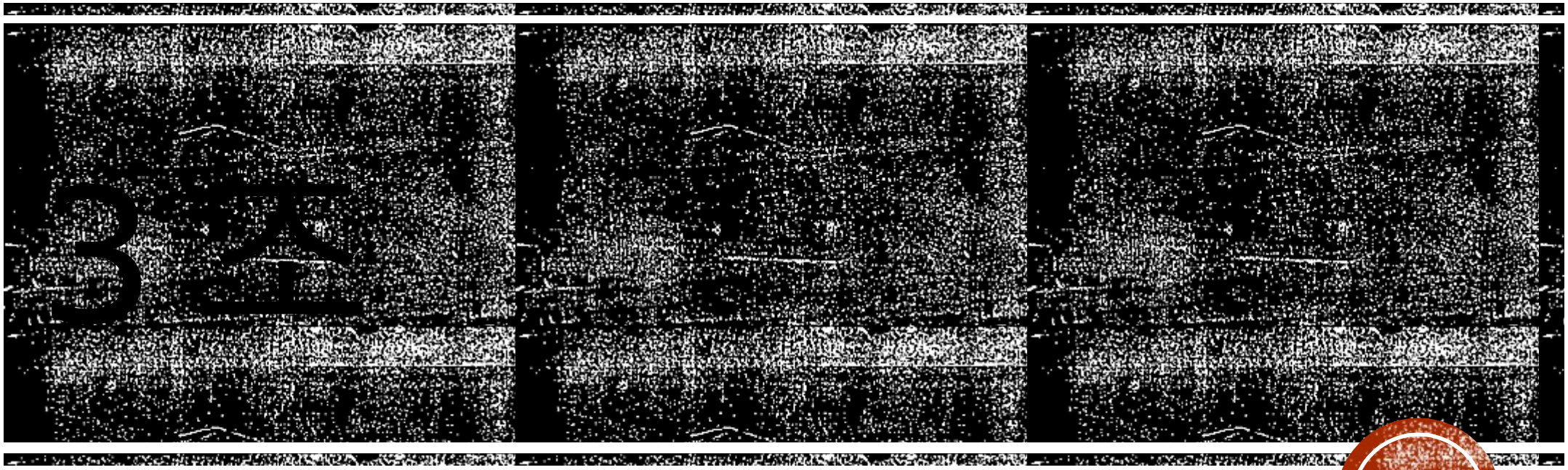

자율주행 자동차 1 주차 18. 07. 04 ~ 18. 07. 11

조원 : 하성용

역할 : GIO 분석

```
/* GIO Interface Functions */
void gioInit(void);
void gioSetDirection(gioPORT_t *port, uint32 dir);
void gioSetBit(gioPORT_t *port, uint32 bit, uint32 value);
void gioSetPort(gioPORT_t *port, uint32 value);
uint32 gioGetBit(gioPORT_t *port, uint32 bit);
uint32 gioGetPort(gioPORT_t *port);
void gioToggleBit(gioPORT_t *port, uint32 bit);
void gioEnableNotification(gioPORT_t *port, uint32 bit);
void gioDisableNotification(gioPORT_t *port, uint32 bit);
void gioNotification(gioPORT_t *port, uint32 bit);
void gioGetConfigValue(gio_config_reg_t *config_reg, config_value_type_t type);
```

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

    /** - Port output values */
    gioPORTA->DOUT

    /** - Port direction */
    gioPORTA->DIR

    /** - Port open drain enable */
    gioPORTA->PDR

    /** - Port pullup / pulldown selection */
    gioPORTA->PSL

    /** - Port pullup / pulldown enable*/
    gioPORTA->PULDIS

    /** - interrupt polarity */
    gioREG->POL

    /** - interrupt level */
    gioREG->LVLSET

    /** - clear all pending interrupts */
    gioREG->FLG = 0xFFU;

    /** - enable interrupts */
    gioREG->ENASET
```

GCR0 // 전역제어레지스터로 모듈 리셋상태를 제어하는 bit0 이 있음
이 bit 에 1 은 정상작동을 의미하고 0 은 리셋상태 (비작동) 상태를 의미

INTDET // 인터럽트 검출 레지스터로 gio 모듈에 상승에지 또는 하강에지가 있을때 CPU 에 대한 인터럽트 요청 생성을 지원 ,
두 에지가 gio 핀에서 감지되면 상승과 하강을 허용

POL // 인터럽트극성레지스터로 상승에지 또는 하강에지의 극성을 구성 ,
디바이스가 저전력모드에 있을때 레벨을 하이 또는 로우로 제어

ENASET // 인터럽트 사용 설정 레지스터로 인터럽트 요청을 CPU 로 전달하고 인터럽트 요청 (IRQ/FIQ) 에 응답하려면 지워야함

ENACLR // 인터럽트를 비활성화시키는 레지스터

LVLSET // 인터럽트 우선순위 설정레지스터로 VIM 으로 가는 하이레벨 인터럽트로
구성되는데 사용되며 하이레벨 인터럽트는 GIOOFF1 및 GIOEMU1 에 기록

LVLCLR // 인터럽트 우선순위 클리어레지스터로 VIM 으로 가는 로우 레벨 인터럽트를
구성하는데 사용하며 로우레벨 인터럽트는 GIOOFF2 및 GIOEMU2 에 기록

FLG // 인터럽트 플래그 레지스터로 플래그는 설정된 플래그에 1 을 쓰는 CPU
혹은 GIOOFF1 또는 GIOOFF2 를 읽음으로써 지워집니다

OFF1 // 현재 보류중인 우선순위가 높은 인터럽트

OFF2 // 현재 보류중인 우선순위가 낮은 인터럽트

EMU1 // 읽기 전용 레지스터로 현재 보류중인 우선순위가 높은 인터럽트를
플래그를 지우지 않고 사용

EMU2 // 읽기 전용 레지스터로 현재 보류중인 우선순위가 낮은 인터럽트를
플래그를 지우지 않고 사용

DIR // 데이터 방향 레지스터로 주어진 포트의 핀이 입력 또는 출력으로 구성여부를 제어

DIN // 데이터 입력 레지스터

DOUT // 데이터 출력 레지스터

DSET // 데이터 출력 설정 레지스터

DCLR // 데이터 출력 클리어 레지스터

PDR // 오픈 드레인 레지스터로 GIODOUTx 비트는 GIO 의 상태를 제어

PULDIS // 풀업 레지스터로 핀의 풀 제어 기능을 활성화 또는 비활성화

PSL // 풀 업 / 다운 선택 레지스터로 핀의 풀업 또는 풀다운 기능을 선택

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

ex) gioSetDirection(gioPORTA, 0xffffffff)

//gioPORTA ->DIR 을 1 로 채워서 핀 모두를 출력으로 설정

```
void gioSetPort(gioPORT_t *port, uint32 value)
{
/* USER CODE BEGIN (6) */
/* USER CODE END */

    port->DOUT = value;

/* USER CODE BEGIN (7) */
/* USER CODE END */

}
```

gioSetPort(gioPORTA, 0xffffffff);

//gioPORTA ->DOUT 을 1 로 채워서 모든 포트가 low 로 설정

```
void gioSetBit(gioPORT_t *port, uint32 bit, uint32 value)
{
/* USER CODE BEGIN (5) */
/* USER CODE END */

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

ex) gioSetBit(gioPORTA, 6, 1)
//6 번 비트의 출력을 1 로 설정

```
uint32 gioGetBit(gioPORT_t *port, uint32 bit)
{
/* USER CODE BEGIN (8) */
/* USER CODE END */

    return (port->DIN >> bit) & 1U;
}

uint32 gioGetPort(gioPORT_t *port)
{
/* USER CODE BEGIN (9) */
/* USER CODE END */

    return port->DIN;
}
```

```
void gioToggleBit(gioPORT_t *port, uint32 bit)
{
/* USER CODE BEGIN (10) */
/* USER CODE END */

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

```
void gioEnableNotification(gioPORT_t *port, uint32 bit)
{
/* USER CODE BEGIN (11) */
/* USER CODE END */

    if (port == gioPORTA)
    {
        gioREG->ENASET = (uint32)1U << bit;
    }
    else if (port == gioPORTB)
    {
        gioREG->ENASET = (uint3
    }
    else
    {
        /* Empty */
    }
}
```

```
void gioDisableNotification(gioPORT_t *port, uint32 bit)
{
/* USER CODE BEGIN (12) */
/* USER CODE END */

    if (port == gioPORTA)
    {
```

```
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;
        g_reg->CONFIG_PORTBPDR    = GIO_PORTBPDR_CONFIGVALUE;
        g_reg->CONFIG_PORTBPSL    = GIO_PORTBPSL_CONFIGVALUE;
        g_reg->CONFIG_PORTBPULDIS = GIO_PORTBPULDIS_CONFIGVALUE;
```

```
gma WEAK(gioNotification)
 gioNotification(gioPORT_t *port, uint32 bit)

enter user code between the USER CODE BEGIN and USER CODE END. */
SER CODE BEGIN (22) */
```