

Xilinx FPGA, TI DSP·MCU

기반의 회로 설계 및 임베디드 전문가 과정

최준호
계획/성과
4주차



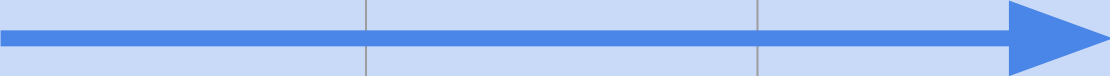
목차

- 내 역할
- 전체 일정
- FPGA 일정
 - FPGA 다음 주 목표
- DSP 일정
 - DSP 다음 주 목표
- 지난 주 성과

내 역할


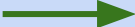
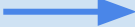
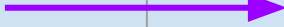

- DSP Linux Device Driver 개발 및 각 장치 구현
- FPGA Linux Porting 및 Linux Device Driver 개발 및 장치 개발 및 PL 구현

전체 일정




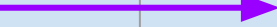

구분	파트	05月	06月	07月
FPGA	Firmware			
DSP	Firmware			
	LDD			

*LDD : Linux Device Driver

FPGA 일정(기존)

파트 \ 주차	22	23	24	25	26	27	28
FPGA 교재							
Linux Porting							
PL 구현							
Motor Control (PL or PS) 구현							
CAN, DMA 구현							

FPGA 일정(변경)

파트 \ 주차	22	23	24	25	26	27	28
FPGA 교재							
Linux Porting							
PL 구현							
Motor Control (PL or PS) 구현							
CAN, DMA 구현							

FPGA 다음 주 목표

5월 5주차

파트 \ 일	29	30	31	01	02	03	04
LDD를 위한 Zybo에 포팅한 커널 코드 분석							
LDD 분석 및 작성							

DSP 일정(변경)

파트 \ 주차	22	23	24	25	26	27	28
LDD							
LDD 공동작업 (I2C 문서화)							

DSP 다음 주 목표

5월 5주차

파트 \ 일	29	30	31	01	02	03	04
LDD 공동작업 (I2C 문서화)							
LDD 공동작업 (Kernel Driving)							

지난 주 성과

FPGA 지난, 이번 주 목표

15일	16일	17일	18일	19일	20일	21일
Zynq 설계 기본 52~104	Custom IP 및 Timer 설계 105~156	Zynq 설계 응용 157~175	리눅스 포팅에 대한 조사 1/2	리눅스 포팅에 대한 조사 2/2	리눅스 포팅 실습 1/4	리눅스 포팅 실습 2/4

22일	23일	24일	25일	26일	27일	28일
리눅스 포팅 실습 3/4	리눅스 포팅 실습 4/4	FPGA Linux Porting 1/2	FPGA Linux Porting 2/2	Zynq PL 조사 및 PL Control 1/3	Zynq PL 조사 및 PL Control 2/3	Zynq PL 조사 및 PL Control 3/3

FPGA 지난, 이번 주 성과(5.18)

The screenshot shows a web browser window with the Xilinx website. The address bar shows the URL https://xilinx.entitlenow.com/AcrossUser/main.gsp?product=&tab=&req_hash=&. The page title is "Product Licensing". A modal window titled "Create New Licenses" is displayed, showing a "Congratulations" message: "Your new license file has been successfully generated and e-mailed to peluza6332@gmail.com. You can also view the license file under the Manage Licenses tab." Below this, it says "Please add this sender (xilinx.notification@entitlenow.com) to your address book." The modal also displays "License File Details" (Node License, Host ID: 4061864f09b) and "Products" (SDSoC Environment, 60 Day Evaluation License (No Charge): 1 seats; Vivado Design Suite: HL WebPACK 2015 and Earlier License (No Charge): 1 seats; ISE WebPACK License (No Charge): 1 seats). The modal includes a table with columns "Host Name" and "Created Date", showing "peluza-B85H3-M7" and "18 MAY 2017". A "Help" link is visible. In the bottom left, a "Tcl Console" window is open, showing a "CRITICAL WARNING" message: "Failed to open handle vivado.jou. Please check access permission of directory '/opt/Xilinx/Vivado/2016.4/bin'. You should restart the application from a writable working directory." The console also shows session information: "# Vivado v2016.4 (64-bit)", "# SW Build 1756540 on Mon Jan 23 19:11:19 MST 2017", "# IP Build 1755317 on Mon Jan 23 20:30:07 MST 2017", and "# Start of session at: Thu May 18 10:01:40 2017".

Home : Support : Product Licensing

Product Licensing

Help

Create New Licenses

Congratulations

Your new license file has been successfully generated and e-mailed to peluza6332@gmail.com. You can also view the license file under the Manage Licenses tab.

Please add this sender (xilinx.notification@entitlenow.com) to your address book.

License File Details

Node License
Host ID: 4061864f09b

Products

SDSoC Environment, 60 Day Evaluation License (No Charge): 1 seats
Vivado Design Suite: HL WebPACK 2015 and Earlier License (No Charge): 1 seats
ISE WebPACK License (No Charge): 1 seats

Host Name	Created Date
peluza-B85H3-M7	18 MAY 2017

Tcl Console

CRITICAL WARNING: [Common 17-183] Failed to open handle vivado.jou. Please check access permission of directory '/opt/Xilinx/Vivado/2016.4/bin'. You should restart the application from a writable working directory.
CRITICAL WARNING: [Common 17-183] Failed to open handle vivado.log. Please check access permission of directory '/opt/Xilinx/Vivado/2016.4/bin'. You should restart the application from a writable working directory.

Vivado v2016.4 (64-bit)
SW Build 1756540 on Mon Jan 23 19:11:19 MST 2017
IP Build 1755317 on Mon Jan 23 20:30:07 MST 2017
Start of session at: Thu May 18 10:01:40 2017

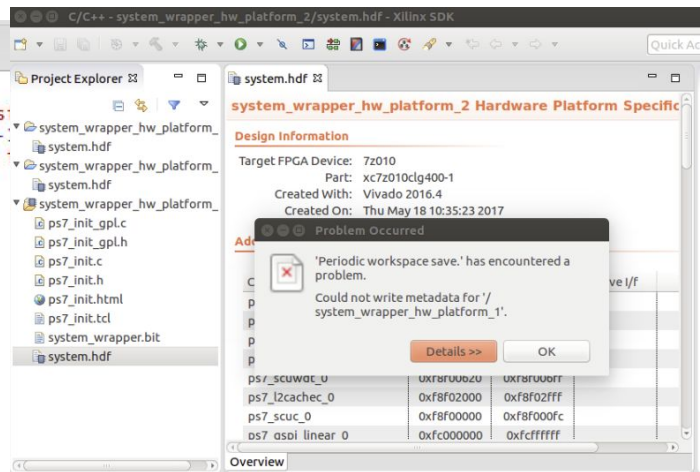
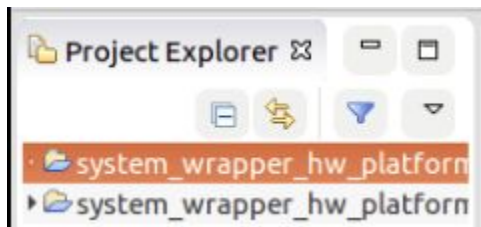
FPGA 지난, 이번 주 성과(5.19)

*FPGA 부분 완료

어제의 심정 : x발 절대 안돼 이걸

SDK Log

```
10:23:29 INFO : Registering command handlers for SDK TCF services
10:23:29 ERROR : Unsatisfied link error while loading librdi_repo_manager.so: java.lang.Unsatisf
10:23:29 INFO : Launching XSCT server: xsct -n -interactive /home/peluza/Vivado/Exercises/EX2-1/EX2-1.s
10:23:36 ERROR : (XSDB Server)couldn't read file "/scripts/xsdb/xsdb/xsdb-server.tcl": no such
while executing
"source /scripts/xsdb/xsdb/xsdb-server.tcl"
(file "/home/peluza/Vivado/Exercises/EX2-1/EX2-1.sdk/temp_xsdb_launch_script.tcl" line 1)
```



SDK Log

```
16:44:48 INFO : Registering command handlers for SDK TCF services
16:44:49 INFO : Launching XSCT server: xsct -n -interactive /home/peluza/Vivado/Exercises/EX2-3/EX2-3.s
16:44:55 INFO : XSCT server has started successfully.
16:45:05 INFO : Successfully done setting XSCT server connection channel
16:45:05 INFO : Successfully done setting SDK workspace
16:46:19 INFO : Connected to target on host '127.0.0.1' and port '3121'.
16:46:20 INFO : 'targets -set -filter {jtag_cable_name =~ "Digilent Zybo 210279778487A" && level==0} -index 1' command is executed.
16:46:21 INFO : FPGA configured successfully with bitstream "/home/peluza/Vivado/Exercises/EX2-3/EX2-3.sdk/system wrapper hw platform 0/system wrapper.bit"
```



FPGA 지난, 이번 주 성과(5.20)

*FPGA

usb 우분투에서 uart 문제 드디어 해결.

serial 통신 자료만 존LLLLLLLLLLLLLLLLLLLLLx게 본듯

등산 하면서도 계속 보다가 문득 깨닫고 현자 됨

그 뒤로 안 보고 등산 잘 하고 와서 해보니 진짜 됨

복병은 망할 Eclipse .plugins

하드웨어 바뀌면 맛탱이 감

개발할 땐 개발 도구의 하드웨어는 바꾸지 않는 편이 나은듯?

고로 다음부터 좋은 노트북 하나 사서 하는 편이 나을듯

이왕이면 리눅스 or OS X

```
* gpio_test.c
peluza@peluza-B85H3-M7: ~
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 0
Switch Status : 2
Switch Status : 2
Switch Status : 2
Switch Status : 2
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
Switch Status : A
```

```
peluza@peluza-B85H3-M7: ~
File Edit View Search Terminal Help
peluza@peluza-B85H3-M7:~$ dmesg | grep tty
[ 0.000000] console [tty0] enabled
[ 0.633501] 00:05: ttyS0 at I/O 0x3f8 (irq = 4, base_baud = 115200) is a 16550A
[ 254.545106] usb 3-5: FTDI USB Serial Device converter now attached to ttyUSB0
[ 254.545380] usb 3-5: FTDI USB Serial Device converter now attached to ttyUSB1
[ 417.416719] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from ttyUSB0
[ 417.417069] ftdi_sio ttyUSB1: FTDI USB Serial Device converter now disconnected from ttyUSB1
[ 431.817546] usb 3-5: FTDI USB Serial Device converter now attached to ttyUSB0
[ 431.820586] usb 3-5: FTDI USB Serial Device converter now attached to ttyUSB1
peluza@peluza-B85H3-M7:~$
```

FPGA 지난, 이번 주 성과(5.21)

*FPGA

Zynq 교재 끝

엄청난 이슈 메이커가 드디어 종결 나는 군 :)

[illegible]

QSPI, SD 비휘발성 메모리에 적재하는 법.

PL에 적재 후 PS와 AXI4를 통한 연동

Time Library 등을 써보았다.

갈 길이 멀다

feat. inipro-

항상 잘 읽어 볼 것

문제는 가까운 곳에 도사리고 있음 하!하!

Program Flash

Connecting to hw server @ TCP:127.0.0.1:3121

Connected to hw server @ TCP:127.0.0.1:3121

Available targets and devices:

Target 0 : jsn-Zybo-210279778487A

Device 0: jsn-Zybo-210279778487A-4ba00477-0

Retrieving Flash info...

```
Initialization done, programming the memory
```

BOOT MODE REG = 0x00000000

f probe 0 0 0

```
Performing Erase Operation...
```

Erase Operation successful.

INFO: [Xicom 50-44] Elapsed time = 7 sec.

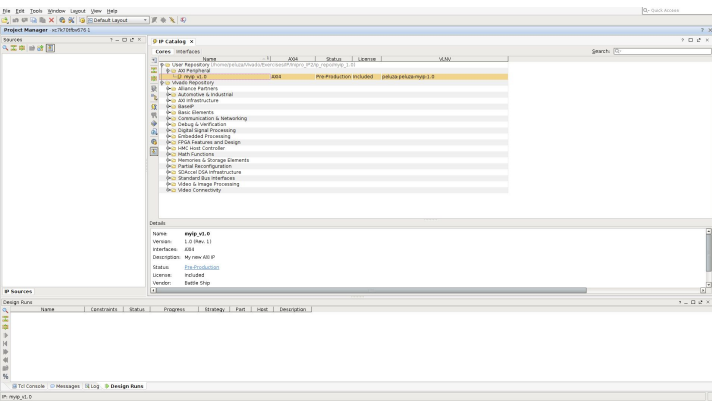
Performing Program Operation...

0%...50%...100%

```
Program Operation successful.
```

INFO: [Xicom 50-44] Elapsed time = 9 sec.

Flash Operation Successful



FPGA 지난, 이번 주 성과(5.22)

*FPGA 완료

Linux Porting에 대한 사전조사를 약 5개 정도 사이트를 돌아다니면서 진행하고 실제 Linux Porting을 위한 시도를 해보았다.

• <http://recipes.edosoo.com/5407905>
(Linux 포팅 완료까지의 절차가 어떻게 되죠?)

부트로더와 커널은 대부분 개발 보드의 BSP에 만들어져 있어서 수정할 것이 없다.
(하지만 실제 제품을 만들 때는 BSP를 기반으로 실제 제품에 맞게 소스를 수정해야 한다.)

- 부트로더
부트로더에서 가장 먼저 실행되는 파일은 Startup.s이다.
그리고 부트로더 포팅 순서는 아래와 같다.
1. Start.s, (startup.s)
2. lowlevel_init.s, (lowlevel_init.s) : 여러 가지 OS 설정들(MMU Enabled 등)
3. cpu_init.s, (cpu_init.s) : 와치독, 인터럽트 관련 내용 설정
4. board.c, (board.c) :

리눅스의 경우 NOR보다 NAND Flash를 많이 사용한다.
그리고 이를 위해 NAND를 초기화 한다.
그 후엔 SD RAM도 초기화 한다.
MMU 초기화
그 외 변수들 OS로 초기화 후 C언어로 된 Main 함수로 간다.

부트로더에서 가장 많이 하는 일은 u-boot, zimage, ramdisk.gz 같은 바이너리들을 쓰거나 SD, RAM으로 복사하는 것이다.
따라서 리눅스 시스템에서 부트로더는 필수적이다.
클래스 메모리에 위와 같은 바이너리들을 넣기 전에 시리얼 메시지를 찍는다.(자세한 사항은 해당 블로그 참조 208 참조)

(해서 실제 제품 보드를 개발할 때는 클러스 메모리를 자유롭게 다룰 수 있는 기술력이 있어야 한다.)

- 커널

Uncompressing Linux, done. booting the kernel
위와 같은 메시지는 ZImage를 SD-RAM에 복사한 후 zimage의 압축을 풀었다는 의미이다.
zimage를 풀면 piggy.gz, head.o, misc.o 파일이 나온다.
piggy.gz는 순수 커널 바이너리이고 misc.o가 이 커널의 압축을 풀고 있는 파일이다.
위 메시지가 piggy.gz를 misc.o가 풀어서 나오는 메시지가이다.

u-boot에서 zimage를 커져 커널이 드디어 커널이 SD-RAM에 적재된다.

(이때 kernelInitMain.c에 start_kernel 함수가 가장 먼저 실행된다.)

(예를 보드 BSP를 가지고 커널 포팅을 한다면 크게 수정할 일은 없어요. 실제 제품 보드를 제작한다면, 흔히

- Linux OS를 Host PC에 설치했고 보드 또한 준비했다면 그 다음은 CROSS-Compiler 설치와 빌드이다.

보통 컴파일러 또한 보드와 함께 제공된다. 그리고 설치 방법 또한 있을 수도 있다.
(자세한 설치 예시는 해당 링크 참조)
(이 CROSS-Compiler를 통해 ARM CPU에서 작동하는 코드를 만들 수 있다. 따라서 이 컴파일러로 만든 바이너리 파일은 해당 ARM이 올라가 있는 보드에서 실행해야 한다.)

- 다음으로 부트로더를 만든다.
이 부트로더는 보드의 flash memory에 존재하며 bootloader binary, kernel binary, filesystem binary로 구분되어 있다.(보통 bootloader는 flash memory의 물리 주소 0번지부터)

부트로더는 Flash memory에 저장된 kernel을 SD-RAM으로 복사시켜 준다.
전원이 인가되면 부트로더가 가장 먼저 실행되며 kernel을 SD-RAM으로 복사한다.
이 때문에 부트로더는 Flash Memory를 쉽게 읽고 쓸 수 있어야 한다.

부트로더의 압축을 풀고 위에 설치했던 크로스 컴파일러를 CROSS_COMPILE = 에 넣어준다.
위 작업이 끝난 후 부트로더를 make 명령어를 통해 빌드하면 u-boot.bin 파일이 생성된다.

- 다음은 커널을 빌드한다.
커널 옵션을 지정한 후 빌드하면 커널 빌드는 끝이 난다.
그러나 옵션이 매우 다양하기 때문에 보통 개발 보드에서 default로 배포하는 것을 써서 만들 수 있어서 쉽게 만들 수 있다.
(예시는 역시 링크 참조)

(# make 6410.config @ 개발 보드에 맞게 커널 옵션이 설정된 파일)

빌드가 끝나면 kernel image는 ../arch/arm/boot 폴더에 zimage로 생성된다.
이때 zimage는 압축된 Image이고 순수한 kernel binary는 ../boot/compressed/piggy.o (obj 파일)이다.

(Makefile을 분석하면 zimage에 대해 잘 알 수 있다. 커널 빌드가 끝나는 마지막 단계를 보시면 최종 piggy.o 파일이 생성되고, 그 파일은 다시 압축이 되지요. 그때 만들어진 파일 이름이 piggy.gz 파일이고, 이 파일은 압축된 형태이기 때문에 압축 풀기 위한 알고리즘 파일인 head와 misc파일이 최종 빌드 되면서 zimage로 만들어 집니다.)

그래서 부트로더는 커널 압축 이미지(zimage)를 에스디램으로 복사한 뒤 재언코딩 커널로 넘기면, 커널 압축 이미지인 zimage의 head이 먼저 실행되고, misc 파일이 실행되면서 piggy.gz 파일을 에스디램의 일정한 주소 영역에 압축을 풀게 되어 있지요. 이 때 압축 풀린 이미지(piggy.o)가 바로 순수한 커널 이미지인거요.

- Contents

- <http://m.blog.naver.com/k5248/120171621337>
Zynq zc720 Linux Porting(k5248)
(주의 : ZYNQ zc702에 관한 리눅스 포팅)

필요한 것들
ToolChain
Bootloader(u-boot?)
Kernel

해당 블로그에서 위 세 가지를 모두 다운로드 후

- > ToolChain은 install하고
- > CROSS_COMPILE(arm-xilinx-linux-gnueabi)과 그에 관련된 환경 설정(PATH)
- > u-boot 컴파일하고
- > kernel 컴파일하고(for ZC702, ZC770이라고 하는 걸 봐선 특정 하드웨어를 위한 kernel이 따로 있는건가?), 컴파일 되면 빌드(make)
- > Device Tree Generation(Kernel 3.0부터 DTS가 생긴)
- > BOOT.bin(boot.bif, zynq_fsb1_0.elf, system.bit) 파일 생성
BOOT.bin : Boot Image로서 하드웨어의 비휘발성 메모리에 저장되어 하드웨어 실행 시 이 이미지에 속해 있는 것을 DDR 메모리에 적재하여 소프트웨어를 동작 시킨다.
여기서는 임베디드 리눅스를 포팅함으로써 비휘발성 메모리에 OS를 적재하고 하드웨어를 켤 때 해당 OS를 부팅하는 것이겠다.

여기서 BOOT.bin에 FBSL, Bitstream, App 이 세 가지가 포함된다. 이 세 가지가 서로 각자의 역할을 하면서 리눅스를 실행시킬 것.

zynq_fsb1(First Stage Bootloader) : 소프트웨어를 하드웨어에 저장하여 실행하려면 필요한 것.
시스템 전원이 켜지면 FSBL은 PL에 Bitstream을 Configuration하고 ZYNQ에서 실행될 소프트웨어(APP)를 DDR Memory에 복사한 후 이 소프트웨어가 실행될 수 있게 한다.

FPGA 지난, 이번 주 성과(5.23)

*FPGA 완료

Linux Porting을 완료했다.

하지만 문제가 있다.

Linux Porting을 ramdisk 로드 없이
linaro fs가 그 역할을 대신해야 하는데
계속 Zybo SD 카드 부팅 시 ramdisk를
로딩하여 하여 문제가 발생한다.
수동으로 다시 bootm해주면
정상적으로 linux가 켜진다.

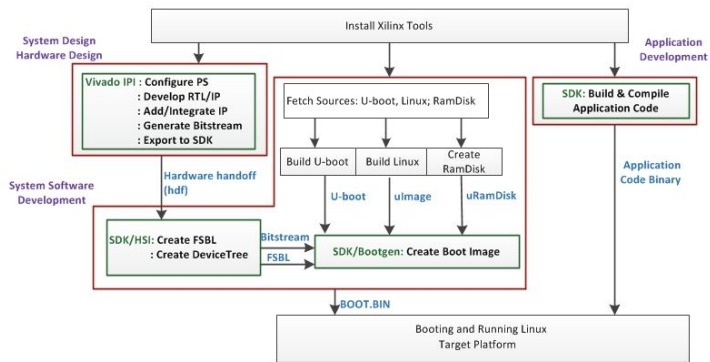


Figure 1: Xilinx Design Flow [1].

```
peluza@peluza-B85:~$  
[ OK ] Stopped GNOME Desktop  
Starting GNOME Desktop  
[ OK ] Started Update Manager  
[FAILED] Failed to start  
See "systemctl status gdm"
```

```
Ubuntu 15.04 linaro-gnome tty, 30
```

```
linaro-gnome login: root (automatic login)
```

```
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.
```

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.
```

```
root@linaro-gnome:~# ls  
root@linaro-gnome:~# cd ..  
root@linaro-gnome:/# ls  
bin dev home lost+found media opt root sbin sys usr  
boot etc lib md5sum.txt mnt proc run srv tmp var  
root@linaro-gnome:/#
```

```
0005-gc29bed9-dirty (May 23 2017 - 10:09:46)
```

```
bled
```

```
ti: 0
```

```
FL128S_64K with page size 256 Bytes, erase size 64 KiB, total 1B
```

```
0000
```

```
top autoboot: 0
```

```
ci
```

```
3
```

```
0000
```

```
2
```

```
es
```

```
1B
```

```
and file uEnv.txt **
```

```
Copying Linux from SD to RAM...
```

```
reading uImage
```

```
3447992 bytes read in 303 ms (10.9 MiB/s)
```

```
reading devicetree.dtb
```

```
7406 bytes read in 15 ms (481.4 KiB/s)
```

```
## Booting kernel from Legacy Image at 03000000 ...
```

```
Image Name: Linux-3.18.0-xilinx-46110-gd627f
```

```
Image Type: ARM Linux Kernel Image (uncompressed)
```

```
Data Size: 3447928 Bytes = 3.3 MiB
```

```
Load Address: 00008000
```

```
Entry Point: 00008000
```

```
Verifying Checksum ... OK
```

```
## Loading ramdisk from FIT Image at 02a00000 ...
```

```
Bad FIT ramdisk image format!
```

```
Ramdisk image is corrupt or invalid
```

```
zynq-uboot>
```

DSP 지난, 이번 주 목표

15일	16일	17일	18일	19일	20일	21일
리눅스 디바이스 드라이버 기초 조사 및 책 대여 1/3	리눅스 디바이스 드라이버 기초 조사 및 책 대여 2/3	리눅스 디바이스 드라이버 기초 조사 및 책 대여 3/3	LDD 위한 Kernel 및 LDD 예제 조사 1/4	LDD 위한 Kernel 및 LDD 예제 조사 2/4	LDD 위한 Kernel 및 LDD 예제 조사 3/4	LDD 위한 Kernel 및 LDD 예제 조사 4/4

22일	23일	24일	25일	26일	27일	28일
LDD 간단한 드라이버 작성 실습 1/4	LDD 간단한 드라이버 작성 실습 2/4	LDD 간단한 드라이버 작성 실습 3/4	LDD 간단한 드라이버 작성 실습 4/4	DSP or FPGA 데이터시트 연계한 LDD 작성 1/3	DSP or FPGA 데이터시트 연계한 LDD 작성 2/3	DSP or FPGA 데이터시트 연계한 LDD 작성 3/3

변경 전

변경 후

LDD 간단한 드라이버 작성 실습 1/4	LDD 간단한 드라이버 작성 실습 2/4	LDD 공동 작업 (I2C 문서화) 1/8	LDD 공동 작업 2/8	LDD 공동 작업 3/8	LDD 공동 작업 4/8	LDD 공동 작업 5/8
------------------------	------------------------	-------------------------	---------------	---------------	---------------	---------------

DSP 지난, 이번 주 성과(5.17)

DSP LDD 위한 Linux Kernel LDD 기초 조사

- 커널 프로그래밍이란
 1. linux kernel core 기능 추가
 2. linux kernel 알고리즘 개선
 3. linux kernel 모듈 프로그래밍

- 수행 방법에서의 차이
app : 순차적 수행
kernel : sys_call, interrupt handler 처리를 위한 비동기적 수행

- kernel의 라이브러리
: 필요한 라이브러리를 link하여 사용할 수 있는 app에 반해
kernel은 kernel에서 export하는 것들만 사용 가능
(linux kernel 개발 커뮤니티에서 배포한 것들만 사용 가능하다는 것인듯?)

- 라이브러리 include 주의사항 : /usr/include/linux, /usr/include/asm 아래에 선언된 헤더파일 만들 include 해야 한다.

- 메모리 접근
app : 직접 접근 및 특정 부분에 대한 접근이 제한
kernel : 제한 없음

- 메모리 영역
: app과 kernel은 서로 다른 매핑 방법으로 메모리 영역에 접근하며
서로 다른 영역을 갖고 있다 (64비트에서 1:1)

- Namespace Pollution
: 프로그램 내에서만 함수 이름들을 구별하는 app과 달리
kernel에서는 현재 개발하는 모듈 외에
전반적인 함수와 변수(함수 포인터 등)들의 이름이 충돌하지 않아야 한다.

- Symbol과 prefix
: kernel 모듈 개발 시 외부 파일과 link할 것은 EXPORT_SYMBOL(name)으로 symbol 테이블에 등록하고
link 하지 않을 파일은 static으로 선언.

전역 변수는 prefix를 붙여준다 ex) sys_open()

디바이스 드라이버의 이해

- 목적
문자 디바이스 드라이버
- 디바이스 드라이버의 개요
디바이스 드라이버는 커널의 일부분인 소프트웨어로
실제 물리 장치인 하드웨어를 제어/관리 한다.
- 커널 인터페이스 함수
-> Port I/O
-> Interrupt
-> Memory
-> Synchronization
-> Kernel message print
-> Device Driver register
- 디바이스 드라이버의 종류와 그 특징
 - 디바이스 드라이버 종류
-> 문자 드라이버 : 파일 단위로 I/O 수행, 스트림 방식
-> 블록 드라이버 : 디스크와 같은 파일 시스템 기반, 블록 단위 I/O
-> 네트워크 드라이버 : 네트워크 물리 계층 있고, 프레임 단위 데이터 I/O
 - 특징
응용 애플리케이션 부분
----- 응용 애플리케이션 레벨(시스템 콜)
(커널 레벨)
프로세스 관리, 메모리 관리, 파일 시스템, ...
각종 디바이스

(하드웨어 레벨)
CPU, RAM, ...
- 문자 디바이스 드라이버의 동작과정
 - 시스템 콜
process read() -> sys_read() -> bread()
-> hd_request() -> hd_io() -> HARDWARE
 - 시스템 콜 동작 과정
user task -> libc.a -> IDT -> (tabl)sys_fork() -> Sys_fork()
 - 문자 디바이스 드라이버 동작
커널과 애플리케이션 둘 모두에 open, write, read, close를 구현한 후
프로그램 단에서 호출 반환 반복하며 애플리케이션 단에서 실제 커널 그리고
최종적으로 하드웨어 제어.

[memory addressing, interrupts등과 같은 마이크로컴퓨터 내부구성 요소에 대한 이해를 필요로 한다. 이러한 것들은 assembly(:12) 프로그래머의 필수 사항이기도 하다.

User Space와 Kernel space

프로그래머와 하드웨어 간의 인터페이스(가교)역할을 담당하는게 device driver이다.
이러한 장치드라이버는 커널의 일부분으로써 작동하기 때문에 Kernel space 영역에서 작동한다고 말한다.

유닉스 shell(:12)이나 GUI 애플리케이션들은 user space에서 작동을 한다. 이러한 애플리케이션도 작동을 위해서는 시스템의 하드웨어를 제어해야한다.
커널에서 제공하는 함수를 통해서, 요청형식으로만 제어를 할 수 있다.

유저영역과 커널영역의 인터페이스 함수

커널은 애플리케이션 프로그래머가 하드웨어를 제어할 수 있도록 도와주기 위한 여러가지 **서브루틴과 함수들을 제공한다.**

디바이스 드라이버 모듈을 커널에 적재하려면, 디바이스 재조정, 메모리/interrupts 할당, 입/출력 포트 할당등과 같은 여러가지 선행작업이 필요하다.

이러한 작업들은 커널영역에서 이루어지며, 이러한 작업을 위해서 **module_init와 module_exit** 라는 2개의 함수를 제공한다.

이들 함수는 각각 유저모드 명령어인 insmod와 rmmod에 대응되어서 작동이 된다.

여기에서는 printk라는 함수가 있는데, 커널영역에서만 작동하는 걸 제외하고는 printf(3) 함수와 동일한 일을 한다. <1> 은 메시지의 우선순위를 정하기 위해서 사용된다. 1은 높은 우선순위를 가지고 있음을 의미한다. **이들 메시지들은 커널 로그 파일에 남겨지게 된다.**

감사합니다