Xilinx Zynq FPGA, TI DSP, MCU 기반의 회로 설계 및 임베디드 전문가 과정

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - TaeYoung Eun(은태영)

zero_bird@naver.com

MCU (Cortex R5F - TMS570)

2 GIO (General-Purpose Input/Output)

2.1 개요

GIO 는 범용 입 / 출력 모듈로서, I / O 기능과 외부 인터럽트 기능을 지원한다.

최대 8 개의 8 bit 포트를 지원하여 64 개의 GIO 터미널을 독립적으로 구성할 수 있다.

GIO 모듈의 주요 기능은 다음과 같다.

- 각 GIO 터미널을 입력 / 출력으로 기능을 구성할 수 있다.
- 각 GIO 터미널에 대하여 프로그래밍을 통한 제어가 가능하다.
- GIO 출력에 있어서 Push-Pull 또는 Open-Drain 을 지원한다.
- GIO 터미널의 인터럽트 생성을 최대 32 개 지원한다.

2.2 모드

2.2.1 Emulation 모드

에뮬레이션 모드는 GIOEMU1 및 GIOEMU2 레지스터를 통해 사용된다. 해당 레지스터는 GIO Offset 레지스터와 동일한 기능을 한다.

에뮬레이션 모드의 기능은 다움과 같다.

- VIM 인터럽트를 사용할 수 없기 때문에 외부 인터럽트가 캡처되지 않는다.
- 레지스터의 읽기가 시스템 상태에 영향을 주지 않는다.
- 레지스터의 쓰기는 시스템 상태에 영향을 준다.

2.2.2 Power-Down 모드 (Low-Power 모드)

저전력 모드에서는 GIO 모듈에 대한 클럭 신호가 비활성화된다. 따라서, 스위칭이 없으며 요구하는 전력만을 발생시킨다.

저전력 모드에서는 인터럽트 핀이 edge 에서 level 로 변경된다. 극성 비트의 설정은 edge 에서 level 트리거로 기능이 변한다.

또한 인터럽트 핀을 이용하여 level 에 따라서 저전력 모드와 해제도 가능하게 해 준다.

2.2.2.1 모듈 level 의 Power-Down

GIO 모듈은 주번 Power Down 레지스터의 비트를 통하여 GIO 주변 모듈을 비활성화 하여 Power Down 상태로 전환할 수 있다.

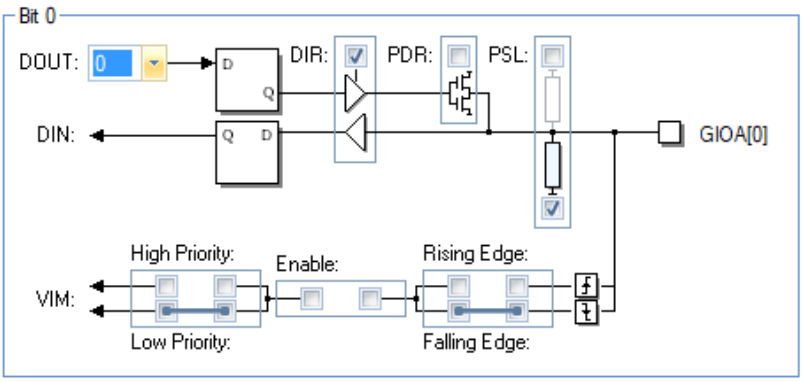
2.2.2.2 디바이스 level 의 Power-Down

디바이스의 Power Down 은 사전 정의된 Power-Down 모드로 설정할 수 있다. (doze, snooze, sleep)

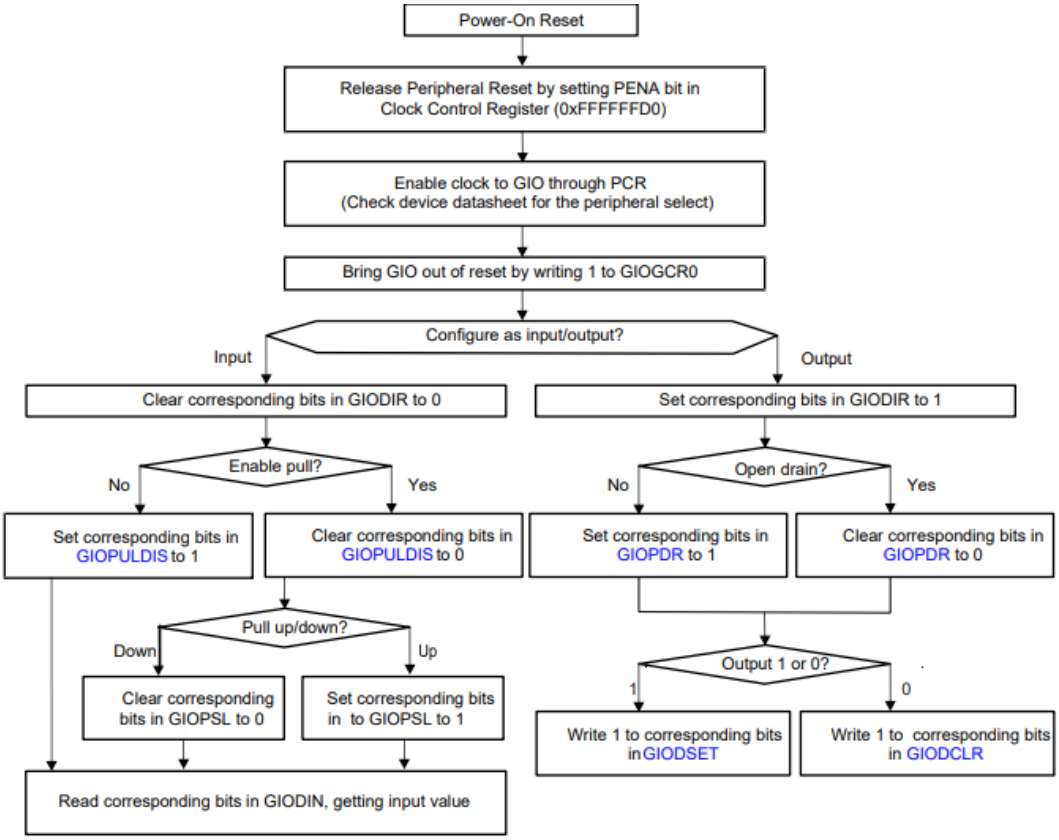
Clock Source 와 Clock Domain 의 레지스터를 통하여 시스템 모듈을 제어할 수 있다.

2.3 GIO 내부 흐름도

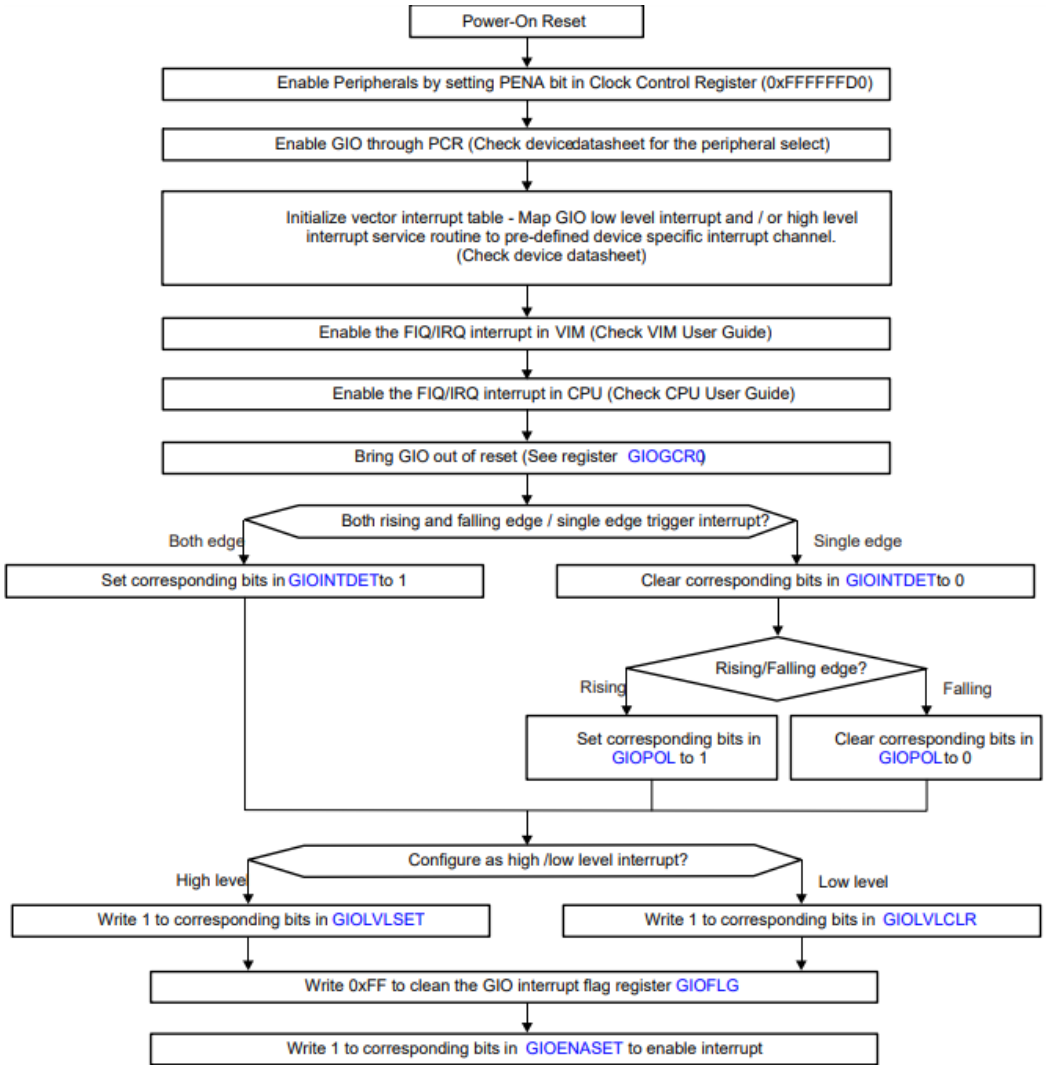
〈HAL Code Generator GIO Port〉



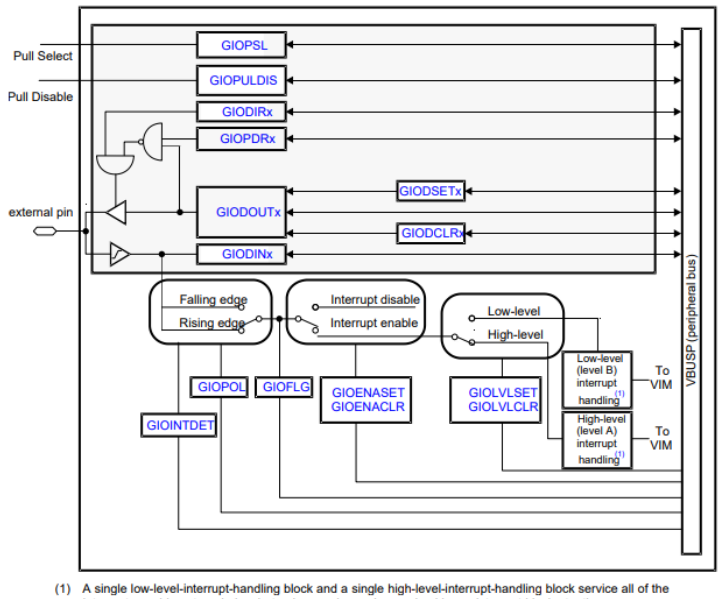
〈I/O 기능 흐름도〉



<인터럽트 생성 함수 흐름도>

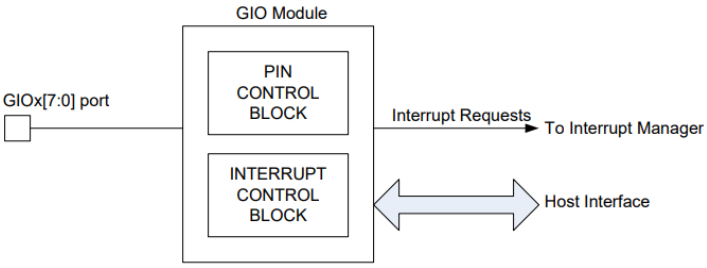


<GIO 블록 도표>



(1) A single low-level-interrupt-handling block and a single high-level-interrupt-handling block service all of the interrupt-capable external pins, but only one pin can be serviced by an interrupt block at a time.

<GIO 모듈 도표>



2.4 I/O 기능

I/O 블록을 통해 각 GIO 터미널을 다음과 같은 범용 입출력으로 사용할 수 있습니다.

2.4.1 데이터 방향 (GIODIR)

GIODIRx 레지스터를 통하여 GIO 터미널의 입력 / 출력 방향을 설정한다.

2.4.2 데이터 입력 (GIODIN)

GIODINx 레지스터를 통해 GIO 터미널의 로직 level 을 반영한다. 핀에 전압이 VIHO 이상 들어올 경우 GIODIN 에 1 이, VIL 이하의 경우 0 이 발생한다.

2.4.3 데이터 출력 (GIODOUT)

출력으로 구성된 GIO 터미널에서 로직 level 을 기록한다. Low 일 경우 GIODOUTx 레지스터가 0 으로, High 일 경우 1 상태로 강제 설정한다. 해당 기능의 사용 시 Open Drain 기능을 사용하면 안된다. 사용할 시, High 일 때 High-impedance 상태로 만든다.

2.4.4 데이터 설정 (GIODSET)

GIODSETx 레지스터를 통하여 GIO 터미널의 출력을 제어할 수 있다. 0 일 경우 Low 상태로, 1 일 경우 High 상태로 만들어 주지만, Open Drain 기능을 사용할 경우 High-impedance 상태가 된다.

2.4.5 데이터 삭제 (GIODCLR)

GIODCLRx 레지스터에 1 을 Read 하면 출력으로 구성된 GIO 터미널에서 Low 를 출력한다.

2.4.6 Open Drain (GIOPDR)

GIOPDR 레지스터를 통하여 Open Drain 활성화 / 비활성화를 제어할 수 있다. 활성화 시 High 가 High-impedance 상태가 된다.

2.4.7 Pull 비활성화 (GIOPULDIS)

GIOPULDIS 레지스터의 Read 에 따라 GIO 입력을 비활성화 한다.

2.4.8 Pull 선택 (GIOPSL)

GIOPULSELx 레지스터를 통하여 Pull Down(default)과 Pull Up 을 선택할 수 있다.

2.5 Control Registers

2.5.1 GIOGCR0 (글로벌 제어)

GIOGCR0 레지스터는 모듈 리셋 상태를 제어하는 1 비트를 포함한다. 이 비트가 0 일 경우 모듈은 리셋 상태이다.

Bit	필드	값	내용
31 - 1	Reserved	0	
0	RESET	0 1	GIO 재설정을 한다. GIO 리셋 상태이다. GIO 정상 작동 상태이다.

2.5.2 GIOINTDET (인터럽트 검출)

GIO 핀에서 상승 edge 와 하강 edge 발생 시 CPU 에 인터럽트 생성을 요청한다.

Bit	필드	값	내용
31 - 24	GIOINTDET 3		인터럽트를 검출한다. GIOD[7:0]. 동일.
23 - 16	GIOINTDET 2		인터럽트를 검출한다. GIOC[7:0]. 동일.
15 - 8	GIOINTDET 1		인터럽트를 검출한다. GIOB[7:0]. 동일.
7 - 0	GIOINTDET 0	0 1	인터럽트를 검출한다. GIOA[7:0]. GIOPOL 에서 상승 혹은 하강에 따라 인터럽트가 발생하도록 설정한다. edge 발생시 인터럽트가 발생한다.

2.5.3 GIOPOL (인터럽트 극성)

GIO 핀에서 상승 edge 또는 하강 edge 를 설정한다.

Bit	필드	값	내용
31 - 24	GIOPOL 3		인터럽트 극성을 선택한다. GIOD[7:0]. 동일.
23 - 16	GIOPOL 2		인터럽트 극성을 선택한다. GIOC[7:0]. 동일.
15 - 8	GIOPOL 1		인터럽트 극성을 선택한다. GIOB[7:0]. 동일.
7 - 0	GIOPOL 0	0 1 0 1	인터럽트 극성을 선택한다. GIOA[7:0]. 하강 edge 로 설정한다. 상승 edge 로 설정한다. 저전력 모드: 인터럽트는 Low-level 에서 트리거가 된다. 인터럽트는 High-level 에서 트리거가 된다.

2.5.4 GIOENASET (인터럽트 활성화)

GIOENASET 및 GIOENACLR 레지스터는 인터럽트 핀을 제어하고, 활성화 되었을 경우 edge 를 통하여 인터럽트 요청을 실행한다.

Bit	필드	값	내용
31 - 24	GIOENASET 3		인터럽트를 활성화한다. GIOD[7:0]. 동일.
23 - 16	GIOENASET 2		인터럽트를 활성화한다. GIOC[7:0]. 동일.
15 - 8	GIOENASET 1		인터럽트를 활성화한다. GIOB[7:0]. 동일.
7 - 0	GIOENASET 0	0 0 1 1	인터럽트를 활성화한다. GIOA[7:0]. Read: 인터럽트를 비활성화 상태를 읽어 온다. Write: 효과가 없다. Read: 인터럽트의 활성화 상태를 읽어 온다. Write: 인터럽트를 활성화한다.

2.5.5 GIOENACLR (인터럽트 비활성화)

Bit	필드	값	내용
31 - 24	GIOENACLR 3		인터럽트를 비활성화한다. GIOD[7:0]. 동일.
23 - 16	GIOENACLR 2		인터럽트를 비활성화한다. GIOC[7:0]. 동일.
15 - 8	GIOENACLR 1		인터럽트를 비활성화한다. GIOB[7:0]. 동일.
7 - 0	GIOENACLR 0		인터럽트를 비활성화한다. GIOA[7:0].
		0	Read: 인터럽트를 비활성화 상태를 읽어 온다.
		0	Write: 효과가 없다.
		1	Read: 인터럽트의 활성화 상태를 읽어 온다.
		1	Write: 인터럽트를 비활성화한다.

2.5.6 GIOLVLSET (High-level 인터럽트 설정)

GIOLVLSET 및 GIOLVLCLR 레지스터는 VIM 의 연결 채널을 설정할 수 있다.

- High-level 인터럽트는 GIOOFF1 과 GIOEMU1 에 기록된다.

- Low-level 인터럽트는 GIOOFF2 와 GIOEMU1 에 기록된다.

GIO 모듈은 두개의 인터럽트 요청을 할 수 있다. 이들은 VIM(Vectored Interrupt Manager)에서 별도의 채널로 연결되며, 이때 번호가 낮은 VIM 채널이 우선순위가 높다. 번호가 낮은 GIO 인터럽트는 High-level 이며, 높은 것은 Low-level 이다.

Bit	필드	값	내용
31 - 24	GIOLVLSET 3		High-level 인터럽트를 설정한다. GIOD[7:0]. 동일.
23 - 16	GIOLVLSET 2		High-level 인터럽트를 설정한다. GIOC[7:0]. 동일.
15 - 8	GIOLVLSET 1		High-level 인터럽트를 설정한다. GIOB[7:0]. 동일.
7 - 0	GIOLVLSET 0		High-level 인터럽트를 설정한다. GIOA[7:0].
		0	Read: Low-level 인터럽트 상태를 읽어 온다.
		0	Write: 효과가 없다.
		1	Read: High-level 인터럽트 상태를 읽어 온다.
		1	Write: High-level 인터럽트로 설정한다.

2.5.7 GIOLVLCLR (Low-level 인터럽트 설정)

Bit	필드	값	내용
31 - 24	GIOLVLCLR 3		Low-level 인터럽트를 설정한다. GIOD[7:0]. 동일.
23 - 16	GIOLVLCLR 2		Low-level 인터럽트를 설정한다. GIOC[7:0]. 동일.
15 - 8	GIOLVLCLR 1		Low-level 인터럽트를 설정한다. GIOB[7:0]. 동일.
7 - 0	GIOLVLCLR 0		Low-level 인터럽트를 설정한다. GIOA[7:0].
		0	Read: High-level 인터럽트 상태를 읽어 온다.
		0	Write: 효과가 없다.
		1	Read: Low-level 인터럽트 상태를 읽어 온다.
		1	Write: Low-level 인터럽트로 설정한다.

2.5.8 GIOFLG (인터럽트 플래그)

GIOFLG 레지스터는 edge 전이 발생 여부를 나타낸다. 적절한 Offset 레지스터를 읽거나 1 을 쓰면 플래그를 지울 수 있다.

해당 인터럽트 플래그는 인터럽트 생성 여부와는 관계없이 전이가 발생할 경우 설정된다.

인터럽트 활성화를 하기 전에 플래그를 지우는 것이 좋다.

Bit	필드	값	내용
31 - 24	GIOFLG 3		GIO 플래그이다. GIOD[7:0]. 동일.
23 - 16	GIOFLG 2		GIO 플래그이다. GIOC[7:0]. 동일.
15 - 8	GIOFLG 1		GIO 플래그이다. GIOB[7:0]. 동일.
7 - 0	GIOFLG 0		GIO 플래그이다. GIOA[7:0].
		0	Read: 마지막 지우기 후 전환이 발생하지 않았다.
		0	Write: 효과가 없다.
		1	Read: 전이가 발생하였다. 적절한 Offset 레지스터가 읽을 경우 지워진다.
		1	Write: 해당 비트가 0 으로 지워진다

2.5.9 GIOOFF1 / 2 (Off Set 설정)

GIOOFF1 레지스터는 보류중인 외부 인터럽트를 높은 우선 순위로 나타내는 숫자 Offset 값을 제공한다.(GIOOFF2 : 낮은 우선 순위)

Offset 값을 통하여 벡터 테이블의 인터럽트 루틴의 위치를 찾을 수 있다. 이 레지스터를 읽으면 해당 GIOEMU 와 GIOFLG 비트가 지워진다. 단, 에뮬레이션 모드에서는 해당 비트가 지워지지 않는다. 둘 이상의 GIO 인터럽트를 보유중일 경우, GIOOFF1(OFF2) 내용이 대기중인 다음 상위 인터럽트 값으로 변경된다. 애플리케이션은 GIOOFF1(OFF2)레지스터를 계속 읽어 0 을 읽을 때까지 동일한 서비스 루틴에서 모든 GIO 인터럽트를 처리하도록 할 수 있다.

Bit	필드	값	내용
31 - 6	Reserved	0	
5 - 0	GIOOFF 1 (GIOOFF 2)		해당 비트는 현재 보류중인 우선순위가 높은(낮은) 인터럽트를 인덱싱한다.
		0	보류중인 인터럽트가 없다.
		1h	인터럽트 0(GIOA0)번이 높은(낮은) 우선순위로 보류되어 있다.
		...	...
		8h	인터럽트 7(GIOA7)번이 높은(낮은) 우선순위로 보류되어 있다.
		9h	인터럽트 8(GIOB0)번이 높은(낮은) 우선순위로 보류되어 있다.
		..	...
		10h	인터럽트 16(GIOB7)번이 높은(낮은) 우선순위로 보류되어 있다.
		..	...
		20h	인터럽트 32(GIOD7)번이 높은(낮은) 우선순위로 보류되어 있다.
		21h~3fh	Reserved

2.5.10 GIOEMU1 / 2 (에뮬레이션 설정)

GIOEMU1(EMU2)레지스터는 읽기 전용 레지스터입니다.

이 레지스터의 내용은 GIOOFF1(OFF2)와 내용이 동일하지만, GIOEMU1(EMU2)레지스터를 읽을 때 GIOFLG 가 지워지지 않는다.

Bit	필드	값	내용
31 - 6	Reserved	0	
5 - 0	GIOEMU 1 (GIOEMU 2)		해당 비트는 현재 보류중인 우선순위가 높은(낮은) 인터럽트를 인덱싱한다.
		0	보류중인 인터럽트가 없다.
		1h	인터럽트 0(GIOA0)번이 높은(낮은) 우선순위로 보류되어 있다.
		...	...
		8h	인터럽트 7(GIOA7)번이 높은(낮은) 우선순위로 보류되어 있다.
		9h	인터럽트 8(GIOB0)번이 높은(낮은) 우선순위로 보류되어 있다.
		..	...
		10h	인터럽트 16(GIOB7)번이 높은(낮은) 우선순위로 보류되어 있다.
		..	...
		20h	인터럽트 32(GIOD7)번이 높은(낮은) 우선순위로 보류되어 있다.
		21h~3fh	Reserved

2.5.11 GIODIRA / B (입 / 출력 방향 설정)

GIODIR 레지스터는 포트 핀의 입 / 출력을 제어한다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIODIR[n]	0 1	GIO 포트의 [n]번 핀의 데이터 방향을 제어한다. [7:0] GIO 핀을 입력으로 설정한다. (입력으로 설정 시, 출력 버퍼가 변환된다.) GIO 핀을 출력으로 설정한다.

2.5.12 GIODINA / B (데이터 입력)

GIODIN 레지스터는 핀의 현재 상태(High : 1 / Low : 0)을 값으로 나타낸다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIODIN[n]	0 1	GIO 포트의 [n]번 핀의 상태를 나타낸다. [7:0] 핀의 상태가 Low 이다. 핀의 상태가 High 이다.

2.5.13 GIODOUTA / B (데이터 출력)

GIODOUT 레지스터는 핀의 출력 상태(High : 1 / Low : 0)를 지정한다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIODOUT[n]	0 1	GIO 포트의 [n]번 핀의 출력을 제어한다. [7:0] 핀의 출력을 Low 로 설정한다. 핀의 출력을 High 로 설정한다.

2.5.14 GIODSETA / B (데이터 High 설정)

이 레지스터의 값은 GIODOUT 비트의 현재 값에 관계없이 데이터 출력 제어 비트를 1 로 설정한다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIODSET[n]	0 1	GIO 포트의 [n]번 핀의 출력을 High 로 제어한다. [7:0] write: 효과가 없다. write: 핀의 출력을 High 로 설정한다. (GIODOUT 비트의 상태도 이 비트로 표시한다.)

2.5.15 GIODCLRA / B (데이터 삭제)

이 레지스터의 값은 GIODOUT 비트의 현재 값에 관계없이 데이터 출력 제어 비트를 0 으로 설정한다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIODCLR[n]	0 1	GIO 포트의 [n]번 핀의 출력을 Low 로 제어한다. [7:0] write: 효과가 없다. write: 핀의 출력을 Low 로 설정한다. (GIODOUT 비트의 상태도 이 비트로 표시한다.)

2.5.16 GIOPDRA / B (Open Drain 설정)

GIOPDR 레지스터는 데이터 핀의 Open Drain 기능을 설정합니다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIOPDR[n]	0 1	GIO 포트의 [n]번 핀의 Open Drain 을 제어한다. [7:0] GIO 핀을 Push-Pull(일반 GIO)모드로 구성된다. GIO 핀을 Open Drain 모드로 구성한다. - GIODOUT = 1 일 경우, 출력 버퍼가 변환된다.

2.5.17 GIOPULDISA / B (입력 설정)

GIOPULDIS 레지스터는 핀의 풀 컨트롤을 제어한다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIOPULDIS[n]	0 1	GIO 포트의 [n]번 핀의 풀 컨트롤을 제어한다. 입력 핀일 경우에만 적용된다.[7:0] 풀 기능을 활성화한다. 풀 기능을 비활성화 한다. GIO 핀은 GIODIR 비트를 0 으로 지우면 입력모드로 설정된다.

2.5.18 GIOPSLA / B (Pull Up / Down 설정)

GIOPSL 레지스터는 Pull Up / Pull Down 기능을 설정한다.

Bit	필드	값	내용
31 - 8	Reserved	0	
7 - 0	GIOPSL[n]	0 1	GIO 포트의 [n]번 핀의 Pull Up / Pull Down 기능을 설정한다.[7:0] GIOPULDIS 비트를 0 으로 지우면 해당 기능이 활성화 된다. Pull Down 으로 설정한다. Pull Up 으로 설정한다.

2.6 HAL Code Generator

2.6.1 gioInit(void)

GIO 를 사용하기에 있어서 지정된 초기값을 세팅한다.

```
void gioInit(void)
{
    gioREG->GCR0    = 1U;
    gioREG->ENACLR = 0xFFU;
    gioREG->LVLCLR = 0xFFU;

    gioPORTA->DOUT = (uint32)((uint32)0U << 0U);
    gioPORTA->DIR   = (uint32)((uint32)0U << 0U);
    gioPORTA->PDR   = (uint32)((uint32)0U << 0U);
    gioPORTA->PSL   = (uint32)((uint32)0U << 0U);
    gioPORTA->PULDIS   = (uint32)((uint32)0U << 0U);

    gioPORTB->DOUT = (uint32)((uint32)0U << 0U);
    gioPORTB->DIR   = (uint32)((uint32)0U << 0U);
    gioPORTB->PDR   = (uint32)((uint32)0U << 0U);
    gioPORTB->PSL   = (uint32)((uint32)0U << 0U);
    gioPORTB->PULDIS   = (uint32)((uint32)0U << 0U);

    gioREG->POL = (uint32)((uint32)0U << 0U);
    gioREG->LVLSET = (uint32)((uint32)0U << 0U);
    gioREG->FLG = 0xFFU;
    gioREG->ENASET = (uint32)((uint32)0U << 0U);
}
```

```
// GIO 를 사용할 준비가 되어있다.
// 초기값으로 인터럽트를 전부 비활성화 한다.
// 초기값으로 인터럽트 채널을 Low-level 로 한다.

// 출력 데이터를 설정한다.
// 입 / 출력 방향을 설정한다.
// Open Drain 설정을 한다.
// Pull Up / Down 을 설정한다.
// Pull 을 설정한다.

// edge 의 상승 / 하강을 설정한다.
// 인터럽트 채널을 설정한다.
// 플래그를 초기화한다.
// 인터럽트를 세팅한다.
```

2.6.2 gioSetDirection(gioPORT_t *, uint32)

GIO 의 입 / 출력 방향을 설정한다.

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

```
// 받아온 port 의 핀을 dir 로 입 / 출력설정을 한다.
```

2.6.3 gioSetBit(gioPORT_t *, uint32, uint32)

GIO 의 출력을 설정한다.

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

```
// 값에 따른 SET / CLR 세팅한다.

// 세팅 값이 1 일 경우, SET 을 통해 High 로 설정한다.

// 세팅 값이 0 일 경우, CLR 를 통해 Low 로 설정한다.
```

2.6.4 gioSetPort(gioPORT_t *, uint32)

출력을 DOUT 을 통해 직접적으로 제어한다.

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

// 출력을 0 / 1 을 통해 직접적으로 제어한다.

2.6.5 uint32 gioGetBit(gioPORT_t *, uint32)

데이터의 입력 상태를 받아온다.

```
uint32 gioGetBit(gioPORT_t *port, uint32 bit)
{
    return (port->DIN >> bit) & 1U;
}
```

// 데이터 입력 상태를 0 / 1 를 리턴으로 받아온다.

2.6.6 uint32 gioGetPort(gioPORT_t *)

해당 포트의 데이터 입력 상태 전부를 받아온다.

```
uint32 gioGetPort(gioPORT_t *port)
{
    return port->DIN;
}
```

// 데이터 입력 상태 전체를 리턴으로 받아온다..

2.6.7 gioToggleBit(gioPORT_t *, uint32)

해당 포트의 핀 출력을 반전시킨다.

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

// 해당 비트의 방향을 비교한다.

// 결과가 1 일 경우 출력을 Low 로 설정한다.

//.결과가 0 일 경우 출력을 High 로 설정한다.

2.6.8 gioEnableNotification(gioPORT_t *, uint32)

해당 핀의 인터럽트를 활성화 한다.

```
void gioEnableNotification(gioPORT_t *port, uint32 bit)
{
    if (port == gioPORTA)                          // port 를 확인하여 구분한다.
    {
        gioREG->ENASET = (uint32)1U << bit;        // 해당 비트의 인터럽트를 활성화 한다.
    }
    else if (port == gioPORTB)
    {
        gioREG->ENASET = (uint32)1U << (bit + 8U); // 인터럽트 활성화는 한번에 관리하기 때문에 + 8U 로 처리한다.
    }
    else
    {

    }
}
```

2.6.9 gioDisableNotification(gioPORT_t *, uint32)

해당 핀의 인터럽트를 비활성화 한다.

```
void gioDisableNotification(gioPORT_t *port, uint32 bit)
{
    if (port == gioPORTA)                          // port 를 확인하여 구분한다.
    {
        gioREG->ENACLR = (uint32)1U << bit;        // 해당 비트의 인터럽트를 비활성화 한다.
    }
    else if (port == gioPORTB)
    {
        gioREG->ENACLR = (uint32)1U << (bit + 8U); // 인터럽트 비활성화는 한번에 관리하기 때문에 + 8U 로 처리한다.
    }
    else
    {

    }
}
```

2.6.10 gioNotification(gioPORT_t *, uint32)

인터럽트 발생시 사용하고자 하는 함수이다. 해당 함수는 유저가 조정할 수 있다.

```
void gioNotification(gioPORT_t *port, uint32 bit)
{

}
```

2.6.11 gioGetConfigValue(gio_config_reg_t *, config_value_type_t)

config_reg 에 디폴트 값이나 현재 상태 값을 받아온다.

type 은 InitialValue (0), CurrentValue (1)로 구분되며, type 에 따라 값을 받아오게 된다.

```
void gioGetConfigValue(gio_config_reg_t *config_reg, config_value_type_t type)
{
    if (type == InitialValue)
    {
        config_reg->CONFIG_INTDET      = GIO_INTDET_CONFIGVALUE;
        config_reg->CONFIG_POL         = GIO_POL_CONFIGVALUE;
        config_reg->CONFIG_INTENASET = GIO_INTENASET_CONFIGVALUE;
        config_reg->CONFIG_LVLSET      = GIO_LVLSET_CONFIGVALUE;

        config_reg->CONFIG_PORTADIR      = GIO_PORTADIR_CONFIGVALUE;
        config_reg->CONFIG_PORTAPDR      = GIO_PORTAPDR_CONFIGVALUE;
        config_reg->CONFIG_PORTAPSL      = GIO_PORTAPSL_CONFIGVALUE;
        config_reg->CONFIG_PORTAPULDIS = GIO_PORTAPULDIS_CONFIGVALUE;

        config_reg->CONFIG_PORTBDIR      = GIO_PORTBDIR_CONFIGVALUE;
        config_reg->CONFIG_PORTBPDR      = GIO_PORTBPDR_CONFIGVALUE;
        config_reg->CONFIG_PORTBPSL      = GIO_PORTBPSL_CONFIGVALUE;
        config_reg->CONFIG_PORTBPULDIS = GIO_PORTBPULDIS_CONFIGVALUE;
    }
    else
    {
        config_reg->CONFIG_INTDET      = gioREG->INTDET;
        config_reg->CONFIG_POL         = gioREG->POL;
        config_reg->CONFIG_INTENASET = gioREG->ENASET;
        config_reg->CONFIG_LVLSET      = gioREG->LVLSET;

        config_reg->CONFIG_PORTADIR      = gioPORTA->DIR;
        config_reg->CONFIG_PORTAPDR      = gioPORTA->PDR;
        config_reg->CONFIG_PORTAPSL      = gioPORTA->PSL;
        config_reg->CONFIG_PORTAPULDIS = gioPORTA->PULDIS;

        config_reg->CONFIG_PORTBDIR      = gioPORTB->DIR;
        config_reg->CONFIG_PORTBPDR      = gioPORTB->PDR;
        config_reg->CONFIG_PORTBPSL      = gioPORTB->PULDIS;
        config_reg->CONFIG_PORTBPULDIS = gioPORTB->PSL;
    }
}
```

2.6.1 gioHighLevelInterrupt(void) / gioLowLevelInterrupt(void)

High / Low Level 에 따른 함수를 사용한다.

LowLevel 의 경우 OFF2 로 변경된다.

```
void gioHighLevelInterrupt(void)
{
    uint32 offset = gioREG->OFF1;

    if (offset != 0U)
    {
        offset = offset - 1U;
        if (offset >= 8U)
        {
            gioNotification(gioPORTB, offset - 8U);
        }
        else
        {
            gioNotification(gioPORTA, offset);
        }
    }
}
```

```
// OFF 값을 받아온다.
// OFF 값이 0(저장된 인터럽트가 없을 때)이 아닐 때
실행된다. 해당 읽기로 OFF 의 해당 비트는 클리어 된다.
// OFF 의 경우 0 번이 인터럽트 없음을 나타내기 때문에
1 부터 0 번 핀을 나타낸다. 이를 맞추기 위하여 -1 처리를
한다.
// OFF 의 8 비트를 기준으로 A / B 구분을 한다.
// 만들어 둔 함수를 읽어온다.



// 만들어 둔 함수를 읽어온다.
```