Проектиране на вградени системи
Лабораторно упражнение №5

Операционна система Linux за вградени системи. U-boot фърмуер от второ ниво за инициализация и старт. Модули за Линукс. Дървесни двоични описания (Device Tree Binaries).

==========

1. За стартиране на Linux за вградени системи на микроконтролер STM32F769I (STM32F769I-DISCO демо платка) е необходим персонален компютър (PC) с операционна система Linux. Настоящите инструкции са за 32-битов Linux, дистрибуция Ubuntu 14.04. Първоначално трябва да се компилира и зареди фърмуер от второ ниво за първоначална инициализация и старт (на английски **second stage bootloader**). Широко използван в практиката е U-boot и затова е избран именно той.

- Стартирайте терминал с комбинацията CTRL + ALT + t
- Изпълнете следните команди:

//Необходим за menuconfig
```
sudo apt-get install libncurses-dev
```

//Необходими за build-ване на Linux и U-boot
```
sudo apt-get install autotools-dev autoconf bison flex build-essential
```

//Необходим за сваляне на сорс кода на Linux и U-boot
```
sudo apt-get install git
```

//Необходим за връзка със сериен RS232 порт (аналогичен на Tera //Term и //Realterm под Windows)

```
sudo apt-get install cutecom
```

//Необходим за flash-ване на STM32F769I-DISCO демо платка
//Забележка: версията на openocd в Ubuntu 14 е стара и не поддържа //въпросната демо платка. Затова потребителя трябва да свали най-//новия сорс (>= 0.10.0) на OpenOCD от сайта на този проект и да го //build-не. Командата по-долу е валидна за по-ново Ubuntu.

```
sudo apt-get install openocd
```

От сайта на ARM се сваля най-новият крос-GCC toolchain за ARM-Cortex микропроцесори

https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
Експортва се пътя до bin директорията на този toolchain, заменете ”/path/to” с пътя до директорията, където се разархивирали toolchain-а

```
gedit ~/.bashrc
export PATH=$PATH:/path/to/gcc-arm-none-eabi/bin
source ~/.bashrc
```

Сваля се сорс кода на U-boot от сайта на този проект посредством софтуерът за контрол на версията Git

```
git clone git://git.denx.de/u-boot.git
```

Влиза се в току-що клонираната директория

```
cd u-boot
```

Зарежда се default конфигурация за build на U-boot, което е необходимо, защото U-boot е универсален фърмуер и се използва от много микроконтролери.

```
make ARCH=arm CROSS_COMPILE=arm-none-eabi- stm32f746-disco_defconfig
```

Редактирайте следният сорс файл:

```
gedit board/st/stm32f746-disco/stm32f746-disco.c
```

Във функцията **void** dram_init(**void**) закоментирайте #ifndef CONFIG_SUPPORT_SPL и съответната затваряща директива #endif. След това редактирайте

```
gedit include/configs/stm32f746-disco.h
```

Намерете секцията от макроси

```
#define CONFIG_BOOTCOMMAND                                      \
        "run bootcmd_romfs"

#define CONFIG_EXTRA_ENV_SETTINGS \
        "bootargs_romfs=uclinux.physaddr=0x08180000 root=/dev/mtdblock0\0" \
        "bootcmd_romfs=setenv bootargs ${bootargs} ${bootargs_romfs};" \
        "bootm 0x08044000 – 0x08042000\0"
```

и я променете на

!!!ВНИМАНИЕ!!!: използвай boot_str.txt в директорията на лабораторното за копиране на стринга:

```
#define CONFIG_BOOTCOMMAND \
        "run bootcmd_romfs"

#define CONFIG_EXTRA_ENV_SETTINGS \
        "bootcmd_romfs=setenv bootargs ${bootargs} mmc dev 0 && " \
        "fatload mmc 0 0xc0700000 /stm32f769-disco.dtb && " \
        "fatload mmc 0 0xc0008000 /zImage && " \
        "icache off;" \
        "bootz 0x0c0008000 - 0xc0700000\0"
```

Ако дистрибуцията ви е Ubuntu 22.04.2, отворете файла:

```
gedit ./scripts/dtc/dtc-parser.tab.c
```

и на ред 1207 променете:

YYLTYPE yylloc -> **extern** YYLTYPE yylloc

След това изпълнете командата, която build-ва фърмуера U-boot. Забележете, че за двоично описание се избира точно за stm32f769-disco, което съответства на нашата демо платка

```
make ARCH=arm CROSS_COMPILE=arm-none-eabi- DEVICE_TREE=stm32f769-disco
```

Отворете два терминала с CTRL + ALT + t. В **първия терминал** пуснете GDB демон (сървърно приложение), което ще се свърже с STM32F769 микроконтролера посредством ST-Link дебъгера. Заменете “/path/to/openocd” с пътя до инсталационната директория на OpenOCD

```
sudo /path/to/openocd/bin/openocd -f board/stm32f7discovery.cfg
```

Във **втория терминал** пуснете универсално клиентско приложение за свързване със сървъри, например telnet, и се свържете с виртуалното IP на вашия компютър – localhost, на порт 4444

```
telnet localhost 4444
reset init
flash probe 0
flash write_image erase /path/to/u-boot/u-boot-dtb.bin 0x08000000
flash write_image erase /path/to/u-boot/u-boot-dtb.bin 0x08008000
reset run
```

ВНИМАНИЕ: Ако в микроконтролера STM32F769 е зареден валиден фърмуер, например някаква програма Blinky, то микропроцесорът ARM Cortex-M7 ще започне изпълнение от адрес 0x8000.0000. U-boot обаче е компилиран за работа от адрес 0x08008000. Затова, първият път, когато започвате работа с U-boot

първо го flash-ваме на адрес 0x8000.0000:

```
flash write_image erase /path/to/u-boot/u-boot-dtb.bin 0x08008000
```

което ще доведе до появата на невалиден код на този адрес (защото е компилиран за адрес 0x08008000) и ще принуди first-stage bootloader-а на STM32F769I да опита да зареди програма от вторият си адрес 0x08008000, където вече ще има коректен обектов код.

Горните команди за flash-ване могат да се изпълнят и без telnet, като за целта се въвежда:

```
openocd -f board/stm32f7discovery.cfg -c"init" -c"reset halt" -
c"flash probe 0" -c"flash write_image erase /path/to/u-boot-
dtb.bin 0x08008000" -c"reset run" -c"exit"
```

Отворете трети терминал и стартирайте терминал за работа с RS232 сериен порт (който е виртуален и се емулира от ST-Link в нашия случай)

```
sudo cutecom
```

Настройте връзката 115200-8-N-1, no parity, no flow control. За номер на порт изберете /dev/ttyACM0 (или понякога излиза като /dev/ttyUSB0), което е аналогично на Windows-кият COM1, COM2 и т.н.

Натиснете бутона RESET на демо борда. Трябва да видите съобщения от вида:

```
U-Boot 2019.07-rc3 (May 29 2019 - 10:25:33 +0300)

Model: STMicroelectronics STM32F769-DISCO board
DRAM:  16 MiB
set_rate not implemented for clock index 4
set_rate not implemented for clock index 4
Flash: 1 MiB
MMC:   sdio2@40011c00: 0
In:    serial
Out:   serial
Err:   serial
usr button is at LOW LEVEL
Net:
Warning: ethernet@40028000 (eth0) using random MAC address -
8e:e3:1c:40:e9:81
eth0: ethernet@40028000
Hit SPACE in 3 seconds to stop autoboot.
Card did not respond to voltage select!
U-Boot >
```

U-Boot се нуждае от валиден обектов код на ядро на операционната система

Linux и двоично описание за конкретната платка, които могат да се зареждат от SD-карта, външен flash чип, Ethernet интерфейс и др. В настоящото упражнение се използва SD карта.

2. Компилиране на Linux ядро става посредством следните команди:

```
git clone https://github.com/fdu/STM32F769I-disco_Buildroot.git
cd STM32F769I-disco_Buildroot
make bootstrap
cd buildroot
make clean
make menuconfig
```

Избира се в менюто Main menu -> Toolchain -> Binutils Version -> *select* binutils 2.28.1 -> Save -> OK -> Exit

Ако вашата фирма/институция е затворила портовете за комуникация, билдването ще се провали на стъпката за сваляне на Bootloader. За да се заобиколи този проблем, може да се използва https:// вместо git:// връзка (която работи на порт 9418). Избира се в менюто Main menu -> Bootloader -> Под менюто U-boot се избира URL of custom repository -> Въвежда се https://git.denx.de/u-boot.git -> OK -> Exit → Exit → Save.

```
cd ..
make build
gedit ./buildroot/output/target/init
```

Добавят се следните редове в инициализиращия скрипт

#!/bin/sh
devtmpfs does not get automounted for initramfs
/bin/mount -t devtmpfs devtmpfs /dev
touch /dev/tty2
touch /dev/tty3
touch /dev/tty4
exec 0</dev/console
exec 1>/dev/console
exec 2>/dev/console
exec /sbin/init "$@"

в противен случай терминалът на Linux ще бъде наводнен със съобщения за липсващ дисплей (в настоящото лабораторно се използва само терминален вход/изход, без графична среда).

```
make build
```

Полученият обектов код се намира в директорията:

STM32F769I-disco_Buildroot/buildroot/output/images

а имената на файловете са:
- zImage (компресиран обектов код на ядрото)
- stm32f769-disco.dtb (двоично описание на микроконтролера/платката)
Тези два файла трябва да се копират на SD карта, форматирата с FATFS и трябва да се намират в root директорията. Поставя се картата в куплунга и се рестартира демо платката. Ако всичко мине добре, потребителят ще види команден ред Bash в терминала Cutecom

```
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.15.10 (lbogdanov@dexter) (gcc version 7.3.0
(Buildroot 2018.02-00002-g0e90cde)) #3 PREEMPT Tue Mar 19 22:00:23 EET 2019
[    0.000000] CPU: ARMv7-M [411fc270] revision 0 (ARMv7M), cr=00000000
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] OF: fdt: Machine model: STMicroelectronics STM32F769-DISCO board
[    0.000000] debug: ignoring loglevel setting.
[    0.000000] On node 0 totalpages: 4096
[    0.000000]   Normal zone: 32 pages used for memmap
[    0.000000]   Normal zone: 0 pages reserved
[    0.000000]   Normal zone: 4096 pages, LIFO batch:0
[    0.000000] pcpu-alloc: s0 r0 d32768 u32768 alloc=1*32768
[    0.000000] pcpu-alloc: [0] 0
[    0.000000] Built 1 zonelists, mobility grouping off.  Total pages: 4064
[    0.000000] Kernel command line: console=ttyS0,115200 earlyprintk
consoleblank=0 ignore_loglevel
[    0.000000] Dentry cache hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.000000] Memory: 13504K/16384K available (1511K kernel code, 148K rwdata,
484K rodata, 284K init, 125K bss, 2880K reserved, 0K cma-reserved)
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0x00000000 - 0x00001000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0x00000000 - 0xffffffff   (4095 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xc1000000   (  16 MB)
[    0.000000]       .text : 0x(ptrval) - 0x(ptrval)   (1512 kB)
[    0.000000]       .init : 0x(ptrval) - 0x(ptrval)   ( 284 kB)
[    0.000000]       .data : 0x(ptrval) - 0x(ptrval)   ( 149 kB)
[    0.000000]        .bss : 0x(ptrval) - 0x(ptrval)   ( 126 kB)
[    0.000000] SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] \0x09Tasks RCU enabled.
[    0.000000] NR_IRQS: 16, nr_irqs: 16, preallocated irqs: 16
[    0.000000] interrupt-controller@40013c00: bank0, External IRQs
available:0x1ffffff
[    0.000000] clocksource: arm_system_timer: mask: 0xffffff max_cycles:
0xffffff, max_idle_ns: 298634427 ns
[    0.000000] ARM System timer initialized as clocksource
[    0.000000] /soc/timer@40000c00: STM32 clockevent driver initialized (32
bits)
[    0.000000] sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps
every 2147483647000000ns
[    0.050000] Calibrating delay loop... 398.13 BogoMIPS (lpj=1990656)
```

```
[    0.050000] pid_max: default: 4096 minimum: 301
[    0.050000] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.050000] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.050000] Hierarchical SRCU implementation.
[    0.050000] devtmpfs: initialized
[    0.070000] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff,
max_idle_ns: 19112604462750000 ns
[    0.070000] pinctrl core: initialized pinctrl subsystem
[    0.070000] random: get_random_u32 called from bucket_table_alloc+0xc3/0xec
with crng_init=0
[    0.070000] NET: Registered protocol family 16
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOA bank added
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOB bank added
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOC bank added
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOD bank added
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOE bank added
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOF bank added
[    0.080000] stm32f746-pinctrl soc:pin-controller: GPIOG bank added
[    0.090000] stm32f746-pinctrl soc:pin-controller: GPIOH bank added
[    0.090000] stm32f746-pinctrl soc:pin-controller: GPIOI bank added
[    0.090000] stm32f746-pinctrl soc:pin-controller: GPIOJ bank added
[    0.090000] stm32f746-pinctrl soc:pin-controller: GPIOK bank added
[    0.090000] stm32f746-pinctrl soc:pin-controller: Pinctrl STM32 initialized
[    0.100000] random: fast init done
[    0.100000] clocksource: Switched to clocksource arm_system_timer
[    0.110000] NET: Registered protocol family 2
[    0.110000] TCP established hash table entries: 1024 (order: 0, 4096 bytes)
[    0.110000] TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
[    0.110000] TCP: Hash tables configured (established 1024 bind 1024)
[    0.110000] UDP hash table entries: 256 (order: 0, 4096 bytes)
[    0.110000] UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
[    0.180000] workingset: timestamp_bits=30 max_order=12 bucket_order=0
[    0.190000] io scheduler noop registered (default)
[    0.190000] io scheduler mq-deadline registered
[    0.190000] io scheduler kyber registered
[    0.210000] STM32 USART driver initialized
[    0.210000] 40011000.serial: ttyS0 at MMIO 0x40011000 (irq = 32, base_baud =
6250000) is a stm32-usart
[    0.630000] console [ttyS0] enabled
[    0.640000] stm32-usart 40011000.serial: rx dma alloc failed
[    0.650000] stm32-usart 40011000.serial: interrupt mode used for rx (no dma)
[    0.660000] stm32-usart 40011000.serial: tx dma alloc failed
[    0.660000] stm32-usart 40011000.serial: interrupt mode used for tx (no dma)
[    0.670000] i2c /dev entries driver
[    0.670000] NET: Registered protocol family 17
[    0.680000] hctosys: unable to open rtc device (rtc0)
[    0.690000] Freeing unused kernel memory: 284K
[    0.690000] This architecture does not have kernel memory protection.

can't run '/etc/init.d/rcS': No such file or directory


BusyBox v1.27.2 (2019-03-19 21:35:12 EET) hush - the humble shell
Enter 'help' for a list of built-in commands.

/ #
```

3. Модули (драйвери) за Линукс се компилират едва след като имаме свален сорс код на Линукс ядрото, неговите хедърни файлове и кроскомпилатор. Сорс кодът на модулите може да бъде в дървото на проекта на ядрото (kernel modules) или извън него (out-of-tree modules). В настоящото лабораторно ще се

компилират модули извън дървото на Линукс. Модулите са програми, които работят във виртуалното адресно поле на Линукс ядрото, още наречено **kernel space**.

Първата стъпка е да се разреши компилацията на модули чрез menuconfig. От top-level директорията на buildroot изпълнете командата:

```
make -C ./buildroot linux-menuconfig
```

Изберете от главното меню “Enable loadable module support” със SPACE клавиша, а след това натиснете Enter и се уверете, че са избрани следните опции:

```
[*] Forced module loading
[*] Module unloading
[*] Forced module unloading
[*] Module version support
[*] Source checksum for all modules
```

Излезте от menuconfig и запазете новата конфигурация. Разрешаването на тези опции не е достатъчно за добавяне на модули. Пакетът от програми (или още - команди), с който сме пуснали нашия Линукс е минимален. Този пакет се нарича Busybox и може да се добавят и махат програми по желание на програмиста. С default конфигурацията липсват командите за пускане, спиране и показване на информация за модули от ядрото. Това са известните modprobe, modinfo, depmod, insmod, lsmod, rmmod.

ВНИМАНИЕ: в по-нови версии на Линукс ядрото, всички тези команди са заместени от само една такава – kmod.

За да се добавят горе-изброените команди трябва да се преконфигурира Busybox с командата:

```
make -C ./buildroot/ busybox-menuconfig
```

откъдето трябва да се избере менюто “Linux module utilities” и със SPACE да се изберат всички необходими команди:

```
[*] Simplified modutils
[*] depmod
[*] insmod
[*] lsmod
[*] modinfo
[*] modprobe
[*] rmmod
```

Излезте от menuconfig и запазете конфигурацията. Компилирайте новото ядро с новите команди чрез командата:

```
make build
```

Уверете се, че командите са добавени, като за целта пуснете новото ядро и от неговия терминал изпълните някоя от командите, например:

```
insmod
```

което трябва да върне текста:

```
BusyBox v1.27.2 (2021-04-06 13:34:05 PDT) multi-call binary.
Usage: insmod FILE [SYMBOL=VALUE]...
Load kernel module
```

Ако терминалът върне:

```
sh: can't execute 'insmod': No such file or directory
```

значи, че добавянето не е било успешно и командите все още липсват.

4. Копирайте директория **05_4** на десктопа и отворете Makefile-a, който е вътре. Ще се използва build системата на Линукс, за да компилираме нашия модул. Три неща трябва да бъдат указани:
- име на модула (например firstmod);
- път и префикс на кроскомпилатора чрез променливата CCPWD, за нашия случай използваме кроскомпилатора на Buildroot, който се намира в:

```
STM32F769I-disco_Buildroot/buildroot/output/host/bin/arm-buildroot-uclinux-
uclibcgnueabi-
```

- път до сорса на Линукс чрез променливата KERNDIR, за нашия случай използваме Линукс 4.15.10 на Buildroot, който се намира в:

```
STM32F769I-disco_Buildroot/buildroot/output/build/linux-custom
```

Забележете името на променливата в Makefile-a – obj-m. Така build системата на Линукс познава, че искаме да компилираме out-of-tree модул, който ще се зарежда след като Линукс е стартиран. Ако модулът ни е част от сорса на ядрото, и ако се стартира заедно с ядрото, трябва да се запише obj-y. Заедно с обектовия файл ще се появят и няколко служебни файла, значението на които е показано по-долу:

Файл	**Значение**
${MODNAME}.ko	Краен обектов файл на Линукс модула (${MODNAME}.o линкнат с ${MODNAME}.mod.o)
${MODNAME}.o	Обектов файл на Линукс модула

${MODNAME}.mod.c	Сорс файл с информация за модула (например версия, имена на функции от модула, разположение в паметта и др.)
${MODNAME}.mod.o	Обектов файл, получен след компилиране и асемблиране на ${MODNAME}.mod.c
Module.symvers	Списък със символи, декларирани като extern в сорса на модула.
modules.order	В случай, че се компилира повече от един модул, в този файл се изброява последователността, в която трябва да бъдат компилирани модулите.

Попълнете Makefile-а и отворете терминал с CTRL + ALT + t. Преместете се в директорията на модула и го компилирайте:

```
cd /path/to/firstmod
make
```

Примерен текст по време на build-а ще изглежда по следния начин:

```
mkdir -p ./debug
make ARCH=arm LOCALVERSION= CROSS_COMPILE=/home/user/STM32F769I-
disco_Buildroot/buildroot/output/host/bin/arm-buildroot-uclinux-uclibcgnueabi- -
C /home/user/STM32F769I-disco_Buildroot/buildroot/output/build/linux-custom
SUBDIRS=/home/user/Desktop/firstmod modules
make[1]: Entering directory
`/home/user/STM32F769I-disco_Buildroot/buildroot/output/build/linux-custom'
  CC [M]  /home/user/Desktop/firstmod/firstmod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/user/Desktop/firstmod/firstmod.mod.o
  LD [M]  /home/user/Desktop/firstmod/firstmod.ko
make[1]: Leaving directory
`/home/user/STM32F769I-disco_Buildroot/buildroot/output/build/linux-custom'
mv firstmod.mod.c firstmod.ko firstmod.mod.o firstmod.o Module.symvers
modules.order ./debug
```

5. Компилирайте **user space** програма за вашата система. За целта използвайте директория **05_5**. Единственото нещо, което трябва да се съобрази при писането на подобни програми е да се използва крос-компилатор, който да знае за съществуващата операционна система. Най-лесно е да се използва компилаторът, с който сме компилирали ядрото.

Копирайте директорията **05_5** на десктопа и редактирайте Makefile-а вътре. След това отворете терминал с CTRL + ALT + t, преместете с в тази директория и компилирайте приложението:

```
cd /path/to/firstapp
make
```

6. Копирайте файла firstmod.ko от точка 4 в

```
STM32F769I-disco_Buildroot/buildroot/output/target/lib/modules/4.15.10
```

Копирайте файла firstapp.bin от точка 5 в домашната директория на главния Линукс потребител:

```
STM32F769I-disco_Buildroot/buildroot/output/target/root
```

В top-level директорията на Buildroot изпълнете командата:

```
make build
```

Пуснете новополученото ядро и в терминалът му изпълнете командите:

```
depmod
modprobe firstmod
```

за да пуснете Линукс модула. Ако всичко е минало успешно, в терминала трябва да ви се изпише съобщението от printk на init( ) функцията. Съобщенията от модулите могат да бъдат видяни и с командата:

```
dmesg
```

която показва съобщения от Линукс ядрото. Тези съобщения обаче се пазят в кръгов буфер и ако принтирате много информация там, по-старите съобщения ще изчезнат.

Изпълнете командата:

```
exec /root/firstapp.bin
```

за да пуснете приложната програма. Ако всичко е минало успешно, в терминала трябва да ви се изпише съобщението от printf на main( ) функцията.

7. Копирайте директориите от **05_7** на десктопа. Компилирайте Линукс модула за инициализация на GPIOA извод 12 (свързан към светодиод LD3 на платката) [1] [2] [3] [4]. Забележете, че този модул е съставен от два файла firstmod_driver и firstmod_device. Копирайте и двата файла в

```
STM32F769I-disco_Buildroot/buildroot/output/target/lib/modules/4.15.10
```

след което компилирайте и приложната програма, която използва този модул. Нея я копирайте в:

```
STM32F769I-disco_Buildroot/buildroot/output/target/root
```

Компилирайте и заредете Линукс ядрото. Току-що добавените ви файлове ще бъдат сляти с обектовия код на ядрото. При стартиране на системата те ще бъдат копирани съответно в /root и /lib/modules/4.15.10 във RAMFS файловата система (това е бездискова файлова система). В терминала на системата изпълнете:

```
depmod
modprobe firstmod_device
modprobe firstmod_driver
```

Проверете дали се е създал системния файл firstmod чрез командата:

```
ls /dev
```

Ако драйверът се е инициализирал успешно, текстът от последната команда ще изглежда така:

```
ls /dev
gpiochip0           gpiochip10          ptmx
gpiochip1           mem                 ttyS0
gpiochip2           null                cpu_dma_latency
gpiochip3           zero                network_latency
gpiochip4           full                network_throughput
gpiochip5           random              memory_bandwidth
gpiochip6           urandom             tty2
gpiochip7           kmsg                tty3
gpiochip8           tty                 tty4
gpiochip9           console             firstmod
```

От командния ред (user space) можете да използвате стандартни програми, за да достъпвате системни файлове. В случая драйверът така е написан, че ако във firstmod се запише 1, светодиодът ще светне. Ако във firstmod се запише 0, светодиодът ще угасне [5]. Опитайте следните команди:

```
echo 1 > /dev/firstmod
echo 0 > /dev/firstmod
```

Ако светодиодът се включва и изключва, може да се премине към пускане на потребителската програма. Тя мига светодиода 10 пъти, след което се терминира. Изпълнете програмата така:

```
exec /root/firsmod_app.bin
```

8. Използвайте DTB описание за RCC и GPIOA модулите [6] [7] [8] [9]. Такова описание е дадено от производителя на микроконтролера, но в настоящото лабораторно то ще бъде допълнено за нашия модул. Копирайте директория **05_8** на десктопа. Отворете сорс файлът с описание на регистрите на модулите от микроконтролера:

```
gedit ~/Desktop/05_8/firstmod_dtb/firstmod.dts
```

и добавете примерно описание:

```
            led: firstmod@40020000 {
                  compatible = "firstmod";
                  reg = <0x40020000 0x400
                         0x40023800 0x400>;
                  interrupts = <22>;
                  status = "enabled";
            };
```

Забележете, че хардуерът ни използва два модула (RCC & GPIOA), затова имаме два обхвата от регистри в полето “reg”.

Укажете в Makefile-а пътя до компилатора за дървесни двоични описания (dtc – device tree compiler):

```
gedit ~/Desktop/05_8/firstmod_dtb/Makefile
```

като въведете пътя:

```
STM32F769I-disco_Buildroot/buildroot/output/build/linux-custom/scripts/dtc
```

Обърнете внимание на този Makefile – към днешна дата DTC компилатора няма предпроцесор, затова се използва съществуващият такъв на C++ компилатор плюс символът за пренасочване на stdout от една програма към друга | (още известен като pipe).

Въведете пътя до кроскомпилатора и неговия префикс (както бе указано в точка 4) в Makefile-а на новия платформен драйвер, използващ DTB:

```
gedit ~/Desktop/05_8/firstmod_drv/Makefile
```

Компилирайте и драйвера, и новото двоично описание. Преименувайте описанието на stm32f769-disco.dtb и го копирайте заедно с zImage на SD картата. Поставете я в макета и го рестартирайте. Изпълнете отново командите както в предходната точка. Забележете – сега няма зареждане на описанието на устройство → тази информация ще бъде взета от DTB чрез една специална API функция – of_address_to_resource( ) и една структура, която има низ, който ще бъде съпоставен със съществуващите низове в DTB и при съвпадение ще бъде предадена информация от DTB към драйвера чрез of_address_to_resource( ).

```
static struct of_device_id firstmod_of_match[] = {
  { .compatible = "firstmod", },
  {}
};
```

```
depmod
modprobe firstmod_driver
echo 1 > /dev/firstmod
echo 0 > /dev/firstmod
exec /root/firstmod_app.bin
```

Ако всичко е минало успешно, при зареждането на драйвера трябва да се изпишат същите физически адреси като от предишната точка:

```
/ # modprobe firstmod_driver
[   19.330000] firstmod_driver: loading out-of-tree module taints kernel.
[   19.330000] Platform driver init
[   19.330000] Physical addresses range - RCC
[   19.330000] 40023800 - 40023bff : 1024
[   19.330000] gpioa_base_virtual = e01d825a
[   19.330000] Physical addresses range - GPIOA
[   19.330000] 40020000 - 400203ff : 1024
[   19.330000] gpioa_base_virtual = cbfe8006
[   19.330000] Device file registered successfully in /dev!
```

Допълнителна информация:
→компилацията на чисто хардуерно описание (без C/C++ макроси) става посредством командата

```
dtc -I dts -O dtb -o [изходен_файл.dtb] [входен_файл.dts/.dtsi]
```

→възможна е и обратната команда (от двоично описание да се изгенерира текстови файл)

```
dtc -I dtb -O dts -f [входен_файл.dtb] -o [изходен_файл.dts]
```

→ често използвани съкращения:
dtb – device tree binary
dtc – device tree compiler
dts – device tree source
dtsi – device tree include
fdt – flattened device tree

* * *

[1] https://www.kernel.org/doc/htmldocs/kernel-api/
[2] https://github.com/jeyaramvrp/kernel-module-programming/tree/master/sample-platform-driver
[3] https://github.com/jeyaramvrp/kernel-module-programming/tree/master/sample-platform-driver
[4] https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch09.html
[5]http://www.opensourceforu.com/2011/04/character-device-files-creation-operations/
[6] Thomas Petazzoni, “Device Tree for Dummies”, http://free-electrons.com

[7] https://crashcourse.ca/dokuwiki/doku.php?id=devm_ioremap_resource
[8] http://xillybus.com/tutorials/device-tree-zynq-3
[9] http://xillybus.com/tutorials/device-tree-zynq-4

доц. д-р инж. Любомир Богданов, 2024 г.
Важни команди:

```
sudo /home/lab1362-01/programs/gnu-mcu-eclipse/openocd/0.10.0-12-20190422-2015/
bin/openocd -f board/stm32f7discovery.cfg

telnet localhost 4444
reset init
flash probe 0
flash write_image erase u-boot-dtb.bin 0x08000000
flash write_image erase u-boot-dtb.bin 0x08008000
reset run

openocd -f board/stm32f7discovery.cfg -c"init" -c"reset halt" -c"flash probe 0" -c"flash
write_image erase u-boot-dtb.bin 0x8000000" -c"flash write_image erase u-boot-dtb.bin
0x8008000" -c"reset run" -c"exit"

setenv fdtcontroladdr c0ea0128
printenv
mmc dev 0
fatls mmc 0
fatload mmc 0 0xc0700000 /stm32f769-disco.dtb
fatload mmc 0 0xc0008000 /zImage
icache off
bootz 0x0c0008000 - 0xc0700000
```

или ако се използва файлова система ext4 на SD картата:

```
mmc dev 0
ext4ls mmc 0
ext4load mmc 0 0xc0700000 /stm32f769-disco.dtb
ext4load mmc 0 0xc0008000 /zImage
icache off
bootz 0x0c0008000 – 0xc0700000
```