Уграђени Рачунарски Системи

Извјештај о урађеном пројектном задатку



Професор: проф. Др. Златко Бундало Студенти: Марко Кнежић 11150/16

Асистент: Миладин Сандић Павле Вигњевић 11132/15

# Увод – Задатак

Текст задатака:

Написати C програм који очитава податке са тастатуре те их приказује на 7SEG дисплејима доступним на DE1-SoC развојном окружењу. По притиску сваког од тастера, карактер се исписује на првом доступном 7SEG дисплеју. Уколико су сви доступни дисплеји попуњени, врши се шифтовање за 1 мјесто те се нови карактер уписује на доступни 7SEG дисплеј, и поступак се понавља за сваки нови унесени карактер. Одабир смијера шифтовања (десно/лијево) треба да буде омогућен преко једног од прекидача доступних на DE1-SoC развојном окружењу. На DE1-SoC плочи треба претходно бити подигнут Linux OS. У извјештају је потребно приложити поступак пројектовања, као и опис кориштених компонената из IP каталога.

# Реализација

## План реализације

За рјешење задатка потребно је оспособити DE1-SoC развојну плочу за рад(омогућити рад са OS-ом са SD картице, омогућити рад седмосегментног дисплеја, омогућити читање позија прекидача), направити Linux/GNU слику и уписати је на SD картицу, написати C код.

## Увод у процесорски систем Cyclone V

Cyclone V уређај је сyстем са једном матрицом на чипу (SoC – System on Chip) који се састоји од два различита дијела HPS (Hard Processor System) дијела и FPGA (Field Programmable Gate Array). HPS у основи саджи (MPU – Microprocessor Unit) микропроцесорску јединицу, контролере флеш меморије, подршку периферним уређајима, могућност уклањања грешака итд., док FPGA дио садржи FPGA fabric (унутрашња структура FPGA), управљачки блок , PCI Express контролере, чврсти меморијски контролер итд. HPS компонента хардвера треба дио софтвера који конфигурише различите могућности које пружа хардвер HPS-a. Ово значи да HPS компонента има мали отисак у FPGA унутрашњој структури, јер је његова једина сврха спојити софтверску и хардверску логику.

Како се Cyclone V SoC може користи у три различите конфигурације:

* FPGA
* HPS
* HPS & FPGA

за израду овог пројектног задатка је кориштена HPS & FPGA konfiguracija.

HPS-FPGA интерфејси пружају различите комуникацијске канале између HPS i FPGA fabric. HPS-FPGA интерфејси укључују:

* **FPGA‐to‐HPS bridge**
* **HPS‐to‐FPGA bridge**
* **Lightweight HPS‐to‐FPGA bridge**
* **FPGA manager interface**
* **Interrupts**
* **HPS debug interface**

Током израде пројектног задатка кориштен је **Lightweight HPS‐to‐FPGA bridge** интерфејс који чини магистрала са 32-битном фиксном ширином података и омогућава HPS-у да савлада трансакције „слејвова“ у FPGA fabric.

## Qsys систем

Циљ је да креирамо систем у којем HPS и FPGA могу да ураде неколико рачунања истовремено. Тачније желимо да имамо следеће могућности:

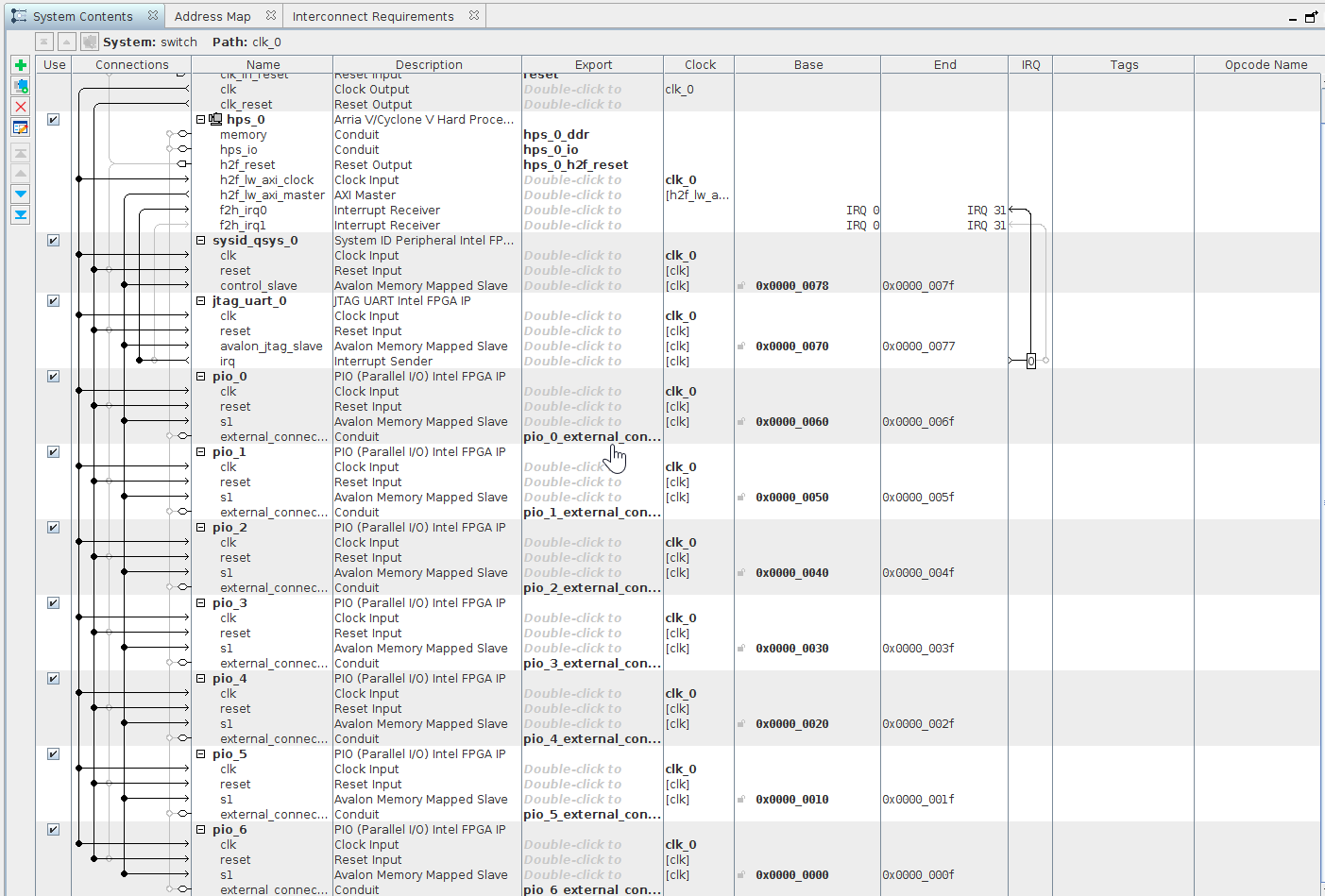
* HPS мора бити у могућности да користи један прекидач и шест 7-сегментних дисплеја повезаних на FPGA дио.
* HPS мора бити у могућности да користи ethernet порт на плочи
* HPS мора бити у могућности да користи micro SD card порт на плочи

Да би смо реализовали претходно наведене кораке користили смо софтвер **Quartus Prime Lite Edition 18.1**. Поред овог софтвера потребно је да имате сачуван и **ModelSim-Intel FPGA Edition као и Cyclone V device support (верзије 18.1) у истом директоријуму гдје се налази и** Quartus Prime Lite Edition 18.0. Додатно је потребно инсталирати и **SoC FPGA Embedded Development Suite (SoC EDS)** (верзија 18.1)(овај софтвер ће вам омогућити да имате приступ embedded\_command\_shell-у).

Све системске компоненте које су потребне HPS-у да би могао да покрене Линукс угрђени систем детаљно су описане у [11]

Поред горе наведених системских компоненти додатно је потребно конфигурисати шест PIO (Parallel I/O) IP компоненти дужине од по 7 бита за мапирање на шест 7-сегментних дисплеја и једну IP компоненту дужине 1 бит за мапирање прекидача који ће нам омогућити да бирамо да ли ће се жељени испис на дисплејима помјерати у лијеву или десну страну (у конкретном случају мапирано је 10 бита тако да се сваки од могућих 10 прекидача може искористи за жељену намјену).

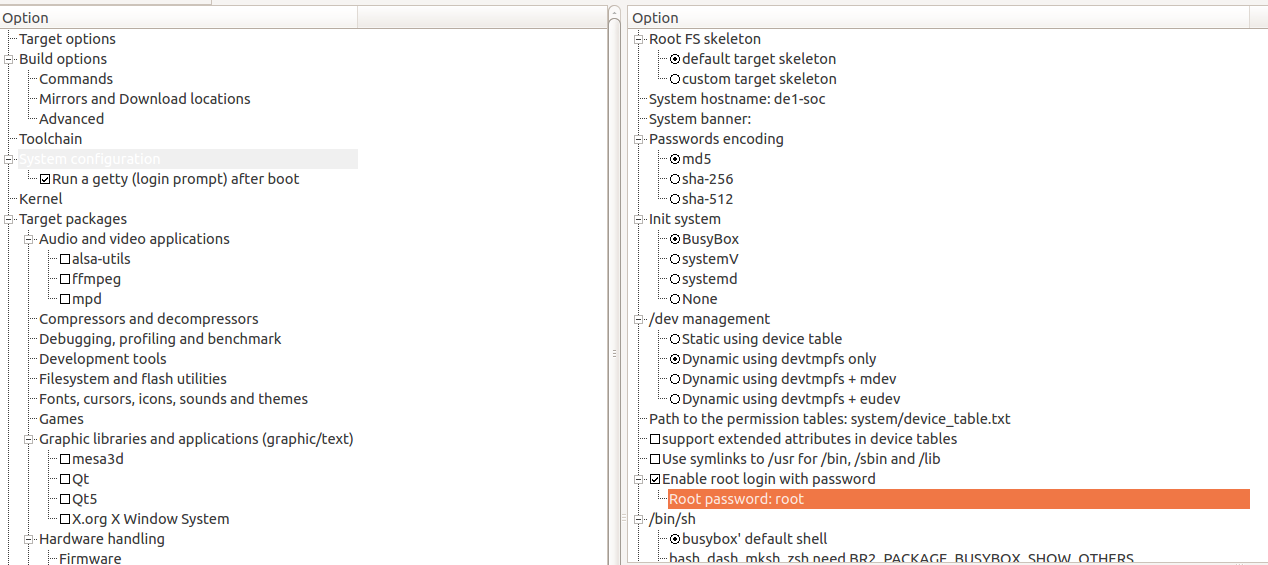
Сљедећа слика приказује Qsys систем (на кориштеној верзији Quartus-а назив је Platform Desgner) са компонентама које су кориштене.



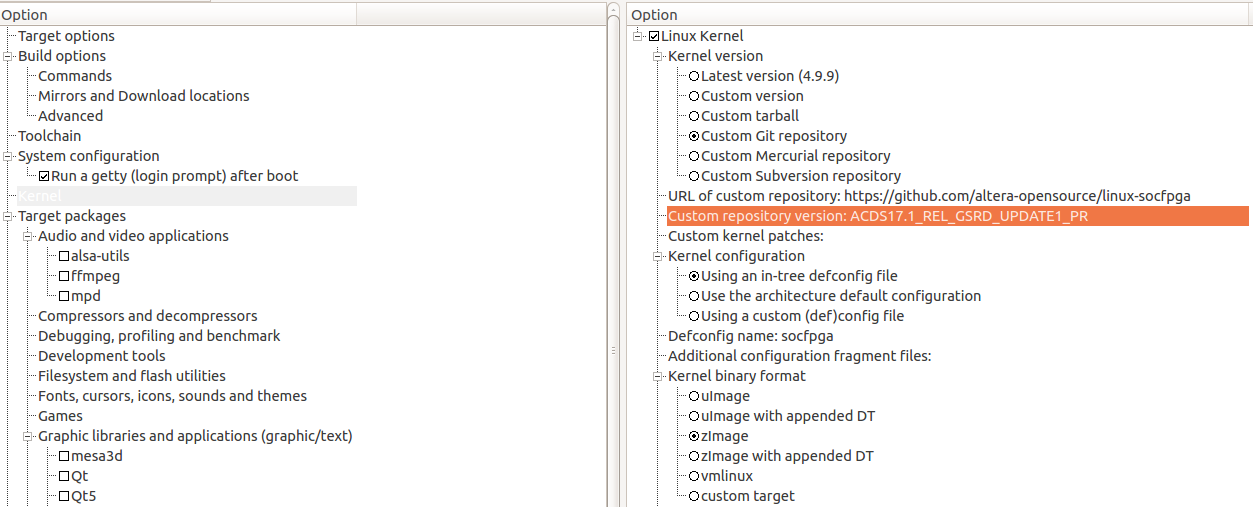
## Креирање Линукс слике

По успјешно завршеном компајлирању у Quartus софтверском пакету добијамо генерисан .sof фајл. За генерисање Линукс слике неопходан нам је .rbf фајл па је потребно да извршимо конверзију генерисаног .sof фајла у .rbf фајл. Послије ове конверзије потребно је да генеришемо Device Tree Blob (.dtb) и Device Tree Source (.dts) фајлове. За генерисање ових фајлова користи се sop2rts програм. Такође неопходно је да имате фајлове који садрже информације о плочи (system\_board\_info.xml и common\_board\_info.xml). Ове информације смо преузели са <https://github.com/mruizglz/UPM-ES-cyclonevbsp>. За креирање специфичних zImage, u-boot.img i rootfs.tar фајлова кориштен је захтијевани „buildroot-2017.02-rc1“ са release tag-a „ACDS17.1\_REL\_GSRD\_UPDATE1\_PR“. Само генерисање Линукс уграђеног система је детаљно описано у [11] па ћемо само навести специфичности на које треба обратити пажњу.

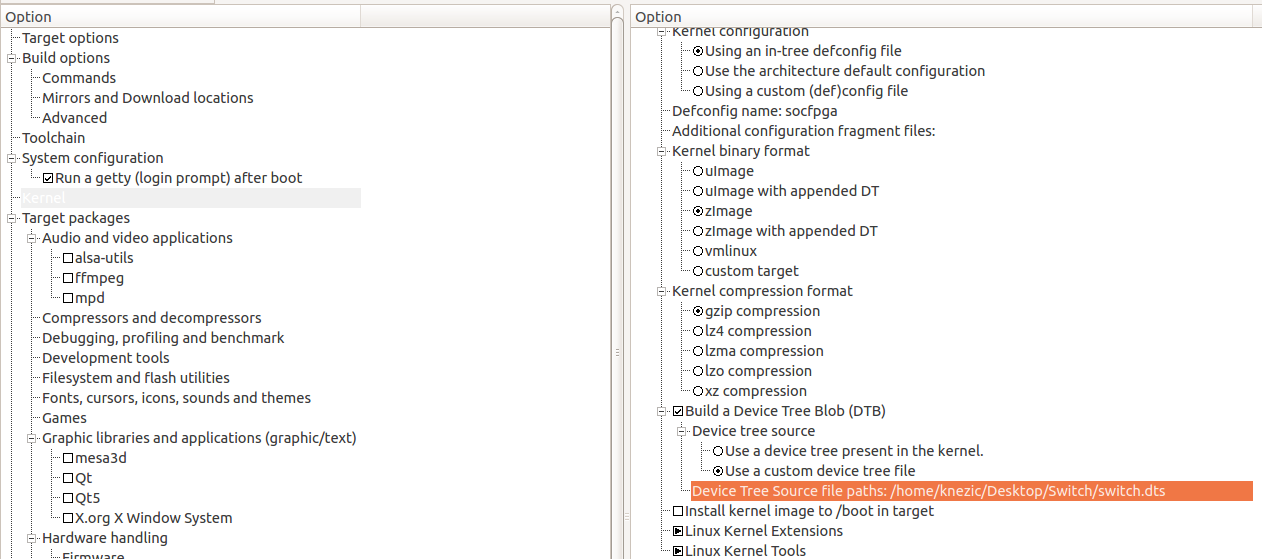
За једноставније конфигурисање ssh i наравно сигурност вашег система поставите жељену лозинку у пољу Root password као што је приказано на слици испод



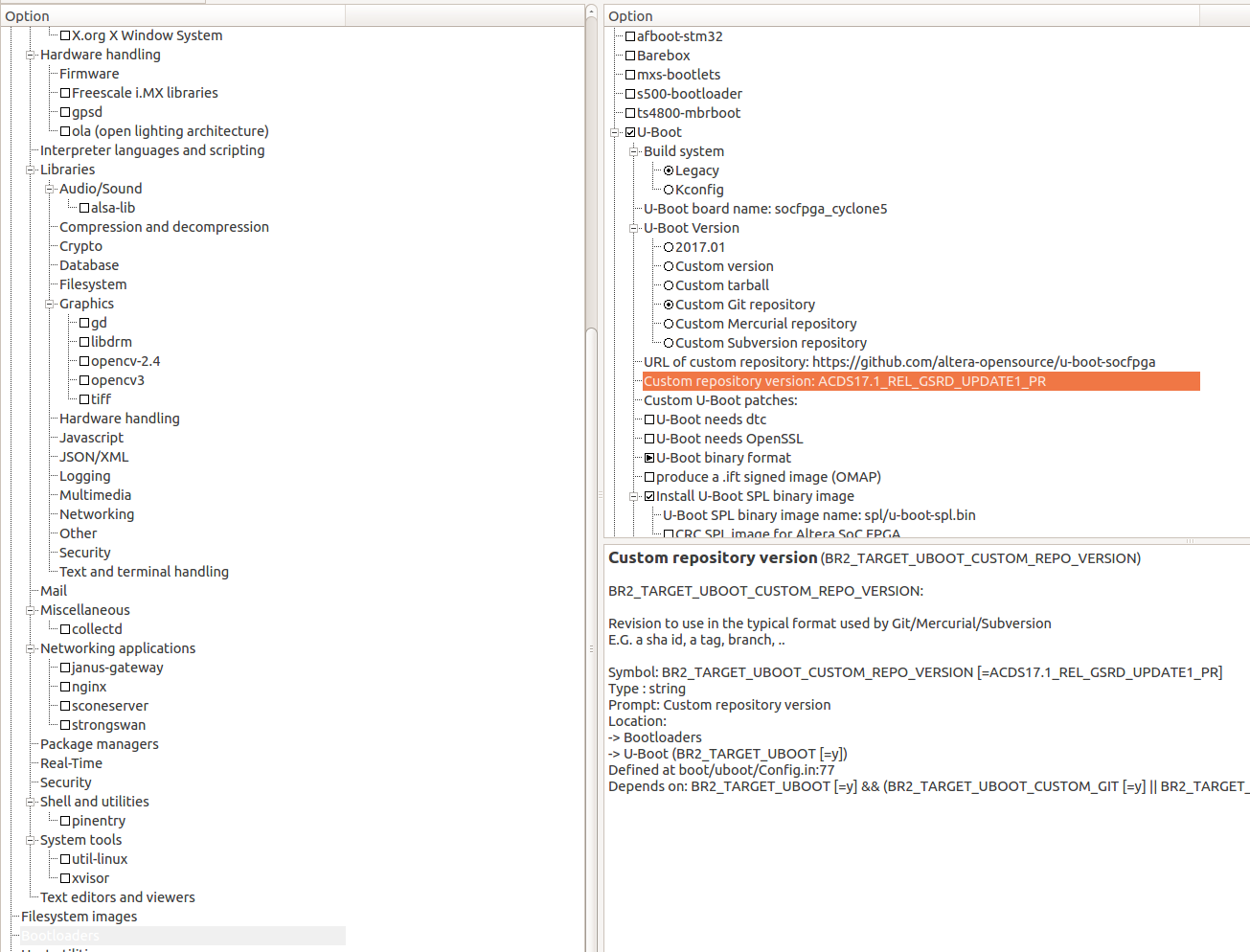
За одговарајући release tag у пољу Custom repository version поставите одговараћи назив tag-a као на слици испод.



Путању до одговарајућег .dts фајла поставите у пољу Device Tree Source Filenames као што је приказано на слици.



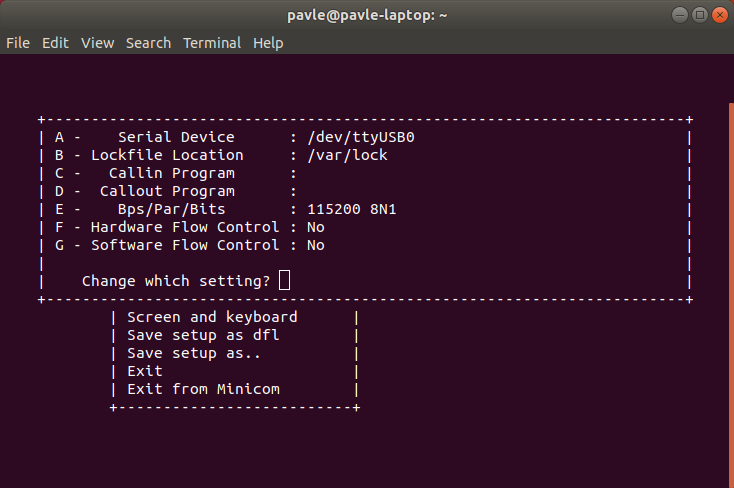
Што се тиче заглавља Bootloaders потребно је да у пољима URL of custom repository i Custom repository version поставите одговарајући линк до buildroot-a и одговарајући назив release tag-a као на слици.



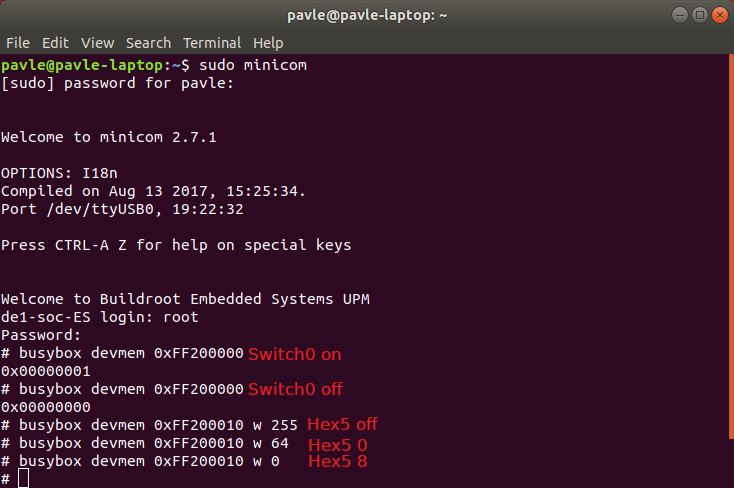
Након успјешно завршене конфигурације потребно је да је сачувате (File -> Save) па затим да у терминалу на путањи гдје вам се налази buildroot директоријум покренете команду make. Сљедећи корак је да у директоријуму buildroot пронађете директоријум output/images гдје вам се налазе неопходни фајлови за генерисање sdcard image-a. Генерисање sdcard image-a је детаљно објашњено у [11].

## Провјера рада плоче

По успјешном боот-овању плоче са Linux/GNU OS-ом вршимо тестирање рада саме плоче и раније дефинисаних компоненти. Развојну плочу смо повезали USB каблом са рачунаром који на себи има покренут Linux/GNU OS и инсталиран minicom комуникациони програм. Користимо **(sudo) minicom -s** за конфигурацију серијске везе.



Тестирамо прекидаче и седмосегментне дисплеје директним уписом вриједности у меморију. Користимо busybox програм са командом **busybox devmem mem\_add** за читање и **busybox devmem mem\_add w value** за упис.



Послије тестирања можемо подесити мрежни протокол **SSH**(кориштењем **ifconfig** команде и преправљањем **SSH** конфигурационог фајла **/etc/ssh/sshd\_config**) и користити њега за комуникацију.

## C код

Последњи кораци су писање **C** кода и компилација истог за дату **ARM-CyclonV** архитектуру(кориштењем **gcc** компајлера са укљученим **cross-compile** флега или кориштењем намјенског Linux/GNU OS-а који је направљен за развој директно на плочи).

Програм треба да омогући испис хексадецималне вриједности карактера унесених кроз стандарни улаз на седмосегментни дисплеј на два различита начина које одређује позиција прекидача sw0.

*/\* Seven segment displays and switch definitions \*/*

#define HEX54 hex5 \* 16 + hex4

#define HEX32 hex3 \* 16 + hex2

#define HEX10 hex1 \* 16 + hex0

int hex5 = 255;

int hex4 = 255;

int hex3 = 255;

int hex2 = 255;

int hex1 = 255;

int hex0 = 255;

int sw0;*//1 <-, 0 ->*

Промјењљиве су дефинисане тако да једна промјењљива одговара једном дисплеју. Промјењљиве чувају вриједности које ће бити уписане у меморију. Првобитне вриједности су 255 што је вриједност за гашење свих сегмената дисплеја.

int get\_display\_value(int value) {

switch (value) {

case 0:

return 0b1000000;*//64*

case 1:

return 0b1111001;*//121*

case 2:

return 0b0100100;*//36*

case 3:

return 0b0110000;*//48*

case 4:

return 0b0011001;*//25*

case 5:

return 0b0010010;*//18*

case 6:

return 0b0000010;*//2*

case 7:

return 0b1111000;*//120*

case 8:

return 0b0000000;*//0*

case 9:

return 0b0010000;*//16*

case 10: *// A*

return 0b0001000;*//8*

case 11: *// b*

return 0b0000011;*//3*

case 12: *//C*

return 0b0000110;*//6*

case 13: *//d*

return 0b0100001;*//33*

case 14: *//E*

return 0b0000110;*//6*

case 15: *//F*

return 0b0001110;*//14*

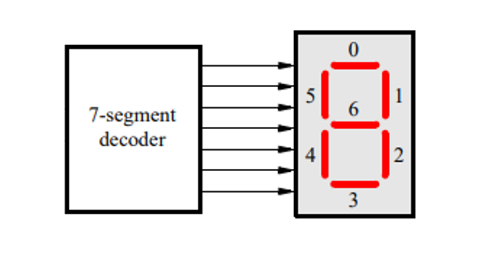
default: *//Display off*

return 0b1111111;*//255*

}

}

Функција за претварање хексадецималне вриједности карактера у ону која одговара седмосгментном дисплеју.



void input(char c) {

if (sw0) {

hex5 = hex3;

hex4 = hex2;

hex3 = hex1;

hex2 = hex0;

hex1 = c / 16;

hex0 = c % 16;

} else {

hex1 = hex3;

hex0 = hex2;

hex2 = hex4;

hex3 = hex5;

hex5 = c / 16;

hex4 = c % 16;

}

}

Функција за унос карактера и помјерање вриједности по већ унесених карактера лијево или десно у зависности од положаја прекидача. При уносу ASCII карактер се дијели на две промјењљиве(хексадецимална прва и друга цифра).

void print\_buffer() {

printf("========================================\n");

printf("CHA: [%c,%c,%c]\n", HEX54, HEX32, HEX10);

printf("HEX: [%0x,%0x,%0x]\n", HEX54, HEX32, HEX10);

printf("DEC: [%d,%d,%d]\n", HEX54, HEX32, HEX10);

printf("----------------------------------------\n");

}

Функција за испис карактера у ASCII, хексадецималном и декадном формату на стандарни излаз.

*// Turn off seven segment displays*

\*hex\_0\_addr = get\_display\_value(hex0);

\*hex\_1\_addr = get\_display\_value(hex1);

\*hex\_2\_addr = get\_display\_value(hex2);

\*hex\_3\_addr = get\_display\_value(hex3);

\*hex\_4\_addr = get\_display\_value(hex4);

\*hex\_5\_addr = get\_display\_value(hex5);

print\_buffer();

*// Main loop*

while (1) {

if (SW0[0] == 1) {

sw0 = 0;

} else {

sw0 = 1;

}

printf("Enter character to be entered on %s side:\n", sw0 ? "right" : "left");

scanf(" %c", &c);

fflush(stdin);

input(c);

print\_buffer();

\*hex\_0\_addr = get\_display\_value(hex0);

\*hex\_1\_addr = get\_display\_value(hex1);

\*hex\_2\_addr = get\_display\_value(hex2);

\*hex\_3\_addr = get\_display\_value(hex3);

\*hex\_4\_addr = get\_display\_value(hex4);

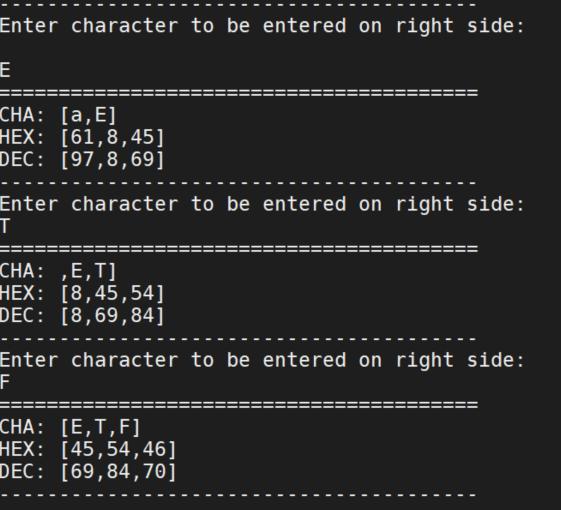
\*hex\_5\_addr = get\_display\_value(hex5);

}

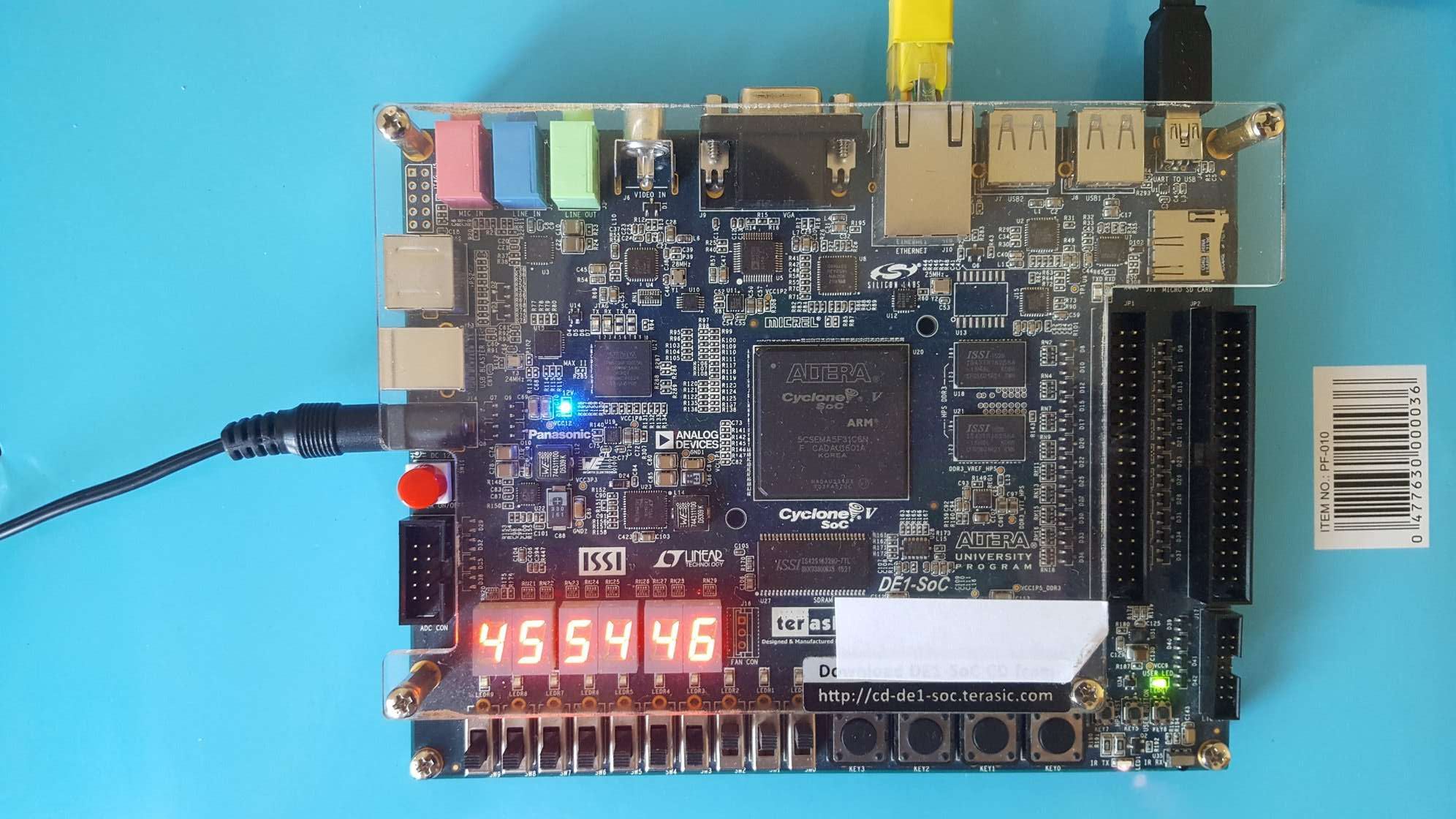
Главни функционални дио кода. Прво гасимо све седмосегментне дисплеје и исписујемо да је бафер који одговара симболима који треба да се прикажу на седмосегментним дисплејима празан. Провјеравамо позицију прекидача sw0 те исписујемо са које стране ће се вршити упис. Тражимо унос на стандардни улаз(прихватају се свe осим whitespace карактера).Врши се промјена стања бафера и он се приказује на стандардни улаз. Вриједности из бафера се конвертују у оне погодне за упис на седмосегментне дисплеје те се уносе у меморијске локације одговарајућих седмосгментних дисплеја.

# Исход

На сљедећим сликама имамо демонстративни приказ рада плоче са покренутим одговарујићим програмом који нам омогућава да унесене карактере са тастатуре прикажемо на 7-сегментном дисплеју у виду хексадецималних карактера. Изглед терминала са унесеним каракретима можете видјети на слици испод.



Приказ унесених карактера ‘E’,’T’,‘F’ на развојној плочи може се видјети на сљедећој слици.



# Закључак

Током израде сусрели смо се са проблемима некомпатабилности неколико верзија Quartus програма и Ubuntu дистрибуција. Qsys је био нарочито нестабилан са тенденцијом крешовања и замрзавања.

Задатак се може ријешити на више начина. Упис у меморију ради исписа на седмосегментне дисплеје се могао урадити другачије. Ради лакоће уписа дефинисали смо шест адреса(једну по дисплеју), али се могло радити и са једном, две или три адресе. Претварање из ASCII карактера у хексадецимални приказ се могао урадити и преко VHDL драјвера.

Једно од могућих унапређења је да се читање карактера за приказ не врши уносом на стандардни улаз, већ да се врши читање притиснутих карактера из **/dev/input/eventX** или/уз кориштење **xinput** команде. На овај начин би се омогућио приказ хексадецималне вриједности карактера на седмосгментни дисплеј одмах при његовом притискању, без чекања и без потребе за притискањем ентер тастера.

# Литература

1. DE1-Soc [de1-soc.terasic.com](file:///C:\Users\pajse\Downloads\de1-soc.terasic.com)
2. FPGA <https://en.wikipedia.org/wiki/Field-programmable_gate_array>
3. Cyclone V Hard Processor SystemTechnical Reference Manual <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_54001.pdf>
4. minicom manual <https://linux.die.net/man/1/minicom>
5. busybox manual <https://linux.die.net/man/1/busybox>
6. xinput maunal <https://linux.die.net/man/1/xinput>
7. dev/input <https://www.kernel.org/doc/Documentation/input/input.txt>
8. ASCII table <http://www.asciitable.com/>
9. Makefile <https://opensource.com/article/18/8/what-how-makefile>
10. Buildroot <https://buildroot.org/>
11. UsingQuartusandBuildrootforbuildingEmbeddedLinuxSystems(De1‐SOC)V1.9 <http://oa.upm.es/45352/1/DE1-SoC_Embedded_Linux_Systems_1_9.pdf>