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

RC-CAR let’s get it

SPI Petalinux

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

목차

1 ZYNQ SPI

spi controller 에 연결할 fpga 포트를 만든다. Spi 마스터는 spi 장치에 write 하기 위한 Chip select, clock, data 출력들이 필요 하다.

만약에 spi 장치로부터 읽기를 할 경우에는 data 입력이 필요하다.

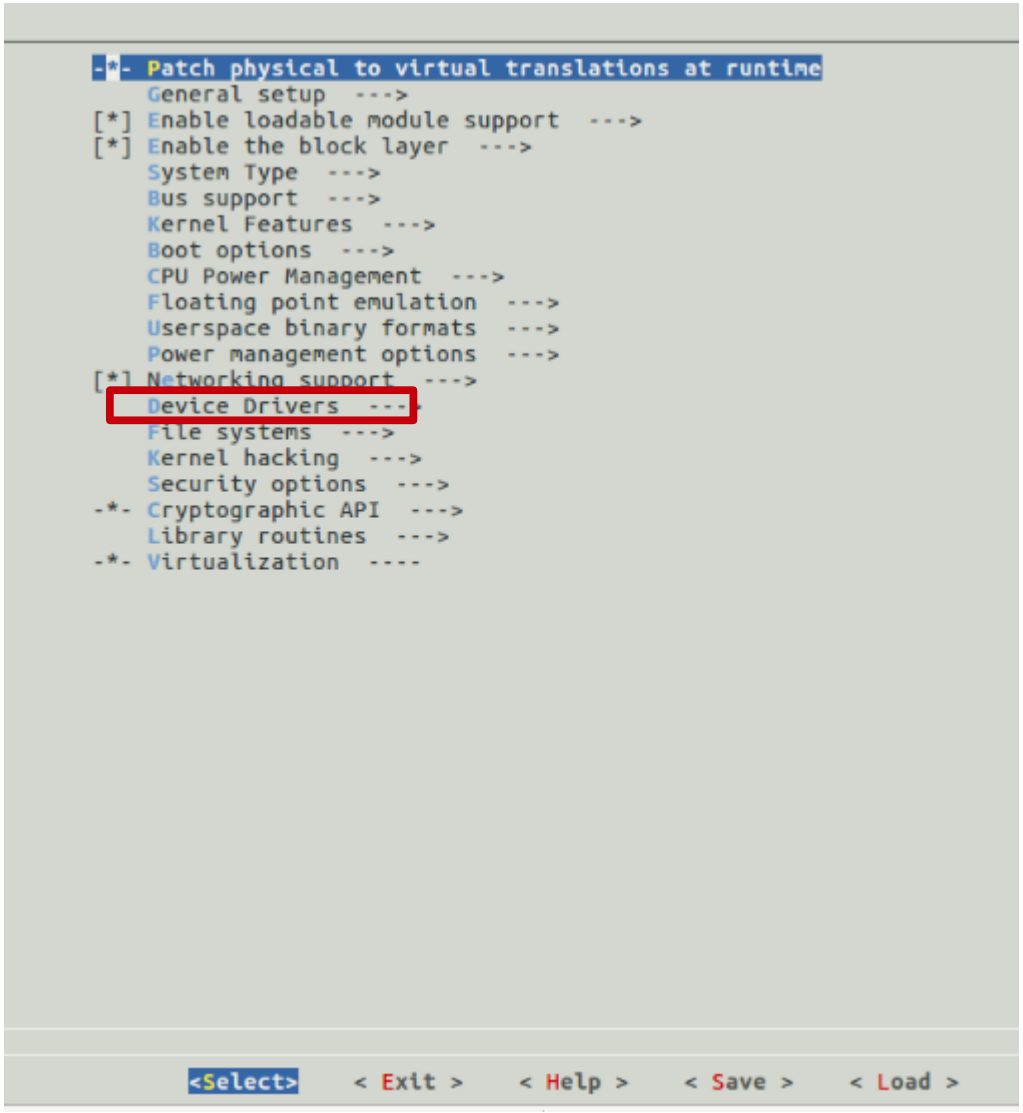
1.	SPI Controller	Zynq and Zynq Ultrascale+ MPSoC	Zynq SPI Driver	Yes	drivers/spi/spi-cadence.c
2.	SPI/QSPI Controller	axi_spi/axi_quad_spi	SPI Driver	Yes	drivers/spi/spi-xilinx.c

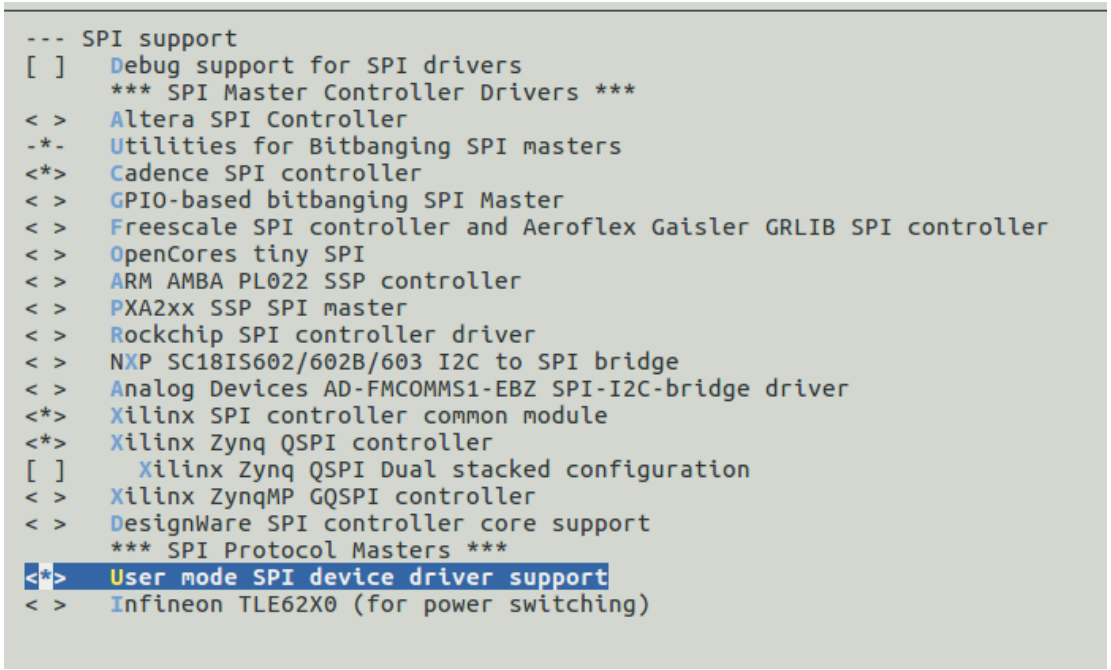
Xilnx 에서 zynq processor 의 spi driver를 제공한다. Spi-cadence.c 드라이버이다.

다음은 유저영역에서 spi통신으로 Pmod CAN 모듈을 제어하기위해 소프트웨어 단을 만들어주는 과정이다.

$ petalinux-config --get-hw-description=~/workspace_v8/spi_proj/hardware/spi_can.sdk

1. Softwere 생성한다.
$ petalinux-create -t project -n testpetalinux --template zynq
2. 하드웨어 정보 커널에 넣고
3. 커널구성 → zimage 생성
$ petalinux-conifg -c kernel
사용자 모드 SPI 지원을 활성화하지 않으면 SPI 장치 파일이 생성되지 않는다.


-*- Patch physical to virtual translations at runtime
General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
System Type --->
Bus support --->
Kernel Features --->
Boot options --->
CPU Power Management --->
Floating point emulation --->
Userspace binary formats --->
Power management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Kernel hacking --->
Security options --->
-*- Cryptographic API --->
Library routines --->
-*- Virtualization ----
<Select> < Exit > < Help > < Save > < Load >

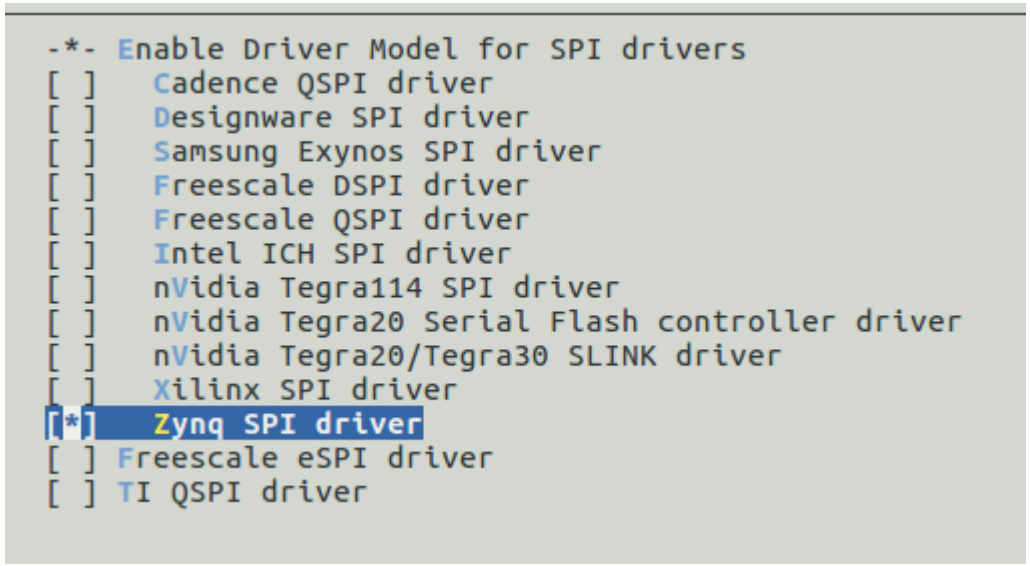

--- SPI support
[ ] Debug support for SPI drivers
*** SPI Master Controller Drivers ***
< > Altera SPI Controller
-*- Utilities for Bitbanging SPI masters
<*> Cadence SPI controller
< > GPIO-based bitbanging SPI Master
< > Freescale SPI controller and Aeroflex Gaisler GRLIB SPI controller
< > OpenCores tiny SPI
< > ARM AMBA PL022 SSP controller
< > PXA2xx SSP SPI master
< > Rockchip SPI controller driver
< > NXP SC18IS602/602B/603 I2C to SPI bridge
< > Analog Devices AD-FMCOMMS1-EBZ SPI-I2C-bridge driver
<*> Xilinx SPI controller common module
<*> Xilinx Zynq QSPI controller
[ ] Xilinx Zynq QSPI Dual stacked configuration
< > XilinxZynqMP GQSPI controller
< > DesignWare SPI controller core support
*** SPI Protocol Masters ***
<*> User mode SPI device driver support
< > Infineon TLE62X0 (for power switching)

4. u-boot 구성 (부트로더 종류 중 하나)

petalinux 프로그램은 부트로더로 u-boot를 사용한다. (부트로더는 운영체제가 시동 되기 이전에 미리 실행되면서 커널이 올바르게 시동 되기 위해 필요한 모든 관련 작업을 마무리 하고 최종적으로 운영체제를 시동 시키기 위한 목적을 가진 프로그램을 말합니다. 즉, 메모리, 하드웨어(네트워크, 프로세서 속도, 인터럽트), 코드/데이터/스택 영역 설정 및 초기화, 커널 로더와 커널 이미지를 로딩, 커널 로더를 실행하여 커널 이미지가 실행되도록 합니다.)

압축된 커널의 이미지인 zimage에 하드웨어 초기화 및 디바이스 드라이버 enable 등의 여러 동작들을 추가하여 uimage를 만든다.
$ petalinux-config -c u-boot

Device Drviers ----→ SPI Support ---> Zynq SPI driver enable



5. spi 장치파일을 만들기 위해 device tree를 수정한다.

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/workspace_v8/spi_proj/testpetalinux/subsystems/linux/configs/device-tree$ ls
pcw.dtsi  pl.dtsi  skeleton.dtsi  system-conf.dtsi  system-top.dts  zynq-7000.dtsi
```

Subsystem/linux/configs/device-tree 디렉터리
zynq-spi를 사용하므로 zynq-7000.dtsi 파일을 확인한다.

```
spi0: spi@e0006000 {
    compatible = "xlnx,zynq-spi-r1p6";
    reg = <0xe0006000 0x1000>;
    status = "disabled";
    interrupt-parent = <&intc>;
    interrupts = <0 26 4>;
    clocks = <&clkc 25>, <&clkc 34>;
    clock-names = "ref_clk", "pclk";
    #address-cells = <1>;
    #size-cells = <0>;
};
```

장치의 주소가 0006000이고, 디바이스 타입으로 spi 라는 장치를 나타내고 있다. Spi0: 는 spi 이라는 라벨을 등록한다. Dtb 파일을 만들 때, 다른 디바이스 트리파일에서 &spi 으로 포인터 주소처럼 사용하여 참조 하도록 표현하는 것이다. compatible 속성을 사용한 빨간 네모칸 “xlnx,zynq-spi-r1p6” 은 디바이스 드라이버의 정보를 찾기 위한 키 값이며, 이를 통해 운영체제가 디바이스 드라이버를 찾는다.

또한 spi의 부모 노드는 속성의 키로 "simple-bus" 을 가르킨다. 그러면 플랫폼 디바이스로 등록이 된다. 그러므로 디바이스로 등록시에 reg 의 정보와 IRQ의 리소스 정보도 같이 수집하여 등록된다.

위에서 Zynq-spi 가 사용하는 드라이버는 spi-cadence.c 라고 했다. 다음 petalinux kernel의 drivers 디렉터리로 가서 디바이스 드라이버를 살펴보자.

Path: ~/petalinux_zynq/petalinux-v2015.4-final/components/linux-kernel/xlnx-4.0/drivers/spi

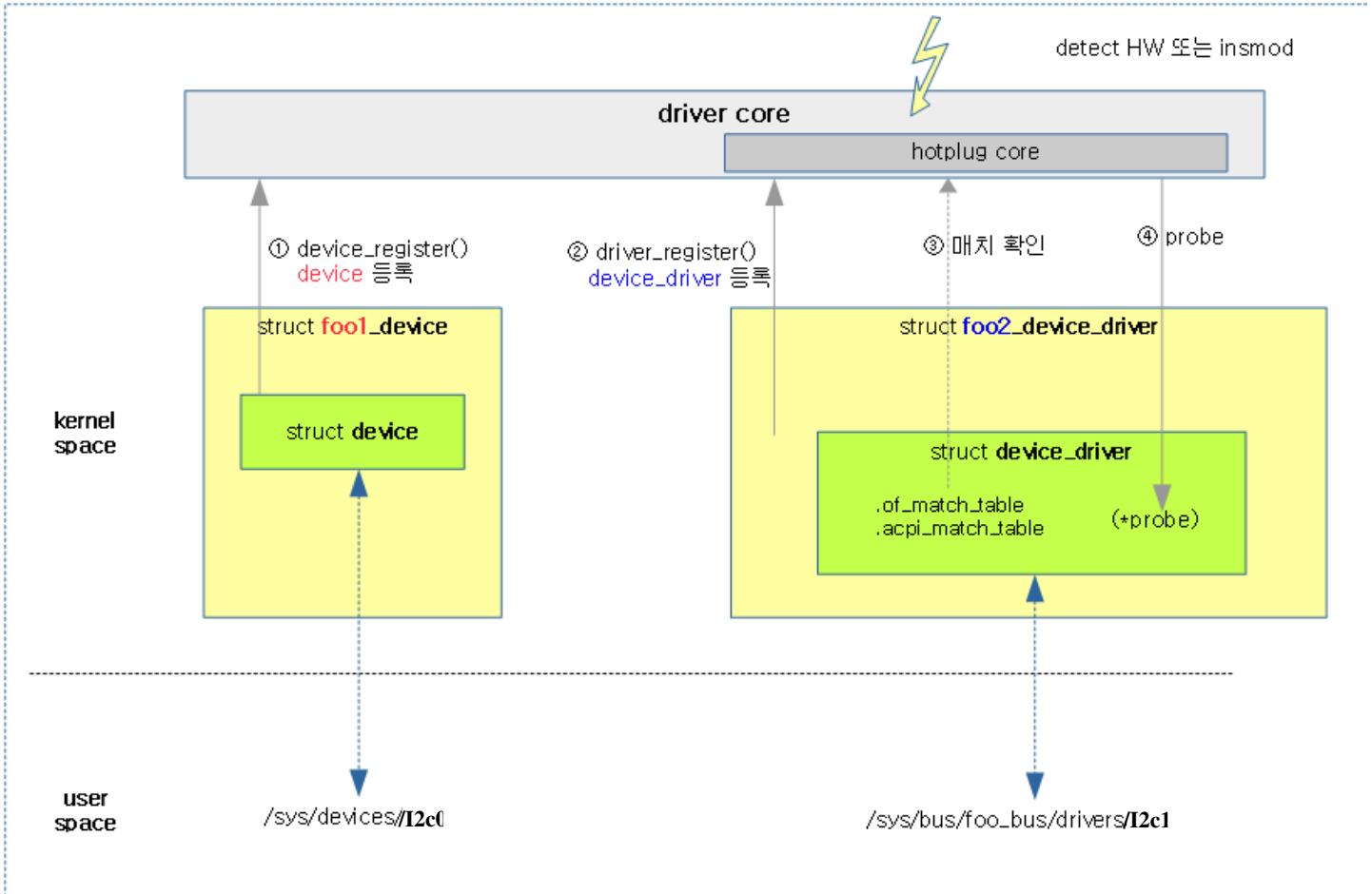
```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/petalinux_zynq/petalinux-v2015.4-final/components/linux-kernel/xlnx-4.0/drivers/spi$ ls
Kconfig          spi-bcm53xx.h        spi-clps711x.c       spi-efm32.c      spi-fsl-spi.c      spi-mpc52xx.c      spi-pl022.c          spi-rspi.c          spi-sh.c          spi-tle62x0.c      spidev.c
Makefile         spi-bcm63xx-hsspi.c  spi-coldfire-qspi.c  spi-ep93xx.c     spi-fsl-spi.h      spi-mxs.c          spi-ppc4xx.c         spi-s3c24xx-fiq.S   spi-sirf.c        spi-topcliff-pch.c
spi-adi-v3.c     spi-bcm63xx.c        spi-davinci.c        spi-falcon.c     spi-gpio.c         spi-nuc900.c       spi-pxa2xx-dma.c     spi-s3c24xx-fiq.h   spi-st-ssc4.c     spi-txx9.c
spi-altera.c     spi-bfin-sport.c     spi-dln2.c           spi-fsl-cpm.c    spi-img-spfi.c     spi-oc-tiny.c      spi-pxa2xx-pci.c     spi-s3c24xx.c       spi-sun4i.c       spi-xcomm.c
spi-ath79.c      spi-bfin5xx.c        spi-dw-mid.c         spi-fsl-cpm.h    spi-imx.c          spi-octeon.c       spi-pxa2xx-pxadma.c  spi-s3c64xx.c       spi-sun6i.c       spi-xilinx.c
spi-atmel.c      spi-bitbang-txrx.h   spi-dw-mmio.c        spi-fsl-dspi.c   spi-lm70llp.c      spi-omap-100k.c    spi-pxa2xx.c         spi-sc18is602.c     spi-tegra114.c    spi-xtensa-xtfpga.c
spi-au1550.c     spi-bitbang.c        spi-dw-pci.c         spi-fsl-espi.c   spi-meson-spifc.c  spi-omap-uwire.c   spi-pxa2xx.h         spi-sh-hspi.c       spi-tegra20-sflash.c  spi-zynq-qspi.c
spi-bcm2835.c    spi-butterfly.c      spi-dw.c             spi-fsl-lib.c    spi-mpc512x-psc.c  spi-omap2-mcspi.c  spi-qup.c            spi-sh-msiof.c      spi-tegra20-slink.c   spi-zynqmp-gqspi.c
spi-bcm53xx.c    spi-cadence.c        spi-dw.h             spi-fsl-lib.h    spi-mpc52xx-psc.c  spi-orion.c        spi-rockchip.c       spi-sh-sci.c        spi-ti-qspi.c     spi.c
```

spi-cadence.c 를 살펴 볼 것이다. Spidev.c 드라이버는 유저영역에서 우리가 사용할 디바이스 드라이버이므로 알아두자. 밑에서 설명하겠다.

```
static const struct of_device_id cdns_spi_of_match[] = {
    { .compatible = "xlnx,zynq-spi-r1p6" },
    { .compatible = "cdns,spi-r1p6" },
    { /* end of table */ }
};
MODULE_DEVICE_TABLE(of, cdns_spi_of_match);

/* cdns_spi_driver - This structure defines the SPI subsystem platform driver */
static struct platform_driver cdns_spi_driver = {
    .probe  = cdns_spi_probe,
    .remove = cdns_spi_remove,
    .driver = {
        .name = CDNS_SPI_NAME,
        .of_match_table = cdns_spi_of_match,
        .pm = &cdns_spi_dev_pm_ops,
    },
};

module_platform_driver(cdns_spi_driver);

MODULE_AUTHOR("Xilinx, Inc.");
MODULE_DESCRIPTION("Cadence SPI driver");
MODULE_LICENSE("GPL");
```

spi-cadence.c 파일의 코드 일부이다. Zynq-7000의 디바이스 트리에서 설정해준 “xlnx,zynq-spi-r1p6” 이라는 문자열을 볼 수 있다.

다음은 디바이스 드라이버가 호출되는 과정을 간략히 보여준다.



Path: Subsystem/linux/configs/device-tree 디렉터리로 와서 다음 코드를 추가 해준다.
디바이스 트리 systop.dtsi 에 추가해줘도 되고, pcw.dtsi 파일에 추가해줘도 된다. 어차피 다 참조하여 dtb파일을 생성하므로..
필자는 이미 &spi0로 spi0를 참조를 하여 spi 장치를 활성화 시켜주고 있는 pcw.dtsi 파일에 추가해주었다.

```
spidev@0x00 {
        compatible = “spidev”
        spi-max-frequency = <1000000> ;
        reg = <0>;
 };
```

```
spi0: spi@e0006000 {
    compatible = "xlnx,zynq-spi-r1p6";
    reg = <0xe0006000 0x1000>;
    status = "disabled";
    interrupt-parent = <&intc>;
    interrupts = <0 26 4>;
    clocks = <&clkc 25>, <&clkc 34>;
    clock-names = "ref_clk", "pclk";
    #address-cells = <1>;
    #size-cells = <0>;
};
```

zynq-7000.dtsi

```
&spi0 {
    is-decoded-cs = <0>;
    num-cs = <3>;
    status = "okay";
    spidev@0x00 {
        compatible = "spidev";
        spi-max-frequency = <1000000>;
        reg = <0>;
    };
};
```

pcw.dtsi

spidev 라는 노드 이름과 0이라는 장치주소를 사용하고 spidev 키 값을 사용하였다. 이를 찾아보자.
Path:~/petalinux_zynq/petalinux-v2015.4-final/components/linux-kernel/xlnx-4.0/drivers/staging/fbtft 디렉터리에 fbtft_device.c 파일에 다음과 같이 표현되어있다,

```
.name = "spidev",
.spi = &(struct spi_board_info) {
    .modalias = "spidev",
    .max_speed_hz = 500000,
    .bus_num = 0,
    .chip_select = 0,
    .mode = SPI_MODE_0,
    .platform_data = &(struct fbtft_platform_data) {
        .gpios = (const struct fbtft_gpio []) {
            {},
        },
    }
}
```

SPI device를 사용하기 위한 가장 간단한 방법은 spi_board_info 에 해당하는 device 내용을 작성하는 것이다. 구조체의 변수 중에 modalias가 보통 “spidev”로 되어 있다. 이 곳을 driver의 이름과 일치시키면 된다. 이름이 맞지 않으면 SPI device가 로드되지 않는다.

modalias 속성은 **모듈의 추가 이름 (별칭)을 정의한다.**

spidev 드라이버 모듈을 별칭으로 쉽게 사용하기 위해 설정 해둔 것이다.
고로 디바이스 트리에 compatible 속성에 spidev 키 값을 넣어주어 spidev 드라이버를 사용한다는 것이다.

다음은 우리가 추가 해주었던 spidev.c 디바이스 드라이버 파일의 compatible 속성이다.
Path: ~/petalinux_zynq/petalinux-v2015.4-final/components/linux-kernel/xlnx-4.0/drivers/spi

```
static const struct of_device_id spidev_dt_ids[] = {
    { .compatible = "rohm,dh2228fv" },
    {},
};
```

우리가 사용하고자 하는 spidev.c 드라이버를 사용하려면 다음 코드를 작성해야 함을 알 수 있다.

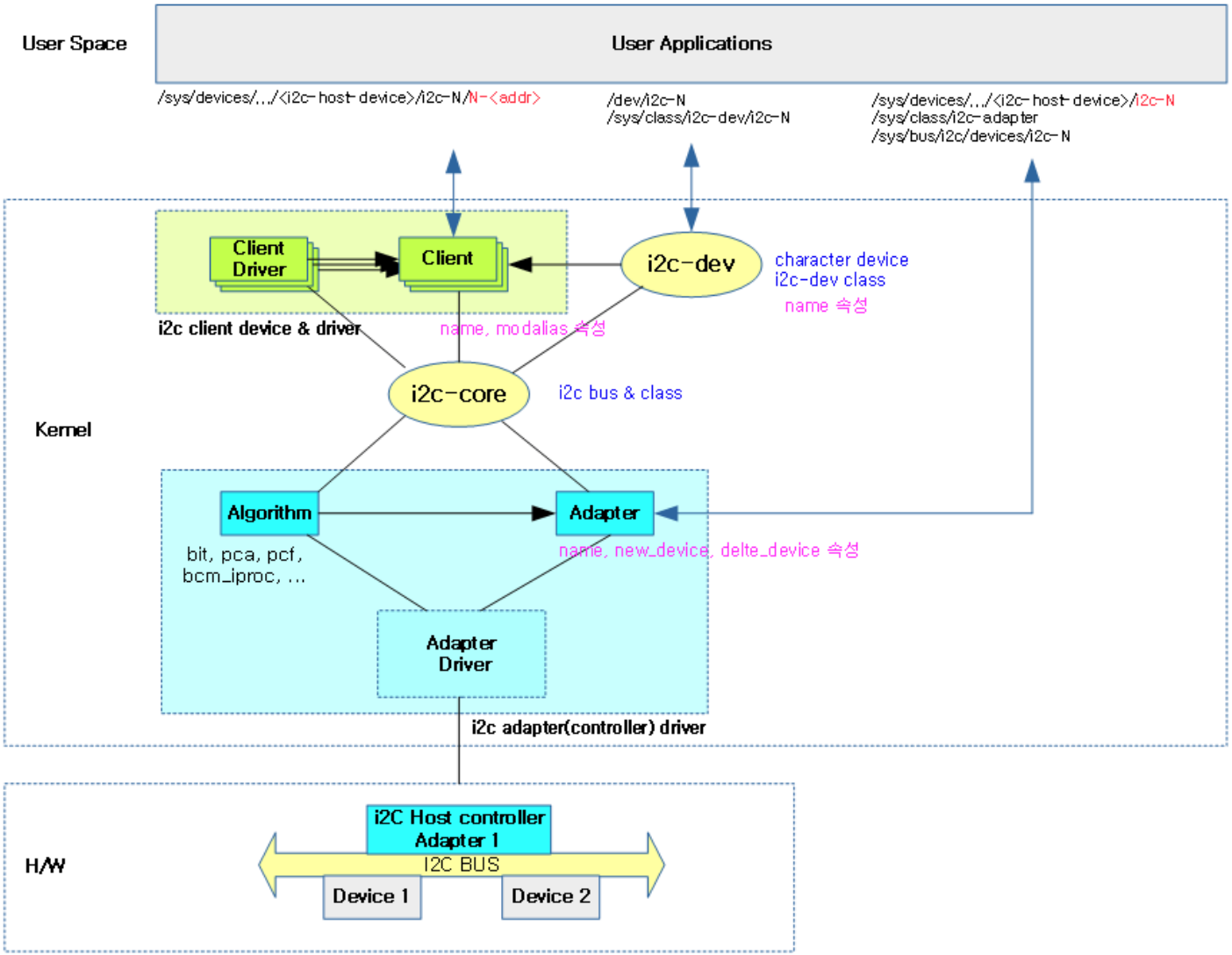
```
spidev@0x00 {
        compatible = “spidev”
        spi-max-frequency = <1000000> ;
        reg = <0>;
 };
```

또는

```
spidev@0x00 {
        compatible = “rohm,dh2228fv”
        spi-max-frequency = <1000000> ;
        reg = <0>;
 };
```

를 디바이스 트리에 추가 시켜주면 된다.

자 여기서 질문..? 우리는 왜 spi-cadence 디바이스 드라이버와 spidev 디바이스 드라이버 두 개를 쓸까..?



그렇다고 한다. 사이트 : http://jake.dothome.co.kr/i2c-1/#comment-180700 설명이 나와있다. 문c 블로그 참고.

```c
struct spi_board_info {
    /* the device name and module name are coupled, like platform_bus;
     * "modalias" is normally the driver name.
     *
     * platform_data goes to spi_device.dev.platform_data,
     * controller_data goes to spi_device.controller_data,
     * irq is copied too
     */
    char        modalias[SPI_NAME_SIZE];
    const void  *platform_data;
    void        *controller_data;
    int     irq;

    /* slower signaling on noisy or low voltage boards */
    u32     max_speed_hz;


    /* bus_num is board specific and matches the bus_num of some
     * spi_master that will probably be registered later.
     *
     * chip_select reflects how this chip is wired to that master;
     * it's less than num_chipselect.
     */
    u16     bus_num;
    u16     chip_select;

    /* mode becomes spi_device.mode, and is essential for chips
     * where the default of SPI_CS_HIGH = 0 is wrong.
     */
    u16     mode;

    /* ... may need additional spi_device chip config data here.
     * avoid stuff protocol drivers can set; but include stuff
     * needed to behave without being bound to a driver:
     *  - quirks like clock rate mattering when not selected
     */
};
```

```
root@spi_can:~# spi -s
spi: option requires an argument -- 's'
Usage: spi [-DsbdlHOLC3vpNR24SI]
  -D --device   device to use (default /dev/spidev1.1)
  -s --speed    max speed (Hz)
  -d --delay    delay (usec)
  -b --bpw      bits per word
  -i --input    input data from a file (e.g. "test.bin")
  -o --output   output data to a file (e.g. "results.bin")
  -l --loop     loopback
  -H --cpha     clock phase
  -O --cpol     clock polarity
  -L --lsb      least significant bit first
  -C --cs-high  chip select active high
  -3 --3wire    SI/SO signals shared
  -v --verbose  Verbose (show tx buffer)
  -p            Send data (e.g. "1234\xde\xad")
  -N --no-cs    no chip select
  -R --ready    slave pulls low to pause
  -2 --dual     dual transfer
  -4 --quad     quad transfer
  -S --size     transfer size
  -I --iter     iterations
```

```
root@spi_can:~# spi -H -b 8 -s 500000  -p "1234567789" -v
spi mode: 0x1
bits per word: 8
max speed: 500000 Hz (500 KHz)
TX | 31 32 33 34 35 36 37 37 38 39 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __  | 1234567789
RX | 31 32 33 34 35 36 37 37 38 39 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __  | 1234567789
```