

Xilinx FPGA, TI DSP·MCU

기반의 회로 설계 및 임베디드 전문가 과정

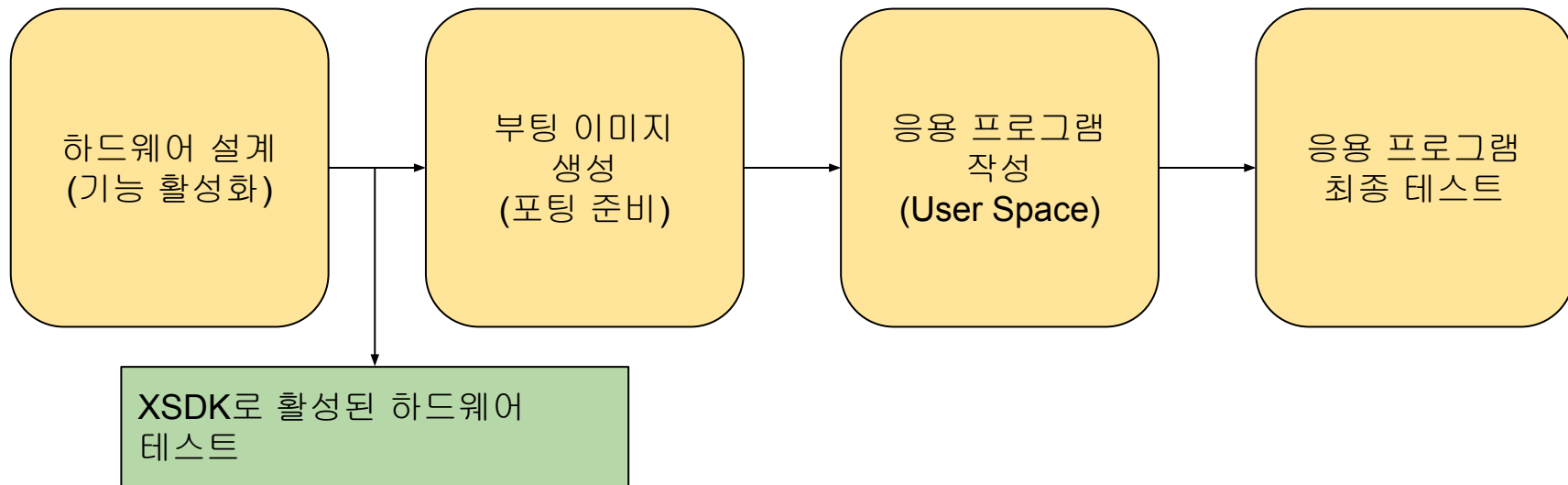
김현,
최준호 I2C
문서화

목차

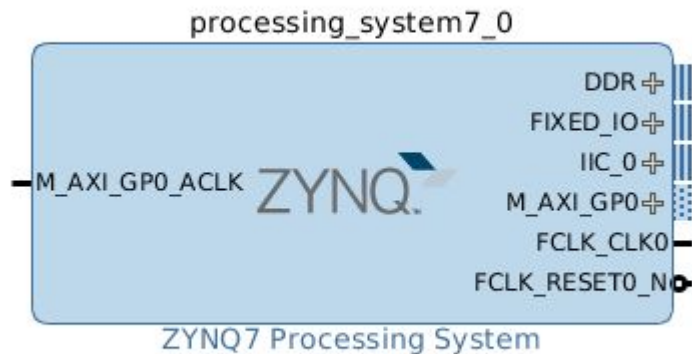
- Xilinx Zynq Zybo PS에서 I2C IP 활성화
- XSDK로 활성화된 I2C IP 테스트(with MPU6050)
- PetaLinux로 설계한 하드웨어 정보 이미지화
- MPU6050 Memory Map을 참고하여 I2C User Space Application 작성
- Zybo에 포팅
- Zybo에서 응용 프로그램 최종 테스트

Xilinx Zynq Zybo PS에서 I2C IP 활성화

I2C 구현 순서

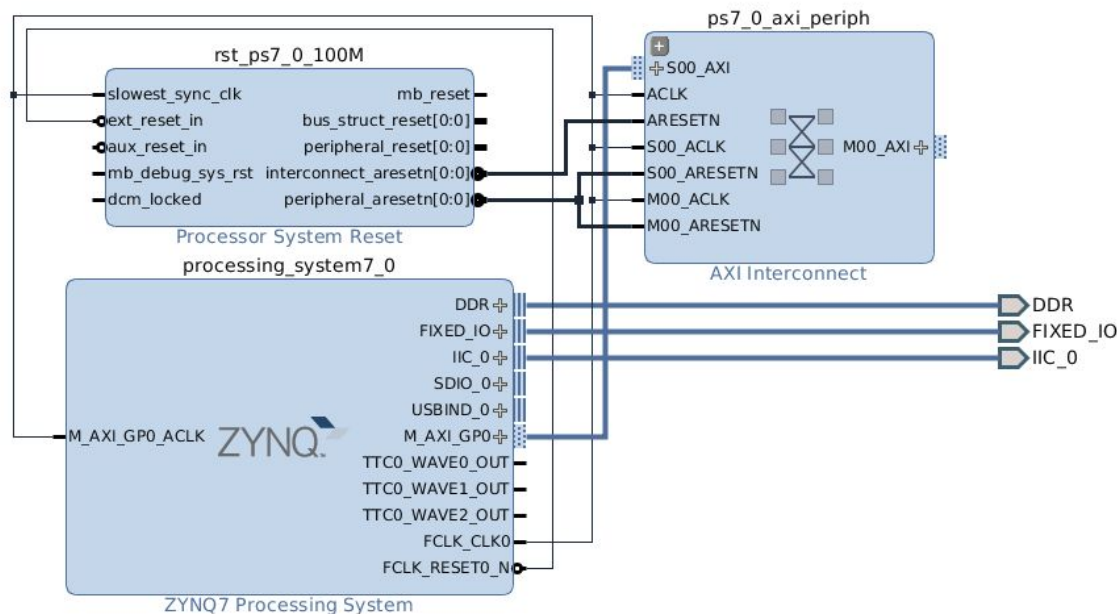


PS에 I2C IP 활성화



PS에 하드코어 돼 있는 I2C를 사용해야 한다.
여기서는 I2C 0을 활성화 해주면서 IO는
EMIO로 설정한다.
좌측 PS IP 블록에서 I2C가 활성화됨을 확인한다.

PS에 I2C IP 활성화



Auto Connection을 통해
기본적인 구성을 완료하고,
I2C 신호를 MIO 핀에서 받을 수
있도록 Make External 한다.

기본적인 구성 및 Product 생성이 끝나면 합성한다.

PS에 I2C IP 활성화

| Name | Direction | Board Part Pin | Board Part Interface | Neg Diff Pair | Package Pin | Fixed | Bank | I/O Std | Vcco | Vref | Drive Strength | Slew Type | Pull Type | Off-Chip Termination | IN TEF |
|--------------------|-----------|----------------|----------------------|---------------|-------------|-------|------------------------|---------|-----------------------|------|----------------|-------------|------------|----------------------|------------|
| All ports (132) | | | | | | | | | | | | | | | |
| DDR_1497 (71) | INOUT | | | | | ✓ | 502 (Multiple)* | | 1.500 (Multiple) | | (Multiple)* | (Multiple)* | NONE | FP_VTT_50 | (Multiple) |
| FIXED_IO_1497 (59) | INOUT | | | | | ✓ | (Multiple) (Multiple)* | | (Multiple) (Multiple) | | (Multiple)* | (Multiple)* | (Multiple) | (Multiple) | |
| IIC_0_1497 (2) | INOUT | | | | | ✓ | 35 LVCMOS33* | | 3.300 | | 12 | SLOW | PULLUP | FP_VTT_50 | |
| Scalar ports (2) | | | | | | | | | | | | | | | |
| iic_0_scl_io | INOUT | | | | J15 | ✓ | 35 LVCMOS33* | | 3.300 | | 12 | SLOW | PULLUP | FP_VTT_50 | |
| iic_0_sda_io | INOUT | | | | H15 | ✓ | 35 LVCMOS33* | | 3.300 | | 12 | SLOW | PULLUP | FP_VTT_50 | |
| Scalar ports (0) | | | | | | | | | | | | | | | |

I2C 0 블록의 SCL, SDA 핀을 설정 해주고 MPU6050 기준으로 3.3v를 그리고 Pull Type을 PULL-UP으로 한다.

바뀐 내용을 저장하고 구현한다.

구현이 완료되면 Bitstream을 만들고 SDK로 Export하면서 이 만든 Bitstream을 include 한다.

XSDK로 활성화된 I2C IP 테스트(with MPU6050)


I2C 레지스터가 존재하는지 확인

| | | | |
|----------------------|------------|------------|----------|
| ps7_uart_1 | 0xe0001000 | 0xe0001fff | REGISTER |
| ps7_coresight_comp_0 | 0xf8800000 | 0xf88fffff | REGISTER |
| ps7_i2c_0 | 0xe0004000 | 0xe0004fff | REGISTER |
| ps7_ttc_0 | 0xf8001000 | 0xf8001fff | REGISTER |

ps7_gpio_0 **gpiops**
ps7_gpv_0 **generic**
ps7_i2c_0 **iicps**
ps7_intc_dist_0 **generic**

[Documentation](#) [Import Examples](#)
[Documentation](#)
[Documentation](#) [Import Examples](#)
[Documentation](#)

```
/*  
 * The slave address to send to and receive from.  
 */  
#define IIC_SLAVE_ADDR 0x55  
#define IIC_SCLK_RATE 100000
```

❏  xiicps_selftest_example

제대로 하드웨어 구성이 됐는지 확인한다.

BSP를 만들고, system.mss 파일에서 i2c examples를 통해

이제 구현된 I2C의 외부 핀과 MPU6050을 연결 해 제대로 I2C 통신이 되는지 확인 한다.

Zybo와 MPU6050의 SCL, SDA 선 그리고 3.3V 전원과 GND를 연결하고 테스트 앱에 MP6050 Slave Address 상수 부분을 0x68로 변경하고 앱을 돌려 테스트 한다.

PetaLinux로 설계한 하드웨어 정보 이미지화

PetaLinux에서 설정해야 할 사항들

I2C에 대한 하드웨어 기능을 활성화하였으므로 그것을 커널과 사용자가 소프트웨어적으로 사용할 수 있도록 각종 필요한 설정을 해야하고 그 설정은 아래와 같다.

- u-boot : config i2c device driver
- device tree source : 이미 zynq template을 통해 설정 돼 있으므로 건드릴 필요가 없다.
- root file system : I2C User Space Application을 작성 후 빌드
- kernel : 커널 소스는 별도의 다른 설정이 필요 없다.

Device Driver 설정

- u-boot에서 디바이스 드라이버 설정

```
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test
petuza@petuza-B85H3-M7: ~
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ cd petalinux/i2c-test/
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ petalinux-config -c u-boot
INFO: Checking component...
INFO: Config linux/u-boot
[INFO ] generate linux/u-boot configuration files
#
# configuration written to .config
#
[INFO ] config linux/u-boot

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$
```

```
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test
petuza@petuza-B85H3-M7: ~
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ cd petalinux/i2c-test/
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ petalinux-config -c u-boot
INFO: Checking component...
INFO: Config linux/u-boot
[INFO ] generate linux/u-boot configuration files
#
# configuration written to .config
#
[INFO ] config linux/u-boot

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$
```

```
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test
petuza@petuza-B85H3-M7: ~
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ cd petalinux/i2c-test/
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ petalinux-config -c u-boot
INFO: Checking component...
INFO: Config linux/u-boot
[INFO ] generate linux/u-boot configuration files
#
# configuration written to .config
#
[INFO ] config linux/u-boot

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$
```

```
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test
petuza@petuza-B85H3-M7: ~
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ cd petalinux/i2c-test/
petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$ petalinux-config -c u-boot
INFO: Checking component...
INFO: Config linux/u-boot
[INFO ] generate linux/u-boot configuration files
#
# configuration written to .config
#
[INFO ] config linux/u-boot

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

petuza@petuza-B85H3-M7: ~/petalinux/i2c-test$
```

MPU6050 Memory Map을 참고하여 I2C User Space Application 작성

MPU6050에서 자이로스코프 값을 가져오기 위한

MPU6050에서 자이로 센서를 사용하기 위한 설정 사항 및 순서



I2C) Transmit
Linux) Write
Mode) Start-Stop

Transmit
Write
Start-Stop

Receive
Read
Repeated Start

응용 프로그램 импорт 또는 작성

[mpu6050_gyro_sensing.c](#) 예제 코드를 사용하거나

필요한 응용 프로그램을 작성한다.

작성하는 코드는 PetaLinux 안의 **app** 컴포넌트를 만들어 **ramdisk**에 올릴 준비를 한다.

Zybo에
포팅

PetaLinux로 최종적인 포팅 이미지 생성

App 작성 이후에 `petalinux-build` 명령을 통해 모든 설정사항을 빌드한다.

이후에 PetaLinux 루트 디렉토리에서 `images/linux/` 디렉토리에서

```
petalinux-package --boot --fsbl zynq_fsbl.elf --fpga ./system_wrapper.bit --u-boot
```

명령어를 통해 build를 통해 만든 여러 이미지 파일들을 포팅할 때 쓰일 통합 이미지로 바꾼다.

이후 만들어진 `BOOT.bin`, `image.ub` 파일을 SD 카드의 `boot` 부분에 옮기고 Zybo에 마운트한다.

Zybo에서 응용 프로그램 최종
테스트

최종적인 응용 프로그램 테스트(디버깅)

```
Try to receive data from 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
Successfully sent data
[acc_xyz]--1 : 152
000 xyz[0] : 152
[acc_xyz]--2 : 242
111 xyz[0] : 38912
222 xyz[0] : 39154
[acc_xyz]--3 : 240
000 xyz[1] : 240
[acc_xyz]--4 : 245
111 xyz[1] : 61440
222 xyz[1] : 61685
[acc_xyz]--5 : 147
000 xyz[2] : 147
[acc_xyz]--6 : 245
111 xyz[2] : 37632
222 xyz[2] : 37877
X : 39154
Y : 61685
Z : 37877
--- 186 ---

Try to receive data from 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
Successfully sent data
[acc_xyz]--1 : 166
000 xyz[0] : 166
[acc_xyz]--2 : 242
111 xyz[0] : 42496
222 xyz[0] : 42738
[acc_xyz]--3 : 240
000 xyz[1] : 240
[acc_xyz]--4 : 245
111 xyz[1] : 61440
222 xyz[1] : 61685
[acc_xyz]--5 : 165
000 xyz[2] : 165
[acc_xyz]--6 : 245
111 xyz[2] : 42240
222 xyz[2] : 42485
X : 42738
Y : 61685
Z : 42485
--- 187 ---
```

만약 위에서 이미 만들어 놓은 예제 응용 프로그램을 사용했다면 다음과 같이 센서 값을 가져와서 뿌리는 것을 볼 수 있다.

따로 응용 프로그램을 만들었다면 Zybo 부팅 후 Linux로 들어가서 /bin 경로에 있는 응용 프로그램 바이너리 파일을 실행하여 실행 결과를 확인한다.

필요하다면 디버깅 한다.