RISC-V 리눅스 커널 및 디바이스드라이버 포팅 방법

마우저 후원 웨비나

RISC-V 리눅스 커널 및 디바이스드라이버 포팅 방법

커널연구회 (www.kernel.bz)

정재준 (rgbi3307@nate.com)

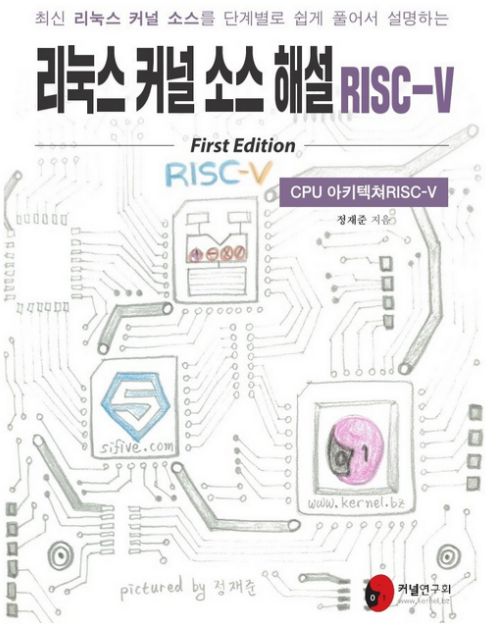
발표자 소개

정재준 (rgbi3307@nate.com)

커널연구회(www.kernel.bz)

저자는 리눅스 커널과 자료구조 알고리즘 관련 서적 집필과 세미나, 교육, 연구 활동을 활발히 하고 있습니다. 자료구조 알고리즘 연구 결과물로 특허 등록한 “맞춤형 문장 자동 번역 시스템 및 이를 위한 데이터베이스 구축방법 (The System for the customized automatic sentence translation and database construction method)”이 있습니다. 저자가 컴퓨터구조와 리눅스 커널, 자료구조 알고리즘 효율성, 머신러닝에 대해서 연구한 결과물들은 커널연구회 웹 페이지에 공유하고 있습니다. 최근에는 리눅스 커널 소스 patchwork.kernel.org 를 통해서 RISC-V 구조에 기여하고 있습니다.

집필서적:



목차

Table of Contents

RISC-V 개발 환경

riscv 구조를 실습해 볼수 있는 보드는 32 비트(HiFive1)와 64 비트(HiFive Unleashed)가 있습니다.

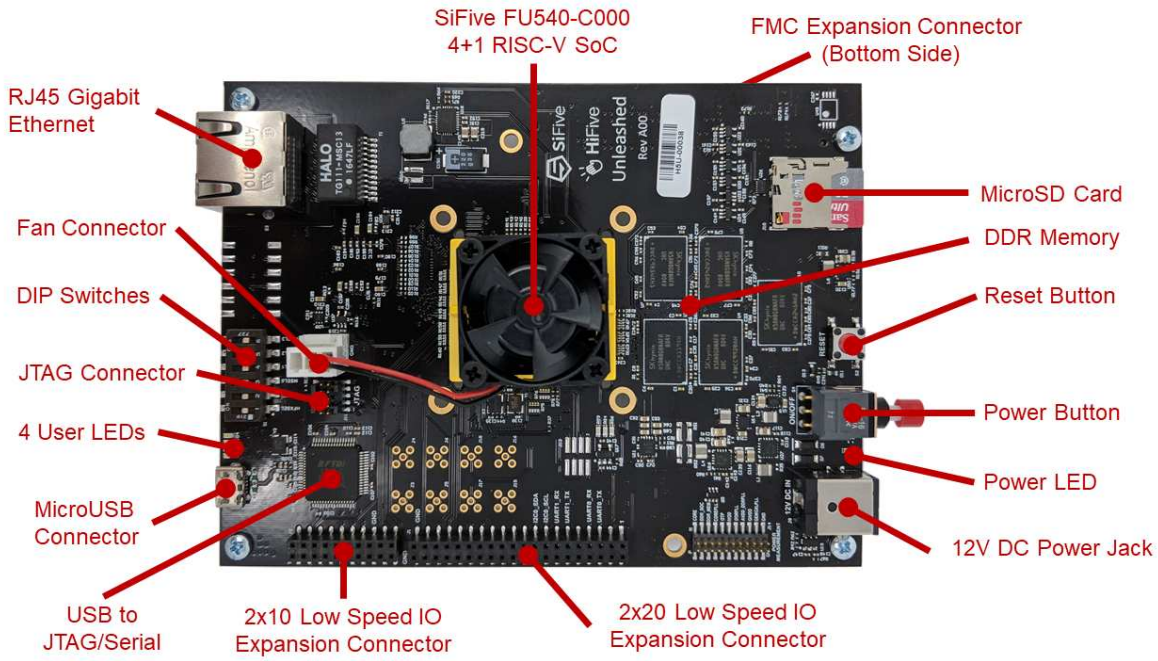
보드에 대한 자세한 정보는 아래의 웹사이트에 있습니다.

https://www.sifive.com/boards

RISC-V 64 비트 FU540

HiFive Unleashed 보드는 SiFive 의 64 비트 RISC-V 코어(E51 코어 1 개, U54 코어 4 개)가 내장된 FU540(Freedom U540)이 탑재되어 있습니다.

HiFive Unleashed(FU540) 보드 외형



RISC-V SDK 소스 설치

위에서 빌드한 SiFive 사의 freedom-u-sdk 는 부트로더, 커널, 램디스크를 한꺼번에 빌드할 수 있도록 Makefile 이 구성되어 있습니다. make 를 한번만 실행하면 모두 자동으로 빌드 되므로 간편하지만, 부트로더, 커널, 램디스크를 어떻게 버전 관리하고 있는지 파악하기 힘듭니다. 특히 커널 소스는 버전별로 소스를 분석하고 업그레이드도 자주 발생하기 때문에 소스를 따로 버전 관리하는 것이 오히려 효율적입니다. RISC-V 보드를 빨리 포팅해서 사용하는데 목적이 있다면 freedom-u-sdk 가 편리하지만, 소스를 하나씩 분석 및 연구하고자 한다면 소스를 따로 설치하여 관리하는 것이 좋습니다. 다행히 riscv.org 에서 대부분의 소스들을 아래 웹사이트에 공개하고 있으므로 소스를 분석 및 연구하고자 하는 독자분들에게는 많은 도움이 될듯합니다.

SDK 소스 배포:
https://github.com/riscv

위의 웹사이트에서 컴파일러, 부트로더, 커널, 램디스크, 시뮬레이터 소스등을 모두 다운로드 받을 수 있습니다. 지금부터는 이 소스들을 어떻게 빌드하고 버전 관리하는지 설명하도록 하겠습니다.

컴파일러(toolchain) 소스

RISC-V 컴파일러는 아래 웹사이트를 참조하여 소스를 다운로드하고 빌드할 수 있습니다.

참조 사이트:
https://github.com/riscv/riscv-gnu-toolchain

소스는 아래와 같이 git clone 으로 다운로드 하는 것이 버전관리하기에 좋습니다.

소스 다운로드

```
$ git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
```

위의 git clone 명령으로 소스가 다운로드 완료 되면 다음과 같은 명령들을 실행하여 빌드합니다.

소스 빌드

```
$ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf
libtool patchutils bc zlib1g-dev libexpat-dev


$ mkdir /opt/riscv
$ mkdir build
$ cd build
$ ../configure --prefix=/opt/riscv --enable-multilib
$ make linux
```

빌드시간은 컴퓨터 속도에 따라서 차이가 있지만 약 3~4 시간정도 소요됩니다. 컴파일 완료되면 실행 파일은 /opt/riscv 경로에 다음과 같이 생성됩니다.

```
jungjaejoon@HP7100U:/opt/riscv$ ll
total 36
drwxrwxr-x 9 risc-v risc-v 4096  4월  4 08:28 ./
drwxrwxrwx 5 root   root   4096  4월  4 08:05 ../
drwxrwxr-x 2 risc-v risc-v 4096  4월  4 10:57 bin/
drwxrwxr-x 3 risc-v risc-v 4096  4월  4 10:57 include/
drwxrwxr-x 3 risc-v risc-v 4096  4월  4 10:57 lib/
drwxrwxr-x 3 risc-v risc-v 4096  4월  4 08:28 libexec/
drwxrwxr-x 7 risc-v risc-v 4096  4월  4 10:45 riscv64-unknown-linux-gnu/
drwxrwxr-x 6 risc-v risc-v 4096  4월  4 10:57 share/
drwxrwxr-x 9 risc-v risc-v 4096  4월  4 09:20 sysroot/
```

컴파일러 버전은 다음과 같이 확인합니다. 필자가 소스를 빌드한 시점의 버전은 8.3.0 입니다.

jungjaejoon@HP7100U:/opt/riscv$ ./bin/riscv64-unknown-linux-gnu-gcc -v
Using built-in specs.
COLLECT_GCC=./bin/riscv64-unknown-linux-gnu-gcc
COLLECT_LTO_WRAPPER=/opt/riscv/libexec/gcc/riscv64-unknown-linux-gnu/8.3.0/lto-wrapper
Target: riscv64-unknown-linux-gnu
Configured with: /home/risc-v/kernel/riscv-gnu-toolchain/build/../riscv-gcc/configure --target=riscv64-unknown-linux-gnu --prefix=/opt/riscv --with-sysroot=/opt/riscv/sysroot --with-system-zlib --enable-shared --enable-tls --enable-languages=c,c++,fortran --disable-libmudflap --disable-libssp --disable-libquadmath --disable-nls --disable-bootstrap --src=../../riscv-gcc --enable-checking=yes --enable-multilib --with-abi=lp64d --with-arch=rv64imafdc --with-tune=rocket 'CFLAGS_FOR_TARGET=-O2 -mcmodel=medlow' 'CXXFLAGS_FOR_TARGET=-O2 -mcmodel=medlow'
Thread model: posix
gcc version 8.3.0 (GCC)

컴파일러에 관한 좀더 자세한 정보는 아래 웹사이트에서 확인 가능합니다.

참조: https://gnu-mcu-eclipse.github.io/toolchain/riscv/

컴파일 실행파일이 있는 경로는 다음과 같이 사용자 계정 설정정보 파일인 .bashrc 파일에 PATH 을 설정하여 어떤 위치에서든지 실행가능 하도록 해줍니다.

~/.bashrc 파일 내용에 PATH 추가

```
export PATH=$PATH:/opt/riscv/bin:/opt/riscv/riscv64-unknown-linux-gnu/bin
```

부트로더 소스

SiFive 사의 FU540 을 위한 부트로드는 riscv-pk 와 OpenSBI 그리고 u-boot 가 있습니다. riscv-pk 는 freedom-u-sdk 에 있는 소스 입니다. 이 소스는 다음과 같이 다운로드 및 빌드할 수 있습니다.

riscv-pk

riscv-pk 는 RISC-V Proxy Kernel 을 의미하며며 버클리 대학에서 제작한 Boot Loader 입니다. riscv-pk 는 오픈 소스이며 아래에서 소스를 다운로드하여 bbl(boot loader, kernel) 이미지를 만들 수 있습니다. bbl 에는 boot loader 와 커널 이미지가 같이 포함되어 있습니다.

소스 다운로드 링크:
https://github.com/riscv/riscv-pk

소스는 git clone 명령으로 다음과 같이 다운로드 합니다.

소스 다운로드

```
$ git clone https://github.com/riscv/riscv-pk.git
```

소스가 많지 않으므로 금방 다운로드 됩니다. 다운로드가 완료되면 다음과 같이 소스를 빌드 합니다.

아래에서 –prefix=$RISCV 옵션은 컴파일되어서 실행파일들이 설치되는 경로 입니다. 필자는

$RISCV 을 소스가 빌드되는 경로인 "/home/risc-v/kernel/riscv-pk/build"로 지정 했습니다.

소스 빌드

```
$ mkdir build
$ cd build
$ ../configure --prefix=$RISCV --host=riscv64-unknown-elf
$ make
```

컴파일이 완료되면 실행파일들은 /home/risc-v/kernel/riscv-pk/build/work 경로에 생성됩니다.

work 경로에 bbl 파일(bootloader+kernel+logo)이 생성됩니다. 이 파일 생성은 다음과 같이 실행

스크립트를 통해서 생성 됩니다.

bbl_make.sh

```
#!/bin/bash

WORK_PATH=$PWD

if [ "$1" == "" ]; then
    echo "Usage ./bbl_make.sh fullpath"
else
    rm -rf $WORK_PATH/work
    mkdir -p $WORK_PATH/work

    cd $WORK_PATH/work

    ../../configure \
```

```
        --prefix=$WORK_PATH/work \
        --host=riscv64-unknown-linux-gnu \
        --with-payload=$1 \
        --enable-logo \
        --with-logo=$WORK_PATH/conf/sifive_logo.txt

    CFLAGS="-mabi=lp64d -march=rv64imafdc"
    make -C $WORK_PATH/work

    riscv64-unknown-linux-gnu-objcopy -S -O binary --change-addresses -
0x80000000 \
    $WORK_PATH/work/bbl $WORK_PATH/bbl.bin

    xxd -c1 -p $WORK_PATH/bbl.bin > $WORK_PATH/bbl.hex
fi
```

필자의 경우 작업경로(WORK_PATH) 는 /home/risc-v/kernel/riscv-pk/build/ 입니다.

그리고 위의 스크립트를 실행할때 첫번째 파라미터($1)로 전달하는 것은 커널 이미지 파일(vmlinux-stripped) 입니다. 위의 스크립트를 다음과 같이 실행하면 작업경로에 bbl.bin 파일이 생성됩니다.

bbl_make.sh 실

```
$ cd /home/risc-v/kernel/riscv-pk/build/


$ ./bbl_make.sh /home/risc-v/kernel/build/build_v5.0_riscv_origin/vmlinux-
stripped
```

생성한 bbl.bin 파일을 SD 카드에 설치하는 방법은 앞장(FU540 SDK 빌드 미치 설치)에 이미 기술하였으므로 참조해 주시기 바랍니다.

OpenSBI

OpenSBI 는 RISC-V 전용 부트로더로서 Supervisor Binary Interface (SBI) 입니다. 필자는 처음에 SiFive 사에서 배포한 freedom-u-sdk 패키지 안에 있는 riscv-pk 을 부트로더로 사용하였으나, 이 책을 집필하는 동안 리눅스 커널이 v5.2 이상으로 버전업 되면서 디바이스 트리가 적용 되었습니다. 그래서 커널에 디바이스트 트리를 적용하기 위한 부트로더로서 OpenSBI 을 선택했습니다.

OpenSBI 소스는 아래 GitHub 에서 다운로드 할 수 있습니다.

https://github.com/riscv/opensbi

소스는 다음과 같이 git clone 명령으로 다운로드 합니다. 소스가 크기 않기 때문에 금방 다운로드 됩니다.

소스 다운로드

```
$ git clone https://github.com/riscv/opensbi
```

소스가 다운로드 되면, 소스가 있는 opensbi 경로로 들어가서 다음과 같이 소스 파일 목록을 확인합니다.

소스 내용 확인

```
$ cd opensbi/
$ ll
///소스 파일 목록
```

```
drwxrwxr-x 9 riscv-stable riscv-stable  4096  1월  9 11:41 ./
drwxrwxr-x 5 riscv-stable riscv-stable  4096  1월  9 11:40 ../
-rw-rw-r-- 1 riscv-stable riscv-stable   390  1월  9 11:41 .clang-format
-rw-rw-r-- 1 riscv-stable riscv-stable   643  1월  9 11:41 CONTRIBUTORS.md
-rw-rw-r-- 1 riscv-stable riscv-stable  1396  1월  9 11:41 COPYING.BSD
drwxrwxr-x 5 riscv-stable riscv-stable  4096  1월  9 11:41 docs/
drwxrwxr-x 3 riscv-stable riscv-stable  4096  1월  9 11:41 firmware/
drwxrwxr-x 8 riscv-stable riscv-stable  4096  1월  9 11:41 .git/
-rw-rw-r-- 1 riscv-stable riscv-stable   110  1월  9 11:41 .gitignore
drwxrwxr-x 4 riscv-stable riscv-stable  4096  1월  9 11:41 include/
drwxrwxr-x 4 riscv-stable riscv-stable  4096  1월  9 11:41 lib/
-rw-rw-r-- 1 riscv-stable riscv-stable 14859  1월  9 11:41 Makefile
drwxrwxr-x 9 riscv-stable riscv-stable  4096  1월  9 11:41 platform/
-rw-rw-r-- 1 riscv-stable riscv-stable 10053  1월  9 11:41 README.md
drwxrwxr-x 2 riscv-stable riscv-stable  4096  1월  9 11:41 scripts/
-rw-rw-r-- 1 riscv-stable riscv-stable   577  1월  9 11:41
ThirdPartyNotices.md
```

위의 소스를 빌드하는 방법은 아래 웹사이트에 자세히 설명되어 있습니다.

https://github.com/riscv/opensbi/blob/master/docs/platform/sifive_fu540.md

필자는 위의 문서를 참고하여 다음과 같이 소스 빌드 스크립트를 만들었습니다.

소스 빌드 스크립트

```
#!/bin/bash
#$1 = version

COMPILER="riscv64-unknown-linux-gnu-"

make CROSS_COMPILE=$COMPILER PLATFORM=sifive/fu540 \
 FW_PAYLOAD_PATH=/home/risc-v/kernel/build/build_$1_riscv_origin/arch/
riscv/boot/Image \
 FW_PAYLOAD_FDT_PATH=/home/risc-v/kernel/build/build_$1_riscv_origin/arch/
riscv/boot/dts/sifive/hifive-unleashed-a00.dtb
```

```
cp build/platform/sifive/fu540/firmware/fw_payload.bin build/kernel-dtb
```

위에서 FW_PAYLOAD_PATH 는 필자가 커널을 빌드하여 Image 파일이 생성되어 있는 경로 입니다. FW_PAYLOAD_FDT_PATH 는 커널 v5.2 부터 FU540 을 위한 디바이스 트리를 빌드한 dtb 파일이 있는 경로 입니다. 이렇게 하면 디바이스 트리 정보가 커널에 적용 됩니다.

위의 빌드 스크립트를 실행할때 전달 하는 옵션($1)이 하나 있는데, 이것은 리눅스 커널 버전 입니다. 필자가 버전별로 빌드하기 위한 용도로 추가 한 것입니다. 예를들어 커널 버전 v5.4.8 을 빌드 했다면 이 스크립트를 실행할때 옵션($1)에 v5.4.8 을 입력합니다.

빌드가 완료되면 최종 이미지 파일(OpenSBI 부트로더 + 커널 + 디바이스트리)이 build/platform/sifive/fu540/firmware/fw_payload.bin 에 생성 됩니다. 이 파일을 SD 카드에 인스톨하면 됩니다. SD 카드에 설치하는 방법은 앞장(FU540 SDK 빌드 미치 설치)에 이미 기술 하였으므로 참조해 주시기 바랍니다.

u-boot

u-boot 는 freedom-u-sdk 와는 별도로 소스를 분석하고 빌드할 수 있습니다. riscv 부트로더로 앞에서 기술한 riscv-pk 또는 OpenSBI 을 사용해도 되지만, u-boot 가 친숙한 독자분들은 아래 내용을 참조해 주시기 바랍니다. u-boot 사용법은 아래 웹사이트에 설명되어 있습니다.

참조 사이트:
https://github.com/sifive/HiFive_U-Boot

소스는 git clone 명령으로 다음과 같이 다운로드 합니다.

소스 다운로드

```
$ git clone https://github.com/Microsemi-SoC-IP/HiFive_U-Boot
```

소스 빌드는 다음과 같이 명령을 실행합니다.

소스 빌드

```
$ make HiFive-U540_defconfig
$ make
```

u-boot 는 예전부터 임베디드 환경의 부트로더로 범용적으로 사용되고 있어서, u-boot 을 많이 사용해본 독자분들은 금방 FU540 에 포팅할 수 있을듯 합니다. 필자는 freedom-u-sdk 에 있는 riscv-pk 와 OpenSBI 을 사용하고 있어서, riscv 부트로더를 u-boot 로 대체할 필요가 있을때 좀더 내용을 기술하도록 하겠습니다.

커널 소스

커널 소스는 Linus Torvalds 에 의해서 아래 웹사이트에서 배포되고 있습니다.

참조 사이트:
https://github.com/torvalds/linux

리눅스 커널 소스 안정화(stable) 버전은 아래와 같이 git clone 으로 다운로드 받습니다. Git 관련 정보까지 모두 받기 때문에 다운로드 되는 시간이 상당히 오래 걸립니다. (약 3~4 시간 정도 소요됨)

소스 다운로드

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git (origin)
```

다운로드가 모두 되면 linux-stable 경로가 생성되어 있습니다. 이 경로에 들어가서 아래와 같이 git log -p 명령으로 최신 커밋 정보를 보면 리눅스 커널 소스 버전을 확인할 수 있습니다.

```
$ git log -p
commit e93c9c99a629c61837d5a7fc2120cd2b6c70dbdd (HEAD -> CV5.1, tag: v5.1, origin/master,
origin/linux-5.1.y, origin/HEAD, cv5.1)
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Sun May 5 17:42:58 2019 -0700

    Linux 5.1
```

필자가 이 책을 집필하는 시점의 커널 최신 버전은 v5.1 입니다. 독자 분들이 이책을 구독하는 시점에는 버전이 더 올라갔을 것입니다. 커널이 지속적으로 업그레이드 되더라도 기본적인 설계사상과 빌드 방법은 크게 달라 지지 않습니다. 아래부터 커널 소스를 빌드하는 방법과 riscv 아키텍쳐인 FU540 을 포팅하는 방법에 대해서 계속 기술 하도록 하겠습니다.

커널 소스 빌드하는 방법은 비교적 간단합니다.

소스 빌드

```
$ cd linux-stable

$ make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu-  defconfig
```

위와 같이 커널 소스 경로(linux-stable)에서 make 명령을 실행하면 됩니다. 실행 옵션으로 ARCH=riscv 이고 크로스 컴파일러는 riscv64-unknown-elf- 입니다. 그리고 커널을 빌드할때 참조하는 환경설정 파일은 최초에는 defconfig 로 합니다. defconfig 옵션에 의해서 .config 파일이 생성됩니다. 이 파일이 생성된 다음부터는 defconfig 대신에 menuconfig 옵션으로 .config 파일의 내용을 수정할 수 있습니다. 앞으로 커널을 포팅하면서 .config 파일을 변경하는 내용들은 계속 언급되기 때문에 그때마다 설명하도록 하겠습니다.

.config 파일

```
CONFIG_RISCV=y
CONFIG_MMU=y
CONFIG_ARCH_PHYS_ADDR_T_64BIT=y
CONFIG_ARCH_DMA_ADDR_T_64BIT=y
CONFIG_PAGE_OFFSET=0xffffffe000000000
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_RWSEM_GENERIC_SPINLOCK=y
CONFIG_GENERIC_BUG=y
CONFIG_GENERIC_BUG_RELATIVE_POINTERS=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_GENERIC_CSUM=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_PGTABLE_LEVELS=3
CONFIG_DMA_NOOP_OPS=y

# Platform type
# CONFIG_ARCH_RV32I is not set
CONFIG_ARCH_RV64I=y
# CONFIG_CMODEL_MEDLOW is not set
CONFIG_CMODEL_MEDANY=y
# CONFIG_MAXPHYSMEM_2GB is not set
CONFIG_MAXPHYSMEM_128GB=y
CONFIG_SMP=y
```

```
CONFIG_NR_CPUS=8
CONFIG_CPU_SUPPORTS_64BIT_KERNEL=y
CONFIG_TUNE_GENERIC=y
CONFIG_RISCV_ISA_C=y
CONFIG_RISCV_ISA_A=y

# Kernel type
CONFIG_64BIT=y
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
CONFIG_HAVE_MEMBLOCK=y
CONFIG_NO_BOOTMEM=y
CONFIG_SPLIT_PTLOCK_CPUS=4
CONFIG_COMPACTION=y
CONFIG_MIGRATION=y
CONFIG_PHYS_ADDR_T_64BIT=y
CONFIG_DEFAULT_MMAP_MIN_ADDR=4096
CONFIG_PREEMPT_NONE=y
CONFIG_HZ_250=y
CONFIG_HZ=250

//이하 생략..
```

buildroot(ramfs)

루트 파일 시스템은 아래 웹사이트 소스를 다운로드 받아서 제작할 수 있습니다.

https://buildroot.org/

소스는 다음과 같이 git clone 으로 다운로드 하여 최신 버전으로 체크아웃 합니다.

소스 다운로드

```
$ git clone git://git.buildroot.net/buildroot

$ cd buildroot/
```

```
$ git tag | grep 2018.11.3
$ git checkout -b CV20181103 2018.11.3
```

소스 빌드는 다음과 같이 make menuconfig 로 환경설정 한후, make 을 실행하여 빌드합니다. 빌드 결과는 output 경로에 생성 됩니다.

소스 빌드

```
$ make menuconfig
///환경설정


$ make
```

파일 시스템 생성에 필요한 소스와 패키지들을 인터넷에서 다운로드 하면서 빌드 되기 때문에 빌드시간이 꽤 오래 걸립니다.(1 시간 이상) 참고로 환경설정에서 너무 많은 파일 시스템 패키지를 설정하면 용량이 커져서 오류가 발생합니다. 또한 커널 버전과도 맞아야 합니다.
buildroot 사용법에 대해서는 https://buildroot.org/ 사이트 또는 인터넷에 많이 공유되어 있으므로 참조하시면 되겠습니다.

위에서 buildroot 로 빌드한 램디스크 파일들(rootfs)은 다음과 같이 커널 빌드 상수에 설정합니다.
menuconfig 메뉴에서 다음과 같이 설정합니다.

Menuconfig

```
Symbol: BLK_DEV_INITRD [=y]
  | Type  : bool
  | Prompt: Initial RAM filesystem and RAM disk (initramfs/initrd) support
  |   Location:
  |     -> General setup
  |   Defined at init/Kconfig:1213
```

```
Symbol: INITRAMFS_SOURCE [=/home/risc-v/kernel/conf/initramfs.txt
/home/risc-v/kernel/rootfs/v04]
  | Type  : string
  | Prompt: Initramfs source file(s)
  |   Location:
  |     -> General setup
  |       -> Initial RAM filesystem and RAM disk (initramfs/initrd)
support (BLK_DEV_INITRD [=y])
  |   Defined at usr/Kconfig:6
  |   Depends on: BLK_DEV_INITRD [=y]


Symbol: INITRAMFS_ROOT_UID [=1000]
  | Type  : integer
  | Prompt: User ID to map to 0 (user root)
  |   Location:
  |     -> General setup
  |       -> Initial RAM filesystem and RAM disk (initramfs/initrd)
support (BLK_DEV_INITRD [=y])
  |   Defined at usr/Kconfig:35
  |   Depends on: BLK_DEV_INITRD [=y] && INITRAMFS_SOURCE
[=/home/risc-v/kernel/conf/initramfs.txt /home/risc-v/kernel/rootfs/v03]!=


 Symbol: INITRAMFS_ROOT_GID [=1000]
  | Type  : integer
  | Prompt: Group ID to map to 0 (group root)
  |   Location:
  |     -> General setup
  |       -> Initial RAM filesystem and RAM disk (initramfs/initrd)
support (BLK_DEV_INITRD [=y])
  |   Defined at usr/Kconfig:45
  |   Depends on: BLK_DEV_INITRD [=y] && INITRAMFS_SOURCE
[=/home/risc-v/kernel/conf/initramfs.txt /home/risc-v/kernel/rootfs/v03]!=
```

riscv simulator

FU540 타겟 보드가 없어도 호스트 PC 에서 커널 소스를 시뮬레이팅하여 실행할 수 있습니다. 아래 웹사이트에 자세한 내용이 있으므로 참조 바랍니다.

참조 사이트:
https://github.com/riscv/riscv-isa-sim

소스 다운로드

```
$ git clone https://github.com/riscv/riscv-isa-sim.git
```

소스 빌드

```
$ apt-get install device-tree-compiler
$ mkdir build
$ cd build
$ ../configure --prefix=$RISCV
$ make
```

qemu

riscv 에서 제공하는 simulator(riscv-isa-sim)는 디바이스 트리를 적용하는데 어려움이 있어서, 필자는 qemu 을 사용하기로 했습니다. qemu 는 가장 많이 사용하고 있는 머신 에뮬레이터(the FAST! processor emulator)로서 아래 웹사이트에서 배포하고 있습니다.

https://www.qemu.org/

qemu 설치는 리눅스 우분투 터미널에서 다음과 같이 apt-get 으로 간단히 설치할 수 있습니다.

qemu 설치

```
$ sudo apt-get install qemu
```

아래 경로에 설치 됩니다.

```
/usr/lib/x86_64-linux-gnu/qemu
/usr/lib/qemu
/usr/share/qemu
/usr/bin/
```

qemu 실행파일들은 /usr/bin 경로에 있습니다. 이 경로의 내용을 다음과 같이 확인해 보면,

```
$ ll /usr/bin/qemu-system-*
-rwxr-xr-x 1 root root 12564688 11월  8 15:30 /usr/bin/qemu-system-
aarch64*
-rwxr-xr-x 1 root root 10076288 11월  8 15:30 /usr/bin/qemu-system-alpha*
-rwxr-xr-x 1 root root 12370192 11월  8 15:30 /usr/bin/qemu-system-arm*
-rwxr-xr-x 1 root root  4965472 11월  8 15:30 /usr/bin/qemu-system-cris*
-rwxr-xr-x 1 root root 11623776 11월  8 15:30 /usr/bin/qemu-system-i386*
-rwxr-xr-x 1 root root  5149824 11월  8 15:30 /usr/bin/qemu-system-lm32*
-rwxr-xr-x 1 root root  5183104 11월  8 15:30 /usr/bin/qemu-system-m68k*
-rwxr-xr-x 1 root root  4961216 11월  8 15:30 /usr/bin/qemu-system-
microblaze*
-rwxr-xr-x 1 root root  4961216 11월  8 15:30 /usr/bin/qemu-system-
microblazeel*
-rwxr-xr-x 1 root root 11081216 11월  8 15:30 /usr/bin/qemu-system-mips*
-rwxr-xr-x 1 root root 11225120 11월  8 15:30 /usr/bin/qemu-system-mips64*
-rwxr-xr-x 1 root root 11259904 11월  8 15:30 /usr/bin/qemu-system-
mips64el*
-rwxr-xr-x 1 root root 11081216 11월  8 15:30 /usr/bin/qemu-system-mipsel*
-rwxr-xr-x 1 root root  4917472 11월  8 15:30 /usr/bin/qemu-system-moxie*
-rwxr-xr-x 1 root root  4910592 11월  8 15:30 /usr/bin/qemu-system-nios2*
-rwxr-xr-x 1 root root  4964928 11월  8 15:30 /usr/bin/qemu-system-or1k*
-rwxr-xr-x 1 root root 11738208 11월  8 15:30 /usr/bin/qemu-system-ppc*
-rwxr-xr-x 1 root root 12280864 11월  8 15:30 /usr/bin/qemu-system-ppc64*
lrwxrwxrwx 1 root root       17 11월  8 15:30 /usr/bin/qemu-system-ppc64le
-> qemu-system-ppc64*
```

```
-rwxr-xr-x 1 root root 11435104 11월  8 15:30 /usr/bin/qemu-system-ppcemb*
-rwxr-xr-x 1 root root  6268288 11월  8 15:30 /usr/bin/qemu-system-s390x*
-rwxr-xr-x 1 root root  9922208 11월  8 15:30 /usr/bin/qemu-system-sh4*
-rwxr-xr-x 1 root root  9922208 11월  8 15:30 /usr/bin/qemu-system-sh4eb*
-rwxr-xr-x 1 root root  5559200 11월  8 15:30 /usr/bin/qemu-system-sparc*
-rwxr-xr-x 1 root root 10248992 11월  8 15:30 /usr/bin/qemu-system-
sparc64*
-rwxr-xr-x 1 root root  5017728 11월  8 15:30 /usr/bin/qemu-system-
tricore*
-rwxr-xr-x 1 root root  4864000 11월  8 15:30 /usr/bin/qemu-system-
unicore32*
-rwxr-xr-x 1 root root 11658464 11월  8 15:30 /usr/bin/qemu-system-x86_64*
-rwxr-xr-x 1 root root  5040256 11월  8 15:30 /usr/bin/qemu-system-xtensa*
-rwxr-xr-x 1 root root  5029760 11월  8 15:30 /usr/bin/qemu-system-
xtensaeb*
```

CPU 종류별로 qemu 실행파일들이 있으나 riscv 를 위한 qemu-system-riscv 는 없습니다. qemu 정식 배포본에 아직 riscv 는 적용 않되어 있는듯 합니다. 인터넷에서 검색해본 결과 qemu 소스를 직접 받아서 컴파일(옵션: --target-list=riscv64-softmmu)하여 설치하면 됩니다. qemu 소스는 git clone 으로 다운로드 받아서 다음과 같이 컴파일하여 설치합니다.

qemu 소스 다운로드 및 설치

```
$ git clone https://github.com/qemu/qemu
$ cd qemu
$ ./configure --target-list=riscv64-softmmu,riscv32-softmmu
$ make -j $(nproc)
$ sudo make install
```

qemu-system-riscv64 는 /usr/local/bin/경로에 설치 됩니다. 다음과 같이 qemu-system-riscv64 도움말을 확인합니다.

qemu-system-riscv64 도움말

```
$ qemu-system-riscv64 -h
```

```
QEMU emulator version 4.2.0 (v4.2.0-dirty)
Copyright (c) 2003-2019 Fabrice Bellard and the QEMU Project developers
usage: qemu-system-riscv64 [options] [disk_image]

'disk_image' is a raw hard disk image for IDE hard disk 0

Standard options:
-h or -help     display this help and exit
-version        display version information and exit
-machine [type=]name[,prop[=value][,…]]

//이하생략..
```

qemu-system-riscv64 사용에 대해서는 아래 웹사이트에서 예제별로 자세히 설명하고 있습니다.

https://twilco.github.io/

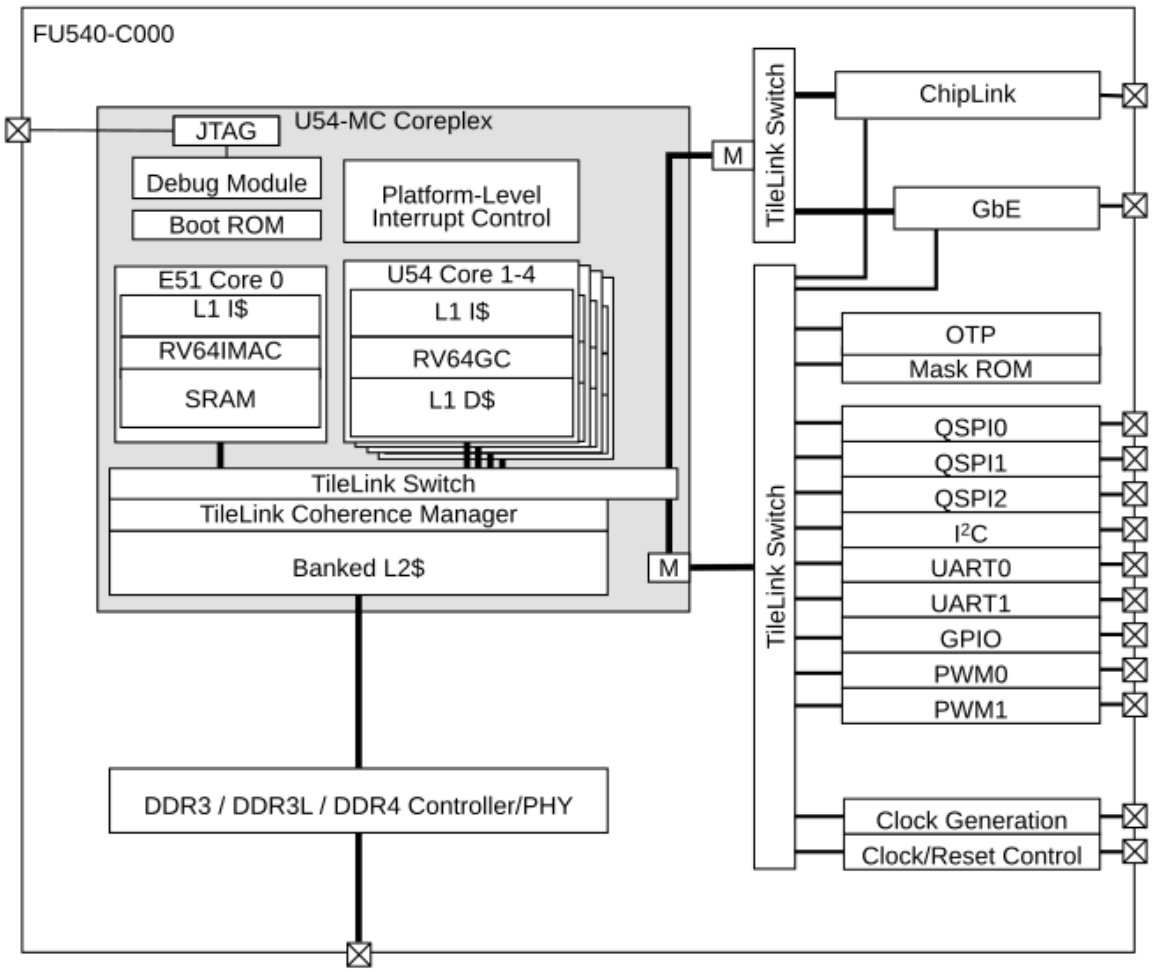
https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-qemu.html

필자도 위의 웹사이트를 참조하여 사용해 보고 있습니다. 그리고 qemu 는 인터넷 검색에서 풍부한 사용예제들을 볼 수 있기 때문에 이책에서는 여기까지만 언급하도록 하겠습니다.

FU540 하드웨어

FU540 은 64 비트 RISC-V 구조로 28nm 공정으로 설계한 SoC 보드 입니다. CPU 코어가 5 개로 구성되어 있으며 E51 코어 1 개, U54 코어 4 개 입니다. 모두 64 비트 명령셋으로 1.5GHz 속도로 동작합니다. E51 코어 내부에는 L1 명령 캐시와 SRAM 이 있고, U54 코어 내부에는 L1 명령 캐시와 데이터 캐시가 내장되어 있습니다. 아래는 FU540 구조를 블럭 다이어그램으로 설명하는 그림입니다.

FU540 블럭 다이어그램



CPU 코어 바깥쪽에는 2MiB 용량의 L2 캐시가 있습니다. 각각의 CPU 마다 내부 인터럽트(PLIC)가 처리됩니다. 외부 메모리는 8GB DDR3/4 가 64 비트 버스폭으로 연결되어 있습니다.

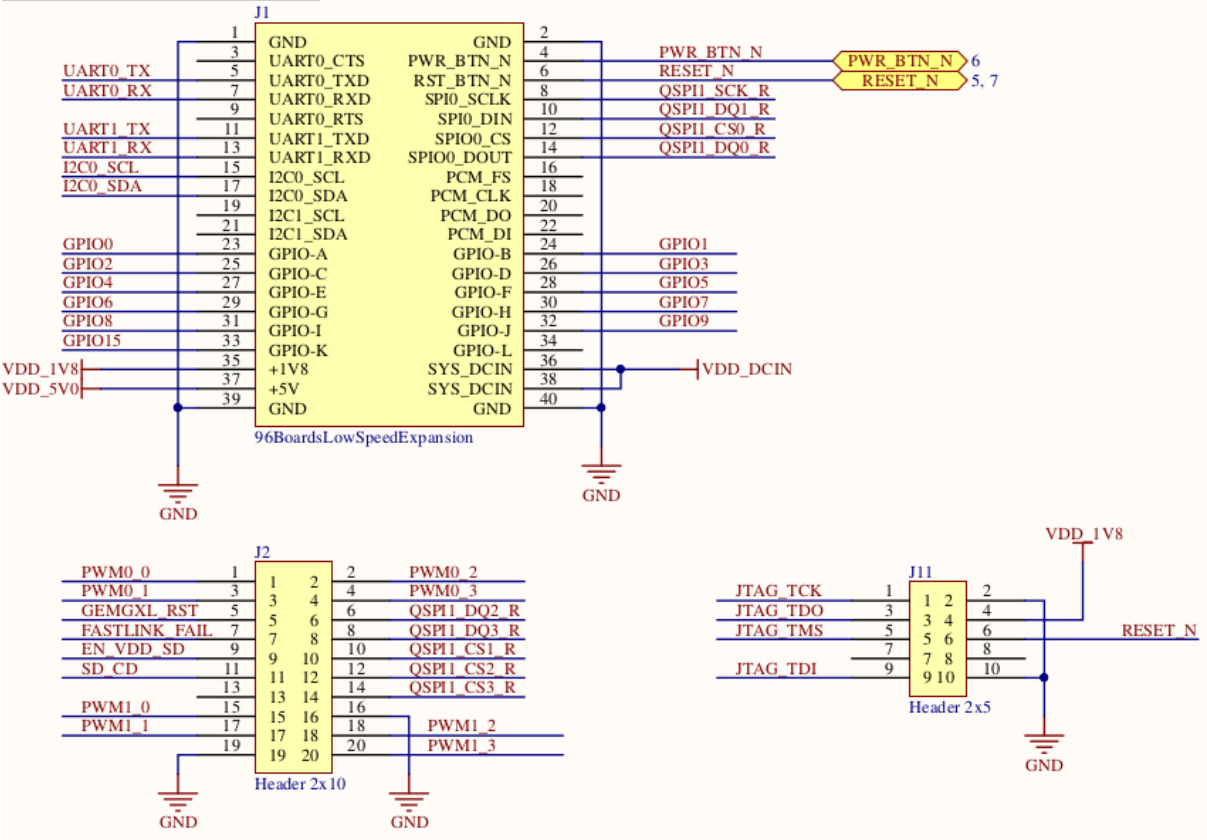
장치 인터페이스는 SPI 3 개, I2C 1 개, UART 2 개, GPIO 16 개, PWM 2 개가 내장되어 있고, 10/100/1000 Ethernet 장치가 연결되어 있습니다. SPI0 에는 32MB Flash 메모리가 연결되어 있고, SPI2 에는 마이크로 SD 카드가 연결되어 있습니다. UART0 는 디버그 용도로 마이크로 USB 포트에 연결되어 있습니다. PWM0 는 동작 상태를 표시하기 위한 LED(4 개)에 연결되어 있습니다. 아래는 지금까지 기술한 내용을 테이블로 정리한 것입니다.

Feature	Description
RISC-V Core	4× U54 RISC-V cores with machine, supervisor, and user mode, 32 KiB 8-way L1 I-cache, and 32 KiB 8-way L1 D-cache. 4× E51 RISC-V cores with machine and user mode, 16 KiB 2-way L1 I-cache, and 8 KiB data tightly integrated memory (DTIM).
L2 Cache	2 MiB 16-way coherent L2 cache.
Interrupts	Software and timer interrupts, 53 peripheral interrupts connected to the PLIC with 7 levels of priority.
DDR3/4 Controller	64 bit + ECC Memory Controller to external DDR3/DDR3L/DDR4 memory
UART 0	Universal Asynchronous/Synchronous Transmitters for serial communication.
UART 1	Universal Asynchronous/Synchronous Transmitters for serial communication.
QSPI 0	Serial Peripheral Interface. QSPI 0 has 1 chip select signal.
QSPI 1	Serial Peripheral Interface. QSPI 1 has 4 chip select signals.
QSPI 2	Serial Peripheral Interface. QSPI 2 has 1 chip select signal.
PWM 0	16-bit Pulse-width modulator with 4 comparators.
PWM 1	16-bit Pulse-width modulator with 4 comparators.
I²C 0	Inter-Integrated Circuit (I²C) controller.
GPIO	16 General Purpose I/O pins.
Gigabit Ethernet MAC	10/100/1000 Ethernet MAC with GMII interface to an external PHY.
OTP	4Kx32b one-time programmable memory.

출처: SiFive FU540-C000 Manual v1p0 (1 장)

위의 장치 인터페이스들 중에서 사용자가 외부에서 접근할 수 있는 것들은 대부분 J1, J2 헤더핀들에 연결되어 있습니다.

장치 인터페이스 헤더핀

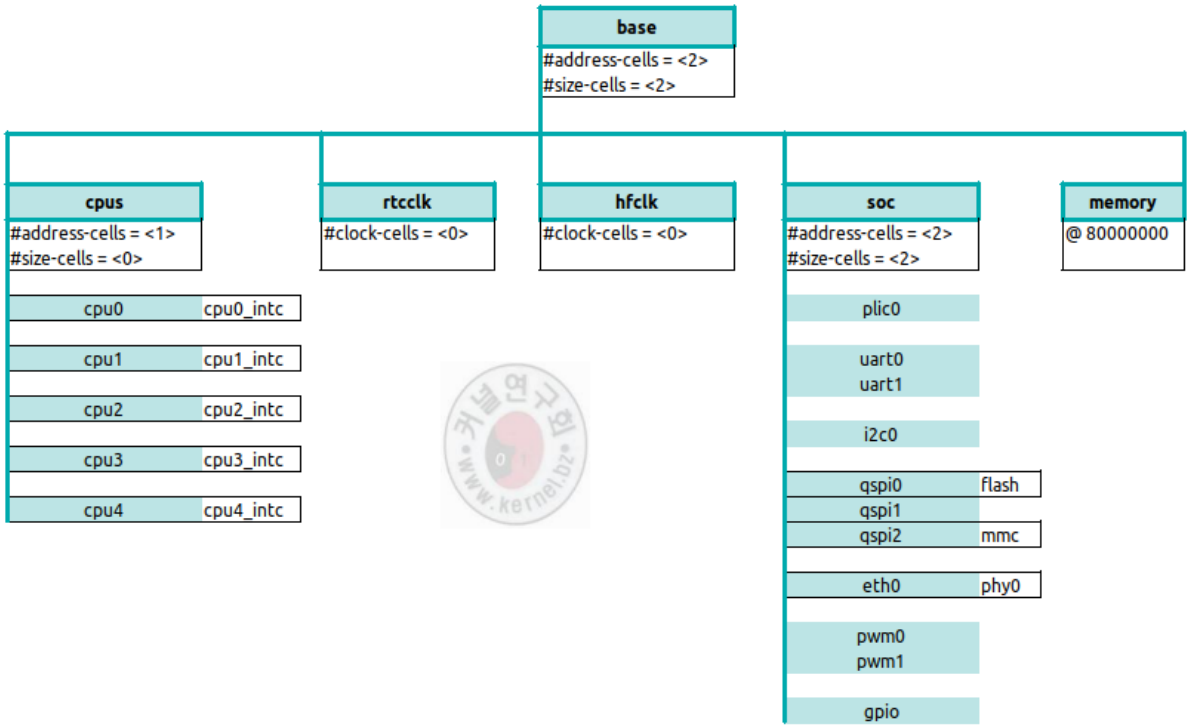


출처: SiFive FU540-C000 회로도

장치 계층 구조

FU540 하드웨어 장치들은 리눅스 커널에서 디바이스 트리로 표현 됩니다. 아래 그림으로 표현한 FU540 장치 계층도는 arch/riscv/boot/dts/sifive/ 경로에 있는 커널 디바이스 트리 소스를 필자가 분석하여 정리한 것입니다. 디바이스 트리에 대해서는 이 책에서 계속 설명이 됩니다만, 일단 계측구조를 보면 위에서 기술한 FU540 하드웨어 계층 구조와 일치 합니다.

Device Tree 장치계층도



FU540 하드웨어 장치들은 cpus, 클럭(rtcclk, hfclk), soc(입출력 인터페이스), memory 로 구성됩니다. cpus 하위 계층에 5 개의 cpu 코어가 배치되고, soc 하위 계층에 여러가지 장치들이 연결됩니다.

FU540 디바이스 트리 소스

커널의 시스템 파일 경로(/sys/firmware/devicetree/base)에 정보들을 노드별로 표출 합니다.

/sys/firmware/devicetree/base

```
RISCV-FU540:/sys/firmware/devicetree/base# ll

-r--r--r--    1 root     root             4 Jan  1 00:00 #address-cells
-r--r--r--    1 root     root             4 Jan  1 00:00 #size-cells
drwxr-xr-x    8 root     root             0 Jan  1 00:00 ./
drwxr-xr-x    3 root     root             0 Jan  1 00:00 ../
drwxr-xr-x    2 root     root             0 Jan  1 00:00 aliases/
drwxr-xr-x    2 root     root             0 Jan  1 00:00 chosen/
-r--r--r--    1 root     root            27 Jan  1 00:00 compatible
drwxr-xr-x    7 root     root             0 Jan  1 00:00 cpus/
drwxr-xr-x    2 root     root             0 Jan  1 00:00 firmware/
drwxr-xr-x    2 root     root             0 Jan  1 00:00 memory@80000000/
-r--r--r--    1 root     root            28 Jan  1 00:00 model
-r--r--r--    1 root     root             1 Jan  1 00:00 name
drwxr-xr-x   39 root     root             0 Jan  1 00:00 soc/

RISCV-FU540:/sys/firmware/devicetree/base# ls soc
#address-cells                 itim@1820000
#size-cells                    memory-controller@100b0000
bus-blocker@100b8000           name
cache-controller@2010000       pci@2000000000
cadence-ddr-mgmt@100c0000      pinctrl@10080000
cadence-gemgxl-mgmt@100a0000   prci@10000000
chiplink@40000000              pwm@10020000
compatible                     pwm@10021000
dma@3000000                    pwmleds
dtim@1000000                   ranges
ememoryotp@10070000            refclk
error-device@18000000          rom@1000
ethernet@10090000              rom@10000
gpio-restart                   rom@a000000
gpio@10060000                  serial@10010000
i2c@10030000                   serial@10011000
interrupt-controller@c000000   spi@10040000
itim@1800000                   spi@10041000
itim@1808000                   spi@10050000
```

```
itim@1810000                    teststatus@4000
itim@1818000                    tlclk
```

SiFive 사의 FU540 을 위한 디바이스 트리는 아래의 커널 소스 파일에 있습니다. 리눅스 커널 v5.2 부터 적용되었습니다.

FU540 디바이스 트리 파일들

```
/include/dt-bindings/clock/sifive-fu540-prci.h
/arch/riscv/boot/dts/sifive/fu540-c000.dtsi
/arch/riscv/boot/dts/sifive/hifive-unleashed-a00.dts
```

위의 파일들은 arch/riscv/boot/dts/sifive/Makefile 에 의해서 빌드 됩니다. 단,

CONFIG_SOC_SIFIVE 빌드 설정 상수가 [=y]로 되어 있어야 합니다.

arch/riscv/boot/dts/sifive/Makefile

```
dtb-$(CONFIG_SOC_SIFIVE) += hifive-unleashed-a00.dtb
```

CONFIG_SOC_SIFIVE 빌드 설정 상수는 커널 menuconfig 에서 다음과 같은 메뉴 위치에 있습니다.

menuconfig

```
Symbol: SOC_SIFIVE [=y]
  | Type  : bool
  | Prompt: SiFive SoCs
  |   Location:
  | (8) -> SoC selection
  |   Defined at arch/riscv/Kconfig.socs:3
  |   Selects: SERIAL_SIFIVE [=y] && SERIAL_SIFIVE_CONSOLE [=y] &&
CLK_SIFIVE [=y] && CLK_SIFIVE_FU540_PRCI [=y] && SIFIVE_PLIC [=y]
```

RISC-V(FU540) 리눅스 커널 포팅

SOC_SIFIVE

리눅스 커널 메인라인 소스 v5.4 부터 **SOC_SIFIVE** 설정이 추가 되었습니다. v5.4 에서는 menuconfig 에서 **SOC_SIFIVE [=y]**를 선택합니다. (디바이스 트리 적용)

arch/riscv/Kconfig.socs

```
menu "SoC selection"

config SOC_SIFIVE
       bool "SiFive SoCs"
       select SERIAL_SIFIVE
       select SERIAL_SIFIVE_CONSOLE
       select CLK_SIFIVE
       select CLK_SIFIVE_FU540_PRCI
       select SIFIVE_PLIC
       help
         This enables support for SiFive SoC platform hardware.

endmenu
```

arch/riscv/kernel/* 소스

arch/riscv/Kconfig

```
# set if we run in machine mode, cleared if we run in supervisor mode
config RISCV_M_MODE
        bool
        default !MMU

# set if we are running in S-mode and can use SBI calls
config RISCV_SBI
        bool
        depends on !RISCV_M_MODE
        default y
```

FU540 클럭 포팅

drivers/clk/sifive/*

```
$ ll drivers/clk/sifive/
drwxrwxr-x  2 risc-v risc-v 4096  4월  9 21:19 ./
drwxrwxr-x 46 risc-v risc-v 4096  4월  9 21:26 ../
-rw-rw-r--  1 risc-v risc-v 3435  4월  9 21:19 gemgxl-mgmt.c
-rw-rw-r--  1 risc-v risc-v  284  4월  9 21:19 Kconfig
-rw-rw-r--  1 risc-v risc-v   88  4월  9 21:19 Makefile
-rw-rw-r--  1 risc-v risc-v 8527  4월  9 21:19 u54-prci.c
```

drivers/clk/Kconfig

```
///중간생략..
source "drivers/clk/qcom/Kconfig"
source "drivers/clk/renesas/Kconfig"
source "drivers/clk/samsung/Kconfig"
source "drivers/clk/sifive/Kconfig"
source "drivers/clk/sprd/Kconfig"
source "drivers/clk/sunxi-ng/Kconfig"
source "drivers/clk/tegra/Kconfig"
source "drivers/clk/ti/Kconfig"
source "drivers/clk/uniphier/Kconfig"
```

```
source "drivers/clk/zynqmp/Kconfig"
```

그리고 v5.x 의 “drivers/clk/Makefile” 파일의 제일 하단부에 아래 내용을 추가합니다.

```
obj-y       += sifive/
```

drivers/clk/Makefile

```
obj-$(CONFIG_ARCH_ZX)           += zte/
obj-$(CONFIG_ARCH_ZYNQ)         += zynq/
obj-$(CONFIG_COMMON_CLK_ZYNQMP) += zynqmp/
obj-y                           += sifive/
```

그리고 v5.x 의 “drivers/clk/sifive/Makefile” 파일에 아래 내용을 신규로 추가합니다.

drivers/clk/sifive/Makefile

```
obj-$(CONFIG_CLK_U54_PRCI)    += u54-prci.o
obj-$(CONFIG_CLK_GEMGXL_MGMT) += gemgxl-mgmt.o
```

그리고 v5.x 의 “drivers/clk/sifive/Kconfig” 파일에 아래 내용을 신규로 추가합니다.

drivers/clk/sifive/Kconfig

```
config CLK_U54_PRCI
      bool "PRCI driver for U54 SoCs"
      ---help---
        Supports Power Reset Clock interface found in U540 SoCs

config CLK_GEMGXL_MGMT
      bool "TX clock switch for GEMGXL in U540 SoCs"
      ---help---
        Supports clock muxing between 10/100Mbit and 1Gbit TX clock on
U540 SoCs
```

그리고 v5.x 의 소스를 빌드하기 위한 상수들을 아래와 같이 menuconfig 에서 [=y]로 설정합니다.

menuconfig

```
Symbol: CLK_U54_PRCI [=y]
  | Type  : boolean
  | Prompt: PRCI driver for U54 SoCs
  |   Location:
  |     -> Device Drivers
  | (1)   -> Common Clock Framework
  |   Defined at drivers/clk/sifive/Kconfig:1
  |   Depends on: COMMON_CLK [=y]


Symbol: CLK_GEMGXL_MGMT [=y]
  | Type  : boolean
  | Prompt: TX clock switch for GEMGXL in U540 SoCs
  |   Location:
  |     -> Device Drivers
  | (1)   -> Common Clock Framework
  |   Defined at drivers/clk/sifive/Kconfig:6
  |   Depends on: COMMON_CLK [=y]
```

위와 같이 작업하면 v5.x 에 FU540 클럭 장치(“drivers/clk/sifive/*”)의 소스파일들이 포팅되고

빌드하면 다음과 같이 drivers/clk/sifive/* 관련 소스들이 컴파일이 됩니다.

FU540 timer 포팅

SiFive 사의 FU540 디바이스 드라이버들 중에서 timer 에 관련되는 소스는 다음과 같습니다.

FU540 timer 소스

```
drivers/clocksource/Kconfig
drivers/clocksource/Makefile
drivers/clocksource/timer-riscv.c
```

위의 소스는 v5.x 에 이미 포팅 되어 있습니다. 이 소스를 빌드하기 위해서 설정해야 하는 Kconfig 와 Makefile 을 다음과 같이 확인합니다.

v5.x 의 "drivers/clocksource/Kconfig" 파일에서 아래 내용이 있는지 확인합니다.

drivers/clocksource/Kconfig

```
config RISCV_TIMER
    bool "Timer for the RISC-V platform"
    depends on GENERIC_SCHED_CLOCK && RISCV
    default y
    select TIMER_PROBE
    select TIMER_OF
    help
      This enables the per-hart timer built into all RISC-V systems, which
      is accessed via both the SBI and the rdcycle instruction.  This is
      required for all RISC-V systems.
```

그리고 v5.x 의 "drivers/clocksource/Makefile" 파일에 아래 내용이 있는지 확인합니다.

```
obj-$(CONFIG_RISCV_TIMER)       += timer-riscv.o
```

drivers/clocksource/Makefile

```
///중간생략..
obj-$(CONFIG_CLKSRC_ST_LPC)     += clksrc_st_lpc.o
obj-$(CONFIG_X86_NUMACHIP)      += numachip.o
obj-$(CONFIG_ATCPIT100_TIMER)       += timer-atcpit100.o
obj-$(CONFIG_RISCV_TIMER)       += timer-riscv.o
obj-$(CONFIG_CSKY_MP_TIMER)     += timer-mp-csky.o
obj-$(CONFIG_GX6605S_TIMER)     += timer-gx6605s.o
//이하생략..
```

v5.x 소스에 있는 "drivers/clocksource/timer_riscv.c"를 빌드하기 위한 상수들을 아래와 같이 menuconfig 에서 [=y]로 설정되 있는지 확인 합니다.

drivers/clocksource/Kconfig

```
Symbol: RISCV_TIMER [=y]
  | Type  : bool
  | Prompt: Timer for the RISC-V platform
  |   Location:
  |     -> Device Drivers
  | (1)   -> Clock Source drivers
  |   Defined at drivers/clocksource/Kconfig:608
  |   Depends on: GENERIC_CLOCKEVENTS [=y] && GENERIC_SCHED_CLOCK [=y] &&
RISCV [=y]
  |   Selects: TIMER_PROBE [=y] && TIMER_OF [=y]
  |   Selected by [y]:
  |   - RISCV [=y]
```

위와 같은 내용들은 v5.x 에 이미 코딩되어 있으므로 FU540 timer 소스파일들은 빌드 됩니다.

FU540 irqchip 포팅

SiFive 사의 FU540 irqchip 디바이스 드라이버는 v5.x 소스 “drivers/irqchip/irq-sifive-plic.c “에 이미 포팅되어 있습니다. 이 소스를 빌드하기 위한 컴파일 상수를 “drivers/irqchip/Kconfig” 파일에서 다음과 같이 [=y]로 설정되어 있는지 확인합니다. 이렇게 되어 있지 않다면 menuconfig 를 실행하여 drivers 메뉴에서 “SiFive Platform-Level Interrupt Controller” 선택을 [*]로 해주시면 됩니다.

drivers/irqchip/Kconfig

```
Symbol: SIFIVE_PLIC [=y]
  | Type  : bool
  | Prompt: SiFive Platform-Level Interrupt Controller
  |   Location:
  |     -> Device Drivers
  |   Defined at drivers/irqchip/Kconfig:411
  |   Depends on: RISCV [=y]
```

FU540 UART 콘솔 포팅

커널이 시작될때 부팅 메세지가 시리얼 콘솔창에 출력되지 않으면 동작 상태를 파악하기 힘듭니다.

그래서 FU540 장치들 중에서 시리얼 콘솔부터 포팅해야 합니다. 필자는 freedom-u-sdk(v4.15)에 있는 시리얼 콘솔 장치 소스들을 다음과 같은 순으로 v5.0 에 포팅했습니다.

arch/riscv/Kconfig

```
config EARLY_PRINTK
  def_bool y
```

그리고 "drivers/tty/Kconfig" 파일에 있는 **Virtual terminal** 상수인 **CONFIG_VT 을 [=n]**로 설정합니다.

drivers/tty/Kconfig

```
Symbol: VT [=n]
  | Type  : bool
  | Prompt: Virtual terminal
  |   Location:
  |     -> Device Drivers
  |       -> Character devices
  | (1)     -> Enable TTY (TTY [=y])
  |   Defined at drivers/tty/Kconfig:12
  |   Depends on: TTY [=y] && !UML
  |   Selects: INPUT [=y]
```

위와 같이 적용 했다면, tty 시리얼 장치를 다음과 같이 포팅합니다.

tty 드라이버 포팅

riscv tty 드라이버 소스(drivers/tty/hvc/hvc_riscv_sbi.c)는 v5.x 에 이미 포팅되어 있습니다.

하지만 “drivers/tty/hvc/Kconfig” 파일에 HVC_RISCV_SBI 빌드 상수가 [=y]로 설정되어 있는지 다음과 같이 확인 합니다.

drivers/tty/hvc/Kconfig

```
Symbol: HVC_RISCV_SBI [=y]
  | Type  : bool
  | Prompt: RISC-V SBI console support
  |   Location:
  |     -> Device Drivers
  | (1)   -> Character devices
  |   Defined at drivers/tty/hvc/Kconfig:91
  |   Depends on: TTY [=y] && RISCV [=y]
  |   Selects: HVC_DRIVER [=y]
```

위와 같이 설정되어 있다면 “drivers/tty/hvc/hvc_riscv_sbi.c” 소스가 빌드 됩니다. **이 소스가 빌드 되면 FU540 이 부팅될때 시리얼 콘솔로 부팅 메세지가 출력 됩니다.** 참고로 arch/riscv/configs/defconfig 에는 이 설정이 [=n]으로 되어 있습니다.

참고로. v5.4 에는 sifive 의 시리얼 콘솔 관련 드라이버 소스가 있습니다. 아래의 menuconfig 에서 이것을 SERIAL_SIFIVE [=y], SERIAL_SIFIVE_CONSOLE [=y]로 설정하면 부팅 과정에서 오류가 발생 합니다. 이 설정은 SOC_SIFIVE 와 연결(의존)되어 있습니다. 오류가 발생하는 원인은 리눅스 커널 v5.2 부터 FU540 보드를 위한 디바이스 트리가 적용 되었습니다. 디바이스 트리를 적용하지 않으면 오류가 발생 합니다.

V5.4 menuconfig

```
Symbol: SERIAL_SIFIVE [=y]
  | Type  : tristate
  | Prompt: SiFive UART support
  |   Location:
  |     -> Device Drivers
  | (5)     -> Serial drivers
  |   Defined at drivers/tty/serial/Kconfig:1022
  |   Depends on: TTY [=y] && HAS_IOMEM [=y] && OF [=y]
  |   Selects: SERIAL_CORE [=y]
  |   Selected by [n]:
  |   - SOC_SIFIVE [=n]


Symbol: SERIAL_SIFIVE_CONSOLE [=y]
  | Type  : bool
  | Prompt: Console on SiFive UART
  |   Location:
  |     -> Device Drivers
  |       -> Character devices
  |         -> Serial drivers
  | (6)       -> SiFive UART support (SERIAL_SIFIVE [=n])
  |   Defined at drivers/tty/serial/Kconfig:1031
  |   Depends on: TTY [=y] && HAS_IOMEM [=y] && SERIAL_SIFIVE [=n]
  |   Selects: SERIAL_CORE_CONSOLE [=y] && SERIAL_EARLYCON [=y]
  |   Selected by [n]:
  |   - SOC_SIFIVE [=n]
  |       -> Character devices
```

FU540 PWM 포팅

커널 v5.4 에는 FU540 장치(디바이스 드라이버)들 중에서 PWM 이 포팅되어 있습니다. menucofig 에서 PWM_SIFIVE [=y]로 선택 하면 소스가 빌드 됩니다.

커널 v5.4 menucofig

```
Symbol: PWM_SIFIVE [=y]
  | Type  : tristate
```

```
  | Prompt: SiFive PWM support
  |   Location:
  |     -> Device Drivers
  | (1)   -> Pulse-Width Modulation (PWM) Support (PWM [=y])
  |   Defined at drivers/pwm/Kconfig:404
  |   Depends on: PWM [=y] && OF [=y] && COMMON_CLK [=y] && (RISCV [=y] ||
COMPILE_TEST [=n])
```

커널 v5.4 이하에서는 FU540 PWM 포팅되어있지 않으므로, v4.15(freedom-u-sdk)에 있는 FU540 PWM 소스를 v5.x 으로 아래와 같이 포팅합니다. (커널 v5.4 이상에서는 아래 내용을 건너 뛰어도 됩니다)

v4.15(freedom-u-sdk)에 있는 "drivers/pwm/pwm-sifive.c"에 있는 소스 파일을 v5.0 에 복사 합니다. (new)

그리고 v5.0 의 "drivers/pwm/Kconfig" 파일에 아래 내용을 신규로 추가합니다. (new)

drivers/pwm/Kconfig

```
config PWM_SIFIVE
      tristate "SiFive PWM support"
      depends on OF
      depends on COMMON_CLK
      help
        Generic PWM framework driver for SiFive SoCs.

        To compile this driver as a module, choose M here: the module
        will be called pwm-sifive.
```

그리고 v5.0 의 "drivers/pwm/Makefile" 파일에 아래 내용을 신규로 추가합니다. (new)

```
obj-$(CONFIG_PWM_SIFIVE)    += pwm-sifive.o
```

drivers/gpio/Makefile

```
///중간생략..
obj-$(CONFIG_PWM_PXA)           += pwm-pxa.o
obj-$(CONFIG_PWM_RCAR)          += pwm-rcar.o
obj-$(CONFIG_PWM_RENESAS_TPU)   += pwm-renesas-tpu.o
obj-$(CONFIG_PWM_ROCKCHIP)      += pwm-rockchip.o
obj-$(CONFIG_PWM_SAMSUNG)       += pwm-samsung.o
obj-$(CONFIG_PWM_SPEAR)         += pwm-spear.o
obj-$(CONFIG_PWM_STI)           += pwm-sti.o
obj-$(CONFIG_PWM_STM32)         += pwm-stm32.o
//이하생략..
obj-$(CONFIG_PWM_SIFIVE)        += pwm-sifive.o
```

v5.0 소스에 추가한 “drivers/pwm/gpio-sifive.c”를 빌드하기 위한 상수들을 아래와 같이 menuconfig 에서 [=y]로 설정합니다.

drivers/pwm/Kconfig

```
Symbol: PWM_SIFIVE [=y]
  | Type  : tristate
  | Prompt: SiFive PWM support
  |   Location:
  | (1)   -> Pulse-Width Modulation (PWM) Support (PWM [=y])
  |   Defined at drivers/pwm/Kconfig:523
  |   Depends on: PWM [=y] && OF [=y] && COMMON_CLK [=y]
  |     -> Device Drivers
```

FU540 SPI 포팅

FU540 SPI 는 v5.x 에도 “drivers/spi/spi-sifive.c” 소스 파일이 포팅되어 있습니다. 단지, 이소스를 빌드하기 위한 Kconfig 와 Makefile 만 다음과 같이 확인합니다.

먼저, “drivers/spi/Kconfig” 파일에 아래 내용이 있는지 확인합니다. 없다면 추가 합니다.

drivers/spi/Kconfig

```
config SPI_SIFIVE
        tristate "SiFive SPI controller"
        depends on HAS_IOMEM
      select SPI_BITBANG
        help
          This exposes the SPI controller IP from SiFive.
```

그리고, drivers/spi/Makefile 파일에 아래 내용이 있는지 확인합니다. 없다면 추가 합니다.

drivers/spi/Makefile

```
obj-$(CONFIG_SPI_SIFIVE)      += spi-sifive.o
```

그리고 커널 빌드 상수 설정(menuconfig)에서 **SPI_SIFIVE [=y]**로 설정합니다.

menuconfig

```
Symbol: SPI_SIFIVE [=y]
  | Type  : tristate
  | Prompt: SiFive SPI controller
  |   Location:
  |     -> Device Drivers
  | (1)   -> SPI support (SPI [=y])
  |   Defined at drivers/spi/Kconfig:654
  |   Depends on: SPI [=y] && SPI_MASTER [=y] && HAS_IOMEM [=y]
```

위와 같이 설정하면 FU540 SPI 장치 드라이버가 포팅 됩니다.

FU540 GPIO 포팅(new)

이번에는 FU540 장치(디바이스 드라이버)들 중에서 GPIO 를 포팅 하도록 하겠습니다.

v4.15(freedom-u-sdk) 버전에는 아래의 FU540 GPIO 가 포팅되어 있으나 v5.x 에는 없습니다.

그래서 필자는 다음과 같은 순으로 v4.15(freedom-u-sdk)에 있는 FU540 GPIO 소스를 v5.x 으로 포팅 했습니다.

필자가 리눅스 RISC-V 패치에 기여한 내용은 아래의 커널 개발자 메일링 리스트에서 확인할 수 있습니다.

https://patchwork.kernel.org/project/linux-riscv/list/

(Summitter: JaeJoon Jung 으로 검색)

먼저 v4.15(freedom-u-sdk)에 있는 “drivers/gpio/gpio-sifive.c”에 있는 소스 파일을 v5.x 에 복사 합니다. (new) 그리고 v5.x 의 “drivers/gpio/Kconfig” 파일에 menu "Memory mapped GPIO drivers" 안에 아래 내용을 신규로 추가합니다. (new)

drivers/gpio/Kconfig

```
menu "Memory mapped GPIO drivers"
        depends on HAS_IOMEM
..

    select GPIOLIB_IRQCHIP
    help
      Say yes here to support the GPIO device on SiFive SoCs.

..
endmenu
config GPIO_SIFIVE
```

```
    bool "SiFive GPIO support"
    depends on OF_GPIO
```

그리고 v5.x 의 “drivers/gpio/Makefile” 파일에 아래 내용을 신규로 추가합니다. (new)

obj-$(CONFIG_GPIO_SIFIVE) += gpio-sifive.o

drivers/gpio/Makefile

```
///중간생략..
obj-$(CONFIG_ARCH_SA1100)   += gpio-sa1100.o
obj-$(CONFIG_GPIO_SCH)      += gpio-sch.o
obj-$(CONFIG_GPIO_SCH311X)  += gpio-sch311x.o
obj-$(CONFIG_GPIO_SIFIVE)   += gpio-sifive.o
obj-$(CONFIG_GPIO_SODAVILLE)    += gpio-sodaville.o
obj-$(CONFIG_GPIO_SPEAR_SPICS)  += gpio-spear-spics.o
obj-$(CONFIG_GPIO_STA2X11)  += gpio-sta2x11.o
//이하생략..
```

v5.x 소스에 추가한 “drivers/gpio/gpio-sifive.c”를 빌드하기 위한 상수들을 아래와 같이 menuconfig 에서 [=y]로 설정합니다.

drivers/gpio/Kconfig

```
Symbol: GPIO_SIFIVE [=y]
  | Type  : boolean
  | Prompt: SiFive GPIO support
  |   Location:
  |     -> Device Drivers
  |       -> GPIO Support (GPIOLIB [=y])
  | (1)     -> Memory mapped GPIO drivers
  |   Defined at drivers/gpio/Kconfig:400
  |   Depends on: GPIOLIB [=y] && HAS_IOMEM [=y] && OF_GPIO [=y]
  |   Selects: GPIOLIB_IRQCHIP [=y]
```

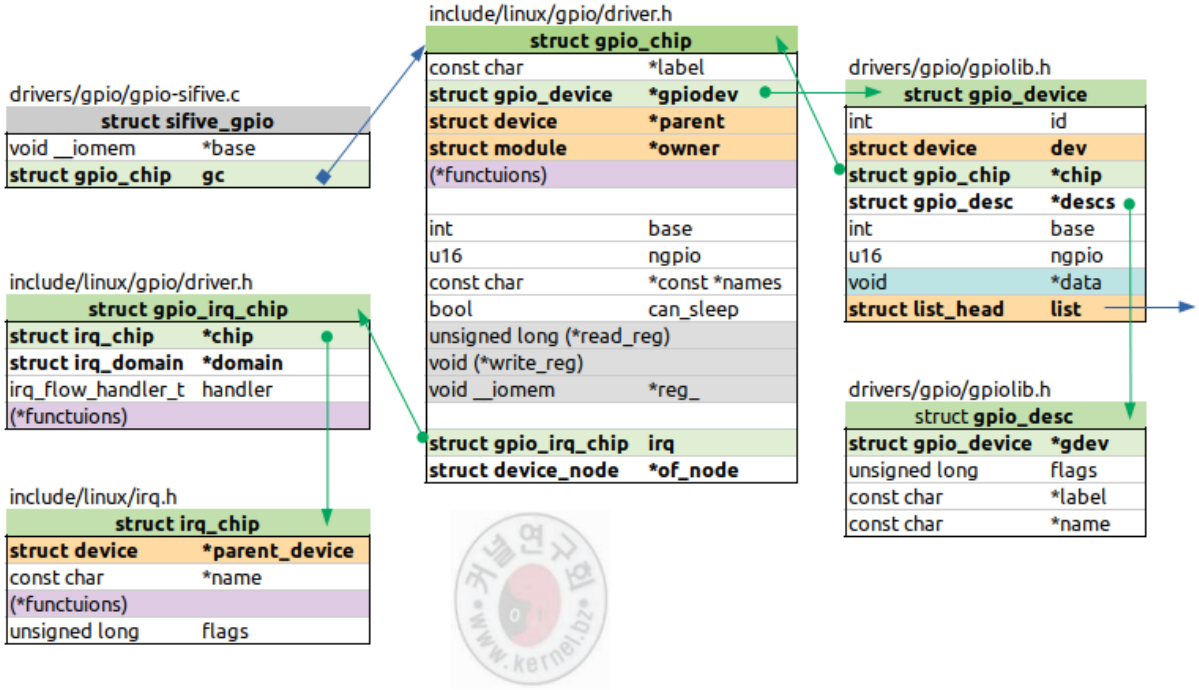
위와 같이 작업하면 v5.x 에 FU540 GPIO 소스파일들이 포팅되고 빌드할 수 있습니다. 빌드 할때 아래와 같이 drivers/gpio/*에 관련되는 소스들이 컴파일 됩니다.

필자가 이글을 작성하는 시점의 최신 리눅스 커널은 v5.4.12 입니다. 그런데 아직까지 FU540 GPIO 장치를 위한 디바이스 트리 소스가 리눅스 커널 메인라인에 적용 않되어 있습니다. 그래서 필자는 SiFive 사에서 배포한 freedom-u-sdk(v4.15) 소스에 있는 drivers/gpio/gpio-sifive.c 을 참고하여 최신 커널에 포팅하려 합니다.

구조체 정리

먼저 GPIO 장치와 관련되는 자료구조(구조체 연결)를 다음과 같이 다이어그램으로 정리 했습니다.

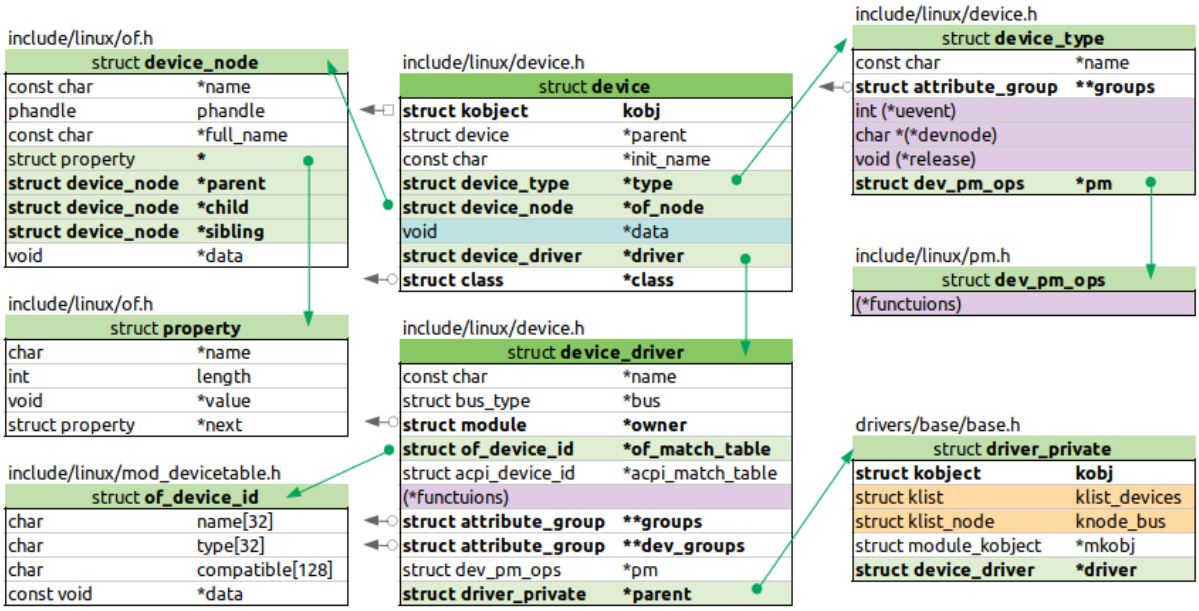
SiFive GPIO 장치 구조체 연결 블럭도



위의 구조체들의 연결이 다소 복잡해 보입니다만, 화살표 포인터 연결을 따라 가면서 이해할 수 있습니다. 커널 소스도 위의 연결 구조에서 데이터들의 이동을 이해하는 방식으로 분석해야 합니다.

연결관계를 보면 규칙성이 있습니다. 위에서 gpio_chip 구조체는 제조사마다 다른 GPIO 장치들을 이 구조체로 통합하여 관리하기 위한 것입니다. GPIO 장치 정의 및 동작구현은 모두 이 구조체에 연결하도록 합니다. 그리고 gpio_device 구조체는 gpio_chip 구조체들을 링크드 리스트로 관리할 수 있도록 연결하는 역할을 합니다. 이 구조체들 안에는 device 구조체가 멤버로 자리하고 있습니다. 리눅스 커널안의 모든 장치들은 device 구조체로 연결되어 커널 코어에서 관리 됩니다. device 구조체가 커널 코어로 연결되는 통로 역할을 합니다. 아래는 device 구조체에서 연결되는 자료구조를 블럭도로 정리한 것입니다.

커널 장치 구조체 연결 블럭도

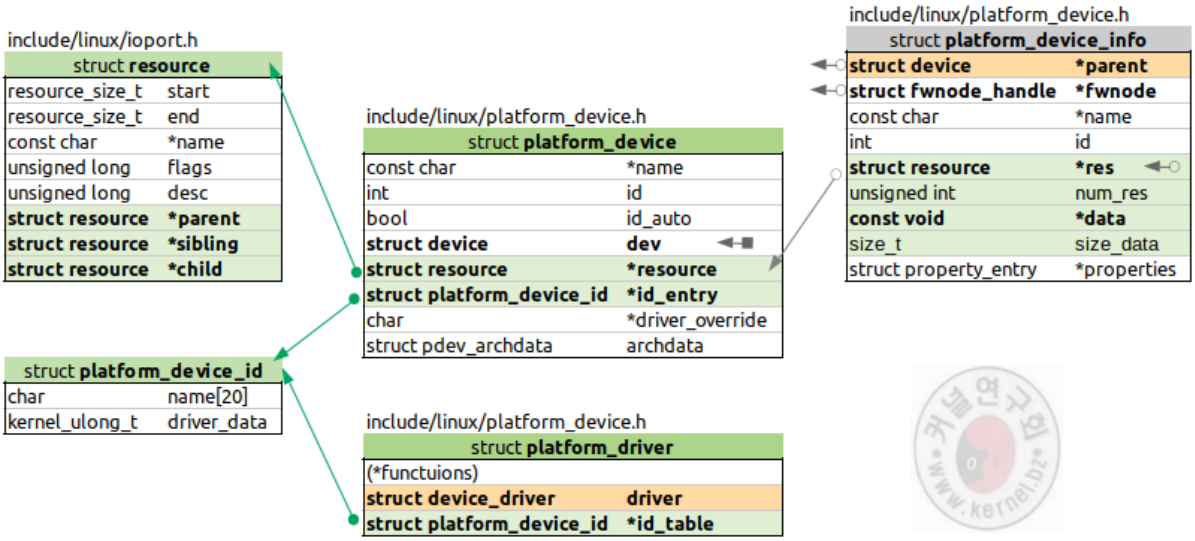


리눅스 커널은 device 구조체를 통해서 장치를 관리합니다. device 구조체안에는 커널에서 장치를 관리하기 위한 kobject, device_type, device_node, data, device_driver, class 등의 멤버 구조체가 있습니다. kobject 와 class 구조체는 장치를 커널 시스템파일(/sys/class/)로 관리할 수 있도록 연결해 줍니다. device_node 구조체는 장치들을 디바이스

트리(/sys/firmware/devicetree/)에 연결하는 구조체입니다. device_driver 구조체는 장치가 실제적으로 동작하는 기능(함수)을 정의합니다. 이러한 구조체들은 device 구조체의 멤버로 모두 통합되어 관리 됩니다.

임베디드 보드 안의 장치들은 SoC 형태로 대부분 내장되어 있기 때문에, 리눅스 커널에서는 platform 디바이스 드라이버로 등록하여 관리합니다. platform 디바이스 드라이버 등록을 위한 구조체들은 다음과 같습니다.

platform 디바이스 드라이버 등록 구조체



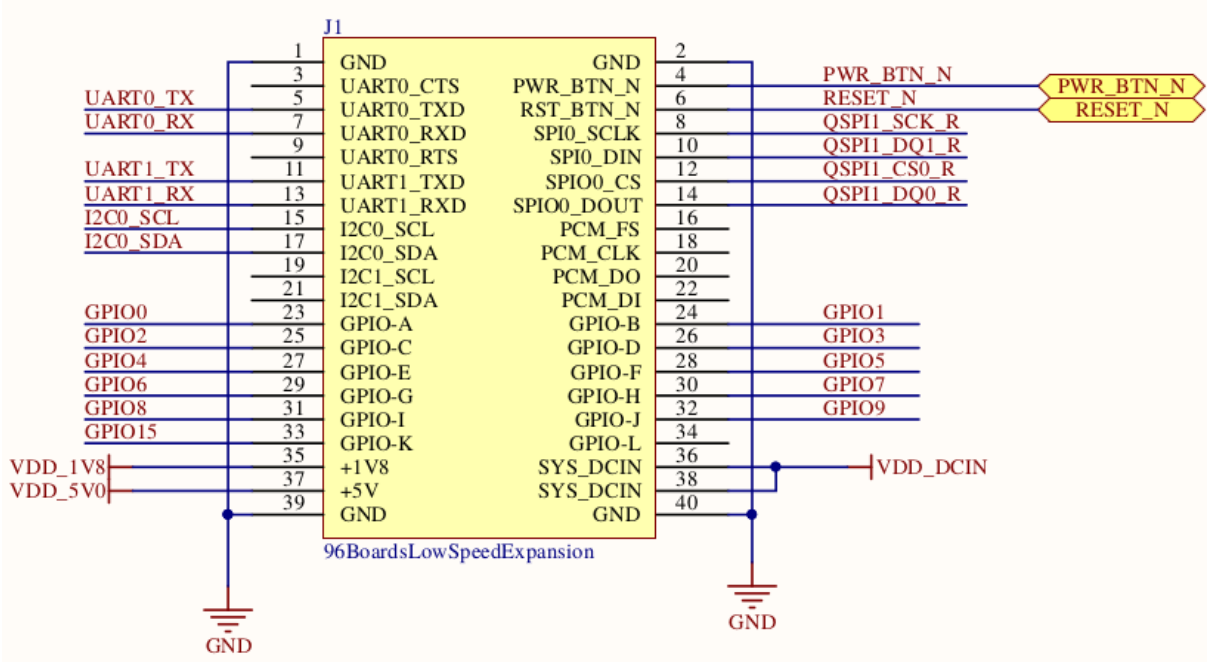
platform 디바이스는 platform_device 구조체로 정의되어 커널의 device 구조체에 연결됩니다. platform 디바이스 동작은 platform_driver 구조체로 정의되어 커널의 device_driver 구조체에 연결됩니다. platform 디바이스는 임베디드 보드(SoC)내의 장치에 접근하기 위한 주소 정보들을 resource 구조체를 사용하여 관리합니다. 그래서 platform_device 구조체안에는 resource 구조체가 멤버로 연결되어 있습니다.

리눅스 커널은 위와 같은 구조체를 사용하여 디바이스 드라이버 작성(정의, 동작)을 점점 표준화하고 있습니다. 특히, 커널에 장치가 추가될때마다 소스 변경을 최소화 하고 유지보수를 효율화하기 위해서 디바이스 트리 파일에 장치들을 표준화된 문법으로 기술하고 이것을 파싱하여 관리합니다.

GPIO 테스트

GPIO 를 테스트하기 전에 다음과 같이 SiFive 사에서 배포한 출력 헤더핀(J1) 회로도를 참조합니다. GPIO 는 16 개까지 설정되어 있으나 실제로 헤더핀에 연결하여 사용할 수 있는 GPIO 는 0 번부터 9 번까지 10 개와 마지막 15 번까지 합해서 11 개입니다. 나머지 5 개는 보드내부에서 예약해서 사용하고 있습니다.

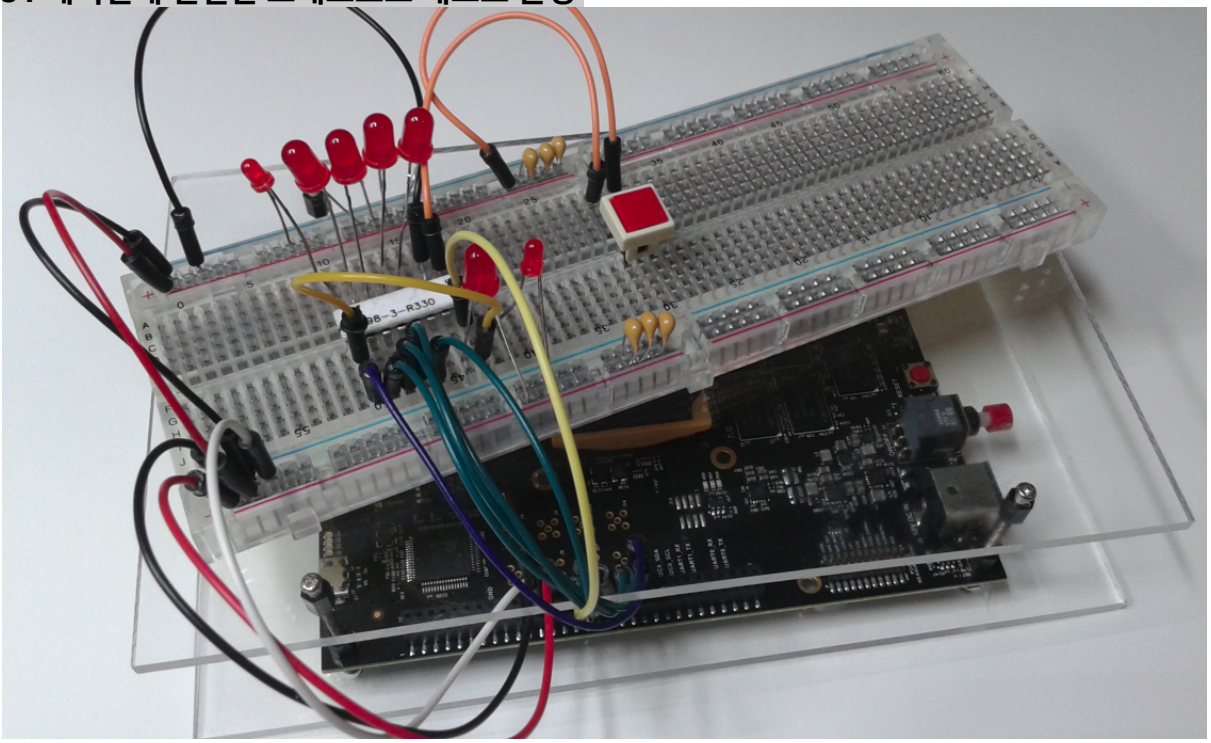
GPIO 출력 헤더핀(J1) 회로도



출처: SiFive FU540-C000 회로도

필자는 위의 회로도의 J1 헤더핀에 점퍼선을 브레드보드와 다음과 같이 열결하여 테스트 환경을 만들었습니다.

J1 헤더핀에 연결한 브레드보드 테스트 환경



FU540 은 저전력으로 동작하는 관계로 GPIO 에서 출력되는 전압은 1.8V 입니다. 여기에 LED 을 연결하면 입출력 되는 데이터를 충분히 확인할 수 있습니다. 그리고 스위치를 연결한 GPIO 핀은 인터럽트 테스트용으로 사용합니다. 필자는 GPIO 7 번에 대해서 인터럽트 테스트 했습니다. 테스트 내용은 아래부터 설명 됩니다.

참고 문서

참고 도서

링크: https://www.kernel.bz/books

(알라딘, 교보문고, 예스 24, 인터파크, 시중 인터넷 서점에서 구매 가능)

RISC-V 컴파일러 참고 문서

https://gnu-mcu-eclipse.github.io/toolchain/riscv/

Running 64-bit RISC-V Linux on SiFive HiFive Unleashed

https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-hifive-u.html

Running 64- and 32-bit RISC-V Linux on QEMU

https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-qemu.html

The Linux Kernel documentation

https://www.kernel.org/doc/html/latest/index.html

Linux RISC-V patchwork

https://patchwork.kernel.org/project/linux-riscv/list/

© SiFive, Inc.

https://www.sifive.com/

RISC-V github

https://github.com/riscv

커널연구회 github (소스공유)

https://github.com/kernel-bz

기타 궁금한 점은 저자(rgbi3307@nate.com)에게 메일 주시기 바랍니다.

감사합니다.