Ugrađeni Računarski Sistemi

Izvještaj o urađenom projektnom zadatku



Profesor: prof. Dr. Zlatko Bundalo Studenti: Marko Knežić

Asistent: Miladin Sandić Pavle Vignjević

# Uvod – Zadatak

Tekst zadatka:

Napisati C program koji očitava podatke sa tastature te ih prikazuje na 7SEG displejima dostupnim na DE1-SoC razvojnom okruženju. Po pritisku svakog od tastera, karakter se ispisuje na prvom dostupnom 7SEG displeju. Ukoliko su svi dostupni displeji popunjeni, vrši se šiftovanje za 1 mjesto te se novi karakter upisuje na dostupni 7SEG displej, i postupak se ponavlja za svaki novi uneseni karakter. Odabir smijera šiftovanja (desno/lijevo) treba da bude omogućen preko jednog od prekidača dostupnih na DE1-SoC razvojnom okruženju. Na DE1-SoC ploči treba prethodno biti podignut Linux OS. U izvještaju je potrebno priložiti postupak projektovanja, kao i opis korištenih komponenata iz IP kataloga.

# Realizacija

## Plan realizacije

Za rješenje zadatka potrebno je osposobiti DE1-SOC razvojnu ploču za rad(omogućiti rad sa OS-om sa SD kartice, omogućiti rad sedmosegmentnog displeja, omogućiti čitanje pozija prekidača), napraviti Linux/GNU sliku i upisati je na SD karticu, napisati C kod.

## Uvod

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:

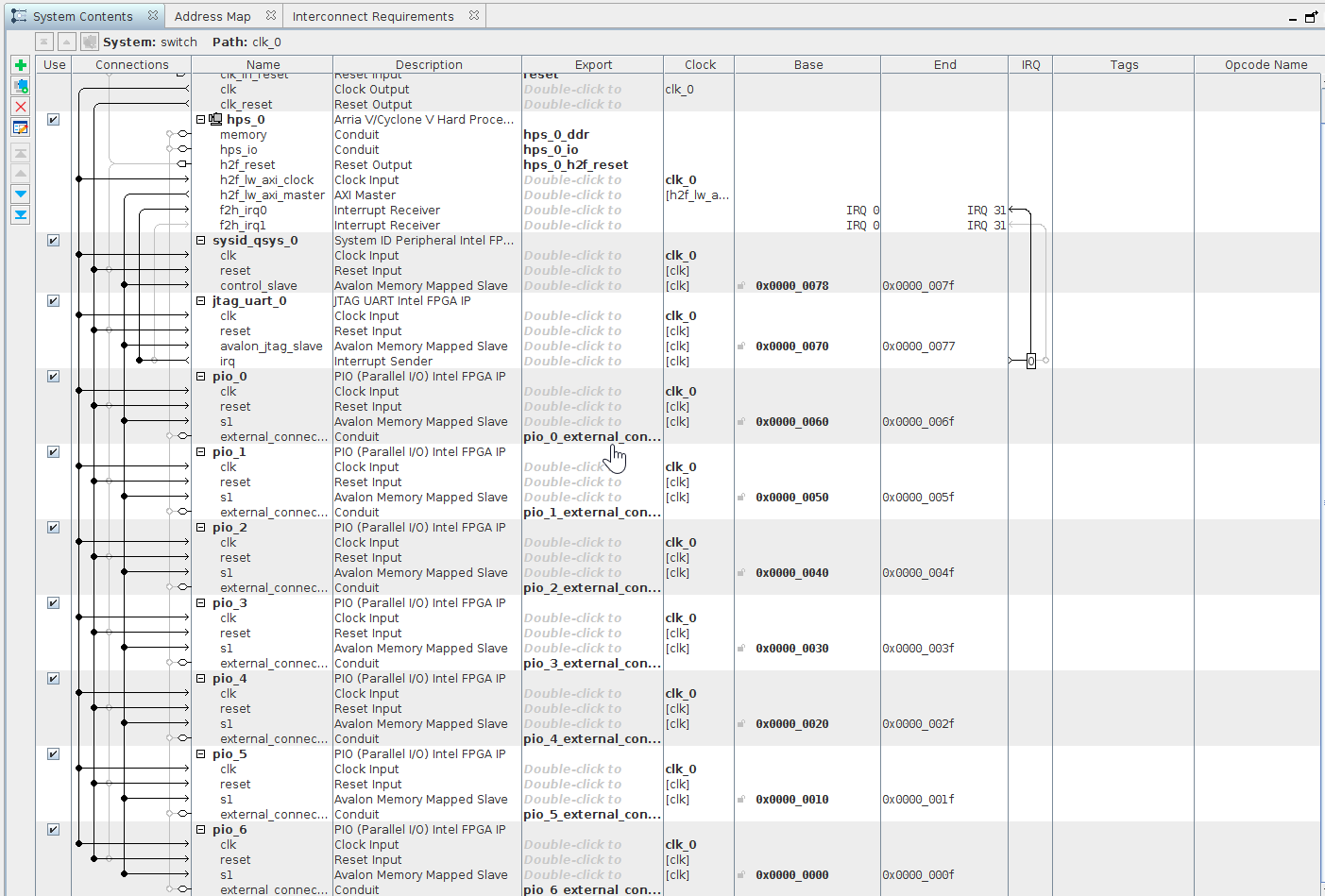
* HPS мора бити у могућности да користи један прекидач и шест 7-сегментних дисплеја повезаних на FPGA дио.
* HPS мора бити у могућности да користи ethernet порт на плочи
* HPS мора бити у могућности да користи micro SD card порт на плочи

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 [Embedded Systems Using Quartus and Buildroot for building Embedded Linux Systems (De1‐SOC)]

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).

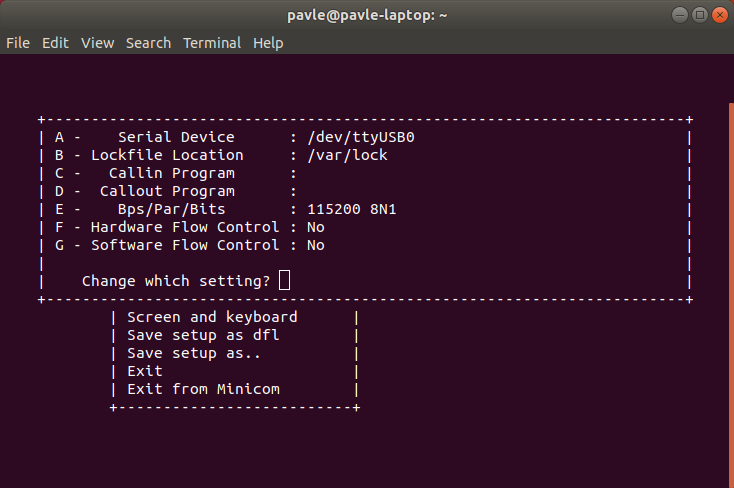
Сљедећа слика приказује Qsys систем (на кориштеној верзији Quartus-а назив је Platform Desgner) са компонентама које су кориштене.



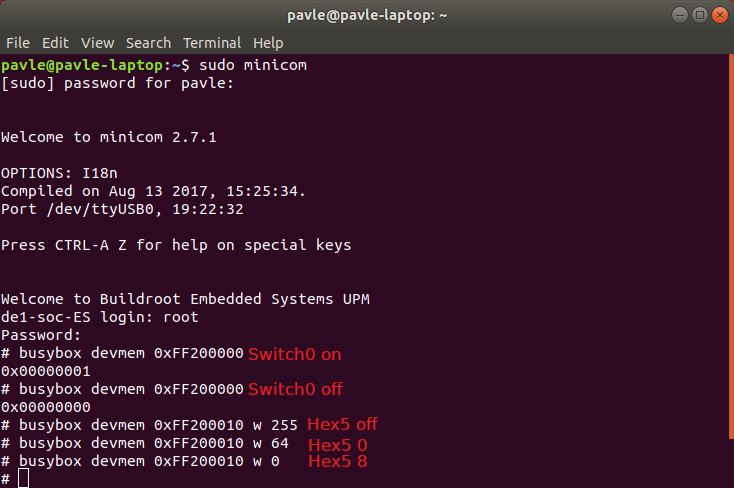
## Generisanje img

## Provjera rada ploče

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 kod

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 ->*

Дефиниција потребних промјењљивих за лак упис и испис 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*

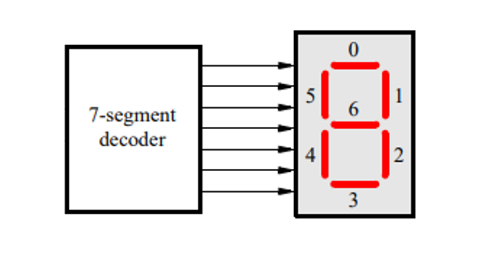
default: *//Display off*

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.

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.

# Ishod

# Zaključak

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 кориштењем **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.

# Literatura