Odsek za elektroniku

Projektovanje digitalnih sistema

Web pristup mernoj stanici

Mentor:
Strahinja Janković

Student:
Pavle Lakić *(2018/3304)*

Sadržaj

Uvod

Ovaj dokument ima za cilj da prikaže rešenje sistema za web pristup mernoj stanici korišćenjem Versatile Express platforme sa Cortex-A9 procesorom (emulirane preko QEMU).

U nastavku dokumenta će biti opisani Projektni zahtevi, Toolchain, Bootloader, Linux kernel, rootfs, drajver i aplikacija, elementi koji su neophodni za realizaciju projektnih zahteva.

U poslednjem odeljku će biti opisani koraci neophodni za demonstraciju funkcionalnosti sistema nakon izvršavanja skripte pds10-priprema.sh, kao što su primenjivanje generisanih patch-eva, i instalacija sqlite3 u okviru toolchaina.

1. Projektni zahtevi

Potrbeno je realizovati softversku podršku za pristup mernoj stanici preko mreže. Merna stanica je implementirana na razvojnom sistemu koji je baziran na modifikovanoj Versatile Express V2P CA9 ploči, i sadrži dva senzora, memorijski mapiran senzor za merenje napona i I2C senzor za merenje temperature. Da bi se omogućio pristup tim senzorima potrebno je prvo portovati U-boot i Linux operativni sistem na dostupni razvnojni sistem i kreirati odgovarajući minimalni root fajlsistem. Nakon toga je potrebno razviti odgovarajući drajver za memorijski mapiran senzor. Takođe je potrebno razviti korisničku aplikaciju koja će:

- komunicirati sa razvijenim drajverom memorijski mapiranog senzora
- komunicirati sa I2C senzorom putem i2c-dev iz korisničkog prostora
- izmerene podatke smeštati u bazu podataka

Dostupna je web aplikacija koju treba kopirati na zadatu lokaciju na root fajlsistemu. Web aplikacija prilikom pristupa web client-a čita podatke iz baze i prikazuje ih u vidu grafika. Svaki student će dobiti različit razvojni sistem, gde su razlike ograničene na:

- baznu adresu RAM memorije
- bazne adrese periferija: MMC, UART0, TIMER01, memorijski mapiranog senzora i I2C kontrolera
- pozicije bita u odgovarajućim registrima i prekidne linije na koju je povezan memorijski mapiran senzor
- adresu I2C senzora

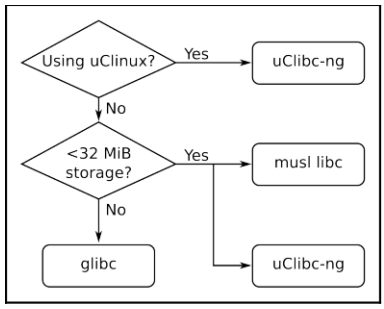
2. Toolchain

Toolchain je set alata koji služi da kompajlira izvorni kod sa hosta na target, tako da taj kod može da se pokrene na tagetu. Generalni pristup za rad sa toolchain-ovima je da ukoliko postoji već gotov, tako da zadovoljava potrebe platforme (arhitektura, floating point, C biblioteka). Takođe je vrlo bitno da se taj toolchain redovno odrzava i da je pouzdan. Ukoliko neki od ovih zahteva nije zadovoljen, potrebno je napraviti toolchain preko navođenih alata kao što su (najčešće Buildroot, Yocto, crosstool-ng) tako da se zadovolje potrebe platforme.

Dele se na dva tipa:

- Native - služi za kompajliranje koda na target koji je isti po arhitekturi kao host
- Cross - služi za kompajliranje koda na target koji se razlikuje po arhitekturi od hosta

Izbor C biblioteke se vezuje za količinu RAM memorije prisutne na razvojnoj platformi, pravilo koje se koristi je na slici 1.



Slika 1: Preporuka za izbor C biblioteke

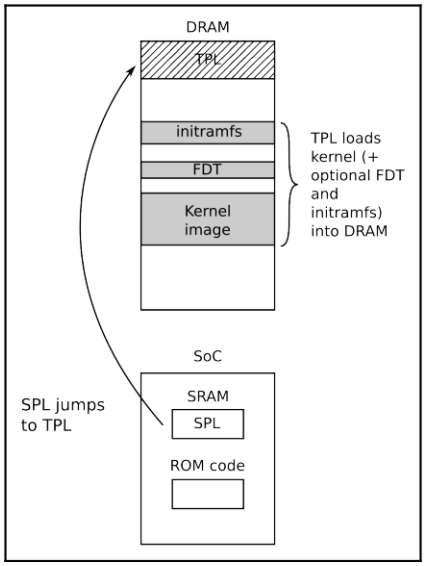
Takođe je bitno da kada se jednom napravi ili izabere toolchain da se u toku razvoja ne menja, jer može dovesti do jako suptilnih bug-ova.

Za našu razvojnu platformu ćemo koristiti gotov Linarov toolchain koje se može dohvatiti sa:
https://releases.linaro.org/components/toolchain/binaries/latest-7/arm-linux-gnueabihf/

3. Bootloader

Glavna uloga bootloadera je da inicijalizuje dovoljno hardvera koji je potreban da učita kernel operativnog sistema u DRAM, i na kraju da učita i pokrene sam kernel, koji dalje preuzima ulogu inicijalizovanja hardvera, kao i pokretanje softvera.

Bootloader je neophodan, jer nakon pokretanja sistema (power-on ili reset) jako mali deo hardvera je uopšte inicijalizovan, samo jedno jezgro CPU-a, kao i neka SRAM kojoj nije potreban kontroler. Zbog jako male količine SRAM na embedded uređajima, bootloader se sastoji iz najmanje dva, a vrlo često iz tri dela: FSBL (First Stage Bootloader), SPL (Second Program Loader) i trećeg u zavisnosti od prisutne količine SRAM, TPL (Tertiary Program Loader). FSBL se nalazi u ROM na razvojnom okruženju, odakle mora da se učita prvi deo koda (kod u ROM-u dostavlja proizvođač razvojnog okruženja u toku proizvodnje, i obično SPL) u SRAM, nakon čega se pokreće SPL, čija je uloga da aktivira DRAM kontroler, i učita TPL u DRAM. TPL inicijalizuje dovoljno hardvera da učita kernel u DRAM, pokrene ga, i izbriše ceo bootloader (FSBL, SPL, TPL) iz memorije jer dalju ulogu preuzima kernel i time bootloader više nije potreban. Kernel zna sa kojim hardverom radi preko Device Tree-ova koje TPL predaje kernelu nakon što ga učita i pokrene. Prikaz celog procesa je na slici 2.



Slika 2: Stanje memorije nakon završenog boot-ovanja

Postoje razni TPL bootloaderi, u projektu se koristi U-boot zbog podržane Versatile Express V2P CA9 ploče, čija će se konfiguracija koristiti za modifikovanu ploču na kojoj je treba realizovati sistem. Potrebno je promeniti određene fajlove kako ne bi narušili postojeće konfiguracije i funkcionalnosti ostalih ploča. Potrebno je prvo dodati u glavni Kconfig u okviru /arch/arm/ foldera postoji konfiguracija za Modifikovanu Versatile Express V2P CA9 ploču, kao na slici 3.

```
config TARGET_VEXPRESS_CA9X4
        bool "Support vexpress_ca9x4"
        select CPU_V7A
        select PL011_SERIAL

config TARGET_VEXPRESS_PDS14
        bool "Support vexpress_pds14"
        select CPU_V7A
        select PL011_SERIAL
```

Slika 3: Dodavanje nove ploče u konfiguraciju

Zatim je potrebno update-ovati Kconfig fajl u okviru board/armltd/vexpress kao na slici 4 .

```
if TARGET_VEXPRESS_CA9X4

config SYS_BOARD
        default "vexpress"

config SYS_VENDOR
        default "armltd"

config SYS_CONFIG_NAME
        default "vexpress_ca9x4"

endif

if TARGET_VEXPRESS_PDS14

config SYS_BOARD
        default "vexpress"

config SYS_VENDOR
        default "armltd"

config SYS_CONFIG_NAME
        default "vexpress_pds14"

endif
```

Slika 4: Dodavanje nove ploče u konfiguraciju

Zatim je bilo potrebno napraviti nove konfiguracije iz postojećih za Versatile Express V2P CA9 ploču, kako ne bi narušili već postojeće. U okviru projekta svaki student je dobio svoj patch fajl za QEMU, odnosno u njemu se mogu videti promene koje treba uneti u odgovarajuće fajlove kako bi bootloader mogao da funkcioniše na modifikovanoj ploči. Neke od informacija iz patch fajla se mogu videti na slici 5.

```
+#define PDS_VE_RAM          (0x80000000)
+#define PDS_VE_MMCI         (0x1001d000)
+#define PDS_VE_UART0        (0x1001e000)
+#define PDS_VE_TIMER01      (0x1000e000)
+#define PDS_VE_MM_SENSOR    (0x1001b000)
+#define PDS_VE_MM_IRQ       (26)
+#define PDS_VE_I2C          (0x1001c000)
+#define PDS_VE_I2C_ADDR     (0x5f)
```

Slika 5: Izmene u okviru pds14.patch fajla

Na osnovu tih izmena je potrebno kopirati i preimenovati postojeći vexpress_ca9x4_defconfig fajl u vexpress_pds14_defconfig i izvršiti izmene prikazane na slici 6.

```
CONFIG_ARM=y
CONFIG_TARGET_VEXPRESS_PDS14=y
CONFIG_SYS_TEXT_BASE=0x80800000
CONFIG_DISTRO_DEFAULTS=y
CONFIG_NR_DRAM_BANKS=2
CONFIG_BOOTCOMMAND="run distro_bootcmd; run bootflash"
# CONFIG_DISPLAY_CPUINFO is not set
# CONFIG_DISPLAY_BOARDINFO is not set
# CONFIG_CMD_CONSOLE is not set
# CONFIG_CMD_BOOTD is not set
# CONFIG_CMD_XIMG is not set
# CONFIG_CMD_EDITENV is not set
# CONFIG_CMD_LOADB is not set
# CONFIG_CMD_LOADS is not set
CONFIG_CMD_MMC=y
```

Slika 6: Izmene u okviru pds14.patch fajla

Kopirati, preimenovati vexpress_common.h u vexpress_pds14_common.h i izmeniti kao na slici 7 .

```
#ifndef __VEXPRESS_COMMON_PDS14_H
#define __VEXPRESS_COMMON_PDS14_H

/*
 * Definitions copied from linux kernel:
 * arch/arm/mach-vexpress/include/mach/motherboard.h
 */
#ifdef CONFIG_VEXPRESS_ORIGINAL_MEMORY_MAP
/* CS register bases for the original memory map. */
#define V2M_PA_CS0              0x40000000
#define V2M_PA_CS1              0x44000000
#define V2M_PA_CS2              0x48000000
#define V2M_PA_CS3              0x4c000000
#define V2M_PA_CS7              0x10000000
#define PDS_VE_MMCI             (0x0001d000)
#define PDS_VE_UART0            (0x0001e000)
#define PDS_VE_TIMER01          (0x0000e000)
#define V2M_PERIPH_OFFSET(x)    (x << 12)
#define V2M_SYSREGS             (V2M_PA_CS7 + V2M_PERIPH_OFFSET(0))
#define V2M_SYSCTL              (V2M_PA_CS7 + V2M_PERIPH_OFFSET(1))
#define V2M_SERIAL_BUS_PCI      (V2M_PA_CS7 + V2M_PERIPH_OFFSET(2))

#define V2M_BASE                0x80000000
#elif defined(CONFIG_VEXPRESS_EXTENDED_MEMORY_MAP)
/* CS register bases for the extended memory map. */
#define V2M_PA_CS0              0x08000000
#define V2M_PA_CS1              0x0c000000
#define V2M_PA_CS2              0x14000000
#define V2M_PA_CS3              0x18000000
#define V2M_PA_CS7              0x1c000000

#define V2M_PERIPH_OFFSET(x)    (x << 16)
#define V2M_SYSREGS             (V2M_PA_CS7 + V2M_PERIPH_OFFSET(1))
#define V2M_SYSCTL              (V2M_PA_CS7 + V2M_PERIPH_OFFSET(2))
#define V2M_SERIAL_BUS_PCI      (V2M_PA_CS7 + V2M_PERIPH_OFFSET(3))

#define V2M_BASE                0x80000000
#endif

/*
 * Physical addresses, offset from V2M_PA_CS0-3
 */
#define V2M_NOR0                (V2M_PA_CS0)
#define V2M_NOR1                (V2M_PA_CS1)
#define V2M_SRAM                (V2M_PA_CS2)
#define V2M_VIDEO_SRAM          (V2M_PA_CS3 + 0x00000000)
#define V2M_ISP1761             (V2M_PA_CS3 + 0x03000000)

/* Common peripherals relative to CS7. */
#define V2M_AACI                (V2M_PA_CS7 + V2M_PERIPH_OFFSET(4))
#define V2M_MMCI                (V2M_PA_CS7 + PDS_VE_MMCI)
#define V2M_KMI0                (V2M_PA_CS7 + V2M_PERIPH_OFFSET(6))
#define V2M_KMI1                (V2M_PA_CS7 + V2M_PERIPH_OFFSET(7))

#define V2M_UART0               (V2M_PA_CS7 + PDS_VE_UART0)
#define V2M_UART1               (V2M_PA_CS7 + V2M_PERIPH_OFFSET(10))
#define V2M_UART2               (V2M_PA_CS7 + V2M_PERIPH_OFFSET(11))
#define V2M_UART3               (V2M_PA_CS7 + V2M_PERIPH_OFFSET(12))

#define V2M_WDT                 (V2M_PA_CS7 + V2M_PERIPH_OFFSET(15))

#define V2M_TIMER01             (V2M_PA_CS7 + PDS_VE_TIMER01)
#define V2M_TIMER23             (V2M_PA_CS7 + V2M_PERIPH_OFFSET(18))
```

Slika 7: Izmene u okviru vexpress_pds14_common.h fajla

Kopirati, preimenovati vexpress.h fajl u vexpress_pds14.h i izmeniti kao na slici 8.

```
#ifndef __VEXPRESS_PDS14_H
#define __VEXPRESS_PDS14_H

#define CONFIG_VEXPRESS_ORIGINAL_MEMORY_MAP
#include "vexpress_common_pds14.h"

#endif /* VEXPRESS_PDS14_H */
```

Slika 8: Izmene u okviru vexpress_pds14_common.h fajla

Nakon izmena potrebno je opet make-ovati u-boot, nakon čega se dobija gotov bootloader, spreman za korišćenje.

4. Linux kernel

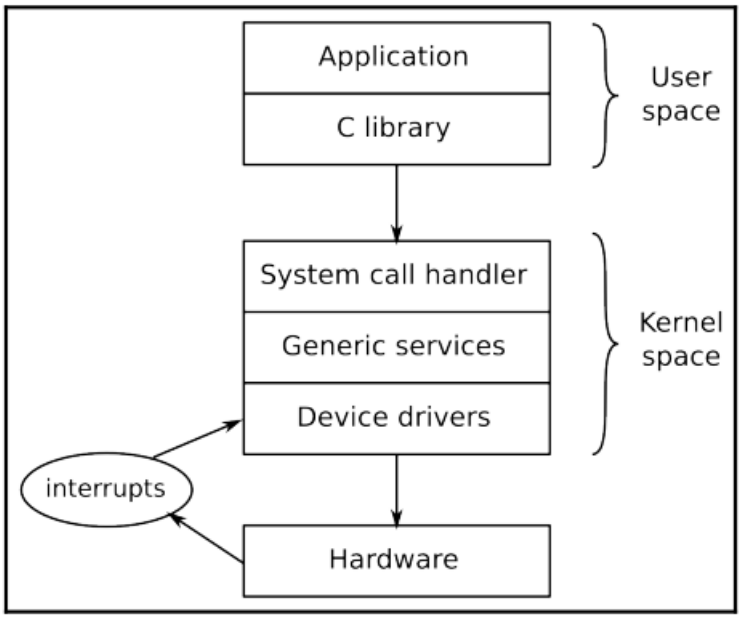
Kada hardver postane previše kompleksan, baremetal razvoj postaje skoro nemoguć i oduzeo bi previše vremena za sinhronizaciju, i optimizaciju. Takođe veliki broj primena bi zahtevao i dosta više softvera. Zato je danas u embedded svetu popularan Linux kernel, koji pruža te mogućnosti. Još neke od prednosti Linux kernela su sledeće:

- besplatan i open source
- mnoštvo podržanih arhitektura
- veliki broj ljudi radi na njegovom održavanju i unapređenju
- jako prilagodljiv

Kernel ima tri glavna zadatka:

- upravljanje resursima
- interakcija sa hardverom
- da pruži dobar API koji nudi dobru apstrakciju u user space programima

Na slici 9 je prikazan dijagram koji ilustruje princip rada kernela.



Slika 9: Princip rada Linux kernela

Linux kernel se deli na dva dela: user space i kernel space. Oba prostora imaju zasebne memorijske delove, odnosno ne postoji nikakvo preklapanje između njih. Korisnik preko C biblioteka generiše sistemske prekide, i na taj način programi iz user space-a komuniciraju sa kernelom. Postoje posebne biblioteke i za jedan i za drugi korisnički prostor. Sam kernel komunicira sa hardverom preko posebnog softvera koji se zove drajver. Osmišljeno je tako da ni na jedan način iz user space-a nije moguće poremetiti rad kernela, koji se uvek sa višim prioritetom izvršava u odnosu na programe iz user space-a.

Sledeći korak bi bio da se izabere verzija Linux kernela, i uglavnom se bira ona koja je stabilna. Postoji širok izbor verzija koji se može videti na:

https://www.kernel.org/

Za ove projektne zahteve je izabrana verzija linux-5.0.2. Nakon preuzimanja source koda u arch/arm/configs se mogu videti konfiguracije za razne ARM bazirane razvojne sisteme. Za Versatile Express V2P CA9, na osnovu koje se pravi razvnojni sistem za ove projektne zahteve se koristi multi_v7_defconfig. Potrebno je kopirati i izmeniti odgovarajući .dts i .dtsi fajl, kao i dodati mogućnost u Makefile-u da bi se kompajlirao odgovarajući .dtb fajl, koji je neophodan u procesu boot-ovanja kernela. Makefile je izmenjen na sledeći način:

```
dtb-$(CONFIG_ARCH_VEXPRESS) += \
        vexpress-v2p-ca5s.dtb \
        vexpress-v2p-ca9.dtb \
        vexpress-pds14.dtb \
        vexpress-v2p-ca15-tc1.dtb \
        vexpress-v2p-ca15_a7.dtb
```

Slika 10: Promene u Makefile-u u okviru /arch/arm/boot/dts

Nakon toga je potrebno kopirati sadžaj iz vexpress-v2p-ca9.dts u vexpress-pds14.dts, kao i iz vexpress-v2p-ca9.dtsi u vexpress-pds14.dtsi. Kernel na osnovu .dts i .dtsi fajlova (device tree source), kao što je već napomenuto u odeljku Bootloader, zna šta i gde se sve nalazi u hardveru. Kompajliranje kernela, kao rezultat daje executable binarni fajl .dtb koji je neophodan u procesu boot-ovanja. Treba uneti promene prikazane na slici 11, po specifikaciji projektnog zadatka u vexpress-pds14.dts fajl, a na slici 12 u vexpress-pds14.dts fajl.

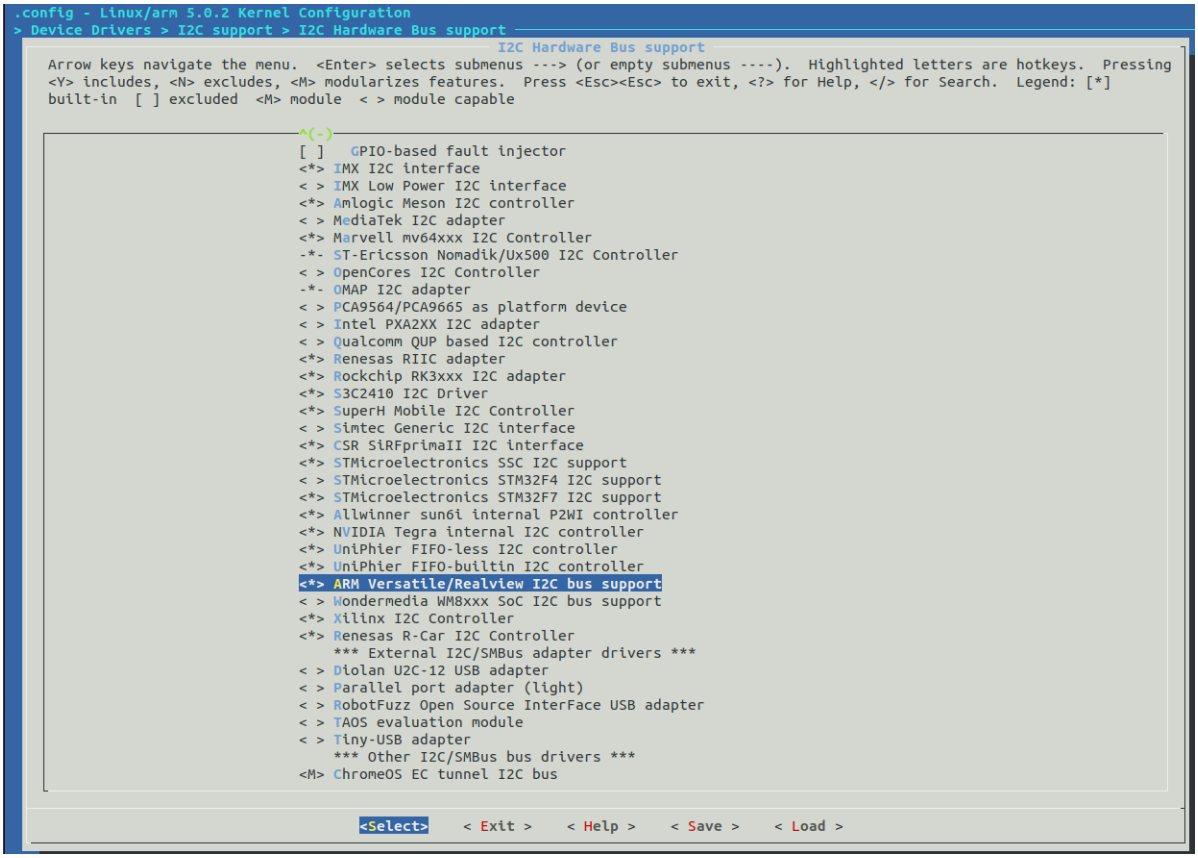
```
memory@80000000 {
        device_type = "memory";
        reg = <0x80000000 0x40000000>;
};
```

Slika 11: Promene u vexpress-pds14.dts fajlu

```
pds_mms_sensor@1b000 {
        compatible = "arm,versatile-mms";
        reg = <0x1b000 0x1000>;
        interrupts = <26>;
};

pds_i2c@1c000 {
        compatible = "arm,versatile-i2c";
        reg = <0x1c000 0x1000>;
        #address-cells = <1>;
        #size-cells = <0>;
```

Slika 12: Promene u vexpress-pds14.dts fajlu

Ove izmene su neophodne kako bi kernel prepoznao da postoji memorijski mapiran senzor za merenje napona, kao senzor za merenje temperature koji komunicira preko I2C protokola u razvojnom okruženju emuliranog preko QEMU dobijene preko pds14.patch fajla. Sada kada kernel zna gde se nalaze ti senzori, potrebno je napraviti za ARCH=arm i CROSS_COMPILE=arm-linux-gnueabihf- vexpress-pds14.dtb fajl za multi_v7_defconfig konfiguraciju, kao i zImage fajl koji je isto neophodan u procesu boot-ovanja. Takođe je potrebno dodati I2C kontrolar u image fajl kao na slici 13:



Slika 13: Uključenje podrške za I2C kontroler u image fajl

Nakon kroskompajliranja generisani su zImage i vexpress-pds14.dtb fajl, preko kojih je moguće boot-ovati Linux kernel.

5. Rootfs

Napravljeni kernel je beskorisan bez root fajlsistema, tako da je sledeći korak napraviti minimalan rootfs koji odgovara projektnim zahtevima. U Linux operativnom sistemu rootfs je zapravo organizovana hijerarhija podataka u direktorijume i fajlove. Da bi rootfs mogao da se koristi on mora da se mount-uje na određeni direktorijum, koji ne mora da se nalazi na samom razvojnom okruženju, već može da bude i na mreži. Organizacija direktorijuma koje se pridržava Linux može se pogledati na:

http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs

Kernelu se može proslediti rootfs načina, preko initramfs, particije SD kartice, particije FLASH memorije, ili preko mreže uz pomoć NFS (Network File System) protokola. Postoje više načina da se napravi rootfs. Može da se, ukoliko odgovara potrebama, da se nabavi već gotov, ali češći način za pravljenje minimalnih rootfs-a je uz pomoć alata koji na osnovu promenljivih konfiguracija prema potrebama korisnika generišu rootfs, odnosno prave simboličke veze između apleta koji se onda ponašaju kao zasebne aplikacije (cat, echo itd). Za potrebe projekta, osim osnovnih, potrebno je dodati kopirati iz sysroot-a toolchaina u rootfs staging direktorijum, a to su sqlite3 (kasnije o ovome), libdl, ibpthread kako bi drajver i aplikacija mogli da se izvršavaju u na razvojnoj ploči.

6. Drajver

Drajver je poseban softver čija je uloga da komunicira sa hardverom preko prekida. Kernel kao što je već napomenuto, komunicira sa hardverom preko drajvera. Drajveri se dele na tri tipa:

- Character
- Block
- Network

Block drajver radi sa uređajima koji imaju velike memorijske blokove (HDD, SSD itd). Network drajveri su isto što i block drajveri sa razlikom u tome što se ti podaci prenose preko mreže. Sve ostalo je character drajver, i ovaj tip drajvera je za ovaj projekat od najvećeg interesa. Potrebno je napraviti character platform device (to su uređaji vezani direktno na sistemsku magistralu) za memorijski mapiran senzor sa sysfs atributima (fajlovi pomoću kojih

se razmenjuju podaci između drajvera i user space-a). Primer pravljenja character uređaja je prikazan na slici 14 (gde su prikazani kreiranje klase uredjaja, dodeljivanje Major(automatski) i minor brojeva itd).

```
static int pds14_setup(struct device *parent)
{
        int ret;
        dev_t devt;
        struct pds14_mms *pds14dev;

        pds14dev = dev_get_drvdata(parent);

        ret = alloc_chrdev_region(&devt, 0, 1, DEVICE_FILE_NAME);
        if (ret < 0) {
                dev_err(parent, "failed to alloc chrdev region\n");
                goto fail_alloc_chrdev_region;
        }
        pds14dev->devt = devt;

        cdev_init(&pds14dev->cdev, &pds14dev_fops);
        ret = cdev_add(&pds14dev->cdev, devt, 1);
        if (ret < 0) {
                dev_err(parent, "failed to add cdev\n");
                goto fail_add_cdev;
        }

        pds14_class = class_create(THIS_MODULE, "pds14");
        if (!pds14_class) {
                ret = -EEXIST;
                dev_err(parent, "failed to create class\n");
                goto fail_create_class;
        }

        pds14dev->dev = device_create_with_groups(pds14_class, parent, devt, pds14dev, pds14_mms_groups, "%s%d", DEVICE_FILE_NAME,
MINOR(devt));
        if (IS_ERR(pds14dev->dev)) {
                pds14dev->dev = NULL;
                ret = -EINVAL;
                dev_err(parent, "failed to create device\n");
                goto fail_create_device;
        }

        return 0;
```

Slika 14: Primer kreiranja platform uređaja

Sysfs atributi se mogu definisati kao na slici 15

```
static struct attribute *pds14_mms_attrs[] = {
        &dev_attr_enable.attr,
        &dev_attr_int_en.attr,
        &dev_attr_data_raw.attr,
        &dev_attr_available_res.attr,
        &dev_attr_current_res.attr,
        &dev_attr_current_freq.attr,
        &dev_attr_available_freq.attr,
        &dev_attr_data_mV.attr,
        NULL,
};

ATTRIBUTE_GROUPS(pds14_mms);
```

Slika 15: Definisani atributi u sysfs

Dalje je prikazan jedan od atributa na slici 16.

```
static ssize_t data_raw_show(struct device *child, struct device_attribute *attr, char *buf)
{
        struct pds14_mms *pds14dev = dev_get_drvdata(child);

        u32 data = ioread32(pds14dev->base_addr + PDS14_DATA_OFFSET);
        data &= DATA_MASK;

        return sprintf(buf, "%d\n", data);
}
static DEVICE_ATTR_RO(data_raw);
```

Slika 16: Primer jednog od atributa sysfs-a

Što se tiče aplikacija, ona vidi atribute kao tekstualne fajlove, gde iz nekih može samo da čita, a u druge može i da čita i piše. Na ovaj način je realizovan platform drajver, sada još treba kernel da ga prepozna. U okviru projektnih zahteva je definisano da ovaj drajver mora da bude deo kernela, odnosno potrebno je izmeniti odgovarajuće Kconfig i Makefile-ove. Radi jednostavnosti drajver će se nalaziti u okviru /drivers/char, pa je i tu potrebno izmeniti Makefile i Kconfig. Izmene za Makefile su prikazane na slici 17 , a na slici 18 izmene za Kconfig.

```
obj-$(CONFIG_XILLYBUS)          += xillybus/
obj-$(CONFIG_POWERNV_OP_PANEL)  += powernv-op-panel.o
obj-$(CONFIG_ADI)               += adi.o
obj-$(CONFIG_PDS14_DRIVER)      += pds-mmsens.o
```

Slika 17: Izmena u /drivers/char/Makefile

```
source "drivers/tty/Kconfig"

config PDS14_DRIVER
        tristate "Driver made by Pavle Lakic PDS14"
        depends on ARCH_VEXPRESS
        default y
        help
          This is student made driver, prone to errors, when in doubt, say "N".
```

Slika 18: Izmene u /drivers/char/Kconfig

7. Aplikacija

Sa gotovim drajverom koji pruža mogućnost da mu se pristupi iz user space-a, potrebno je napraviti aplikaciju koja će mu dodati još funkcionalnosti. Aplikacija se sastoji iz glavnog thread-a koji komunicira sa memorijski mapiranim senzorom napona, i pomoćnog thread-a koji preko I2C komunicira sa senzorom. Takođe je bilo potrebno da aplikacija komunicira sa bazom podataka preko sqlite3 biblioteke.

Kako se pokazivač db može koristiti iz oba thread-a, potrebni je bilo zaštititi ga sa pthread_mutex-a, što je prikazano na slici 19 .

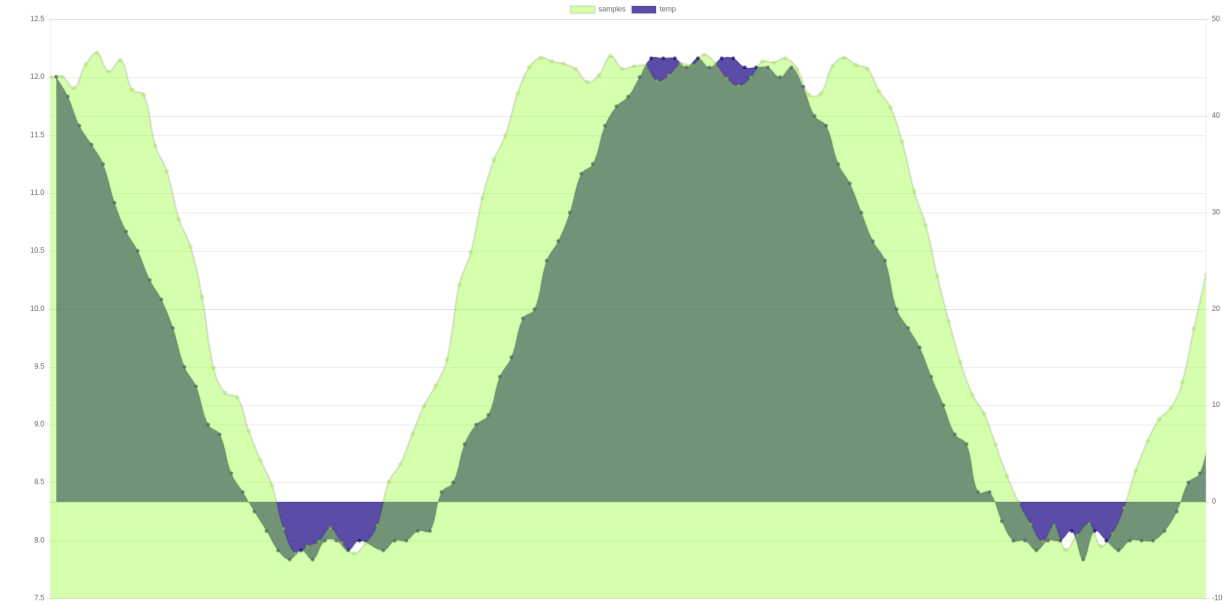
```
pthread_mutex_lock(&sql_mutex);
sprintf(query , "INSERT  INTO  Temperature(Time , Data) VALUES  (%u,%f);",(unsigned)time(NULL), temperature);

rc = sqlite3_exec(db, query , 0, 0, &err_msg);

pthread_mutex_unlock(&sql_mutex);
```

Slika 19: Primer korišćenja pthread_mutex-a za pristup db iz pomoćnog thread-a

Glavni thread pravi bazu podataka preko sqlite3 biblioteke, enable-uje memorijski mapiran drajver za napon, enable-uje mu interrupt i čita podatke u mV. Sve to radi preko već napravljenih sysfs atributa, odnosno radi sa time kao da upisuje odnosno čita iz fajlova. Takođe poliranjem proverava da li je došao nov podatak od hardvera, preko drajvera i update-uje bazu podataka sa vrednostima Samples (naponom) u voltima.

Pomoćni thread, odnosno za rad sa temperaturnim senzorom preko I2C kontrolera, ima za zadatak da otvori fajl iz /dev direktorijuma i da preko njega enable-uje I2C kontroler, preko kojeg stižu podaci o izmerenoj temperaturi. Potrebno je da pročita DATA_L i DATA_H iz registara kontrolera, i da update-uje bazu podataka. Tajmerom je postignuto da se to dešava na svake dve dve sekunde.

Celokupna funkcionalnost sistema, odnosno rad aplikacije impelentiranom na razvojnom okruženju emulirane preko QEMU je prikazana na slici 20 .



Slika 20: Demonstracija rada sistema

8. Primena patch-eva

$ git clone https://github.com/Pavle-Lakic/pds_projekat.git

$ wget --user user --password password http://tnt.etf.rs/ 13m041pds/pdf/projekti/1920/pds14.patch

$ bash pds10-priprema.sh

$ gedit ~/env.sh

Promeniti vexpress-v2p-ca9.dtb u vexpress-pds14.dtb
Dodati sledeće u env.sh

export SYSROOT=$(arm-linux-gnueabihf-gcc -print-sysroot)

$ source ~/env.sh

$ cp pds14.patch qemu-3.1.0/

$ cd qemu-3.1.0/ $ patch -p1 <pds14.patch

$ cd bin/arm

$ make

$ cd ../../../u-boot-2019.01

$ cp u-boot_pds14.patch $PWD

$ patch -p1 <u-boot_pds14.patch

$ make ARCH=arm vexpress_pds14_defconfig

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j8

$ cd ../linux-5.0.2/arch/arm/boot/dts

$ cp kernel_pds14.patch $PWD

$ patch <kernel_pds14.patch

$ cd ../../../../drivers/char

$ cp driver.patch $PWD

$ patch <driver.patch

$ cd ../../

$ make ARCH=arm multi_v7_defconfig

$ make ARCH=arm menuconfig

enableovati kontroler za I2C

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j8

$ cd ../

$ mkdir app

$ cd app

$ cp App.patch $PWD

$ patch <App.patch

$ cd ../

$ wget http://www.sqlite.org/2015/sqlite-autoconf-3081101.tar.gz

$ tar xf sqlite-autoconf-3081101.tar.gz

$ cd sqlite-autoconf-3081101/

$ CC=arm-linux-gnueabihf-gcc ./configure --host=arm-linux-gnueabihf -prefix=/usr

$ make

$ make DESTDIR=$(arm-linux-gnueabihf-gcc -print-sysroot) install

$ cp -a $SYSROOT/usr/bin/sqlite3 $STAGING/usr/bin

$ cp -a $SYSROOT/usr/lib/libsqlite3.la $STAGING/usr/lib

$ cp -a $SYSROOT/usr/lib/libsqlite3.so $STAGING/usr/lib

$ cp -a $SYSROOT/usr/lib/libsqlite3.so.0 $STAGING/usr/lib

$ cp -a $SYSROOT/usr/lib/libsqlite3.so.0.8.6 $STAGING/usr/lib

$ cp -a $SYSROOT/lib/libpthread.so.0 $STAGING/lib

$ cp -a $SYSROOT/lib/libpthread-2.28.so $STAGING/lib

$ cp -a $SYSROOT/lib/libdl.so.2 $STAGING/lib

$ cp -a $SYSROOT/lib/libdl-2.28.so $STAGING/lib

$ cd $STAGING

$ mkdir www

$ cd www/

$ cp -r /cgi-bin $PWD

$ cd ../../app

$ make

$ cd ../tools/

$ bash pds-create-sdcard.sh

$ bash pds-mount-sdcard.sh sd.img

Proveriti da li je u env.sh skripti $VE_DTB pokazuje na vexpress-pds14.dtb! Ako nije, opet source ~/env.sh

$ sudo cp $VE_DTB /mnt/boot/

$ sudo cp $ZIMAGE /mnt/boot/

$ sudo cp boot.scr /mnt/boot/

$ cd $STAGING

$ sudo cp -ar * /mnt/rootfs/

$ sync

Sada se cd gde opet u tools gde je izvršena pds10-priprema.sh

$ cp ../app/app /mnt/rootfs/home

$ bash pds-umount-sdcard.sh sd.img

$ bash pds-enable-qemu-network.sh

$ qemu-system-arm -M vexpress-pds14 -m 1G -kernel $UBOOT -nographic
-drive file=sd.img,format=raw,if=sd -net nic -net tap,ifname=tap0,script=no

Nakon startovanja kernela

root

cd /

mount -o remount,rw /

ifdown eth0

ifup eth0

httpd -p 80 -h /www/

cd /home/

./app

Nakon ovoga otici na google i ukucati

http://192.168.10.101/cgi-bin/samples

9. Zaključak

Na osnovi slike 20 se može zaključiti da je projektovan sistem prema traženim projektnim zahtevima. Takođe se na target sistemu u okviru /sys/class/pds14/pds14_mmsensor0/ mogu videti fajlovi u kojima je moguće modifikovati brzinu očitavanja, preciznost vrednosti napona i same podatke koje dostavlja drajver.

Literatura

[1] Materijali sa predavanja i vežbi
http://tnt.etf.rs/ 13m041pds/

[2] Jako dobra knjiga koju je preporučio Strahinja Janković
Mastering Embedded Linux Programming 2nd Edition, Chris Simmonds

[3] https://www.kernel.org/

[4] https://www.linaro.org/

[5] Razni tutoriali za drajvere
https://linuxhint.com/linux-device-driver-tutorial/
http://www.tutorialsdaddy.com/linux-device-drivers/writing-simple-character-device-driver/

[6] https://www.sqlite.org/docs.html

[7] https://www.busybox.net/