TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
47 일차 (2018. 05. 03)

목차

```
#include "HL_sys_common.h"
#include "HL_gio.h"

int main(void)
{
/* USER CODE BEGIN (3) */
    gioInit();
    gioSetDirection(gioPORTA, 0xffffffff);
    gioSetPort(gioPORTA, 0xffffffff);
    gioSetBit(gioPORTA, 4, 1);

    while(1);
/* USER CODE END */

    return 0;
}
```

※ 코드에서 함수로 들어가기 위해서는 해당 함수에서 F3 을 눌러주면 된다. 함수에서 코드로 다시 나올때는 alt ← 를 쓰면 다시 원래로 돌아간다.

```
void gioInit(void)
{
/* USER CODE BEGIN (2) */
/* USER CODE END */

    /** bring GIO module out of reset */
    gioREG->GCR0    = 1U;
    gioREG->ENACLR = 0xFFU;
    gioREG->LVLCLR = 0xFFU;
```

25.5 GIO Control Registers

Table 25-1 shows the summary of the GIO registers. The registers are accessible in 8-, 16-, and 32-bit reads or writes.

The start address for the GIO module is FFF7 BC00h.

The GIO module supports up to 8 ports. Refer to your device-specific data manual to identify the actual number of GIO ports and the number of pins in each GIO port implemented on this device.

The GIO module supports up to 4 interrupt-capable ports. Refer to the device datasheet to identify the actual number of interrupt-capable GIO ports and the number of pins in each GIO port implemented on this device.

Table 25-1. GIO Control Registers

Offset	Acronym	Register Description	Section
00h	GIOGCR0	GIO Global Control Register	Section 25.5.1
08h	GIOINTDET	GIO Interrupt Detect Register	Section 25.5.2
0Ch	GIOPOL	GIO Interrupt Polarity Register	Section 25.5.3
10h	GIOENASET	GIO Interrupt Enable Set Register	Section 25.5.4.1
14h	GIOENACLR	GIO Interrupt Enable Clear Register	Section 25.5.4.2
18h	GIOLVLSET	GIO Interrupt Priority Set Register	Section 25.5.5.1
1Ch	GIOLVLCLR	GIO Interrupt Priority Clear Register	Section 25.5.5.2
20h	GIOFLG	GIO Interrupt Flag Register	Section 25.5.6
24h	GIOOFF1	GIO Offset 1 Register	Section 25.5.7
28h	GIOOFF2	GIO Offset 2 Register	Section 25.5.8
2Ch	GIOEMU1	GIO Emulation 1 Register	Section 25.5.9
30h	GIOEMU2	GIO Emulation 2 Register	Section 25.5.10
34h	GIODIRA	GIO Data Direction Register	Section 25.5.11
38h	GIODINA	GIO Data Input Register	Section 25.5.12
3Ch	GIODOUTA	GIO Data Output Register	Section 25.5.13
40h	GIODSETA	GIO Data Set Register	Section 25.5.14
44h	GIODCLRA	GIO Data Clear Register	Section 25.5.15
48h	GIOPDRA	GIO Open Drain Register	Section 25.5.16
4Ch	GIOPULDISA	GIO Pull Disable Register	Section 25.5.17
50h	GIOPSLA	GIO Pull Select Register	Section 25.5.18
54h	GIODIRB	GIO Data Direction Register	Section 25.5.11
58h	GIODINB	GIO Data Input Register	Section 25.5.12
5Ch	GIODOUTB	GIO Data Output Register	Section 25.5.13
60h	GIODSETB	GIO Data Set Register	Section 25.5.14
64h	GIODCLRB	GIO Data Clear Register	Section 25.5.15
68h	GIOPDRB	GIO Open Drain Register	Section 25.5.16
6Ch	GIOPULDISB	GIO Pull Disable Register	Section 25.5.17
70h	GIOPSLB	GIO Pull Select Register	Section 25.5.18

⇒ 해당 코드를 분석하기 위해서는 원래는 데이터베이스에서 gio 로 찾아서 자료들을 해석하면서 control register 로 들어가야 한다. 그러나 시간이 오래 걸리므로 giogcr 로 찾아 들어간 것이다. Tabel 25-1 에서 gio 레지스터들에 대한 간략한 설명이 있다. 이 레지스터는 8, 16, 32 번 비트에 읽고 쓰는 것에 대한 접근 권한이 있다. 읽고 쓰는 것이 가능하다는 뜻이다. gio 모듈은 최대 8 개의 포트를, 4 개의 인터럽트 가능 포트를 지원한다. Gio 포트의 실제 gio 포트 수와 핀 수, 인터럽트 가능 gio 포트 등 을 확인하려면 데이터 시트를 보라는 뜻이다. 우리는 gioInit 함수에 정의 된 것들과 비교해서 보면 된다.

※ 코드를 보면 숫자 뒤에 u 가 붙는데 이는 unsigned 로 음수가 없는 상태를 쓴다는 뜻으로 사용된 것이다.

```
// gioREG->GCR0    = 1U;
```

25.5.1 GIO Global Control Register (GIOGCR0)

The GIOGCR0 register contains one bit that controls the module reset status. Writing a 0 to this bit puts the module in a reset state. After system reset, this bit must be set to 1 before configuring any other register of the GIO module. Figure 25-5 and Table 25-2 describe this register.

Figure 25-5. GIO Global Control Register (GIOGCR0) [offset = 00h]

31	16
Reserved	
R-0	

15	1	0
Reserved		RESET
R-0		R/WP-0

LEGEND: R/W = Read/Write; R = Read only; WP = Write in privileged mode only; -*n* = value after reset

Table 25-2. GIO Global Control Register (GIOGCR0) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reads return 0. Writes have no effect.
0	RESET		GIO reset.
		0	The GIO is in reset state.
		1	The GIO is operating normally.

NOTE: Note that putting the GIO module in reset state is not the same as putting it in a low-power state.

- gioREG->GCR0 = 1U; 는 GIOGCR0 에 1 을 대입하여 0 번째 비트를 1 로 활성화 시켜주는 것이다. 이 레지스터는 모듈 리셋 상태를 제어하는 비트 한 개를 포함하고 있다. 이 비트에 0 을 쓰면 모듈을 리셋 상태로 둔다. **시스템을 리셋한 후, 이 비트는 GIO 모듈의 다른 레지스터들을 구성하기 전에 반드시 1 로 셋팅** 해야한다. 1-31 번 비트는 예약 되어 있는 상태로 무엇을 쓰던 영향을 받지 않는다. 0 번째 비트에 0 이 되면 GIO 는 리셋 상태가 되고 1 이면 GIO 는 정상 작동 중이다.
  리셋 상태인 GIO 는 저전력 상태에 있는 것과는 다른 것이다.

//gioREG->ENACLR = 0xFFU;

- 이 레지스터에 0xFF 입력하고 인터럽트를 비활성화한다.
  0x000000ff 이므로 GIOENCLR0 이 1 로 셋팅 1 이면 인터럽트를 무시한다.

// gioREG->LVLCLR = 0xFFU;

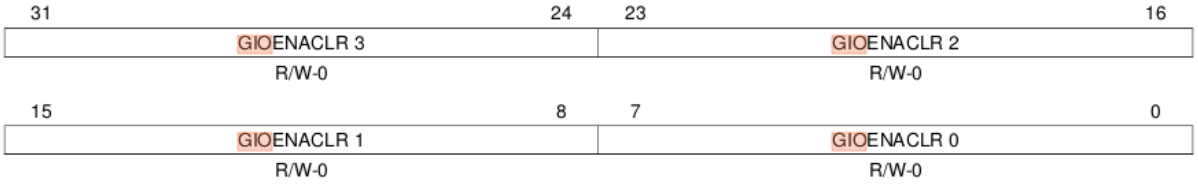
- 이 레지스터에 0xFF 입력하고 인터럽트를 최하위로 변경한다. 저수준 인터럽트로 바뀐 것이다.
- 0 을 쓰는 효과가 없다.

⇒ 이 3 줄은 코드를 만드는데 매우 중요하다. 특히 gioREG->GCR0 = 1U;는 무조건 리셋으로 초기화해주는 작업이므로 초반에 꼭 해주어야 하는 작업이다.

25.5.4.2 GIOENACLR Register

This register disables the interrupt. Figure 25-9 and Table 25-6 describe this register.

Figure 25-9. GIO Interrupt Enable Clear Register (GIOENACLR) [offset = 14h]



LEGEND: R/W = Read/Write; -*n* = value after reset

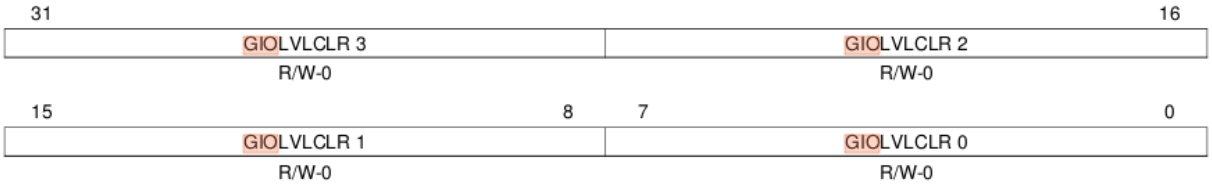
Table 25-6. GIO Interrupt Enable Clear Register (GIOENACLR) Field Descriptions

Bit	Field	Value	Description
31-24	GIOENACLR 3		Interrupt disable for pins GIOD[7:0]
		0	Read: The interrupt is disabled. Write: Writing a 0 to this bit has no effect.
		1	Read: The interrupt is enabled. Write: Disables the interrupt.
23-16	GIOENACLR 2		Interrupt disable for pins GIOC[7:0]
		0	Read: The interrupt is disabled. Write: Writing a 0 to this bit has no effect.
		1	Read: The interrupt is enabled. Write: Disables the interrupt.
15-8	GIOENACLR 1		Interrupt disable for pins GIOB[7:0]
		0	Read: The interrupt is disabled. Write: Writing a 0 to this bit has no effect.
		1	Read: The interrupt is enabled. Write: Disables the interrupt.
7-0	GIOENACLR 0		Interrupt disable for pins GIOA[7:0]
		0	Read: The interrupt is disabled. Write: Writing a 0 to this bit has no effect.
		1	Read: The interrupt is enabled. Write: Disables the interrupt.

25.5.5.2 GIOLVLCLR Register

The GIOLVLCLR register is used to configure an interrupt as a low-level interrupt going to the VIM. An interrupt can be configured as a low-level interrupt by writing a 1 into the corresponding bit of the GIOLVLCLR register. Writing a 0 has no effect. Figure 25-11 and Table 25-8 describe this register.

Figure 25-11. GIO Interrupt Priority Register (GIOLVLCLR) [offset = 1Ch]



LEGEND: R/W = Read/Write; -*n* = value after reset

/** @b initialize @b Port @b A */

- 0 을 옆의 숫자만큼 시프트 연산을 해주어, OR 연산으로 비트값을 구한다.

```
/** - Port A output values */
gioPORTA->DOUT = (uint32)((uint32)0U << 0U)  /* Bit 0 */
               | (uint32)((uint32)0U << 1U)  /* Bit 1 */
```

```
            | (uint32)((uint32)0U << 2U)  /* Bit 2 */
            | (uint32)((uint32)0U << 3U)  /* Bit 3 */
            | (uint32)((uint32)0U << 4U)  /* Bit 4 */
            | (uint32)((uint32)0U << 5U)  /* Bit 5 */
            | (uint32)((uint32)0U << 6U)  /* Bit 6 */
            | (uint32)((uint32)0U << 7U); /* Bit 7 */
```

- DOUT 를 모두 0 으로 세팅한다.

```
/** - Port A direction */

gioPORTA->DIR  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
            | (uint32)((uint32)0U << 1U)  /* Bit 1 */
            | (uint32)((uint32)0U << 2U)  /* Bit 2 */
            | (uint32)((uint32)0U << 3U)  /* Bit 3 */
            | (uint32)((uint32)1U << 4U)  /* Bit 4 */
            | (uint32)((uint32)0U << 5U)  /* Bit 5 */
            | (uint32)((uint32)0U << 6U)  /* Bit 6 */
            | (uint32)((uint32)0U << 7U); /* Bit 7 */
```

- Bit 4 를 1 로 셋팅한다.

```
/** - Port A open drain enable */
gioPORTA->PDR  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
            | (uint32)((uint32)0U << 1U)  /* Bit 1 */
            | (uint32)((uint32)0U << 2U)  /* Bit 2 */
            | (uint32)((uint32)0U << 3U)  /* Bit 3 */
            | (uint32)((uint32)0U << 4U)  /* Bit 4 */
            | (uint32)((uint32)0U << 5U)  /* Bit 5 */
            | (uint32)((uint32)0U << 6U)  /* Bit 6 */
            | (uint32)((uint32)0U << 7U); /* Bit 7 */
```

- POR 을 모두 0 으로 셋팅한다.

```
/** - Port A pullup / pulldown selection */
```

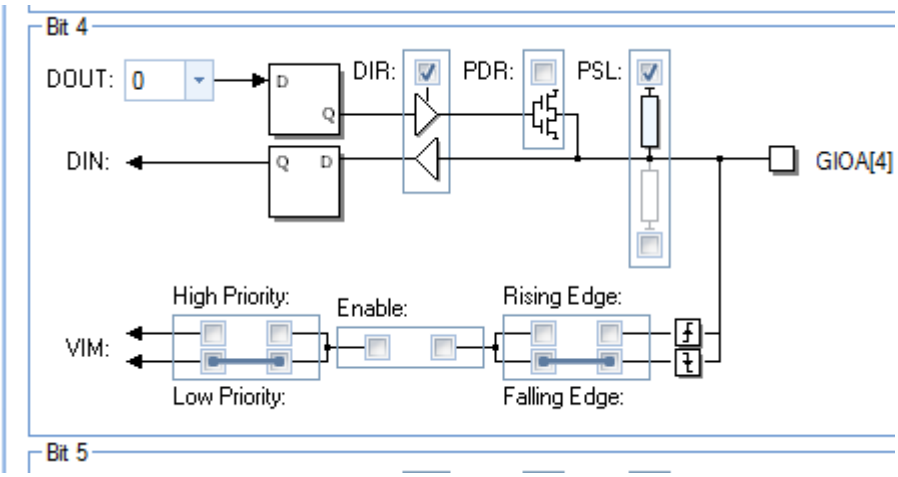
- Pullup/pulldown 은 안정화를 위한 것이다.

```
gioPORTA->PSL  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
            | (uint32)((uint32)0U << 1U)  /* Bit 1 */
            | (uint32)((uint32)0U << 2U)  /* Bit 2 */
            | (uint32)((uint32)0U << 3U)  /* Bit 3 */
            | (uint32)((uint32)1U << 4U)  /* Bit 4 */
            | (uint32)((uint32)0U << 5U)  /* Bit 5 */
            | (uint32)((uint32)0U << 6U)  /* Bit 6 */
            | (uint32)((uint32)0U << 7U); /* Bit 7 */
```

➢ 4 번 비트를 1 로 셋팅해준다.

```
/** - Port A pullup / pulldown enable*/
gioPORTA->PULDIS  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
                  | (uint32)((uint32)0U << 1U)  /* Bit 1 */
                  | (uint32)((uint32)0U << 2U)  /* Bit 2 */
                  | (uint32)((uint32)0U << 3U)  /* Bit 3 */
                  | (uint32)((uint32)0U << 4U)  /* Bit 4 */
                  | (uint32)((uint32)0U << 5U)  /* Bit 5 */
                  | (uint32)((uint32)0U << 6U)  /* Bit 6 */
                  | (uint32)((uint32)0U << 7U); /* Bit 7 */
```

➢ 모두 0 으로 셋팅한다.



회로를 보면 port B 는 사용하지 않고 있다.
그래서 지금은 이 밑은 무시해도 된다. 또한 port A 와 같은 내용이므로 설명하지 않는다.

```
/** @b initialize @b Port @b B */

/** - Port B output values */
gioPORTB->DOUT = (uint32)((uint32)0U << 0U)  /* Bit 0 */
               | (uint32)((uint32)0U << 1U)  /* Bit 1 */
               | (uint32)((uint32)0U << 2U)  /* Bit 2 */
               | (uint32)((uint32)0U << 3U)  /* Bit 3 */
               | (uint32)((uint32)0U << 4U)  /* Bit 4 */
               | (uint32)((uint32)0U << 5U)  /* Bit 5 */
               | (uint32)((uint32)0U << 6U)  /* Bit 6 */
               | (uint32)((uint32)0U << 7U); /* Bit 7 */

/** - Port B direction */
gioPORTB->DIR  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
```

```
                | (uint32)((uint32)0U << 1U)  /* Bit 1 */
                | (uint32)((uint32)0U << 2U)  /* Bit 2 */
                | (uint32)((uint32)0U << 3U)  /* Bit 3 */
                | (uint32)((uint32)0U << 4U)  /* Bit 4 */
                | (uint32)((uint32)0U << 5U)  /* Bit 5 */
                | (uint32)((uint32)0U << 6U)  /* Bit 6 */
                | (uint32)((uint32)0U << 7U); /* Bit 7 */

    /** - Port B open drain enable */
    gioPORTB->PDR  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
                | (uint32)((uint32)0U << 1U)  /* Bit 1 */
                | (uint32)((uint32)0U << 2U)  /* Bit 2 */
                | (uint32)((uint32)0U << 3U)  /* Bit 3 */
                | (uint32)((uint32)0U << 4U)  /* Bit 4 */
                | (uint32)((uint32)0U << 5U)  /* Bit 5 */
                | (uint32)((uint32)0U << 6U)  /* Bit 6 */
                | (uint32)((uint32)0U << 7U); /* Bit 7 */

    /** - Port B pullup / pulldown selection */
    gioPORTB->PSL  = (uint32)((uint32)0U << 0U)  /* Bit 0 */
                | (uint32)((uint32)0U << 1U)  /* Bit 1 */
                | (uint32)((uint32)0U << 2U)  /* Bit 2 */
                | (uint32)((uint32)0U << 3U)  /* Bit 3 */
                | (uint32)((uint32)0U << 4U)  /* Bit 4 */
                | (uint32)((uint32)0U << 5U)  /* Bit 5 */
                | (uint32)((uint32)0U << 6U)  /* Bit 6 */
                | (uint32)((uint32)0U << 7U); /* Bit 7 */

    /** - Port B pullup / pulldown enable*/
    gioPORTB->PULDIS  = (uint32)((uint32)0U << 0U) /* Bit 0 */
                    | (uint32)((uint32)0U << 1U)  /* Bit 1 */
                    | (uint32)((uint32)0U << 2U)  /* Bit 2 */
                    | (uint32)((uint32)0U << 3U)  /* Bit 3 */
                    | (uint32)((uint32)0U << 4U)  /* Bit 4 */
                    | (uint32)((uint32)0U << 5U)  /* Bit 5 */
                    | (uint32)((uint32)0U << 6U)  /* Bit 6 */
                    | (uint32)((uint32)0U << 7U); /* Bit 7 */

/* USER CODE BEGIN (3) */
/* USER CODE END */

    /** @b initialize @b interrupts */
```

```
/** - interrupt polarity */
gioREG->POL = (uint32)((uint32)0U << 0U)   /* Bit 0 */
            | (uint32)((uint32)0U << 1U)   /* Bit 1 */
            | (uint32)((uint32)0U << 2U)   /* Bit 2 */
            | (uint32)((uint32)0U << 3U)   /* Bit 3 */
            | (uint32)((uint32)0U << 4U)   /* Bit 4 */
            | (uint32)((uint32)0U << 5U)   /* Bit 5 */
            | (uint32)((uint32)0U << 6U)   /* Bit 6 */
            | (uint32)((uint32)0U << 7U)   /* Bit 7 */
            | (uint32)((uint32)0U << 8U)   /* Bit 8  */
            | (uint32)((uint32)0U << 9U)   /* Bit 9  */
            | (uint32)((uint32)0U << 10U)  /* Bit 10 */
            | (uint32)((uint32)0U << 11U)  /* Bit 11 */
            | (uint32)((uint32)0U << 12U)  /* Bit 12 */
            | (uint32)((uint32)0U << 13U)  /* Bit 13 */
            | (uint32)((uint32)0U << 14U)  /* Bit 14 */
            | (uint32)((uint32)0U << 15U); /* Bit 15 */


/** - interrupt level */
gioREG->LVLSET = (uint32)((uint32)0U << 0U)   /* Bit 0 */
               | (uint32)((uint32)0U << 1U)   /* Bit 1 */
               | (uint32)((uint32)0U << 2U)   /* Bit 2 */
               | (uint32)((uint32)0U << 3U)   /* Bit 3 */
               | (uint32)((uint32)0U << 4U)   /* Bit 4 */
               | (uint32)((uint32)0U << 5U)   /* Bit 5 */
               | (uint32)((uint32)0U << 6U)   /* Bit 6 */
               | (uint32)((uint32)0U << 7U)   /* Bit 7 */
               | (uint32)((uint32)0U << 8U)   /* Bit 8  */
               | (uint32)((uint32)0U << 9U)   /* Bit 9  */
               | (uint32)((uint32)0U << 10U)  /* Bit 10 */
               | (uint32)((uint32)0U << 11U)  /* Bit 11 */
               | (uint32)((uint32)0U << 12U)  /* Bit 12 */
               | (uint32)((uint32)0U << 13U)  /* Bit 13 */
               | (uint32)((uint32)0U << 14U)  /* Bit 14 */
               | (uint32)((uint32)0U << 15U); /* Bit 15 */


/** - clear all pending interrupts */
```

```
    gioREG->FLG = 0xFFU;

    /** - enable interrupts */
    gioREG->ENASET = (uint32)((uint32)0U << 0U)   /* Bit 0 */
                   | (uint32)((uint32)0U << 1U)   /* Bit 1 */
                   | (uint32)((uint32)0U << 2U)   /* Bit 2 */
                   | (uint32)((uint32)0U << 3U)   /* Bit 3 */
                   | (uint32)((uint32)0U << 4U)   /* Bit 4 */
                   | (uint32)((uint32)0U << 5U)   /* Bit 5 */
                   | (uint32)((uint32)0U << 6U)   /* Bit 6 */
                   | (uint32)((uint32)0U << 7U)   /* Bit 7 */
                   | (uint32)((uint32)0U << 8U)   /* Bit 8  */
                   | (uint32)((uint32)0U << 9U)   /* Bit 9  */
                   | (uint32)((uint32)0U << 10U)  /* Bit 10 */
                   | (uint32)((uint32)0U << 11U)  /* Bit 11 */
                   | (uint32)((uint32)0U << 12U)  /* Bit 12 */
                   | (uint32)((uint32)0U << 13U)  /* Bit 13 */
                   | (uint32)((uint32)0U << 14U)  /* Bit 14 */
                   | (uint32)((uint32)0U << 15U); /* Bit 15 */
}

void gioSetDirection(gioPORT_t *port, uint32 dir)
{
    port->DIR = dir;
}

void gioSetPort(gioPORT_t *port, uint32 value)
{
    port->DOUT = value;
}

void gioSetBit(gioPORT_t *port, uint32 bit, uint32 value)
{
    if (value != 0U)
    {
        port->DSET = (uint32)1U << bit;
    }
    else
    {
        port->DCLR = (uint32)1U << bit;
    }
}
```