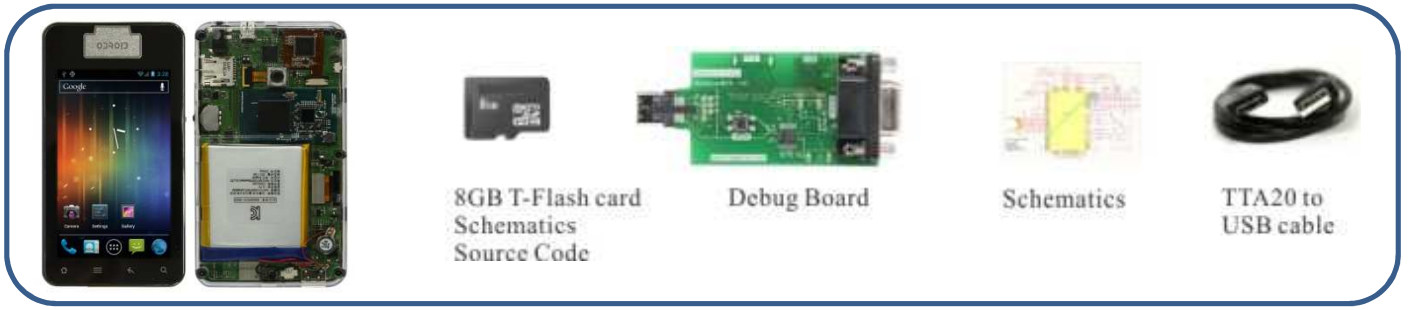
삼성 Exynos-4210 듀얼코어 프로세서로 안드로이드 배우는 플랫폼 학습

Agenda

시작하기에 앞서(1)

- 본 과정은 ODROID-A4를 사용하여 안드로이드 시스템 개발에서부터 앱 개발까지의 과정을 포함하고 있다.
- ODROID-A4는 안드로이드 OS가 설치되어진 8GB T-Flash카드와 시스템을 디버깅할 수 있는 디버그보드, 컴퓨터와 통신할 수 있는 TTA20 to USB케이블로 구성되어 있다.



- 안드로이드 시스템을 개발하기 위해서는 USB to Serial 변환장치가 별도로 필요하다.
- 옆의 사진은 전 과정을 수행하기 위한 개발환경이다.
  기초과정에서는 Debug Board를 바로 ODROID-A4에 연결하여 사용하면 된다.

시작하기에 앞서(2)

- 실전 응용 과정은 ODROID-A4의 입출력 장치들에 대한 설명과 확장보드인 ODROID-A4 IO 보드를 활용하여 외부 하드웨어를 제어하는 방법을 설명하고 있다.
- 회로도에 대한 설명부터, 플랫폼에서 드라이버 만들고, 이를 활용할 수 있는 앱을 제작하여 테스트할 수 있는 방법을 설명하고 있다. 한다. 는 방법과 드라이버를 기준으로 앱을 만들고, 앱이 어떻게 연동되는지를 익힐 수 있다.
- 궁금한 내용은 게시판에 질문을 남기자. 단, 다른 사람이 한 질문 중에 내가 아는 답이 있으면 꼭 답을 달아주자. 혹시 아는가? 답을 잘 달아주는 사람은 하드커널에서 선물이라도 보내줄지.

  http://com.odroid.com/sigong/nf_board/nboard.php?brd_id=odroidaf

3. 응용 과정 : 하드웨어 확장 및 활용

Agenda

- What interfaces are available in Exynos-4210?
- 하드웨어 확장 – GPIO
  - Android Application : labbook-KeyInput
- 하드웨어 확장 – ADC
  - Android Application : labbook-ADC
- 하드웨어 확장 – ADC(Oscilloscope)
  - Android Application : OdroidOscilloscope
- 하드웨어 확장 – UART(GPS)
  - Android Application : Labbook-GPS
- 하드웨어 확장 – I2C(Sensor)
  - Android Application : Labbook-BMP180
- 하드웨어 확장 – I2C(Expender)
  - Android Application : Labbook-MCP23017

ODROID
Hardkernel

What interfaces are available in Exynos-4210? (Brief explanation of how it works: UART, GPIO, I2C, ADC)

Agenda

- UART란?
- UART of Exynos4210
- GPIO란?
- GPIO of Exynos4210
- GPIO Configuration of Exynos4210
- I2C 란?
- I2C of Exynos4210
- ADC 란?
- ADC of Exynos4210

UART 란 ?

- **UART**(범용 비동기화 송수신기: Universal asynchronous receiver/transmitter)는 병렬 및 직렬 방식으로 데이터를 번역하는 컴퓨터 하드웨어의 일종이다. UART는 일반적으로 EIA RS-232, RS-422, RS-485와 같은 통신 표준과 함께 사용한다. UART의 U는 범용을 가리키는데 이는 자료 형태나 전송 속도를 직접 구성할 수 있고 실제 전기 신호 수준과 방식(이를테면 차분 신호)이 일반적으로 UART 바깥의 특정한 드라이버 회로를 통해 관리를 받는다는 뜻이다.
- UART는 일반적으로 컴퓨터나 주변 기기 직렬 포트의 직렬 통신을 위해 사용되는 개별 집적 회로이다. UART는 보통 마이크로컨트롤러에도 포함되어 있다. 듀얼 UART, 곧 **DUART**는 두 개의 UART를 하나의 칩에 합친 것이다. 수많은 현대의 집적 회로(IC)는 동기화 통신도 지원하는 UART와 함께한다. 이러한 장치들은 **USARTs**(범용 동기화 송수신기: universal synchronous/asynchronous receiver/transmitter)로 부른다.

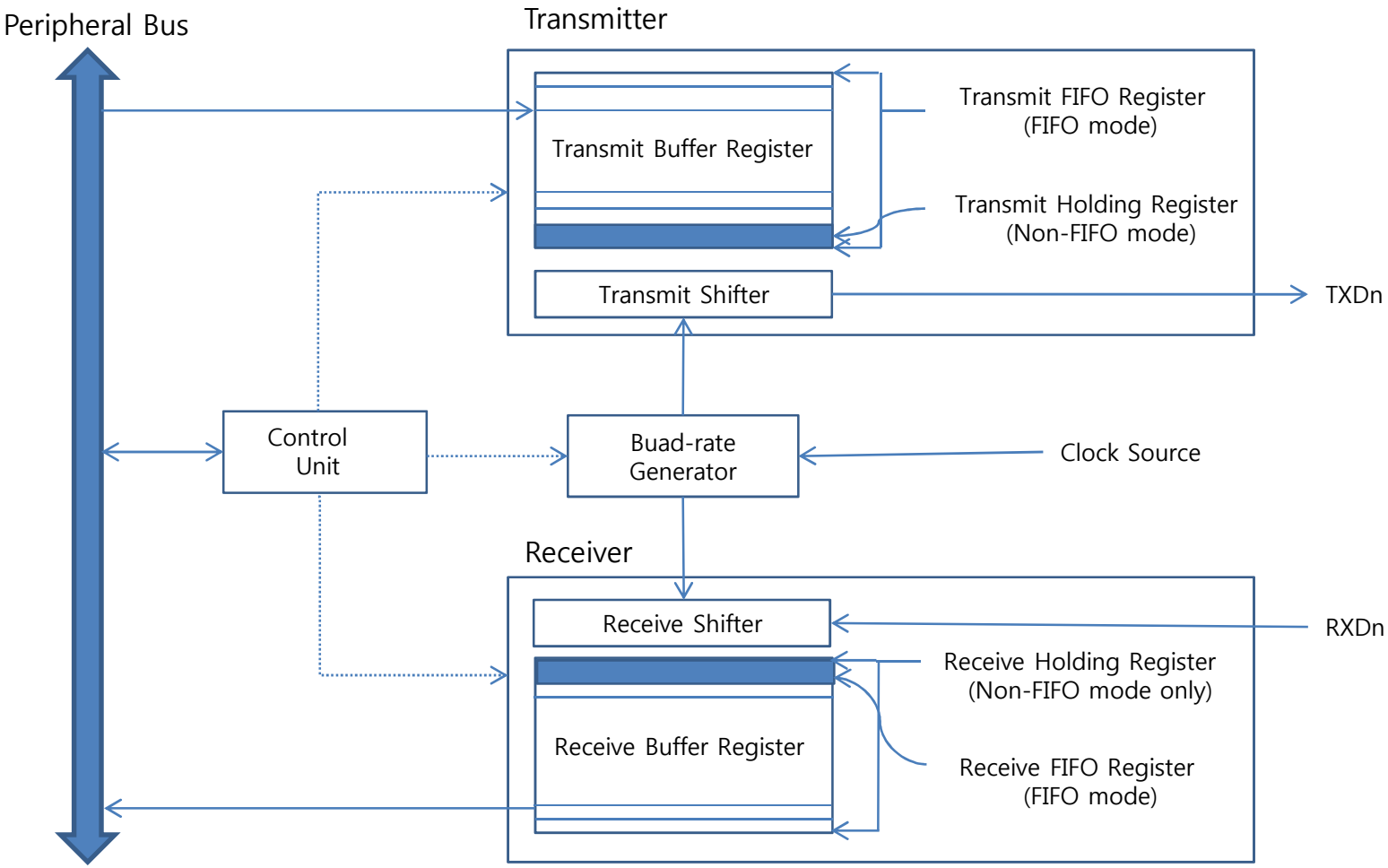
UART Of Exynos4210(1)

Exynos 4210의 비동기 통신(UART) 주요 특징

- DMA 기반 또는 인터럽트 기반 으로 작동하는 5개의 UART
- 5-bit, 6-bit, 7-bit, or 8-bit serial data 송신/수신 지원
- Rx/Tx independent 256nbyte FIFO for UART0, 64 byte FIFO for UART1 and 16 byte FIFO for UART2/3/4
- Programmable baud rate
- IrDA 1.0 SIR (115.2 Kbps) mode 지원
- Loop back mode for testing
- Non-integer clock divides in Baud clock generation

Base Address	IP
0x1380_0000	UART0
0x1381_0000	UART1
0x1382_0000	UART2
0x1383_0000	UART3
0x1384_0000	UART4

Block Diagram of UART



UART Of Exynos4210(3)

UART 송신

데이터 송신 레지스터로는 UTXHn(UART Transmit Buffer)가 있고, Transmit Shifter는 사용자가 프로그램적으로 접근 할 수 없고, 회로적으로 UTXHn의 내용이 적재되어, 변조 주파수에 맞추어 Shift 되면서, TXDn 핀으로 출력된다.

UART 수신

데이터 수신 레지스터로는 URXHn(UART Receive Buffer) 가 있고, Receive Shifter는 RXDn 핀으로 수신된 신호가 변조 주파수에 맞추어 Shift 되다가, Stop bit가 들어오면, URXHn으로 저장된다. URXHn의 내용은 수신된 데이터로 Peripheral Bus를 통해서 사용자가 읽어서 처리한다.

제어/상태

주요 레지스터로는 ULCONn(UART Line Control), UTRSTATn(UART RX/TX Status), UFCONn(UART FIFO Control), UFSTATn(UART FIFO Status) 등이 있다.

UTXHn (n = 0 to 4)

- Address =Base address + 0x0020, Reset value = 0x0000_0000

Name	bit	Type	Decription	Reset value
RSVD	[31:8]	-	Reserved	-
UTXHn	[7:0]	W	Transmit data for UARTn	-

URXHn (n = 0 to 4)

- Address =Base address + 0x0024, Reset value = 0x0000_0000

Name	bit	Type	Decription	Reset value
RSVD	[31:8]	-	Reserved	-
URXHn	[7:0]	W	Receive data for UARTn	-

ULCONn (n = 0 to 4)

- Address =Base address + 0x0000, Reset value = 0x0000_0000

Name	bit	Type	Decription	Reset value
RSVD	[31:7]	-	Reserved	0
Infrared mode	[6]	RW	Determines whether to use the Infrared mode. 0 = Normal mode operation	0
Parity mode	[5:3]	RW	Specifies the type of parity generation to be performed and checking during UART transmit and receive operation. 0xx = No parity, 100 = Odd parity, 101= Even parity, 110 = Parity forced/ checked as 1 111 = Parity forced/ checked as 0	000
Number of Stop bit	[2]	RW	Specifies how many stop bits are used to signal end-of-frame signal. 0 = One stop bit per frame, 1 = Two stop bit per frame	0
Word Length	[1:0]	RW	Indicates the number of data bits to be transmitted or received per frame. 00 = 5-bit, 01 = 6-bit, 10 = 7-bit, 11 = 8-bit	00

UART Of Exynos4210(6)

- Interface Port Description of UART for ODROID-A4

Singal	ODROID-A4 용도	비고
UART0	Bluetooth	GB8632
UART1	Debug	TTA20
UART2	GPS Module	I/O 확장
UART3	사용하지 않음	
UART4	내부 GPS 모듈이 있는 경우에 사용하기 때문에 UART4 는 밖으로 나와 있는 IO PORT가 없다	

GPIO 란 ?

- GPIO (General Purpose Input Output) 장치는 말 그대로 어떠한 특별한 임무를 하는 핀이 아닌, 소프트웨어 제어로 하나의 하드웨어 핀이 입력도 되거나 출력 되는 장치 이다.
- GPIO 기능
  - 방향 : GPIO 핀은 입력 또는 출력을 설정
  - 입력 핀을 인터럽트로 설정
  - PULL-UP/PULL-DOWN 설정

GPIO of Exynos4210(1)

Exynos4210 GPIOs

Exynos4210는 252개의 다기능 input/output 과 156 메모리 port 핀을 가지고 있다. 아래 처럼 37개의 일반 port 그룹과 2개의 메모리 port그룹으로 나누어 관리한다.

- GPA0,GPA1: 14 in/out ports - 3xUART with flow control, UART without flow control, and/ or 2xI2C
- GPB: 8 in/out ports - 2xSPI and/or I2C and/ or IEM
- GPC0,GPC1: 10 in/out ports - 2xI2S, and/or 2xPCM, and/or AC97, SPDIF, I2C, and/ or SPI
- GPD0,GPD1: 8 in/out ports - PWM, I2C, and/or LCD I/F, I2C
- GPE0,GPE1,GPE2,GPE3,GPE4: 35 in/ out ports - Modem I/F, and/or CAM I/F, and/or TS I/F, and/ or LCD I/F, and/ or Trace I/F
- GPF0, GPF1,GPF2,GPF3: 30 in/ out ports - LCD I/F
- GPJ0, GPJ1: 13 in/out ports - CAM I/F
- GPK0, GPK1,GPK2,GPK3: 28 in/out ports - 4xMMC(4-bit MMC), and/ or 2xMMC(8-bit MMC)
- GPL0, GPL1: 11 in/out ports - GPS I/F
- GPL2: 8 in/ out ports - GPS debugging I/F or Key pad I/F
- GPX0, GPX1, GPX2, GPX3: 32 in/out ports - External wake-up, and/ or Key pad I/F

GPIO of Exynos4210(2)

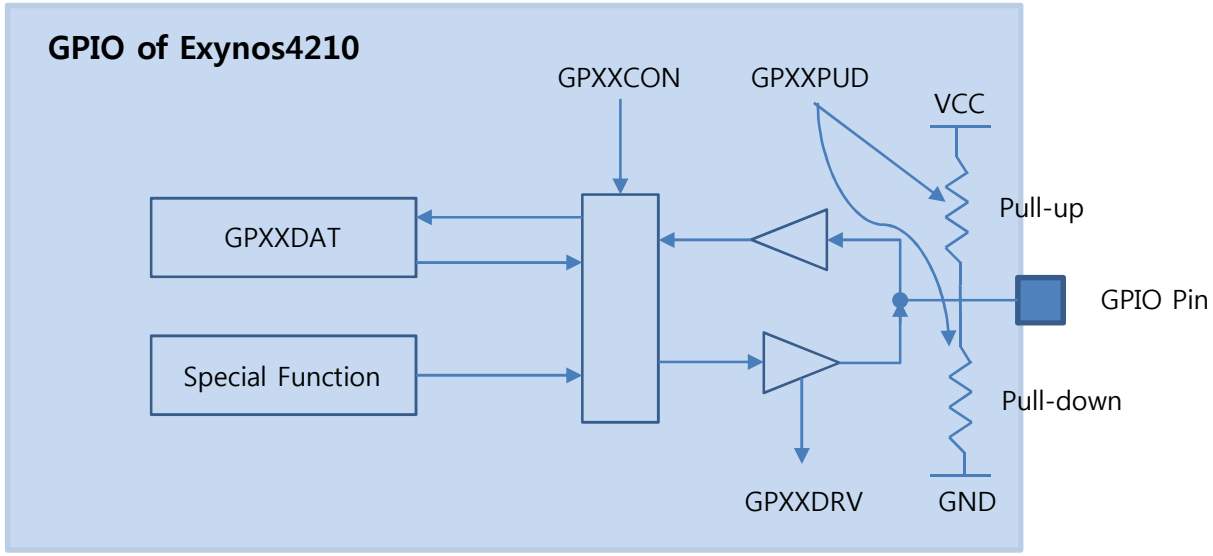
- GPZ: 7 in/out ports - Low Power I2S and/or PCM
- GPY0, GPY1, GPY2: 16 in/ out ports - Control signals of EBI (SROM, NF, OneNAND)
- GPY3,GPY4,GPY5,GPY6: 32 in/ out memory ports - EBI (For more information about EBI configuration, refer to Chapter 5, and 6)
- MP1_0 – MP1_9: 78 DRAM1 ports
- ETC0,ETC1,ETC6: 18 in/out ETC ports - JTAG, SLIMBUS, RESET, CLOCK

➢ Exynos4210의 GPIO 특징

- Controls 172 External Interrupts
- Controls 32 External Wake-up Interrupts
- 252 multi-functional input/ output ports
- Controls pin states in Sleep Mode except GPX0, GPX1, GPX2, and GPX3 (GPX* pins are alive-pads)
- 일반적인 입력/출력 기능에 외에 다른 신호 핀으로 사용 될 수 있도록 다중 방식으로 되어 있다.

GPIO Configuration of Exynos4210

- Block Diagram of GPIO



- GPIO를 제어하는 레지스터는 네 가지로 나누어 볼 수 있다.
  - GPXXCON 은 입력, 출력 , 특수기능, Interrupt 등을 선택 할 수 있다.
  - GPXXDAT 은 입력을 설정 할 때는 읽을 수 있고, 출력을 설정 할 때는 쓸 수 있고, 특수 기능을 사용 할 때는 정의되지 않는 값이 읽힌다.
  - GPXXPUD 은 Pull-up/down 끄기, Pull-down 켜기, Pull-up 켜기를 설정 할 수 있다.
  - GPXXDRV 은 출력 강도를 1x, 2x, 3x, 4x 기본 값의 배수로 선택 할 수 있다.

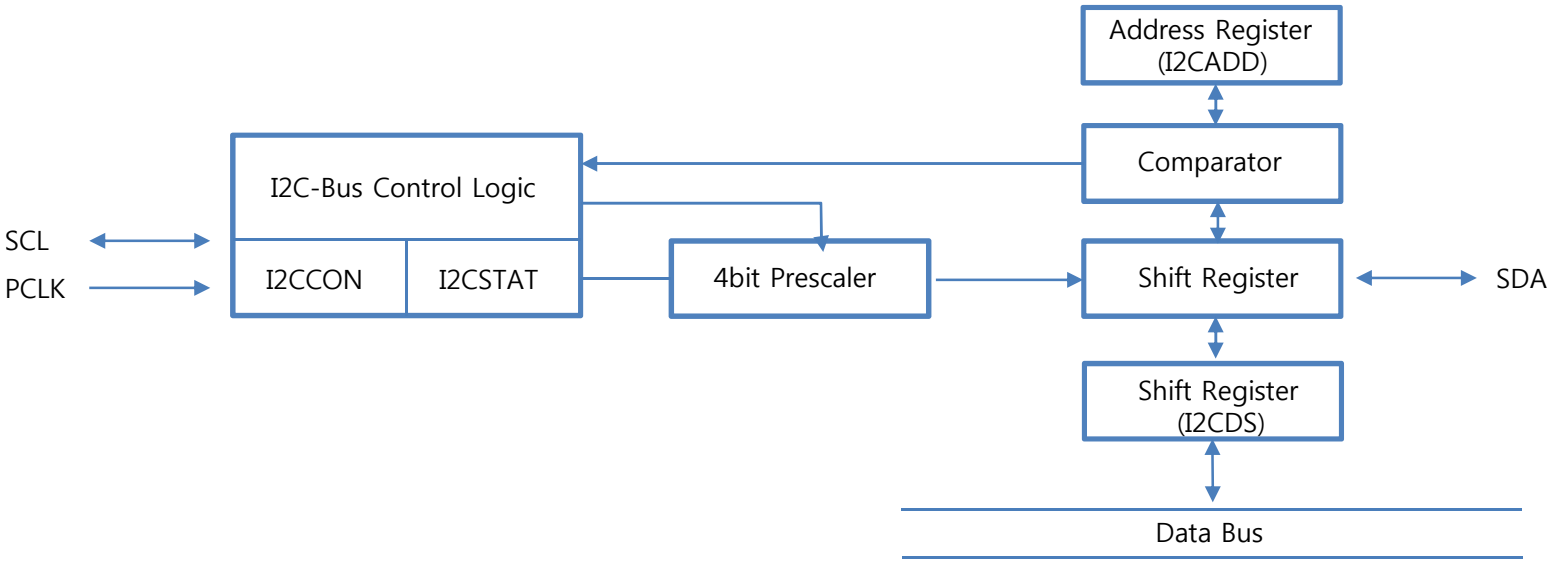
I2C 란?

- I2C(I-square-C, '아이스퀘어시'라고 보통 부른다)란 필립스가 제안한 통신 방식이다. Inter-Integrated IC라고도 불리지만 이 명칭은 그리 잘 쓰이지 않는 명칭이다. I2C는 로컬 버스라고 부르는 병렬 버스와 다르게 주변 장치를 단지 두 가닥의 신호선으로만 연결하여 동작하는 양방향 직렬 버스 규격이다. 필립스는 TV, VCR, 오디오 장비 등과 같은 대량 생산되는 제품용으로 I2C 버스를 이미 20년 전에 소개했는데 지금은 내장 장치를 다루기 위한 사실상의 표준 솔루션이 되었다. I2C 버스에는 표준, 고속, 초고속 등 속도에 따라 세 가지 데이터 전송 모드가 있다. 표준 모드는 100Kbps, 고속은 400Kbps 그리고 초고속 모드에서는 최고 3.4Mbps의 속도를 지원한다. 이 세 가지 모두 하위 호환성을 갖는다. I2C 버스는 각 장치에 7비트와 10비트 주소를 지정하여 여러 장치들을 독립적으로 접근할 수 있다.

I2C of Exynos4210(1)

- Exynos4210 I2C 특징
  - 9 channels Multi-Master, Slave I2C BUS interfaces (8 channels for general purpose, 1 channel for HDMI dedicated)
  - 7-bit addressing mode
  - Serial, 8-bit oriented, and bidirectional data transfer
  - Supports up to 100kbit/s in the Standard mode
  - Supports up to 400kbit/s in the Fast mode
  - Supports master transmit, master receive, slave transmit and slave receive operation
  - Supports interrupt or polling events
- 다중 마스터 I2C bus 작동을 위해서는 다음 레지스터를 써야 한다.
  - I2CCON(I2C Control Register)
  - I2CSTAT(I2C Control/Status Register)
  - I2CDS(I2C Tx/Rx Data Shift Register)
  - I2CADD(I2C bus Address Register)

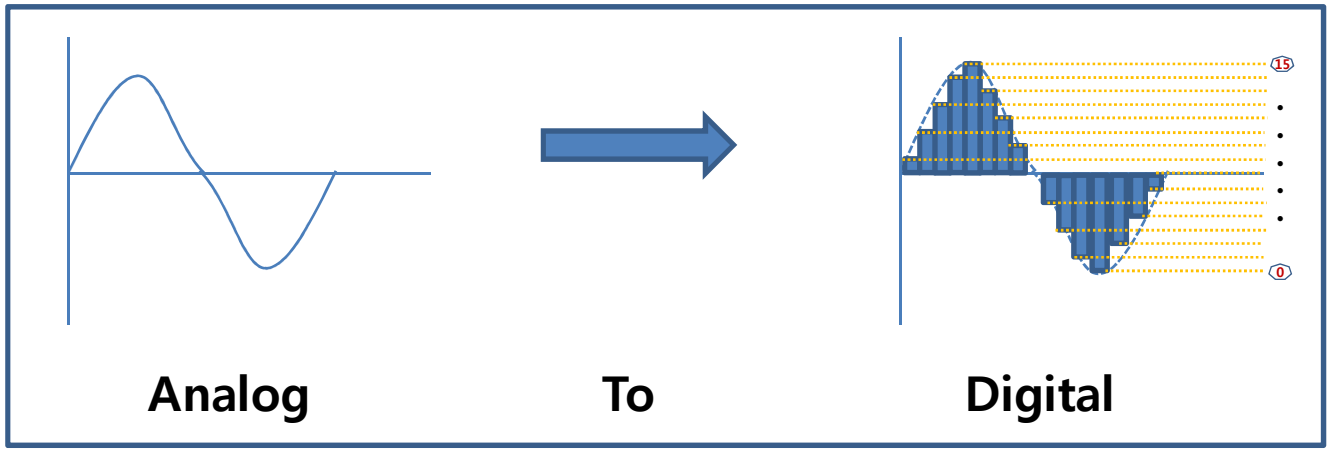
Block Diagram 의 Exynos4210 I2C



- 시리얼 클럭(SCL)의 주파수를 제어 하기 위해서는 I2CCON 레지스터의 4 bit prescaler 값을 쓰고, I2C의 Slave 주소는 I2CADD에 저장한다. I2CDS 레지스터는 8bit data shift register로 I2CSTAT의 Serial output enable = 1 일 때, I2CDS를 쓸 수 있고, 읽는 것은 언제든지 가능하다.

ADC 란?

: ADC 를 우리말로 바꾸면 아날로그 디지털 변환기이다.



위 그림은 오로지 ADC 를 이해하기 위한 그림이다.

위 그림 우측의 Digital 쪽의 예에서 막대그래프의 총 층수가 16개인 것을 두고,
2^4(0 ~ 15) 으로 4bit 분해능(resolution)를 가졌다라고 이야기한다.
분해능(resolution)이 높을 수록 더욱더 촘촘하게 표현 가능하여, 아날로그신호와 가깝게 표현할 수 있다.

아날로그신호를 디지털신호로 변환하는 데 걸리는 시간을 A/D Conversion time 이라고 한다.
위 그림에서 막대그래프로 하나 변환하는 데 걸리는 시간을 의미한다.

ADC of Exynos4210(1)

: **Exynos4210 CPU 의 ADC 핀에 Analog 신호가 들어오면 CPU의 해당 ADC 핀 관련 레지스터를 설정함으로써, 아날로그신호가 디지털신호로 변환된 값을 숫자로 볼 수 있다.**

1. Exynos4210 ADC 의 특징

- 10-channel Analog inputs : ADC 는 터치스크린 인터페이로 설계되었으며, 순수한 ADC 로 사용 가능하다.
- Resolution : 10-bit / 12-bit(optional)
- Maximum Conversion Rate : 1MSPS(Mega Samples per Second)

: When the PCLK frequency is 100MHz and the prescaler value is 19, total 12-bit conversion time is as follows.

A/D converter freq. = 100MHz/(19+1) = 5MHz

Conversion time = 1/(5MHz / 5cycles) = 1/1MHz = 1us

NOTE: This A/D converter was designed to operate at maximum 5MHz clock, so the conversion rate can go up to 1MSPS.

- Power Supply Voltage : 3.3V
- Analog Input Range : 0 ~ 3.3V
- Normal Conversion Mode
- Waiting for Interrupt Mode
- IDLE, DIDLE, STOP and DSTOP mode wakeup source
- Differential Nonlinearity Error : +- 1.0 LSB(Max.)
- Integral Nonlinearity Error : +- 4.0 LSB(Max.)

: ADC Registers

2. ADC Register Map Summary (Base Address = 0x1391_0000, 0x1391_1000)

Register	Offset	Descripton	Reset Value
TSADCCONn	0x0000	Specifies the TS0 – ADC Control Register	0x0000_3FC4
TSCONn	0x0004	Specifies the TS0 - Touch Screen Control Register	0x0000_0058
TSDLYn	0x0008	Specifies the TS0 - ADC Start or Interval Delay Register	0x0000_00FF
TSDATXn	0x000C	Specifies the TS0 - ADC Conversion Data X Register	Undefined
TSDATYn	0x0010	Specifies the TS0 - ADC Conversion Data Y Register	Undefined
TSPENSTATn	0x0014	Specifies the TS0 - Pen0 Up or Down Status Register	0x0000_0000
CLRINTADCn	0x0018	Specifies the TS0 - Clear ADC0 Interrupt	Undefined
ADCMUX	0x001C	Specifies the Analog input channel selection	0x0000_0000
CLRINTPENn	0x0020	Specifies the TS0 - Clear Pen0 Down/Up Interrupt	Undefined

: ADC 데이터 읽어오기

3. ADC 데이터 읽어오기(데이터쉬트 ADC part 참조)

A. TSADCCON0(Specifies the TS0 – ADC Control Register) 레지스터에서 분해능 (resolution)10bit/12bit 선택, Prescaler 값을 19 ~ 255 범위에서 선택하여 설정한다.

B. TSDLYn(Specifies the TS0 - ADC Start or Interval Delay Register) 레지스터에서 DELAY 값을 설정한다. 이 DELAY 은 디지털로 변환된 데이터를 읽어올 때, 인터럽트 서비스 루틴의 리턴 시간이나 데이터 엑세스 타임으로, 총 변환하는 데 필요한 시간의 일부이다.

C. ADCMUX(Specifies the Analog input channel selection) 레지스터에서 H/W 적으로 연결된 ADC channel 를 설정한다.

D. TSADCCON0(Specifies the TS0 – ADC Control Register) 레지스터의 ENABLE_START bit 를 Set 한 후, TSDATX0(Specifies the TS0 - ADC Conversion Data X Register) 레지스터의 XPDATA(Normal ADC) 비트값을 읽어 들인다.

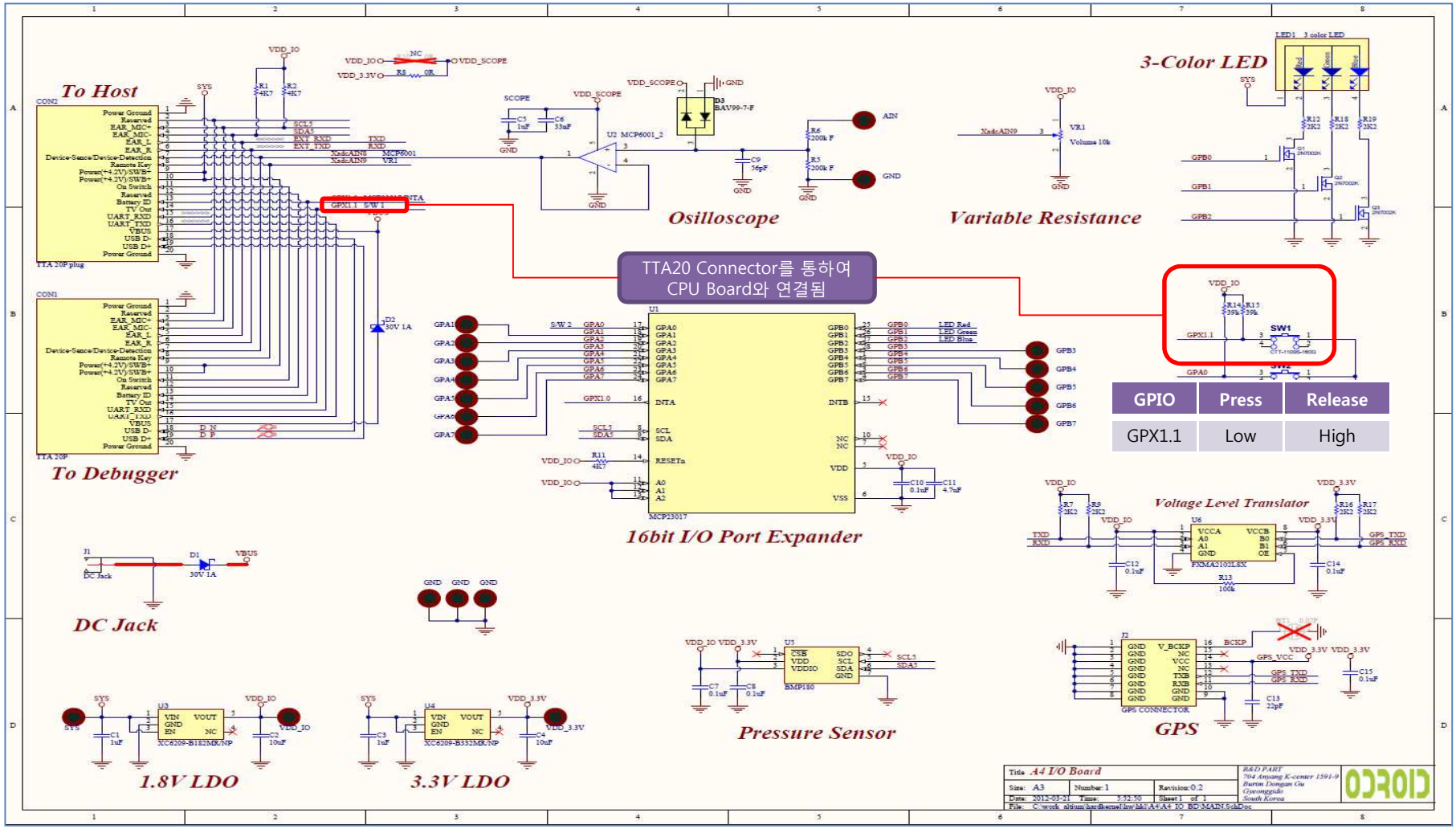
하드웨어 확장 - GPIO

Agenda

- ODROID-A4 IO Board GPIO Hardware Interface
- Timer IRQ를 사용한 GPIO Key Driver
  - 100 ms 마다 GPIO를 검사하여 'A' Key report 구현
- Long Press시에 다른 Key를 Report
  - 3 초 누르고 있는 경우 'B' Key report 구현
- Platform Data Struct
  - Platform Data Struct를 구성하고 및 초기값 설정
- SYSFS를 추가하기
  - SYSFS를 통하여 Driver Control
- External IRQ를 사용할 수 있도록 드라이버 수정
  - Platform Data에 Interrupt 방식을 설정하도록 수정
- Android Application : labbook-KeyInput
  - 작성된 Key Report Data를 표시

ODROID-A4 IO Board GPIO Hardware Interface

Odroid-A4 I/O Board의 GPIO연결



Timer IRQ를 사용한 GPIO Key Driver(1)

Platform driver register

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static struct platform_driver ioboard_gpio_driver = {
    .driver = {
        .name   = IOBOARD_DRIVER_NAME,
        .owner  = THIS_MODULE,
    },
    .probe      = ioboard_gpio_probe,
    .remove     = ioboard_gpio_remove,
    .suspend    = ioboard_gpio_suspend,
    .resume     = ioboard_gpio_resume,
};
...
static int ioboard_gpio_probe(struct platform_device *pdev)
{
...
}
...
static int __init ioboard_gpio_init(void)
{
    return platform_driver_register(&ioboard_gpio_driver);
}

static void __exit ioboard_gpio_exit(void)
{
    platform_driver_unregister(&ioboard_gpio_driver);
}

module_init(ioboard_gpio_init);
module_exit(ioboard_gpio_exit);
...
```

Platform driver probe를 위하여 Driver name을 동일하게 설정한다.

Platform device에 등록된 Name과 같은 Name을 갖는 module이 있는 경우 해당 module의 probe함수를 실행한다.

"ioboard-gpio" Name을 갖는 Platform Device Driver 등록

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.h

```
#define    IOBOARD_DRIVER_NAME      "ioboard-gpio"

#define    KEY_PRESS        1
#define    KEY_RELEASE      0

#define    SCAN_TIMER_PERIOD          100000000

#define    KEY_GPIO                   EXYNOS4_GPX1(1)
#define    KEY_GPIO_NAME              "ioboard key"

#define    KEY_CODE                    KEY_A
```

Kernel/arch/arm/mach-exynos/mach-odroid-4210.c

```
...
#include <../../../drivers/misc/odroida4-io/gpio/ioboard-gpio.h>

static struct platform_device  ioboard_gpio_device = {
    .name   = IOBOARD_DRIVER_NAME,
    .id     = -1,
};
...
static struct platform_device *smdkv310_devices[] __initdata = {
...
    &ioboard_gpio_device,
...
}
...
```

Platform device를 등록한다.

Timer IRQ를 사용한 GPIO Key Driver(2)

Platform driver struct 및 초기화

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
struct ioboard_gpio  {
    struct  input_dev   *input;
    struct  hrtimer     scan_timer;
    bool                enable;
};
...
static int ioboard_gpio_probe (struct platform_device *pdev)
{
    struct ioboard_gpio *ioboard_gpio;

    if(!(ioboard_gpio = kzalloc(sizeof(struct ioboard_gpio), GFP_KERNEL)))
        return -ENOMEM;

    dev_set_drvdata(&pdev->dev, ioboard_gpio);

    if(ioboard_gpio_port_init(ioboard_gpio))        goto err_gpio_init;

    if(ioboard_gpio_input_init(ioboard_gpio))       goto err_input_init;

    if(ioboard_gpio_irq_init(ioboard_gpio))         goto err_irq_init;

    if(ioboard_gpio_irq_enable(ioboard_gpio, true)) goto err_driver_init;

    return 0;
...
}
...
```

Driver 에서 사용할 struct 선언

Input device control

Key Scan시 사용할 timer 선언

Timer enable/disable control

초기화된 Kernel memory 할당

Driver 초기화 함수 호출

Scan Timer enable

Header file의 구성

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.h

```
#define    IOBOARD_DRIVER_NAME      "ioboard-gpio"

#define    KEY_PRESS        1
#define    KEY_RELEASE      0

#define    SCAN_TIMER_PERIOD        100000000

#define    KEY_GPIO                 EXYNOS4_GPX1(1)
#define    KEY_GPIO_NAME            "ioboard key"

#define    KEY_CODE                 KEY_A
```

Report 되어질 Key값 설정
Kernel/include/linux/input.h
파일의 KEY Code 참조

I/O Board의 Key와 연결된
GPIO 설정

Scan Timer 주기 설정
(nano sec 값으로 설정)

Platform Driver Name 정의

Timer IRQ를 사용한 GPIO Key Driver(3)

GPIO request 및 초기화

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_port_init (struct ioboard_gpio *ioboard_gpio)
{
    if(gpio_request(KEY_GPIO, KEY_GPIO_NAME))    return  -1;
    gpio_direction_input(KEY_GPIO);

    return  0;
}
...
```

사용할 GPIO 요청

입력포트로 설정

KEY_GPIO = EXYNOS4_GPX1(1)
KEY_GPIO_NAME = "ioboard key"

Input device 초기화 및 register

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int  ioboard_gpio_input_init (struct ioboard_gpio *ioboard_gpio)
{
    if(!(ioboard_gpio->input = input_allocate_device()))  return  -1;

    ioboard_gpio->input->name = IOBOARD_DRIVER_NAME;

    set_bit(EV_KEY, ioboard_gpio->input->evbit);
    set_bit(KEY_CODE & KEY_MAX, ioboard_gpio->input->keybit);

    if(input_register_device(ioboard_gpio->input))    return  -1;

    return  0;
}
...
```

Input driver memory 할당

Input driver 초기화

Input driver Register

KEY_CODE = KEY_A

Scan Timer 초기화

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static enum hrtimer_restart ioboard_gpio_scan_timer(struct hrtimer *timer)
{
...
}

static int ioboard_gpio_irq_init (struct ioboard_gpio *ioboard_gpio)
{
    hrtimer_init(&ioboard_gpio->scan_timer, CLOCK_MONOTONIC,
                 HRTIMER_MODE_REL);
    ioboard_gpio->scan_timer.function = ioboard_gpio_scan_timer;
    return  0;
}

...
```

Timer callback 함수 등록

Timer IRQ를 사용한 GPIO Key Driver(4)

Scan Timer enable/disable

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_irq_enable(struct ioboard_gpio *ioboard_gpio, bool enable)
{
    if(ioboard_gpio->enable != enable)    {
        ioboard_gpio->enable = enable;

        if(enable)  {
            hrtimer_start(&ioboard_gpio->scan_timer,
                          ktime_set(0, SCAN_TIMER_PERIOD),
                          HRTIMER_MODE_REL);
        }
        else    {
            hrtimer_cancel(&ioboard_gpio->scan_timer);
        }
    }
    return  0;
}
...
```

SCAN_TIMER_PERIOD = 100000000 (100 msec)

Timer start

Timer stop

Scan Timer callback

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static enum hrtimer_restart ioboard_gpio_scan_timer(struct hrtimer *timer)
{
    struct ioboard_gpio *ioboard_gpio =
                        container_of(timer, struct ioboard_gpio, scan_timer);
    static  bool    old_status = KEY_RELEASE;
            bool    new_status;

    new_status =
      (gpio_get_value(KEY_GPIO) == 0) ? KEY_PRESS : KEY_RELEASE;

    if(old_status != new_status)    {
        old_status = new_status;

        if(new_status == KEY_PRESS) {
            input_report_key(ioboard_gpio->input, KEY_CODE, KEY_PRESS);
        }
        else    {
            input_report_key(ioboard_gpio->input, KEY_CODE, KEY_RELEASE);
        }
        input_sync(ioboard_gpio->input);
    }

    ioboard_gpio->enable = false;
    ioboard_gpio_irq_enable(ioboard_gpio, true);

    return HRTIMER_NORESTART;
}
...
```

GPIO Status Check (0 = Press, 1 = Release)

KEY_CODE = KEY_A
KEY_PRESS = 1
KEY_REPEASE = 0

KEY Data Report

Timer Restart

Timer IRQ를 사용한 GPIO Key Driver(5)

Driver 동작 검증 Snapshot

Long Press시에 다른 Key를 Report(1)

Platform driver struct 수정

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
struct ioboard_gpio  {
    struct  input_dev   *input;
    struct  hrtimer      scan_timer;
    bool                 enable;
    bool                 long_key;
    struct  hrtimer      long_timer;
};
...
static int ioboard_gpio_probe (struct platform_device *pdev)
{
    struct ioboard_gpio *ioboard_gpio;

    if(!(ioboard_gpio = kzalloc(sizeof(struct ioboard_gpio), GFP_KERNEL)))
        return -ENOMEM;

    dev_set_drvdata(&pdev->dev, ioboard_gpio);

    if(ioboard_gpio_port_init(ioboard_gpio))            goto err_gpio_init;

    if(ioboard_gpio_input_init(ioboard_gpio))           goto err_input_init;

    if(ioboard_gpio_irq_init(ioboard_gpio))             goto err_irq_init;

    if(ioboard_gpio_irq_enable(ioboard_gpio, true))     goto err_driver_init;

    return 0;
...
}
...
```

Long Press 감지 Flag

Long Press 감지 Timer 선언

Platform driver 설정값 추가

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.h

```
#define    IOBOARD_DRIVER_NAME     "ioboard-gpio"

#define    KEY_PRESS       1
#define    KEY_RELEASE     0

#define    SCAN_TIMER_PERIOD        100000000

#define    LONG_TIMER_PREIOD        3

#define    KEY_GPIO                 EXYNOS4_GPX1(1)
#define    KEY_GPIO_NAME            "ioboard key"

#define    KEY_CODE                 KEY_A
#define    LONG_KEY_CODE            KEY_B
```

Report 되어질 Key값 설정
Kernel/include/linux/input.h
파일의 KEY Code 참조

Long Timer 주기 설정
(sec 값으로 설정)

Long Press시에 다른 Key를 Report(2)

Report Key 추가

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int  ioboard_gpio_input_init (struct ioboard_gpio *ioboard_gpio)
{
    if(!(ioboard_gpio->input = input_allocate_device()))  return  -1;

    ioboard_gpio->input->name = IOBOARD_DRIVER_NAME;

    set_bit(EV_KEY, ioboard_gpio->input->evbit);
    set_bit(KEY_CODE & KEY_MAX, ioboard_gpio->input->keybit);

    set_bit(LONG_KEY_CODE & KEY_MAX, ioboard_gpio->input->keybit);

    if(input_register_device(ioboard_gpio->input))   return  -1;

    return  0;
}
...
```

KEY_CODE = KEY_A
LONG_KEY_CODE = KEY_B

Long Key 감지시
전달될 Event Enable

Long Timer 초기화

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static enum hrtimer_restart ioboard_gpio_long_timer(struct hrtimer *timer)
{
...
}
...
static int ioboard_gpio_irq_init (struct ioboard_gpio *ioboard_gpio)
{
    hrtimer_init(&ioboard_gpio->long_timer, CLOCK_MONOTONIC,
                 HRTIMER_MODE_REL);
    ioboard_gpio->scan_timer.function = ioboard_gpio_long_timer;

    hrtimer_init(&ioboard_gpio->long_timer, CLOCK_MONOTONIC,
                 HRTIMER_MODE_REL);
    ioboard_gpio->scan_timer.function = ioboard_gpio_long_timer;
    return  0;
}
...
```

Long Timer callback 함수 등록

Long Press시에 다른 Key를 Report(3)

Long timer enable/disable

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_long_timer_enable
                        (struct ioboard_gpio *ioboard_gpio, bool enable)
{
    ioboard_gpio->long_key = false;

    if(enable)  {
        hrtimer_start(&ioboard_gpio->long_timer,
                      ktime_set(LONG_TIMER_PERIOD, 0),
                      HRTIMER_MODE_REL);
    }
    else    {
        hrtimer_cancel(&ioboard_gpio->long_timer);
    }

    return  0;
}
...
static enum hrtimer_restart ioboard_gpio_long_timer(struct hrtimer *timer)
{
    struct ioboard_gpio  *ioboard_gpio =
                        container_of(timer, struct ioboard_gpio, long_timer);

    ioboard_gpio->long_key = true;

    return HRTIMER_NORESTART;
}
...
```

Long Key Flag 초기화

Timer start

LONG_TIMER_PERIOD = 3 (3 sec)

Timer stop

Long Key Flag Set

Scan timer callback function 수정

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static enum hrtimer_restart ioboard_gpio_scan_timer(struct hrtimer *timer)
{
    struct ioboard_gpio *ioboard_gpio =
                        container_of(timer, struct ioboard_gpio, scan_timer);
    static  bool    old_status = KEY_RELEASE;
            bool    new_status;
    new_status =
      (gpio_get_value(KEY_GPIO) == 0) ? KEY_PRESS : KEY_RELEASE;

    if(old_status != new_status)    {
        old_status = new_status;

        if(new_status == KEY_PRESS) {
            ioboard_gpio_long_timer_enable(ioboard_gpio, true);
        }
        else    {
            if(ioboard_gpio->long_key)  {
                input_report_key(ioboard_gpio->input,
                                 LONG_KEY_CODE, KEY_PRESS);
                input_report_key(ioboard_gpio->input,
                                 LONG_KEY_CODE, KEY_RELEASE);
            }   else    {
                ioboard_gpio_long_timer_enable(ioboard_gpio, false);
                input_report_key(ioboard_gpio->input, KEY_CODE, KEY_PRESS);
                input_report_key(ioboard_gpio->input, KEY_CODE, KEY_RELEASE);
            }
        }
        input_sync(ioboard_gpio->input);
    }
    ioboard_gpio->enable = false;
    ioboard_gpio_irq_enable(ioboard_gpio, true);

    return HRTIMER_NORESTART;
}
...
```

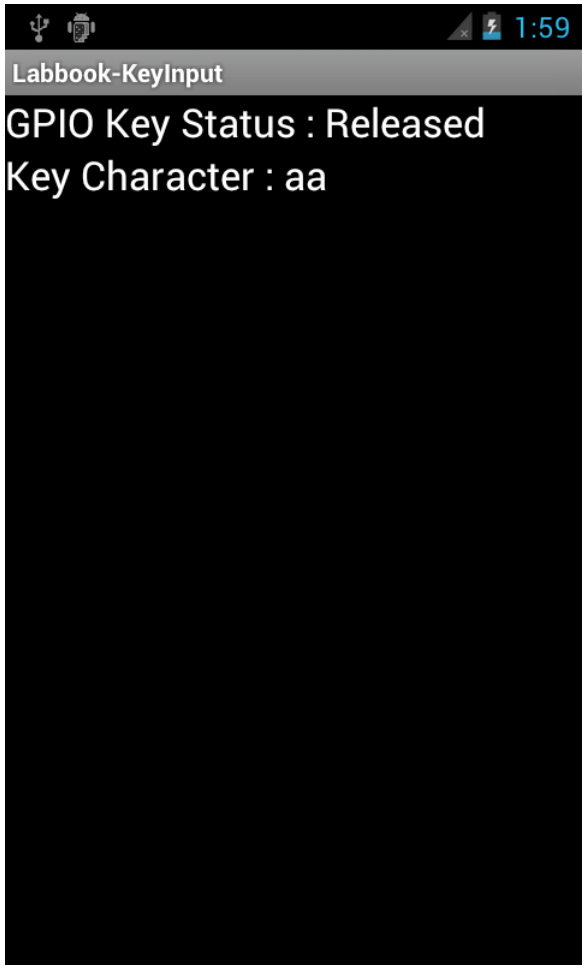
Long Press Keycode Report

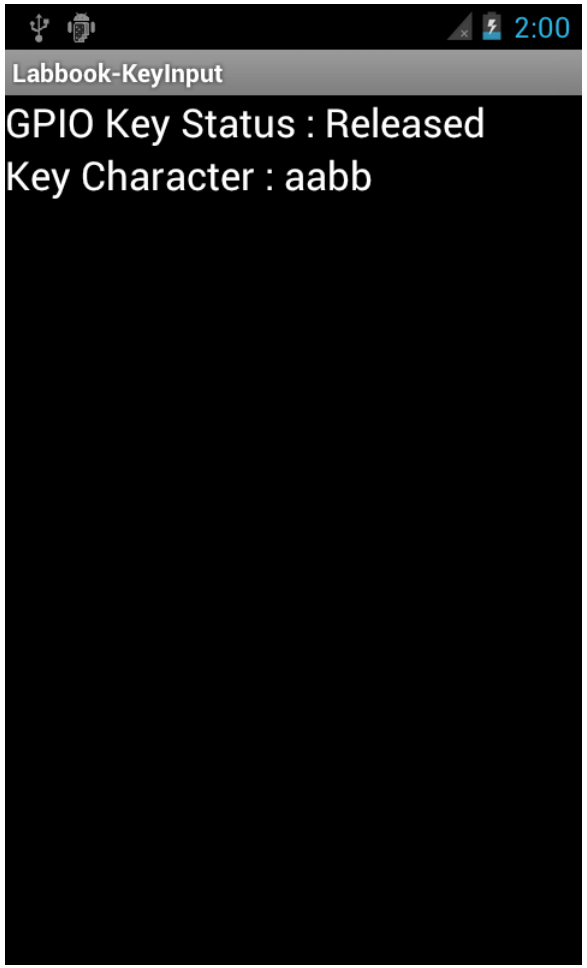
Long Timer Stop Short Key report

KEY_CODE = KEY_A
LONG_KEY_CODE = KEY_B

Long Press시에 다른 Key를 Report(4)

Driver 동작 검증 Snapshot





Platform Data Struct(1)

Platform Data Struct 구성

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.h

```
#define    IOBOARD_DRIVER_NAME     "ioboard-gpio"

#define    KEY_PRESS       1
#define    KEY_RELEASE     0

struct ioboard_gpio_pdata   {

    int     gpio;
    char    *gpio_name;

    int     keycode;

    int     scan_interval;

    int     long_keycode;

    int     long_interval;

};
```

Control 할 GPIO 설정값

Report Data 설정값

Scan Timer 주기 설정값

Long Press Report Data 설정값

Long Timer 주기 설정값

Platform Data Struct 초기화 및 등록

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
#include <../../../drivers/misc/odroida4-io/gpio/ioboard-gpio.h>

struct ioboard_gpio_pdata   ioboard_gpio_pdata = {

    .gpio           = EXYNOS4_GPX1(1),
    .gpio_name      = "ioboard key",

    .keycode        = KEY_C,

    .scan_interval  = 200,      // msec unit

    .long_keycode = KEY_D,
    .long_interval  = 5000,     // msec unit
};

static struct           platform_device     ioboard_gpio_device = {
    .name               = IOBOARD_DRIVER_NAME,
    .id                 = -1,

    .dev.platform_data = &ioboard_gpio_pdata,
};
...
```

Platform Data Struct(2)

Platform Driver에서 Platform Data 참조

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
struct ioboard_gpio  {
    struct  input_dev   *input;
    struct  hrtimer      scan_timer;
    bool                 enable;
    bool                 long_key;
    struct  hrtimer      long_timer;

    struct  ioboard_gpio_pdata *pdata;
};
...
static int ioboard_gpio_probe (struct platform_device *pdev)
{
    struct ioboard_gpio *ioboard_gpio;

    if(!(ioboard_gpio = kzalloc(sizeof(struct ioboard_gpio), GFP_KERNEL)))
        return -ENOMEM;

    ioboard_gpio->pdata = pdev->dev.platform_data;

    dev_set_drvdata(&pdev->dev, ioboard_gpio);

    if(ioboard_gpio_port_init(ioboard_gpio))          goto err_gpio_init;

    if(ioboard_gpio_input_init(ioboard_gpio))         goto err_input_init;

    if(ioboard_gpio_irq_init(ioboard_gpio))           goto err_irq_init;

    if(ioboard_gpio_irq_enable(ioboard_gpio, true))   goto err_driver_init;

    return 0;
...
}
...
```

설정된 Platform data를 참조하기 위한 pointer 선언

Platform driver로 Platform data의 전달

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
#include <../../../drivers/misc/odroida4-io/gpio/ioboard-gpio.h>

struct ioboard_gpio_pdata   ioboard_gpio_pdata = {

    .gpio             = EXYNOS4_GPX1(1),
    .gpio_name     = "ioboard key",

    .keycode          = KEY_C,

    .scan_interval  = 100,       // msec unit

    .long_keycode = KEY_D,
    .long_interval  = 3000,     // msec unit
};

static struct          platform_device      ioboard_gpio_device = {
    .name                  = IOBOARD_DRIVER_NAME,
    .id                    = -1,

    .dev.platform_data = &ioboard_gpio_pdata,
};
...
```

Platform Data Struct(3)

GPIO request 및 초기화

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_port_init (struct ioboard_gpio *ioboard_gpio)
{
    if(gpio_request(ioboard_gpio->pdata->gpio,
                    ioboard_gpio->pdata->gpio_name))    return  -1;

    gpio_direction_input(ioboard_gpio->pdata->gpio);

    return  0;
}
...
```

Platform data의
설정값으로 초기화

Input device 초기화 및 register

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int  ioboard_gpio_input_init (struct ioboard_gpio *ioboard_gpio)
{
    if(!(ioboard_gpio->input = input_allocate_device()))  return  -1;

    ioboard_gpio->input->name = IOBOARD_DRIVER_NAME;

    set_bit(EV_KEY, ioboard_gpio->input->evbit);

    set_bit(ioboard_gpio->pdata->keycode & KEY_MAX,
            ioboard_gpio->input->keybit);

    set_bit(ioboard_gpio->pdata->long_keycode & KEY_MAX,
            ioboard_gpio->input->keybit);

    if(input_register_device(ioboard_gpio->input))   return  -1;

    return  0;
}
...
```

Platform Data Struct(4)

Scan Timer enable/disable

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_irq_enable(struct ioboard_gpio *ioboard_gpio, bool enable)
{
    if(ioboard_gpio->enable != enable)    {
        ioboard_gpio->enable = enable;

        if(enable)  {
            hrtimer_start(&ioboard_gpio->scan_timer,
                          ktime_set((ioboard_gpio->pdata->scan_interval / 1000), ((ioboard_gpio->pdata->scan_interval % 1000) * 1000000)),
                          HRTIMER_MODE_REL);
        }    else    {
            hrtimer_cancel(&ioboard_gpio->scan_timer);
        }
    }
    return  0;
}
...
```

Long Timer enable/disable

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_long_timer_enable
                              (struct ioboard_gpio *ioboard_gpio, bool enable)
{
    ioboard_gpio->long_key = false;

    if(enable)  {
        hrtimer_start(&ioboard_gpio->long_timer,
                      ktime_set((ioboard_gpio->pdata->long_interval / 1000), ((ioboard_gpio->pdata->long_interval % 1000) * 1000000)),
                      HRTIMER_MODE_REL);
    }    else    {
        hrtimer_cancel(&ioboard_gpio->long_timer);
    }
    return  0;
}
...
```

Platform data의
설정값으로 Timer 초기화

Platform Data Struct(5)

Scan Timer Callback Function

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

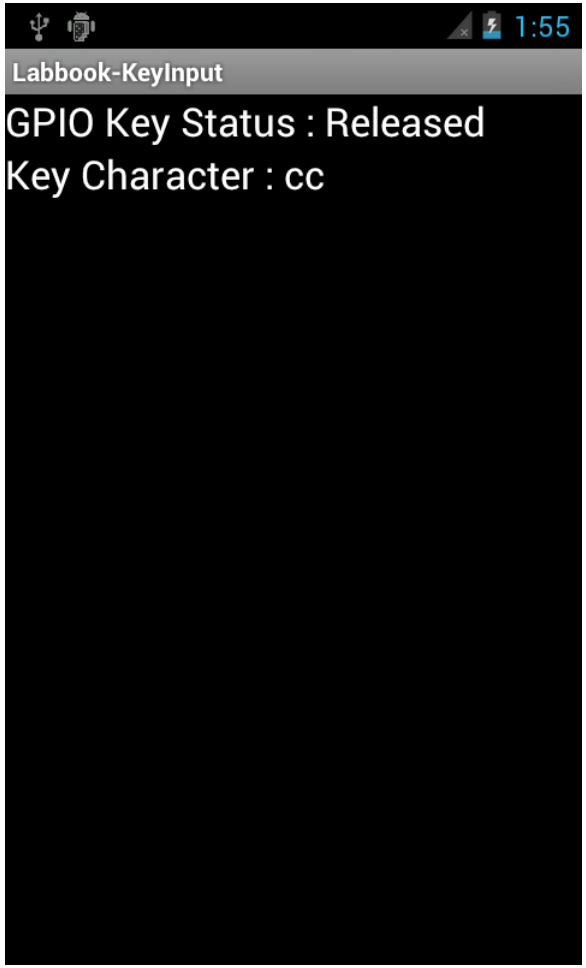
```
...
static enum hrtimer_restart ioboard_gpio_scan_timer(struct hrtimer *timer)
{
    struct ioboard_gpio *ioboard_gpio =
                        container_of(timer, struct ioboard_gpio, scan_timer);
    static  bool    old_status = KEY_RELEASE;
            bool    new_status;

    new_status = (gpio_get_value(ioboard_gpio->pdata->gpio) == 0) ? KEY_PRESS : KEY_RELEASE;

    if(old_status != new_status)    {
        old_status = new_status;

        if(new_status == KEY_PRESS) {
            ioboard_gpio_long_timer_enable(ioboard_gpio, true);
        }
        else    {
            if(ioboard_gpio->long_key)  {
                input_report_key(ioboard_gpio->input, ioboard_gpio->pdata->long_keycode, KEY_PRESS);
                input_report_key(ioboard_gpio->input, ioboard_gpio->pdata->long_keycode, KEY_RELEASE);
            }   else    {
                ioboard_gpio_long_timer_enable(ioboard_gpio, false);
                input_report_key(ioboard_gpio->input, ioboard_gpio->pdata->keycode, KEY_PRESS);
                input_report_key(ioboard_gpio->input, ioboard_gpio->pdata->keycode, KEY_RELEASE);
            }
        }
        input_sync(ioboard_gpio->input);
    }
    ioboard_gpio->enable = false;
    ioboard_gpio_irq_enable(ioboard_gpio, true);

    return HRTIMER_NORESTART;
}
...
```

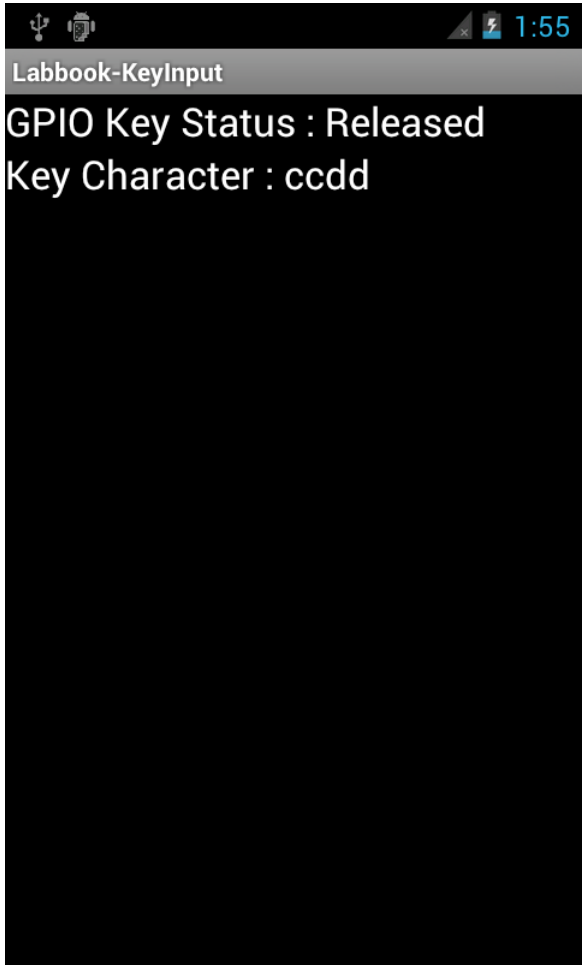
Platform data로 설정된 GPIO Port check

Platform data로 설정된 Keycode report

Platform Data Struct(6)

Driver 동작 검증 Snapshot





SYSFS를 추가하기(1)

SYSFS 초기화 및 Register

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static   int      ioboard_gpio_sysfs_init (struct platform_device *pdev)
{
    if(sysfs_create_group(&pdev->dev.kobj, &ioboard_gpio_attr_group)
        return -1;
    return 0;
}
...
static int ioboard_gpio_probe (struct platform_device *pdev)
{
...
    if(ioboard_gpio_sysfs_init(pdev))          goto err_sysfs_init;
...
}
...
```

SYSFS File 구성 및 DEVICE_ATTR 메크로

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static ssize_t show_enable          (struct device *dev, struct device_attribute *attr, char *buf);
static ssize_t store_enable         (struct device *dev, struct device_attribute *attr,
                                     const char *buf, size_t count);

static ssize_t show_scan_interval (struct device *dev, struct device_attribute *attr, char *buf);
static ssize_t store_scan_interval (struct device *dev, struct device_attribute *attr,
                                     const char *buf, size_t count);
static ssize_t show_long_interval (struct device *dev, struct device_attribute *attr, char *buf);
static ssize_t store_long_interval (struct device *dev, struct device_attribute *attr,
                                     const char *buf, size_t count);
```

Sysfs File Permission / File rwrite function

```
static DEVICE_ATTR( enable,      S_IRWXUGO,   show_enable,      store_enable);
```

Sysfs File name / File read function

```
static DEVICE_ATTR(scan_interval, S_IRWXUGO, show_scan_interval,   store_scan_interval);
static DEVICE_ATTR(long_interval, S_IRWXUGO, show_long_interval,   store_long_interval);

static struct attribute *ioboard_gpio_entries[] = {

    &dev_attr_enable.attr,

    &dev_attr_scan_interval.attr,
    &dev_attr_long_interval.attr,
    NULL
};

static struct attribute_group ioboard_gpio_attr_group = {

    .name = NULL,
    .attrs  = ioboard_gpio_entries,
};
...
```

SYSFS를 추가하기(2)

Platform Driver에서 사용하는 Data Struct 공유

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_probe (struct platform_device *pdev)
{
    struct ioboard_gpio *ioboard_gpio;

    if(!(ioboard_gpio = kzalloc(sizeof(struct ioboard_gpio), GFP_KERNEL)))
        return -ENOMEM;

    ioboard_gpio->pdata = pdev->dev.platform_data;

    dev_set_drvdata(&pdev->dev, ioboard_gpio);

    if(ioboard_gpio_port_init(ioboard_gpio))          goto err_gpio_init;

    if(ioboard_gpio_input_init(ioboard_gpio))         goto err_input_init;

    if(ioboard_gpio_irq_init(ioboard_gpio))           goto err_irq_init;

    if(ioboard_gpio_irq_enable(ioboard_gpio, true))   goto err_driver_init;

    return 0;
...
}
...
```

Scan Timer Enable/Disable Control Sysfs

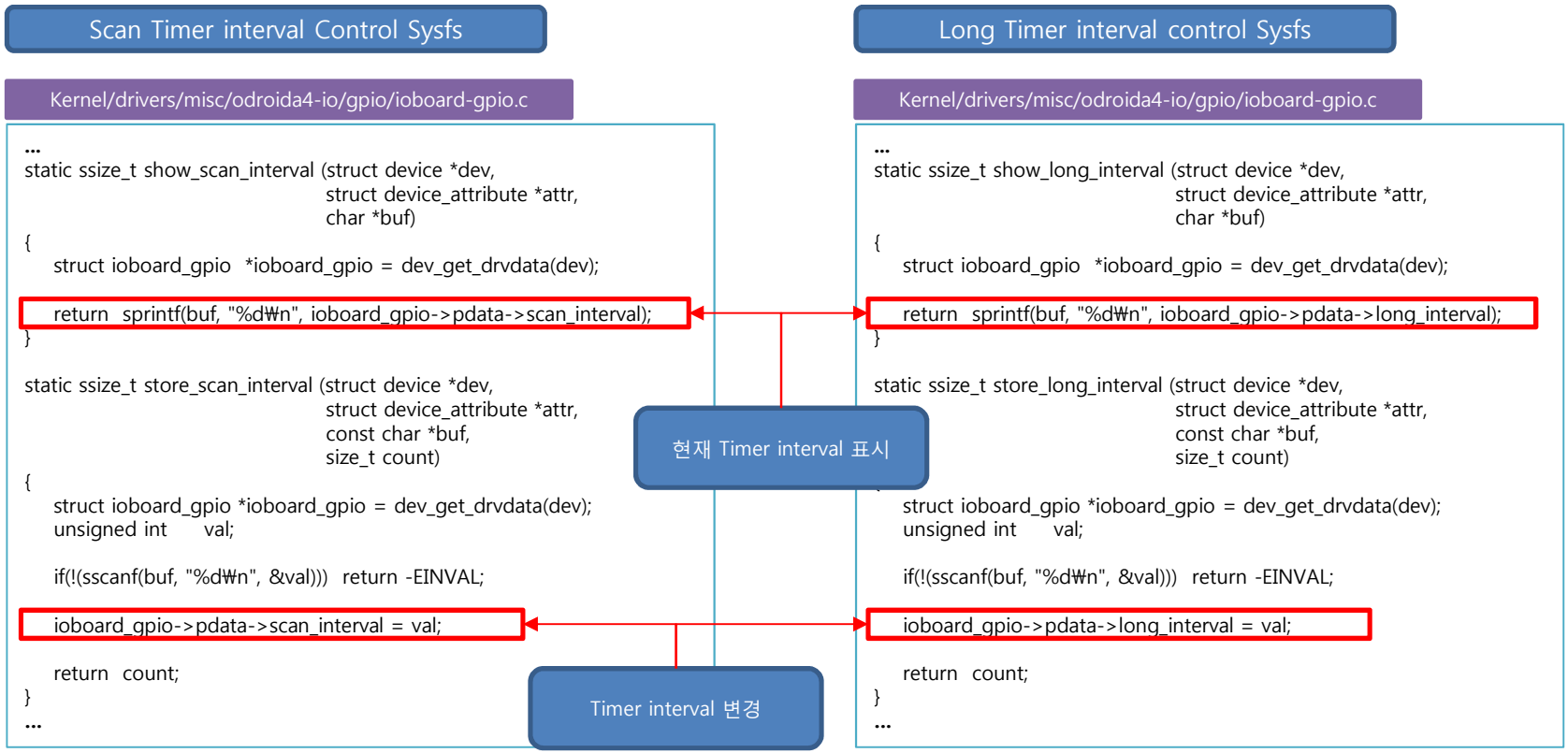
Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static ssize_t show_enable (struct device *dev,
                            struct device_attribute *attr,
                            char *buf)
{
    struct ioboard_gpio  *ioboard_gpio = dev_get_drvdata(dev);

    return  sprintf(buf, "%d₩n", ioboard_gpio->enable);
}

static ssize_t store_enable(struct device *dev,
                            struct device_attribute *attr,
                            const char *buf,
                            size_t count)
{
    struct ioboard_gpio  *ioboard_gpio = dev_get_drvdata(dev);

    unsigned int    val;

    if(!(sscanf(buf, "%d₩n", &val)))      return -EINVAL;

    ioboard_gpio_irq_enable(ioboard_gpio, (val != 0) ? true : false);

    return count;
}
...
```

현재 Scan Timer 상태표시

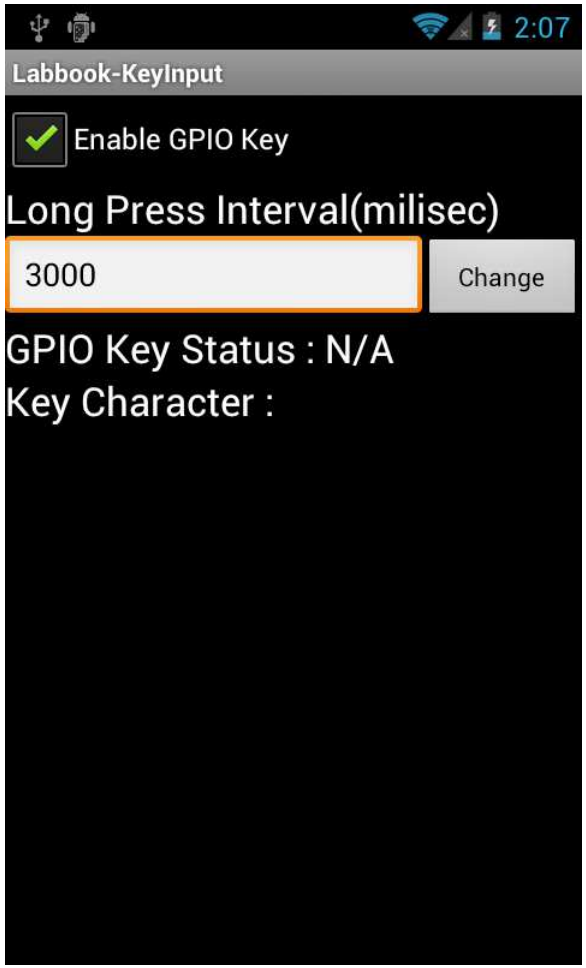
Scan Timer
Enable/Disable control

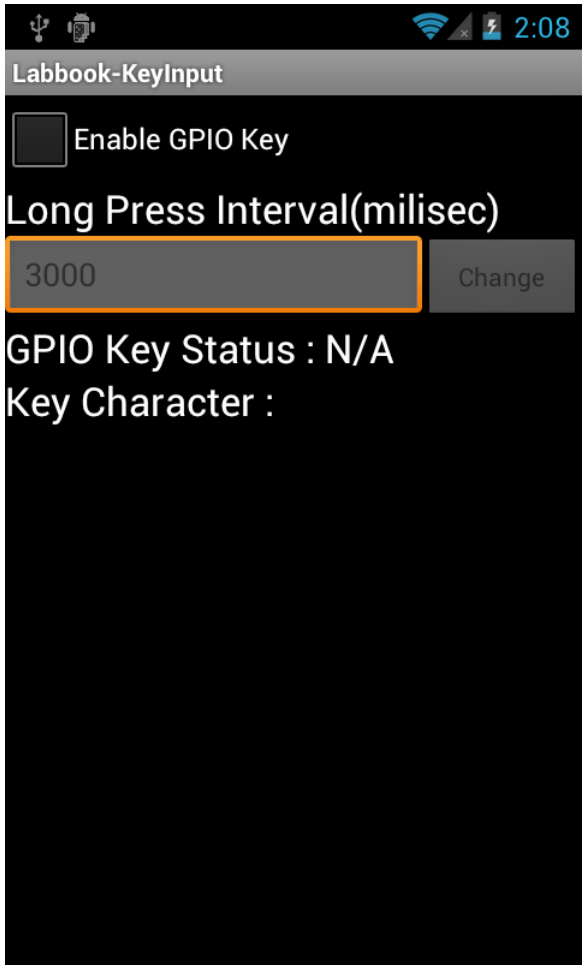
SYSFS를 추가하기(3)



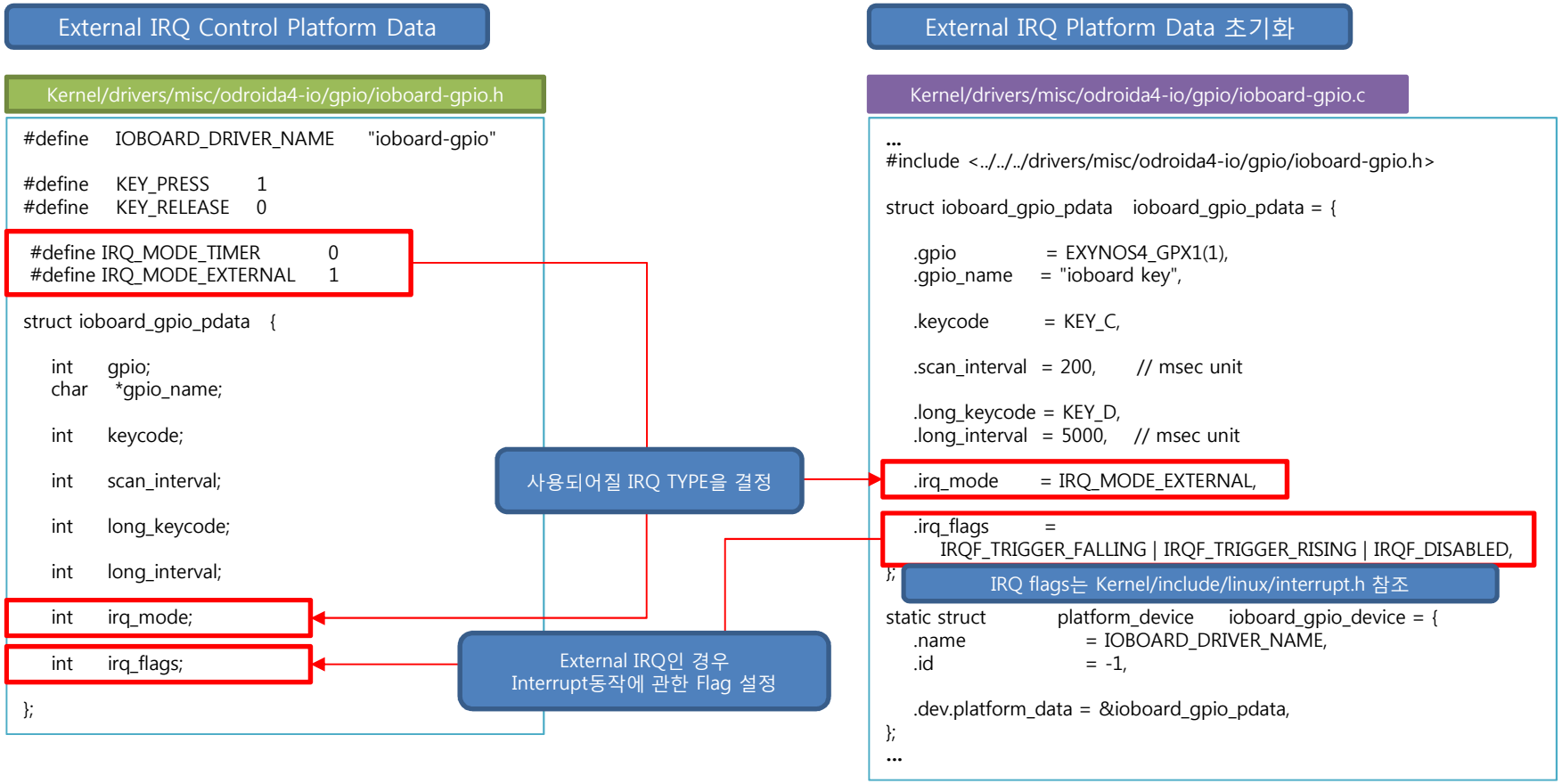
SYSFS를 추가하기(4)

Driver 동작 검증 Snapshot





External IRQ를 사용할 수 있도록 드라이버 수정(1)



External IRQ를 사용할 수 있도록 드라이버 수정(2)

External IRQ Register

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
static  irqreturn_t   ioboard_gpio_irq    (int irq, void *handle)
{
...
}
...
static  int     ioboard_gpio_irq_init   (struct ioboard_gpio *ioboard_gpio)
{
    switch(ioboard_gpio->pdata->irq_mode)
    {
        default :
        case    IRQ_MODE_TIMER:
            hrtimer_init(&ioboard_gpio->scan_timer,
                         CLOCK_MONOTONIC, HRTIMER_MODE_REL);
            ioboard_gpio->scan_timer.function = ioboard_gpio_scan_timer;
        break;

        case    IRQ_MODE_EXTERNAL:
            if(request_irq(gpio_to_irq( ioboard_gpio->pdata->gpio),
                                        ioboard_gpio_irq,
                                        ioboard_gpio->pdata->irq_flags,
                                        ioboard_gpio->pdata->gpio_name,
                                        ioboard_gpio))      return  -1;
            disable_irq_nosync(gpio_to_irq(ioboard_gpio->pdata->gpio));
        break;
    }

    hrtimer_init(&ioboard_gpio->long_timer,
                 CLOCK_MONOTONIC, HRTIMER_MODE_REL);
    ioboard_gpio->long_timer.function = ioboard_gpio_long_timer;

    return  0;
}
...
```

External IRQ Callback 함수 등록

GPIO IRQ를 Platform Data에 설정한 irq_flag 값으로 등록

External IRQ Enable/Disable

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static int ioboard_gpio_irq_enable(struct ioboard_gpio *ioboard_gpio, bool enable)
{
    if(ioboard_gpio->enable != enable)  {

        ioboard_gpio->enable = enable;

        switch(ioboard_gpio->pdata->irq_mode)
        {
            default :
            case    IRQ_MODE_TIMER:
                if(enable)  {
                    if(ioboard_gpio->pdata->scan_interval)  {
                        hrtimer_start(&ioboard_gpio->scan_timer,
                                ktime_set((ioboard_gpio->pdata->scan_interval / 1000),
                                ((ioboard_gpio->pdata->scan_interval % 1000) * 1000000)),
                                HRTIMER_MODE_REL);
                    }
                    else    {
                        hrtimer_cancel(&ioboard_gpio->scan_timer);
                    }

                break;

            case    IRQ_MODE_EXTERNAL:
                if(enable)  enable_irq (gpio_to_irq(ioboard_gpio->pdata->gpio));
                else        disable_irq(gpio_to_irq(ioboard_gpio->pdata->gpio));
            break;
        }
    }
    return  0;
}
...
```

External IRQ Enable/Disable

External IRQ를 사용할 수 있도록 드라이버 수정(3)

External IRQ Callback Function 구조

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static  irqreturn_t   ioboard_gpio_irq    (int irq, void *handle)
{
    struct  ioboard_gpio *ioboard_gpio = handle;

    static  bool    old_status = KEY_RELEASE;
            bool    new_status;

    new_status =
        (gpio_get_value(ioboard_gpio->pdata->gpio) == 0) ? KEY_PRESS : KEY_RELEASE;

    if(old_status != new_status)    {
        old_status = new_status;

        if(new_status == KEY_PRESS) {
            ioboard_gpio_long_timer_enable(ioboard_gpio, true);
        }
        else    {
            if(ioboard_gpio->long_key)  {
                input_report_key(ioboard_gpio->input,
                                 ioboard_gpio->pdata->long_keycode, KEY_PRESS);
                input_report_key(ioboard_gpio->input,
                                 ioboard_gpio->pdata->long_keycode, KEY_RELEASE);
            }   else    {
                ioboard_gpio_long_timer_enable(ioboard_gpio, false);
                input_report_key(ioboard_gpio->input,
                                 ioboard_gpio->pdata->keycode, KEY_PRESS);
                input_report_key(ioboard_gpio->input,
                                 ioboard_gpio->pdata->keycode, KEY_RELEASE);
            }
        }
        input_sync(ioboard_gpio->input);
    }
    return IRQ_HANDLED;
}
...
```

Platform Driver Struct 전달

Kernel/drivers/misc/odroida4-io/gpio/ioboard-gpio.c

```
...
static  int     ioboard_gpio_irq_init   (struct ioboard_gpio *ioboard_gpio)
{
    switch(ioboard_gpio->pdata->irq_mode)
    {
        default :
        case    IRQ_MODE_TIMER:
            hrtimer_init(&ioboard_gpio->scan_timer,
                         CLOCK_MONOTONIC, HRTIMER_MODE_REL);
            ioboard_gpio->scan_timer.function = ioboard_gpio_scan_timer;
        break;

        case    IRQ_MODE_EXTERNAL:
            if(request_irq(gpio_to_irq( ioboard_gpio->pdata->gpio),
                                        ioboard_gpio_irq,
                                        ioboard_gpio->pdata->irq_flags,
                                        ioboard_gpio->pdata->gpio_name,
                                        ioboard_gpio))    return  -1;
            disable_irq_nosync(gpio_to_irq(ioboard_gpio->pdata->gpio));
        break;
    }
    hrtimer_init(&ioboard_gpio->long_timer,
                 CLOCK_MONOTONIC, HRTIMER_MODE_REL);
    ioboard_gpio->long_timer.function = ioboard_gpio_long_timer;

    return  0;
}
...
static int ioboard_gpio_probe (struct platform_device *pdev)
{
...
    if(ioboard_gpio->pdata->irq_mode != IRQ_MODE_EXTERNAL)
        if(ioboard_gpio_port_init(ioboard_gpio))        goto err_gpio_init;
...
}
...
```

IRQ Mode에서는 GPIO를 Control하지 않도록 한다.

Android Application : labbook-KeyInput(1)

- **JNI function을 만든다.**

```
static {
        System.loadLibrary("KeyDriverSysfs");
}
```

```
native int checkNode();                    //sysfs 노드가 있는지 확인
native int getState();                     //enable 상태 확인
native void setState(int enable);          //enable/disable 변경
native int readInterval();                 //interval 값 확인
native void setInterval(String value);     //interval 값 변경
```

Android Application : labbook-KeyInput(2)


```
if (checkNode() == 0) { //sysfs node를 확인하여 UI를 그린다.
                mKeyDriverEnable.setVisibility(View.GONE);
                mLongPressIntervalText.setVisibility(View.GONE);
                mInterval.setVisibility(View.GONE);
                mChangeInterval.setVisibility(View.GONE);
} else {
                mKeyDriverEnable.setVisibility(View.VISIBLE);
                mLongPressIntervalText.setVisibility(View.VISIBLE);
                mInterval.setVisibility(View.VISIBLE);
                mChangeInterval.setVisibility(View.VISIBLE);

                mInterval.setText(readInterval() + "");          //interval을 jni로부터 읽어서 표시한다.
}
```

Android Application : labbook-KeyInput(3)


```
    int state = getState();            //enable jni로 부터 읽어 상태 확인
    if (state == 1) {
            mKeyDriverEnable.setChecked(true);
            update(true);
    } else {
            mKeyDriverEnable.setChecked(false);
            update(false);
    }

private void update(boolean enable)  {        //controller 상태 업데이트
            if (enable)  {
                        mInterval.setEnabled(true);
                        mChangeInterval.setEnabled(true);
            } else {
                        mInterval.setEnabled(false);
                        mChangeInterval.setEnabled(false);
            }
}
```

Android Application : labbook-KeyInput(4)

- **두개의 function을 override하여 상태와 KEYCODE를 출력한다.**

public boolean onKeyDown(int keyCode, KeyEvent event)

public boolean onKeyUp(int keyCode, KeyEvent event)

```
mKeyStatus.setText("Released“/[“Press”]);
if (keyCode == KeyEvent.KEYCODE_A)
        mKeyCharacter.append("a");
else if (keyCode == KeyEvent.KEYCODE_B)
        mKeyCharacter.append("b");
else if(keyCode == KeyEvent.KEYCODE_C)
        mKeyCharacter.append("c");
else if (keyCode == KeyEvent.KEYCODE_D)
        mKeyCharacter.append("d");
```

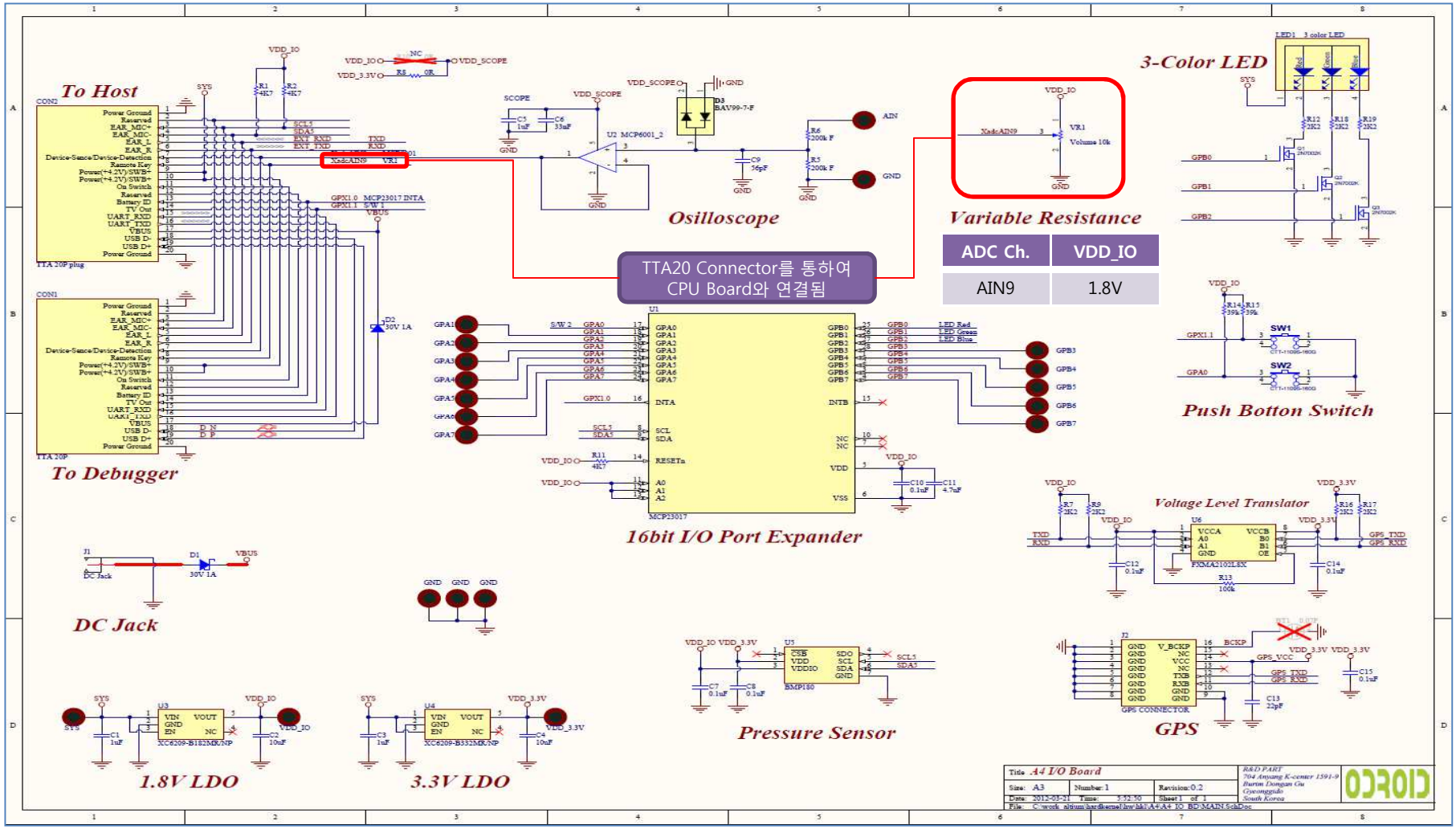
하드웨어 확장 - ADC

Agenda

- ODROID-A4 IO Board ADC Hardware Interface
- ADC Common Driver
  - Platform에서 제공하는 ADC Common Driver
  - Platform ADC Common Driver Enable
- Voltage Monitor Driver
  - ADC Common Driver를 이용한 Voltage Monitor Driver
  - Platform Data에 설정한 시간마다 Voltage 연산 구현
  - Voltage 값을 표시하는 SYSFS구성
- Android Application
  - JNI를 통하여 Voltage 값을 읽어 화면에 표시하는 APP
- Android Application : labbook-ADC

ODROID-A4 IO Board ADC Hardware Interface(1)

Odroid-A4 I/O Board의 ADC연결



ADC Ch.	VDD_IO
AIN9	1.8V

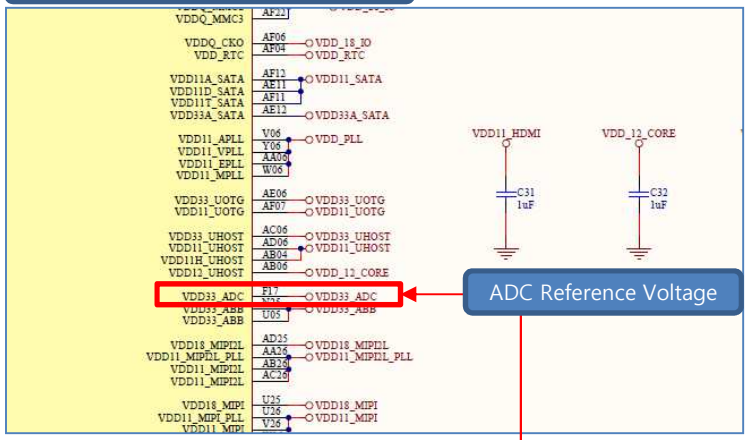
ODROID-A4 IO Board ADC Hardware Interface(2)

ADC Reference Voltage Configuration

Kernel/arch/arm/mach-exynos/pmic-8997.c

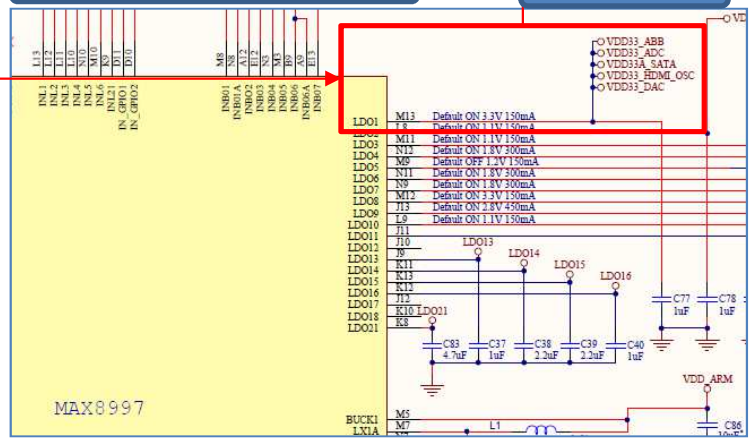
```
...
static struct regulator_init_data max8997_ldo1_data = {
    .constraints = {
        .name       = "VDD_ADC_3.3V",
        .min_uV     = 3300000,
        .max_uV     = 3300000,
        .apply_uV   = 1,
        .always_on  = 1,
        .valid_ops_mask = REGULATOR_CHANGE_STATUS,
        .state_mem = {
            .uV         = 3300000,
            .enabled    = 1,
        },
    },
//    .num_consumer_supplies   = 1,
//    .consumer_supplies   = &ldo1_consumer[0],
};
...
```

MCU_S5PC210_GEN_POWER.SchDoc



MCU_S5PC210_GEN_POWER.SchDoc

LDO1 = 3.3V



ADC Common Driver(1)

ADC Common Driver Probe

Kernel/arch/arm/plat-samsung/adc.c

```
...
static struct platform_device_id s3c_adc_driver_ids[] = {
    {
        .name          = "s3c24xx-adc",
        .driver_data   = TYPE_S3C24XX,
    }, {
        .name          = "s3c64xx-adc",
        .driver_data   = TYPE_S3C64XX,
    }, {
        .name          = "s5pv210-adc",
        .driver_data   = TYPE_S5PV210,
    }, {
        .name          = "exynos4412-adc",
        .driver_data   = TYPE_EXYNOS4412,
    },
    { }
};
MODULE_DEVICE_TABLE(platform, s3c_adc_driver_ids);

static struct platform_driver s3c_adc_driver = {

    .id_table = s3c_adc_driver_ids,

    .driver    = {
        .name    = "s3c-adc",
        .owner   = THIS_MODULE,
    },
    .probe     = s3c_adc_probe,
    .remove    = __devexit_p(s3c_adc_remove),
    .suspend   = s3c_adc_suspend,
    .resume    = s3c_adc_resume,
};
...
```

Kernel/arch/arm/mach-exynos/cpu-exynos4.c

```
...
void __init exynos4_map_io(void)
{
    if (soc_is_exynos4210())     s3c_adc_setname("s5pv210-adc");
    else                         s3c_adc_setname("exynos4412-adc");
}
...
```

Kernel/arch/arm/plat-samsung/dev-adc.c

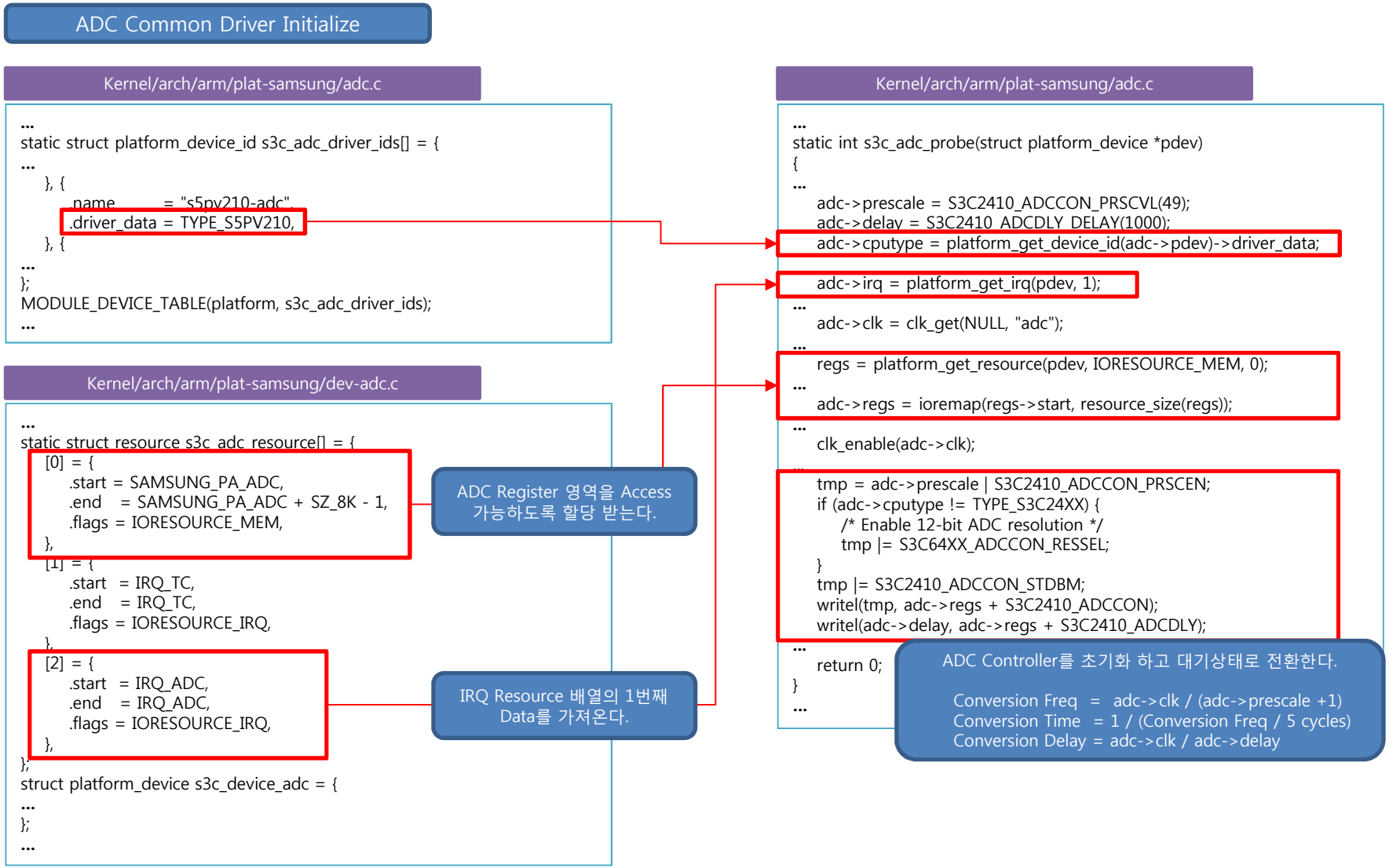
```
...
static struct resource s3c_adc_resource[] = {
...
};

struct platform_device s3c_device_adc = {
    .name          = "samsung-adc",
    .id            = -1,
    .num_resources = ARRAY_SIZE(s3c_adc_resource),
    .resource      = s3c_adc_resource,
};
...
```

System Boot Processor에서 CPU Type에 따라 ADC Device 이름을 바꾼다.

Kernel/arch/arm/mach-exynos/mach-odroid-4210.c

```
...
static struct platform_device *smdkv310_devices[] __initdata = {
...
    &s3c_device_adc,
...
};
...
```

ADC Common Driver(2)



ADC Common Driver(3)

ADC Common Driver Export Function

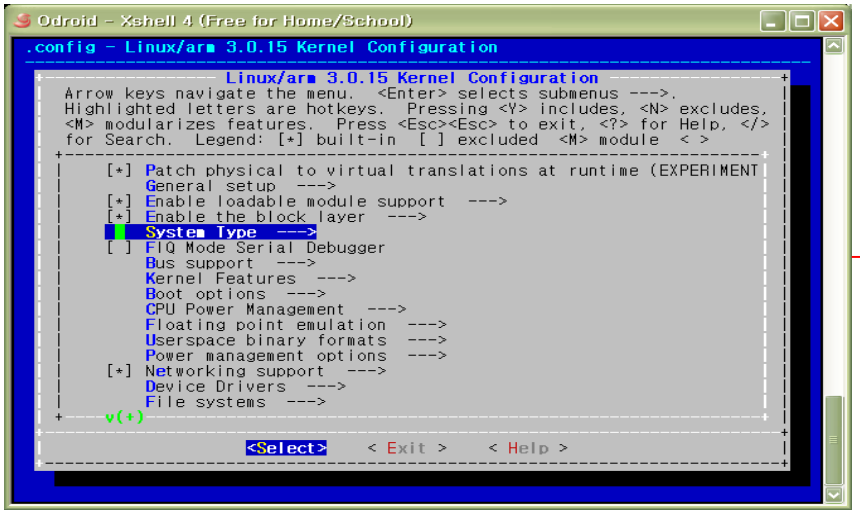
Kernel/arch/arm/plat-samsung/adc.c

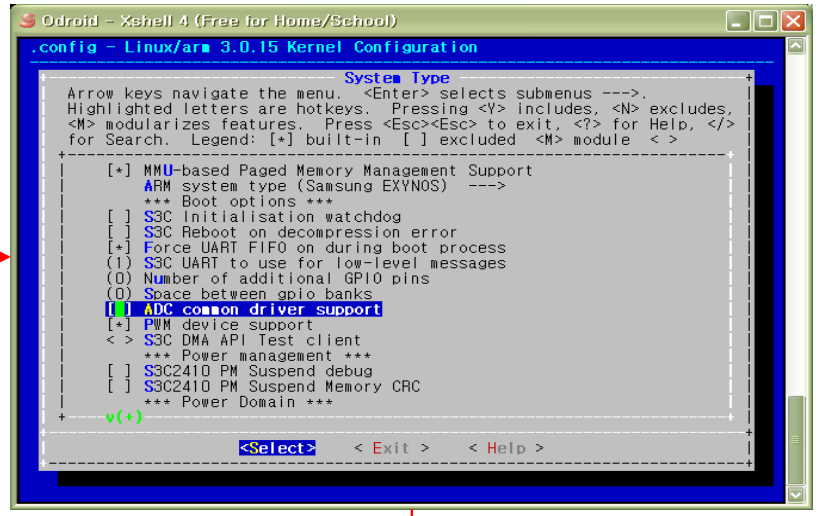
```
...
struct s3c_adc_client *s3c_adc_register(
                        struct platform_device *pdev,
                        void (*select)(struct s3c_adc_client *client,
                                unsigned int selected),
                        void (*conv)(struct s3c_adc_client *client,
                                unsigned d0, unsigned d1,
                                unsigned *samples_left),
                        unsigned int is_ts)
{
    return client;
}
EXPORT_SYMBOL_GPL(s3c_adc_register);

void s3c_adc_release(struct s3c_adc_client *client)
{
}
EXPORT_SYMBOL_GPL(s3c_adc_release);

int s3c_adc_read(struct s3c_adc_client *client, unsigned int ch)
{
    return client->result;
}
EXPORT_SYMBOL_GPL(s3c_adc_read);

int s3c_adc_start(struct s3c_adc_client *client,
                  unsigned int channel, unsigned int nr_samples)
{
    return 0;
}
EXPORT_SYMBOL_GPL(s3c_adc_start);
...
```

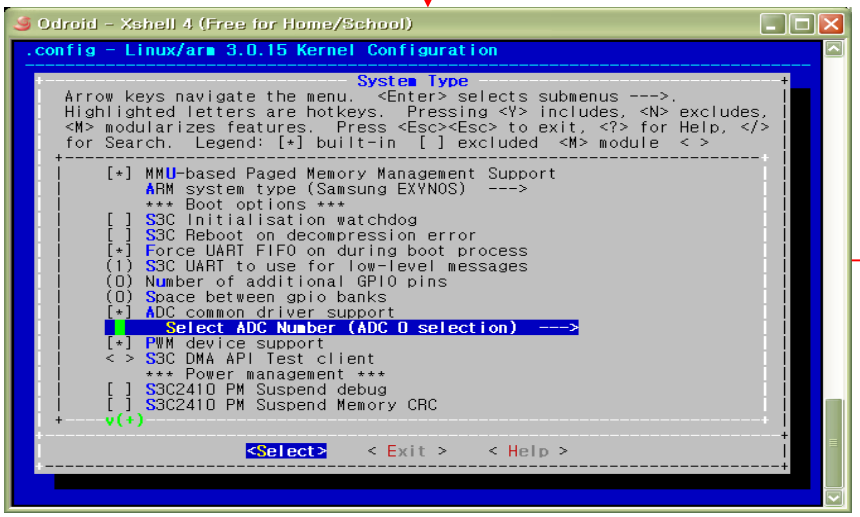
- Convert가 완료되는 경우 Interrupt에서 호출되는 함수 포인터
- ADC를 Touch screen용으로 사용하는지 구분
- ADC를 LIST에 등록하고 ADC Control 포인터를 되돌려 준다.
- 사용된 ADC를 STOP하고 ADC List에서 제거한다.
- 입력된 채널의 ADC 변환값을 되돌려 준다.
- 입력된 채널의 ADC 변환을 nr_samples 만큼 실행한다. 실행 결과는 conv , select 함수포인터에 지정된 함수로 전달된다.

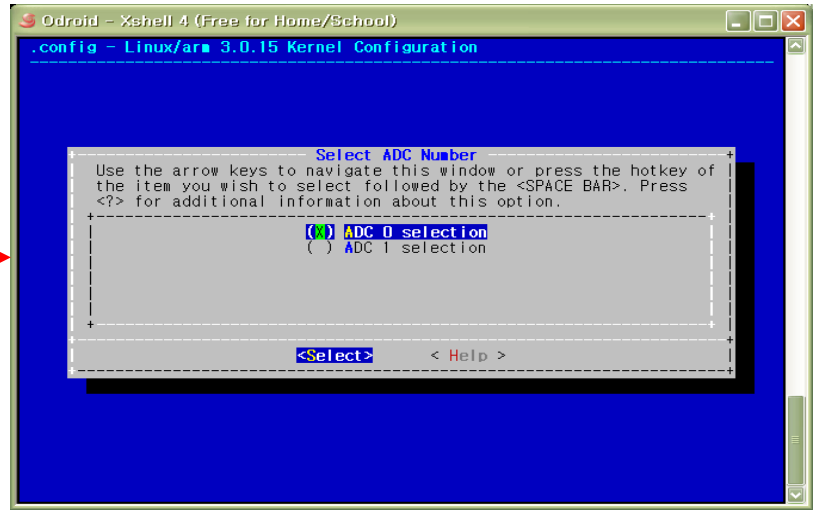
ADC Common Driver(4)

ADC Common Driver Enable

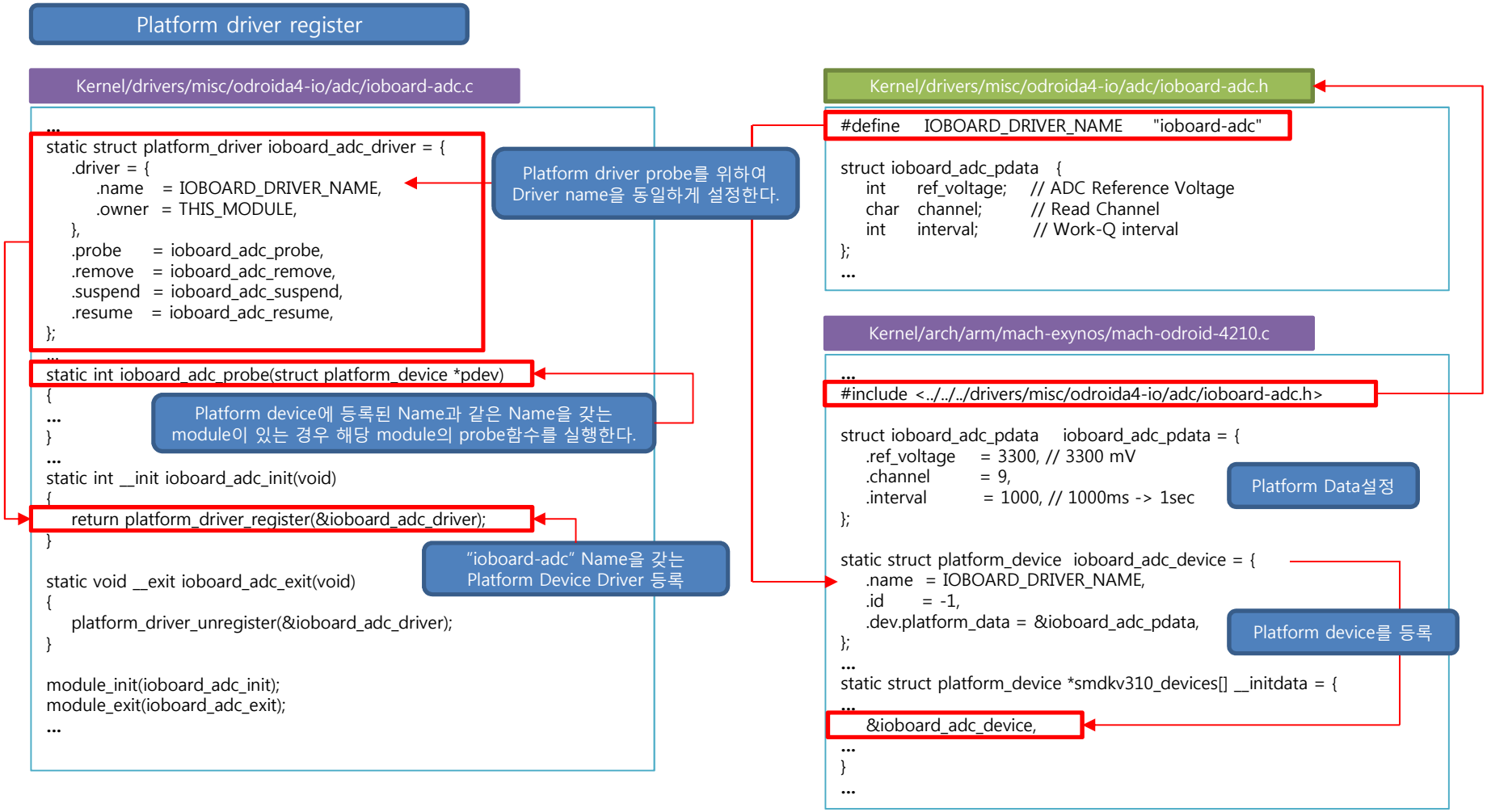








Voltage Monitor Driver(1)



Voltage Monitor Driver(2)

ADC Common Driver Register

Kernel/drivers/misc/odroida4-io/adc/ioboard-adc.c

```
#include <plat/adc.h>
...
struct ioboard_adc  {

    struct  s3c_adc_client          *client;

    struct  ioboard_adc_pdata  *pdata;
    unsigned int                    voltage;
    unsigned int                    value;
    bool                            enable;
    struct delayed_work             work;
};
...
static int ioboard_adc_probe (struct platform_device *pdev)
{
    struct ioboard_adc *ioboard_adc;

    if(!(ioboard_adc = kzalloc(sizeof(struct ioboard_adc), GFP_KERNEL)))
        return -ENOMEM;
...
    ioboard_adc->client = s3c_adc_register( pdev, NULL, NULL, 0 );

    if(ioboard_adc_irq_init(ioboard_adc))               goto err_irq_init;

    if(ioboard_adc_sysfs_init(pdev))                    goto err_input_init;

    if(ioboard_adc_irq_enable(ioboard_adc, true))   goto err_driver_init;

    return 0;
...
}
...
```

- ADC Common 드라이버에 Export되어진 함수를 사용하기 위하여 선언함.
- Raw Data를 Voltage값으로 변환된 Data 저장
- ADC Raw Data 저장
- Work Q enable/disable flag
- 설정된 ADC Channel을 주기적으로 읽기 위하여 사용
- ADC Control Handle을 위하여 Common Driver에 Register한다.
- Work Q 초기화
- SYSFS 초기화
- Work Q Enable

Platform Data Struct 구성

Kernel/drivers/misc/odroida4-io/adc/ioboard-adc.h

```
#define     IOBOARD_DRIVER_NAME       "ioboard-gpio"

struct ioboard_adc_pdata   {
    int     ref_voltage;    // ADC Reference Voltage
    char    channel;        // Read Channel
    int     interval;       // Work-Q interval
};
```

- ADC 동작 주기 설정(msec)
- 읽어낼 ADC Channel
- ADC Channel에서 읽은값을 Voltage값으로 변환하기 위한 ADC Controller의 Reference voltage값
- Platform Driver Name 정의

Voltage Monitor Driver(3)

Work Q 초기화

Kernel/drivers/misc/odroida4-io/adc/ioboard-adc.c

```
static void ioboard_adc_work(struct work_struct *work)
{
    struct ioboard_adc    *ioboard_adc =
                                container_of(work, struct ioboard_adc, work);

    ioboard_adc->value =
        s3c_adc_read(ioboard_adc->client, ioboard_adc->pdata->channel);

    if(ioboard_adc->value && ioboard_adc->pdata->ref_voltage)
        ioboard_adc->voltage =
            (ioboard_adc->pdata->ref_voltage * ioboard_adc->value) / 4096;

    else
        ioboard_adc->voltage = 0;

    ioboard_adc->enable = false;
    ioboard_adc_irq_enable(ioboard_adc, true);
}

static  int     ioboard_adc_irq_init   (struct ioboard_adc *ioboard_adc)
{
    INIT_DELAYED_WORK(&ioboard_adc->work, ioboard_adc_work);

    return  0;
}
...
```

ADC Common Driver로 부터 Platform data에 설정된 Channel의 ADC Data를 가져옴.

Work Q Callback 함수 등록

Work Q Enable/Disable

Kernel/drivers/misc/odroida4-io/adc/ioboard-adc.c

```
...
static  int     ioboard_adc_irq_enable
                                (struct ioboard_adc *ioboard_adc, bool enable)
{
    if(ioboard_adc->enable != enable)   {

        ioboard_adc->enable = enable;

        if(enable)  {
            if(ioboard_adc->pdata->interval)   {

                schedule_delayed_work(&ioboard_adc->work,
                                msecs_to_jiffies(ioboard_adc->pdata->interval));
            }
        }
        else    {
            cancel_delayed_work(&ioboard_adc->work);
        }
    }
    return  0;
}
...
```

대기중인 Work Q를 종료함

Platform data에 정의 되어진 Interval msec 만큼 지연 후에 동작

Voltage Monitor Driver(4)

SYSFS File 구성

Kernel/drivers/misc/odroida4-io/adc/ioboard-adc.c

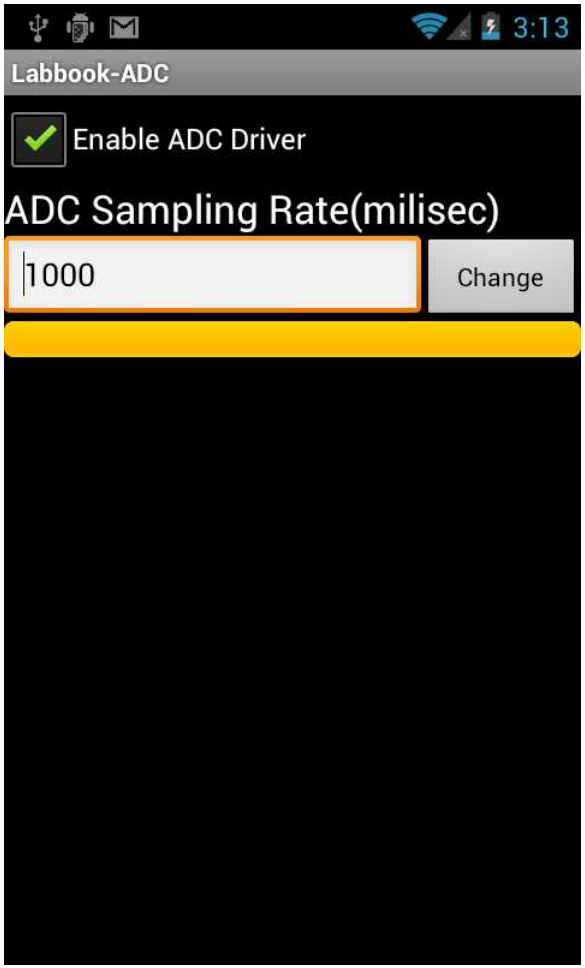
```
...
static ssize_t show_enable   (struct device *dev, struct device_attribute *attr, char *buf);
static ssize_t store_enable  (struct device *dev, struct device_attribute *attr,
                              const char *buf, size_t count);

static ssize_t show_voltage  (struct device *dev, struct device_attribute *attr, char *buf);

static ssize_t show_value    (struct device *dev, struct device_attribute *attr, char *buf);

static ssize_t show_interval  (struct device *dev, struct device_attribute *attr, char *buf);
static ssize_t store_Interval (struct device *dev, struct device_attribute *attr,
                               const char *buf, size_t count);

static DEVICE_ATTR( enable,      S_IRWXUGO,   show_enable,   store_enable);
static DEVICE_ATTR( voltage,     S_IRWXUGO,   show_voltage,  NULL);
static DEVICE_ATTR( value,       S_IRWXUGO,   show_value,    NULL);
static DEVICE_ATTR( linterval,   S_IRWXUGO,   show_interval, store_interval);

static struct attribute *ioboard_adc_entries[] = {
    &dev_attr_enable.attr,
    &dev_attr_voltage.attr,
    &dev_attr_value.attr,
    &dev_attr_interval.attr,
    NULL
};

static struct attribute_group ioboard_adc_attr_group = {

    .name = NULL,
    .attrs  = ioboard_adc_entries,
};
...
```

Kernel/drivers/misc/odroida4-io/adc/ioboard-adc.c

```
...
static ssize_t show_voltage
            (struct device *dev, struct device_attribute *attr, char *buf)
{
    struct ioboard_adc  *ioboard_adc = dev_get_drvdata(dev);

    return sprintf(buf, "%d₩n", ioboard_adc->voltage);
}

static  ssize_t show_value
            (struct device *dev, struct device_attribute *attr, char *buf)
{
    struct ioboard_adc *ioboard_adc = dev_get_drvdata(dev);

    return  sprintf(buf, "%d₩n", ioboard_adc->value);
}
...
```

Voltage Monitor Driver(5)

SYSFS를 통한 Driver 검증



Android Application : labbook-ADC(1)

- **JNI function을 만든다.**

```
static {
        System.loadLibrary(“ADCDriverSysfs");
 }
```

```
native int checkNode();                          //sysfs 노드가 있는지 확인
native int getState();                           //enable 상태 확인
native void setState(int enable);                //enable/disable 변경
native int readSamplingRate();                   //sampling rate 값 확인
native void setSamplingRate(String value);       //sampling rate 값 변경
native int readVoltage();                        //ADC 값 확인
```

Android Application : labbook-ADC(2)


```
if (checkNode() == 0) { //sysfs node를 확인하여 UI를 그린다.
                mADCDriverEnable.setVisibility(View.GONE);
                mSamplingRateText.setVisibility(View.GONE);
                mSamplingRate.setVisibility(View.GONE);
                mChangeSamplingRate.setVisibility(View.GONE);
        } else {
                mADCDriverEnable.setVisibility(View.VISIBLE);
                mSamplingRateText.setVisibility(View.VISIBLE);
                mSamplingRate.setVisibility(View.VISIBLE);
                mChangeSamplingRate.setVisibility(View.VISIBLE);

                mSamplingRate.setText(readSamplingRate() + ""); //sampling rate을 jni로 부터 읽어서 표
시한다.

        }
```

Android Application : labbook-ADC(3)


```
int state = getState(); //enable jni로 부터 읽어 상태 확인
if (state == 1) {
        mADCDriverEnable.setChecked(true);
        update(true);
} else {
        mADCDriverEnable.setChecked(false);
        update(false);
}
private void update(boolean enable) {          //controller 상태 업데이트
        if (enable) {
                mSamplingRateText.setEnabled(true);
                mSamplingRate.setEnabled(true);
                mChangeSamplingRate.setEnabled(true);
        } else {
                mSamplingRateText.setEnabled(false);
                mSamplingRate.setEnabled(false);
                mChangeSamplingRate.setEnabled(false);
        }
```

Android Application : labbook-ADC(4)

- **ADC를 sampling rate 값으로 주기적으로 읽어 오는 handler를 만든다.**

```
mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            // TODO Auto-generated method stub
            super.handleMessage(msg);
            sendEmptyMessageDelayed(0, mSamplingRateMilisec);
            mVoltage.setProgress(readVoltage());  //읽어온 ADC값을 Progress Bar에 표시한다
        }
    };
    mHandler.sendEmptyMessage(1);
```

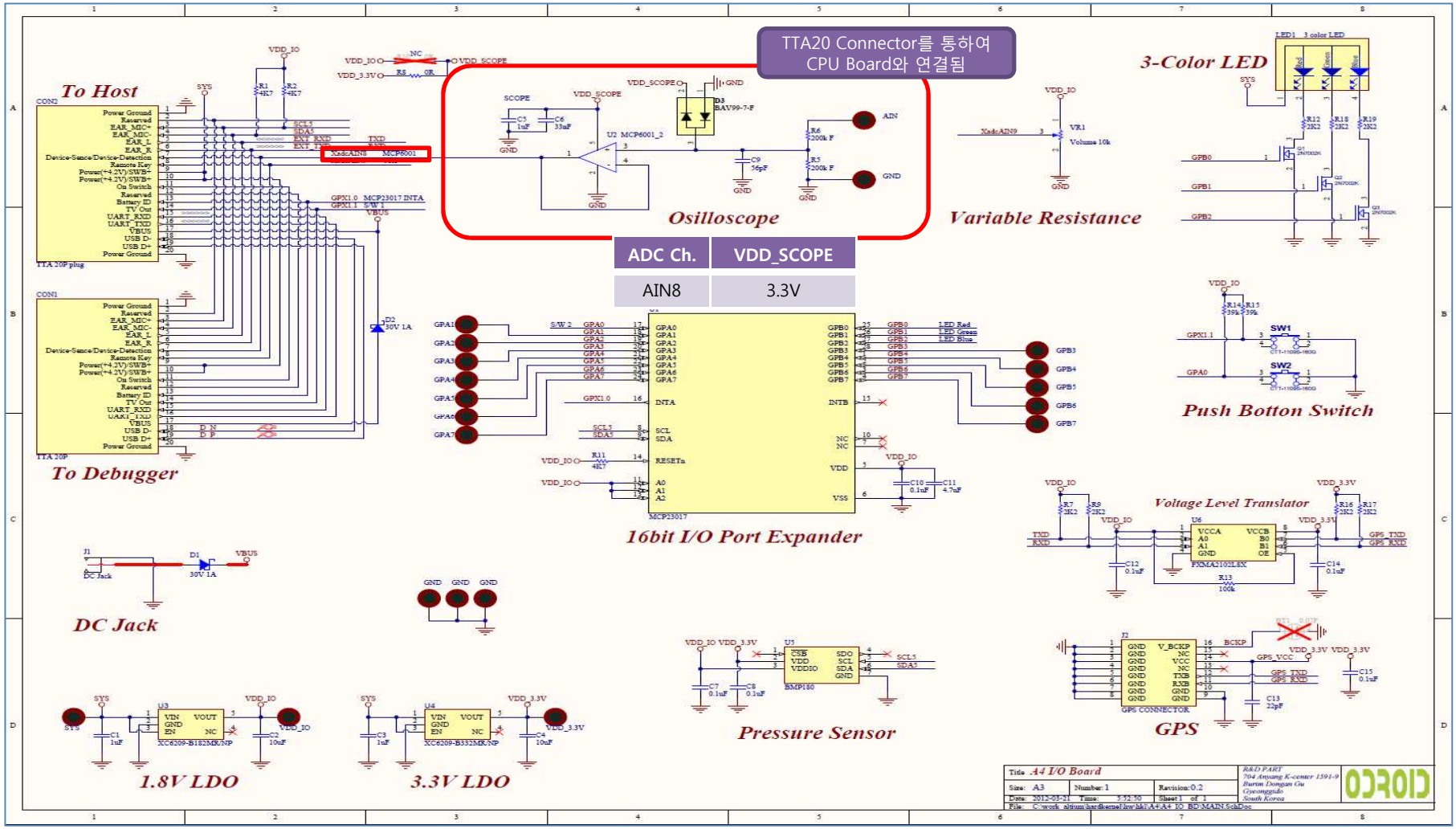
하드웨어 확장 – **ADC**(Oscilloscope)

Agenda

- ODROID-A4 IO Board ADC (Channel 8) Hardware Interface
- Simple Oscilloscope 구조
  - Simple Oscilloscope App 구성
- ADC Data 가공
- Simple Oscilloscope Driver
  - 데이터 전송을 위한 Misc driver 등록
  - IOCTL을 이용한 Data전송
- Android Application : OdroidOscilloscope
  - Display Data 연산과정
  - Simple Oscilloscope APP

ODROID-A4 IO Board ADC Hardware Interface

Odroid-A4 I/O Board의 ADC연결



Simple Oscilloscope 구조

Simple Oscilloscope App 구성

ADC Data 가공

- 가로축 : 일정한 간격으로 Sampling되어진 Data를 분해능의 시간만큼 가져옴
  - 1ms의 분해능인 경우 1ms * 10[가로축 칸의 개수] = 10ms의 Data를 가져옴.
  - 화면의 가로축이 600 pixel이므로 가져온 데이터 개수에서 600개를 Display
  - 드라이버에서 10us 주기로 10000개의 Data가 sampling되어 저장하고 있다면
    - 10us * 10000 = 100 ms : 드라이버에서 저장하고 있는 ADC데이터 실시간 데이터
    - 6ms의 분해능으로 설정된 경우

      6ms * 10 [가로축 칸의 개수] = 60ms 데이터가 필요, 6000개의 데이터를 가져옴

      화면에 표시되는 pixel은 600개 이므로 10개중 1개 (6000/600)만 화면에 display함.
    - 600us의 분해능으로 설정된 경우

      600us * 10[가로축 칸의 개수] = 6ms 데이터가 필요, 600개의 데이터를 가져옴
    - 300us의 분해능으로 설정된 경우

      300us * 10 = 3ms 데이터가 필요, 300개의 데이터를 가져옴.

      화면에 1개의 데이터를 2번 표시하여 600개를 맞춤.

- 세로축 : 입력된 ADC데이터를 Voltage(mV)값으로 전환하고 화면에 표시
  - 입력은 ½로 되어있고 ADC Resolution은 12bit, ADC Reference Voltage는 3.3V
    - Voltage(mV) = 2 * ADC Val * 3300 / 4096
  - 화면의 세로축이 380 pixel이므로 1칸당 38 pixel
  - 1V의 분해능인 경우 38 pixel / 1000(mV) * ADC Voltage(mV)

Simple Oscilloscope Driver(1)

ADC Common Driver modified

Kernel/arch/arm/plat-samsung/adc.c

```
...
void s3c_adc_setup(unsigned int prescale, unsigned int delay)
{
    struct adc_device   *adc = adc_dev;
    unsigned int        tmp;

    adc->prescale   = S3C2410_ADCCON_PRSCVL(prescale);
    adc->delay      = S3C2410_ADCDLY_DELAY(delay);

    tmp = adc->prescale | S3C2410_ADCCON_PRSCEN;

    if (adc->cputype != TYPE_S3C24XX) {
        /* Enable 12-bit ADC resolution */
        tmp |= S3C64XX_ADCCON_RESSEL;
    }
    tmp |= S3C2410_ADCCON_STDBM;

    writel(tmp, adc->regs + S3C2410_ADCCON);
    writel(adc->delay, adc->regs + S3C2410_ADCDLY);
}
EXPORT_SYMBOL_GPL(s3c_adc_setup);
...
static irqreturn_t s3c_adc_irq(int irq, void *pw)
{
    if (client->nr_samples > 0) {
        /* fire another conversion for this */
        client->select_cb(client, 1);

        writel(0, adc->regs + S3C64XX_ADCCLRINT);

        s3c_adc_convert(adc);

        return IRQ_HANDLED;
    } else {
...
    return IRQ_HANDLED;
}
...
```

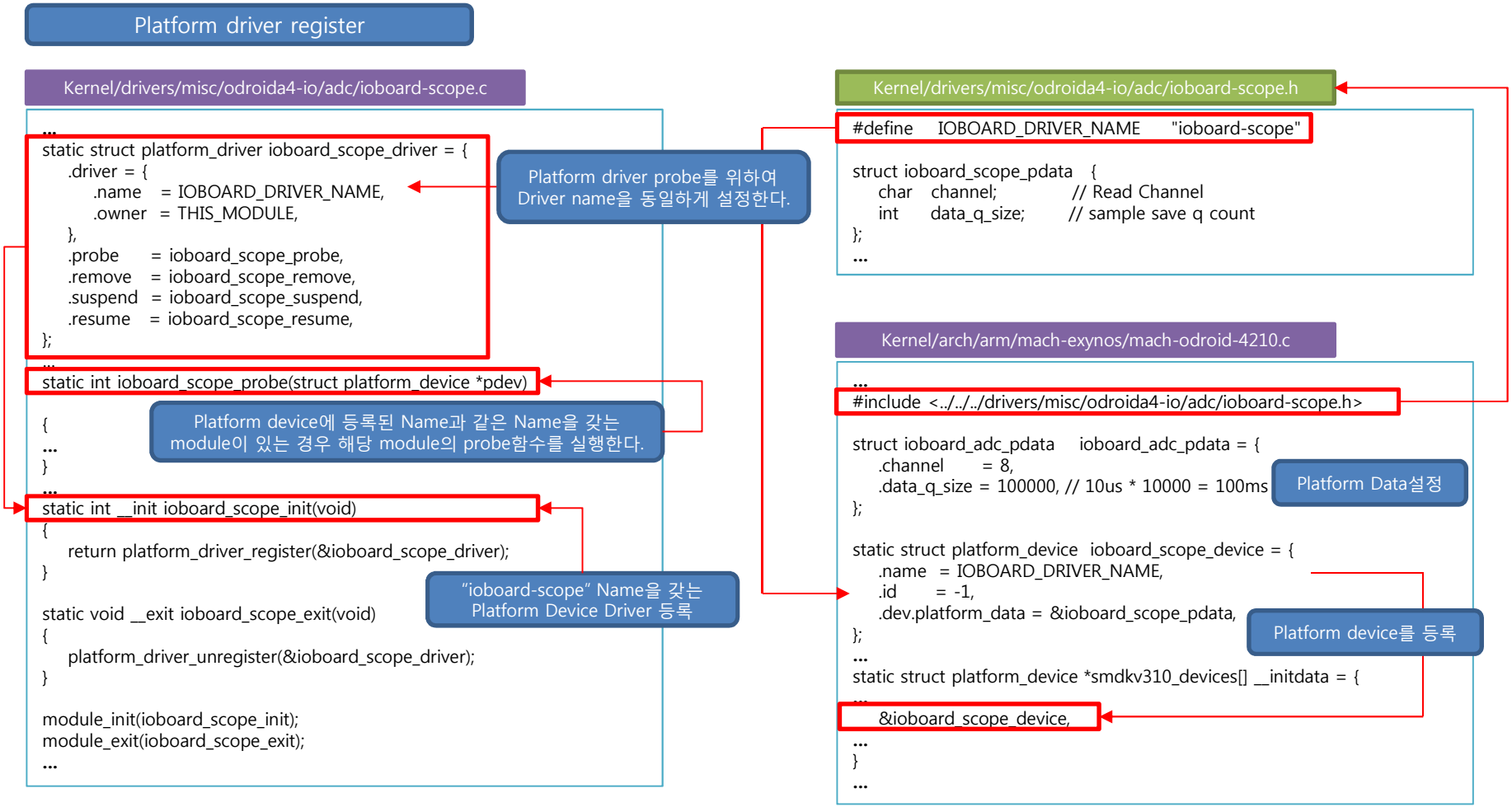
Kernel/arch/arm/plat-samsung/include/plat/adc,h

```
...
extern int s3c_adc_start(struct s3c_adc_client *client,
                         unsigned int channel, unsigned int nr_samples);

extern int s3c_adc_read(struct s3c_adc_client *client, unsigned int ch);

extern struct s3c_adc_client  *s3c_adc_register(
                         struct platform_device *pdev,
                         void (*select)(struct s3c_adc_client *client,
                                        unsigned selected),
                         void (*conv)(struct s3c_adc_client *client,
                                      unsigned d0, unsigned d1,
                                      unsigned *samples_left),
                         unsigned int is_ts);

extern void s3c_adc_release(struct s3c_adc_client *client);

extern void s3c_adc_setup(unsigned int prescale, unsigned int delay);

...
```

ADC Sampling주기를
설정하기 위한 함수를 작성하고
외부에서 실행하도록 설정한다.

ADC가 빠르게 동작하는 경우
ADC Int flag를 먼저 Clear
하고 Convert를 실행한다.

Simple Oscilloscope Driver(2)



Simple Oscilloscope Driver(3)

ADC Common Driver Register

Kernel/drivers/misc/odroida4-io/adc/ioboard-scope.c

```
#include <plat/adc.h>
...
struct ioboard_scope  {

    struct  s3c_adc_client          *client;

    struct  ioboard_scope_pdata  *pdata;
    unsigned int                        *data_q;
    unsigned int                        data_pos,;
    volatile bool                       enable;
    unsigned int                        data_get_count;
};
static struct ioboard_scope *gScope;
...
static int ioboard_scope_probe (struct platform_device *pdev)
{
    struct ioboard_scope *ioboard_scope;

    if(!(ioboard_scope = kzalloc(sizeof(struct ioboard_scope), GFP_KERNEL)))
        return -ENOMEM;

    gScope = ioboard_scope;
...
    ioboard_adc->client = s3c_adc_register( pdev,
                                            NULL,
                                            ioboard_scope_convert,
                                            0 );

    // 1 us Sampling, 1 us ADC delay
    s3c_adc_setup(19, 100);
...
    return 0;
...
}
...
```

ADC Common 드라이버에 Export되어진 함수를 사용하기 위하여 선언함.

Raw Data를 저장하는 Queue 변수 선언

Data Q에 저장될 위치

Data enable/disable control

몇 개의 Data를 전송 할 것인지 저장하는 변수

Platform driver와 ADC Common driver 사이에 데이터를 공유하기 위하여 사용

Platform Data Struct 구성

Kernel/drivers/misc/odroida4-io/adc/ioboard-scope.h

```
#define     IOBOARD_DRIVER_NAME      "ioboard-gpio"

struct ioboard_adc_pdata   {
    char    channel;          // Read Channel
    int     data_q_sizel;     // Storage data q size
};
```

드라이버에서 가지고 있는 Sample Data의 최대 갯수

읽어낼 ADC Channel

Platform Driver Name 정의

Kernel/drivers/misc/odroida4-io/adc/ioboard-scope.c

```
...
static  void    ioboard_scope_convert (struct s3c_adc_client *client,
                                       unsigned d0, unsigned d1,
                                       unsigned *samples_left)
{
    struct ioboard_scope *ioboard_scope = gScope;
...
}
...
```

ADC의 Convert가 완료되는 경우 ADC common driver로 부터 호출되는 function

Simple Oscilloscope Driver(4)

ADC Sample Data 처리 (Enable/Disable)

Kernel/drivers/misc/odroida4-io/adc/ioboard-scope.c

```
...
static  void    ioboard_scope_enable
                        (struct ioboard_scope *ioboard_scope, bool enable)
{
    if(enable)  {
        if(ioboard_scope->enable != true)  {
            ioboard_scope->enable = true;
            ioboard_scope->data_pos = 0;
            memset(ioboard_scope->data_q, 0x00,
                    sizeof(unsigned int) * ioboard_scope->pdata->data_q_size);

            s3c_adc_start(ioboard_scope->client,
                          ioboard_scope->pdata->channel,
                          ioboard_scope->enable);
        }
    }
    else    ioboard_scope->enable = false;
}

static  void    ioboard_scope_convert (struct s3c_adc_client *client,
                                       unsigned d0, unsigned d1,
                                       unsigned *samples_left)
{
    struct ioboard_scope *ioboard_scope = gScope;

    ioboard_scope->data_q[ioboard_scope->data_q_ep] = d0;
    ioboard_scope->data_q_ep++;
    ioboard_scope->data_q_ep %= ioboard_scope->pdata->data_q_size;

    *samples_left = ioboard_scope->enable;
}
...
```

설정되어진 Channel로 ADC Convert Start

ADC 데이터를 Data Q에 저장함

다음 데이터 Sampling 하도록 설정

Kernel/arch/arm/plat-samsung/adc.c

```
static irqreturn_t s3c_adc_irq(int irq, void *pw)
{
...
    if (client->convert_cb)
        (client->convert_cb)(client, data0, data1, &client->nr_samples);

    if (client->nr_samples > 0) {
        /* fire another conversion for this */
        client->select_cb(client, 1);

        /* Clear ADC interrupt */
        writel(0, adc->regs + S3C64XX_ADCCLRINT);

        s3c_adc_convert(adc);

        return IRQ_HANDLED;
    } else {
        spin_lock(&adc->lock);
        (client->select_cb)(client, 0);
        adc->cur = NULL;

        s3c_adc_try(adc);
        spin_unlock(&adc->lock);
    }
exit:
    if (adc->cputype != TYPE_S3C24XX) {
        /* Clear ADC interrupt */
        writel(0, adc->regs + S3C64XX_ADCCLRINT);
    }
    return IRQ_HANDLED;
}
...
```

ADC Conversion End Interrupt Function

ADC Common Driver Register시에 등록되어진 함수 호출

다음 데이터 Sampling Start

Simple Oscilloscope Driver(5)

Data 전송을 위한 Misc Driver Register

Kernel/drivers/misc/odroida4-io/adc/ioboard-scope.c

```
...
#include <plat/adc.h>
...

static int ioboard_scope_open(struct inode *inode, struct file *file)
{
    return 0;
}

static int ioboard_scope_ioctl
                (struct inode *inode, unsigned int cmd, void *arg)
{
...
}

static const struct file_operations ioboard_scope_fops = {
    .owner          = THIS_MODULE,
    .open           = ioboard_scope_open,
    .unlocked_ioctl = ioboard_scope_ioctl,
};

static struct miscdevice ioboard_scope_miscdev = {

    .minor      = MISC_DYNAMIC_MINOR,

    .name       = "adc",

    .fops       = &ioboard_scope_fops,
};

static int ioboard_scope_probe (struct platform_device *pdev)
{
...
    if(misc_register(&ioboard_scope_miscdev))  goto err_driver_init;
...
}
...
```

Minor number 자동 할당

Dev node 이름 설정 (/dev/adc)

Misc Driver IOCTL Commands

Kernel/drivers/misc/odroida4-io/adc/ioboard-scope.c

```
...
#define ADC_IO_MAGIC        "S"

#define ADC_ENABLE          _IOW(ADC_IO_MAGIC, 0x0, int)
#define ADC_SET_PERIOD      _IOW(ADC_IO_MAGIC, 0x1, int)
#define ADC_REQ_BUF         _IOR(ADC_IO_MAGIC, 0x2, void*)

static int ioboard_scope_ioctl
                (struct inode *inode, unsigned int cmd, void *arg)
{
    struct  ioboard_scope   *ioboard_scope  = gScope;
    unsigned int            *request_buffer, pos, copy_pos, copy_size;

    switch (cmd) {
        case ADC_ENABLE:
            ioboard_scope_enable (ioboard_scope, (int)arg);
        break;
        case ADC_SET_PERIOD:
            ioboard_scope->data_get_count = (int)arg;
        break;
        case ADC_REQ_BUF:
            ...
        break;
        default:
            return -ENOIOCTLCMD;
    }
    return 0;
}
...
```

APP과 통신시 사용되는 IOCTL Magic 번호

APP에서 필요한 data 갯수 전달

APP에서 요청하는 데이터 개수 만큼 ADC Data Q에서 User영역으로 복사하여 준다.

Data Sampling Enable/Disable

Android Application : OdroidOscilloscope(1)

- jni에서 접근 가능하도록 ramdisk의 init에서 /dev/adc 권한 변경

- init.odroida4.rc에 추가

chown system system /dev/adc

chmod 0666 /dev/adc

Android Application : OdroidOscilloscope(2)

➢ jni 함수 설명

```
public native int openScope();                  //ADC node open
public native int [] readScope();               //driver로 부터 ADC 값 배열을 읽어 옴
public native int requestADC(int h, int v);     //volt와 sampling interval을 설정
public native void closeScope();                //ADC node close
public native int powerOnScope();               //ADC Sampling 시작
public native int powerOffScope();              //ADC Sampling 끝
```

- 현재 설정되어 있는 Volt와 sampling 개수를 설정한다.

```
powerOnScope();
requestADC(mAmpScaleIndex, mTimeBaseIndex);
mMessage = mHandler.obtainMessage();
mMessage.what = MSG_GET_ADC_REQ;
```

- readScope()로 부터 600개 ADC값을 읽어와서 mVal 배열에 복사한다.

```
case MSG_GET_ADC_REQ:

        if(mBtnRun.isChecked()) {
                mVal = readScope();
                msg = mHandler.obtainMessage();
                msg.what = MSG_GET_ADC_READ;
                msg.obj = 0;
                mHandler.sendMessage(msg);
        }
        break;
```

Android Application : OdroidOscilloscope(5)

➢ **readScope()로 부터 600개 ADC값을 읽어와서 mVal 배열에 복사한다.**

```
if(mVal[i] == NOTI_ADC_START)  {
//배열 첫 번째일 때 설정 값을 다시 설정하는 message를 보낸다.
            msg = mHandler.obtainMessage();
            msg.what = MSG_SAVE_ADC_REQ;
            msg.obj = 0;
            mHandler.sendMessage(msg);
} else if (mVal[i] == NOTI_ADC_END)  {
//배열의 마지막일 경우 다시 데이터를 요청하도록 반복한다.
            if(mScopedraw.setData(mVal)) {    //SurfaceView에 그래프를 그릴 데이터를 복사한다.
                        msg = mHandler.obtainMessage();
                        msg.what = MSG_GET_ADC_REQ;
                        msg.obj = 0;
                        mHandler.sendMessage(msg);
            }
```

Android Application : OdroidOscilloscope(6)

➢ Driver로 부터 읽은 adc 값을 app으로 가공하여 전달

```
int             GetPlotData     (unsigned int *pData)
{
    float    plot_data, mV_scale;
    int data_pos_offset = 0;
    int     i;

    data_pos_offset = ((GetDataSize() -1) / (SCREEN_HORIZONTAL_PIXEL -1)); //sampling interval에 맞게 읽어온다.

    mV_scale        = 2 * 3300 / 0xfff;

    for(i = 0; i < SCREEN_HORIZONTAL_PIXEL; i++)    {
        plot_data = gDataPool[data_pos_offset * i] * mV_scale;      //sampling 개수에 맞게 읽어 옴
        if(plot_data)   plot_data = plot_data * GetVoltScale() / 1000;              //Volt에 맞게 좌표 수정
        if(plot_data > (SCREEN_VERTICAL_PIXEL -1))      plot_data = SCREEN_VERTICAL_PIXEL -1;
        pData[i] = (unsigned int)plot_data;
    }

    return  0;
}
```

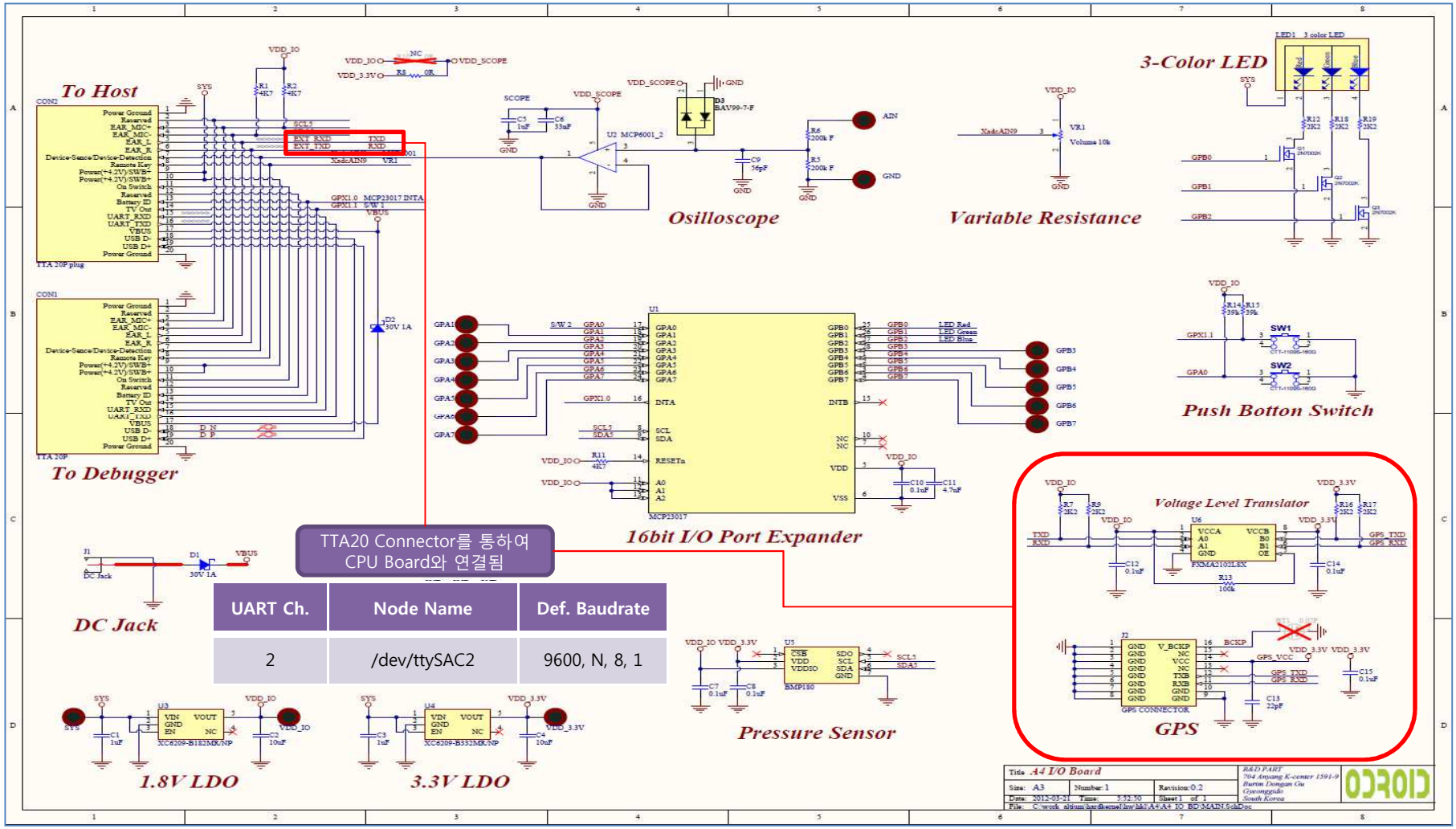
하드웨어 확장 – **UART(GPS)**

Agenda

- ODROID-A4 IO Board UART (GPS) Hardware Interface
- Android Application : Labbook-GPS

ODROID-A4 IO Board UART (GPS) Hardware Interface

Odroid-A4 I/O Board의 GPS연결



UART Ch.	Node Name	Def. Baudrate
2	/dev/ttySAC2	9600, N, 8, 1

Android Application : Labbook-GPS(1)

- jni에서 접근 가능하도록 ramdisk의 init에서 /dev/ttySAC2 권한 변경

- init.odroida4.rc에 추가

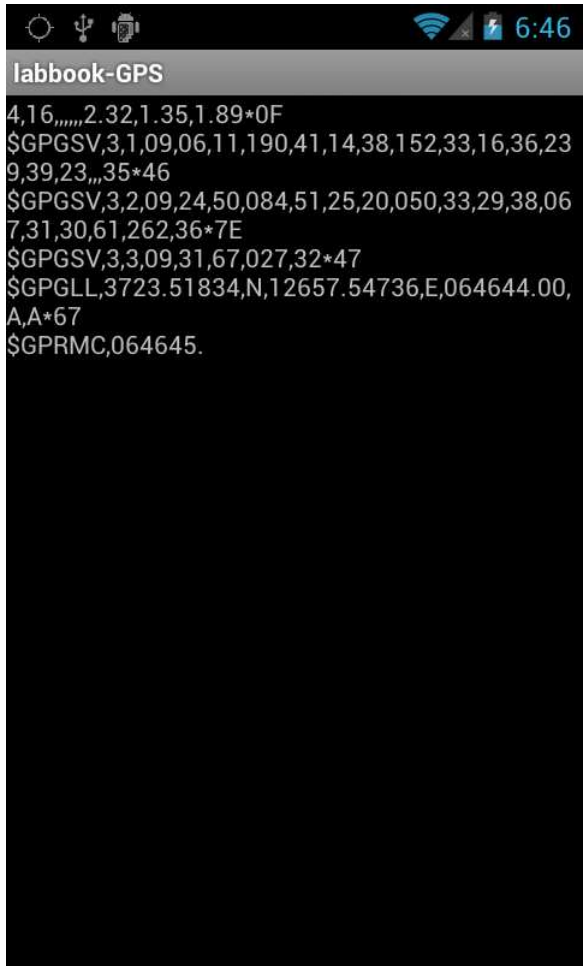
chmod 0666 /dev/ttySAC2

- uevent.odroida4.rc에 추가

/dev/ttySAC2 0666 gps gps

Android Application : Labbook-GPS(2)

/dev/ttySAC2로부터 nmea 값을 읽어서 표시



➢ 1초 마다 readGPS() 함수를 호출하여 nmea 값 표시

```
mHandler = new Handler() {
public void handleMessage(Message msg) {
            super.handleMessage(msg);
            sendEmptyMessageDelayed(0, 1000);
            update();
          }
      };
      mHandler.sendEmptyMessage(1);

   public void update() {
      TextView tv = (TextView) findViewById(R.id.tv_gps);
      tv.setText(readGPS() + "");
   }
```

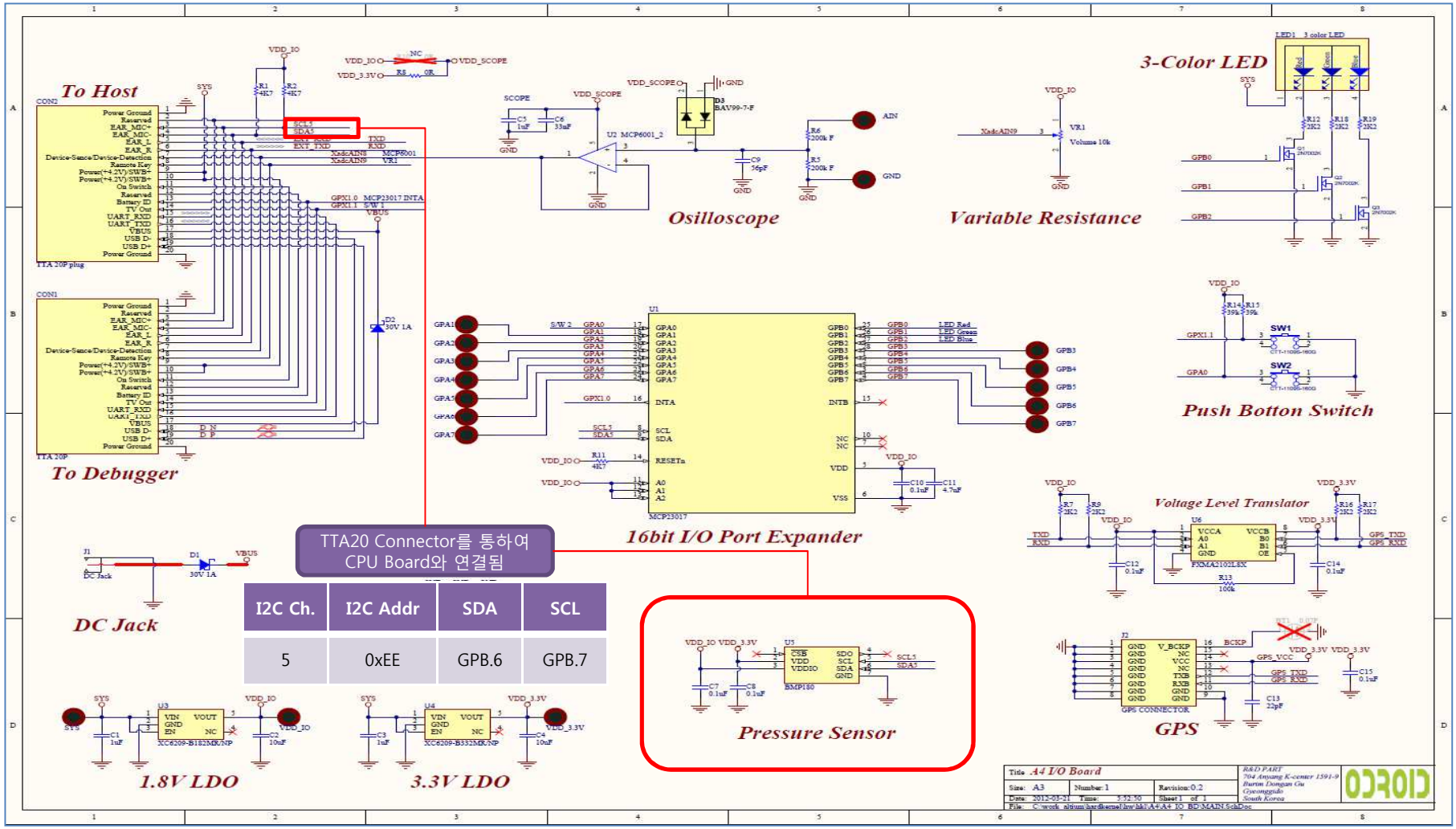
하드웨어 확장 – I2C(Sensor)

Agenda

- ODROID-A4 IO Board I2C (Sensor) Hardware Interface
- Pressure Measurement
  - BMP180 Measurement Flow
- Pressure, Temperature Data Calculate
- Sensor(BMP180) I2C Driver
  - GPIO I2C Platform driver register
  - Sensor(BMP180) I2C driver register
- Sensor Control을 위한 Input device 및 Sysfs 초기화
- Sensor(BMP180) Driver Function Flow
- Android Application : Libsensor(sensor.odroida4.so)
  - LibSensor 구조
  - Pressure Sensor Control 및 Data Report
- Android Application : Labbook-BMP180

ODROID-A4 IO Board I2C(Sensor) Hardware Interface

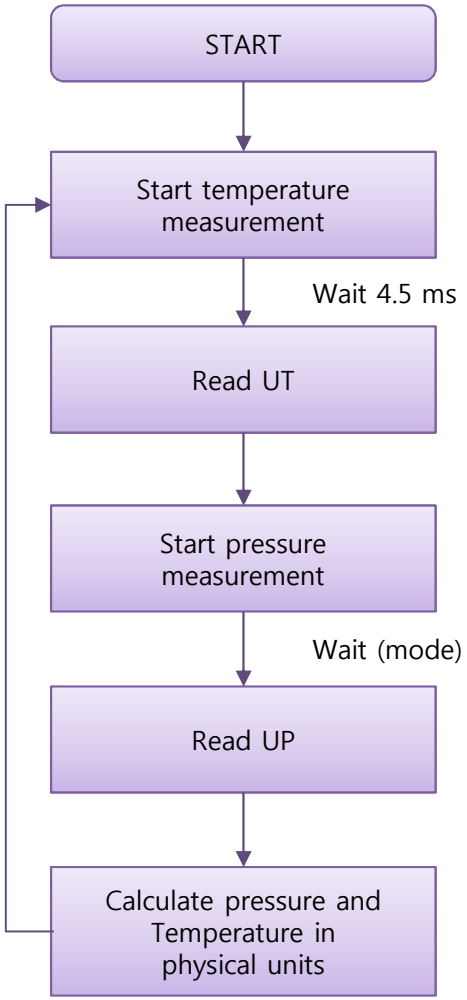
Odroid-A4 I/O Board의 Sensor 연결



I2C Ch.	I2C Addr	SDA	SCL
5	0xEE	GPB.6	GPB.7

Pressure Measurement

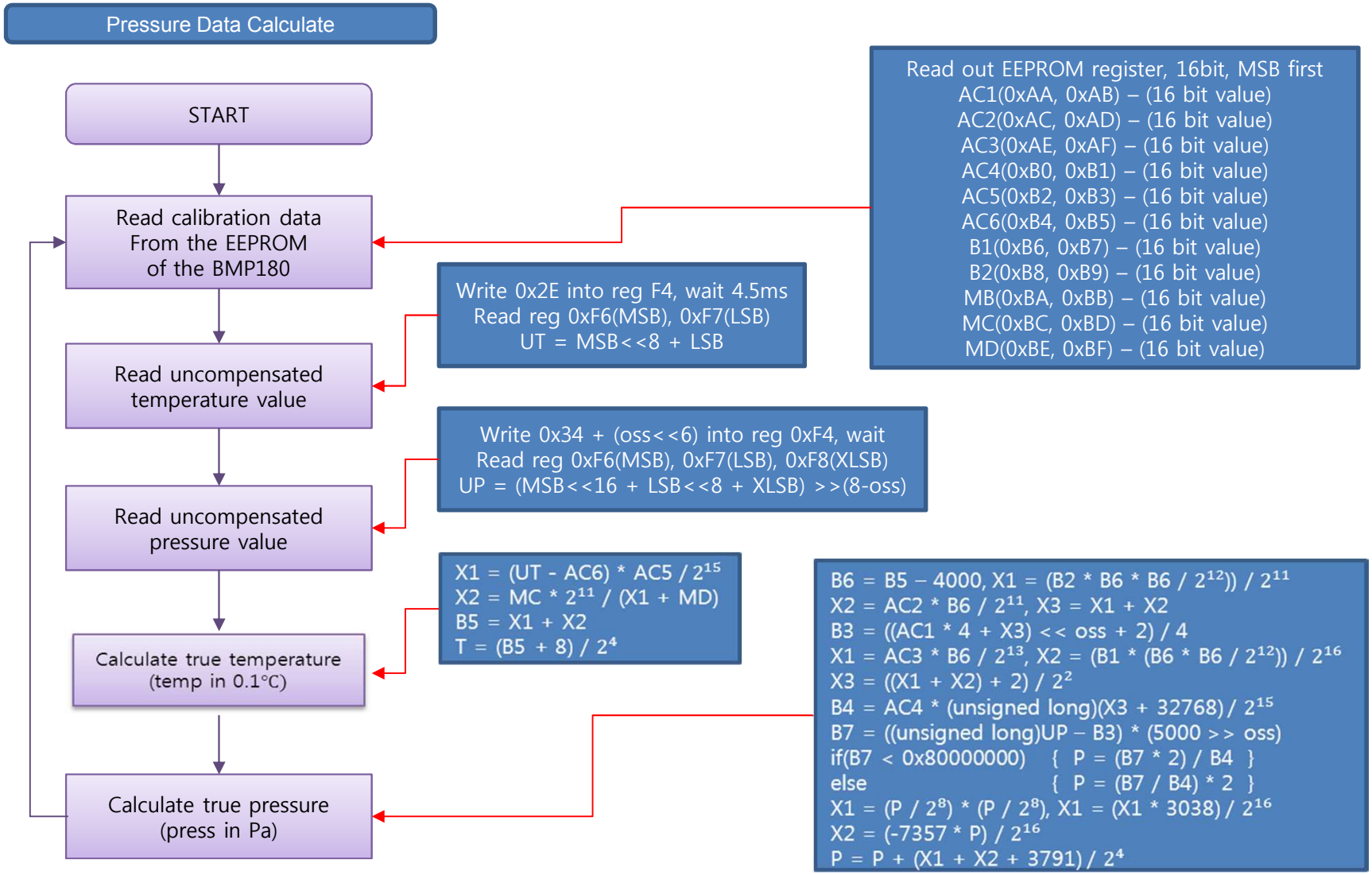
BMP180 Measurement Flow



BMP180 Modes

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure	0x34	4.5
Pressure	0x74	7.5
Pressure	0xB4	13.5
Pressure	0xF4	25.5

Pressure, Temperature Measurement



Sensor(BMP180) I2C Driver(1)

GPIO-I2C Platform driver register

Kernel/drivers/i2c/busses/i2c-gpio.c

```
...
static struct platform_driver i2c_gpio_driver = {
    .driver   = {
        .name   = "i2c-gpio",
        .owner  = THIS_MODULE,
    },
    .probe    = i2c_gpio_probe,
    .remove   = __devexit_p(i2c_gpio_remove),
};
...
static int __devinit i2c_gpio_probe(struct platform_device *pdev)
{
...
    ret = gpio_request(pdata->sda_pin, "sda");
    if (ret)
        goto err_request_sda;

    ret = gpio_request(pdata->scl_pin, "scl");
    if (ret)
        goto err_request_scl;
...
    if (pdata->udelay)
        bit_data->udelay = pdata->udelay;
    else if (pdata->scl_is_output_only)
        bit_data->udelay = 50;    /* 10 kHz */
    else
        bit_data->udelay = 5;     /* 100 kHz */

    if (pdata->timeout)
        bit_data->timeout = pdata->timeout;
    else
        bit_data->timeout = HZ / 10;     /* 100 ms */

}
...
```

Platform driver probe를 위하여 Driver name을 동일하게 설정한다.

Adaptor number가 중복되지 않도록 주의하여 설정한다.

Platform Data에 설정된 값으로 초기화

Kernel/include/linux/i2c-gpio.h

```
...
struct i2c_gpio_platform_data {
    unsigned int    sda_pin;
    unsigned int    scl_pin;
    int             udelay;
    int             timeout;
    unsigned int    sda_is_open_drain:1;
    unsigned int    scl_is_open_drain:1;
    unsigned int    scl_is_output_only:1;
};
...
```

Kernel/arch/arm/mach-exynos/mach-odroid-4210.c

```
...
#define    GPIO_I2C_SDA    EXYNOS4_GPB(6)
#define    GPIO_I2C_SCL    EXYNOS4_GPB(7)

static struct  i2c_gpio_platform_data  i2c_gpio_platdata = {
    .sda_pin = GPIO_I2C_SDA,
    .scl_pin   = GPIO_I2C_SCL,
    .udelay   = 5,   // 100Khz SCL clock
};

static struct          platform_device     ioboard_i2c_device = {
    .name                = "i2c-gpio",
    .id                  = 5,
    .dev.platform_data = &i2c_gpio_platdata,
};
...
static struct platform_device *smdkv310_devices[] __initdata = {
...
    &ioboard_i2c_device,
...
}
...
```

Sensor(BMP180) I2C Driver(2)

Sensor(BMP180) I2C driver register

Kernel/drivers/misc/odroida4-io/i2c/ioboard-sensor.c

```
...
static const struct i2c_device_id ioboard_sensor_id[] = {
    {   IOBOARD_DRIVER_NAME, 0  },
    {},
};

MODULE_DEVICE_TABLE(i2c, ioboard_sensor_id);

struct i2c_driver ioboard_sensor_driver ={
    .driver = {
        .name    = IOBOARD_DRIVER_NAME,
        .owner   = THIS_MODULE,
    },
    .probe       = ioboard_sensor_probe,
    .remove      = ioboard_sensor_remove,
    .suspend     = ioboard_sensor_suspend,
    .resume      = ioboard_sensor_resume,

    .id_table    = ioboard_sensor_id,
};
...
static   int ioboard_sensor_probe (struct i2c_client *client,
                                   const struct i2c_device_id *id)
{
...
}
...
```

Platform driver probe를 위하여 Driver name을 동일하게 설정한다.

Kernel/drivers/misc/odroida4-io/i2c/ioboard-sensor.h

```
...
#define IOBOARD_DRIVER_NAME      "ioboard-sensor"

struct ioboard_sensor_pdata   {
    // 0 ~ 3 (0 = single, 1 = 2 times, 2 = 4 times, 3 = 8 times)
    unsigned char    oversampling;
};
...
```

Kernel/arch/arm/mach-exynos/mach-odroid-4210.c

```
...
#include <../../../drivers/misc/odroida4-io/i2c/ioboard-sensor.h>

struct ioboard_sensor_pdata      ioboard_sensor_pdata = {
    .oversampling = 3,  // 8 times
};

static struct i2c_board_info i2c_gpio_devs[] __initdata = {
    {
        I2C_BOARD_INFO(IOBOARD_DRIVER_NAME, (0xEE>>1)),
        .platform_data = &ioboard_sensor_pdata,
    },
};
...
static void __init smdkv310_machine_init(void)
{
...
    i2c_register_board_info(5, i2c_gpio_devs, ARRAY_SIZE(i2c_gpio_devs));
...
}
...
```

Adaptor에 연결된 I2C Device의 정보

I2C Adaptor 5번(GPIO-I2C)에 연결되어 있는 Device를 등록한다.

Sensor Control을 위한 Input device 및 Sysfs 초기화

Sensor Data Report를 위한 Input Device설정

Kernel/drivers/misc/odroida4-io/i2c/ioboard-sensor.c

```
// max = 1200.00 hpa, min = 800.00 hpa
#define SENSOR_PRESSURE_MAX             120000
#define SENSOR_PRESSURE_MIN              80000

// max = 100.0 , min 0.0
#define SENSOR_TEMPERATURE_MAX          1000
#define SENSOR_TEMPERATURE_MIN          0
...
static int  ioboard_sensor_input_init
                                (struct ioboard_sensor *ioboard_sensor)
{
    if(!(ioboard_sensor->input = input_allocate_device()))      return  -1;

    ioboard_sensor->input->name = IOBOARD_DRIVER_NAME;

    input_set_capability(ioboard_sensor->input, EV_ABS, ABS_MISC);

    input_set_abs_params(ioboard_sensor->input, ABS_X,
                         SENSOR_PRESSURE_MIN,
                         SENSOR_PRESSURE_MAX, 0, 0);

    input_set_abs_params(ioboard_sensor->input, ABS_Y,
                         SENSOR_TEMPERATURE_MIN,
                         SENSOR_TEMPERATURE_MAX, 0, 0);

    if(input_register_device(ioboard_sensor->input))  {
        input_free_device(ioboard_sensor->input);
        return  -1;
    }
    return  0;
}
...
```

Report Data의 Max, Min 설정

EV_ABS Tyoe으로 설정
ABS_X에는 Pressure
ABS_Y에는 Temperature
값을 Report 하도록 설정

Sensor Control을 위한 SYSFS설정

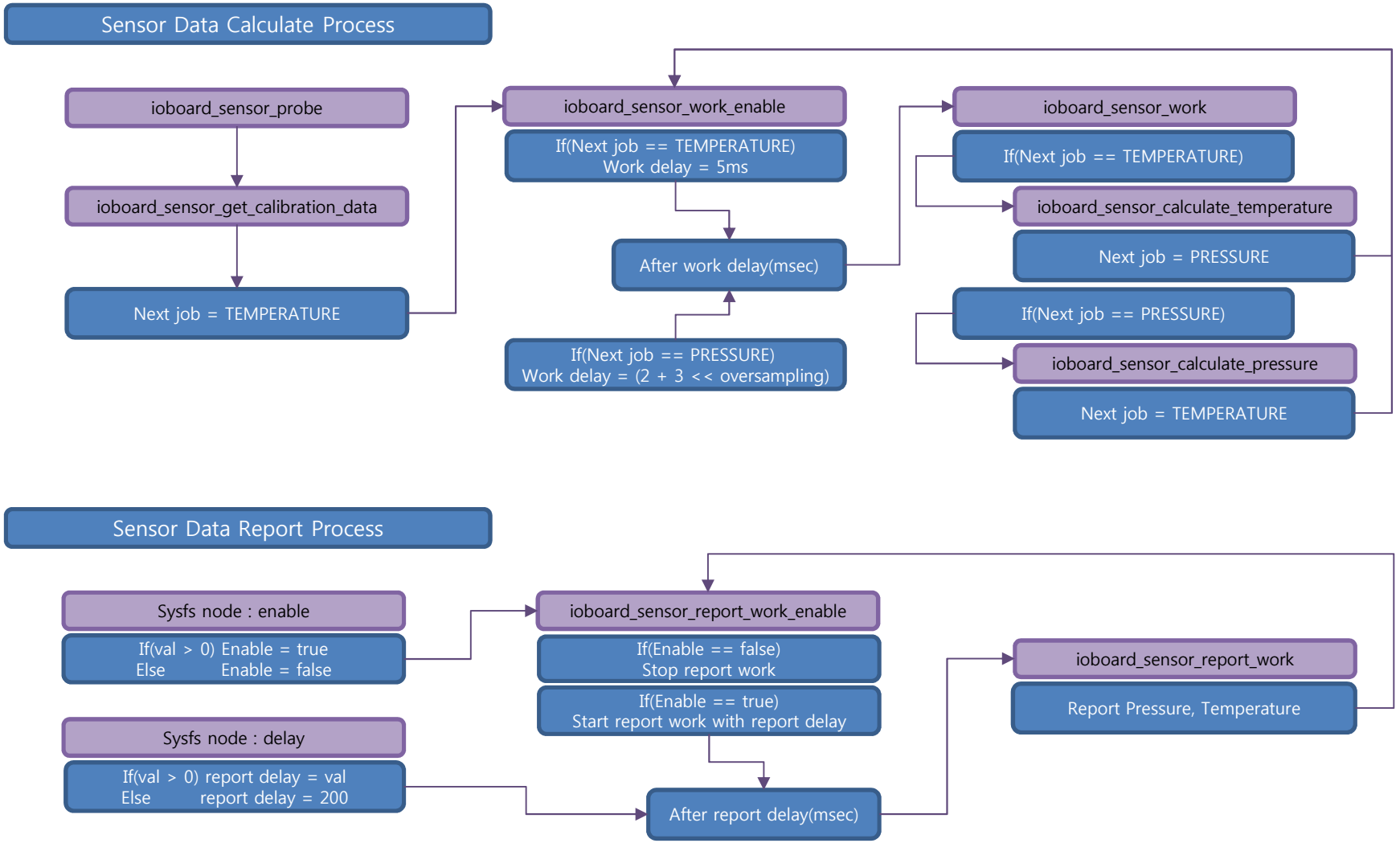
Kernel/drivers/misc/odroida4-io/i2c/ioboard-sensor.c

```
...
static ssize_t show_enable (struct device *dev,
                            struct device_attribute *attr, char *buf);
static  ssize_t store_enable(struct device *dev,
                            struct device_attribute *attr, const char *buf,
                            size_t count);

static ssize_t show_delay  (struct device *dev,
                            struct device_attribute *attr, char *buf);
static ssize_t store_delay (struct device *dev,
                            struct device_attribute *attr, const char *buf,
                            size_t count);

static DEVICE_ATTR(enable,  S_IRWXUGO, show_enable,  store_enable);
static DEVICE_ATTR(delay,   S_IRWXUGO, show_delay,   store_delay);

static struct attribute *ioboard_sensor_entries[] = {
    &dev_attr_enable.attr,
    &dev_attr_delay.attr,
    NULL
};

static struct attribute_group ioboard_sensor_attr_group = {
    .name   = NULL,
    .attrs      = ioboard_sensor_entries,
};

static int ioboard_sensor_sysfs_init (struct device *dev)
{
    if(sysfs_create_group(&dev->kobj, &ioboard_sensor_attr_group) < 0)
        return  -1;
    return  0;
}
...
```

Data Report Work Enable/Disable

Data Report Work Delay Control

Sensor(BMP180) Driver Function Flow



Android Application : Libsensor(sensor.odroida4.so)(1)

- Libsensor 압축 파일을 device/hardkernel/odroida4/ 폴더에 푼다.
- #source build/envsetup.sh
- #export TARGET_PRODUCT=odroida4
- #mmm device/hardkernel/odroida4/libsensor
- out/target/product/odroida4/system/lib/hw/sensors.odroida4.so가 생성된다.
- adb를 이용하여 기존 파일을 교체한다.

Android Application : Libsensor(sensor.odroida4.so)(2)

- sensor_odroid.cpp
- Android sensor에 callback function 등록

```
static int open_sensors(const struct hw_module_t* module, const char* id,
                        struct hw_device_t** device)
{
    int status = -EINVAL;
    sensors_poll_context_t *dev = new sensors_poll_context_t();

    memset(&dev->device, 0, sizeof(sensors_poll_device_t));

    dev->device.common.tag = HARDWARE_DEVICE_TAG;
    dev->device.common.version  = 0;
    dev->device.common.module   = const_cast<hw_module_t*>(module);
    dev->device.common.close    = poll__close;
    dev->device.activate        = poll__activate;
    dev->device.setDelay        = poll__setDelay;
    dev->device.poll            = poll__poll;

    *device = &dev->device.common;
    status = 0;

    return status;
```

- sensor_odroid.cpp

```
static const struct sensor_t sSensorList[] = {
    { "Ambient Light sensor",
      "ROHM",
      1, SENSORS_LIGHT_HANDLE,
      SENSOR_TYPE_LIGHT, 65535.0f, 1.0f, 0.2f, 10000, { } },
    { "BMP180 Pressure sensor",
       "Bosch",
       1, SENSORS_PRESSURE_HANDLE,
       SENSOR_TYPE_PRESSURE, 1100.0f, 0.01f, 0.67f, 20000, { } },
    { "BMP180 Temperature sensor",
       "Bosch",
       1, SENSORS_TEMPERATURE_HANDLE,
       SENSOR_TYPE_AMBIENT_TEMPERATURE, 600.0f, 1.0f, 0.67f, 20000, { } },
};
```

- Light, Pressure, Temperature 센서를 등록한다.

- sensor_odroid.cpp

```
mSensors[pressure] = new PressureSensor();
mPollFds[pressure].fd = mSensors[pressure]->getFd();
mPollFds[pressure].events = POLLIN;
mPollFds[pressure].revents = 0;

// Temperature는 Pressure의 같은 객체를 사용한다.
mSensors[temperature] = mSensors[pressure];
mPollFds[temperature].fd = mSensors[pressure]->getFd();
mPollFds[temperature].events = POLLIN;
mPollFds[temperature].revents = 0;
```

- PressuresSensor는 Galaxy Nexus(tuna)에 포함되어 있는 SamsungSensorBase를 수정하여 Pressure와 Temperature를 input으로부터 읽어 올 수 있도록 수정.

```
class PressureSensor:public SamsungSensorBase {
    virtual bool handleEvent(input_event const * event);
private:
    int mSensors[numSensors];
public:
    PressureSensor();
};
```

➢ Pressure와 Temperature

```
PressureSensor::PressureSensor()
    : SamsungSensorBase(NULL, "ioboard-sensor", NULL)
{
    mPendingEvent[Pressure].sensor = ID_PR;
    mPendingEvent[Pressure].type = SENSOR_TYPE_PRESSURE;
    mPendingEvent[Temperature].sensor = ID_T;
    mPendingEvent[Temperature].type = SENSOR_TYPE_AMBIENT_TEMPERATURE;
    mSensors[Pressure] = ABS_X;
    mSensors[Temperature] = ABS_Y;
    setSensors(mSensors);
}
```

BMP180 driver의 input name은 ioboard-sensor이다.

SENSOR_TYPE_AMBIENT_TEMPERATURE는 API Level 14이상 지원합니다.

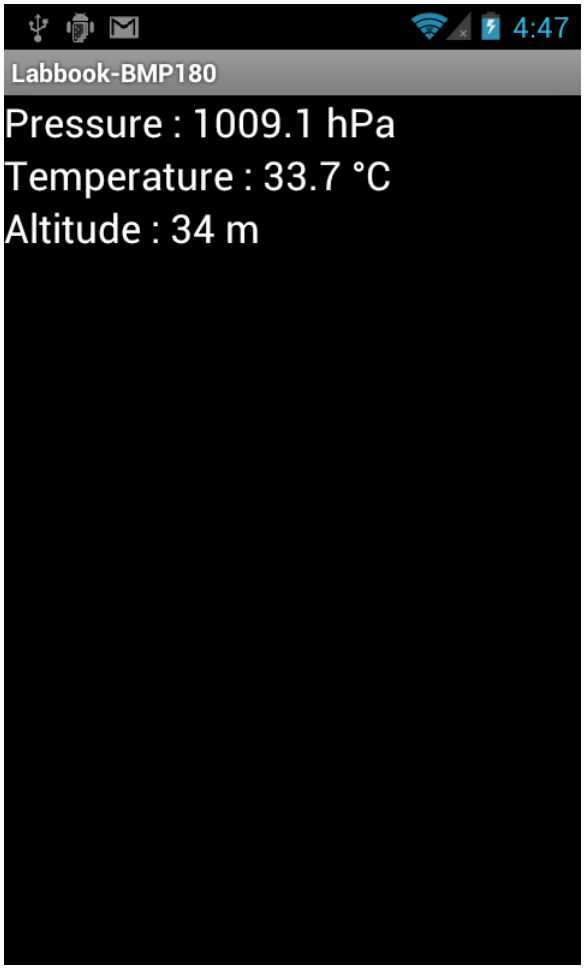
SENSOR_TYPE_TEMPERATURE와 다릅니다.

- PressureSensor

```
bool PressureSensor::handleEvent(input_event const *event) {
    if (event->code == ABS_X) {
        mPendingEvent[Pressure].pressure = event->value * PRESSURE_HECTO;
    } else if (event->code == ABS_Y) {
        mPendingEvent[Temperature].temperature = event->value * (1.0f/10.0f);
    }
    return true;
}
```

handleEvent에 event로 부터 받은 값을 pressure, temperature에 각각 대입하도록 한다.

Android Application : Libsensor(sensor.odroida4.so)(8)



➢ SensorEventListener 구현

```
public class BMP180Activity extends Activity implements SensorEventListener {
    …
mSensorManager =
(SensorManager)getSystemService(Context.SENSOR_SERVICE);
mSensorPressure =
mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
mSensorTemperature =
mSensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATUR
E);
…
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {

        @Override
        public void onSensorChanged(SensorEvent event) {
```

Android Application : Labbook-BMP180(2)

- onSensorChanged

```
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_PRESSURE) {
        mTextViewPressure.setText(String.format("%.1f", event.values[0]) + "
hPa");
        long altitude = (long)(44330.0 * (1.0 - Math.pow((double)(event.values[0] *
100 / 101325.0), 1/5.255)));
//Pressure(hPa)를 이용하여 Altitude 계산
        mTextViewAltitude.setText(altitude + " m");
    } else if (event.sensor.getType() ==
Sensor.TYPE_AMBIENT_TEMPERATURE) {
        mTextViewTemperature.setText(String.format("%.1f", event.values[0]) + "
°C ");
    }
}
```

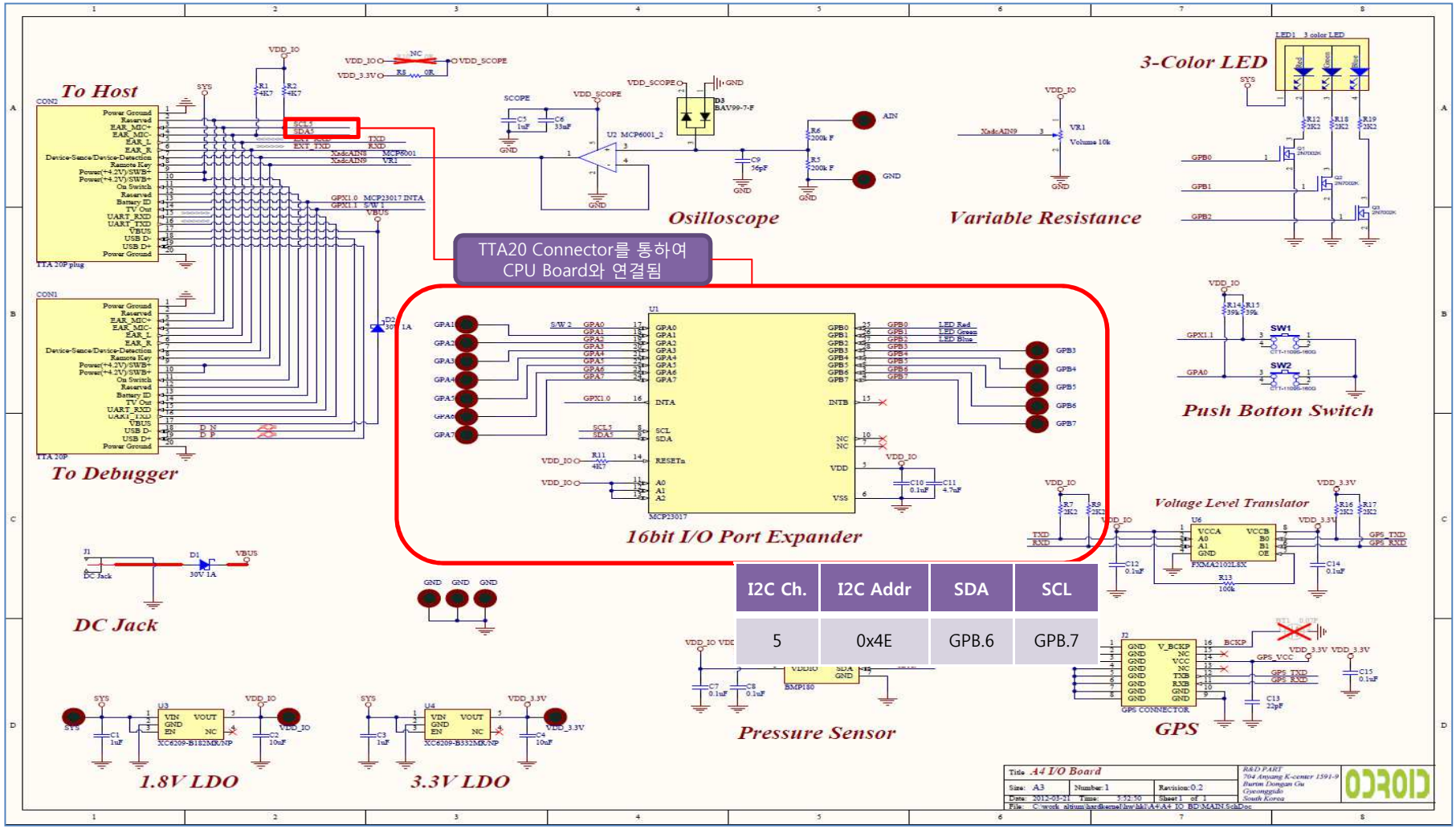
하드웨어 확장 – I2C(Expender)

Agenda

- ODROID-A4 IO Board I2C (Expander) Hardware Interface
- I2C (Expander) Driver
  - Expander(MPC21017) Driver 구조
  - Export Function
  - Expander(MPC21017) I2C driver register
  - Expander Interrupt 설정
  - Expander 정보를 확인하는 SYSFS등록
  - I2C Expander SYSFS를 통한 정보의 표시
- I2C (Expander) Driver를 사용한 Key Input Driver
  - Key Driver Probe
- I2C (Expander) Driver를 사용한 LED Driver
  - Led Driver Probe
  - Led Control SYSFS
- Android Application : Labbook-MCP23017

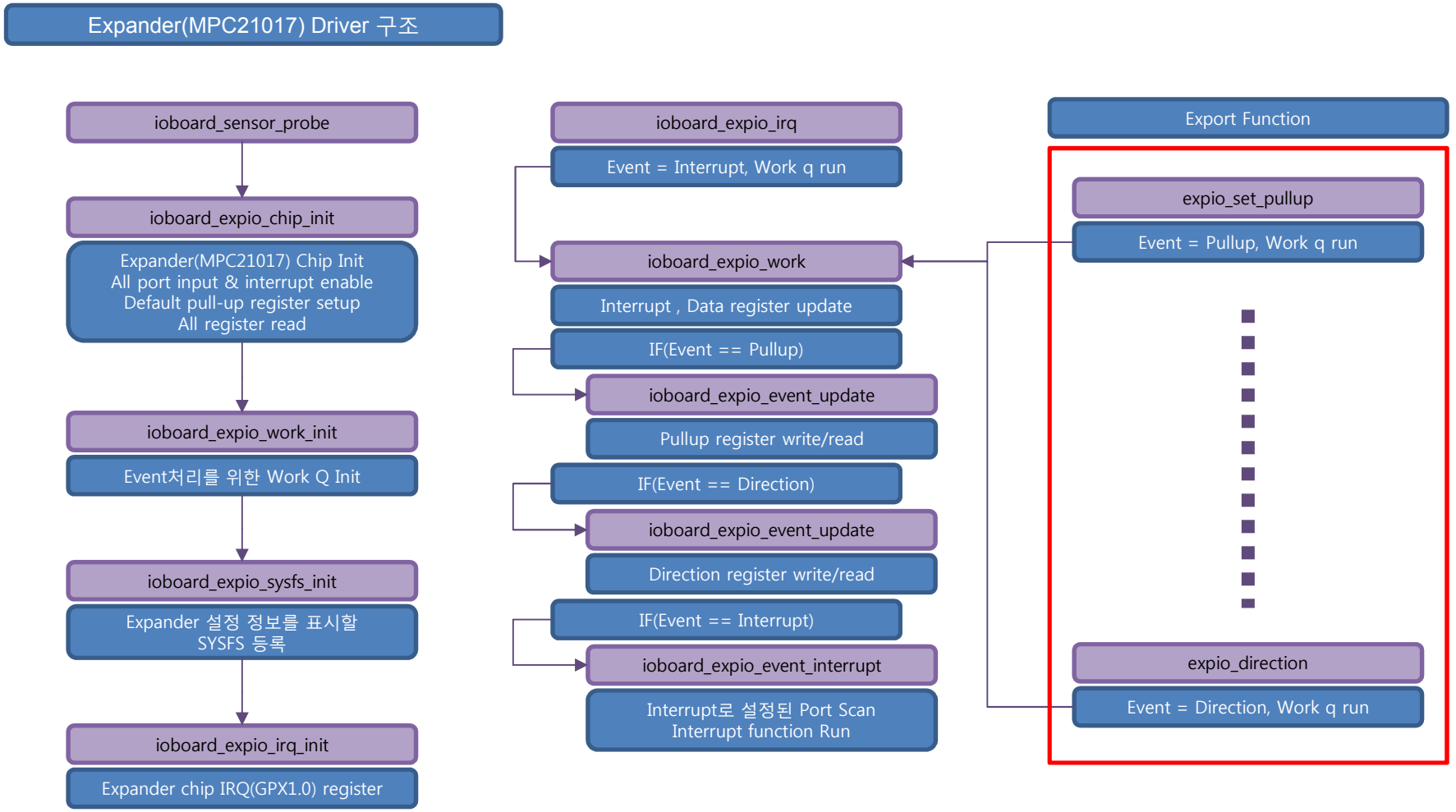
ODROID-A4 IO Board I2C (Expander) Hardware Interface

Odroid-A4 I/O Board의 Expander 연결



I2C Ch.	I2C Addr	SDA	SCL
5	0x4E	GPB.6	GPB.7

I2C (Expander) Driver(1)



I2C (Expander) Driver(2)

Export Function - 1

Function
int expio_request (int expio_bit, const char *label)
Description
Expander 해당 Bit를 사용하도록 요청한다.
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_request (EXPIO_GPA0, "expio gpio);**

Function
int expio_free (int expio_bit)
Description
Expander 해당 Bit의 사용을 해제한다..
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_free (EXPIO_GPA0);**

Function
int expio_get_value (int expio_bit)
Description
Expander 해당 Bit의 상태를 요청한다.
Return
해당 Bit의 상태가 1인 경우 1, 0 이면 0을 return함..
Usage
ret = **expio_get_value (EXPIO_GPA0);**

Function
int expio_set_value (int expio_bit, bool level)
Description
Expander 해당 Bit의 출력 상태를 level값으로 설정한다..
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_set_value (EXPIO_GPA0, 1);**

I2C (Expander) Driver(3)

Export Function - 2

Function
int expio_direction (int expio_bit, bool output)
Description
Expander 해당 Bit를 입력 또는 출력으로 사용하도록 요청한다.
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_direction (EXPIO_GPA0, 1); // output mode로 설정**

Function
int expio_direction_input (int expio_bit)
Description
Expander 해당 Bit를 Input으로 사용하도록 요청한다.
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_direction_input (EXPIO_GPA0);**

Function
int expio_direction_output (int expio_bit, bool level)
Description
Expander 해당 Bit를 출력으로 설정하고 level값으로 초기값을 설정한다.
Return
해당 Bit의 상태가 1인 경우 1, 0 이면 0을 return함..
Usage
ret = **expio_direction_output (EXPIO_GPA0, 1); // output mode이고 출력은 1로 설정한다.**

Function
int expio_set_pullup (int expio_bit, bool enable)
Description
Expander 해당 Bit의 내부 Pull-up Resister 상태를 설정한다..
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_set_pullup (EXPIO_GPA0, 1); // GPA0 Bit의 내부 Pull-up Resister를 활성화 시킨다.**

I2C (Expander) Driver(4)

Export Function - 3

Function
int expio_request_irq (int expio_bit, void (*func)(int irq, void *handle), int irq_trigger, const char *label, void *handle)
Description
Expander 해당 Bit를 입력으로 설정 및 Interrupt로 사용하도록 요청한다.
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_request_irq(EXPIO_GPA0, expio_irq_func, EXPIO_IRQ_TRIGGER_BOTH, "expio irq", NULL);**

Function
int expio_free_irq (int expio_bit)
Description
Expander 해당 Bit의 IRQ 및 Port 사용을 해제한다.
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_free_irq (EXPIO_GPA0);**

Function
int expio_enable_irq (int expio_bit)
Description
Expander 해당 Bit의 IRQ를 Enable 한다.
Return
해당 Bit의 상태가 1인 경우 1, 0 이면 0을 return함..
Usage
ret = **expio_enable_irq (EXPIO_GPA0);**

Function
int expio_disable_irq (int expio_bit, bool level)
Description
Expander 해당 Bit의 IRQ를 Disable 한다..
Return
에러인 경우 -1 , 유효한 경우 0을 return함.
Usage
ret = **expio_disable_irq (EXPIO_GPA0);**

I2C (Expander) Driver(5)

Expander(MPC21017) I2C driver register

Kernel/drivers/misc/odroida4-io/i2c/ioboard-expio.c

```
...
static const struct i2c_device_id ioboard_expio_id[] = {
    { IOBOARD_EXPIO_NAME, 0  },
    {},
};

MODULE_DEVICE_TABLE(i2c, ioboard_expio_id);

struct i2c_driver ioboard_expio_driver ={
    .driver = {
        .name   = IOBOARD_EXPIO_NAME,
        .owner  = THIS_MODULE,
    },
    .probe      = ioboard_expio_probe,
    .remove     = ioboard_expio_remove,
    .suspend    = ioboard_expio_suspend,
    .resume     = ioboard_expio_resume,

    .id_table   = ioboard_expio_id,
};
...
static  int ioboard_expio_probe(struct i2c_client *client,
                                const struct i2c_device_id *id)
{
...
}
...
```

Platform driver probe를 위하여 Driver name을 동일하게 설정한다.

Kernel/drivers/misc/odroida4-io/i2c/ioboard-expio.h

```
...
#define IOBOARD_EXPIO_NAME      "ioboard-expio"
...
struct ioboard_expio_pdata  {
    unsigned short     def_pullup;
};

#define SET_PULLUP(x)           (1 << x)
...
```

Default Pull-up Resister설정

Kernel/arch/arm/mach-exynos/mach-odroid-4210.c

```
...
#include <../../../drivers/misc/odroida4-io/i2c/ioboard-expio.h>

struct ioboard_expio_pdata      ioboard_expio_pdata = {
    .def_pullup = SET_PULLUP(EXPIO_GPA1) | SET_PULLUP(EXPIO_GPA2) |
                  SET_PULLUP(EXPIO_GPA3) | SET_PULLUP(EXPIO_GPA4) |
    ...
};

static struct i2c_board_info i2c_gpio_devs[] __initdata = {
    {
        I2C_BOARD_INFO(IOBOARD_EXPIO_NAME, (0x4E>>1)),
        .irq = EXYNOS4_GPX1(0),
        .platform_data = &ioboard_expio_pdata,
    },
};
...
static void __init smdkv310_machine_init(void)
{
...
    i2c_register_board_info(5, i2c_gpio_devs, ARRAY_SIZE(i2c_gpio_devs));
...
}
...
```

Adaptor에 연결된 I2C Device의 정보

Expander에 IRQ로 연결되어 있는 GPIO설정

I2C Adaptor 5번(GPIO-I2C)에 연결되어 있는 Device를 등록한다.

I2C (Expander) Driver(6)

Expander Interrupt 설정

Kernel/drivers/misc/odroida4-io/i2c/ioboard-sensor.c

```
...
irqreturn_t ioboard_expio_irq(int irq, void *handle)
{
    struct  ioboard_expio *ioboard_expio = handle;

    ioboard_expio->event |= EVENT_INTERRUPT;
    queue_work(ioboard_expio->work_queue, &ioboard_expio->work);
    return IRQ_HANDLED;
}
...
static int ioboard_expio_irq_init(struct ioboard_expio *ioboard_expio)
{
    if(request_irq(gpio_to_irq(ioboard_expio->client->irq),
                   ioboard_expio_irq,
                   IRQF_DISABLED | IRQF_TRIGGER_FALLING,
                   ioboard_expio->client->name,
                   ioboard_expio))                  {
        return  -1;
    }
    return  0;
}
...
static int ioboard_expio_probe(struct i2c_client *client,
                               const struct i2c_device_id *id)
{
...
    if(ioboard_expio_irq_init(ioboard_expio))    goto err_driver_init;.
...
}
...
```

Register Update를 위한 Work Q 실행

Platform에서 정의한 GPIO(GPX1.0)를 IRQ로 전환

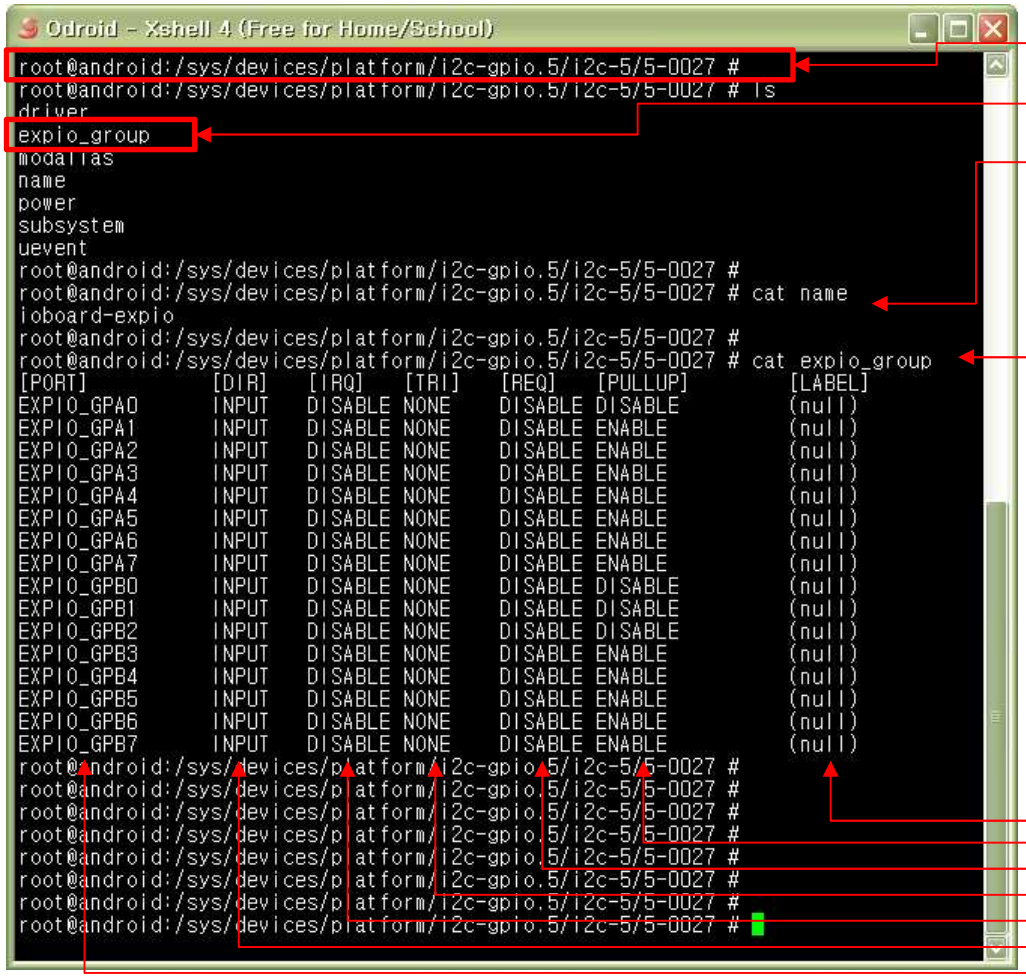
Expander 정보를 확인하는 SYSFS 등록

Kernel/drivers/misc/odroida4-io/i2c/ioboard-sensor.c

```
...
static ssize_t show_expio_group(struct device *dev,
                                struct device_attribute *attr,
                                char *buf);

static DEVICE_ATTR(expio_group, S_IRWXUGO, show_expio_group,   NULL);

static struct attribute *ioboard_expio_entries[] = {
    &dev_attr_expio_group.attr,
    NULL
};

static struct attribute_group ioboard_expio_attr_group = {
    .name   = NULL,
    .attrs    = ioboard_expio_entries,
};

static int ioboard_expio_sysfs_init(struct device *dev)
{
    if(sysfs_create_group(&dev->kobj, &ioboard_expio_attr_group) < 0)    {
        return  -1;
    }
    return  0;
}

static int ioboard_expio_probe(struct i2c_client *client,
                               const struct i2c_device_id *id)
{
...
    if(ioboard_expio_sysfs_init(&ioboard_expio->client->dev))
                                                goto err_sysfs_init;
...
}
...
```

I2C (Expander) Driver(7)

I2C Expander SYSFS를 통한 정보의 표시



SYSFS 생성경로 : i2c-gpio의 5번 채널이고 ID는 0x27(0x4E>>1)

SYSFS 생성 이름

Device Driver의 이름 정보

Expander 정보 표시

등록 이름

내부 Pull-up 상태

PORT 사용 여부

IRQ Trigger 상태

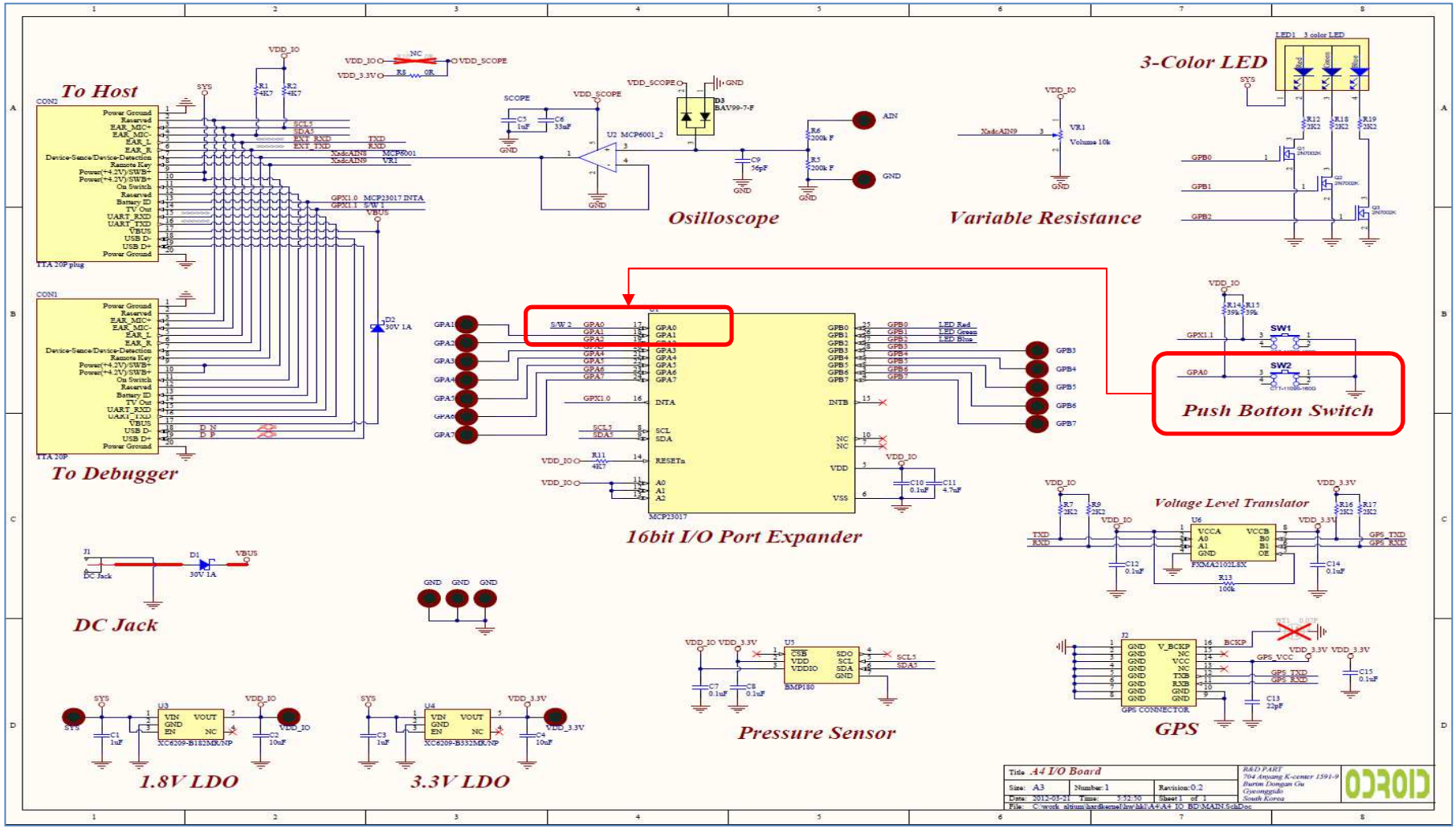
IRQ 등록 여부

각 Port의 상태

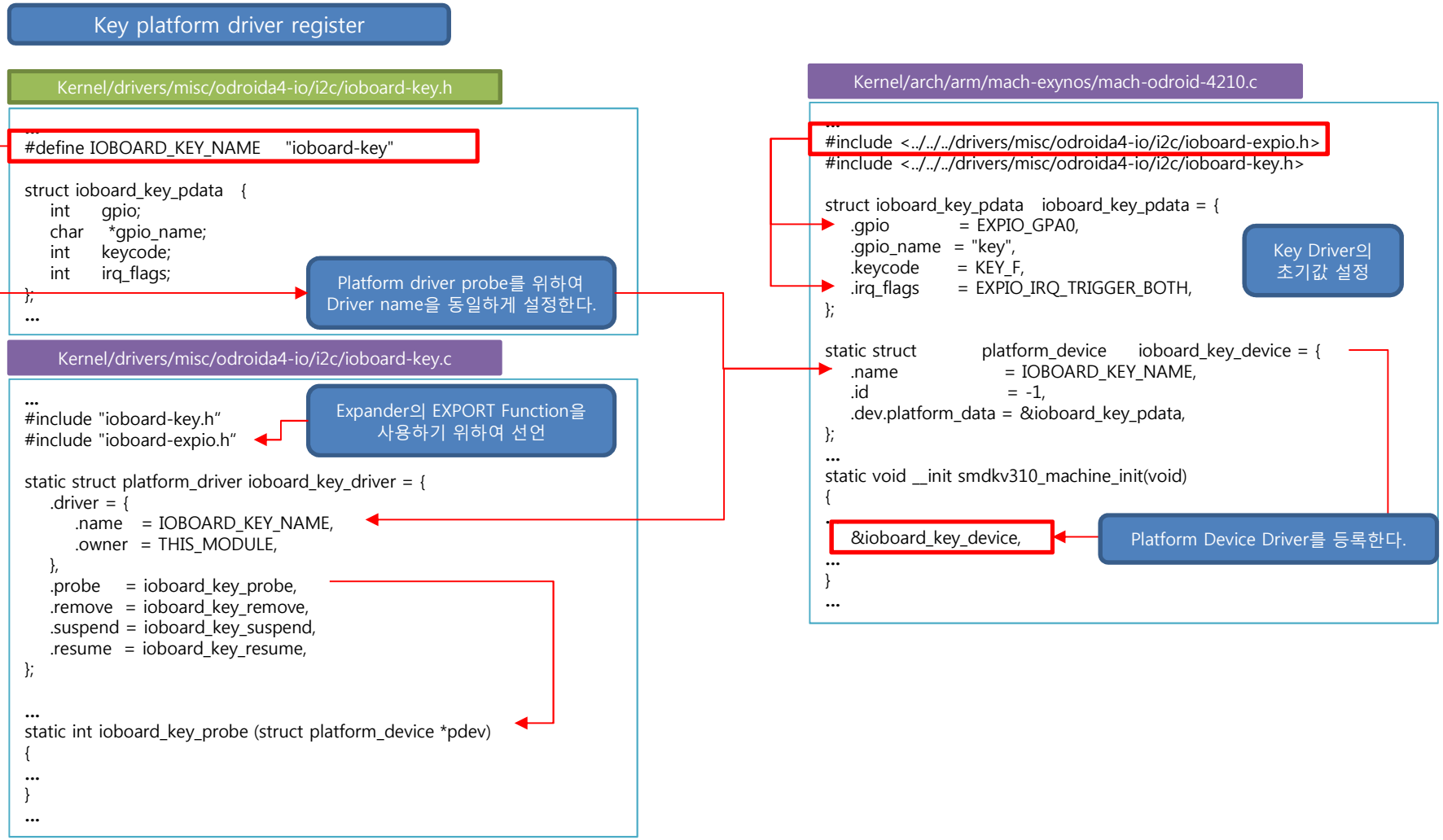
Expander의 Port 이름

I2C (Expander) Driver를 사용한 Key Input Driver(1)

SW2와 Expander 연결



I2C (Expander) Driver를 사용한 Key Input Driver(2)



I2C (Expander) Driver를 사용한 Key Input Driver(3)

Key Interrupt Register

Kernel/drivers/misc/odroida4-io/i2c/ioboard-key.c

```
...
#include "ioboard-key.h"
#include "ioboard-expio.h"
...
static void ioboard_key_irq (int irq, void *handle)
{
    struct  ioboard_key  *ioboard_key = handle;

    if(expio_get_value(ioboard_key->pdata->gpio))
        input_report_key(ioboard_key->input,
                         ioboard_key->pdata->keycode, false);
    else
        input_report_key(ioboard_key->input,
                         ioboard_key->pdata->keycode, true);

    input_sync(ioboard_key->input);
}
...
static int ioboard_key_irq_init(struct ioboard_key *ioboard_key)
{
    if(expio_request_irq ( ioboard_key->pdata->gpio,
                           ioboard_key_irq,
                           ioboard_key->pdata->irq_flags,
                           ioboard_key->pdata->gpio_name,
                           ioboard_key ))
        return -1;
...
}
...
static int ioboard_key_probe (struct platform_device *pdev)
{
...
    if(ioboard_key_irq_init(ioboard_key))     goto err_irq_init;
...
}
...
```

Expander Driver에 등록되어 있는 IRQ Handler를 실행한다.

Export되어 있는 함수를 통하여 IRQ를 등록한다.

Kernel/drivers/misc/odroida4-io/i2c/ioboard_expio.c

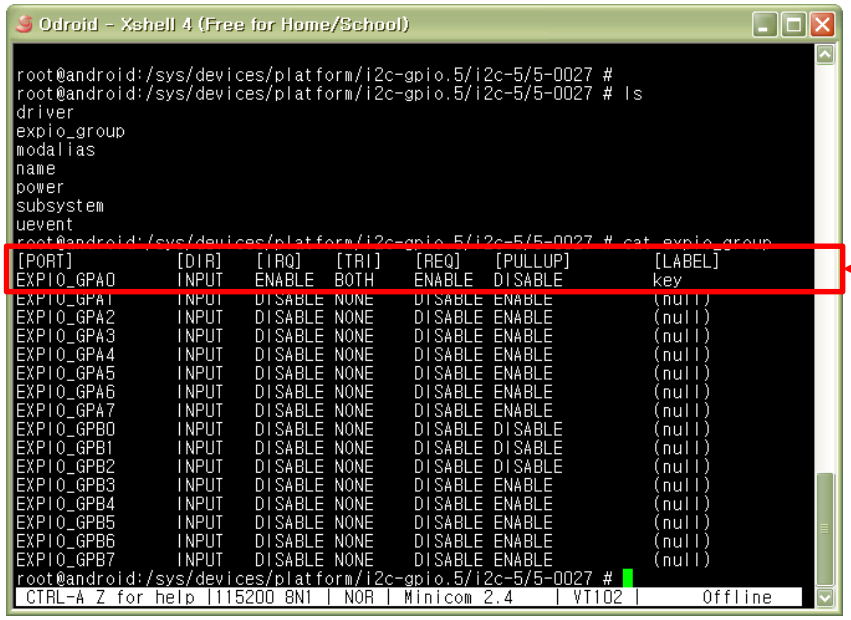
```
...
static void ioboard_expio_event_interrupt
                        (struct ioboard_expio *ioboard_expio)
{
...
    if(ioboard_expio->bit[i].irq_enabled &&
       ioboard_expio->bit[i].irq_func)  {

        switch(ioboard_expio->bit[i].irq_trigger) {
            case    EXPIO_IRQ_TRIGGER_BOTH:
                ioboard_expio->bit[i].irq_func(i,
                                ioboard_expio->bit[i].irq_handle);
                break;
...
}
...
static void  ioboard_expio_work (struct work_struct *work)
{
...
    if(ioboard_expio->event & EVENT_INTERRUPT)  {
        ioboard_expio_event_interrupt(ioboard_expio);
        ioboard_expio->event &= (~EVENT_INTERRUPT);
    }
...
}
...
irqreturn_t ioboard_expio_irq (int irq, void *handle)
{
    ioboard_expio->event |= EVENT_INTERRUPT;
    queue_work(ioboard_expio->work_queue, &ioboard_expio->work);
}

int expio_request_irq (int expio_bit,  void (*func)(int irq, void *handle),
                       int irq_trigger, const char *label,  void *handle)
{
...
}
...
```

Interrupt 등록 상태를 검사하고 Trigger에 맞추어 등록되어진 IRQ Function을 실행한다.

Expander Input Port의 상태가 변경되는 경우 Interrupt가 발생한다.

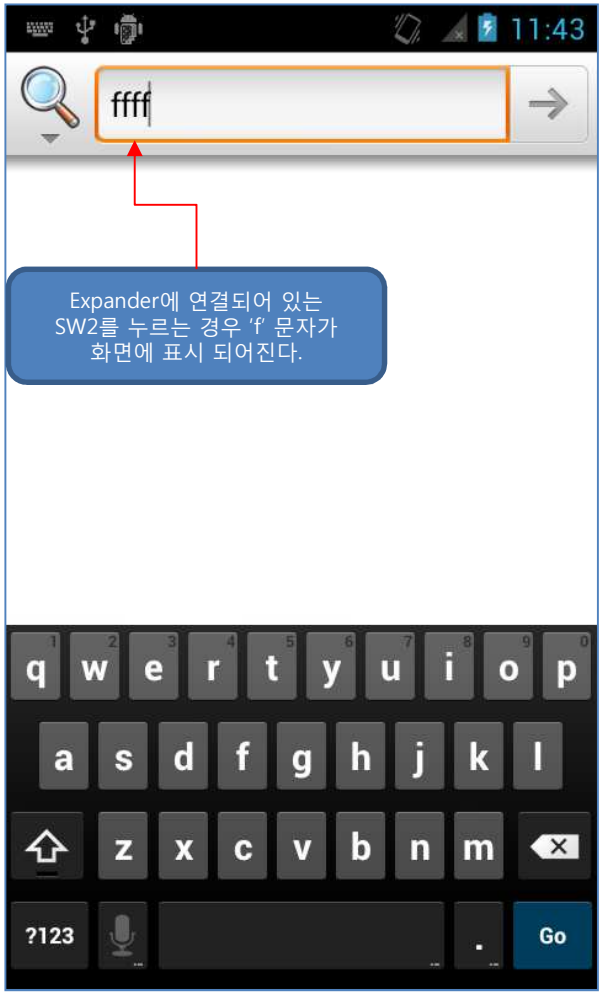
I2C (Expander) Driver를 사용한 Key Input Driver(4)

I2C Expander SYSFS를 통한 정보의 표시



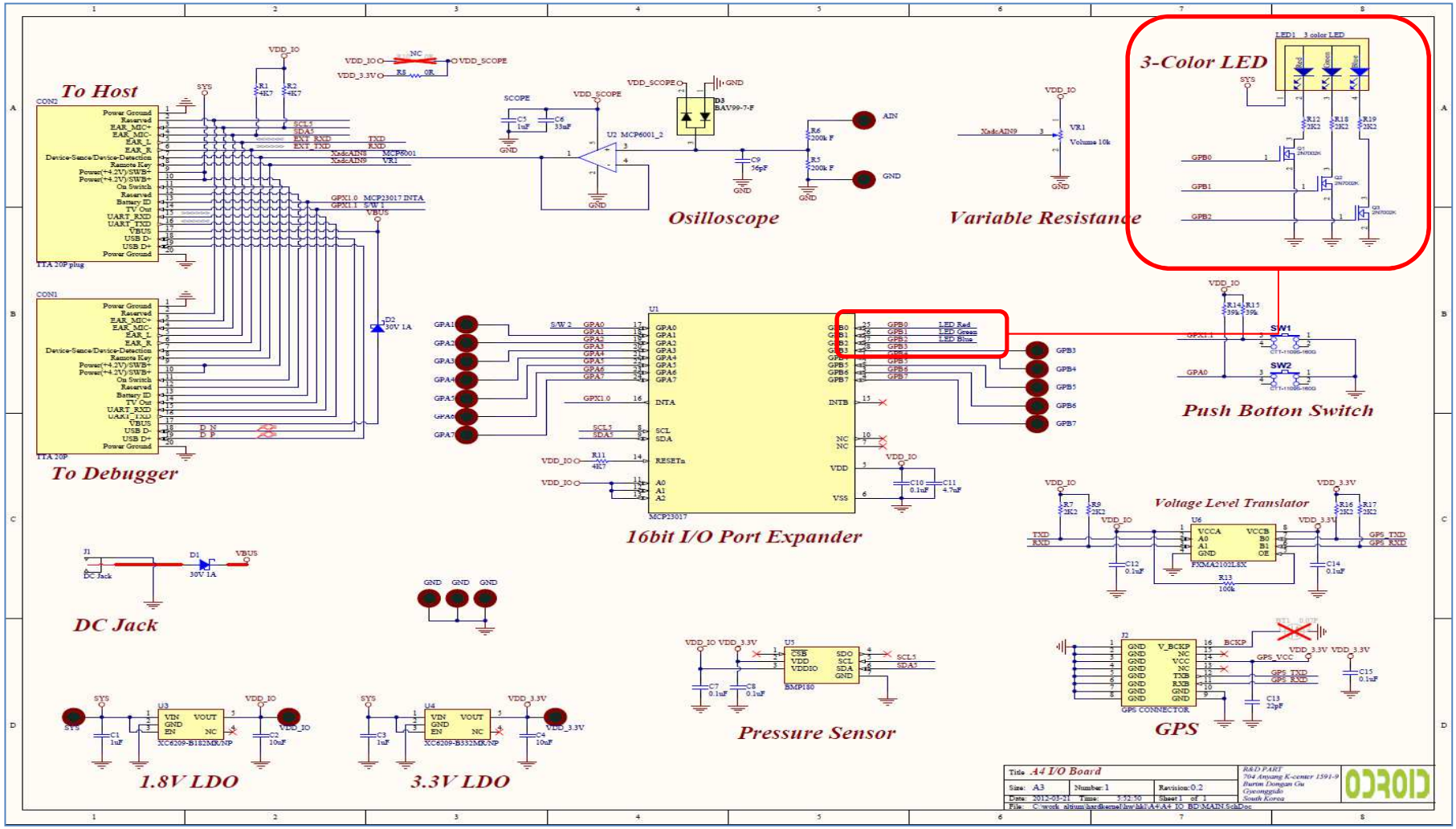
EXPIO_GPA0 핀의 상태를 확인 하여
정상적으로 등록되어 있는지 확인한다.

Key Driver의 동작 검증

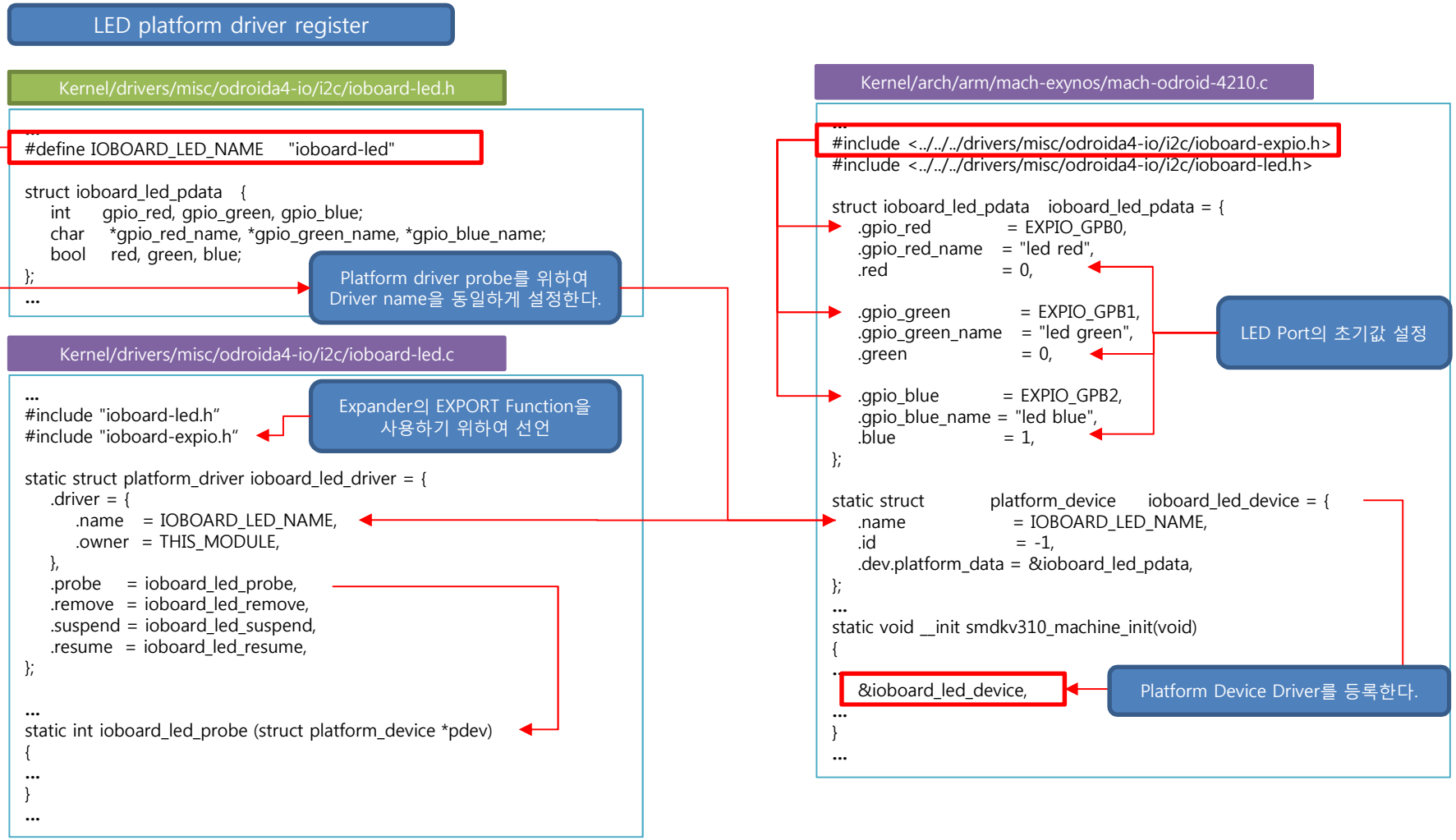


I2C (Expander) Driver를 사용한 LED Driver(1)

LED와 Expander 연결



I2C (Expander) Driver를 사용한 LED Driver(2)



I2C (Expander) Driver를 사용한 LED Driver(3)

LED Output Register

Kernel/drivers/misc/odroida4-io/i2c/ioboard-led.c

```
...
#include "ioboard-key.h"
#include "ioboard-expio.h"
...
static  int     ioboard_led_output_init (struct ioboard_led *ioboard_led)
{
    if(expio_request(ioboard_led->pdata->gpio_red,
                    ioboard_led->pdata->gpio_red_name))     goto err_red;
    if(expio_request(ioboard_led->pdata->gpio_green,
                    ioboard_led->pdata->gpio_green_name))   goto err_green;
    if(expio_request(ioboard_led->pdata->gpio_blue,
                    ioboard_led->pdata->gpio_blue_name))    goto err_blue;

    expio_direction_output(ioboard_led->pdata->gpio_red,
                                ioboard_led->pdata->red);
    expio_direction_output(ioboard_led->pdata->gpio_green,
                                ioboard_led->pdata->green);
    expio_direction_output(ioboard_led->pdata->gpio_blue,
                                ioboard_led->pdata->blue);

    return  0;

err_blue:
    expio_free(ioboard_led->pdata->gpio_green);
err_green:
    expio_free(ioboard_led->pdata->gpio_red);
err_red:
    return  -1;
}
...
static int ioboard_led_probe (struct platform_device *pdev)
{
...
    if(ioboard_led_output_init(ioboard_led))     goto err_output_init;
...
}
...
```

Export되어 있는 함수를 통하여 GPIO를 할당 받고 초기값을 설정한다.

LED Control SYSFS

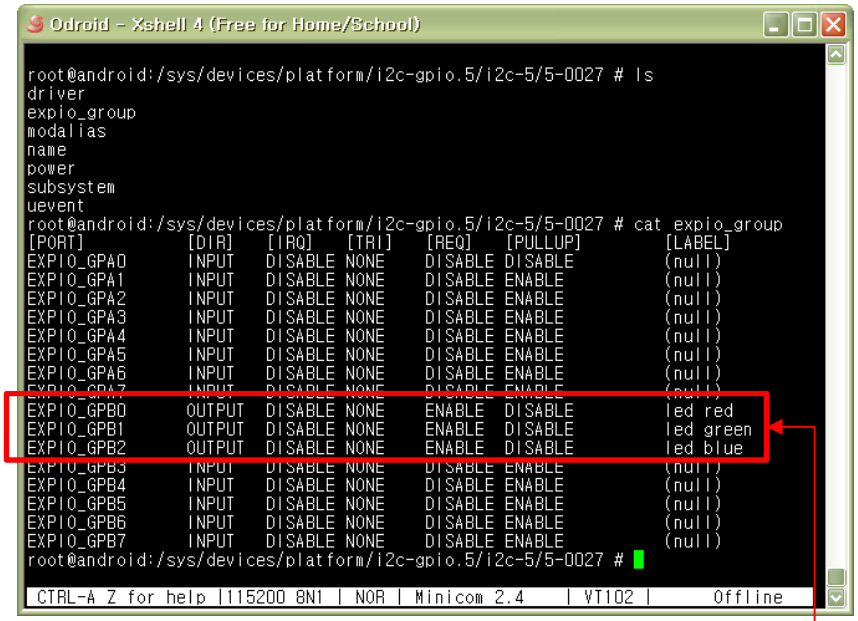
Kernel/drivers/misc/odroida4-io/i2c/ioboard_led.c

```
static ssize_t show_expio   (struct device *dev,
                             struct device_attribute *attr, char *buf);
static ssize_t store_expio (struct device *dev,
                            struct device_attribute *attr, const char *buf,
                            size_t count);

static DEVICE_ATTR(red,    S_IRWXUGO, show_expio,   store_expio);
static DEVICE_ATTR(green, S_IRWXUGO, show_expio,   store_expio);
static DEVICE_ATTR(blue,   S_IRWXUGO, show_expio,   store_expio);

static struct attribute *ioboard_led_entries[] = {
    &dev_attr_red.attr,
    &dev_attr_green.attr,
    &dev_attr_blue.attr,
    NULL
};

static struct attribute_group ioboard_led_attr_group = {
    .name    = NULL,
    .attrs     = ioboard_led_entries,
};

static  int     ioboard_led_sysfs_init  (struct platform_device *pdev)
{
    if(sysfs_create_group(&pdev->dev.kobj, &ioboard_led_attr_group) < 0)
        return  -1;
    return  0;
}
...
```

Expander의 각 Bit를 Control하는 SYSFS

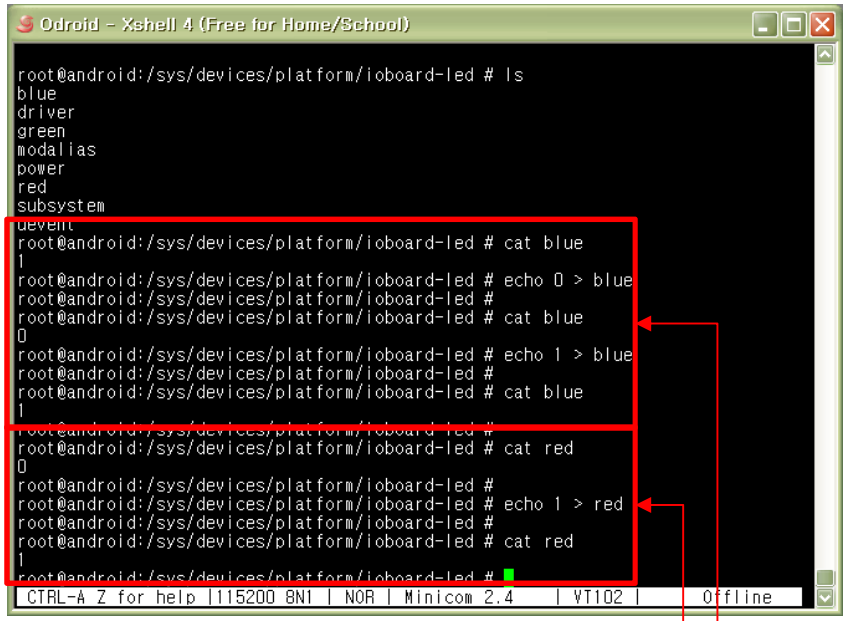
I2C (Expander) Driver를 사용한 LED Driver(4)

I2C Expander SYSFS를 통한 정보의 표시



LED와 연결된 PORT의 상태를 확인 하여 정상적으로 등록되어 있는지 확인한다.

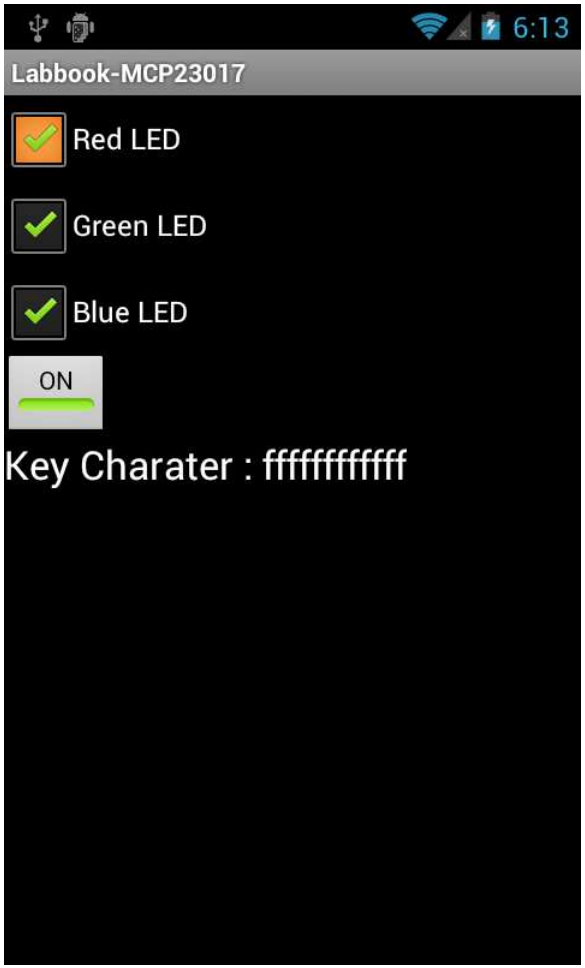
LED Driver의 SYSFS를 통한 LED Control



RED LED의 상태 확인 및 변경
LED상태가 변경되는지 확인한다.

BLUE LED의 상태 확인 및 변경
LED상태가 변경되는지 확인한다.

Android Application : Labbook-MCP23017(1)



Android Application : Labbook-MCP23017(2)

➢ **JNI function을 만든다.**

```
static {
        System.loadLibrary("MCP23017DriverSysfs");
  }

native int checkKeyNode();                  //key sysfs 노드가 있는지 확인
native int checkLEDNode();                  //led sysfs 노드가 있는지 확인
native int getKeyState();                   //key enable/disable 확인
native void setKeyState(int enable);        //key enable/disable 변경
native int readLED(int led);                //LED enable/disable 확인
native void setLED(int led, int onoff);     //LED enable/disable 변경
```

- **GPIO Key와 동일하게 onKeyDown/Up function에서 key 입력을 확인한다.**

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
        // TODO Auto-generated method stub
        if (keyCode == KeyEvent.KEYCODE_F)
                mTVKeyCharater.append("f");
        return super.onKeyDown(keyCode, event);
}

public boolean onKeyUp(int keyCode, KeyEvent event) {
        // TODO Auto-generated method stub
        if (keyCode == KeyEvent.KEYCODE_F)
                mTVKeyCharater.append("f");
        return super.onKeyUp(keyCode, event);
}
```

➢ **Check Box check/uncheck시 setLED ()함수를 호출한다.**

```
mCBRed.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
                // TODO Auto-generated method stub
                if (isChecked)
                        setLED(RED_LED, 1);
                else
                        setLED(RED_LED, 0);
        }
});
```

- **Toggle Button toggle시** setKeyState**()함수를 호출한다.**

```
mTBKeyEnable = (ToggleButton)findViewById(R.id.tb_key_enable);
mTBKeyEnable.setOnCheckedChangeListener(new OnCheckedChangeListener() {

        @Override
        public void onCheckedChanged(CompoundButton buttonView,
                        boolean isChecked) {
                        // TODO Auto-generated method stub
                        if (isChecked)
                                        setKeyState(1);
                        else
                                        setKeyState(0);
        }
});
```