming

GPTM Interrupt Mask



bit	Name	Description
0	TATOIM	Timer A Time-out Interrupt Mask (0: interrupt is disabled, 1: interrupt is enabled)
1	CAMIM	Timer A Capture mode Match Interrupt Mask (0: interrupt is disabled, 1: enabled)
2	CAEIM	Timer A Capture Mode Event Interrupt Mask (0: interrupt is disabled, 1: enabled)
3	RTCIM	RTC Interrupt Mask (0: interrupt is disabled, 1: interrupt is enabled)
4	TAMIM	Timer A Match Interrupt Mask (0: interrupt is disabled, 1: interrupt is enabled)
8	TBTOIM	Timer B Time-out Interrupt Mask (0: interrupt is disabled, 1: interrupt is enabled)
9	CBMIM	Timer B Capture mode Match Interrupt Mask (0: interrupt is disabled, 1: enabled)
10	CBEIM	Timer B Capture Mode Event Interrupt Mask (0: interrupt is disabled, 1: enabled)
11	TBMIM	Timer B Match Interrupt Mask (0: interrupt is disabled, 1: interrupt is enabled)
16	WUEIM	32/64-Bit Wide GPTM Write Update Error Interrupt Mask (0: disabled, 1: enabled)

To enable TIMER1A and TIMER2A,

```
NVIC->ISER[0] |= 0x00200000; /*enable IRQ21 */
NVIC->ISER[0] |= 0x00800000; /*enable IRQ23 */
```

TIMER1A toggles Red LED(PF1), TIMER2A toggles Green LED (PF3)

```
/* Timer1A is configure to timeout once every second. In the interrupt handler, the red LED is toggled. Timer2A is configure to timeout at 10 Hz. In the interrupt handler, the green LED is toggled. The infinite loop in the main program is blinking the blue LED while the interrupts are going on. */

#include "TM4C123GH6PM.h"

int main (void)
{
    /* enable clock to GPIOF at clock gating control register */
    SYSCTL->RCGCGPIO |= 0x20;
    /* enable the GPIO pins for the LED (PF3, 2 1) as output */
    GPIOF->DIR = 0x0e;
    /* enable the GPIO pins for digital function */
    GPIOF->DEN = 0x0e;

    timer1A_init(); /* setup timer1A interrupt */
    timer2A_init(); /* setup timer2A interrupt */
    __enable_irq(); /* global enable IRQs */

    while(1)
    {
        GPIOF->DATA ^= 4;           /* toggle blue LED */
        delayMs(500);              /* wait for half second */
    }
}
```

TIMER1A toggles Red LED(PF1), TIMER2A toggles Green LED (PF3)

```
/* Setup Timer1A to create 1 Hz interrupt */
void timer1A_init(void)
{
    SYSCTL->RCGCTIMER |= 2;          /* enable clock to Timer Block 1 */

    TIMER1->CTL = 0;                /* disable Timer1 before initialization */
    TIMER1->CFG = 0x04;             /* 16-bit option */
    TIMER1->TAMR = 0x02;            /* periodic mode and down-counter */
    TIMER1->TAPR = 250;             /* 16000000 Hz / 250 = 64000 Hz */
    TIMER1->TAILR = 64000;           /* 64000 Hz / 64000 = 1 Hz */
    TIMER1->ICR = 0x1;              /* clear the Timer1A timeout flag */

    TIMER1->IMR |= 0x01;            /* enable Timer1A timeout interrupt */

    TIMER1->CTL |= 0x01;            /* enable Timer1A after initialization */

    NVIC->ISER[0] |= 0x00200000; /* enable IRQ21 (D21 of ISER[0]) */

}
```

TIMER1A toggles Red LED(PF1), TIMER2A toggles Green LED (PF3)

```
/* Setup Timer2A to create 10 Hz interrupt */
void timer2A_init(void)
{
    SYSCTL->RCGCTIMER |= 4;      /* enable clock to Timer Block 2 */
    TIMER2->CTL = 0;            /* disable Timer2 before initialization */
    TIMER2->CFG = 0x04;          /* 16-bit option */
    TIMER2->TAMR = 0x02;         /* periodic mode and down-counter */
    TIMER2->TAPR = 250;          /* 16000000 Hz / 250 = 64000 Hz */
    TIMER2->TAILR = 6400;        /* 64000 Hz / 6400 = 10 Hz */
    TIMER2->ICR = 0x1;           /* clear the Timer2A timeout flag */
    TIMER2->IMR |= 0x01;          /* enable Timer2A timeout interrupt */
    TIMER2->CTL |= 0x01;          /* enable Timer2A after initialization */
    NVIC->ISER[0] |= 0x00800000; /* enable IRQ23 (D23 of ISER[0]) */
}
```

TIMER1A toggles Red LED(PF1), TIMER2A toggles Green LED (PF3)

```
void TIMER1A_Handler(void)
{
    volatile int readback;

    if (TIMER1->MIS & 0x1)          /* Timer1A timeout flag */
    {
        GPIOF->DATA ^= 2;          /* toggle red LED */
        TIMER1->ICR = 0x1;         /* clear the Timer1A timeout flag */
        readback = TIMER1->ICR;    /* to force clearing of interrupt flag */
    }
    else
    {   /* should not get here, but if we do */
        TIMER1->ICR = TIMER1->MIS; /* clear all flags */
        readback = TIMER1->ICR;    /* to force clearing of interrupt flag */
    }
}
```

TIMER1A toggles Red LED(PF1), TIMER2A toggles Green LED (PF3)

```
void TIMER2A_Handler(void)
{
    volatile int readback;
    if (TIMER2->MIS & 0x1)          /* Timer2A timeout flag */
    {
        GPIOF->DATA ^= 8;          /* toggle green LED */
        TIMER2->ICR = 0x1;         /* clear the Timer2A timeout flag */
        readback = TIMER2->ICR;    /* to force clearing of interrupt flag */
    }
    else
    {   /* should not get here, but if we do */
        TIMER2->ICR = TIMER2->MIS; /* clear all flags */
        readback = TIMER2->ICR;    /* to force clearing of interrupt flag */
    }
}

/* delay n milliseconds (16 MHz CPU clock) */
void delayMs(int n)
{
    int i, j;
    for(i = 0 ; i < n; i++)
        for(j = 0; j < 3180; j++)
            {} /* do nothing for 1 ms */
}
```

Interrupt Priority Programming

Interrupt Priority

- ⦿ 처음 3개의 인터럽트가 **highest priority**를 가짐
 - ⦿ Reset (-3), NMI (-2), Hard Fault (-1)
- ⦿ 다른 모든 인터럽트는 모두 동일한 우선순위 0을 가짐
- ⦿ 동일 우선순위에서는 **IRQ번호가 낮을 수록 높은 우선순위**

PRI registers and Priority Grouping



- To assign IRQn a priority p,

NVIC->IP[n] = p << 5; or

```
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority);
```

Interrupt Priority Demo

```
/* Timer1A is setup to interrupt at 1 Hz. In timer interrupt handler, the red LED  
is turned on and a delay 500 ms function is called. The LED is turned off at the  
end of the delay. */  
  
/* Timer2A is setup to interrupt at 10 Hz. In timer interrupt handler, the green  
LED is turned on and a delay 20 ms function is called. The LED is turned off at the  
end of the delay. */  
  
/* When Timer1A has higher priority, the Timer2A interrupts are blocked by Timer1A  
interrupt handler. You can see that when the red LED is on, the green LED is not  
blinking. When Timer2A has higher priority, the Timer1A interrupt handler is  
preempted by Timer2A interrupts and the green LED is blinking all the time. */
```

```
#include "TM4C123GH6PM.h"  
  
void Timer1_init(void);  
void Timer2_init(void);  
void delayMs(int n);
```

Interrupt Priority Demo

```
int main (void)
{
    /* enable clock to GPIOF at clock gating control register */
    SYSCTL->RCGCGPIO |= 0x20;
    /* enable the GPIO pins for the LED (PF3, 2 1) as output */
    GPIOF->DIR = 0x0e;
    /* enable the GPIO pins for digital function */
    GPIOF->DEN = 0x0e;

    Timer1_init();
    Timer2_init();
    __enable_irq();

    while(1)
    {
    }
}
```

Interrupt Priority Demo

```
void TIMER1A_Handler(void)
{
    volatile int readback;

    GPIOF->DATA |= 2; /* turn on red LED */
    delayMs(500);
    GPIOF->DATA &= ~2; /* turn off red LED */
    TIMER1->ICR = 0x1;
    readback = TIMER1->ICR; /* a read to force clearing of interrupt flag */
}

void TIMER2A_Handler(void)
{
    volatile int readback;

    TIMER2->ICR = 0x1;
    GPIOF->DATA |= 8; /* turn on green LED */
    delayMs(20);
    GPIOF->DATA &= ~8; /* turn off green LED */
    readback = TIMER2->ICR; /* a read to force clearing of interrupt flag */
}
```

Interrupt Priority Demo

```
void Timer1_init(void)
{
    SYSCTL->RCGCTIMER |= 2; /* enable clock to Timer Module 1 */
    TIMER1->CTL = 0;        /* disable Timer1 before initialization */
    TIMER1->CFG = 0x04;     /* 16-bit option */
    TIMER1->TAMR = 0x02;    /* periodic mode and up-counter */
    TIMER1->TAPR = 250;     /* 16000000 Hz / 250 = 64000 Hz */
    TIMER1->TAILR = 64000;   /* 64000 Hz / 64000 = 1 Hz */
    TIMER1->ICR = 0x1;      /* clear the Timer1A timeout flag */
    TIMER1->IMR |= 0x01;    /* enable Timer1A timeout interrupt */
    TIMER1->CTL |= 0x01;    /* enable Timer1A after initialization */

    NVIC->IP[21] = 4 << 5; /* set interrupt priority to 4 */
    NVIC->ISER[0] |= 0x00200000; /* enable IRQ21 (D21 of ISER[0]) */
}
```



이 값을 7로 바꾸고 다시 실행해본다.

Interrupt Priority Demo

```
void Timer2_init(void)
{
    SYSCTL->RCGCTIMER |= 4; /* enable clock to Timer Module 2 */
    TIMER2->CTL = 0;        /* disable Timer2 before initialization */
    TIMER2->CFG = 0x04;     /* 16-bit option */
    TIMER2->TAMR = 0x02;    /* periodic mode and up-counter */
    TIMER2->TAPR = 250;     /* 16000000 Hz / 250 = 64000 Hz */
    TIMER2->TAILR = 6400;   /* 64000 Hz / 6400 = 10 Hz */
    TIMER2->ICR = 0x1;      /* clear the Timer1A timeout flag */
    TIMER2->IMR |= 0x01;    /* enable Timer2A timeout interrupt */
    TIMER2->CTL |= 0x01;    /* enable Timer2A after initialization */

    NVIC->IP[23] = 5 << 5; /* set timer2A interrupt priority to 5 */

    NVIC->ISER[0] |= 0x00800000; /* enable IRQ23 (D23 of ISER[0]) */
}

/* delay n milliseconds (16 MHz CPU clock) */
void delayMs(int n)
{
    int32_t i, j;
    for(i = 0 ; i < n; i++)
        for(j = 0; j < 3180; j++)
            {} /* do nothing for 1 ms */
}
```

1. 타이머 인터럽트를 이용하여 1초 단위로 내부 **RGB LED가 000에서 111까지 2진 카운터로 작동하도록 프로그램을 작성하라.** 스위치를 누르면 **000으로 reset되는 기능을 추가하라.**
2. 노트북으로부터 시리얼로 **000~111 사이의 값을 전송받아 RGB LED를 켜는 프로그램을 작성하라.** 시리얼 수신 부분을 인터럽트로 구현하라.

SysTick Programming and Interrupt

SysTick의 구조

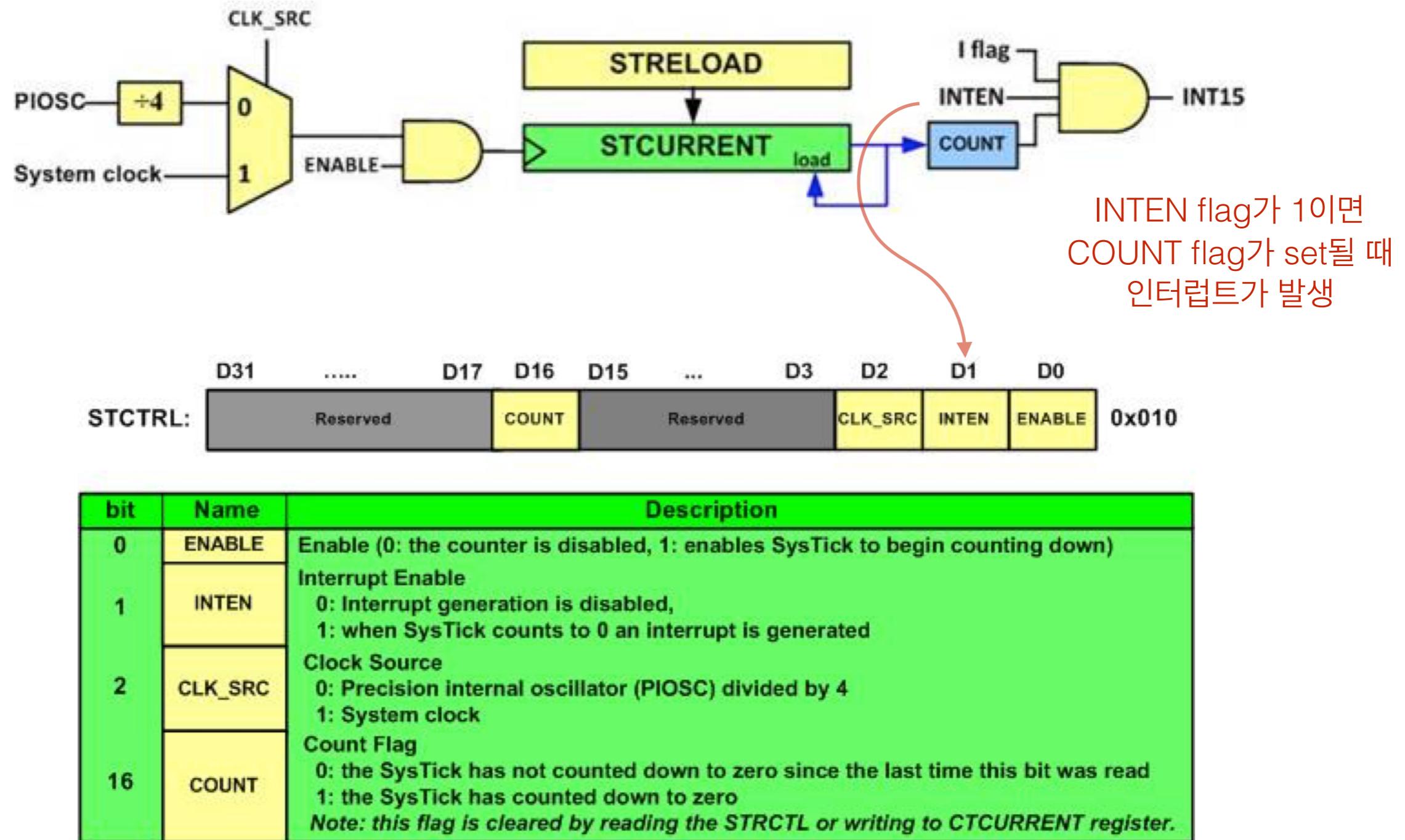
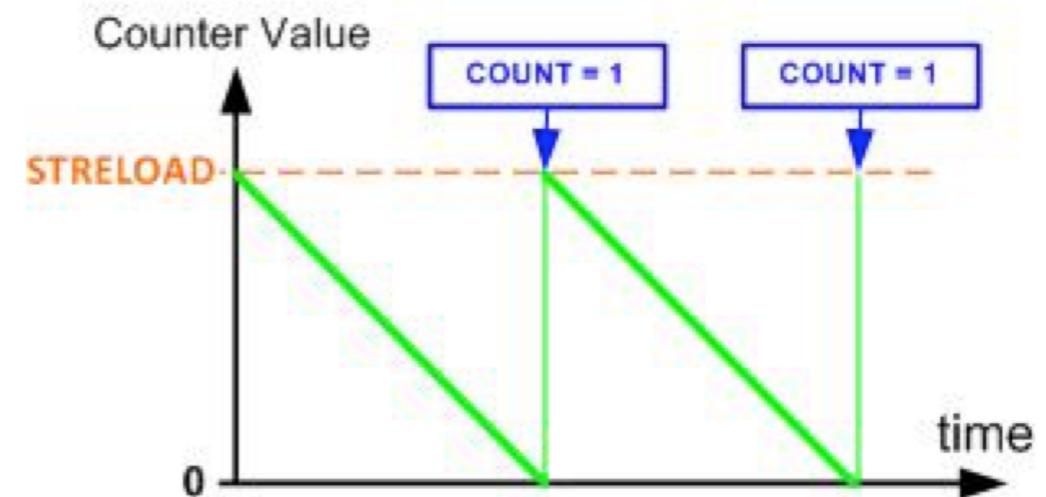


Figure 6-24: SysTick Control and Status Register (STCTRL)

Using SysTick with Interrupt

- Used to initiate an action on a periodic basis
- Interrupt #15 is assigned to the SysTick
- Toggle RED LED of PF1 every second
 - RELOAD value = 16,000,000-1
 - Since SysTick is an internal interrupt,
 - the COUNT flag is cleared automatically when the interrupt occurs.
 - we must use the CTR register to enable the interrupt



SysTick Interrupt

```
/* Toggle the red LED using the SysTick interrupt. This program sets up the SysTick to
interrupt at 1 Hz. The system clock is running at 16 MHz. 1sec/62.5ns=16,000,000 for RELOAD
register. In the interrupt handler, the red LED is toggled. */
```

```
#include "TM4C123GH6PM.h"

int main (void)
{
    /* enable clock to GPIOF at clock gating control register */
    SYSCTL->RCGCGPIO |= 0x20;
    /* enable the GPIO pins for the LED (PF3, 2, 1) as output */
    GPIOF->DIR = 0x0e;
    /* enable the GPIO pins for digital function */
    GPIOF->DEN = 0x0e;

    /* Configure SysTick */
    SysTick->LOAD = 16000000-1;      /* reload with number of clocks per second */
    SysTick->CTRL = 7;            /* enable SysTick interrupt, use system clock */

    __enable_irq();                /* global enable interrupt */

    while (1) { }

void SysTick_Handler(void)
{
    GPIOF->DATA ^= 2;        /* toggle the red LED */
}
```

An Extremely Simple RTOS

(<http://users.ece.utexas.edu/~valvano/arm/index.htm>)

```
#include "TM4C123GH6PM.h"
#include <stdint.h>
#include "os.h"

#define TIMESLICE TIME_2MS      // thread switch time in system time units

uint32_t Count1;    // number of times thread1 loops
uint32_t Count2;    // number of times thread2 loops
uint32_t Count3;    // number of times thread3 loops
```

```
void Task1(void){  
    Count1 = 0;  
    for(;;){  
        Count1++;  
        GPIOD->DATA ^= 0x02;          // toggle PD1  
    }  
}  
  
void Task2(void){  
    Count2 = 0;  
    for(;;){  
        Count2++;  
        GPIOD->DATA ^= 0x04;          // toggle PD2  
    }  
}  
  
void Task3(void){  
    Count3 = 0;  
    for(;;){  
        Count3++;  
        GPIOD->DATA ^= 0x08;          // toggle PD3  
    }  
}
```

```
int main(void){  
  
    OS_Init();          // initialize, disable interrupts, 50 MHz  
  
    SYSCTL->RCGCGPIO |= 0x08;      // activate clock for Port D  
    while((SYSCTL->RCGCGPIO & 0x08) == 0){}; // allow time to stabilize  
  
    GPIOD->DIR |= 0x0E;  
    GPIOD->AFSEL &= ~0x0E;        // disable alt funct on PD3-1  
    GPIOD->DEN |= 0x0E;          // enable digital I/O on PD3-1  
  
    GPIOD->PCTL &= 0xFFFF000F;    // configure PD3-1 as GPIO  
    GPIOD->AMSEL &= ~0x0E;        // disable analog functionality on PD3-1  
  
    OS_AddThreads(&Task1, &Task2, &Task3);  
  
    OS_Launch(TIMESLICE); // doesn't return, interrupts enabled in here  
  
    return 0;              // this never executes  
}
```

```
#ifndef __OS_H
#define __OS_H 1

#define TIME_1MS 50000
#define TIME_2MS 2*TIME_1MS

void OS_Init(void);

int OS_AddThreads(void(*task0)(void),
                  void(*task1)(void),
                  void(*task2)(void));

void OS_Launch(uint32_t theTimeSlice);

#endif
```

```
#include "TM4C123GH6PM.h"
#include <stdint.h>
#include "os.h"

#define NVIC_SYS_PRI3_R          (*((volatile uint32_t *)0xE000ED20))

// function definitions in osasm.s
void DisableInterrupts(void);      // Disable interrupts
void EnableInterrupts(void);       // Enable interrupts
int32_t StartCritical(void);
void EndCritical(int32_t primask);
void StartOS(void);
```

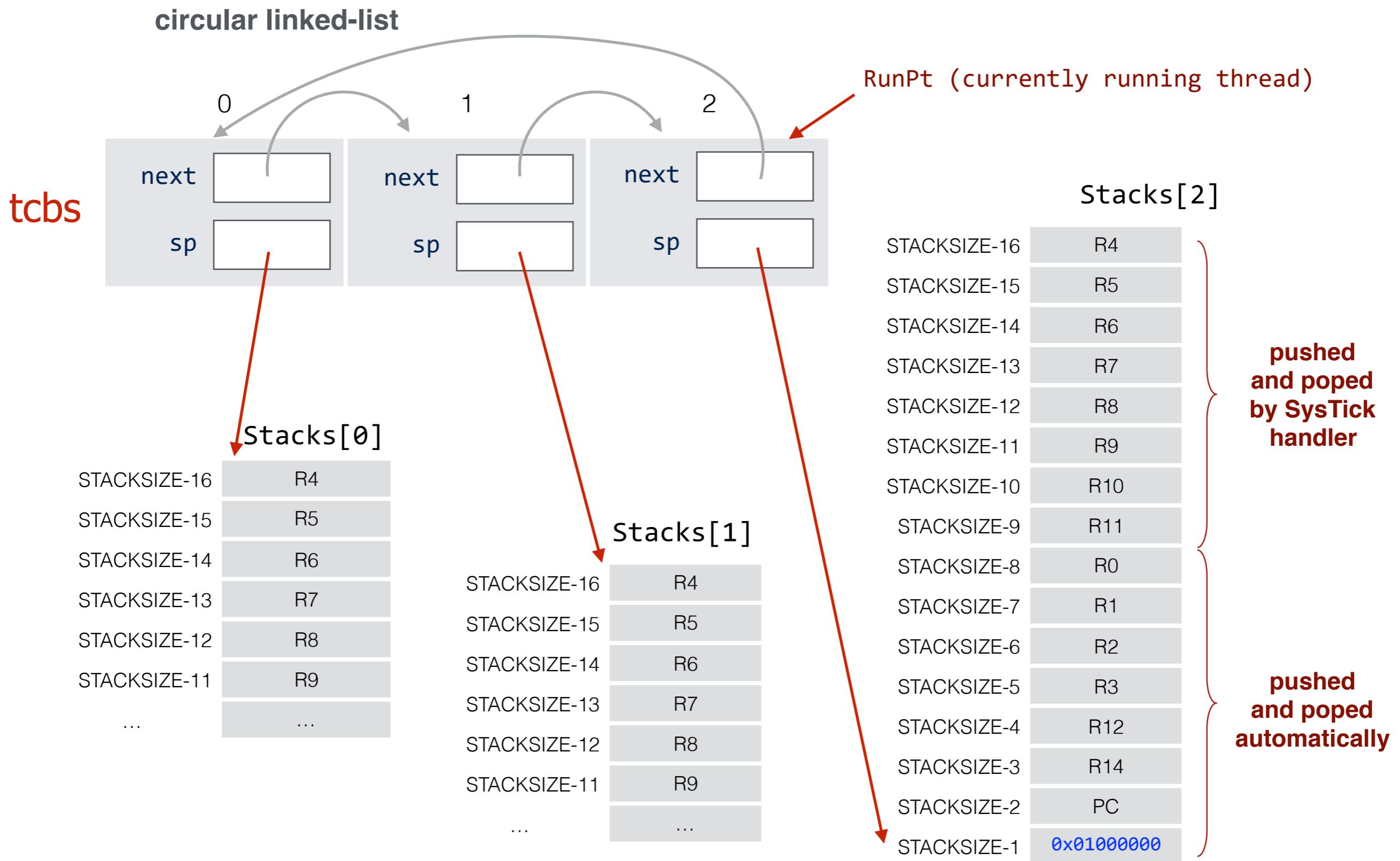
```
#define NUMTHREADS 3           // maximum number of threads
#define STACKSIZE 100          // number of 32-bit words in stack
```

```
struct tcb{
    int32_t *sp;            // pointer to stack
    struct tcb *next;       // link to the tcb of the next thread
};
typedef struct tcb tcbType;
```

```
tcbType tcbs[NUMTHREADS];
tcbType *RunPt;
```

```
int32_t Stacks[NUMTHREADS][STACKSIZE];
```

Thread Control Blocks



```
void OS_Init(void){  
    DisableInterrupts();  
  
    SysTick->CTRL = 0; // disable SysTick during setup  
    SysTick->VAL = 0;  
}
```

```
void SetInitialStack(int i){  
    tcbs[i].sp = &Stacks[i][STACKSIZE-16]; // thread stack pointer  
    Stacks[i][STACKSIZE-1] = 0x01000000; // thumb bit, PSR  
    Stacks[i][STACKSIZE-3] = 0x14141414; // R14  
    Stacks[i][STACKSIZE-4] = 0x12121212; // R12  
    Stacks[i][STACKSIZE-5] = 0x03030303; // R3  
    Stacks[i][STACKSIZE-6] = 0x02020202; // R2  
    Stacks[i][STACKSIZE-7] = 0x01010101; // R1  
    Stacks[i][STACKSIZE-8] = 0x00000000; // R0  
    Stacks[i][STACKSIZE-9] = 0x11111111; // R11  
    Stacks[i][STACKSIZE-10] = 0x10101010; // R10  
    Stacks[i][STACKSIZE-11] = 0x09090909; // R9  
    Stacks[i][STACKSIZE-12] = 0x08080808; // R8  
    Stacks[i][STACKSIZE-13] = 0x07070707; // R7  
    Stacks[i][STACKSIZE-14] = 0x06060606; // R6  
    Stacks[i][STACKSIZE-15] = 0x05050505; // R5  
    Stacks[i][STACKSIZE-16] = 0x04040404; // R4  
}
```

```
int OS_AddThreads(void(*task0)(void),
                  void(*task1)(void),
                  void(*task2)(void)){
    int32_t status;
    status = StartCritical();
    tcbs[0].next = &tcbs[1]; // 0 points to 1
    tcbs[1].next = &tcbs[2]; // 1 points to 2
    tcbs[2].next = &tcbs[0]; // 2 points to 0

    SetInitialStack(0);
    Stacks[0][STACKSIZE-2] = (int32_t)(task0); // PC
    SetInitialStack(1);
    Stacks[1][STACKSIZE-2] = (int32_t)(task1); // PC
    SetInitialStack(2);
    Stacks[2][STACKSIZE-2] = (int32_t)(task2); // PC

    RunPt = &tcbs[0];           // thread 0 will run first
    EndCritical(status);
    return 1;
}
```

```
void OS_Launch(uint32_t theTimeSlice){  
    SysTick->LOAD = theTimeSlice - 1; // reload value  
    SysTick->CTRL = 3; // enable, core clock and interrupt arm  
  
    StartOS(); // start on the first task  
}
```

osasm.s



download
this file

```
EXTERN RunPt           ; currently running thread
EXPORT DisableInterrupts
EXPORT EnableInterrupts
EXPORT StartOS
EXPORT SysTick_Handler
EXPORT StartCritical
EXPORT EndCritical

SECTION .text : CODE (2)
```

DisableInterrupts

```
CPSID I           ; __disable_irq();
BX    LR          ; returns
```

EnableInterrupts

```
CPSIE I          ; __enable_irq();
BX   LR
```

SysTick_Handler

```

CPSID    I          ; 1) Saves R0-R3,R12,LR,PC,PSR
PUSH    {R4-R11}     ; 2) Prevent interrupt during switch
LDR     R0, =RunPt   ; 3) Save remaining regs r4-11
LDR     R1, [R0]      ; 4) R0=pointer to RunPt, old thread
                      ;     R1 = *RunPt i.e., R1 = &(RunPt->sp);

STR     SP, [R1]      ; 5) Save SP into TCB; SP = RunPt->sp;

LDR     R1, [R1,#4]    ; 6) R1 = RunPt->next
STR     R1, [R0]      ;     RunPt = R1

LDR     SP, [R1]      ; 7) new thread SP; SP = RunPt->sp;

POP    {R4-R11}     ; 8) restore regs r4-11
CPSIE   I          ; 9) tasks run with interrupts enabled
BX      LR         ; 10) restore R0-R3,R12,LR,PC,PSR

```

address of RunPt

StartOS

```
LDR    R0, =RunPt      ; R0 = &RunPt;
LDR    R2, [R0]        ; R2 = *RunPt, i.e., R2 = &(RunPt->sp);
```

```
LDR    SP, [R2]        ; first thread SP; SP = RunPt->sp;
```

```
POP   {R4-R11}        ; restore regs r4-11
POP   {R0-R3}          ; restore regs r0-3
POP   {R12}
POP   {LR}             ; discard LR from initial stack
```

```
stacksize-2 POP   {LR}          ; start location
```

function pointer
of first thread
(return address로 사용됨)

```
stacksize-1 POP   {R1}          ; discard PSR
```

```
CPSIE I               ; Enable interrupts at processor level
BX    LR              ; start first thread
```

StartCritical

```
MRS    R0, PRIMASK ; save old status  
CPSID I           ; mask all (except faults)  
BX    LR
```

EndCritical

```
MSR    PRIMASK, R0  
BX    LR
```

```
END
```

MRS:

Move the contents of a special register to a general-purpose register.

MSR:

Move the contents of a general-purpose register into the specified special register.

PRIMASK:

The PRIMASK register prevents activation of all exceptions with configurable priority.